The Application of Architectural and Design Patterns in Enterprise Systems

By

Seamus Kelly

Dissertation submitted to

Sligo Institute of Technology

in partial fulfilment of the requirements for the

M.Sc. in Computing

September 2005

Dr. Kate O'Dubhchair
Project Supervisor

Acknowledgements

I would like to thank my supervisor, Dr. Kate O'Dubhchair, for undertaking the supervision of this project. Having changed direction in her career, it required a return to her roots to take on this work and I very much appreciate her doing so. Her experience, advice, attention to detail and encouragement were invaluable throughout.

I would like to thank my wife, Carmel, for her support throughout the duration of the project, without which this work would not have been completed. Her support, as in life, has been unbending. The many household chores and gardening jobs, which have been long-fingered during this work, will now be completed.

I would like to thank Damien, Sinéad and Claire for everything they have taught me and for the pleasure and happiness they have brought to my life. They have grown into fine young citizens in spite of my worst attempts at parenting.

I would like to thank the many students and colleagues, teaching and non-teaching, that I have had the pleasure to work with over many years. Their enthusiasm has kept alive my interest in teaching. A special word of thanks is due to two hard working members of the technical staff in the Computing Department in DkiT, Brigid Conlon and Maura Ryan and also to Mary McKenna in the administrative section.

Abstract

This thesis investigates the use of various J2EE technologies and the application of best practice in the use of these technologies. It uses the knowledge gained in this investigation to develop a demonstration application designed for use in the teaching of these technologies. The demonstration application is designed to lead students from the development of standalone components to the development and integration of composite components in a complete working application. The learning experience is enhanced as students must identify design faults at various stages of development, and attempt solutions, before being introduced to design patterns that resolve the faults. In this way students get a real appreciation of the benefits of the patterns. Various exercises are identified throughout the thesis to re-inforce the learning.

Java 2 Enterprise Edition (J2EE) is a specification from Sun Microsystems for developing multi-tiered distributed applications. J2EE covers a wide range of technologies that are applied across various tiers in an n-tiered architecture. Learning to use these various technologies is not a trivial task and there is a strong demand for experienced J2EE developers.

"*J2EE requires significant knowledge and is not for the faint-hearted. Enterprise platforms are inherently complex, slowing down advanced Java developers and creating a barrier to entry for many mainstream developers. There is a critical shortage of advanced Java developers, and especially Java developers with expertise in Enterprise Java Beans (EJBs)*". [1] John Crupi, Distinguished Engineer, Chief Java Architect, Sun Microsystems, Frank Baerveldt, Director of Software Architecture, Compuware Corporation. "Implementing Sun Microsystems' Core J2EE Patterns", 2005

However, learning to use the various J2EE technologies is no guarantee that these technologies will be used to apply good design practice. By applying architectural and design patterns, developers learn from the experiences of experts.

"*Learning to design comes from experience and from sharing knowledge on best practices and bad practices*". [2] Deepak Alur, John Crupi and Dan Malks.
Core J2EE Patterns (Best Practices and Design Strategies) 2nd Edition (Prentice Hall, 2003).

Chapter 1: Introduction

This chapter gives background information on enterprise systems, design patterns and J2EE technologies and it outlines the aims of the thesis. It also indicates some topics not included in the dissertation.

1.1 Background

Before outlining the aims of the thesis, some background information on the technologies, and the terminology used to describe these technologies, is presented.

1.1.1 Enterprise Applications

Examples of enterprise systems include payroll, shipping tracking, insurance, accounting, foreign exchange trading, etc. Enterprise applications usually involve a large volume of persistent data that is accessed concurrently by a large number of users and usually require a large number of user interface screens. Not all enterprise applications are large, but they usually provide a lot of value to the enterprise. [4] Martin Fowler, *Patterns of Enterprise Application Architecture*, (Addison-Wesley, 2003).

1.1.2 Design Patterns

In the introduction to the classic text book, *Design Patterns: Elements of Reusable Object-Oriented Software*, by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides (also known as the Gang of Four, or GoF), the authors state:

"One thing expert designers know *not* to do is to solve every problem from first principles. Rather, they reuse solutions that have worked for them in the past. When they find a good solution, they use it again and again. Such experience is what makes them experts." [5]

The study of patterns is the study of these "good solutions" which were arrived at by experts. By studying the solutions of experts, developers can stand on the shoulders of these experts and do not have to re-invent the wheel. [3]

The GoF patterns are general purpose patterns that arise in software solutions in many different problem domains. Many of these patterns therefore arise in enterprise systems. In addition to the GoF patterns, there are many patterns that arise in the context of enterprise applications. This project therefore uses patterns from the GoF catalogue as well as patterns that are specific to enterprise applications.

1.1.3 Java 2 Enterprise Edition (J2EE)

The J2EE platform uses a multi-tiered distributed application model. Application logic is divided into components according to function, and the various application components that make up a J2EE application may be installed on different machines, depending on the tier to which the application component belongs [6]. Client-tier components run on the client machine. The client machine could be a PC running a web browser or a Java client, or it could be a mobile device. Web-tier components and business-tier components run on the J2EE server and the DBMS runs on the database server. In fact the web-tier and business-tier components may be spread across several machines. In a development environment all three tiers may be installed on a single machine. The development environment for this project implements the various tiers on a single machine. The following diagram illustrates a typical J2EE architecture.

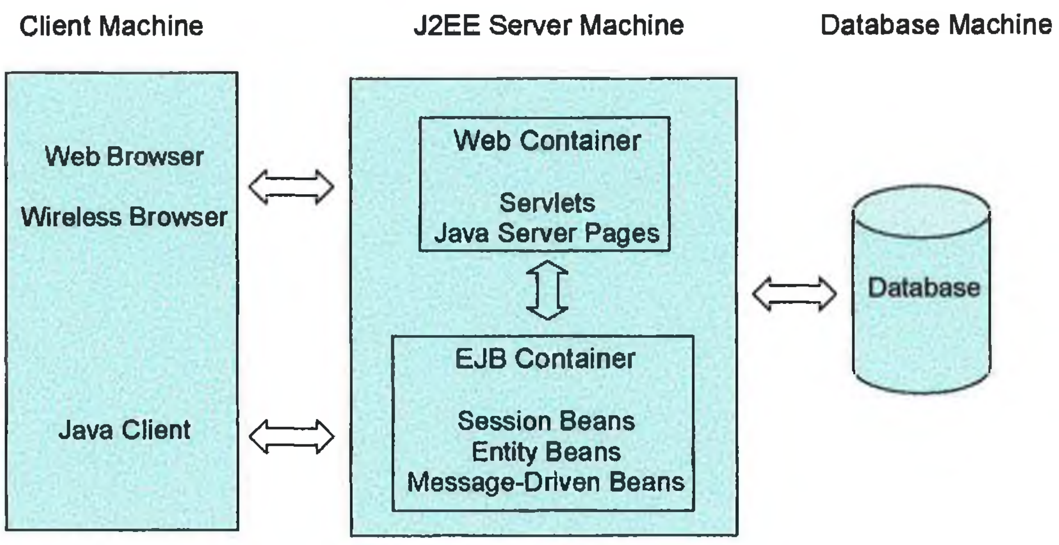


Figure 1 : J2EE Components

1.1.4 J2EE Technologies

J2EE application servers must provide containers that allow developers to build and deploy web applications and Enterprise Java Beans. A web container is responsible for handling the life-cycle of web components (Servlets and Java Server Pages). An Enterprise Java Bean (EJB) container manages the life-cycle of EJBs and is responsible for creating, pooling and destroying beans in order to best manage resources. The EJB container may also handle object persistence, security management and transaction management. J2EE servers are also responsible for providing the various other technologies that make up the

J2EE specification. This project uses the Oracle Application Server Container for J2EE (OC4J 10g, v10.1.2). This container supports J2EE 1.3 technologies and some of these technologies are listed below.

1.1.4.1 Servlet API v2.3

Servlets are platform-independent, server-side Java components used to extend the capabilities of a web server. A Servlet class extends the HttpServlet class and may be used to parse incoming requests, execute business logic, update databases and render a response in a web browser. However, a Servlet is normally used in a J2EE application to act as a Front Controller [2] in a Model-View-Controller (MVC) architecture [7] to process all requests from a client and to control the flow of the application. The Servlet API also provides the facility to define filters and listeners. Filters may be used to intercept a request in order to perform some pre-processing before accepting or rejecting the request, or to intercept a response to perform some post-processing before returning the response to the client. This facility is used to implement the Intercepting Filter pattern. [2] Listeners may be used to monitor and to react to events in a Servlet's life-cycle, e.g. Servlet creation and destruction, session creation and destruction, etc.

1.1.4.2 Java Server Pages (JSP) v1.3

JSPs are a server-side technology based on Java Servlets. A JSP is actually compiled into a Servlet before deployment to a web container. JSPs use a combination of HyperText Markup Language (HTML), Wireless Markup Language (WML), eXtensible Markup Language (XML), Java scriptlets and custom tags to render a response in a browser. JSPs are often used in conjunction with JavaBean components, i.e. Java classes that conform to the JavaBeans specification [8] (not to be confused with Enterprise Java Beans). Like a Servlet, a JSP may be used to parse incoming requests, execute business logic, update databases and render a response in a web browser. However, in a J2EE application JSPs are normally used to interact with the user by accepting input and by displaying information. JSPs usually form the *view* in a Model-View-Controller (MVC) architecture. [7]

1.1.4.3 Java Database Connectivity API (JDBC) v2.0

JDBC allows Java programmers to connect to a database and to query and update the database using the Structured Query Language (SQL). The J2EE server must provide an implementation of the DataSource interface to allow developers to connect to the database. J2EE servers also provide an implementation of the ConnectionPoolDataSource interface to provide connection pooling. JDBC is a platform and vendor independent API. JDBC may be used to implement the Data Access Object pattern [2] to encapsulate the SQL implementation details and to provide a uniform interface to allow access to the database. The Java Naming and Directory Interface API is used to locate the various data sources.

1.1.4.4 Java Message Service (JMS) v1.0.2b

The JMS API allows J2EE components to communicate asynchronously using either queues (also known as point-to-point messaging) where messages are sent to a single consumer, or topics (also known as publish/subscribe messaging) where messages are broadcast to all registered listeners.

1.1.4.5 Enterprise Java Beans (EJBs) v2.0

EJBs are a major topic with a steep learning curve. As EJBs are the most difficult J2EE technology to master, the emphasis in this project is on this technology. A brief introduction is given here, as EJBs are covered in detail in Chapter 4. There are three types of EJB: session, entity and message-driven beans.

(i) Session beans are responsible for managing processes. The most common type of session bean is a stateless session bean, i.e. a bean that does not maintain any client state. Stateless session beans are used extensively to implement the Session Façade pattern [2]. Stateful session beans, on the other hand, do maintain client state and this state is maintained across several client requests.

(ii) Entity beans are used for object persistence. This persistence may be managed by the container. This type of entity bean is called a Container Managed Persistent (CMP) bean and relieves the developer from the overhead of reading, writing and updating the database. In addition, the container is responsible for managing transactions and security. Bean Managed Persistent (BMP) beans require the developer to write the code to read, write and update the database, possibly using JDBC. The Data Access Object (DAO) [2] pattern may be used in conjunction with BMP beans.

(iii) Message-Driven Beans are stateless, server-side components for processing asynchronous JMS messages. A MDB can consume and process hundreds of messages concurrently as numerous instances of the MDB can execute concurrently. The MDB may consume messages from queues or topics, and based on the message received, may access other EJBs, e.g. an MDB may use an entity bean to update a database. MDBs may be used to implement the Service Activator pattern [2].

In order to implement an EJB, it is necessary to create four components. These components are

(i) a component interface, i.e. a Java interface that acts as the client's view of the bean.

(ii) a home interface, i.e. a Java interface that provides the client with methods to create, locate and remove beans, i.e. lifecycle methods of the bean.

(iii) a bean class, i.e. a Java class that implements the business methods required by the client.

(iv) a deployment descriptor file, i.e. an XML file that describes the various components of the EJB. A single deployment descriptor can hold the entries for several beans.

In addition to these components, an entity bean may also require a Primary Key class. A MDB does not have component interfaces.

The component interface represents the client's view of the bean. The client may be a remote client, i.e. residing on a different machine, meaning that the bean must expose a remote interface, or the client may be a local client, i.e. the client is co-located with the bean in the same container, meaning that the bean must expose a local interface. In fact a bean may expose both a remote interface and a local interface, i.e. the bean may have remote and local clients.

1.1.4.6 Java Naming and Directory Interface (JNDI) v1.2.1

A J2EE application uses many resources, e.g. data sources to access a database; home interfaces for the creation and location of EJBs; queues and topics for messaging services, etc. A J2EE server provides a service to store and locate these resources in a directory structure. The JNDI API provides a uniform way to locate and access these resources across a network. As it can be an expensive overhead constantly looking up the same resources, the Service Locator pattern [2] may be used to provide an efficient lookup mechanism by caching resources that have previously been retrieved. In this way the resources may be retrieved from the cache, thus avoiding the lookup overhead.

1.1.4.7 Java Connector Architecture (JCA) v1.0

Although most enterprise systems use relational database technology for data persistence, many J2EE applications are required to access legacy systems and other external systems. JCA provides connectivity with other types of external system.

1.1.4.8 Java Transaction API (JTA) v1.0.1b

JTA provides the J2EE developer with the facility to manage database transactions programmatically. In addition to using JTA to handle programmatic transaction management, J2EE provides for declarative transaction management in the deployment descriptor file, i.e. an XML file used to describe various resources used in a J2EE application.

1.2 Aims

The M.Sc. in Computing, which forms part of the Higher Education Staff Development Network programme in the Institutes of Technology, is designed for teaching and support staff in the areas of computing and technology. This programme is designed to facilitate staff to pursue further study at postgraduate level to advance their knowledge and skills in new and developing subject areas to enable them to become more effective in their teaching, research and support activities. [55]

As a lecturer in computing in Dundalk Institute of Technology, the author is charged with delivering courses on Enterprise Systems Development and Patterns for Enterprise Systems to fourth year students in the B.Sc. (Hons.) in Internet Technologies (Level 8) starting in September 2005.

With advances in technology the software systems being developed today are becoming more and more sophisticated requiring a deep knowledge and understanding of the underlying technologies. As the use of J2EE is clearly a demanding topic [1], the teaching of this technology presents many challenges.

There are a number of approaches to teaching design patterns, whether these are general purpose patterns or J2EE patterns. One approach is to examine each pattern in a catalogue in isolation, explain its use and provide a programming implementation of the pattern. Another approach is to implement a solution to a particular problem using a large variety of patterns. According to Eric Gamma [3], there is a danger in each of these approaches. He suggests that one should not simply enumerate each pattern in a catalogue and he also warns against trying to apply as many patterns as possible in an application. He recommends creating a real world application and, by understanding the solution and the short-comings of the solution, he recommends refactoring the solution by applying appropriate patterns.

The aims of this project are therefore:

(i) To gain a good working knowledge of the J2EE platform in general and EJB technology in particular.

(ii) To gain a good understanding of best practice in applying J2EE technology for the development of enterprise systems through the use of patterns.

(iii) To ease the learning curve of students studying these technologies by using appropriate tools to ease the development and understanding of the technologies.

(iv) To share the knowledge and experience gained during the research for this thesis with students in order to enhance their knowledge and learning experience.

In order to achieve this, a demonstration application for use in the teaching of J2EE and J2EE patterns is developed. As components for the application are developed, students are encouraged to identify problems with the design and to propose revised solutions before appropriate patterns are introduced to refactor the solution. The application is

(a) large enough to require a range of J2EE technologies and patterns for its solution.

(b) small enough to allow students to fully understand the problem and the solution.

(c) sufficiently interesting to engage the students.

(d) developed using an architecture that allows the seamless addition of wireless clients thereby increasing the students' appreciation of the benefits of the architecture used.

1.3 What this project is not

This project is not about the merits or otherwise of the J2EE platform nor is it about a comparison of J2EE and .NET technologies. J2EE is used as the platform for this project as the author is currently engaged in teaching various Java technologies and J2EE is a natural progression in this area. Is the J2EE platform suitable for developing enterprise systems? Suffice to say that eBay, the online auction company, chose the J2EE platform for version 3.0 of its software. A case study on the implementation of this software is available at [9]. As eBay can handle up to one billion hits per day [10], it can be assumed that a good knowledge of J2EE technologies and knowledge of best practice in the use of these technologies would be a useful skill for any student of computing.

This project is not about a comparison of the various J2EE servers available nor is it about a comparison of the various Integrated Development Environments (IDEs) available for developing J2EE applications. Clearly certain decisions have been made regarding the tools used, but these decisions have been influenced by various factors, e.g. the development and production environment available in Dundalk Institute of Technology, the cost of the tools used (all tools used were available without any outlay on cost), the availability of these tools for use by students, etc.

There are various Object-Relational tools that may be used in conjunction with J2EE, e.g. Java Data Objects (JDOs) [11], Hibernate [12], Toplink [13] and POJOs (Plain Old Java Objects) [14], Duncan Mills, Oracle Corporation, "The Rise of the POJO", July 2005. These technologies may be used instead of entity beans for database operations. It is not the intention of this project to compare any of these technologies, nor to recommend any technology in preference to another. As this project is about J2EE technologies, entity beans are used for object persistence.

This project is not about how students coped with learning the technologies, as the author does not yet have experience of teaching J2EE technologies (apart from simple web applications using Servlets and JSPs). During delivery of the above mentioned courses, feedback from students on the mode of delivery and the effectiveness of using the application will be obtained. Therefore, a follow on from this project would be a study into the success of using the application as a learning tool.

1.4 Development and Production Environment

There are many different J2EE servers available, e.g. JBoss[15], WebLogic[16], OC4J [17], etc. There are also many different integrated development environment (IDEs) available for developing and deploying J2EE applications, e.g. JBuilder[18], NetBeans [19], JDeveloper [20]. The tools used for this project were influenced by a number of factors.

(i) A desire to get a very good understanding of J2EE technologies.
(ii) A desire to get a very good understanding of J2EE patterns.
(iii) The need to use tools that would run on a fairly basic PC.
(iv) The need to use tools that would be freely available to students.
(v) The requirement that the completed application would be deployed to DkIT's Oracle Application Server accessing an Oracle database.
(vi) A desire to use development tools that would make application development easier once the earlier requirements were satisfied.

In order to get a good understanding of the various J2EE technologies, it is planned to initially develop the J2EE components for the demonstration application using only a simple text editor to write all components and all deployment descriptors. It is planned to use Oracle's standalone OC4J 10G (v9.0.4) J2EE server, accessing a database developed with MySQL v4.1.7 [21]. These tools run on a fairly basic PC and are freely available for download.

Having used the basic tools to learn the various J2EE technologies, Oracle's JDeveloper IDE will be used to develop the project's demonstration application and to explore the use of various J2EE patterns with this tool. The database used will be Oracle's Database 10g. These tools are also freely available but require a PC with a fairly high specification (Pentium III, 512 MB RAM, 1.5GB hard drive). For the wireless client, Oracle's JDeveloper Wireless Extension (JWE) will be used together with OpenWave SDK v6.2.2. [57]

In addition, it is intended to obtain a 30-day trial version of Compuware's OptimalJ [22] to examine the various patterns generated by this tool when used to automatically generate a simple data maintenance J2EE application from a domain class diagram.

It is intended that the final application will access DkIT's Oracle database and will be deployed on DkIT's Oracle Application Server.

1.5 Summary

This thesis is inspired by a desire to provide students with the tools necessary to develop J2EE applications by sharing the knowledge and experience of experts in a manner that will enhance the learning experience. Before this knowledge and experience can be shared with students it is necessary to research and investigate the J2EE technologies and best practice in the application of these technologies. Having researched and investigated J2EE technologies and J2EE patterns, a demonstration application will be developed using Oracle's JDeveloper IDE. This IDE provides a wide range of facilities for developing components through the use of wizards, visual tools, reverse engineering of database tables, etc., or through a combination of these.

The following is an outline of the remaining chapters of the dissertation.

Chapter 2 reviews the literature on design patterns and architectures for J2EE applications. In Chapter 3 the demonstration application is introduced. Chapter 4 provides an in-depth examination of EJBs and related patterns in the format they would be used in the classroom. In Chapter 5 the demonstration application is developed. This consists of a

Java application that acts as an outside agent sending messages to the main application. The main application is developed using components and patterns already developed in Chapter 4, as well as introducing new components and patterns. Chapter 5 also examines the patterns used to implement MVC and describes the view, including the wireless client. Chapter 6 examines how the aims were achieved and indicates future areas of research.

Chapter 2: Literature Review

This chapter discusses various pattern catalogues and also examines various development tools with in-built support for J2EE patterns. It also explores various other types of pattern including archetype patterns.

2.1 Patterns

The use of patterns became popular in the building and construction industry based on the original architectural patterns book, *A Pattern Language: Towns/Buildings/Construction* by Christopher Alexander, (Oxford University Press, 1977). Alexander defined a pattern as follows. *"Each pattern is a three part rule, which expresses a relation between a certain context, a problem and a solution."* He further stated that *"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"*. [23] Although Alexander was referring to problems and their solutions in the building environment, his comments apply equally to object-oriented design patterns. [5]

Patterns became popular in the software community with the publication of the GoF book in 1995. Anyone with experience in software development will get a feeling of déjà vu when she first studies software design patterns. This is because these problems arise in many projects and many developers have provided their own solutions to these problems. [5]

It is important to note that the authors did not invent the various patterns but rather they recognised recurring designs in numerous projects, identified and documented them. Patterns also provide a vocabulary, which improves communication between architects and designers. A designer who does not rely on patterns must expend more effort to communicate a particular design. [2]

2.2 Pattern Catalogues

There are a number of pattern catalogues available including general purpose design patterns, patterns for enterprise systems and patterns that are specific to J2EE.

2.2.1 Gang of Four Patterns

The GoF provided a catalogue of 23 patterns and categorized them as shown below. [5] Although this pattern catalogue was published in 1995, there is still a very strong demand for courses in these patterns. Not only do these patterns help improve design but they also help people learn object-oriented thinking. [3]

Creational	Structural	Behavioral
Factory Method	Adapter (class)	Interpreter
Abstract Factory	Adapter (object)	Template Method
Builder	Bridge	Chain of Responsibility
Prototype	Composite	Command
Singleton	Decorator	Iterator
	Façade	Mediator
	Flyweight	Memento
	Proxy	Observer
		State
		Strategy
		Visitor

Figure 2 : GoF Pattern Catalogue

The GoF also indicated how to describe patterns. They suggested that, although graphical notations were important in describing a design (Object-Modelling Technique, OMT [25] was the modeling language used at the time of publication), it was also important to record decisions, alternatives, trade-offs and concrete examples to describe the design. They suggested the following template. [5]

Pattern Name and Classification	A good name is vital to succinctly describe the pattern.
Intent	What design problem does the pattern address?
Also Known As	Other well known names, if any.
Motivation	A scenario describing a design problem solved by the pattern.
Applicability	Examples of poor designs that the pattern can address.
Structure	A graphical representation.
Participants	Classes and / or objects participating in the pattern.
Collaborations	How the participants collaborate to fulfill responsibilities.
Consequences	What are the trade-offs and results of using the pattern?
Implementation	Hints and techniques for implementing the pattern.
Sample Code	Code fragments to illustrate implementation (C++ and Smalltalk at the time of publication).
Known Uses	Examples of the pattern found in real systems.
Related Patterns	Other patterns that should be used with this one.

As these patterns solve real-world design problems, an experienced developer without any exposure to the GoF pattern catalogue could actually guess the purpose of some of the patterns from their names. Many developers are already using their own versions of documented patterns and strategies without even being aware of it. [5] Whether it is the simple but elegant Singleton pattern (used to ensure only a single instance of a class), one of the Factory patterns (used for runtime instantiation of objects), the Façade pattern (used to hide the complexity of a system by providing clients with a simple interface to the system), the Command pattern (used to encapsulate a request as an object to handle the request) or the Iterator pattern (used to traverse a collection of objects), it is difficult to imagine any non-trivial software project that would not benefit from the use of some of these patterns. Most developers will have devised their own solutions to these problems and it is interesting to compare one's own solutions with those in the catalogue. It is not the intention to reproduce these patterns in this dissertation as they are well catalogued in the text. However, as these patterns are intended to solve general purpose software problems, it is inevitable that some of these patterns will surface in enterprise applications. The GoF patterns that surface in J2EE applications are explained as they arise throughout the dissertation.

Eric Gamma advises against trying to fit as many patterns as possible into a solution and that, where possible, solutions should be kept simple. He suggests that the real benefit of patterns comes from "feeling the pain" of a poor design and that a pattern is only appreciated when one has felt this design pain. He suggests that "throwing patterns into a design" is the wrong way to teach patterns and they should only be added to a solution following a good understanding of the problem they resolve. "Because of this I really like to use patterns after the fact, refactoring to patterns". [3]

Following the publication of the GoF book, research into patterns became popular with research into areas such as tools for automating the patterns (converting the design patterns into code) as well as research into new candidate patterns. The primary sources of this research are to be found at [26, 27, 28]. The hillside [26] web site organises and maintains details of the various PLoP (Programming Languages of Patterns) conferences that take place around the world each year, e.g. PLoP, EuroPlop, KoalaPLoP, Mensore PLoP, SugarLoaf PLoP, Viking PLoP and ChiliPLoP. The ChiliPlop 2003 conference, a conference dedicated to "hot topics", had a discussion theme of "Enterprise Patterns".

2.2.2 Sun's Core J2EE Patterns

With the advent of the J2EE technologies the developers at the Sun Java Centre [29] realized that it was important to provide developers with, not just the knowledge of how to use the various J2EE technologies, but also the knowledge of how to use the technologies well. A blueprint was drawn up on best practices in the use of J2EE. The blueprint was based on the experience of the Sun developers working on a large number of projects and also on the feedback of customers. This blueprint resulted in the presentation of a catalogue of J2EE patterns at the JavaOne conference in June 2000. Following this presentation, a catalogue of 15 patterns was published in the text book Core J2EE Patterns, First Edition [30]. This publication received an overwhelming response from architects and developers and resulted in further work on the patterns culminating in the publication of the second edition of the Core J2EE [2] book with 23 patterns. In addition to the patterns, the book contains a section on design considerations and bad practices and introduces various J2EE refactorings to improve design. Refactoring, first introduced by Martin Fowler, identifies a problem, offers motivation for improving the problem and suggests the means for doing so. [31]

As the focus of this project is on J2EE technologies, the focus is on Sun's Core J2EE patterns, although some patterns not mentioned in Core J2EE are taken from other sources. These patterns are explained as they arise. It is not the intention to reproduce the Core

J2EE catalogue here, but a very brief description of each pattern is given, as these patterns are mentioned throughout the dissertation. The authors divided the patterns in their catalogue into three layers: presentation, business and integration (also know as data access) layers. A brief description follows:

Core J2EE Patterns	Description
Presentation Layer	
Intercepting Filter	Used to intercept a request before it is processed or a response before it is sent. This pattern may be implemented by writing a class that implements the Filter interface which is part of the Servlet API.
Front Controller	Used to provide a centralized point of access for all requests. This pattern is usually implemented as a single Servlet and forms the "controller" part of the MVC architecture. [7]
Context Object	Used to encapsulate platform specific objects, e.g. request, response objects. May be implemented using a simple HashMap.
Application Controller	Used to centralize action and view management. Used in Jakarta Struts framework [35] as part of the MVC architecture
View Helper	Used to separate a view from processing logic (as in MVC)
Composite View	Used to form a page as a composite of other pages using various include directives, e.g. header, footer, menu, body, etc.
Service To Worker	Used to centralize control and to handle request processing logic before dispatching to a view. This pattern is a composition of several other patterns (Front Controller, Application Controller and View Helper).
Dispatcher View	Used when a view is required to handle a request and to generate a response as well as performing simple business processing. Sometimes used to provide a simple testing facility, but of limited use.

Business Layer	
Business Delegate	Used to encapsulate the complexity of remote lookup, exception handling and to provide the client with a clean interface to business services.
Service Locator	Used to locate and cache various services using JNDI (e.g. data sources, home interfaces, queues and topics, etc.).
Session Façade	Used to provide efficient access to the services of one or more EJBs. This pattern provides a coarse grained interface to the EJB(s). One of the most widely used patterns in J2EE. Implemented using a stateless Session Bean.
Application Service	Used to provide use-case logic that applies across several cooperating components by providing clients with an interface to those services. Reduces the business logic in Session Facades.
Business Object	Used to separate business state and behaviour from the rest of the application. Used when an application uses a conceptual domain model with sophisticated business logic and relationships. May use POJOs with DAO or JDO for persistence or may use entity beans.
Composite Entity	Used to implement the conceptual domain model using CMP entity beans for persistence. Relationships between the beans are implemented using local references.
Transfer Object	Used to provide efficient transfer of data across a network. Usually used in conjunction with a Session Façade pattern to provide coarse grained access to the services of one or more EJBs.
Transfer Object Assembler	Used to build an application model as a composite of Transfer Objects and to provide the client with this model.
Value List Handler	Used to provide clients with an efficient searching and iteration mechanism over a large set of results by caching the results on the server side and only sending part of the list across the network.

Integration Layer	
Data Access Object	Used to encapsulate data access code (e.g. SQL calls) and to provide a uniform interface to persistent data. May use the JDBC API to access relational data.
Service Activator	Used to provide asynchronous access to business services (e.g. POJOs, EJBs). Usually implemented as a Message-Driven Bean.
Domain Store	Used to separate the persistence mechanism from the object model. Used to avoid the use of entity beans or when the application is required to run outside a J2EE server.
Web Service Broker	Used to expose Web Services using XML and web protocols.

Figure 3 : Core J2EE Pattern Catalogue

The above patterns do not usually occur in isolation and the authors offer various different strategies for applying the patterns, e.g. The Command and Controller strategy for implementing Front Controller. Many of these patterns, and the strategies for using them, involve the use of GoF patterns, e.g. Session Façade is just a special type of Façade; the Command and Controller strategy may use GoF's Command, Factory and Singleton patterns; DAO may use Factory Method or Abstract Factory, etc.

2.2.3 Other Enterprise Pattern Catalogues

Other well known enterprise catalogues include Fowler's catalogue [4] and TheServeSide.com [32] catalogue. Fowler's catalogue includes some of the Core J2EE patterns, e.g. Front Controller, Application Controller, Data Transfer Object, Remote Façade (Business Delegate), Domain Store. However, as Fowler's patterns are not applied to any particular platform, e.g. his patterns do not use EJBs, many of the patterns in his catalogue are about object persistence as the pattern names suggest, e.g. Active Record, Association Table Mapping, Lazy Load, Repository, Row Data Gateway, etc.

TheServerSide.com is an online community for enterprise Java architects and developers. It provides news, articles, interviews and a discussion forum on enterprise development and design patterns. A pattern catalogue evolved and this catalogue was published as a text book, *EJB Design Pattern* by Floyd Marinescu, (Wiley, 2002). [33] Many of these patterns are similar to Sun's Core J2EE patterns and also make use of some of GoF's patterns as their names suggest, e.g. Session Façade, EJB Command, Data Transfer Object, EJBHomeFactory, Business Delegate. Marinescu also provides various patterns and strategies for persistence and transaction management, e.g. JDBC for Reading, Data Access Command Beans, Primary Key Generation strategies. TheServerSide.com site provides a forum for developers to discuss existing patterns, provide alternative strategies for implementing those patterns as well as presenting their own candidate patterns. In [10], John Crupi suggests that what is now required for the pattern repository on TheServerSide is to set up a review process to look at the various patterns and perhaps bring those patterns to a wider community.

2.3 Architectures for Web Applications

The simplest Java-based web applications consist of a series of JSPs with all presentation logic, control logic, business logic and data access code embedded in the JSPs as shown in the diagram below. A large amount of code is duplicated in many of the JSPs. This type of application becomes a maintenance nightmare when even the smallest changes take place, e.g. a change to a single database table, could involve a change to every JSP that accesses that table. Two well known architectures are used to solve some of the problems associated with this type of solution.

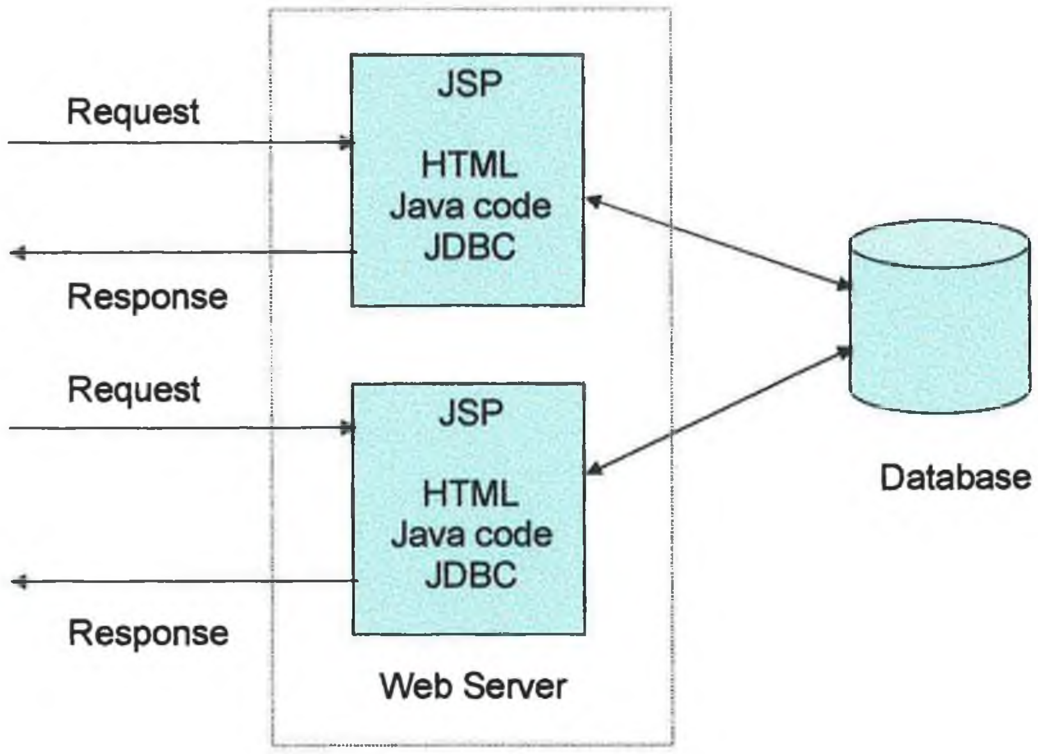


Figure 4 : Simple JSP Application

2.3.1 JSP Model 1 Architecture

Early Java-based web applications used a Model 1 architecture as illustrated in the diagram below. A Model 1 architecture consists of a browser directly accessing JSPs, with the JSPs accessing JavaBeans that represent the application model. The next view to display is determined by links selected in the current page. Control in a Model 1 application is decentralised, as the current page determines the next page to display. In addition, each JSP or Servlet processes its own input parameters, i.e. request parameters. In this model a JSP mixes presentation code (HTML), control logic and business logic (Java code), i.e. the view and controller are mixed in each JSP. As indicated in Sun's Blueprints for web-tier design, this architecture quickly breaks down when used with larger applications. [7]

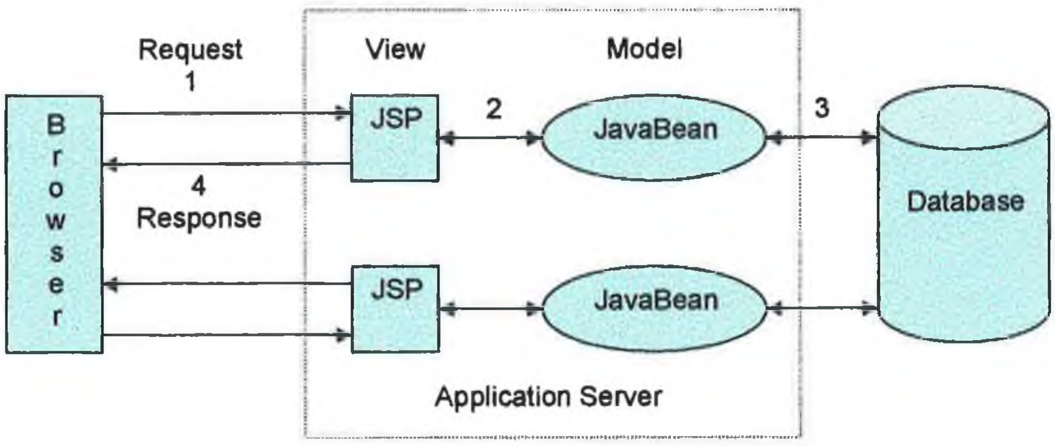


Figure 5 : JSP Model 1 Architecture

2.3.2 Model-View-Controller Architecture

The Model-View-Controller (MVC) paradigm was originally used to build user interfaces in Smalltalk-80 and is cited in early literature, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80", G. E. Krasner and S. T. Pope, (JOOP, Vol 1, no 3, August/ September, 1988), but has since been much more widely used. [34] The *model* represents the data in an application, the *view* represents a client's view of the data and the *controller* interacts between the user and the view. The data element is developed independently of the view, i.e. the data should be unaware of how or where it will be displayed. This separation of the data and the view allows various different views of the same data. Any changes to the data should be reflected in a change to the view(s). A variation on MVC is the Document-View paradigm used in Microsoft

Visual Studio where the view and the controller are both represented in the view and the model is represented in the Document.

The MVC architecture, when applied to web applications, is sometimes called the Model 2 architecture. Except for the smallest applications, MVC is the recommended architecture for developing interactive web applications. [7] MVC separates an interactive application into three separate modules.

1. The model maintains the application's data representation and business logic.
2. The view provides for data presentation and user input.
3. The controller is used as a central point of access for all requests and it controls the flow of the application.

The MVC design pattern provides many benefits. MVC allows for separate development of the business layer, data access layer and the presentation layer. This allows developers with different skills to work on the layer appropriate to their skills. It also allows developers with different levels of experience to work at a level appropriate to that experience, e.g. a junior programmer may be introduced to a project by developing views of data gathered in another layer and made available through a well defined interface. MVC centralizes control and makes it easier to add new data sources or new views to an application. [7] These benefits are illustrated in the application developed for this project, e.g. a wireless client is seamlessly integrated into the application without any changes to the other layers of the application.

There are a variety of strategies for implementing the MVC architecture. In its simplest form, a Front Controller (usually a single Servlet) is used to handle all requests from the client. All client requests are directed to the Front Controller. To ensure that clients do not attempt to directly access JSPs, a simple strategy is to place all JSPs in a folder below the WEB-INF folder of the web module. These JSPs cannot be directly accessed by clients. This means that each JSP does not have to contain code to limit access. [2]

Requests to the Front Controller could originate in an ordinary web browser running in a PC or requests could originate in a wireless device. Irrespective of the source of the request, the Front Controller passes the request to some Request Handler object which may communicate with various other resources (e.g. EJBs to access the database), to handle the request. The Front Controller then passes the requested information to a JSP to render the information to the client. Depending on the origin of the request, control is passed to the appropriate JSP to display the results in a web browser or in a mobile device. Although a single controller may be used for web browser and wireless clients, to allow for different protocols that may arise in new devices, a separate controller could be added for each new type of device. [7] The following diagram illustrates how the MVC architecture may be used in a web application to process requests from a client originating in a web browser or in a wireless device. The controller will forward control to the appropriate JSP depending on the type of device used. This means that the only difference between the code written for a web browser and a wireless device is the code in the JSPs.

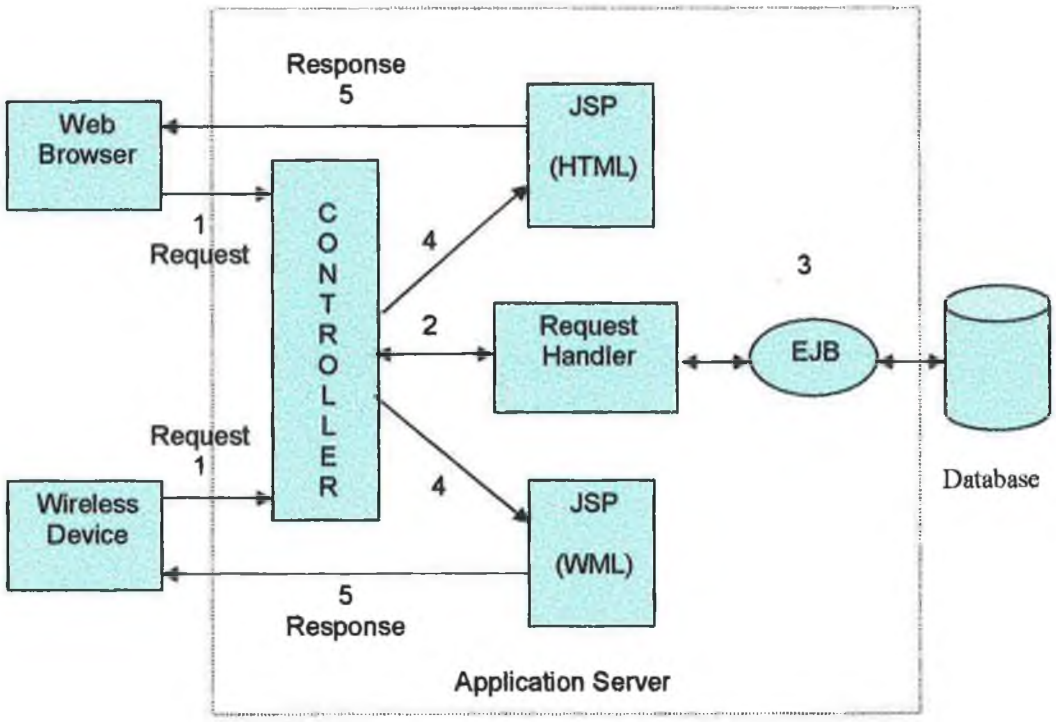


Figure 6 : Model-View-Controller Architecture

1. The client, using a web browser or a wireless device, sends a request.
2. The controller (possibly a Servlet) selects the appropriate object to handle the request. This object may be a Command [5] object created using a Factory [5] object.
3. The request handler communicates with the database (e.g. using an entity EJB, a DAO [2], etc.) to obtain the requested data from the database.

4. The controller dispatches control to the appropriate JSP (e.g. a JSP using HTML to render the view in a web browser or a JSP using WML to render the view in a wireless device) to display the requested information.
5. The information is displayed in the client's device.

The MVC architecture may be implemented in a J2EE application by using a variety of design patterns and by employing a variety of strategies. One such strategy is known as the Command and Controller strategy [2]. An implementation of this strategy is illustrated in project's application and is shown in Appendix B. Some of the patterns used include GoF's Singleton, Command, Factory [5] and Core J2EE's Front Controller and Context Object [2]. This strategy also uses Token Synchronizer [2] to avoid duplicate request submissions.

2.4 Patterns and Frameworks

A lot of effort is required from individual developers to write applications using various patterns and strategies, and this effort also requires a lot of repetitive work, as the same patterns are re-applied using different strategies. A number of different frameworks and IDEs implement various patterns and strategies to relieve the developer from the repetitive work of developing the various patterns and thus allowing the developer to focus on implementing the business logic of the application.

2.4.1 The Struts Framework

One framework that attempts to reduce the effort in building web applications is the Jakarta Struts framework from the Apache organization. The Struts framework encourages application architectures based on the MVC architecture and provides a template for the separation of model, view and controller. The Struts framework provides its own Controller component, known as the ActionServlet, and integrates with other technologies to provide the Model and the View. For the model, Struts can interact with standard data access technologies, like JDBC, EJB, as well as most third-party packages like Hibernate [12], iBatis [36], or Object Relational Bridge [37]. For the view, Struts works well with JSP and other presentation technologies. Struts uses an XML configuration file, struts-config.xml, to initialize its resources. These resources include ActionForms to collect input from users, ActionMappings to direct input to server-side Actions (Command classes [5]) and ActionForwards to select output pages. [35]

In employing an ActionServlet as the Front Controller and an Action class as the Command Handler, the Struts framework is an application of the Front Controller pattern, using the Command and Controller strategy. [2] Struts also uses the Synchronizer Token [2] to prevent duplicate request submissions from a client. There are a number of situations when a duplicate request can occur, e.g. a user refreshes a page or a user presses the Back button and then re-submits the request. The Action class maintains the actual token. An example of Synchronizer Token is shown in Appendix B.

It is not recommended that the Struts framework be used for all web applications. For small applications with just a handful of pages, it is recommended to use a Model 1 architecture. [35]

So popular is the Struts framework that many different IDEs, including JDeveloper and OptimalJ, use it to implement the MVC architecture.

2.4.2 JDeveloper

Oracle JDeveloper 10g (v.10.1.2) is an integrated development environment (IDE) for building J2EE applications. JDeveloper supports the complete development life cycle with integrated features for modelling, coding, debugging, testing, profiling, tuning and deploying applications. JDeveloper comes with a built-in application server, OC4J 10g (v. 10.1.2). JDeveloper should work with most SQL-compliant database servers (although the author encountered problems with MySQL), but it is certified to work with DB2 [38] and Microsoft SQL Server[39] as well as Oracle databases. In addition to deploying J2EE applications to Oracle Application Servers, JDeveloper is certified to deploy to WebLogic [40] and JBoss [41] servers.

JDeveloper comes bundled with the Struts framework to implement the MVC architecture, though the developer has a choice of using this framework or not. As Oracle wanted a single environment that would adapt to any developer [42], JDeveloper provides the developer with a variety of templates for developing J2EE applications. When the developer chooses a particular template, the technologies, e.g. JSP, EJB, Struts, etc., named in that template are made available to the developer.

JDeveloper may be used for developing Java 2 Standard Edition (J2SE) applications as well as J2EE applications. Using JDeveloper, a developer may develop an application using a visual tool to design the application using UML diagrams (Use Case, Class and Activity diagrams), EJB diagrams, Entity (database) diagrams, Java class diagrams, Web Service diagrams and Business Component (ADF) diagrams. The diagrams may then be used to generate code. Code and diagrams are synchronized as any change to the model is reflected in the code, or vice versa. JDeveloper may be used to reverse engineer CMP entity beans from existing database tables. This technique is used in the project's application.

As indicated in the JDeveloper Documentation, JDeveloper facilitates the use of J2EE patterns by providing the facility to generate an application that uses the MVC architecture and also allows for easy addition of the following J2EE patterns.

1. The Session Façade pattern centralizes complex interactions between lower-level EJBs (often entity beans). It provides clients with a simpler interface to the business services of the application.
2. The Data Transfer Object (aka Transfer Object [2]) pattern provides better maintainability by separating use cases from the object model, allows for reuse of entity beans across different applications, and increases performance when the attributes from multiple entity beans can be passed to the client with a single call.
3. The Business Delegate [2] pattern decouples clients and business services, hiding the underlying implementation details of the business service. This pattern is implemented by the data control, which is represented in JDeveloper by the Data Control Palette.

Of course, in order to use these patterns with JDeveloper, it is necessary to understand the patterns. In addition to these patterns, a developer with knowledge of patterns can apply additional J2EE patterns as appropriate, although there is no specific facility for this. As a tool for teaching J2EE technologies and patterns, JDeveloper could be used to develop an application and then through refactoring, appropriate patterns could be added, as recommended by Gamma [3]. This is the approach used for the application developed for this project.

JDeveloper also allows for the addition of extensions to the IDE by providing a facility to add third party tools to the IDE, e.g. JUnit [44].

2.4.3 Oracle Application Development Framework

JDeveloper also includes the Application Development Framework (ADF). ADF simplifies J2EE development by minimizing the need to write code that implements J2EE design patterns. ADF is a set of libraries that runs on standard J2EE application servers, not just Oracle application servers. Its job is to reduce a lot of the complexity in building a J2EE application and help developers focus on the business logic. [42]

ADF uses the Struts framework to implement the MVC architecture and is based on four layers:

1. The View layer provides the user interface to the application.
2. The Controller layer controls the flow of the application.
3. The Model layer provides an abstraction layer on top of the Business Services layer, enabling the View and Controller layers to work with different implementations of Business Services in a consistent way.
4. The Business Services layer provides access to data from various sources and handles business logic.

The following diagram illustrates these layers.

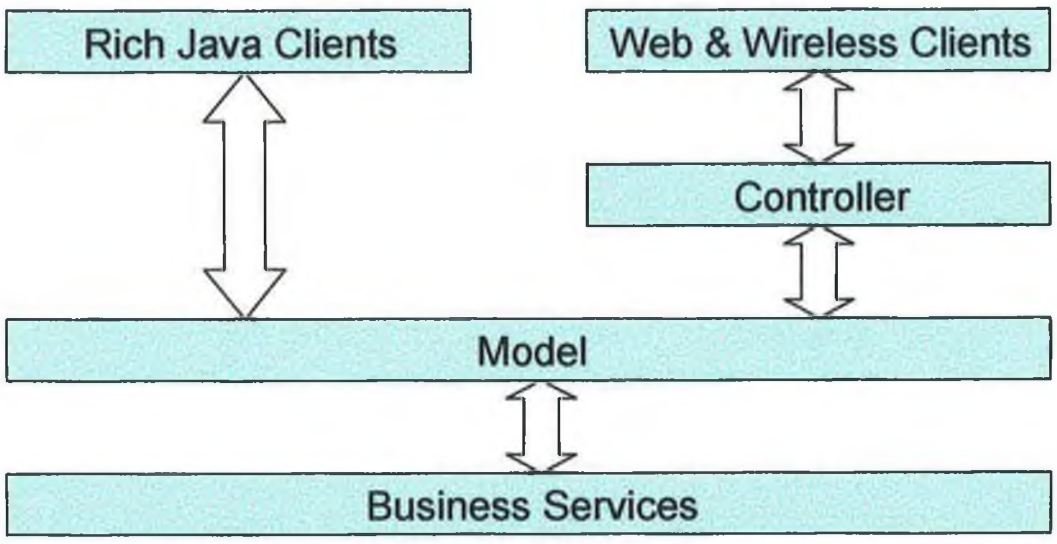


Figure 7 : Oracle ADF Layers

ADF implements many of the patterns from Sun's J2EE Design Pattern catalogue and adds some others of its own as shown below and described in "ADF Business Components J2EE Design Pattern Catalogue", Steve Muench, ADF Development Team (June, 2005). [45]

ADF Pattern	Description, or Also Known As
MVC	Implemented using Struts
Interface/Implementation Separation	ADF Business Components enforce a logical separation of the client's view (via Java interfaces) and the business tier implementation of those interfaces.
Service Locator	Service Locator [2]
Inversion of Control	ADF components contain a number of easy-to-override methods that the framework invokes at the appropriate time. The developer may override these methods without being concerned when the methods will be called.
Dependency Injection	All ADF components are configured in external metadata (data about data) configuration files. The framework automatically injects dependent objects (e.g. view objects into service components or entity objects into view rows at runtime) at runtime.
Active Record	Active Record [4]
Data Access Object	Data Access Object [2]
Session Façade	Session Façade [2]
Value Object	Transfer Object [2]
Page-By-Page Iterator	Value List Handler [2]
Fast-Lane Reader	Allows for efficient access to read-only data by retrieving only the required fields using JDBC instead of entity beans. The developer need only write the required SQL statements in an XML file.
Front Controller	Front Controller [2]
(Bean) Factory	Allows for runtime instantiation of beans based on XML configuration files and through Factory [5] classes.
Entity Façade	Provides a restricted view of data and behaviour of one or more entity objects.
Value Messenger	Used to keep client Value Objects in synch with business entity data. Client attribute changes are updated in the entity objects, and changes to entity objects are updated in client Value Objects.

Figure 8 : ADF Pattern Catalogue

The fact that JDeveloper provides such a variety of development templates as well as the ADF, means that there is a steep learning curve in learning to use the IDE. However, it allows developers with various skill levels to apply those skills in the most appropriate manner without being restricted to any particular template. Also, by initially focusing on a single template, students can quickly develop an application and then refactor the solution by the application of appropriate patterns.

2.5 Model Driven Architecture (MDA)

2.5.1 What is MDA?

The Object Management Group (OMG) [46] launched an initiative known as Model Driven Architecture (MDA) in 2001 [47]. MDA is a software architecture based on modelling at different layers. OMG provides the following definitions in its specification for MDA [47]

"A *model* of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language".

"*MDA* is an approach to system development, which increases the power of models in that work. It is *model-driven* because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification".

"The *architecture* of a system is a specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors. The Model-Driven Architecture prescribes certain kinds of models to be used, how those models may be prepared and the relationships of the different kinds of models."

"A *platform* is a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented."

The following definitions are provided in *Enterprise Patterns and MDA (Building Better software with Archetype Patterns and UML)* by Jim Arlow and Ila Neustadt, (Addison-Wesley, 2004). [48]

"An *analysis* class represents a crisp abstraction in the problem domain and maps onto real-world business concepts".

"A *design* class is a class whose specification is complete to such a degree that it may be implemented. It incorporates features from both the problem domain and the solution domain (implementation technology)".

This means that an analysis class is independent of any programming language, technology or platform that may be used in the implementation of the class. A design class on the other hand, uses knowledge of its implementation environment (e.g. J2EE, .NET, etc.) to describe its solution.

MDA involves developing a Platform Independent Model (PIM) of an application's business functions. This model is usually developed using the Unified Modelling Language (UML) and describes analysis classes and the associations between them, i.e. the PIM is developed without knowledge of any technologies that will be used in the development of the application. The MDA approach, and the standards that support it, allow the same model specifying a system's functionality to be realized on multiple platforms and provide for system evolution as platform technologies come and go. [47]

An MDA-enabled tool transforms the PIM into a Platform Specific Model (PSM) for the target platform. The target platform could be, for example, J2EE or .NET. The tool would then generate an implementation of the model in a specific language, e.g. Java or C#. The PIM would remain stable and could be used with another MDA enabled-tool to transform the model to a different PSM and a different programming language. In this way the PIM is not subject to change when the technology changes. When an MDA-enabled tool is available, the PIM is said to be an *executable model.*

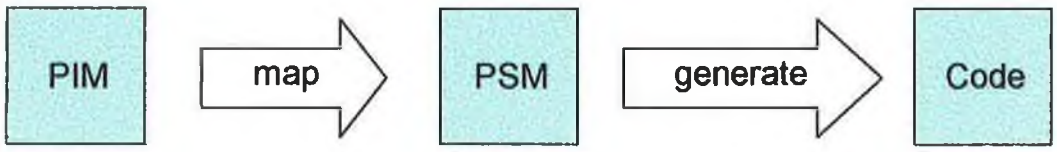


Figure 9 : Model-Driven Architecture

2.5.2 Agile MDA

The reliance of MDA on developing a PIM to drive the development would seem to imply that MDA is only suited for traditional "heavy" development methodologies. It is not the intention to enter the debate of Agile Development versus Heavy Methodologies. The reader is referred to [49] for a comparative study.

With an appropriate MDA-enabled tool, MDA may also be used for agile development. An executable model, because it is executable, can be constructed, run, tested and modified in short incremental, iterative cycles. Agile MDA is based on the notion that code and executable models are operationally the same, so the principles of the Agile Alliance [51] (testing first, immediate execution, moving from analysis to implementation in short cycles), can be applied to models. An executable model, because it is executable, can be constructed, run, tested and modified in short incremental, iterative cycles. "Agile MDA", Stephen J. Mellor, Project Technology, Inc. [50]

2.6 OptimalJ and MDA

OptimalJ [22] is an MDA-enabled tool that adds a pattern-driven development paradigm on top of the MDA concepts. [1] OptimalJ sits on top of NetBeans [19] and incorporates the Struts Framework and also DreamWeaver [52] for generating web content. OptimalJ uses a variety of pattern types, including Sun's Core J2EE patterns, to generate fully working applications from a PIM. OptimalJ also supports agile development by making it easy to quickly do some modelling, and then run the JSP-based application to test the model [1].

2.6.1 OptimalJ Models

OptimalJ considers three different model layers, the Domain Model (i.e. the PIM), the Application Model (i.e. the PSM) and the Code Model.

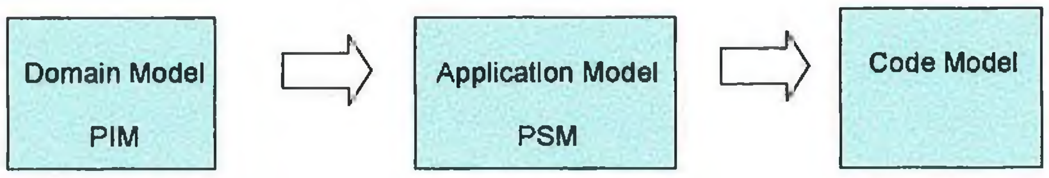


Figure 10 : OptimalJ MDA

Each model layer consists of a number of models as illustrated in the following diagram.

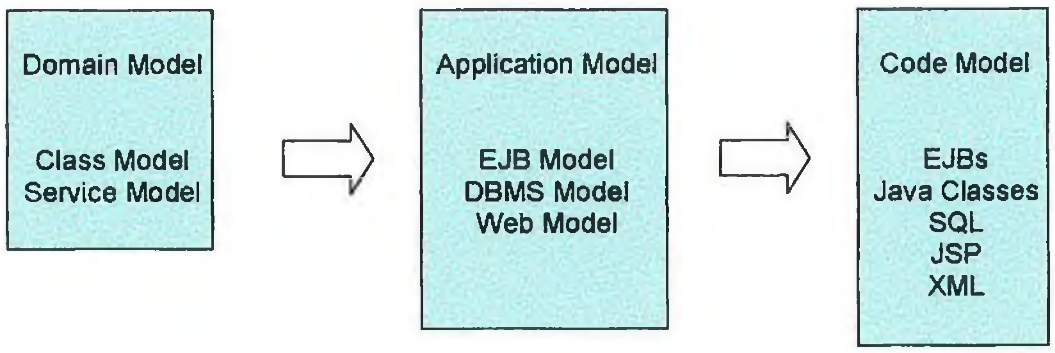


Figure 11 : OptimalJ Models

2.6.1.1 The Domain Model (PIM)

The Domain Model is a UML model describing the business domain without any specific technology or platform detail. This normally consists of a class model and a service model. [1]

1. The Class Model in OptimalJ captures the static structure of a system. The main elements of this model are the domain classes and their associations. This model may be used to generate a complete working application.

2. The Service Model captures behavioral information in the form of domain views and domain services. The Service Model is used to generate EJB session components in the business logic model and web components in the Web Model. At code level, this results in the generation of session beans, JSPs, and other related code. Service operations become methods on the session bean and web actions in the web tier.

From an EJB perspective, domain services may be considered as placeholders at domain model level for EJB session components in the business logic model, and session beans at code level.

2.6.1.2 The Application Model (PSM)

The Application Model defines a specification for the application based on the technologies used. This model provides a logical overview of the various layers, e.g. the business layer (EJB Model), the database layer (DBMS Model) and the presentation layer (Web Model). The Application Model consists of various XML meta-data (i.e. data about data) configuration files and diagrammatic models that form the specification for the system to be developed. [1]

1. The EJB Model provides a framework to make the domain model's data available in a distributed environment in order to deal with transactions, security and persistence. The EJB components consist of entity components, data schemas, and other EJB-related components.
2. The DBMS Model in OptimalJ is relational. This model consists of various XML files as well as an entity diagram showing the various tables derived from the Class Model. This entity diagram shows the primary keys and foreign keys generated by OptimalJ representing the various constraints.
3. The web model contains data schemas which are used by web components to exchange data with the EJB layer and to exchange data with the user. [1]

2.6.1.3 The Code Model

The Code Model is easy to understand as it consists of the actual code generated for the application. This code consists of various EJBs and Java classes (the business layer), SQL scripts for generating database tables (the database layer) and the JSPs (the presentation files) as well as various deployment descriptors (XML files) required to configure J2EE applications ready for deployment. [1]

2.6.2 OptimalJ Patterns

The only effort required by the developer to generate a simple data maintenance application with a facility to insert, update and delete database rows is to create the Domain Class Model. OptimalJ generates the other models. This type of tool is particularly useful for developing data-centric applications. [42]

Crupi and Baerveldt [1] in their paper describe OptimalJ, the various patterns used to transform the models as well as the Core J2EE patterns implemented by OptimalJ. After defining the Domain Model, OptimalJ uses its pattern-driven generator to rapidly create an application according to Sun's Core J2EE patterns. Patterns are also used to create the implementation classes in the presentation, business logic and persistency layers. The following diagram illustrates these patterns.

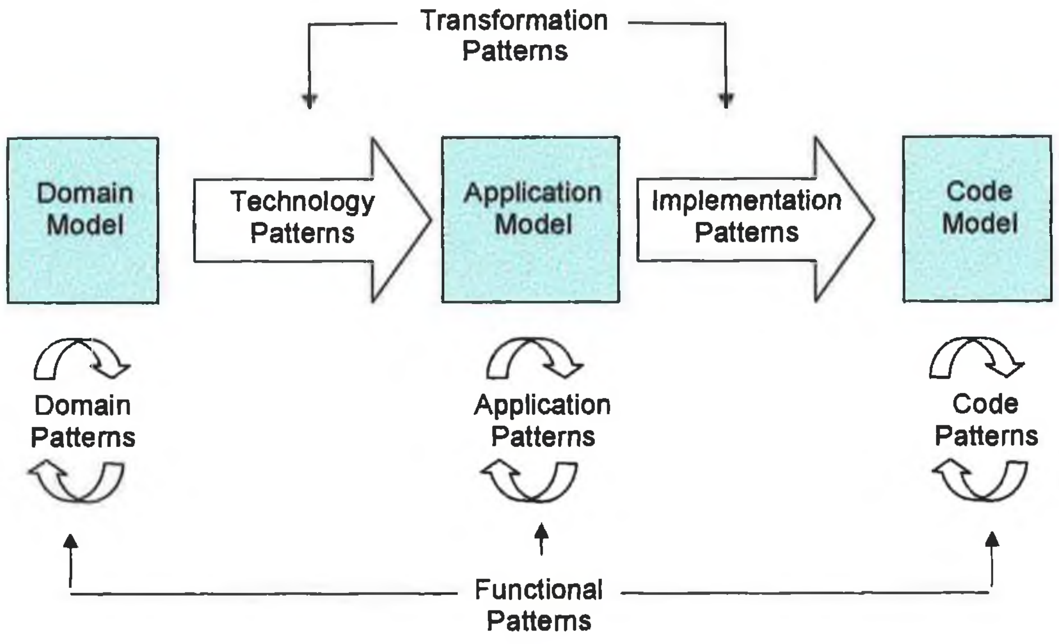


Figure 12 : OptimalJ Patterns

2.6.2.1 Transformation Patterns

Transformation patterns are used to transform elements from one model to elements in a lower level model as shown in the diagram. There are two types of Transformation pattern.

(i) Technology patterns are used to transform the Domain Model to a platform specific technology. As OptimalJ uses the J2EE specification, the Technology Patterns used are J2EE patterns. Technology patterns do not generate any code.

(ii) Implementation Patterns are used to transform application specifications, in the form of XML files, to the actual code used in the Code Model. These patterns are used to determine the actual Java classes required to implement each EJB (e.g. home and remote interfaces, the bean class, Primary Key class, etc.). The implementation patterns generate code based on Sun's Core J2EE Patterns, adhering to the de facto design standard. [1]

2.6.2.2 Functional Patterns

Functional patterns are used to increase developer productivity by re-using parts of a model at a single model level.

(i) Domain patterns are UML models that can be re-used (sometimes with modifications) in different applications, e.g. a Customer class used in one Domain Model could be re-used in another application, or two domain models could be combined to form a new model in a different application.

(ii) Application patterns allow the re-use of artifacts from the Application Model of one application in other applications.

(iii) Code patterns are simply code templates that can be re-used, the most common types being implementations of various GoF patterns [1].

2.6.2.3 OptimalJ J2EE Patterns

The Core J2EE patterns implemented in OptimalJ and certified to conform to Sun's Blueprints [1] are as follows:

Core J2EE Patterns	Implemented in OptimalJ using
Presentation Layer	
Intercepting Filter	Struts
Front Controller	Struts
View Helper	Struts
Composite View	JSP templates and includes
Service To Worker	Struts
Business Layer	
Business Delegate	Business Façade
Service Locator	Helper class
Session Façade	Business Façade
Transfer Object	DataClass
Object Assembler	Compound DataClass
Value List Handler	Page Iterator
Composite Entity	Composite Entity Bean
Integration Layer	
Data Access Object	DAO
Service Activator	Message-Driven Bean

Figure 13: OptimalJ Core J2EE Patterns

"The way in which OptimalJ makes use of patterns is impressive. Although the patterns are not directly visible to the end user, they make the generated code easy to understand. All patterns in the reviewed version of OptimalJ are implemented correctly in compliance with Sun's Core J2EE Pattern Calalogue". [1]

2.6.3 OptimalJ Application

In order to get a better understanding of OptimalJ, the patterns used and models generated, a 30-day trial version of OptimalJ Professional Edition was obtained. As part of the application developed for this project, a fully working application implementing the maintenance use cases (insert, update delete) for a number of related tables was generated from a simple Domain Class model. Sample screen shots are shown in Appendix F.

2.7 JDeveloper and MDA

Oracle's position on the MDA approach to application development is that it can only be effective if it enables code-oriented developers to work seamlessly with those that prefer visual representation. JDeveloper v10.1.2 supports the UML Class, Use Case and Activity diagrams. UML Sequence diagrams will be supported in v10.1.3. These diagrams permit an application to be expressed in a technology independent way. The Class diagram may be used to develop a PIM. In addition, JDeveloper supports a set of technology focused modellers (Java, EJB, Entity, Business Component, Web Service models) to express the detailed application design, i.e. a PSM. A "MDA style of working" is supported by capabilities for linking and transforming between the PIM and the PSM. The PSM is transformed into code to produce Java classes, EJB components, etc. However, JDeveloper does not provide for complete application development based on a PIM.

In a statement of direction, Oracle has stated that full implementation of MDA in JDeveloper will be provided by allowing third party extensions to JDeveloper. [54] Oracle and Softeam have collaborated to produce Objecteering which works with JDeveloper to provide an MDA-enabled tool for generating working J2EE applications from UML diagrams. Objecteering has not been examined in this project.

2.8 Archetype Patterns

Just as design patterns are a recurring theme in many applications, at a higher level of abstraction, many business concepts (e.g. customer, product, order, etc.) occur in many business domains, possibly with variations in the concepts across different domains, e.g. a "customer" in the health sector is a variation of a "customer" in the education sector. [48]

The following definitions are given by Arlow and Neustadt. [48]

"A business archetype is primordial thing that occurs consistently and universally in business domains and business software systems".

"A business archetype pattern is defined as a collaboration between business archetypes that occurs consistently and universally in business environments and software systems".

Archetypes are at a higher level of abstraction than normal analysis classes, as archetypes are concerned with the recognition and capture of *universal* concepts, whereas analysis classes are not necessarily concerned with universality. In order to address the issue of universality, Arlow and Neustadt introduce *the principle of variation*: different domains often seem to require different models of the same thing. This means that a single archetype may generate one or more analysis classes, e.g. a single Customer archetype may generate several different analysis classes for Customer depending on the problem domain. Arlow and Neustadt suggest two ways in which an archetype may be varied.

(i) Some new features may be added
(ii) Optional features may be omitted.

In order to deal with this variation the core semantics of an archetype must remain fixed for every variant of the stereotype. Arlow and Neustadt introduce the stereotype «o» to indicate that a feature is optional and may be omitted without violating the core semantics of the archetype.

2.8.1 Archetype Pattern Catalogue

Arlow and Neustadt have produced a catalogue of archetype patterns which includes Party, PartyRelationship, Customer Relationship Management, Product, Inventory, Order, Quantity, Money, and Rule archetype patterns. Each pattern consists of certain essential parts required to form a consistent pattern, and several optional parts that may be omitted to suit a particular problem domain. According to Seve Vinosky, Chief Engineer of

Product Innovation, IONA Technologies, "*the patterns presented here have the potential to impact business applications in the same way the Gang of Four patterns have impacted general software development*". [53]

The following is an incomplete example taken from the work of Arlow and Neustadt. The diagram illustrates an incomplete Money Archetype Pattern which consists of several archetypes with optional attributes, optional operations and an optional relationship. This simple example illustrates how, in attempting to capture universal concepts that apply to a variety of domains, even a simple archetype pattern becomes quite complicated.

2.8.2 Money Archetype Pattern

The following definitions are given in [48].

"Money is an official or commonly accepted medium of exchange that can be used to buy goods and services".

"The Metric archetype represents a standard of measurement".

"The Quantity archetype represents an amount measured in some Metric".

"The Locale archetype represents a general notion of place, location, or context".

"The Currency archetype represents a Metric or standard of value for measuring Money".

"The Money archetype represents an amount of a specific Currency. This Currency is accepted in one or more Locales".

By omitting some, or all, of the optional features, or by adding new features, different analysis classes for the Money archetype could be created.

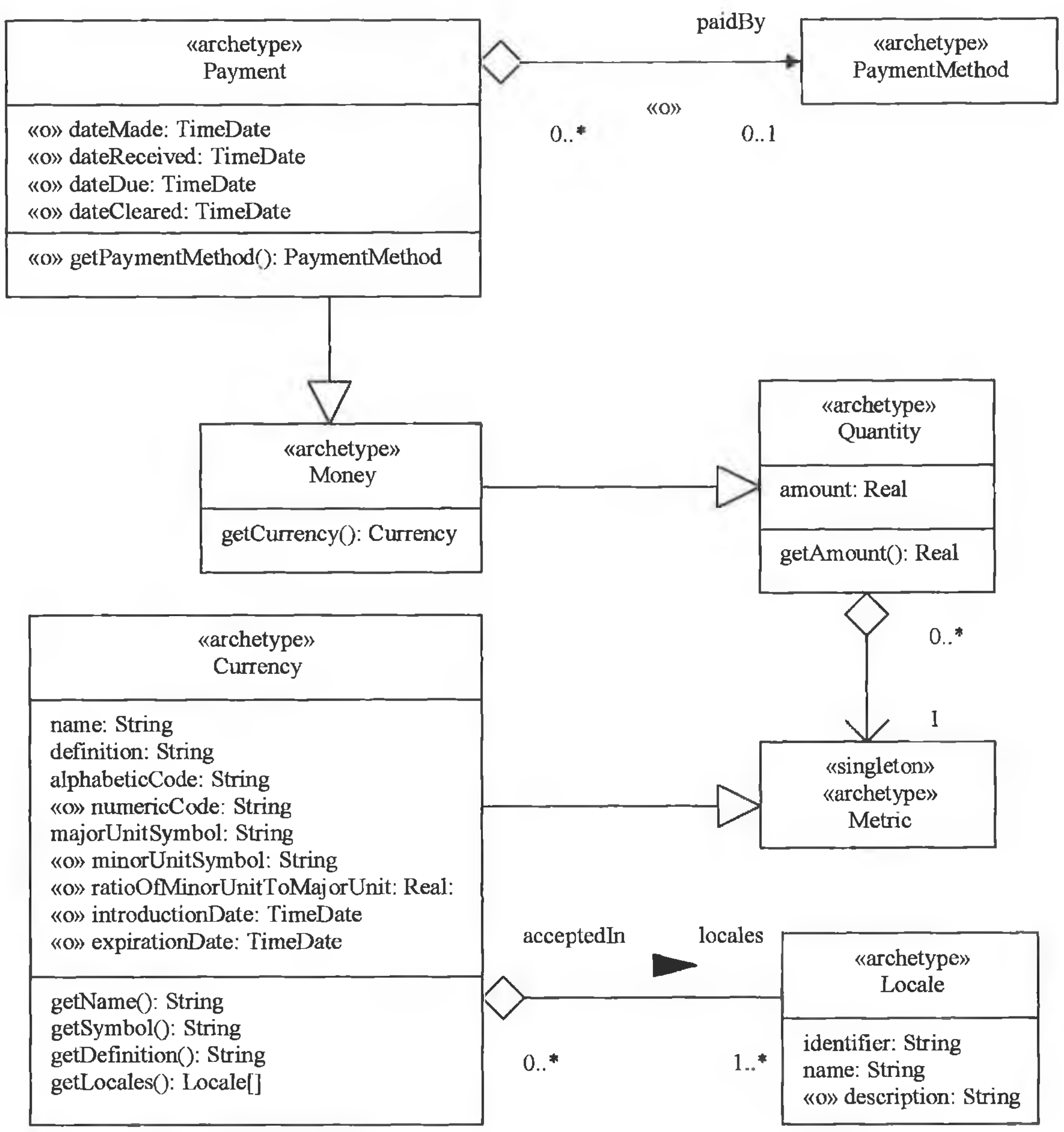


Figure 14: Money Archetype Pattern

2.8.3 Archetype Patterns and MDA

Instead of creating a PIM from scratch, if an archetype pattern catalogue is available, it is easier to adapt an archetype pattern from the catalogue to create the PIM. By omitting some or all of the optional features, or by adding new features, a suitable Platform Independent Model can be created with, or imported into, an MDA-enabled tool such as

OptimalJ. The following diagram illustrates how archetype patterns may be adapted and fed into an MDA-enabled tool such as OptimalJ. [48]

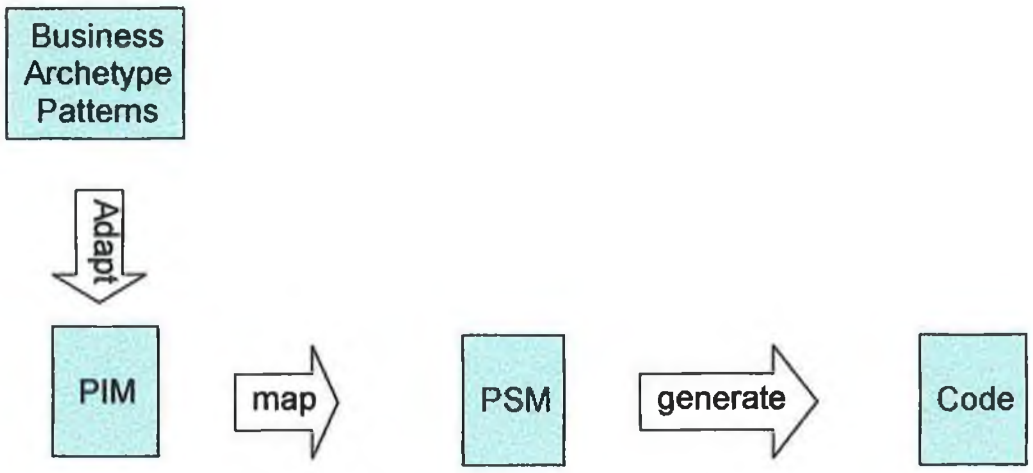


Figure 15: Archetype Patterns and MDA

2.9 Summary

Initial investigation into various design pattern catalogues and into architectures for web applications lead to an investigation of the integration of these patterns and architectures into various development tools, including an MDA-enabled tool. The incorporation of patterns into development tools is a clear vote of confidence in the benefits of these patterns.

The availability of sophisticated development tools means that experienced J2EE developers can quickly develop and test components and integrate and deploy these components to produce reliable applications. The ability to adapt Archetype Patterns and to feed these patterns into MDA-enabled tools further increases productivity. This ability to quickly develop reliable J2EE applications should mean an increase in the use of J2EE technologies and an increase in the demand for experienced J2EE developers.

Chapter 3 introduces the application that will be used to teach the various J2EE technologies and J2EE patterns. This application is used throughout the remainder of the dissertation.

Chapter 3: Defining the Demonstration Application

To provide students with a good learning experience, an application is required that will bring the student through the development of the application starting with standalone components, progressing to more complicated components and finally to the integration of these components. Throughout the development of the application it is important that students are encouraged to recognise design faults and invited to propose solutions to these design faults. It is only when students have a good grasp of the design issues that patterns are introduced as a possible alternative to the students' own solutions. For this to succeed it is important that the application is sufficiently interesting to engage the students for the duration of the course. The following application meets these requirements.

3.1 The Problem Statement

A business lecturer requires an application that she will use as part of a course on stock exchange trading. She requires the application to allow a large group of students (up to 350), known as players, to trade for short periods, e.g. the duration of a class period. At the beginning of each trading session each player is given the same starting balance, e.g. €50,000. During the session players may purchase or sell shares at the current share price assuming they have sufficient funds. It is assumed that the buying and selling prices are the same. At the end of each session all stock is sold at the current share price and each student's profit or loss for the session is calculated. A league table is maintained showing the profit or loss for each game played. The league table should be maintained over several sessions. A player should be able to view the league table, in descending order of profit, at any time as well as her own list of games.

As the trading sessions are short, the application is required to simulate changes to the share prices at short intervals, e.g. every 60 seconds. A player should be able to view the current share prices at any time. A player should also be able to view the content of her portfolio at any time.

A record of all players, including codename, password, last name, first name and department should be maintained by the system, but it is not necessary to maintain the players' trading record. Players are required to login to gain access to the system. Players should have the facility to register their details, amend their details and delete their details.

A record should be kept of all companies including symbol, name, share price, highest price and lowest price.

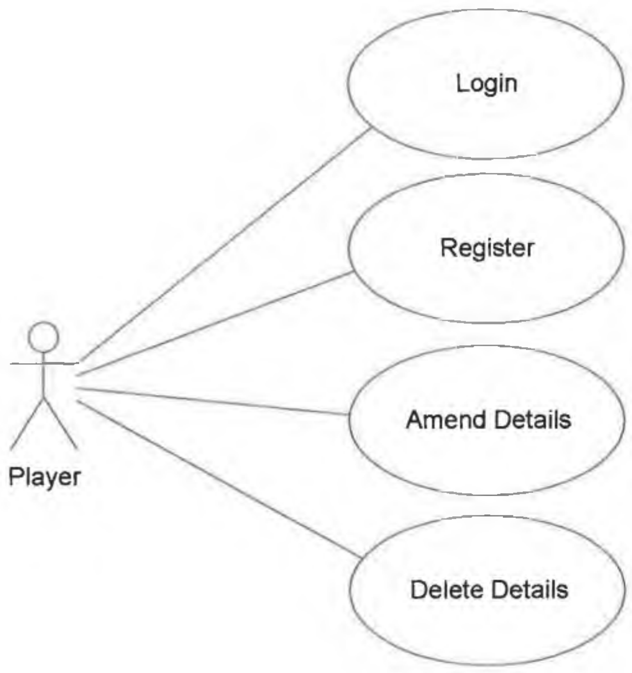
The league table should hold the player, the closing balance and the date and time of each game played.

A facility should also be provided to allow a player to login using a mobile device and to view the share prices as well as details of her own games.

3.2 Use Cases

It should be remembered that the purpose of this project is to teach students various J2EE technologies and various J2EE patterns. In particular, the emphasis is on EJB components and related patterns as well as on the architecture for the application, i.e. the emphasis will be on the model and controller modules and not on the view module. **Therefore, it is not intended to develop a full production application**. Rather it is the intention to use this application to introduce various J2EE technologies and then to improve the design by introducing various J2EE patterns.

With this in mind the following use cases are developed.



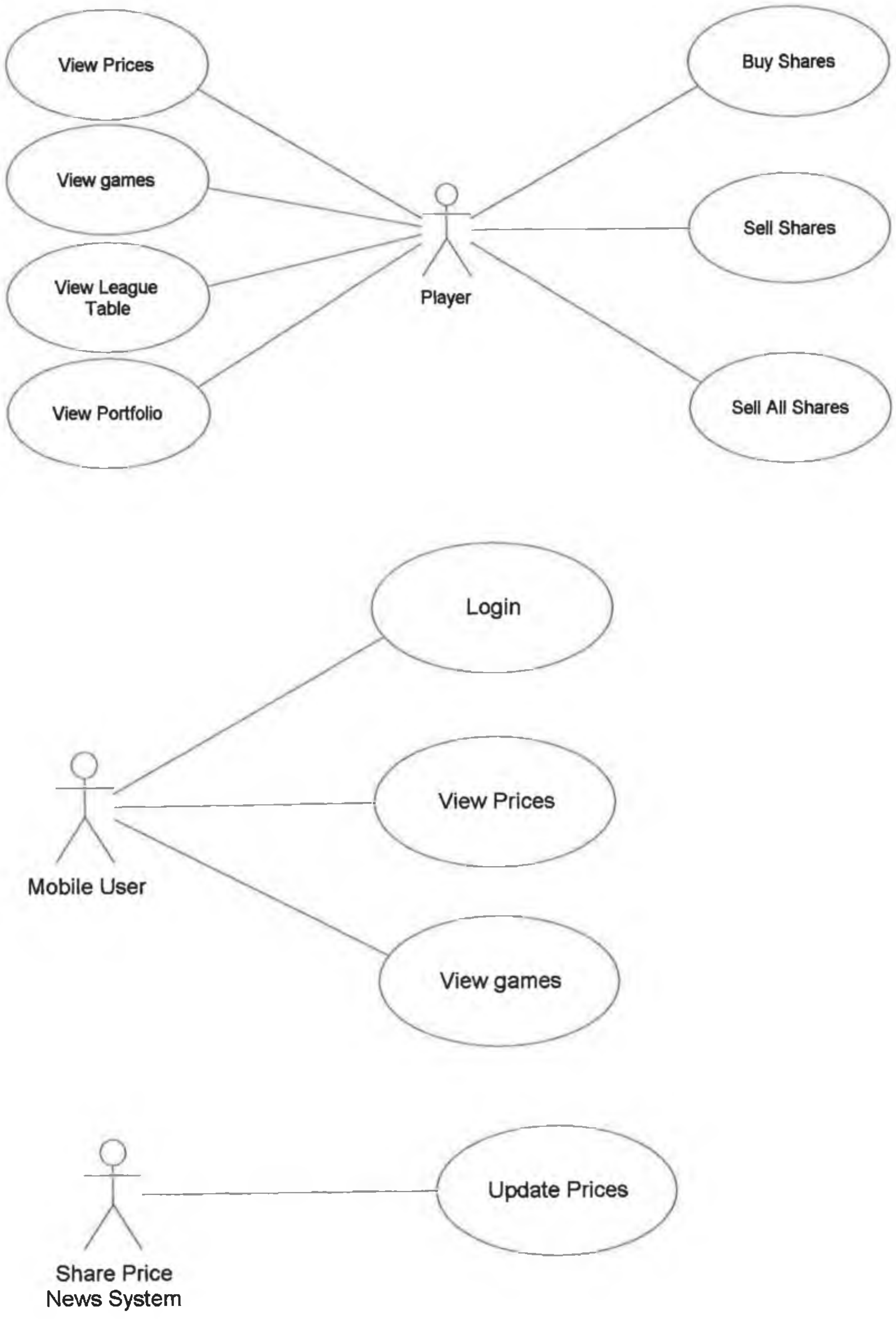


Figure 16: Use Case Diagrams

3.3 The Entity Model

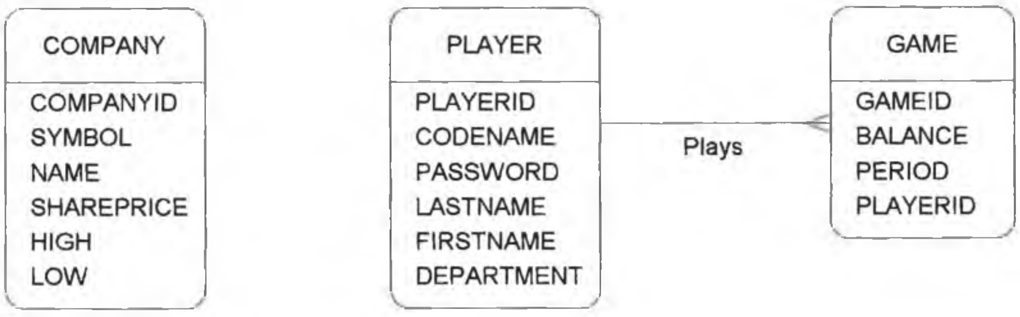


Figure 17: Entity Model

The database for the application is quite simple. As the purpose of the application is to teach students about EJB components and related patterns, a single standalone table such as COMPANY is ideal for introducing session beans that read database tables and for introducing patterns such as Data Access Object. It is also ideal for introducing Container Managed Persistence (CMP) entity beans. The simple one-to-many relationship between PLAYER and GAME is ideal for introducing Container Managed Relationships.

For this application numeric primary keys are used for each of the tables. These keys have no meaning outside the context of the database, i.e. the keys do not represent real world values such as a student's identification number or a company's symbol. This decision has implications for the strategy developed later in the project for generating primary keys.

3.4 Summary

As a tool for teaching J2EE technologies and patterns, the requirements are deliberately quite simple. They do however require the use of all the EJB components that a student will need to learn. They also require the use of quite a few general purpose patterns as well as J2EE patterns.

Chapter 4 develops some of the required EJB components and related patterns. As the components are developed a number of exercises for students are identified. These exercises are highlighted in the text in blue. At various points in the development, design flaws arise. Students are asked to identify these flaws and to attempt to provide solutions to eliminate the flaws. Only when the students have attempted to resolve the design flaws will appropriate patterns be introduced. These points in the development are also highlighted in the text, this time in red.

Chapter 4: Teaching EJBs and Patterns

It should be remembered that the purpose of developing the application for this project is not to provide a fully working production application. The purpose of this application is to provide students with examples that will help them to understand the various J2EE technologies. As the various examples are explored, problems in the solutions will arise. In line with Eric Gamma's thoughts on teaching patterns [3], various GoF patterns and J2EE patterns will be applied to refactor the solution when the student recognises and understands the problems.

Before creating the application a number of EJB components are developed. These components have been carefully chosen to allow students to build up a variety of components starting with simple standalone components that are fully unit-tested using remote clients. Gradually more complicated components are introduced.

4.1 EJB components

In order to implement an EJB, it is necessary to create four components. These components are

(i) a component interface, i.e. a Java interface that acts as the client's view of the bean.

(ii) a home interface, i.e. a Java interface that provides the client with methods to create, locate and remove beans, i.e. lifecycle methods of the bean.

(iii) a bean class, i.e. the class that implements the business methods required by the client.

(iv) a deployment descriptor file, i.e. an XML file that describes the various components.

The component interface represents the client's view of the bean. The client may be a remote client i.e. residing in a different container, meaning that the bean must expose a remote interface, or the client may be a local client (i.e. the client is co-located with the bean in the same container) meaning that the bean must expose a local interface. In fact a bean may expose both a remote interface and a local interface, i.e. the bean may have remote and local clients.

4.1.1 Naming convention

In order to clearly distinguish between remote and local interfaces, the following naming convention is used throughout this project. In order to write an EJB to represent an account, the following names are used.

Account**Remote**	The remote interface (remote client's view).
Account**RemoteHome**	The remote home interface (remote client's lifecycle methods).
Account**Local**	The local interface (local client's view).
Account**LocalHome**	The local home interface (local client's lifecycle methods).
Account**Bean**	The business class (the class that performs the business methods required by the client).
Account**EJB**	The collective name of the above components and the name used by JNDI to identify the bean

As local interfaces were only introduced in EJB v2.0, all interfaces prior to EJB 2.0 were remote. By convention the word "remote" was not used in the interface names, i.e. the remote interface was referred to as Account and the home interface as AccountHome. As this could lead to confusion, the above naming convention is used throughout this project.

There are three types of Enterprise Java Beans.

(i) Session Beans

(ii) Entity Beans

(iii) Message Driven Beans

4.2 Session Beans

Session beans come in two flavours, stateless session beans and stateful session beans.

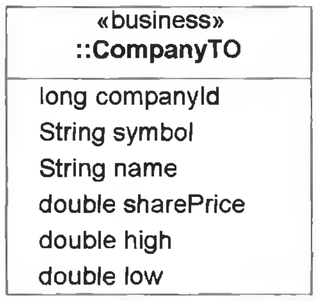
4.2.1 Stateless Session Beans

The simplest type of EJB is a stateless session bean. A stateless session bean does not maintain any client state. When a client calls a method on a bean the container provides a bean from a pool. When the method completes, the bean is returned to the pool. The client only has access to the bean for the duration of the method call. (Actually, a client never has direct access to a bean, as will be explained shortly). If the same client calls another method on a bean the container again provides a bean from the pool. This may be a different bean and so it is not possible to maintain client state between method calls.

A stateless session bean, CompanyListerEJB, was developed to illustrate the features of a stateless session bean (Appendix A). This bean is used to read data from a single table, Company, in the project's database. Normally entity beans are used for data access, however a light-weight session bean is ideal for providing read-only access to a database.

A simple utility class, CompanyTO (Transfer Object), was developed to transport the data from the database to the client. The real significance of using a class like CompanyTO for transporting data across a network will be explored later. The following diagram illustrates the attributes of CompanyTO and the fields of Company Entity object.

Set/get methods not shown.



Company entity diagram

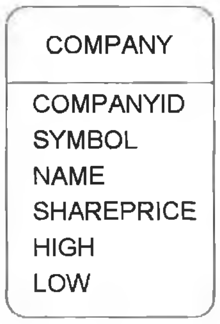


Figure 18: CompanyTO and Company Entity

The CompanyListerEJB was developed with remote and local interfaces to allow remote clients and local clients to invoke the services of the EJB. Examination of the various interfaces shows that the only difference between the code for remote and local interfaces is that remote methods must throw a java.rmi.RemoteException.

Recall that the home interfaces (remote and local) are for creating and locating EJBs, and component interfaces (remote and local) represent the client's view of the EJB. The following diagrams illustrate the inheritance hierarchy of the various components in CompanyListerEJB. Examination of the home interfaces reveals a single method, create(). The create() method of an EJB has a different meaning for different types of beans. A stateless session may have only a single create() method and this method has no arguments. With stateless session beans the create() method is used by the client to gain access to a bean.

The component interfaces (remote and local) expose the services of the EJB to the client, i.e. the methods that clients may call on the EJB. Examination of these interfaces reveals two methods, one to get a collection of companies and the other to get the share price of a company.

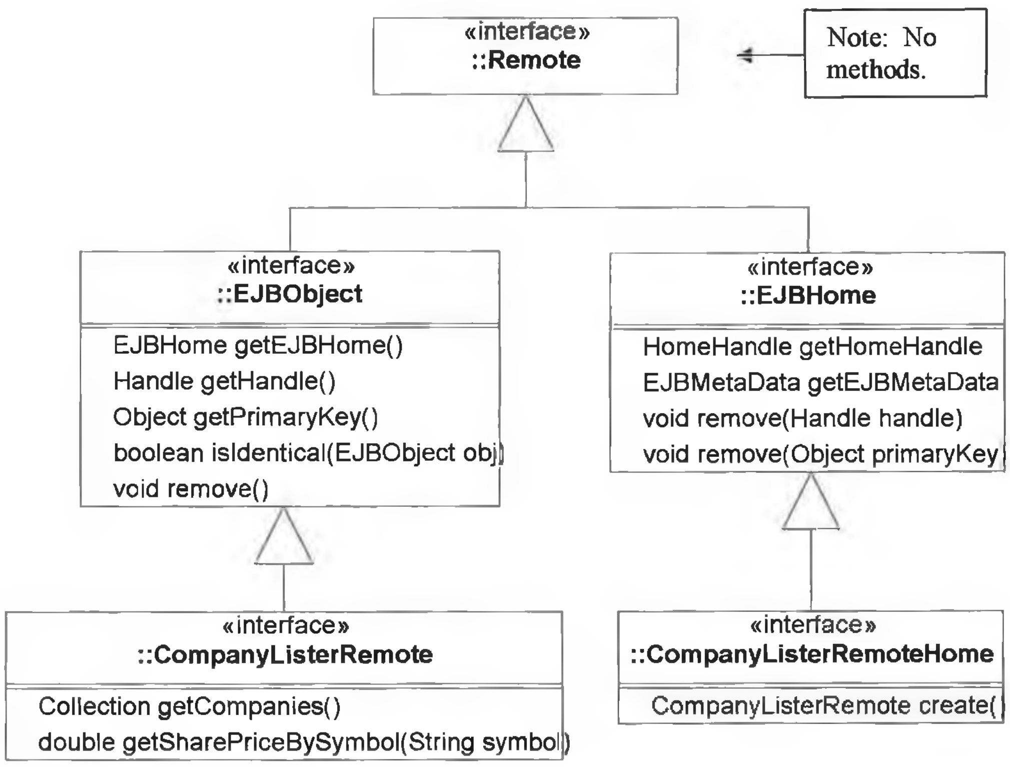

«interface»
::Remote
Note: No methods.
«interface»
::EJBObject
EJBHome getEJBHome()
Handle getHandle()
Object getPrimaryKey()
boolean isIdentical(EJBObject obj)
void remove()
«interface»
::EJBHome
HomeHandle getHomeHandle
EJBMetaData getEJBMetaData
void remove(Handle handle)
void remove(Object primaryKey
«interface»
::CompanyListerRemote
Collection getCompanies()
double getSharePriceBySymbol(String symbol)
«interface»
::CompanyListerRemoteHome
CompanyListerRemote create(

Figure 19: CompanyListerRemote and CompanyListerRemoteHome

CompanyListerRemote extends, but does not implement, EJBObject. As CompanyListerRemote is an interface, a class must be provided to implement this interface. This means that a class must be provided to implement all the methods in EJBObject and all the methods in CompanyListerRemote. There are no methods in the Remote interface.

The business methods in CompanyListerRemote represent the remote client's view of the bean, i.e. these are the business services provided for the client.

CompanyListerRemoteHome extends, but does not implement, EJBHome. As CompanyListerRemoteHome is an interface, a class must be provided to implement this interface. This means that a class must be provided to implement all the methods in EJBHome and all the methods in CompanyViewerRemoteHome.

A remote client must obtain a reference to the remote home interface using JNDI. This home interface is then used to obtain the remote interface. The client then uses this remote reference to call the business methods on the bean.

The bean provider must write the CompanyListerRemote interface and the CompanyListerRemoteHome interface.

The following class diagrams illustrate the hierarchy for the CompanyLister**Local** and the CompanyLister**Local**Home interfaces. It should be noted that these interfaces do not extend the Remote interface.

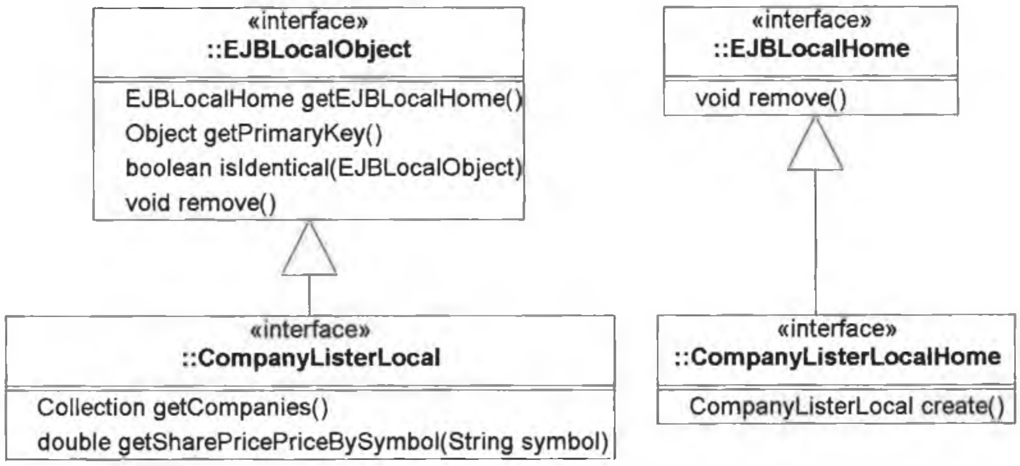


Figure 20: CompanyListerLocal and CompanyListerLocalHome

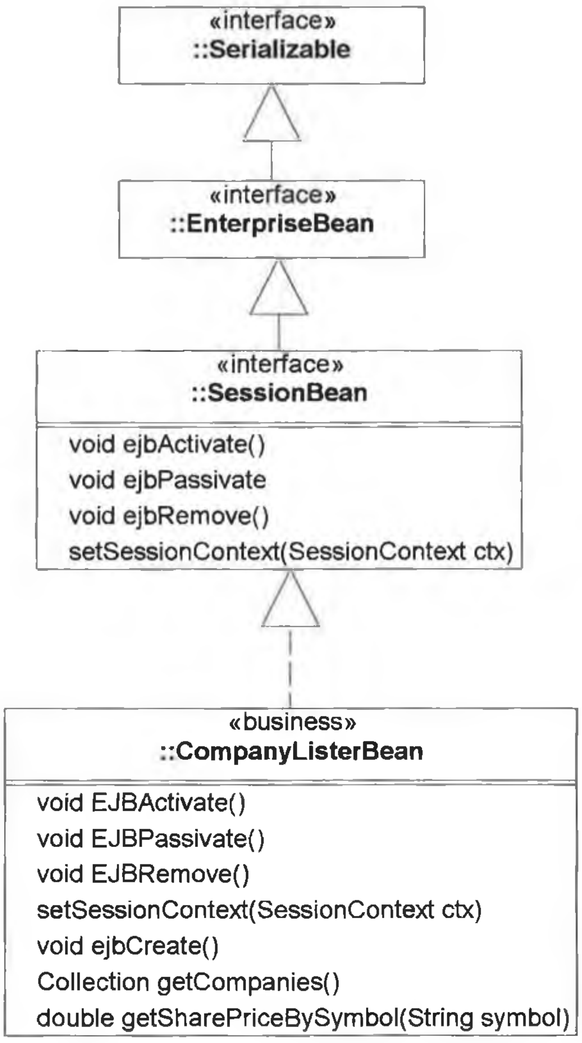
The bean provider must write the code for the CompanyListerLocal interface and for the CompanyListerLocalHome interface.

A client will normally call the following methods:

(i) the create() method on the local home or the remote home interface
(ii) the getCompanies() method on the remote or local interface.
(iii) the getSharePriceBySymbol() on the remote or local interface.

A client may also call the remove() method on the local home or remote home interface. This call simply invalidates the reference to the component and has no effect on the bean. Recall, the container maintains the beans in a pool and the client has no role to play in the creation or destruction of stateless session beans.

It is now necessary to provide an implementation of the business methods. This is done in the CompanyListerBean class. The following diagram illustrates the inheritance hierarchy of the CompanyListerBean class.



CompanyListerBean implements the SessionBean interface. This means that the Bean Provider must write methods to implement the 4 methods in SessionBean (there are no methods in the EnterpriseBean interface or the Serializable interface). These 4 methods are known as "callback methods". These methods are not called directly by the client, but are called by the container at certain points during the life cycle of the bean. These methods will be explained as they arise. For a stateless session bean, the methods EJBActivate() and EJBPassivate() are never called by the container. Empty implementations must however be provided as the bean must implement all the methods in the interface.

In addition to the callback methods, the Bean Provider must also implement an ejbCreate() method for each create() method in the home interface. Stateless session beans are permitted only one create() method and this must be a "no argument" method. This means that one "no argument" ejbCreate() method must be implemented in the bean.

The two business methods that were previously declared in the remote interface must also be implemented. These business methods provide the services for which the bean was created in the first place.

Figure 21: CompanyListerBean

It should be noted that there is nothing in the code to indicate that the session bean is stateless. The bean provider must implement all the methods in the CompanyListerBean class. Students should carefully study the implementation of the various methods in the bean class. It is a useful exercise to place println() statements in the various callback methods to see them being called by the container. ejbActivate() and ejbPassivate() are never called. The business methods use JNDI to look up the data source, use the data source to obtain a connection to the database and then use SQL to access the Company table. Students should have a good understanding of how and where the data source is defined (data-sources.xml). The following extract illustrates how the bean accesses the data source.

```
Context ctx = new InitialContext() ;
DataSource ds = (DataSource) ctx.lookup ("jdbc/sharesDS") ;
Connection con = ds.getConnection() ;
String query = "SELECT SharePrice FROM Company WHERE Symbol = ?" ;
stmt = con.prepareStatement(query) ;
stmt.setString(1, symbol) ;
rs = stmt.executeQuery() ;
```

As well as examining the code for the EJB, students should also pay particular attention to the contents of the ejb-jar.xml configuration file that describes the various EJBs. This file is available in Appendix D and will be referred to regularly. It should be noted that there is nothing in the Java code for the EJB to indicate that the bean is a *stateless* session bean. It is only in the deployment descriptor that the bean is defined as *stateless*.

As an exercise, students should implement additional business methods in the EJB.

4.2.2 Remote and Local Clients

A simple remote Java client, CompanyListerClient, is used to test the EJB. The following code extract illustrates how a remote client uses JNDI to gain access to the EJB.

```
Context context = new InitialContext() ;
CompanyListerRemoteHome
        companyListerRemoteHome = (CompanyListerRemoteHome)
                        PortableRemoteObject.narrow(context.lookup("CompanyListerEJB"),
                                                            CompanyListerRemoteHome.class) ;
CompanyListerRemote companyListerRemote = companyListerRemoteHome.create() ;
double sharePrice = companyListerRemote.getSharePriceBySymbol("Iona");
```

The steps are as follows:

1. Use JNDI to obtain a reference to the remote home interface.
2. Call the create() method on the remote home interface to obtain a reference to the remote component.
3. Use the remote interface to call the business methods.

Web components may be deployed in the same J2EE server as the EJB, meaning that they may act as local clients to the EJB, or web components may be deployed in a different server and act as remote clients. In order to illustrate local clients, a simple servlet, CompanyListerRemoteClientServlet.java, was developed to display the companies in a web page. A simple Java Server Page, CompanyListerRemoteClientJSP.jsp was also developed to display the same details. These examples use JNDI to perform a local lookup on the EJB to access its services, as shown.

```
Context context = new InitialContext() ;
CompanyListerLocalHome home = (CompanyListerLocalHome)
                context.lookup("java:comp/env/ejb/CompanyListerEJB") ;
CompanyListerLocal local = home.create() ;
companies = local.getCompanies() ;
```

These examples are shown in Appendix A. It is clear from these simple examples that

1. a Servlet is not suitable for rendering output to a web page.
2. A JSP is not suitable for implementing large amounts of Java scriptlet code.

Students should carefully study the web.xml file (Appendix C) for the Servlet entry. Particular attention should be paid to the <ejb-local-ref> entry.

4.2.3 The EJBObject

CompanyListerRemote and CompanyListerRemoteHome are interfaces and therefore a class must be developed that implements these interfaces. One might suspect that CompanyViewerBean is the class that implements these interfaces as it implements all the methods declared in the two interfaces. However, it is clear from the class diagram and the code that CompanyViewerBean implements the SessionBean interface. So who implements the remote interface and the remote home interface? The answer is that the container implements these interfaces. This means that the container creates a class to implement these two interfaces and instantiates an object of this class. This object is known as the EJBObject. When a client obtains a reference to a remote interface, it is not a reference to the actual bean, but rather a reference to this EJBObject. The client NEVER gains access to the actual bean. The client invokes methods on the EJBObject and the EJBObject in turn calls these methods on the actual bean.

4.2.4 The Deployment descriptor

All beans used in an application must be described in an ejb-jar.xml file (Appendix C) in the section for enterprise beans. Students should carefully study the entries in this file for each EJB. Most of the fields are self-explanatory. It is only in this file that a distinction is made between *stateless* and *stateful* session beans. The deployment descriptor may be written by hand. However, this is tedious and error-prone, and most J2EE IDEs generate this xml file from the EJB.

4.2.5 Stateful Session Beans

A *stateful* session bean maintains client state for the duration of the client's conversation with the server. The standard example used to illustrate a *stateful* session bean is a shopping basket. The basket remains alive throughout the client's shopping experience with the particular web site. A *stateful* session bean is not persistent, i.e. its data remains in memory and is not persisted to a secondary storage device. If an application requires the bean to be persisted (e.g. if a shopping basket is required to survive a server crash), then a *stateful* session bean would not be suitable.

To illustrate a stateful session bean, a very simple AccountEJB bean is developed. This EJB maintains the balance of a player's account during a single session of buying shares in a trading simulation game. This example assumes remote clients only. The bean, residing in the container in the J2EE server, maintains state between various method calls from the client. This example does not make use of the database.

The following class diagram illustrates the inheritance hierarchy of the AccountRemote interface and the AccountRemoteHome interface. The business methods available to remote clients are, as usual, the methods shown in the remote interface. In this case there are methods to get and set the balance and a method to make a purchase of a quantity of shares. There are two create() methods in the remote home interface. The fact that a session bean has more than one create() method, or has a create() method with arguments, means that the bean must be stateful.

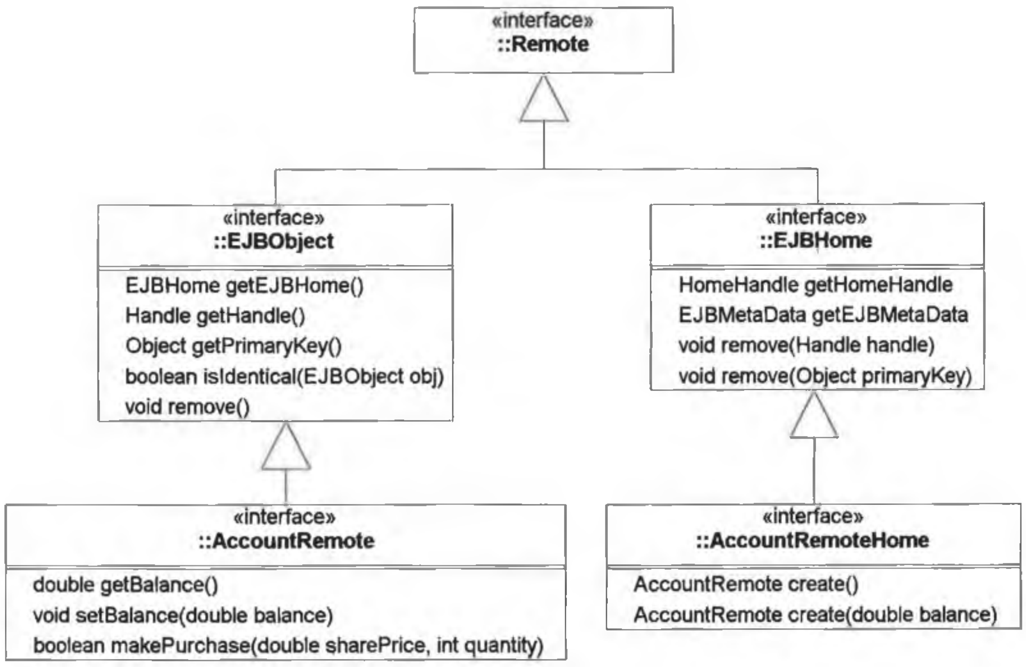


Figure 22: AccountRemote and AccountRemoteHome

The following diagram illustrates the inheritance hierarchy of the AccountBean class. The container may passivate (put to sleep) the bean during periods of inactivity to conserve resources. Prior to passivating the bean, the container calls the ejbPassivate() method to provide an opportunity for the bean provider to nullify any non-serializable object references (e.g. a datasource connection). Prior to re-activating the bean the container calls ejbActivate() to provide an opportunity to restore any object references nullified during passivation. The container will serialise and restore any member data that is serializable (e.g. balance) and so no action is needed by the bean provider in this case.

Unlike stateless session beans which are allowed only one create() method, a stateful session bean may have several create() methods and these methods may have arguments to initialise the attributes of the bean. There must be an ejbCreate() method to match each create() method in the home interface. When the client calls a create() method on a stateful session bean, the corresponding ejbCreate() method in the bean is called. When the client calls a remove() method, the bean is destroyed.

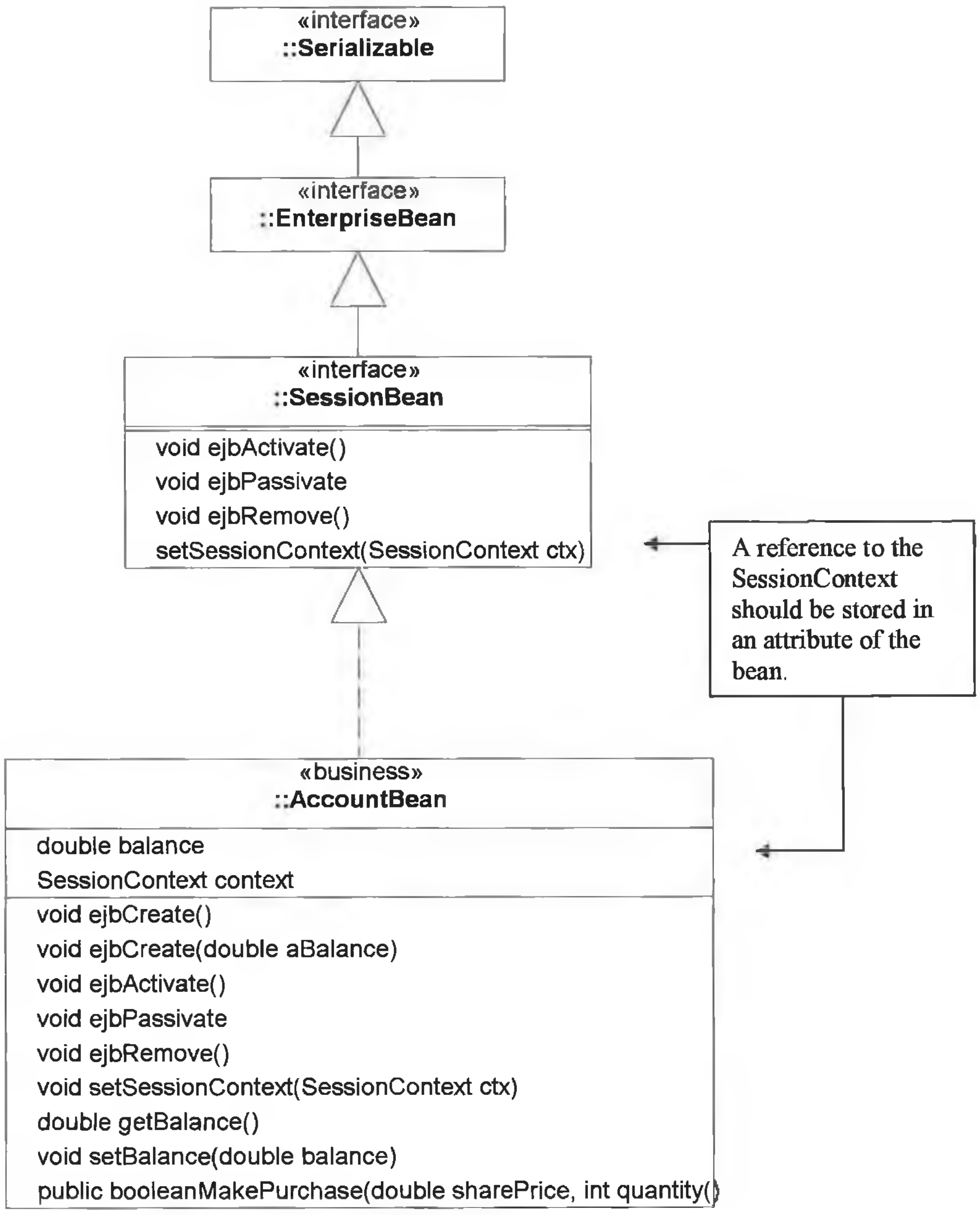

«interface»
::Serializable
«interface»
::EnterpriseBean
«interface»
::SessionBean
void ejbActivate()
void ejbPassivate
void ejbRemove()
setSessionContext(SessionContext ctx)
A reference to the SessionContext should be stored in an attribute of the bean.
«business»
::AccountBean
double balance
SessionContext context
void ejbCreate()
void ejbCreate(double aBalance)
void ejbActivate()
void ejbPassivate
void ejbRemove()
void setSessionContext(SessionContext ctx)
double getBalance()
void setBalance(double balance)
public booleanMakePurchase(double sharePrice, int quantity()

Figure 23: AccountBean

The ejbRemove() method is normally called by the container when the client invokes the remove() method on the remote interface. However, if the bean is asleep (i.e. if it has been passivated) when the remove() method is called, then the container does not call the ejbRemove() method. This means that if there is any clean-up code that needs to be called before the bean is destroyed, it should be placed in ejbPassivate() as well as in ejbRemove(). A simple remote Java client was created to test the EJB. The full listing of AccountRemoteClient is shown in Appendix A.

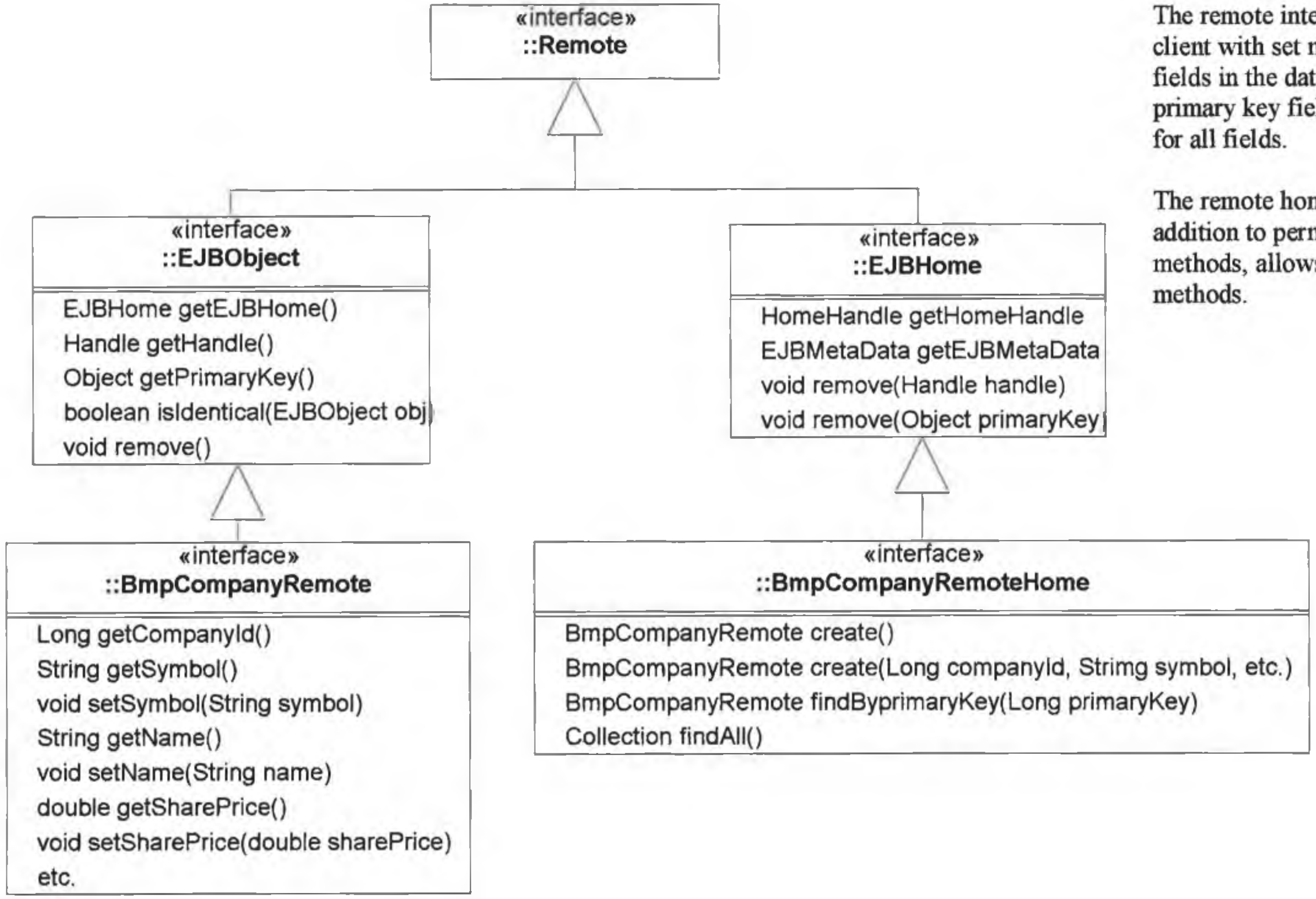
4.3 Entity Beans

There are 2 types of entity beans, Bean Managed Persistent (BMP) and Container Managed Persistent (CMP) beans. Unlike session beans, entity beans are persisted to secondary storage. In most cases the secondary storage is a relational database but it could be any data source. An entity bean may be thought of as an in-memory object that represents a row in a database table. It is the responsibility of the container to ensure that the state of this in-memory object is synchronized with the state of the row in the database table. This means that the container must provide some form of object-to-relational mapping. This mapping is vendor specific. Entity beans give the developer the benefit of using object-oriented techniques while working with relational data. The container handles concurrency issues when several clients attempt to access the same entity bean.

4.3.1 Bean Managed Persistence (BMP)

With BMP beans the persistence is managed by the bean itself, i.e. the bean provider must write all the data access code. This code may be written using JDBC. There are a number of callback methods that are called by the container to ensure that the bean and its corresponding database row remain consistent. The following example, BmpCompanyEJB, illustrates a BMP entity bean where the bean represents a row in the Company table used earlier. In addition to the usual components for an EJB, an entity bean also has a Primary Key class to represent the primary key in the database table. This key may be a field in the table or it may consist of composite fields from the table. For this example a column in the table, COMPANYID, is used as the primary key.

The home interfaces may have several create() methods. The create() methods of entity bean are completely different from the create() methods of session beans. Recall that stateless session beans are created and stored in a pool by the container and the client has no control over this creation process. The container provides a bean from the pool when the client invokes a method on the bean. When a client calls one of the create() methods of a stateful session bean, an actual EJB is created and its component interface is returned to the client. When a client calls one of the create() methods of an entity bean, a new row is inserted in the corresponding database table and the primary key of the entity is returned to the client.


«interface»
::Remote
«interface»
::EJBObject
EJBHome getEJBHome()
Handle getHandle()
Object getPrimaryKey()
boolean isIdentical(EJBObject obj)
void remove()
«interface»
::EJBHome
HomeHandle getHomeHandle
EJBMetaData getEJBMetaData
void remove(Handle handle)
void remove(Object primaryKey)
«interface»
::BmpCompanyRemote
Long getCompanyId()
String getSymbol()
void setSymbol(String symbol)
String getName()
void setName(String name)
double getSharePrice()
void setSharePrice(double sharePrice)
etc.
«interface»
::BmpCompanyRemoteHome
BmpCompanyRemote create()
BmpCompanyRemote create(Long companyId, Strimg symbol, etc.)
BmpCompanyRemote findByprimaryKey(Long primaryKey)
Collection findAll()

The remote interface provides the client with set methods for all the fields in the database table except the primary key field, and get methods for all fields.

The remote home interface, in addition to permitting create() methods, allows various finder methods.

Figure 24: **BmpCompanyRemote and BmpCompanyRemoteHome**

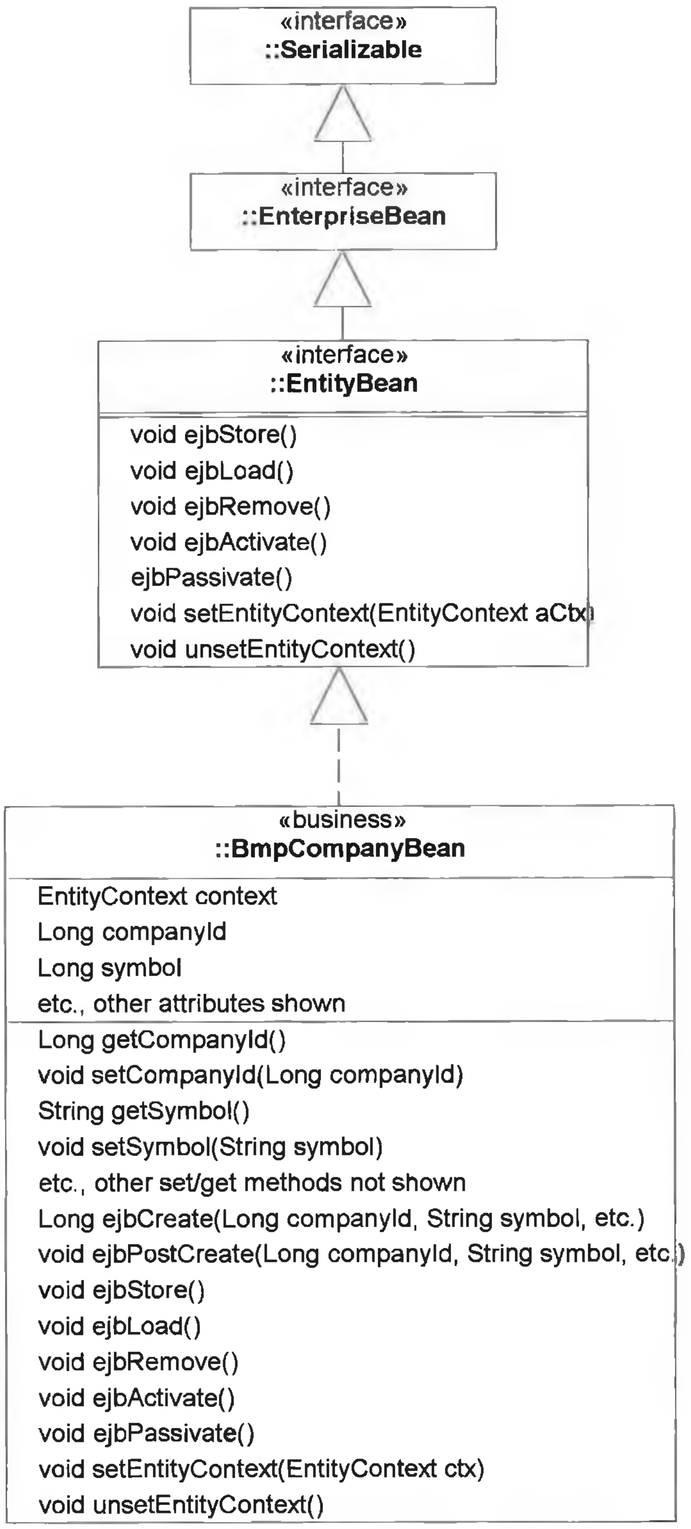

«interface»
::Serializable
«interface»
::EnterpriseBean
«interface»
::EntityBean
void ejbStore()
void ejbLoad()
void ejbRemove()
void ejbActivate()
ejbPassivate()
void setEntityContext(EntityContext aCtx)
void unsetEntityContext()
«business»
::BmpCompanyBean
EntityContext context
Long companyId
Long symbol
etc., other attributes shown
Long getCompanyId()
void setCompanyId(Long companyId)
String getSymbol()
void setSymbol(String symbol)
etc., other set/get methods not shown
Long ejbCreate(Long companyId, String symbol, etc.)
void ejbPostCreate(Long companyId, String symbol, etc.)
void ejbStore()
void ejbLoad()
void ejbRemove()
void ejbActivate()
void ejbPassivate()
void setEntityContext(EntityContext ctx)
void unsetEntityContext()

Figure 25: BmpCompanyBean

ejbCreate()	Each create method in the remote home interface must have a corresponding ejbCreate() method in the bean. When the client calls a create() method on the remote home interface, the corresponding ejbCreate() method is called and causes a new row to be added to the table.
ejbPostCreate()	Each create() method in the remote home interface must have a corresponding ejbPostCreate() method. This method is called immediately after the creation of the bean. This provides an opportunity to carry out additional initialization.

ejbPassivate()
ejbActivate()
	These methods are called by the container when the bean is passivated or activated. This provides an opportunity to perform proper clean up before a bean is passivated and an opportunity to restore values when the bean is activated.

ejbRemove()	Called when the client calls remove() on the remote interface. This method removes the row from the table.
ejbStore()	Called by the container to update the state of the row in the table when the state of the in-memory bean is changed.
ejbLoad()	Called by the container to update the state of the in-memory bean when the row in the table is changed.
set methods	These methods update the fields in the bean. The container then calls ejbStore() to update the row in the table..
get methods	These methods retrieve the values of the fields in the bean. These are the same values as the fields in the row in the table
Finder methods	These methods use SELECT statements to obtain details from the table. The methods may return a single entity or a collection of entities. The names are self-explanatory.
setEntityContext()	This method is called immediately after the creation of the bean. This method should always be used to store the value of the EntityContext as this is the only opportunity to gain access to the EntityContext.
Constructor	The constructor should always be left empty, or omitted altogether. All initialization should be carried out in the ejbCreate() or ejbPostCreate() methods.
getConnection()	This is a utility method used to obtain a connection to the database.

This EJB was developed with local and remote interfaces to allow for local and remote clients. Examination of the BmpCompanyBean class shows that the bean class has attributes that match the fields of the COMPANY table. The component interfaces (remote and local) have methods to set all the fields (with the exception of the primary key field) and get methods to access the fields. The home interfaces (remote and local) have methods to create() and find entity beans. The create methods have the effect of inserting rows in the company table.

The container calls the various callback methods (the names of these methods begin with "ejb") to synchronise the in-memory EJB with the corresponding row in the database. It is the responsibility of the bean provider to implement each of these methods and students should pay particular attention to these implementations. As an exercise, students should implement some of these methods themselves. Each of these methods uses JNDI to obtain a reference to the Datasource and then uses this Datasource to obtain a connection object. The methods then use JDBC to carry out the database operations. When the operation is completed the Connection object is released. The set methods are used to update the fields in the database and the get methods are used to retrieve the fields.

Particular attention should be paid to the setEntityContext() method and ejbLoad() method. Just as stateful session beans should store a reference to the SessionContext in an attribute, entity beans should always store the EntityContext reference in an attribute. The only opportunity to gain access to the EntityContext is in the setEntityContext() method. The EntityContext is the key that allows communication with the container, and it should always be stored as an attribute in the bean class. The ejbLoad() method illustrates a very important use of the EntityContext, i.e. to obtain the primary key. If the EntityContext is not saved then there is no way for this method to know which entity should be loaded.

Students should examine the code for these methods and should be encouraged to think about the efficiency of JNDI lookup for each call to getConnection(). They should consider possible improvements. The problem will be addressed when the Service Locator pattern is implemented later in the project.

In order to test the EJB a remote client, BmpCompanyRemoteClient, is used (Appendix A). As usual, the remote client obtains a reference to the remote home interface using a JNDI remote lookup. Unlike session beans, the create() method is not used to obtain a reference to the remote interface. The create() method for an entity bean has the effect of adding a new row to the database table. In order to obtain a remote reference to the EJB, the findByPrimaryKey() method is used. It is important that students note the return type of the findByPrimaryKey() method as shown in the home interface (the return type is BmpCompanyRemote, i.e. a reference to the remote interface) and the return type of the findByPrimaryKey() method as implemented in the BmpCompanyBean class (the return type is Long, i.e. the type of the primary key). The bean provider uses a SELECT statement to verify the existence of the row in the table and returns the primary key. The container uses this primary key to return a remote reference to the client. The client then uses this reference to call the business methods on the EJB. The return values of findAll() are used in the same way. The return value of findAll() in BmpCompanyBean is a Collection of primary keys. The container uses these primary keys to return remote references to the various companies. The client then uses these remote references to call the various methods.

4.3.2 Refactor with Patterns

As this dissertation is concerned with giving the student a valuable learning experience by recognizing poor designs and inviting improvements to the design, students are now invited to nominate the problems with the current design and attempt solutions at improving the design. The student should fairly quickly recognise two problems but may require more time to investigate a third problem.

The problems are as follows:

(i) The client supplies the primary key when adding a new company to the database. This is clearly unsatisfactory. The student should propose possible solutions to this problem. A possible solution is to provide a Primary Key Generator pattern.

(ii) BmpCompanyBean has lots of JDBC code, meaning that the bean provider is also responsible for all the data access code. It also means that any changes to the database require changes to the bean code. In fact, if the relational database were replaced with on object-oriented database, the bean code would have to be completely rewritten. Good practice dictates that the bean code and the data access code should be de-coupled. Again the student should recognise the problem and should again propose possible solutions. A possible solution to this problem is to introduce a Data Access Object pattern.

(iii) The less obvious problem concerns the use of remote method calls and the overhead this entails. Students should be encouraged to think about this and should try to understand what the problem is. Any student who can correctly identify the problem should be encouraged to devise a strategy to solve the problem. A solution is provided later in the project.

4.3.3 Primary Key Block Generator Pattern

J2EE applications that access relational databases must be able to generate primary keys. Different databases have various ways to generate primary keys ranging from the auto numbering feature of some databases to the sequence generator of Oracle. With some databases it is not possible to retrieve this key after it has been generated. It is essential in J2EE applications to be able to retrieve this key. Most Java based applications rely on a Java object to generate keys on request.

The PK Block Generator Pattern used in the example (Appendix A) is based on an implementation in *J2EE Design Patterns* by Crawford William, Kaplan Jonathan, (O'Reilly Press, 2003). [56] This class makes use of the sequence feature of Oracle. It works as follows. The first request for a primary key retrieves the initial sequence number 10010 from the database. The PK generator then issues keys based on this number as follows. The initial sequence number is multiplied by the block size (10 in this case), so the first 10 primary keys generated are:

100100, 100101, 100102, 100103, 100104, 100105, 100106, 100107, 100108, 100109

The block is now exhausted so the next number in the sequence, 10011, is retrieved from the database, multiplied by 10, and the next 10 primary keys generated are:

100110, 100111, 100112, 100113, 100114, 100115, 100116, 100117, 100118, 100119

If the server is stopped for any reason the block is lost and so, on restart, the block begins with the next sequence number. This means that there may be gaps in the primary keys. For this reason, as was mentioned earlier, each table in the database has a numeric primary key which has no meaning outside the context of the database. It is merely a number for retrieving rows from a table and has no real world meaning, such as a social security number. It also means that the same generator may be used to generate keys for the different tables, as gaps in the keys have no real significance. Any time a new row is added to any table, the method getNextPK() is called. The developer may now forget about primary keys and get on with the business of developing the application. The primary key class in this example is implemented as a GoF Singleton [5]. If a database is used that does not have a sequence feature then it is merely a case of modifying the Java class to use a different strategy for generating keys. No other part of the application is affected.

As an exercise, students should investigate alternate primary key generators.

4.3.4 Data Access Object Pattern

A Java class, CompanyDao (Appendix A) is developed to provide the services a client would expect when accessing the Company table. This class uses JNDI to locate the data source and provides finder methods findByPrimaryKey, findAll() and findBySymbol() as well as methods to add, amend and delete companies. As an exercise, students should add additional methods. Each method obtains a connection object to connect to the database and uses appropriate SQL statements to access the data in the tables. At the end of each method the connection is released to free resources. The add() method uses the primary key generator previously discussed to generate the primary keys.

It should be noted that the various methods throw DaoExceptions. The DaoException class is shown in Appendix A. This means that clients of CompanyDao are not aware that the data is coming from a relational database. The DAO provides a uniform interface to clients irrespective of the actual data source. Clients do not catch SQL exceptions but rather DaoExceptions. If for some reason the application were to take its data from a non-relational data source, e.g. an object-oriented database, it would be necessary to re-write the CompanyDao class, but the new class would still throw DaoExceptions, so no change would be required in client.

The CompanyDao class uses an object of the simple class CompanyTO (Appendix A) to transfer data to and from the database. This simple class, called a Transfer Object class, has much greater significance than one might expect. Students should be encouraged to think about the significance of this object. The significance will be explained later.

To improve efficiency, a variation on a GoF Factory [5] class (Appendix A) is provided to create Dao objects as required. As each new Dao is created it is cached and available for future use. A Hashset is used for caching the DAOs. As an exercise, students should write additional DAO classes, e.g. PlayerDao, GameDao. The GameDao is required later in the application.

To test the DAO class a new BMP EJB, BmpCompanyPatternsEJB, was developed as shown (Appendix A). As can be seen in the code for BmpCompanyPatternsBean, the methods are now much simpler and no reference is made to the data source and no SQL calls are made. The data access code has been de-coupled from the EJB.

4.3.5 Container Managed Persistence (CMP)

Having developed a BMP entity bean, the benefits of CMP beans will shortly become obvious to students. With BMP entity beans the bean provider has to write all the data access code. Sometimes BMP is used to provide greater flexibility over data access, but generally CMP is preferable and involves much less work for the bean provider. The following example, CompanyEJB, uses a CMP bean to provide the same functionality as the previous BMP example but with some additional finder methods. This EJB was developed with remote and local interfaces to allow for remote and local clients. The class diagram below illustrates the remote interface and the remote home interface of the CMP bean. Everything is exactly the same as in the diagram for the BMP bean, except of course, the name of the various components. The major difference arises in the implementation of the actual bean class (Appendix A).

Students should carefully examine the CompanyBean class and the following points should be noted.

1. The attributes (companyId, symbol, etc.) do not appear in the bean class. The attributes are implied from the set / get methods.
2. The set / get methods are all abstract so no implementations are provided.
3. No data access code appears in the class.
4. No finder implementations appear in the code.

The container provides the implementation of all these methods that were previously written by the bean provider, hence the name Container Managed Persistence.

It is very important that students become familiar with the entries in ejb-jar.xml for CompanyEJB as well as the entries in orion-ejb-jar.xml (a vendor specific file to provide mapping from the EJB implied attributes to the fields in the database). These files are shown in Appendix C.

As was mentioned earlier, it is necessary to map the abstract schema to the actual database table. How this mapping is performed is vendor specific. Most J2EE servers have a vendor specific file which holds these mappings. Most IDEs will create this file automatically but sometimes modifications may have to be made. With OC4J the name of this file is orion-ejb-jar.xml. Students should carefully study the mappings in this file. Without the use of an IDE, this file must be created manually.

The following diagrams illustrate the inheritance hierarchy of the CompanyRemote and the CompanyRemoteHome interfaces and of the CompanyBean class.

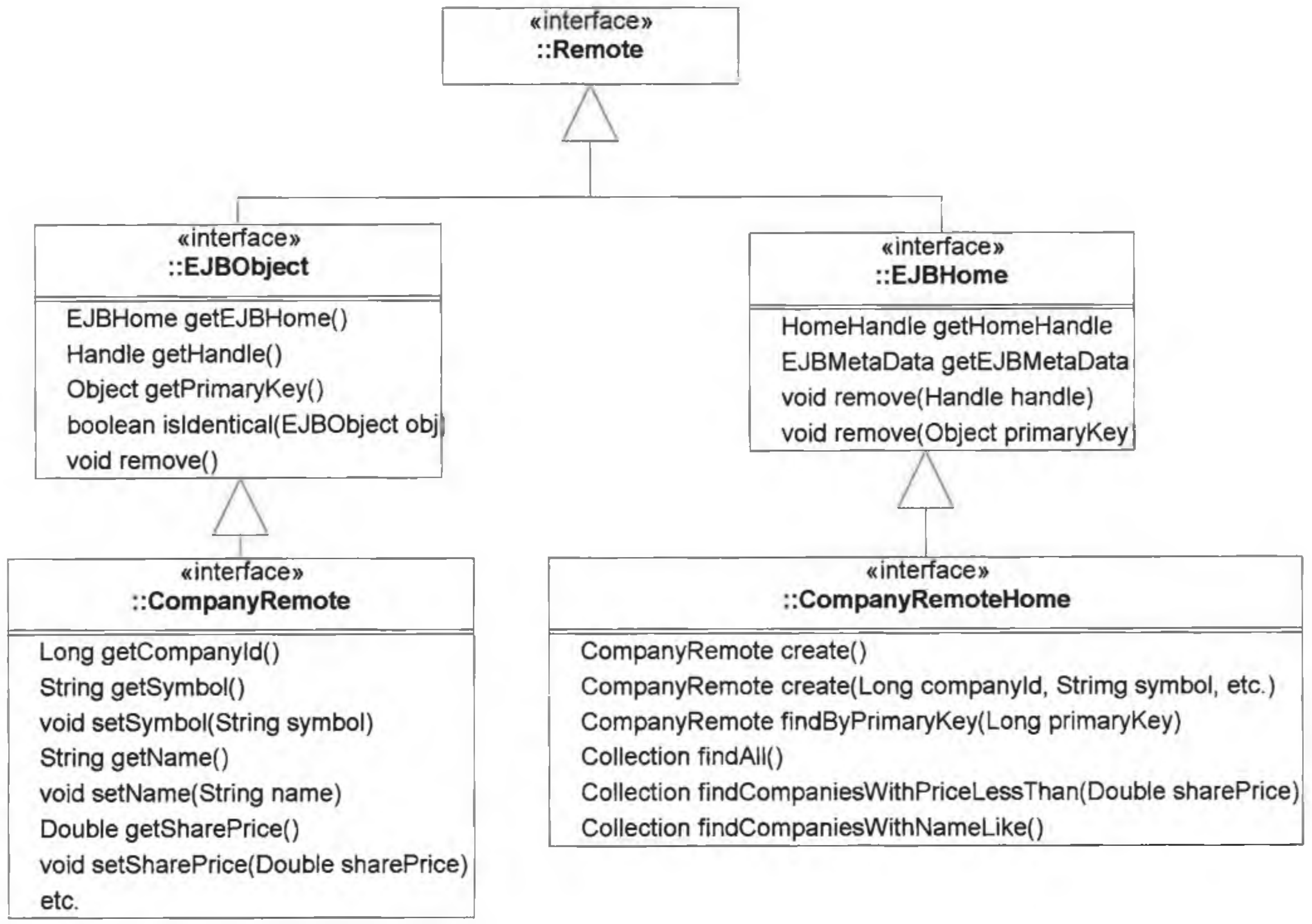

«interface»
::Remote
«interface»
::EJBObject
EJBHome getEJBHome()
Handle getHandle()
Object getPrimaryKey()
boolean isIdentical(EJBObject obj)
void remove()
«interface»
::EJBHome
HomeHandle getHomeHandle
EJBMetaData getEJBMetaData
void remove(Handle handle)
void remove(Object primaryKey)
«interface»
::CompanyRemote
Long getCompanyId()
String getSymbol()
void setSymbol(String symbol)
String getName()
void setName(String name)
Double getSharePrice()
void setSharePrice(Double sharePrice)
etc.
«interface»
::CompanyRemoteHome
CompanyRemote create()
CompanyRemote create(Long companyId, Strimg symbol, etc.)
CompanyRemote findByPrimaryKey(Long primaryKey)
Collection findAll()
Collection findCompaniesWithPriceLessThan(Double sharePrice)
Collection findCompaniesWithNameLike()

Figure 26: **CompanyRemote and CompanyRemoteHome**

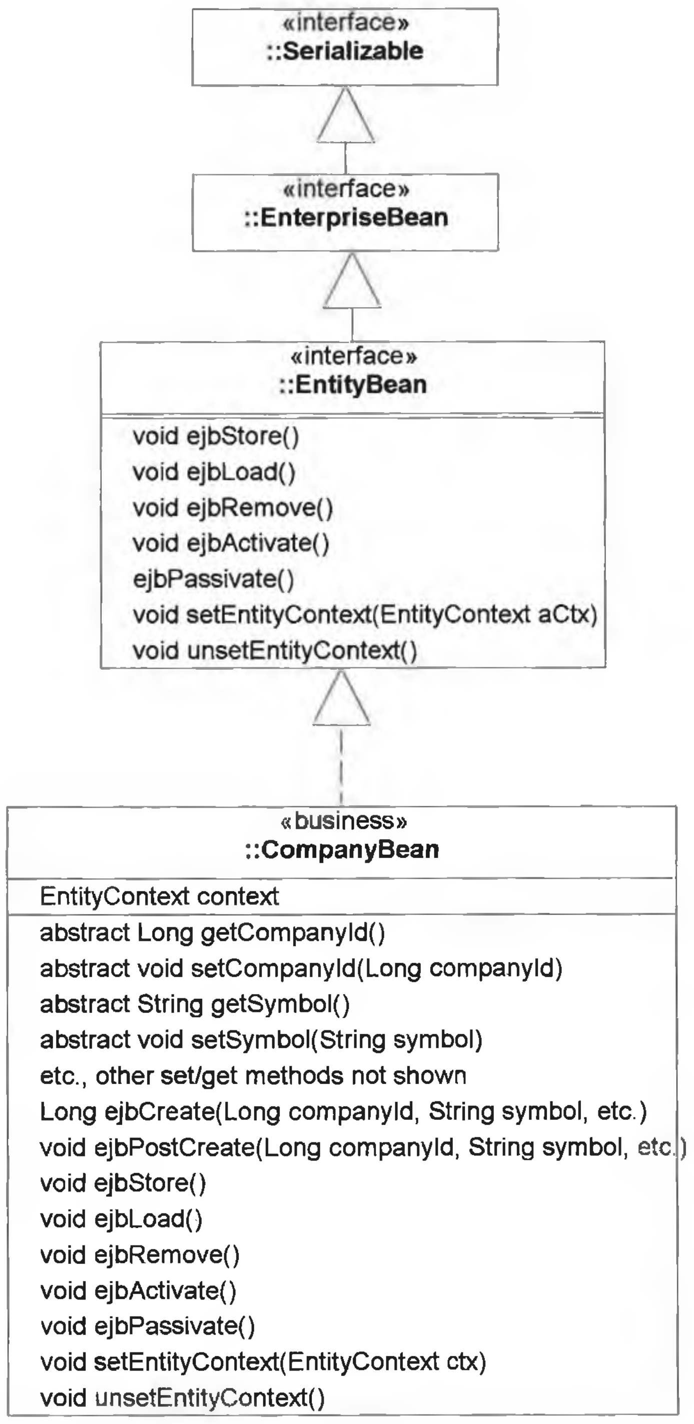

«interface»
::Serializable
«interface»
::EnterpriseBean
«interface»
::EntityBean
void ejbStore()
void ejbLoad()
void ejbRemove()
void ejbActivate()
ejbPassivate()
void setEntityContext(EntityContext aCtx)
void unsetEntityContext()
«business»
::CompanyBean
EntityContext context
abstract Long getCompanyId()
abstract void setCompanyId(Long companyId)
abstract String getSymbol()
abstract void setSymbol(String symbol)
etc., other set/get methods not shown
Long ejbCreate(Long companyId, String symbol, etc.)
void ejbPostCreate(Long companyId, String symbol, etc.)
void ejbStore()
void ejbLoad()
void ejbRemove()
void ejbActivate()
void ejbPassivate()
void setEntityContext(EntityContext ctx)
void unsetEntityContext()

Figure 27: CompanyBean

4.3.6 EJB Query Language (EJB QL)

EJB QL is a declarative query language similar to SQL but is designed to work with the abstract persistence schema of entity beans and provides a mechanism for persistence that is independent of the type of data source used, i.e. EJB QL works with relational and non-relational databases. The developer need not concern herself with the type of data source. In order to provide the container with the information it needs to manage the persistence of the bean, certain information must be included in the ejb-jar.xml file (Appendix C).

It is important that students study the EJB QL code added to the ejb-jar.xml file. Anyone familiar with SQL will recognise the similarity, e.g. some of statements used by the finder methods are shown below:

```
1.    findAll()
      select object(c) from CompanyEJB c
2.    FindCompaniesWithNameLike()
      select object(c) from CompanyEJB c where c.name like '%Bank%'
3.    FindCompaniesWithSharePriceLessThan(double price)
      select object(c) from CompanyEJB c where c.shareprice < ?1
```

The third select statement illustrates a paramaterised query

Note: The findByPrimaryKey() method does not appear in the ejb-jar.xml file as the container has sufficient information to carry out this operation.

The remote client used to test the CompanyEJB is CompanyRemoteClient and is shown in Appendix A.

As an exercise, students should provide additional finder methods.

4.3.7 Remote Method Calls

It is now time to address the problem mentioned earlier regarding the overhead of using remote calls. In particular, this problem will be examined in the context of calling the finder method, findAll(), that returns a collection of remote interface references. All the companies in the Company table are returned to the remote client by this call. This could involve hundreds of companies. For each company, the remote client calls each of the get methods (getSymbol(), getSharePrice(), etc.). The whole point of RMI and remote calls is to hide the complexity of accessing remote objects across a network from the client. However, it is important that students think about, and understand, the implications of these remote calls.

J2EE does a very good job of hiding the underlying complexity of remote calls. From the client's point of view, when the client calls a remote method it appears that the client is talking directly to the remote object. The following diagram illustrates what is actually going on in the background. The client calls the method getSymbol() on a "stub" object that looks like the remote object. However this method is now passed across the network where it is intercepted by the EJBObject. The EJBObject now calls the method on the actual bean. The reply to the client goes back via the EJBObject, across the network to the stub which passes the reply to the client.

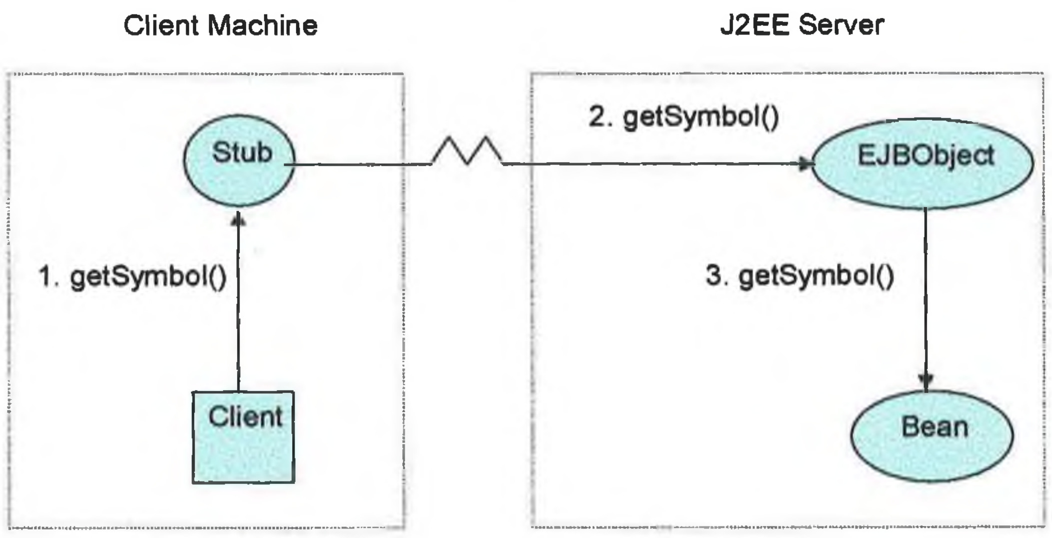


Figure 28: Remote Method Calls

This "fine-grained" access to the remote object is clearly very expensive. Each single call involves the above scenario. In the remote client application there are 5 remote calls for each company, and when multiplied by the number of companies, yields the total number of remote calls. So this simple remote client could be making hundreds of trips across the network to display the company details. Students were invited earlier to identify the problem and propose a solution. A strategy for solving this problem is to use a Transfer Object [2] (aka Data Transfer Object) pattern together with a Session Façade [2] pattern.

4.3.8 Transfer Object Pattern

The CompanyTO class has already been used to transfer data across the network in the CompanyListerEJB. The idea is to package the required data into a simple Java class and then to pass an object of this class, or a collection of objects, to the remote client. This provides the client with "coarse-grained" access to the required data. When the client receives the object, or collection of objects, it may then display the data, or whatever, without further trips across the network. It should of course be pointed out that the Transfer Object is not synchronised with the underlying data, so subsequent changes to the data on the server side are not reflected in the Transfer Object, and vice versa.

4.3.9 Session Façade Pattern

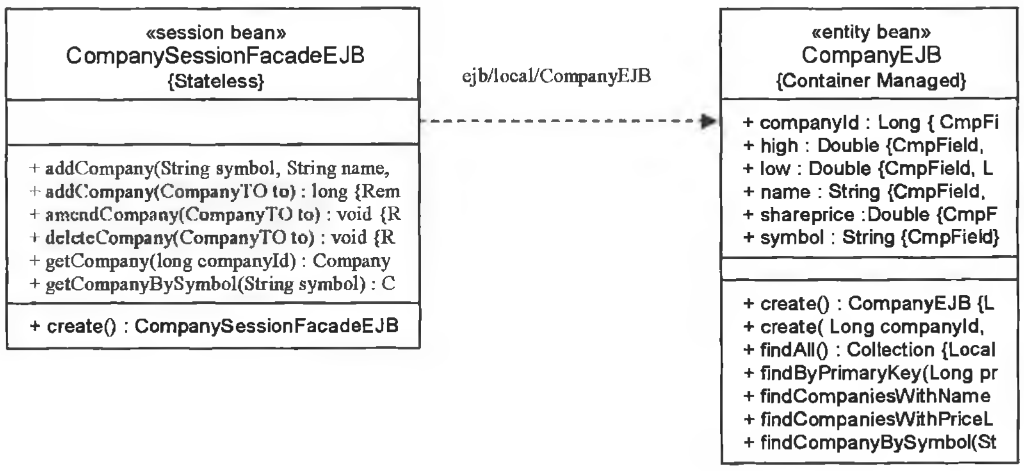

«session bean»
CompanySessionFacadeEJB
{Stateless}
+ addCompany(String symbol, String name,
+ addCompany(CompanyTO to) : long {Rem
+ amendCompany(CompanyTO to) : void {R
+ deleteCompany(CompanyTO to) : void {R
+ getCompany(long companyId) : Company
+ getCompanyBySymbol(String symbol) : C
+ create() : CompanySessionFacadeEJB
ejb/local/CompanyEJB
«entity bean»
CompanyEJB
{Container Managed}
+ companyId : Long { CmpFi
+ high : Double {CmpField,
+ low : Double {CmpField, L
+ name : String {CmpField,
+ shareprice :Double {CmpF
+ symbol : String {CmpField}
+ create() : CompanyEJB {L
+ create(Long companyId,
+ findAll() : Collection {Local
+ findByPrimaryKey(Long pr
+ findCompaniesWithName
+ findCompaniesWithPriceL
+ findCompanyBySymbol(St

Figure 29: CompanySessionFacadeEJB

It is now accepted practice that remote clients do not communicate directly with entity beans. Entity beans should be created with just a local interface and a local home interface. A stateless session bean with a local reference to the entity bean (this means that the session bean and the entity bean must reside in the same container) should be created to provide the required services to the client. The Session Façade [2] should provide a remote interface and a remote home interface to allow remote clients to communicate with it. The J2EE Session Façade pattern (Façade is of course a GoF pattern [5]) is used to provide the business services to implement a use case, or more likely, a number of related use cases for the client. When the remote client requires data from the database, the Session Façade, communicating locally with the entity bean, stores the data in a Transfer Object (or in a collection of Transfer Objects) and sends this data across the network to the remote client. Students should carefully study the code for the interfaces and the bean class of CompanySessionFacadeEJB (Appendix A). In particular they should pay attention to how the Session Façade uses JNDI to obtain a local reference to the entity bean. It should also be noted that the addCompany() method uses the primary key generator developed earlier.

Now a remote client simply uses JNDI to locate the remote home interface of the Session Façade, calls the create() method to obtain a remote reference to the Session Façade and then invokes the methods provided. The client has a much simpler interface and does not need any knowledge of how to invoke methods on the entity bean.

As an exercise, students should now provide a Session Façade together with a CompanyTO object to solve the problem of remote method calls when using the BMP entity bean that was developed earlier.

4.3.10 Service Locator Pattern

Each of the methods in CompanySessionFacadeBean (Appendix A) uses JNDI to look up the local home interface and then uses this interface to obtain a local reference to CompanyEJB. Using JNDI to continually look up home interfaces has a certain overhead. Students were invited earlier to identify this problem. Instead of looking up the home interface each time it is required, if the home interface is cached the first time it is obtained then further lookups are not be necessary. An implementation of Service Locator Pattern is given in ServiceLocator.java (Appendix A) for caching home interfaces. As an exercise students should now modify this class to cache data sources to solve the problem of multiple lookups identified earlier. A different HashMap may be used for different types of resources, e.g. data sources, queues, queue connection factories, etc. Now when any client requires a resource, instead of doing a JNDI lookup, a call is made on the Service Locator to obtain the resource. Service Locator is implemented as a GoF Singleton [5]. As a further exercise, students should now replace all earlier calls to JNDI for home interfaces and data sources with a call to the Service Locator.

4.4 Message-Driven Beans (MDBs)

Message-Driven Beans which were introduced in EJB v2.0 are stateless server-side components for processing asynchronous JMS messages. As a MDB does not have clients in the way that other EJBs do, there is no client view, so a MDB consists of a single bean class. To illustrate the use of MDBs a simple MDB called SimpleMessageDrivenEJB is developed. This MDB acts as a consumer of messages sent by some other application. The MDB illustrates how to initialize a queue to receive messages, how to read the messages and how to close all resources when finished. The example illustrates the MDB reading three types of message: a text message, a MapMessage and an ObjectMessage. The example simply prints the contents of the various messages. To illustrate the use of the MDB a simple Java application was created to act as a message producer. This application illustrate how to initialize the queue for sending messages, sends the three different types of messages and finally closes all resources. Students should carefully study the entries for the MDB in the ejb-jar.xml, orion-ejb-jar.xml files and also in the global file jms.xml (Appendix C).

4.4.1 Service Activator Pattern

MDBs as implemented in EJB v2.0 are examples of the Service Activator Pattern [2]. Prior to EJB 2.0, developers had to develop their own Service Activator Patterns by implementing JMS listeners.

4.5 Summary

This chapter provides extensive coverage of EJB components required by the application and reflects how these components would be developed in a class situation. It also indicates where students are expected to recognise the need for patterns, where they attempt solutions and where they are introduced to various patterns. Various exercises are also identified.

Chapter 5 develops the application that is used as a teaching tool using components and patterns already developed in this chapter together with additional components and patterns developed in Chapter 5. As in Chapter 4, the various exercises identified in Chapter 5 are again highlighted in blue. The points in the development where students are asked to identify design flaws are again highlighted in red.

Chapter 5: The Applications

The project uses two applications. The first application acts as message producer, sending messages which are consumed by the main application.

5.1 The Message Producer

The first application (Appendix B) is a Java application called SharePriceNews that acts as a producer for sending messages to an MDB in the main application. A utility class, MyRandomGenerator(), was developed to generate random numbers as required by the application. The Java application performs the following tasks:

1. It reads the companies from the database into an ArrayList using the stateless session bean, CompanyListerEJB, developed and tested earlier.
2. It randomly selects a number of these companies. The selected companies will have their share prices updated by the second application.
3. It randomly selects the number of times the selected companies will have their share prices updated.
4. It randomly selects the trend (i.e. up or down) for the share price for each of the selected companies.
5. It randomly selects the percentage change in the share price for each of the selected companies.
6. It stores the primary key and the percentage change in the share price for each of the selected companies in an ArrayList.
7. It places the ArrayList in a message queue waiting for a consumer to read the share price changes.

Steps 2 to 7 are repeated at regular time intervals, e.g. every 60 seconds.

5.2 The Main Application

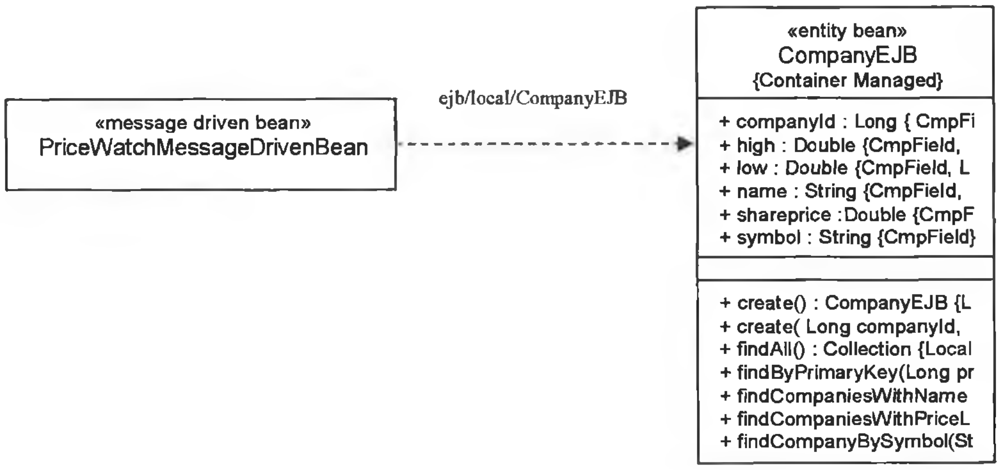
The main application, which reads and processes the messages from the first application, uses a MVC architecture and the model, view and controller are developed in this chapter.

5.3 The Model

The model is the main focus of the application as this is where the various EJB components are required. The components and patterns developed in chapter 4 are used here. Additional components and patterns are also developed.

5.3.1 The Message Consumer

The PriceWatchMessageDrivenEJB (Appendix B) acts as a message consumer and reads the messages sent to the message queue by the message producer Java application developed in chapter 5. This MDB maintains a local reference to the CMP entity bean, CompanyEJB, developed earlier as shown in the diagram (these beans reside in the same container). As the MDB reads each message consisting of a collection of share price changes for different companies, the MDB uses the Service Locator to obtain the home interface for each company in the collection. The findByPrimaryKey() method is called on the home interface to obtain a local interface to the appropriate CompanyEJB. The setSharePrice() method is called on the local interface to update the share price for the company. In this way the different randomly selected companies have their share prices updated.



«message driven bean»
PriceWatchMessageDrivenBean
ejb/local/CompanyEJB
«entity bean»
CompanyEJB
{Container Managed}
+ companyId : Long { CmpFi
+ high : Double {CmpField,
+ low : Double {CmpField, L
+ name : String {CmpField,
+ shareprice :Double {CmpF
+ symbol : String {CmpField}
+ create() : CompanyEJB {L
+ create(Long companyId,
+ findAll() : Collection {Local
+ findByPrimaryKey(Long pr
+ findCompaniesWithName
+ findCompaniesWithPriceL
+ findCompanyBySymbol(St

Figure 30: PriceWatchMessageDrivenEJB

5.3.2 Container Managed Relationships (CMRs)

In EJB 1.0 the only type of interfaces available were remote interfaces, meaning that the only clients allowed were remote clients. EJB 2.0 introduced the idea of local interfaces. The main driving force behind the introduction of local interfaces was to allow relationships between beans in the server. The types of relationships permitted are:

(i) one-to-one
(ii) one-to-many
(iii) many-to-many

These relationships may be implemented in one direction only (A knows about B, B does not know about A) or they may be implemented in both directions (A knows about B. B knows about A). The Session Facades already developed have a one-to-one relationship with the corresponding EJB. The Session Façade knows (i.e. has a local reference to) its corresponding EJB, but the EJB knows nothing about the Session Façade.

In order to illustrate relationships, the two related tables in the database, Player and Game, are used. The entity relationship diagram for these two tables is shown below.

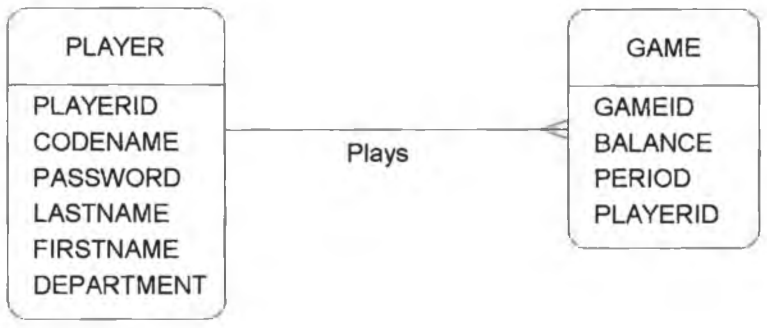


Figure 31: Player-Game Entities

For this example the player bean will know about all the games she has played and each game will know the player associated with it, i.e. the relationship will be implemented in both directions. The two EJBs representing these tables are PlayerEJB and GameEJB (Appendix B). Each of these EJBs has a local interface, a local home interface and a bean class. The local interfaces (i.e. the local client's view) of the 2 EJBs are shown below to illustrate how the relationship is implemented.

```
public interface PlayerLocal extends EJBLocalObject
{
  Long getPlayerid();
  String getCodename();
  void setCodename(String codename);
  String getPassword();
  void setPassword(String password);
  String getLastname();
  void setLastname(String lastname);
  String getFirstname();
  void setFirstname(String firstname);
  String getDepartment();
  void setDepartment(String department);

  Collection getGameEJB_playerid();
  void setGameEJB_playerid(Collection gameEJB_playerid);
}

public interface GameLocal extends EJBLocalObject
{
  Long getGameid();
  Double getBalance();
  void setBalance(Double balance);
  Timestamp getPeriod();
  void setPeriod(Timestamp period);

  PlayerLocal getPlayerEJB_playerid();
  void setPlayerEJB_playerid(PlayerLocal playerEJB_playerid);
}
```

These methods indicate the nature of the relationship

Unlike the relational database world, in the object-oriented world there is no concept of a foreign key. This means that the foreign key field, PlayerId in the Game table, will not feature in the corresponding EJB. In the OO world, a player maintains a collection of references to games and a game maintains a reference to a player. CMP entity beans do not have attributes corresponding to the fields in the database tables. Rather the set / get methods indicate the attributes. Examining the code above shows that a player has a collection of games and a game has a player.

A useful feature of JDveleoper is the ability to generate CMP beans based on tables in a database. This feature was used to generate PlayerEJB and GameEJB from the two tables, PLAYER and GAME. This has the effect of creating the PlayerEJB and the GameEJB CPM entity beans with the appropriate references in place. Students should closely examine the GameLocal (i.e. local interface) file. The get and set methods indicate the fields of the GAME table (i.e. gameid, etc.) There is no get / set method for the foreign key, PLAYERID. Instead there is a set / get method for the PlayerLocal interface, indicating the GameEJB has a local reference (i.e. knows about) the PlayerEJB. Also the PlayerLocal (i.e. local interface file) has set / get methods indicating the fields of the PLAYER table (i.e. Playerid, etc.) This file also has get / set methods for a Collection of local interfaces for the GameEJB. This is the many side of the relationship. This is an example of a Composite Entity [2]. Students should carefully examine the entries in the ejb-jar.xml for GameEJB and PlayerEJB (Appendix C). Considering GameEJB first, each of the persistent fields are shown as well as the primary key field. The EJB QL statement for the findAll() method is also shown. Looking now at the PlayerEJB entry, the various persistent fields of PLAYER as well as the primary key are shown. Again the EJB QL statement is automatically generated for the findAll() method. A number of additional finder methods required by the application have also been added. As an exercise, students should develop further finder methods. There is nothing new in entries for PlayerEJB and GameEJB. However, in the relationships section of the ejb-jar.xml, the one-to-many relationship is shown. As it makes sense to delete all the games belonging to player when the player is deleted, students should be made aware of the "cascade-delete" entry that was added to the "many" side in the relationship entry. Students should also closely examine the entries in orion-ejb-jar.xml for PlayerEJB and GameEJB.

As Player knows about Game (get/set methods) and Game knows about Player (get/set methods), it is clear that this is a bi-directional implementation of the relationship. If a unidirectional implementation is required then the get/set methods are omitted from the appropriate interface.

In order to allow remote clients to access these two EJBs, a new Session Façade, PlayerGameSessionFacade, was created as shown in the diagram. This Session Façade has a local reference to PlayerEJB and to GameEJB. As usual, the Service Locator is used to obtain a reference to the home interfaces. The Session Façade has various methods to add players and games. Instead of writing a Transfer Object for each of these entity beans, JDeveloper was used to automatically generate these objects (called Data Transfer Objects in JDeveloper). Students should closely examine these DTOs. As well as the ordinary attributes representing the fields in the tables, GameLocalDTO has a reference to a PlayerLocalDTO, and PlayerDTO has a collection of GameDTOs. This is a standard object-oriented implementation of a one-to-many bi-directional relationship. When passing DTOs across the network it is possible to pass a player together with the player's games, i.e. this is an example of a Composite Transfer Object [2].

Remote clients wishing to use the PlayerEJB or GameEJB do so by invoking methods on this session façade. The services available to remote clients are shown in the diagram and also in the code in Appendix B. This session façade pattern is fully tested as shown in Appendix B

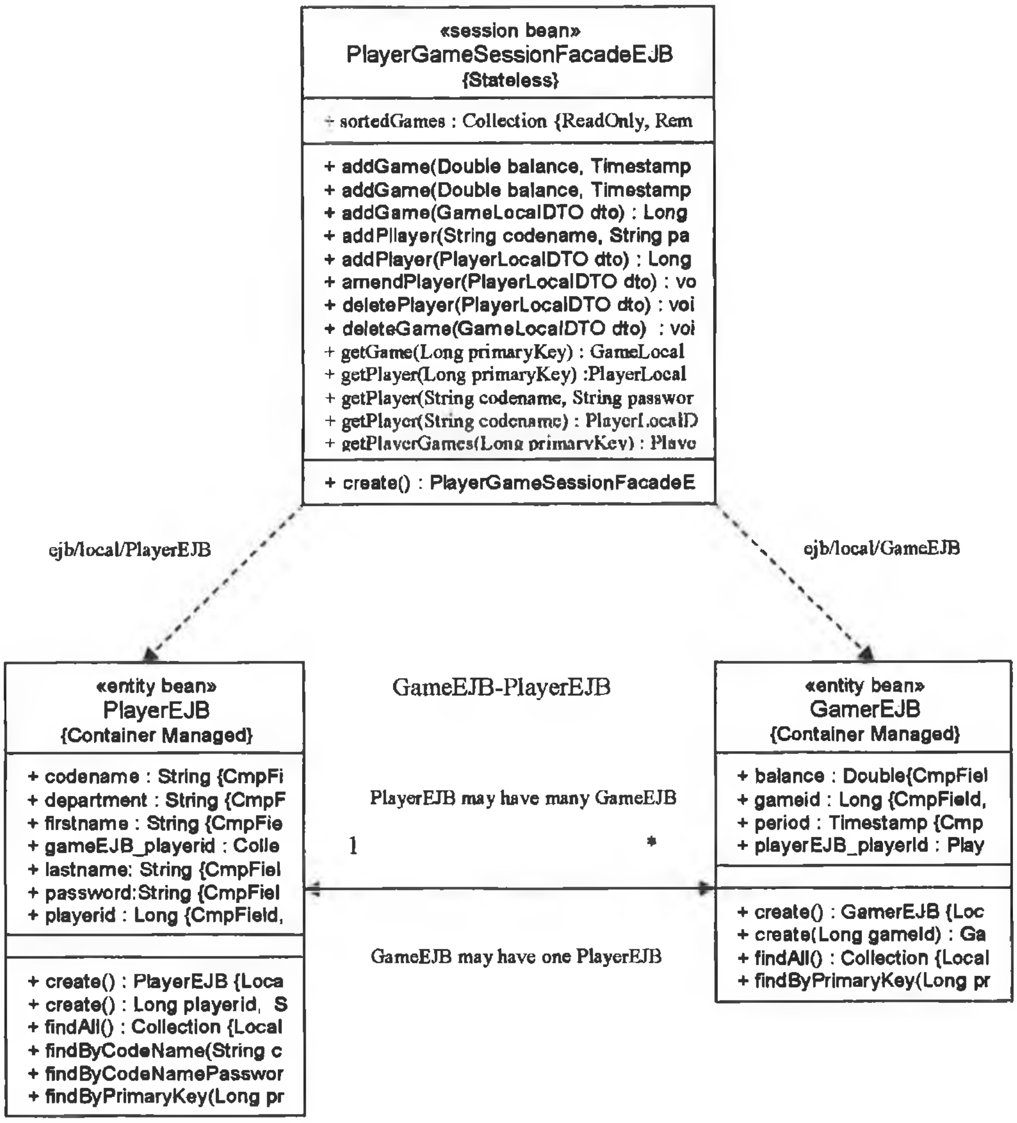

«session bean»
PlayerGameSessionFacadeEJB
{Stateless}
+ sortedGames : Collection {ReadOnly, Rem
+ addGame(Double balance, Timestamp
+ addGame(Double balance, Timestamp
+ addGame(GameLocalDTO dto) : Long
+ addPlayer(String codename, String pa
+ addPlayer(PlayerLocalDTO dto) : Long
+ amendPlayer(PlayerLocalDTO dto) : vo
+ deletePlayer(PlayerLocalDTO dto) : voi
+ deleteGame(GameLocalDTO dto) : voi
+ getGame(Long primaryKey) : GameLocal
+ getPlayer(Long primaryKey) :PlayerLocal
+ getPlayer(String codename, String passwor
+ getPlayer(String codename) : PlayerLocalD
+ getPlayerGames(Long primaryKey) : Playe
+ create() : PlayerGameSessionFacadeE
ejb/local/PlayerEJB
ejb/local/GameEJB
«entity bean»
PlayerEJB
{Container Managed}
+ codename : String {CmpFi
+ department : String {CmpF
+ firstname : String {CmpFie
+ gameEJB_playerid : Colle
+ lastname: String {CmpFiel
+ password:String {CmpFiel
+ playerid : Long {CmpField,
+ create() : PlayerEJB {Loca
+ create() : Long playerid, S
+ findAll() : Collection {Local
+ findByCodeName(String c
+ findByCodeNamePasswor
+ findByPrimaryKey(Long pr
GameEJB-PlayerEJB
PlayerEJB may have many GameEJB
1
*
GameEJB may have one PlayerEJB
«entity bean»
GamerEJB
{Container Managed}
+ balance : Double{CmpFiel
+ gameid : Long {CmpField,
+ period : Timestamp {Cmp
+ playerEJB_playerId : Play
+ create() : GamerEJB {Loc
+ create(Long gameId) : Ga
+ findAll() : Collection {Local
+ findByPrimaryKey(Long pr

Figure 32: PlayerGameSessionFacadeEJB

The various business methods that have been developed for the Session Façade are much more complicated as the relationship between player and game is now part of the equation. Students will need to carefully examine each of these methods to get a good understanding of how to handle the relationship. Each of the methods of PlayerGameSessionFacade has been fully tested in the remote Java client, PlayerGameSessionFacadeClient. As an exercise, students should add additional methods to PlayerGameSessionFacade.

5.3.3 The Portfolio as a Stateful Session Bean

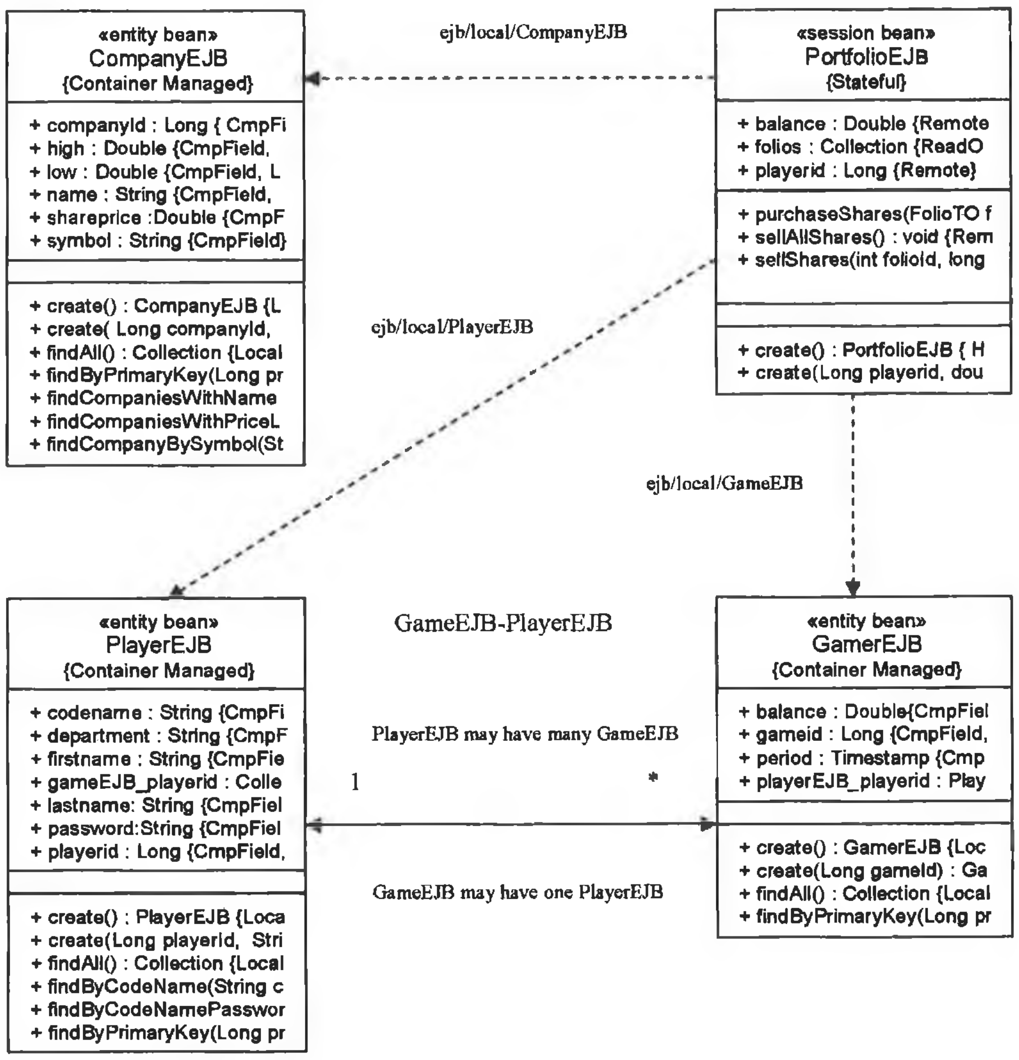

«entity bean»
CompanyEJB
{Container Managed}
+ companyId : Long { CmpFi
+ high : Double {CmpField,
+ low : Double {CmpField, L
+ name : String {CmpField,
+ shareprice :Double {CmpF
+ symbol : String {CmpField}
+ create() : CompanyEJB {L
+ create(Long companyId,
+ findAll() : Collection {Local
+ findByPrimaryKey(Long pr
+ findCompaniesWithName
+ findCompaniesWithPriceL
+ findCompanyBySymbol(St
ejb/local/CompanyEJB
«session bean»
PortfolioEJB
{Stateful}
+ balance : Double {Remote
+ folios : Collection {ReadO
+ playerid : Long {Remote}
+ purchaseShares(FolioTO f
+ sellAllShares() : void {Rem
+ sellShares(int folioId, long
+ create() : PortfolioEJB { H
+ create(Long playerid, dou
ejb/local/PlayerEJB
ejb/local/GameEJB
«entity bean»
PlayerEJB
{Container Managed}
+ codename : String {CmpFi
+ department : String {CmpF
+ firstname : String {CmpFie
+ gameEJB_playerid : Colle
+ lastname: String {CmpFiel
+ password:String {CmpFiel
+ playerid : Long {CmpField,
+ create() : PlayerEJB {Loca
+ create(Long playerId, Stri
+ findAll() : Collection {Local
+ findByCodeName(String c
+ findByCodeNamePasswor
+ findByPrimaryKey(Long pr
GameEJB-PlayerEJB
PlayerEJB may have many GameEJB
1
*
GameEJB may have one PlayerEJB
«entity bean»
GamerEJB
{Container Managed}
+ balance : Double{CmpFiel
+ gameid : Long {CmpField,
+ period : Timestamp {Cmp
+ playerEJB_playerid : Play
+ create() : GamerEJB {Loc
+ create(Long gameId) : Ga
+ findAll() : Collection {Local
+ findByPrimaryKey(Long pr

Figure 33: PortfolioEJB

As a player buys and sells shares, a record of the player's portfolio must be maintained. This information could be stored in the database using a CMP entity bean. For this application a stateful session bean, PortfolioEJB, is used. As an exercise, students should investigate other strategies for implementing the portfolio.

When a player logs into the system a stateful session bean, PortfolioEJB, is created as shown Appendix B. This bean maintains a collection of the various investments made by the player. The details of each investment are shown in the Transfer Object class FolioTO

(Appendix B). The FolioTO class holds the folioId, companyId, symbol, purchase price and the quantity of shares. At the end of the game, all the player's shares are sold at the current share price and the profit / loss is calculated. The details of the player's game (GAMEID, BALANCE, PERIOD, PLAYERID) are then written to the GAME table. The PortfolioEJB requires a local reference to CompanyEJB (to obtain current share prices), to PlayerEJB (to link the game to the player) and to GameEJB (to write to the Game table) as shown in the diagram.

5.4 The Controller

The Struts Framework could be used to implement the controller and to dispatch control to the various JSPs. However, as the intention is to teach students the value of patterns and how to use the patterns, the controller module is implemented using various patterns. Up to this point, the teaching method employed was to follow Gamma's advice [3] and let the student understand the problem with a particular solution before refactoring with appropriate patterns. However, any students studying EJBs should already be familiar with writing web applications using Servlets and JSPs. Certainly the target audience for this particular course will have this experience. These students should therefore be able to identify problems with using a Model 1 architecture so the approach to developing the controller module is to apply patterns from the start.

The strategy used to develop the controller is the Command and Controller Strategy [2] and the implemented version of this strategy employs the FrontController [2] pattern as well as a version of GoF's Command [5] and Factory [5] and Singleton [5] patterns. The FrontController is implemented as a Servlet as shown in Appendix B. All requests to the application are directed to this servlet. Students should carefully examine the web.xml file (Appendix C) and the mapping applied to this Servlet.

The FrontController reads the value of the "command" passed as a request attribute. It also detects the type of browser (wireless or ordinary) that issued the command. A simple naming strategy is used for directing control to the appropriate command handler to execute the command.

Command	CommandHandler
Login	LoginCommand
Register	RegisterCommand
etc	

The command (Login, Register, etc.) is passed to the FrontController as a request parameter, i.e. as a String. The FrontController passes this String to the CommandFactory. The CommandFactory instantiates the appropriate command handler object (e.g. LoginCommand, RegisterCommand, etc.). These command handler objects carry out the command by invoking appropriate methods on the PlayerGameSessionFacadeEJB. As the web container and the EJB container can reside in the same J2EE server, the command handlers could use local references to the session façade. However, as much of this project has involved the use of remote calls, remote references are used. This allows the flexibility of moving the web tier to a different server. A sample command handler is shown in Appendix B, together with the CommandFactory class

The command handler returns the name of the next page, e.g. Login.jsp, to the FrontController. As FrontController has already detected the type of browser, control is now passed to Login.jsp for ordinary browsers or to w_Login.jsp for wireless browsers. The only difference between handling a wireless device is in the code for the JSPs. This demonstrates the value of using the Model-View-Controller architecture.

As an exercise, students should develop additional use cases. This involves adding new methods to the Session Façade (or to a new Session Façade) if the methods do not already exist. For each use case a new Command Handler and one or more pages are also required. This has the benefit of demonstrating to students the ease with which additional use cases can be added to the application because of the architecture adopted.

Students should now identify a problem with the parameters passed to the various command handlers. As an exercise, students could research the Context Object [2] pattern and use it to encapsulate the protocol specific parameters (e.g. request parameters). The simplest implementation of this pattern involves the use of a HashMap to store the various parameters. Instead of passing HttpServletRequest objects to the various command handlers, this Context Object could be passed. The command handlers would then be protocol free and could be tested using simple Java clients.

As part of the controller module, a Synchronizer Token [2] has been used to prevent duplicate form submissions. The implementation of Token is shown in Appendix B. The use of this token can be seen in FrontController.java and in BuySharesCommand.java.

5.5 The View

As was mentioned at the outset, this project is focused on the J2EE components and patterns involved in the model and, and to a lesser extent, in the controller modules. Less time was devoted to developing the view. The view is used to illustrate the various components developed but it is not the intention to develop a full production application with a sophisticated user interface. Although pages were developed for all use cases, only a few of pages are shown in Appendix B. However, as students may wish to improve the user interface, it is important that pages are developed in a way that makes it easy to provide a uniform look and feel across all pages. Solution: Use a Composite View pattern [2]

5.5.1 Composite View pattern

A simple view was developed manually with each page consisting of:

1. A banner
2. A menu
3. The main page
4. A footer

As each new use case is developed, a new page (or pages) is developed. Each new page developed simply requires the development of the main page element. All pages automatically include the banner, menu and footer. Adding a new menu item is a simple matter of adding a new line to the header page.

An obvious problem arises in the view when a player chooses the option to display the league table, i.e. the details of all games played. As there is a large number of games, this view requires a lot of scrolling to view the entire table. Usually when a player views the league table (s)he will wish to view the top of the table only and not the entire list of games. The method getSortedGames() in PlayerGameSessionFacade uses the entity bean method, findAll(), to return a collection of games. Students should identify the problem with this approach and should identify a more efficient way of obtaining the data and returning it to the client in manageable amounts as required. A possible solution is to introduce the Value List Handler pattern.

5.5.2 Value List Handler pattern

Although this pattern is not part of the presentation tier, it is included at this point as the problem requiring solution manifests itself at this point. As with all the patterns used throughout the project, this pattern is only introduced when the students become aware of the problem.

The idea behind this pattern is to obtain the requested data from the database and to cache this data on the server side. The information is then sent to the client in manageable chunks. There are a number of strategies for developing the value list handler. The strategy used here involves using the Value List Handler Session Façade strategy. This involves using a stateful session bean as the Session Façade. Clearly the bean must be stateful if the data is to be cached. As this operation on the database involves read only access, the light-weight Data Access Object, GameDao, together with the light-weight transfer object, GameTO, are used. These classes were developed earlier in the project.

The EJB developed is called ValueListHandlerSessionFacadeEJB and is shown in Appendix B. This EJB has two create methods. The default create() method sets the number of games returned to the player at 20. The other create() method allows the player to specify the number of games returned. There are just three business methods available to the player, one to return the complete list, one to return the number of sublists available and one to return any sublist. The code for ValueListHandlerSessionFacadeEJB and the code to test it are shown in Appendix B.

5.6 The Wireless Client

A simple wireless client was developed to illustrate the benefit of the architecture used in the application. A sample JSP is shown in Appendix B. All requests, irrespective of their origin, are directed to the front controller. When the first request from a client reaches the front controller, the controller detects if the request comes from a wireless browser or an ordinary browser. This code is shown below.

```
Locale locale = request.getLocale() ;  ResourceBundle messages ;
String header = request.getHeader("User-Agent") ;
if (header.indexOf("Mozilla") == -1)
{
    session.setAttribute("device", "wireless") ;
    messages = ResourceBundle.getBundle("sk.sharesapp.resources.w_MyResources", locale) ;
}
else
{
    session.setAttribute("device", "pc") ;
    messages = ResourceBundle.getBundle("sk.sharesapp.resources.MyResources", locale) ;
}
```

Irrespective of the origin of the request, the controller hands control to the appropriate command handler to handle the request. The command handler returns the name of the next page to the front controller. The controller then directs control to the appropriate page, i.e. a JSP to handle the display in an ordinary browser or a JSP to handle the display in a wireless device. The code below illustrates this.

```
if (device.equals("wireless") )
    page = "w_" + page ;
```

For example, if control is directed to a page to show the list of companies, e.g. ViewCompanies.jsp, control will go to ViewCompanies.jsp for an ordinary browser or to w_ViewCompanies.jsp for a wireless device. Apart from the code in the JSPs, there is no difference between handling a request from an ordinary browser or a wireless browser.

The additional code above referring to a ResourceBundle is used as follows. A separate resource file for all supported languages must be provided (only the English language file has been provided for this application). The naming convention for these files is as follows: MyResources.java (default locale), MyResources_fr.java (France), MyResources_de.java (Germany), MyResources_zh.java (China), etc. These files contain all messages, prompts, etc. in the supported languages. The FrontController detects the current locale, and based on the locale selects the appropriate resource file. Examination of various JSP files, e.g. Header.jsp, illustrates the use of this feature (the feature has not been implemented in all pages).

In addition to the resource files for various languages, separate resource files are used for the wireless client, as clearly prompts, messages etc. must be kept short.

In order to add the wireless client to the application, Oracle's JDeveloper Wireless Extension, JWE, was added to JDeveloper. The OpenWave SDK 6.2.2 SDK [57] was used to provide a mobile emulator. The screens used for the wireless client are shown in Appendix D.

As an exercise, students should implement additional use cases for the wireless client.

5.7 Summary

In this chapter the applications that form the basis of the teaching tool were developed. As in the previous chapter, this chapter reflects how the various components and patterns would be developed in a class situation. It also indicates where students are expected to recognise the need for patterns and where they attempt solutions. Various exercises are also identified. These points are highlighted in the text.

The Java application, SharePriceNews, simulated share price changes and sent these to a message queue where they were read by the main application. The main application was developed using a MVC architecture. Most of the work was concentrated in the model where various EJB components and patterns already developed in Chapter 4 were integrated with additional components and patterns developed in this chapter.

Some of the benefits of the MVC architecture were illustrated, e.g.

(i) the ease with which new uses cases are added to the application.
(ii) the ease with which the the application runs in the language of the current locale.
(iii) the ease with which a wireless client is added.

The facility to run the application in the language of choice is particularly useful in the context of using this application as a teaching tool in DkIT. The target class for this application is made up of students from France, Germany, Spain and China as well as Ireland.

The view was developed to illustrate all the of use cases as set out in Chapter 3. The wireless client was added without making any changes to the EJB components, command handlers or JSPs already developed. All that was required were new JSPs to render the view in the wireless device.

Chapter 6 examines how the aims, as set out in Chapter 1, were achieved and suggests further areas of research.

Chapter: 6 Conclusions

The aims of the project were outlined in chapter 1 and this chapter examines how these aims were met and also looks at possible areas of research following from this dissertation.

6.1 Aims

(i) To gain a good working knowledge of the J2EE platform in general and EJB technology in particular.

In order to gain a good understanding of EJB, all the components used in the application were developed initially with a basic editor and the standalone OC4J server and a MySQL database. As well as writing the various components from first principles, this involved manually modifying various global configuration files (server.xml, application.xml, data-sources.xml, http-web-site.xml, jms.xml, jazn.xml). In addition, all entries in the project's ejb-jar.xml and orion-ejb-jar.xml were entered manually. This involved a lot of hard work and is not something that every student should have to experience (and, in hindsight, something the author would probably avoid if starting again). It was particularly difficult to obtain reliable information on developing CMP entity beans and CMRs (container managed relationships) for MySQL using OC4J. However, all problems encountered were eventually overcome. The knowledge gained proved invaluable when developing the application using JDeveloper. At times when errors were made in selecting options within the IDE, it was knowledge of the various configuration files that enabled the errors to be corrected. This knowledge should also prove invaluable when teaching J2EE technologies.

(ii) To gain a good understanding of best practice in applying J2EE technology for the development of enterprise systems through the use of patterns.

Initially the author read through the Core J2EE catalogue, as well as other catalogues, examining the various patterns. Some of the patterns used in implementing the controller were used in earlier work, with modifications and improvements for this application. However, the study of the other patterns only became focused when Gamma's advice [3] was followed, i.e. "feeling the pain" of a design before applying design patterns to improve the design. This technique should also prove invaluable in teaching patterns to students.

(iii) To ease the learning curve of students studying these technologies by using appropriate tools to ease the development and understanding of the technologies.

The students who are the target audience for the application developed for this dissertation are familiar with Oracle database. The choice of JDeveloper as a development tool for J2EE should certainly ease the learning curve for these students. As one would expect, integrating the various Oracle tools (Oracle 10g, JDeveloper, JWE) makes development much easier. A word of caution is necessary here. Students who use IDEs often fail to get a good understanding of the underlying technology. Using a good development tool is no substitute for understanding the technologies. JDeveloper allows developers to become more productive without hiding these underlying technologies and it also facilitates refactoring. Developers using JDeveloper still need a good understanding of the various components, the life-cycles of the different types of components and how the components interact. It is important when assessing students that they are examined on these details.

(iv) To share the knowledge and experience gained during the research for this thesis with students in order to enhance their knowledge and learning experience.

This was to be achieved through the development of a demonstration application for use as a learning tool.

Because of the timescale involved in developing this thesis, and because of the starting date of the courses in Enterprise Systems in DkIT, it was not possible to evaluate the application as a learning tool or to relate the experiences of students in using the application. The application as outlined in the aims was developed and tested and will shortly be used with students. Various EJB components were developed, beginning with simpler examples and progressing to more difficult components. Patterns were applied to improve the design as development progressed, but only when design problems were identified. A small wireless client was added to the application to demonstrate the benefit of the architecture used. As developing this application in the manner outlined was a useful exercise for the author learning about patterns, it should also prove useful in enhancing the learning experience of students.

6.2 Deployment

It was planned to deploy the completed application on the Oracle Application Server in Dundalk Institute of Technology. Unfortunately, when the Application Server (v. 10) was installed on the Unix server which already housed the Oracle database (v 9i), a conflict arose between the versions. As the Oracle database is required, at the time of going to press, to service existing courses in the institute, the application server was removed from the system. It is hoped that this problem will be resolved and that the application will be deployed to this server at a future date.

6.3 Areas of Further Research

There are several areas of further research that could be pursued by the author, or indeed by target students for this project should they decide to pursue postgraduate research.

6.3.1 The Application as a Learning Tool

As the application for this project was chosen and developed with the specific intention of teaching J2EE technologies and J2EE patterns, a logical follow-on from the project is to investigate the success of using the application as a learning tool. A class of students could be divided into two groups. One group could examine a particular pattern in isolation and could develop an implementation of this pattern. The second group could encounter the problem of a poor design in the application as recommended by Gamma [3]. Upon recognition of the problem, these students could be encouraged to solve the problem. This is similar to the techniques used in Problem Based Learning (PBL). Only when these students have devised their own solutions would they be introduced to the appropriate patterns. The roles of the two groups of students could then be reversed with a different pattern, or set of patterns. Measuring how the students coped with the different approaches could prove to be an interesting and useful research topic. As members of staff in another department in DkIT are already involved in research into PBL, collaboration could be initiated with these researchers.

6.3.2 OptimalJ and Archetype Patterns

The OptimalJ tool used in this project was obtained for a 30-day trial period. It is hoped to obtain this tool under the Compuware University licence agreement for use in Dundalk Institute of Technology. An investigation could be carried out into the productivity of students developing J2EE applications using this tool, together with a catalogue of Archetype patterns. In particular, the productivity of students with a good background in J2EE technologies against the productivity of students with good modeling skills, but only basic J2EE knowledge, could be examined.

6.3.3 Wireless Clients

The wireless client developed for this project was very basic, using WML as the markup language. The purpose of the wireless client was to illustrate one of the benefits of the architecture used in the development of the application. However, further work with Oracle's JWE, together with the OpenWave SDK, could be pursued.

6.4 Summary

This dissertation set out with clear aims and boundaries. These have been met. The subject of the dissertation, *The Application of Architectural and Design Patterns in Enterprise Systems*, has been well researched, the material has been presented in an instructional format with suggested exercises and a prototype teaching tool has been developed. At the time of going to press, the application has been introduced to the target class and some initial components have been developed using JDeveloper. Initial response from students has been very positive.

References

[1] "Implementing Sun Microsystems' Core J2EE Patterns".
John Crupi and Frank Baerveldt, April 2005
http://www.compuware.com/whitepapers/default.asp

[2] Core J2EE Patterns (Best Practices and Design Strategies) Second Edition.
Alur Deepak, Crupi John and Malks Dan.
Prentice Hall, 2003.

[3] How to Use Design Patterns
A Conversation with Eric Gamma, Part 1, by Bill Venners, May 2005
http://www.artima.com/lejava/articles/gammadp.html

[4] Patterns of Enterprise Application Architecture
Fowler Martin
Addison-Wesley, 2003

[5] Design Patterns (Elements of Reusable Object-Oriented Software)
Gamma Eric, Helm Richard, Johnson Ralph and Vlissides John
(These authors are usually referred to as the Gang of Four, GoF)
Addison-Wesley, 1995

[6] http://java.sun.com/j2ee/learning/tutorial/

[7] Sun Blueprints, Web-Tier Application Framework Design
http://java.sun.com/blueprints/guidelines/
designing_enterprise_applications_2e/web-tier/web-tier5.html

[8] http://java.sun.com/products/javabeans/docs/spec.html

[9] http://www.sun.com/service/about/success/ebay.xml

[10] An Interview with John Crupi, theServerSide.com, 2003
http://www.theserverside.com/talks/videos/JohnCrupi/interview.tss?bandwidth=56k

[11] http://java.sun.com/products/jdo/index.jsp

[12] http://www.hibernate.org/

[13] http://www.oracle.com/technology/products/ias/toplink/index.html

[14] The Rise of the POJO
Duncan Mills, Oracle Corporation, July 2005
http://www.oracle.com/technology/tech/java/newsletter/
articles/rise_of_the_pojo.html

[15] http://www.jboss.com/developers/index

[16] http://www.bea.com

[17] http://www.oracle.com/technology/tech/java/oc4j/index.html

[18] http://www.borland.com/us/products/jbuilder/index.html

[19] http://www.netbeans.org/

[20] http://www.oracle.com/technology/products/jdev/index.html

[21] http://www.mysql.com/

[22] http://www.compuware.com/products/optimalj/

[23] A Pattern Language: Towns/Buildings/Construction
Christopher Alexander
Oxford University Press, 1977

[25] Object-Modelling and Design
Rumbaugh James, Blaha Michael, Premerlani William, Eddy Frederick, Lorensen William
Prentice-Hall, 1991

[26] http://www.hillside.net/

[27] https://patternscentral.dev.java.net/

[28] http://patternshare.org/default.aspx/Home.GOF.HomePage

[29] http://www.sun.com/service/sunjavasystem/

[30] Core J2EE Patterns (Best Practices and Design Strategies).
Alur Deepak, Crupi John and Malks Dan.
Prentice Hall, 2001.

[31] Refactorings – Improving the Design of Existing Code
Martin Fowler
Addison-Wesley, 1999

[32] http://www.TheServerSide.com

[33] EJB Design Patterns
Floyd Marinescu,
Wiley, 2002

[34] A Cookbook for Using the Model-View- Controller User Interface Paradigm in Smalltalk-80.
G. E. Krasner and S. T. Pope.
JOOP, Vol 1, no 3, August/ September, 1988

[35] http://jakarta.apache.org/struts/index.html

[36] http://ibatis.apache.org/

[37] http://db.apache.org/ojb/

[38] http://www-306.ibm.com/software/data/db2/

[39] http://www.microsoft.com/sql/default.mspx

[40] http://www.bea.com/

[41] http://www.jboss.com/developers/index

[42] An Interview with Ted Anderson, Oracle Corporation, Sept. 2004
http://www.itwriting.com/jdev1.php

[43] JDeveloper Help Documentation

[44] www.junit.org

[45] ADF Business Components J2EE Design Pattern Catalogue
Steve Muench, ADF Development Team
June, 2005
http://www.oracle.com/technology/products/jdev/tips/muench/
designpatterns/index.html

[46] http://www.omg.org

[47] http://www.omg.org/mda

[48] Enterprise Patterns and MDA
(Building Better Software with Archetype Patterns and UML)
Arlow Jim and Neustadt Ila
Addison-Wesley, 2004

[49] The Decision is in: Agile versus Heavy Methodologies
Cutter vol. 2 No.19, Charette R. 2003

[50] Agile MDA
Stephen J. Mellor, Project Technology, Inc.
http://www.omg.org/mda/mda_files/Agile_MDA.pdf

[51] www.agilealliance.org

[52] http://www.macromedia.com/software/dreamweaver/

[53] Seve Vinosky, Chief Engineer of Product Innovation, IONA Technologies
Preface to [48]

[54] UML Modeling and MDA in Oracle JDeveloper 10g

[55] Higher Education Staff Development Network
An Initiative of the Department of Education & Science and the council of Directors the Institutes of Technology under the NDP 2000-2006.

[56] J2EE Design Patterns
Crawford William, Kaplan Jonathan
O'Reilly Press, 2003

[57] www.developer.openwave.com/dvl/

Appendix A: Chapter 4 Code

```
/* Author:   Seamus Kelly
 * Project:  M.Sc. in Computing
 * Date:     September 2005
 * This comment is omitted from remaining files
*/

package sk.sharesapp.ejb.session.stateless;
import            // imports not shown

public interface CompanyListerRemote extends EJBObject
{
  Collection getCompanies() throws SQLException,
                    RemoteException, NamingException;
  double getSharePriceBySymbol(String symbol) throws
                    SQLException, RemoteException, NamingException;

}

package sk.sharesapp.ejb.session.stateless;
import            // imports not shown

public interface CompanyListerRemoteHome extends EJBHome
{
  CompanyListerRemote create() throws RemoteException,
                                                 CreateException;
}

package sk.sharesapp.ejb.session.stateless;
import            // imports not shown

public interface CompanyListerLocal extends EJBLocalObject
{
  Collection getCompanies() throws SQLException, NamingException;
  double getSharePriceBySymbol(String symbol) throws SQLException,
                                  NamingException;
}
```

```
package sk.sharesapp.ejb.session.stateless;
import            // imports not shown

public interface CompanyListerLocalHome extends EJBLocalHome
{
  CompanyListerLocal create() throws CreateException;
}

package sk.sharesapp.ejb.session.stateless;
import            // imports not shown

public class CompanyListerBean implements SessionBean
{
  public void ejbCreate()  { }
  public void ejbActivate()  { }
  public void ejbPassivate()  { }
  public void ejbRemove()  { }
  public void setSessionContext(SessionContext ctx)  { }

  public Collection getCompanies() throws SQLException, NamingException
  {
   ArrayList companies = new ArrayList() ;
   Connection con = null ;
   PreparedStatement stmt = null ;
   ResultSet rs = null ;
   try
   {
     Context ctx = new InitialContext() ;
     DataSource ds = (DataSource) ctx.lookup ("jdbc/sharesDS") ;
     con = ds.getConnection() ;
     String query = "SELECT * FROM Company" ;
     stmt = con.prepareStatement(query) ;
     rs = stmt.executeQuery() ;

     while (rs.next() )
     {
       long companyId = rs.getLong("COMPANYID") ;
       String symbol = rs.getString("SYMBOL") ;
       String name = rs.getString("NAME") ;
       double sharePrice = rs.getDouble("SHAREPRICE") ;
       double high = rs.getDouble("HIGH") ;
```

```
        double low = rs.getDouble("LOW") ;
        CompanyTO to = new CompanyTO(companyId, symbol, name, sharePrice,
                                                                      high, low) ;
        companies.add(to) ;
      }
  }
  catch(NamingException e)
  {
     // Exception handling not shown
  }
  return companies ;
}

public double getSharePriceBySymbol(String symbol) throws SQLException,
                                                          NamingException
{
  Connection con = null ;
   PreparedStatement stmt = null ;
   double sharePrice = -1 ;
   ResultSet rs = null ;
   try
   {
     Context ctx = new InitialContext() ;
     DataSource ds = (DataSource) ctx.lookup ("jdbc/sharesDS") ;
     con = ds.getConnection() ;
     String query = "SELECT SharePrice FROM Company WHERE Symbol = ?" ;
     stmt = con.prepareStatement(query) ;
     stmt.setString(1, symbol) ;
     rs = stmt.executeQuery() ;
     if (rs.next() )
     {
         sharePrice = rs.getDouble("SharePrice") ;
     }
      else
        throw new SQLException("No such symbol " + symbol) ;
   }
   catch(NamingException e)  { /* Exception handling not shown */ }
   return sharePrice ;
  }
}
```

```
package sk.sharesapp.utils;
import java.io.Serializable;

public class CompanyTO implements Serializable
{
  private long companyId ;
     private String symbol ;
     private String name ;
     private double sharePrice ;
     private double high ;
     private double low ;
  public CompanyTO()
  {
   companyId = 0L ;
   symbol = "" ;
   name = "" ;
   sharePrice =  0.0 ;
    high =  0.0 ;
    low =  0.0 ;
  }
  public CompanyTO(long companyId, String symbol, String name, double sharePrice,
                                                    double high, double low)
  {
    this.companyId = companyId ;
    this.symbol = symbol ;
    this.name = name ;
    this.sharePrice = sharePrice ;
    this.high = high ;
    this.low = low ;
   }

   //  set / get methods not shown

}
```

```
package sk.sharesapp.ejb.session.stateless;
import              // imports not shown

public class CompanyListerRemoteClient
{
  public static void main(String [] args)
  {
    CompanyListerRemoteClient companyListerRemoteClient =
                                        new CompanyListerRemoteClient();
    try
    {
      Context context = getInitialContext();
      CompanyListerRemoteHome companyListerRemoteHome =
            (CompanyListerRemoteHome)PortableRemoteObject.narrow
                          (context.lookup("CompanyListerEJB"),
                                        CompanyListerRemoteHome.class);
      CompanyListerRemote companyListerRemote;

      companyListerRemote = companyListerRemoteHome.create();
      Collection companies = companyListerRemote.getCompanies( );
      System.out.println("Company details") ;
      Iterator it = companies.iterator() ;
      while (it.hasNext() )
      {
        CompanyTO to = (CompanyTO) it.next() ;
        System.out.println(to.getCompanyId() + "\t" + to.getSymbol() + "\t" +
                  to.getName() + "\t" + to.getSharePrice() + "\t" +
                  to.getHigh() + "\t" + to.getLow()) ;
      }
      String symbol = "Iona" ;
      double sharePrice = companyListerRemote.getSharePriceBySymbol(symbol);
      System.out.println("\nShare price of " + symbol + " is " + sharePrice) ;

    }
    catch( /* Exception handling not shown */)    { etc. }
  }
  private static Context getInitialContext() throws NamingException
  {
    return new InitialContext();
  }
}
```

```
package sk.sharesapp.servlet;
import              // imports not shown

public class CompanyListerLocalClientServlet extends HttpServlet
{
  private static final String CONTENT_TYPE = "text/html; charset=windows-1252";
  CompanyListerLocalHome home ;
  public void init(ServletConfig config) throws ServletException
  {
    super.init(config);
    try
    {
      Context context = new InitialContext() ;
      home = (CompanyListerLocalHome)
                   context.lookup("java:comp/env/ejb/CompanyListerEJB") ;
    }
    catch(NamingException e)
    {
      throw new ServletException("Error looking up home", e) ;
    }
  }

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
                         throws ServletException, IOException
  {
    CompanyListerLocal local = null ;
    Collection companies = null ;
    try
    {
      local = home.create() ;
      companies = local.getCompanies() ;
    }
     catch( /* Exception handling not shown */)    { etc. }
    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>CompanyListerLocalClientServlet</title></head>");
    out.println("<body>");
    out.println("<p>Company Details</p>");

    Iterator it = companies.iterator() ;
    while (it.hasNext() )
    {
      CompanyTO to = (CompanyTO) it.next() ;
      out.println(to.getSymbol() + "&nbsp"  + to.getName() + "&nbsp" +
            to.getSharePrice() + "&nbsp"  + to.getHigh() + "&nbsp" +
                                          to.getLow() + "<br>" ) ;
    }
    out.println("</body></html>");
    out.close();
  }
}

//  CompanyListerLocalClientJSP.jsp

<%@ page contentType="text/html;charset=windows-1252"%>
<%@ page import="javax.naming.*" %>
<%@ page import="java.sql.*" %>
<%@ page import="java.util.*" %>
<%@ page import="sk.sharesapp.ejb.session.stateless.*" %>
<%@ page import="sk.sharesapp.utils.*" %>

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
    <title>untitled</title>
  </head>
  <body>
  <form>
```

```
 <table cellspacing="3" cellpadding="2" border="1" width="80%">
   <%

     Context context = new InitialContext() ;
     CompanyListerLocalHome home = (CompanyListerLocalHome)
            context.lookup("java:comp/env/ejb/CompanyListerEJB") ;

     CompanyListerLocal local = null ;
     Collection companies = null ;

     local = home.create() ;
     companies = local.getCompanies() ;
     Iterator it = companies.iterator() ;
     while (it.hasNext() )
     {
       CompanyTO to = (CompanyTO) it.next() ;
   %>
     <tr>
       <td><%= to.getCompanyId() %>&nbsp;</td>
       <td><%= to.getSymbol() %>&nbsp;</td>
       <td><%= to.getName() %>&nbsp;</td>
       <td><%= to.getSharePrice() %>&nbsp;</td>
       <td><%= to.getHigh() %>&nbsp;</td>
       <td><%= to.getLow() %>&nbsp;</td>
     </tr>

   <%
     }
   %>

   </table>
  </form>
  </body>
</html>
```

```
package sk.sharesapp.ejb.session.stateful;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface AccountRemote extends EJBObject
{
 double getBalance() throws RemoteException;

 void setBalance(double balance) throws RemoteException;

 boolean makePurchase(double sharePrice, int quantity) throws RemoteException;
}

package sk.sharesapp.ejb.session.stateful;
import javax.ejb.EJBHome;
import java.rmi.RemoteException;
import javax.ejb.CreateException;

public interface AccountRemoteHome extends EJBHome
{
 AccountRemote create() throws RemoteException, CreateException;

 AccountRemote create(double balance) throws RemoteException, CreateException;
}

package sk.sharesapp.ejb.session.stateful;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class AccountBean implements SessionBean
{
 public double balance;
 private SessionContext context;

 public void ejbCreate()
 {
  System.out.println("Entered ejbCreate()") ;
  balance = 50000 ;
 }
public void ejbCreate(double balance)
{
 System.out.println("Entered ejbCreate(double balance)") ;
 this.balance = balance ;
}
public void ejbActivate()
{
 System.out.println("Entered ejbActivate()") ;
}
public void ejbPassivate()
{
 System.out.println("Entered ejbPassivate()") ;
}
public void ejbRemove()
{
 System.out.println("Entered ejbRemove()") ;
}
public void setSessionContext(SessionContext ctx)
{
 System.out.println("Entered setSessionContext()") ;
 this.context = ctx;
}
public double getBalance()  {
 return balance;
}
public void setBalance(double balance)  {
 this.balance = balance;
}
public boolean makePurchase(double sharePrice, int quantity)
{
 double newBalance = balance  - sharePrice * quantity ;
 if (newBalance < 0)
   return false ;
 else
 {
   balance = newBalance ;
   return true ;
 }
 }
}
```

```
package sk.sharesapp.ejb.session.stateful;
import            // imports not shown

public class AccountRemoteClient
{
 public static void main(String [] args)
 {
  AccountRemoteClient accountRemoteClient = new AccountRemoteClient();
  try
  {
   Context context = getInitialContext();
   AccountRemoteHome accountRemoteHome =
       (AccountRemoteHome)PortableRemoteObject.narrow
             (context.lookup("AccountEJB"), AccountRemoteHome.class);
   AccountRemote accountRemote;

   // accountRemote = accountRemoteHome.create();
   accountRemote = accountRemoteHome.create(10000);

   System.out.println("Initial balance = " + accountRemote.getBalance() );
   accountRemote.makePurchase(2.5, 1000 );
   System.out.println("Cost of purchase = " + 2.5 * 1000) ;
   System.out.println("New Balance = " + accountRemote.getBalance() ) ;

   accountRemote.makePurchase(10.0, 500 );
   System.out.println("Cost of purchase = " + 10 * 500) ;
   System.out.println("New Balance = " + accountRemote.getBalance() );
  }
  catch(Throwable ex)
  {
   ex.printStackTrace();
  }

 }

 private static Context getInitialContext() throws NamingException
 {
  return new InitialContext();
 }
}
```

```
package sk.sharesapp.ejb.entity.bmp;
import            // imports not shown

public interface BmpCompanyRemote extends EJBObject
{
 Long getCompanyId() throws RemoteException;
 String getSymbol() throws RemoteException;
 void setSymbol(String symbol) throws RemoteException;
 String getName() throws RemoteException;
 void setName(String name) throws RemoteException;
 double getSharePrice() throws RemoteException;
 void setSharePrice(double sharePrice) throws RemoteException;
 double getHigh() throws RemoteException;
 void setHigh(double high) throws RemoteException;
 double getLow() throws RemoteException;
 void setLow(double low) throws RemoteException;
}

package sk.sharesapp.ejb.entity.bmp;
import            // imports not shown

public interface BmpCompanyRemoteHome extends EJBHome
{
 BmpCompanyRemote create() throws RemoteException, CreateException;

 BmpCompanyRemote create(Long companyId, String symbol, String name,
                    double sharePrice, double high, double low)
                         throws RemoteException, CreateException;

 BmpCompanyRemote findByPrimaryKey(Long primaryKey)
                         throws RemoteException, FinderException;
 Collection findAll() throws RemoteException, FinderException;
}
```

```
package sk.sharesapp.ejb.entity.bmp;
import javax.ejb.EJBLocalObject;

public interface BmpCompanyLocal extends EJBLocalObject
{
  Long getCompanyId();
  String getSymbol();
  void setSymbol(String symbol);
  String getName();
  void setName(String name);
  double getSharePrice();
  void setSharePrice(double sharePrice);
  double getHigh();
  void setHigh(double high);
  double getLow();
  void setLow(double low);
}

/*  Author:   Seamus Kelly
 *  Project:  M.Sc. in Computing
 *  Date:     September 2005
*/

package sk.sharesapp.ejb.entity.bmp;
import              // imports not shown

public interface BmpCompanyLocalHome extends EJBLocalHome
{
  BmpCompanyLocal create() throws CreateException;
  BmpCompanyLocal findByPrimaryKey(Long primaryKey) throws FinderException;
  BmpCompanyLocal create(Long companyId, String symbol, String name,
                   double sharePrice, double high, double low) throws CreateException;
  Collection findAll() throws FinderException;
}
```

```
package sk.sharesapp.ejb.entity.bmp;
import              // imports not shown

public class BmpCompanyBean implements EntityBean
{
  public Long companyId;
  public String symbol;
  public String name;
  public double sharePrice;
  public double high;
  public double low;
  private EntityContext context;
  public Long ejbCreate() {    return null;  }
  public void ejbPostCreate()   { }
  public Long ejbCreate(Long companyId, String symbol, String name,
                   double sharePrice, double high, double low) throws CreateException
  {
    System.out.println("ejbCreate") ;
    Connection con = null ;
    PreparedStatement ps = null ;
    ResultSet rs = null ;
    try
    {
      con = getConnection() ;
      ps = con.prepareStatement("INSERT INTO Company VALUES(?, ?, ?, ?, ?, ?)") ;
      ps.setLong(1, companyId.longValue() ) ;
      ps.setString(2, symbol) ;
      ps.setString(3, name) ;
      ps.setDouble(4, sharePrice) ;
      ps.setDouble(5, high) ;
      ps.setDouble(6, low) ;
      int rows = ps.executeUpdate() ;
      if (rows != 1) throw new CreateException("Failed to create " + companyId) ;
      setCompanyId(companyId) ;
      setSymbol(symbol) ;
      setName(name) ;    setSharePrice(sharePrice) ;
      setHigh(high) ;        setLow(low) ;
    }
    catch(NamingException e)  { /* Exception handling not shown */     }
    return companyId ;
  }
```

```
public void ejbPostCreate(Long companyId, String symbol, String name,
                          double sharePrice, double high, double low)
{
 System.out.println("Entered ejbPostCreate()") ;
}
public Long ejbFindByPrimaryKey(Long primaryKey) throws FinderException
{
 Connection con = null ;
 PreparedStatement ps = null ;
 ResultSet rs = null ;
 try   {
  con = getConnection() ;
  ps = con.prepareStatement
      ("SELECT COMPANYID FROM COMPANY WHERE COMPANYID = ?") ;
  ps.setLong(1, primaryKey.longValue()) ;
  rs = ps.executeQuery() ;
  if (!rs.next() )
     throw new FinderException("Cannot find Company " + companyId) ;
 }
 catch ( /* Exception handling not shown */ ) { /* etc. */}
 return primaryKey;
}
public Collection ejbFindAll() throws FinderException
{
 Connection con = null ;
 PreparedStatement ps = null ;
 ResultSet rs = null ;
 ArrayList a = new ArrayList() ;
 try
 {
  con = getConnection() ;
  ps = con.prepareStatement("SELECT * FROM COMPANY") ;
  rs = ps.executeQuery() ;
  while (rs.next() )
  {
   a.add(new Long(rs.getLong("COMPANYID")) ) ;
  }
 }
 catch ( /* Exception handling not shown */ ) { /*  etc.  */}
 return a;
}
```

```
public void ejbActivate()  {  System.out.println("Entered ejbActivate()") ;  }
public void ejbPassivate()  { System.out.println("Entered ejbPassivate()") ;  }
public void ejbLoad()
{
 System.out.println("Entered ejbLoad()") ;
 Long primaryKey = (Long) context.getPrimaryKey() ;
 Connection con = null ;
 PreparedStatement ps = null ;
 ResultSet rs = null ;
 try
 {
   con = getConnection() ;
   ps = con.prepareStatement
             ("SELECT * FROM COMPANY WHERE COMPANYID = ?") ;
  ps.setLong(1, primaryKey.longValue() ) ;
  rs = ps.executeQuery() ;
  if (rs.next() )
  {
   companyId = primaryKey ;
   symbol = rs.getString("SYMBOL") ;
   name = rs.getString("NAME") ;
   sharePrice = rs.getDouble("SHAREPRICE") ;
   high = rs.getDouble("HIGH") ;
   low = rs.getDouble("LOW") ;
  }
  else
   throw new EJBException("Failed loading Company " + companyId) ;
 }
 catch ( /* Exception handling not shown */ ) { /*  etc.  */}
 finally
 {
  closeResources(con, ps, rs) ;
 }
}
```

```
public void ejbRemove() throws RemoveException
{
 System.out.println("Entered ejbRemove()") ;
 Connection con = null ;
 PreparedStatement ps = null ;
 ResultSet rs = null ;
 try
 {
  con = getConnection() ;
  ps = con.prepareStatement
                  ("DELETE FROM COMPANY WHERE COMPANYID = ?") ;
  ps.setLong(1, companyId.longValue() ) ;
  int rows = ps.executeUpdate() ;
  if (rows != 1)
    throw new SQLException("Failed to remove Company " + companyId) ;
 }
  catch ( /* Exception handling not shown */ ) { /* etc. */}
}
public void ejbStore()
{
  System.out.println("Entered ejbStore()") ;
  Connection con = null ;
  PreparedStatement ps = null ;
  ResultSet rs = null ;
  try
  {
   con = getConnection() ;
   String command =
     "UPDATE COMPANY SET SYMBOL = ?, NAME = ?, SHAREPRICE = ?, " +
                  "HIGH = ?, LOW = ? WHERE COMPANYID = ?" ;
   ps = con.prepareStatement(command) ;
   ps.setString(1, symbol) ;
   ps.setString(2, name) ;
   ps.setDouble(3, sharePrice) ;
   ps.setDouble(4, high) ;
   ps.setDouble(5, low) ;
   ps.setLong(6, companyId.longValue() ) ;
   int rows = ps.executeUpdate() ;
   if (rows != 1) {throw new SQLException("Store failed " + companyId) ;}
  } catch (/* Exception handling not shown */ ) { /* etc. */ }
 }
```

```
public void setEntityContext(EntityContext ctx)
{
 System.out.println("Entered setEntityContext()") ;
 this.context = ctx;
}

public void unsetEntityContext()
{
 System.out.println("Entered unsetEntityContext()") ;
 this.context = null;
}
private Connection getConnection() throws NamingException, SQLException
{
  Context context = new InitialContext() ;
  DataSource ds = (DataSource) context.lookup ("jdbc/sharesDS") ;
  Connection con = ds.getConnection();
  return con ;
}
private void closeResources(Connection con, PreparedStatement ps, ResultSet rs)
                     throws EJBException
{
 try
 {
  if (rs != null)    {    rs.close() ;    rs = null ;   }
  if (ps != null)    {    ps.close() ;    ps = null ;   }
  if (con != null)   {    con.close() ;   con = null ;  }
 }
 catch(SQLException e)
 {
  throw new EJBException(e.getMessage() ) ;
 }
}
public Long getCompanyId() {   return companyId;  }
public void setCompanyId(Long companyId)  {   this.companyId = companyId;
}
// Other set / get methods not shown
}
```

```
package sk.sharesapp.ejb.entity.bmp;
import              // imports not shown

public class BmpCompanyRemoteClient
{
 public static void main(String [] args)
 {
  BmpCompanyRemoteClient bmpCompanyRemoteClient =
  new BmpCompanyRemoteClient();
  try
  {
   Context context = getInitialContext();
   BmpCompanyRemoteHome bmpCompanyRemoteHome =
       (BmpCompanyRemoteHome)PortableRemoteObject.narrow
       (context.lookup("BmpCompanyEJB"), BmpCompanyRemoteHome.class);
   BmpCompanyRemote bmpCompanyRemote;
   System.out.println("Test findByPrimaryKey(10001)") ;
   bmpCompanyRemote =
        bmpCompanyRemoteHome.findByPrimaryKey(new Long(10001)) ;
   System.out.println(bmpCompanyRemote.getSymbol( ) + "\t" +
                 bmpCompanyRemote.getName() + "\t" +
                 bmpCompanyRemote.getSharePrice() + "\t" +
                 bmpCompanyRemote.getHigh() + "\t" +
                 bmpCompanyRemote.getLow()) ;
   System.out.println("Test findAll()") ;
   Collection col =  bmpCompanyRemoteHome.findAll() ;
   Iterator it = col.iterator() ;
   while (it.hasNext() )
   {
    BmpCompanyRemote to = (BmpCompanyRemote) it.next() ;
    System.out.println(to.getSymbol() + "\t" +
                 to.getName() + "\t" +
                 to.getSharePrice() + "\t" +
                 to.getHigh() + "\t" +
                 to.getLow()) ;
   }
Long primaryKey = new Long(20001) ;
System.out.println("Test Create(" + primaryKey + ")") ;
bmpCompanyRemoteHome.create(primaryKey, "XXX", "XXXX", 2.51, 2.61, 2.41);

System.out.println("Now test findByPrimaryKey(" + primaryKey + ")") ;
 bmpCompanyRemote = bmpCompanyRemoteHome.findByPrimaryKey(primaryKey) ;
 System.out.println(bmpCompanyRemote.getSymbol( ) + "\t" +
                 bmpCompanyRemote.getName() + "\t" +
                 bmpCompanyRemote.getSharePrice() + "\t" +
                 bmpCompanyRemote.getHigh() + "\t" +
                 bmpCompanyRemote.getLow()) ;
 System.out.println("Now test set methods)") ;
 bmpCompanyRemote.setSymbol("YYY") ;
 bmpCompanyRemote.setName("YYYY") ;
 bmpCompanyRemote.setSharePrice(3.51) ;
 bmpCompanyRemote.setHigh(3.61) ;
 bmpCompanyRemote.setLow(3.41) ;
 System.out.println("Now test findByPrimaryKey(" + primaryKey + ")") ;
 bmpCompanyRemote = bmpCompanyRemoteHome.findByPrimaryKey(primaryKey) ;
 System.out.println(bmpCompanyRemote.getSymbol( ) + "\t" +
                 bmpCompanyRemote.getName() + "\t" +
                 bmpCompanyRemote.getSharePrice() + "\t" +
                 bmpCompanyRemote.getHigh() + "\t" +
                 bmpCompanyRemote.getLow()) ;
 System.out.println("Now test Remove()") ;
 bmpCompanyRemote.remove();
 System.out.println("Now test findByPrimaryKey(" + primaryKey + ")") ;
 bmpCompanyRemote = bmpCompanyRemoteHome.findByPrimaryKey(primaryKey) ;
 }
 catch( /* Exception handling not shown */ )   {  /* etc. */ ) ;
 }
}
private static Context getInitialContext() throws NamingException
{
   return new InitialContext();
}
}
```

```
/*
    The following implementation of the PK Block Generator Pattern
    is a variation on the implementation provided in the following text:
    Title:       J2EE DEsign Patterns
    Author:      Crawford and Kaplan
    Publisher:   O'Reilly, 2003
    ISBN:        0-596-00427-3
    Page:        156
    This version is implemented as a Singleton
*/
package sk.sharesapp.dao ;
import           // imports not shown

public class PrimaryKeyGenerator
{
 private static PrimaryKeyGenerator keyGenerator = null ;
 DataSource ds = null ;
 private static int BLOCK_SIZE = 10 ;
 private static long current = -1 ;
 private static long getNextAt = -1 ;

 private PrimaryKeyGenerator() throws DaoException
 {
  try
  {
   Context context = new InitialContext() ;
   ds = (DataSource) context.lookup ("jdbc/sharesDS") ;
  }
  catch(NamingException e)
  {
   throw new DaoException(e.getMessage() ) ;
  }
 }
 public synchronized static PrimaryKeyGenerator getInstance() throws DaoException
 {
  if (keyGenerator == null)
  {
   keyGenerator = new PrimaryKeyGenerator() ;
  }
  return keyGenerator ;
 }
```

```
public synchronized long getNextPK() throws DaoException
{
 if (current > -1 && current < getNextAt)
  return current++ ;
   // Retrieve a new block from the database
  Connection con = null ;
  PreparedStatement ps = null ;
  ResultSet rs = null ;
  try
  {
   con = getConnection() ;   // DUAL is a system table in Oracle
   String query = "SELECT NEXTPK.NEXTVAL FROM DUAL" ;
   ps = con.prepareStatement(query) ;
   rs = ps.executeQuery() ;
   if (rs.next() )
   {
    long nextSeq = rs.getLong("NEXTVAL") ;
    current = nextSeq * BLOCK_SIZE ;
    getNextAt = current + BLOCK_SIZE ;
   }
  }
  catch( /* Exception handling not shown */ )  { /* etc. */ ) ;
  return current++ ;
}
private Connection getConnection() throws DaoException
{
 try
 {
  Connection con = ds.getConnection();

  return con ;
 }
 catch(SQLException e)
 {
  throw new DaoException(e.getMessage() ) ;
 }
}
```

```
 private void closeResources(Connection con, PreparedStatement ps, ResultSet rs)
                          throws DaoException
 {
  try
  {
    if (rs != null)      {  rs.close() ;      rs = null ;    }
    if (ps != null)      {  ps.close() ;      ps = null ;   }
    if (con != null)     {  con.close() ;   con = null ; }
  }
  catch(SQLException e)    {      throw new DaoException(e.getMessage() ) ;    }
 }

 public static void main(String[] args) throws DaoException
 {
    for (int i = 0 ; i < 25 ; i++)
    {
      PrimaryKeyGenerator keyGenerator = PrimaryKeyGenerator.getInstance() ;
      System.out.println("Next primary key: " + keyGenerator.getNextPK() ) ;
    }
 }
}

package sk.sharesapp.exceptions;

import java.sql.* ;

public class DaoException extends SQLException
{
       public DaoException()
       {
       }
       public DaoException(String aMessage)
       {
             super(aMessage) ;
       }
}
```

```
package sk.sharesapp.dao;
import              // imports not shown

public interface Dao
{
  Collection findAll() throws DaoException ;
}
```

```
package sk.sharesapp.dao ;
import              // imports not shown

public class CompanyDao implements Dao
{
  DataSource ds = null ;
  public CompanyDao() throws DaoException
  {

    try
    {
      Context context = new InitialContext() ;
      ds = (DataSource) context.lookup ("jdbc/sharesDS") ;
    }
    catch(NamingException e)
    {
      throw new DaoException(e.getMessage() ) ;
    }
  }
  public CompanyTO findByPrimaryKey(long companyId) throws DaoException
  {
    Connection con = null ;
    PreparedStatement ps = null ;
    ResultSet rs = null ;
    CompanyTO to = null ;
    try
    {
      con = getConnection() ;
      ps = con.prepareStatement
                         ("SELECT * FROM COMPANY WHERE COMPANYID = ?") ;
      ps.setLong(1, companyId) ;
    rs = ps.executeQuery() ;

    if (!rs.next() )
      throw new DaoException("Cannot find Company " + companyId) ;
    else
    {
     to = new CompanyTO() ;
     to.setCompanyId(rs.getLong("COMPANYID"));
     to.setSymbol(rs.getString("SYMBOL"));
     to.setName(rs.getString("NAME"));
     to.setSharePrice(rs.getDouble("SHAREPRICE"));
     to.setHigh(rs.getDouble("HIGH")) ;
     to.setLow(rs.getDouble("LOW"));
    }
  }
  catch (SQLException e)
  {
   throw new DaoException(e.getMessage() ) ;
  }
  finally   {     closeResources(con, ps, rs) ;    }
  return to;
}
public Collection findAll() throws DaoException
{
  Connection con = null ;
  PreparedStatement ps = null ;
  ResultSet rs = null ;
  ArrayList a = new ArrayList() ;
  try
  {
   con = getConnection() ;
   ps = con.prepareStatement("SELECT * FROM COMPANY") ;
   rs = ps.executeQuery() ;

   while (rs.next() )
   {
    CompanyTO to = new CompanyTO() ;
    to.setCompanyId(rs.getLong("COMPANYID"));
    to.setSymbol(rs.getString("SYMBOL"));
    to.setName(rs.getString("NAME"));
    to.setSharePrice(rs.getDouble("SHAREPRICE"));
    to.setHigh(rs.getDouble("HIGH")) ;
    to.setLow(rs.getDouble("LOW"));
    a.add(to) ;
   }
  }
  catch (  /* Exception handling not shown */ ) { /* etc.   }
  return a;
}
public CompanyTO findBySymbol(String symbol) throws DaoException
{
  Connection con = null ;
  PreparedStatement ps = null ;
  ResultSet rs = null ;
  CompanyTO to = null ;
  try
  {
   con = getConnection() ;
   ps = con.prepareStatement
               ("SELECT * FROM COMPANY WHERE SYMBOL = ?") ;
   ps.setString(1, symbol) ;
   rs = ps.executeQuery() ;
   if (!rs.next() )
     throw new DaoException("Cannot find Company with symbol " + symbol) ;
   else
   {
    to = new CompanyTO() ;
    to.setCompanyId(rs.getLong("COMPANYID"));
    to.setSymbol(rs.getString("SYMBOL"));
    to.setName(rs.getString("NAME"));
    to.setSharePrice(rs.getDouble("SHAREPRICE"));
    to.setHigh(rs.getDouble("HIGH")) ;
    to.setLow(rs.getDouble("LOW"));
   }
  }
  catch (  /* Exception handling not shown */ )    { /* etc. */ }
  return to;
}
```

```
public long add(CompanyTO to) throws DaoException
{
 Connection con = null ;
 PreparedStatement ps = null ;
 ResultSet rs = null ;
 long pk = -1 ;
 try
 {
   PrimaryKeyGenerator pkGen = PrimaryKeyGenerator.getInstance() ;
   pk = pkGen.getNextPK() ;
   con = getConnection() ;
   ps = con.prepareStatement("INSERT INTO Company VALUES(?, ?, ?, ?, ?, ?)") ;
   ps.setLong(1, pk) ;
   ps.setString(2, to.getSymbol() ) ;
   ps.setString(3, to.getName() ) ;       ps.setDouble(4, to.getSharePrice() ) ;
   ps.setDouble(5, to.getHigh() ) ;
   ps.setDouble(6, to.getLow() ) ;
   int rows = ps.executeUpdate() ;
   if (rows != 1)
        throw new DaoException("Failed to create Company " + to.getCompanyId()) ;
 }
 catch ( /* Exception handling not shown */ )    { /* etc. */ }
 return pk ;
}
public void delete(CompanyTO to) throws DaoException
{
 Connection con = null ;
 PreparedStatement ps = null ;    ResultSet rs = null ;
 try
 {
   con = getConnection() ;
   ps = con.prepareStatement
              ("DELETE FROM COMPANY WHERE COMPANYID = ?") ;
   ps.setLong(1, to.getCompanyId() ) ;
   int rows = ps.executeUpdate() ;
   if (rows != 1)
     throw new DaoException("Failed to remove Company " + to.getCompanyId()) ;
 }
 catch ( /* Exception handling not shown */ )    { /* etc. */ }
}
```

```
public void amend(CompanyTO to) throws DaoException
{
 Connection con = null ;
 PreparedStatement ps = null ;
 ResultSet rs = null ;
 try
 {
  con = getConnection() ;
  String command =
     "UPDATE COMPANY SET SYMBOL = ?, NAME = ?, SHAREPRICE = ?, " +
                                   "HIGH = ?, LOW = ? WHERE COMPANYID = ?" ;
  ps = con.prepareStatement(command) ;

  ps.setString(1, to.getSymbol()) ;
  ps.setString(2, to.getName()) ;
  ps.setDouble(3, to.getSharePrice()) ;
  ps.setDouble(4, to.getHigh()) ;
  ps.setDouble(5, to.getLow()) ;
  ps.setLong(6, to.getCompanyId() ) ;
  int rows = ps.executeUpdate() ;
  if (rows != 1)
  {
   throw new DaoException("Failed to store Company " + to.getSymbol()) ;
  }
 }
 catch ( /* Exception handling not shown */ )    { /* etc. */ }
}

private Connection getConnection() throws DaoException
{
 try
 {
  Connection con = ds.getConnection();
  return con ;
 }
 catch ( /* Exception handling not shown */ )    { /* etc. */ }
}
```

```
 private void closeResources(Connection con, PreparedStatement ps, ResultSet rs)
                    throws DaoException
 {
  try
  {
   if (rs != null)    {      rs.close() ;       rs = null ;     }
   if (ps != null)    {      ps.close() ;       ps = null ;     }
   if (con != null)  {      con.close() ;      con = null ;  }
  }
  catch(SQLException e)   {     throw new DaoException(e.getMessage() ) ;   }
 }
}

package sk.sharesapp.dao;
import              // imports not shown

public class DaoFactory
{
 private static final String packageName = "sk.sharesapp.dao." ;

 private static DaoFactory factory = null ;
 private HashMap map = null ;
 private DaoFactory()
 {
  map = new HashMap() ;
 }
 public synchronized static DaoFactory getInstance()
 {
    if (factory == null)    // first time
         factory = new DaoFactory() ;
    return factory ;
}

public synchronized Dao createDao(String daoName) throws DaoException
{
  Dao dao = null ;
  try
  {
    daoName = packageName + daoName ;

    if (map.containsKey(daoName) )
   {
    dao = (Dao) map.get(daoName) ;
   }
   else
   {
    Class theClass = Class.forName(daoName) ;
    Object theObject = theClass.newInstance() ;

    dao = (Dao) theObject ;
    map.put(daoName, dao) ;
   }
  }
  catch ( /* Exception handling not shown */ )   { /* etc. */ }
  return dao ;
 }
}

package sk.sharesapp.dao;
import              // imports not shown

public class TestCompanyDao
{
 // CompanyDao dao = null ;
 public TestCompanyDao() throws DaoException
 {
  // dao = new CompanyDao() ;
 }
 public void testFind() throws DaoException
 {
  DaoFactory factory = DaoFactory.getInstance() ;
  CompanyDao dao = (CompanyDao) factory.createDao("CompanyDao") ;
  CompanyTO to ;
  System.out.println("Test findAll()") ;
  Collection col = dao.findAll() ;
  Iterator it = col.iterator() ;
  while (it.hasNext() )
  {
   to = (CompanyTO) it.next() ;
   display(to) ;
  }
  System.out.println("Test findByPrimaryKey(10001)") ;
  to = dao.findByPrimaryKey(10001) ;
  display(to) ;
  System.out.println("Test findByPrimaryKey(10005)") ;
  to = dao.findByPrimaryKey(10005) ;
  display(to) ;
  System.out.println("Test findBySymbol(\"AIB\")") ;
  to = dao.findBySymbol("AIB") ;
  display(to) ;
}
public void testAddAmendDelete() throws DaoException
{
  DaoFactory factory = DaoFactory.getInstance() ;
  CompanyDao dao = (CompanyDao) factory.createDao("CompanyDao") ;
  CompanyTO to = new CompanyTO(20001, "XXX", "XXXX", 2.51, 2.61, 2.41) ;
  System.out.println("Test add") ;
  long pk = dao.add(to);
  to = dao.findByPrimaryKey(pk) ;
  display(to) ;
  to = new CompanyTO(pk, "YYY", "YYYY", 3.51, 3.61, 3.41) ;
  System.out.println("Test amend") ;
  dao.amend(to);
  to = dao.findByPrimaryKey(pk) ;
  display(to) ;
  System.out.println("Test delete") ;    dao.delete(to);
  System.out.println("After delete") ;    to = dao.findByPrimaryKey(pk) ;
}
public void display(CompanyTO to)
{
  System.out.println(to.getCompanyId() + "\t" + to.getSymbol() + "\t" +
                     to.getName() + "\t" + to.getSharePrice() + "\t" +
                     to.getHigh() + "\t" + to.getLow() ) ;
}
public static void main(String[] args)
{
  try
  {
    TestCompanyDao t = new TestCompanyDao() ;
    t.testFind() ;     t.testAddAmendDelete() ;
  }
  catch ( /* Exception handling not shown */ )    { /* etc. */ }
}
```

```
package sk.sharesapp.ejb.entity.bmp;
import              // imports not shown

public interface BmpCompanyPatternsRemoteHome extends EJBHome
{
  BmpCompanyPatternsRemote create() throws RemoteException, CreateException;

  BmpCompanyPatternsRemote findByPrimaryKey(Long primaryKey)
                                          throws RemoteException, FinderException;
  BmpCompanyPatternsRemote
       create(String symbol, String name, double sharePrice, double high, double low)
                                          throws RemoteException, CreateException;
}

package sk.sharesapp.ejb.entity.bmp;
import              // imports not shown

public interface BmpCompanyPatternsRemote extends EJBObject
{
  Long getCompanyId() throws RemoteException;
  //void setCompanyId(Long companyId) throws RemoteException;
  String getSymbol() throws RemoteException;
  void setSymbol(String symbol) throws RemoteException;
  String getName() throws RemoteException;
  void setName(String name) throws RemoteException;
  double getSharePrice() throws RemoteException;
  void setSharePrice(double sharePrice) throws RemoteException;
  double getHigh() throws RemoteException;
  void setHigh(double high) throws RemoteException;
  double getLow() throws RemoteException;
  void setLow(double low) throws RemoteException;
}
```

```
package sk.sharesapp.ejb.entity.bmp;
import            // imports not shown

public class BmpCompanyPatternsBean implements EntityBean
{
 public double low;
 public double high;
 public double sharePrice;
 public String name;
 public String symbol;
 public Long companyId;
 private EntityContext context;

 public Long ejbCreate()
 {
  return null;
 }
 public void ejbPostCreate()
 {
 }
 public Long ejbCreate(String symbol, String name, double sharePrice,
                  double high, double low) throws CreateException
 {
  System.out.println("ejbCreate") ;
  try
  {
   DaoFactory factory = DaoFactory.getInstance() ;
   CompanyDao dao = (CompanyDao) factory.createDao("CompanyDao") ;

   CompanyTO to = new CompanyTO(companyId.longValue(),
                       symbol, name, sharePrice, high, low) ;
   long newId = dao.add(to);
   companyId = new Long(newId) ;
  }
  catch(DaoException e)
  {
   throw new CreateException(e.getMessage() + " ejbCreate Failed") ;
  }
  return companyId ;
 }
```

```
public void ejbPostCreate(String symbol, String name, double sharePrice,
                                          double high, double low)
{
 System.out.println("Entered ejbPostCreate()") ;
}
public Long ejbFindByPrimaryKey(Long primaryKey) throws FinderException
{
 System.out.println("Entered ejbFindByPrimaryKey()") ;
 Connection con = null ;
 try
 {
  DaoFactory factory = DaoFactory.getInstance() ;
  CompanyDao dao = (CompanyDao) factory.createDao("CompanyDao") ;
  dao.findByPrimaryKey(primaryKey.longValue()) ;
 }
 catch(DaoException e)   {    throw new FinderException(e.getMessage() ) ;   }
 return primaryKey;
}
public void ejbLoad() throws EJBException
{
 System.out.println("Entered ejbLoad()") ;
 try
 {
  Long primaryKey = (Long) context.getPrimaryKey() ;

  DaoFactory factory = DaoFactory.getInstance() ;
  CompanyDao dao = (CompanyDao) factory.createDao("CompanyDao") ;

  CompanyTO to = dao.findByPrimaryKey(primaryKey.longValue()) ;
  companyId = primaryKey ;
  symbol = to.getSymbol() ;
           name = to.getName() ;
           sharePrice = to.getSharePrice() ;
           high = to.getHigh() ;
           low = to.getLow() ;
 }
 catch(DaoException e)   {   throw new EJBException(e.getMessage() ) ;   }
}
```

```
public void ejbRemove() throws RemoveException
 {
  System.out.println("Entered ejbRemove()") ;
  try
  {
    CompanyTO to = new CompanyTO() ;

    to.setCompanyId(companyId.longValue()) ;
    to.setSymbol(symbol) ;
    to.setName(name) ;
    to.setSharePrice(sharePrice) ;
    to.setHigh(high) ;
    to.setLow(low) ;

    DaoFactory factory = DaoFactory.getInstance() ;
    CompanyDao dao = (CompanyDao) factory.createDao("CompanyDao") ;
    dao.delete(to);
  }
  catch(DaoException e)   {   throw new RemoveException(e.getMessage() ) ;   }
}
 public void ejbStore() throws EJBException
 {
  System.out.println("Entered ejbStore()") ;
  try
  {
    CompanyTO to = new CompanyTO() ;
    to.setCompanyId(companyId.longValue()) ;
    to.setSymbol(symbol) ;
    to.setName(name) ;
    to.setSharePrice(sharePrice) ;
    to.setHigh(high) ;
    to.setLow(low) ;
    DaoFactory factory = DaoFactory.getInstance() ;
    CompanyDao dao = (CompanyDao) factory.createDao("CompanyDao") ;
    dao.amend(to);
  }
  catch(DaoException e)   {  throw new EJBException(e.getMessage() ) ;   }

 }

 public void ejbActivate()  {   System.out.println("Entered ejbActivate()") ; }
```

```
 public void ejbPassivate()  {    System.out.println("Entered ejbPassivate()") ; }
 public void setEntityContext(EntityContext ctx)
 {
  System.out.println("Entered setEntityContext()") ;
  this.context = ctx;
 }
 public void unsetEntityContext()
 {
  System.out.println("Entered unsetEntityContext()") ;
  this.context = null;
 }
 public Long getCompanyId()
 {
  return companyId;
 }
 public void setCompanyId(Long companyId)
 {
  this.companyId = companyId;
 }
 public String getSymbol()  {   return symbol;  }
 public void setSymbol(String symbol) {   this.symbol = symbol;  }
 public String getName()  {   return name;  }
 public void setName(String name)  {   this.name = name;  }
 public double getSharePrice()  {   return sharePrice;  }
 public void setSharePrice(double sharePrice)  {   this.sharePrice = sharePrice;  }
 public double getHigh()  {   return high;  }
 public void setHigh(double high)  {   this.high = high;  }
 public double getLow()  {   return low;  }
 public void setLow(double low)  {   this.low = low;  }
}

package sk.sharesapp.ejb.entity.bmp;
import            //         imports not shown
public class BmpCompanyPatternsRemoteClient
{
 public static void main(String [] args)
 {
  BmpCompanyPatternsRemoteClient bmpCompanyPatternsRemoteClient =
                                          new BmpCompanyPatternsRemoteClient();
try
{
 Context context = getInitialContext();
 BmpCompanyPatternsRemoteHome bmpCompanyPatternsRemoteHome =
           (BmpCompanyPatternsRemoteHome)PortableRemoteObject.narrow
                (context.lookup("BmpCompanyPatternsEJB"),
                     bmpCompanyPatternsRemoteHome.class);
 BmpCompanyPatternsRemote bmpCompanyPatternsRemote;
 System.out.println("Test findByPrimaryKey(10001)") ;
 bmpCompanyPatternsRemote =
    bmpCompanyPatternsRemoteHome.findByPrimaryKey
                                          (new Long(10001)) ;
 System.out.println(bmpCompanyPatternsRemote.getCompanyId( ) ) ;
 System.out.println(bmpCompanyPatternsRemote.getSymbol( ) + "\t" +
           bmpCompanyPatternsRemote.getName() + "\t" +
           bmpCompanyPatternsRemote.getSharePrice() + "\t" +
           bmpCompanyPatternsRemote.getHigh() + "\t" +
           bmpCompanyPatternsRemote.getLow()) ;

 System.out.println("Test Create(XXX, XXXX, 2.51, 2.61, 2.41)") ;
 BmpCompanyPatternsRemote newRemote =
     bmpCompanyPatternsRemoteHome.create("XXX", "XXXX",
                                                2.51, 2.61, 2.41) ;
 Long newId = newRemote.getCompanyId() ;
 System.out.println("Now test findByPrimaryKey(" + newId  + ")") ;
 bmpCompanyPatternsRemote =
     bmpCompanyPatternsRemoteHome.findByPrimaryKey(newId) ;
 System.out.println( bmpCompanyPatternsRemote.getCompanyId() + "\t" +
           bmpCompanyPatternsRemote.getSymbol( ) + "\t" +
           bmpCompanyPatternsRemote.getName() + "\t" +
           bmpCompanyPatternsRemote.getSharePrice() + "\t" +
           bmpCompanyPatternsRemote.getHigh() + "\t" +
           bmpCompanyPatternsRemote.getLow()) ;
 System.out.println("Now test set methods)") ;
 bmpCompanyPatternsRemote.setSymbol("YYY") ;
 bmpCompanyPatternsRemote.setName("YYYY") ;
 bmpCompanyPatternsRemote.setSharePrice(3.51) ;
 bmpCompanyPatternsRemote.setHigh(3.61) ;
 bmpCompanyPatternsRemote.setLow(3.41) ;
 System.out.println("Now test findByPrimaryKey(" + newId + ")") ;
 bmpCompanyPatternsRemote =
    bmpCompanyPatternsRemoteHome.findByPrimaryKey(newId) ;
 System.out.println(bmpCompanyPatternsRemote.getSymbol( ) + "\t" +
              bmpCompanyPatternsRemote.getName() + "\t" +
              bmpCompanyPatternsRemote.getSharePrice() + "\t" +
              bmpCompanyPatternsRemote.getHigh() + "\t" +
              bmpCompanyPatternsRemote.getLow()) ;
 System.out.println("Now test Remove()") ;
 bmpCompanyPatternsRemote.remove();
 System.out.println("Now test findByPrimaryKey(" + newId + ")") ;
 bmpCompanyPatternsRemote =
       bmpCompanyPatternsRemoteHome.findByPrimaryKey(newId) ;
}
catch(/* Exception handling not shown */)   {/* etc. */   }
}

 private static Context getInitialContext() throws NamingException
 {
  return new InitialContext();
 }
}

package sk.sharesapp.ejb.entity.cmp;
import            // imports not shown

public interface CompanyRemoteHome extends EJBHome
{
 CompanyRemote create() throws RemoteException, CreateException;
 CompanyRemote create(Long companyid, String symbol, String name,
                           Double shareprice, Double high, Double low)
               throws RemoteException, CreateException;
 CompanyRemote findByPrimaryKey(Long primaryKey)
               throws RemoteException, FinderException;
 Collection findAll() throws RemoteException, FinderException;
 Collection findCompaniesWithPriceLessThan(Double sharePrice)
                           throws RemoteException, FinderException;
 Collection findCompaniesWithNameLike()
                    throws RemoteException, FinderException;
 CompanyRemote findCompanyBySymbol(String symbol)
                    throws RemoteException, FinderException;
}
```

```
package sk.sharesapp.ejb.entity.cmp;
import          // imports not shown

public interface CompanyRemote extends EJBObject
{
 Long getCompanyid() throws RemoteException;
 String getSymbol() throws RemoteException;
 void setSymbol(String symbol) throws RemoteException;
 String getName() throws RemoteException;
 void setName(String name) throws RemoteException;
 Double getShareprice() throws RemoteException;
 void setShareprice(Double shareprice) throws RemoteException;
 Double getHigh() throws RemoteException;
 void setHigh(Double high) throws RemoteException;
 Double getLow() throws RemoteException;
 void setLow(Double low) throws RemoteException;
}

package sk.sharesapp.ejb.entity.cmp;
import          // imports not shown

public abstract class CompanyBean implements EntityBean
{
 private EntityContext context;

 public Long ejbCreate() {   return null; }

 public void ejbPostCreate() { }

 public Long ejbCreate(Long companyid, String symbol, String name,
                   Double shareprice, Double high, Double low)
 {
  setCompanyid(companyid);
  setSymbol(symbol);
  setName(name);
  setShareprice(shareprice);
  setHigh(high);
  setLow(low);
  return companyid;
 }
```

```
 public void ejbPostCreate(Long companyid, String symbol, String name,
                   Double shareprice, Double high, Double low)
 {
 }
 public void ejbActivate() { }
 public void ejbLoad() { }
 public void ejbPassivate() { }
 public void ejbRemove() { }
 public void ejbStore() { }
 public void setEntityContext(EntityContext ctx)
 {
  this.context = ctx;
 }
 public void unsetEntityContext()
 {
  this.context = null;
 }
 public abstract Long getCompanyid();
 public abstract void setCompanyid(Long companyid);
 public abstract String getSymbol();
 public abstract void setSymbol(String symbol);
 public abstract String getName();
 public abstract void setName(String name);
 public abstract Double getShareprice();
 public abstract void setShareprice(Double shareprice);
 public abstract Double getHigh();
 public abstract void setHigh(Double high);
 public abstract Double getLow();
 public abstract void setLow(Double low);
}

package sk.sharesapp.ejb.entity.cmp;
import          // imports not shown

public class CompanyRemoteClient
{
 public static void main(String [] args)
 {
  CompanyRemoteClient companyRemoteClient = new CompanyRemoteClient();
try
{
 Context context = getInitialContext();
 CompanyRemoteHome companyRemoteHome =
   (CompanyRemoteHome)PortableRemoteObject.narrow
       (context.lookup("CompanyEJB"), CompanyRemoteHome.class);
 CompanyRemote companyRemote;

 Collection companies = companyRemoteHome.findAll();
 Iterator iter = companies.iterator();
 while (iter.hasNext())
 {
   companyRemote = (CompanyRemote)iter.next();
   System.out.println(companyRemote.getCompanyid() + "\t" +
        companyRemote.getSymbol() + "\t" +  companyRemote.getName() + "\t" +
         companyRemote.getShareprice() + "\t" + companyRemote.getHigh() + "\t" +
         companyRemote.getLow());
 }

 System.out.println("Now findCompanyBySymbol(Iona)" ) ;
 companyRemote = companyRemoteHome.findCompanyBySymbol("Iona") ;
 System.out.println(companyRemote.getCompanyid() + "\t" +
       companyRemote.getSymbol() + "\t" + companyRemote.getName() + "\t" +
        companyRemote.getShareprice() + "\t" + companyRemote.getHigh() + "\t" +
         companyRemote.getLow());
 Double sharePrice = new Double(140.0) ;
 System.out.println("Now list companies with share price < " + sharePrice) ;
 companies =
    companyRemoteHome.findCompaniesWithPriceLessThan(sharePrice) ;
 iter = companies.iterator();
 while (iter.hasNext())
 {
   companyRemote = (CompanyRemote)iter.next();
   System.out.println(companyRemote.getCompanyid() + "\t" +
        companyRemote.getSymbol() + "\t" + companyRemote.getName() + "\t" +
         companyRemote.getShareprice() + "\t" +  companyRemote.getHigh() + "\t" +
          companyRemote.getLow());
 }
 System.out.println("Now list companies with name like bank") ;
 companies = companyRemoteHome.findCompaniesWithNameLike() ;
 iter = companies.iterator();
 while (iter.hasNext())
 {
   companyRemote = (CompanyRemote)iter.next();
   System.out.println(companyRemote.getCompanyid() + "\t" +
        companyRemote.getSymbol() + "\t" + companyRemote.getName() + "\t" +
        companyRemote.getShareprice() + "\t" +  companyRemote.getHigh() + "\t" +
        companyRemote.getLow());
 }
 Long pk = new Long(2001) ;
 System.out.println("Add new company: 2001, AAA, AAAA, 4.75, 4.85, 4.65") ;
 companyRemote = companyRemoteHome.create( pk, "AAA", "AAAA",
      new Double(4.75), new Double(4.85), new Double(4.95) );
 System.out.println("Now findByPrimaryKey(" + pk + ")") ;
 companyRemote = companyRemoteHome.findByPrimaryKey(pk) ;
 System.out.println(companyRemote.getCompanyid() + "\t" +
      companyRemote.getSymbol() + "\t" + companyRemote.getName() + "\t" +
      companyRemote.getShareprice() + "\t" + companyRemote.getHigh() + "\t" +
      companyRemote.getLow());
 System.out.println
 ("Now modify company: 2001, BBB, BBBB, 5.75, 5.85, 5.65, using set methods") ;
 companyRemote.setSymbol("BBB") ;
 companyRemote.setName("BBBB") ;
 companyRemote.setShareprice(new Double(5.75) ) ;
 companyRemote.setHigh(new Double(5.85) ) ;
 companyRemote.setLow(new Double(5.65) ) ;
 System.out.println("Now findByPrimaryKey(" + pk + ")") ;
 companyRemote = companyRemoteHome.findByPrimaryKey(pk) ;
 System.out.println(companyRemote.getCompanyid() + "\t" +
      companyRemote.getSymbol() + "\t" + companyRemote.getName() + "\t" +
      companyRemote.getShareprice() + "\t" + companyRemote.getHigh() + "\t" +
     companyRemote.getLow());
 System.out.println("Now remove company " + pk ) ;
 companyRemote.remove();
 System.out.println("Now findByPrimaryKey(" + pk + ")") ;
 companyRemote = companyRemoteHome.findByPrimaryKey(pk) ;
}
catch( /* Exception handling not shown */ )    { /* etc. */   }
}
private static Context getInitialContext() throws NamingException  {
 return new InitialContext();
} }
```

```
package sk.sharesapp.ejb.session.stateless;
import            // imports not shown

public interface CompanySessionFacadeRemoteHome extends EJBHome
{
 CompanySessionFacadeRemote create()
                    throws RemoteException, CreateException;
}

package sk.sharesapp.ejb.session.stateless;
import            // imports not shown

public interface CompanySessionFacadeRemote extends EJBObject
{
 long addCompany(String symbol, String name, double sharePrice,
                                     double high, double low)
     throws DaoException,  RemoteException, CreateException,
                              NamingException, ServiceLocatorException;
 long addCompany(CompanyTO to) throws RemoteException, CreateException,
                              NamingException,  DaoException, ServiceLocatorException;
 void amendCompany(CompanyTO to) throws  FinderException,
                    NamingException, RemoteException, ServiceLocatorException;
 void deleteCompany(CompanyTO to) throws RemoteException, FinderException,
          RemoveException, NamingException, ServiceLocatorException;
 CompanyTO getCompany(long companyId) throws RemoteException,
                    FinderException, NamingException, ServiceLocatorException;
 CompanyTO getCompanyBySymbol(String symbol) throws RemoteException,
                    FinderException, NamingException, ServiceLocatorException;
}

package sk.sharesapp.ejb.session.stateless;
import            // imports not shown

public class CompanySessionFacadeBean implements SessionBean
{
 public void ejbCreate()  { }
 public void ejbActivate()  { }
 public void ejbPassivate()  { }
 public void ejbRemove()  { }
 public void setSessionContext(SessionContext ctx)  { }
```

```
public long addCompany(String symbol, String name, double sharePrice,
                              double high, double low)
        throws CreateException, NamingException,
                                     DaoException, ServiceLocatorException
{
 PrimaryKeyGenerator pkGenerator = PrimaryKeyGenerator.getInstance() ;
 long pk = pkGenerator.getNextPK() ;
 CompanyLocalHome localHome = getCompanyLocalHome() ;
 CompanyLocal local =
        localHome.create(new Long(pk), symbol, name, new Double(sharePrice),
                    new Double(high), new Double(low) ) ;
 return pk ;
}

public long addCompany(CompanyTO to) throws CreateException,
                  NamingException, DaoException, ServiceLocatorException
{
 PrimaryKeyGenerator pkGenerator = PrimaryKeyGenerator.getInstance() ;
 long pk = pkGenerator.getNextPK() ;

 CompanyLocalHome localHome = getCompanyLocalHome() ;
 CompanyLocal local =
        localHome.create(new Long(pk), to.getSymbol(), to.getName(),
                              new Double(to.getSharePrice()),
                              new Double(to.getHigh()),
                              new Double(to.getLow()) ) ;
  return pk ;
}

public void amendCompany(CompanyTO to) throws  FinderException,
                                     NamingException, ServiceLocatorException
{
 CompanyLocalHome localHome = getCompanyLocalHome() ;
 CompanyLocal local =
        localHome.findByPrimaryKey(new Long(to.getCompanyId())) ;
 local.setSymbol(to.getSymbol());
 local.setName(to.getName());
 local.setShareprice(new Double(to.getSharePrice()));
 local.setHigh(new Double(to.getHigh()));
 local.setLow(new Double(to.getLow()));
}
```

```
public void deleteCompany(CompanyTO to) throws NamingException,
                FinderException, RemoveException, ServiceLocatorException
{
  CompanyLocalHome localHome = getCompanyLocalHome() ;
  CompanyLocal local =
    localHome.findByPrimaryKey(new Long(to.getCompanyId())) ;
  local.remove();
}

public CompanyTO getCompany(long companyId) throws NamingException,
                 FinderException, ServiceLocatorException
{
  CompanyLocalHome localHome = getCompanyLocalHome() ;
  CompanyLocal local = localHome.findByPrimaryKey(new Long(companyId)) ;
  CompanyTO to = new CompanyTO() ;
  to.setCompanyId(local.getCompanyid().longValue());
  to.setSymbol(local.getSymbol());
  to.setName(local.getName());
  to.setSharePrice(local.getShareprice().doubleValue());
  to.setHigh(local.getHigh().doubleValue());
  to.setLow(local.getHigh().doubleValue());

  return to ;
}

public CompanyTO getCompanyBySymbol(String symbol) throws NamingException,
                                      FinderException, ServiceLocatorException
{
  CompanyLocalHome localHome = getCompanyLocalHome() ;
  CompanyLocal local = localHome.findCompanyBySymbol(symbol) ;
  CompanyTO to = new CompanyTO() ;
  to.setCompanyId(local.getCompanyid().longValue());
  to.setSymbol(local.getSymbol());
  to.setName(local.getName());
  to.setSharePrice(local.getShareprice().doubleValue());
  to.setHigh(local.getHigh().doubleValue());
  to.setLow(local.getHigh().doubleValue());

  return to ;
}
```

```
  private CompanyLocalHome getCompanyLocalHome()
        throws ServiceLocatorException //NamingException
  {
   //final InitialContext context = new InitialContext();
   //return
   //(CompanyLocalHome)context.lookup("java:comp/env/ejb/local/CompanyEJB");

   ServiceLocator serviceLocator = ServiceLocator.getInstance() ;
   CompanyLocalHome localHome = (CompanyLocalHome)
                        serviceLocator.getLocalHome("CompanyEJB") ;
   return localHome ;
  }
}

package sk.sharesapp.ejb.session.stateless;
import            // imports not shown

public class CompanySessionFacadeRemoteClient
{
  public static void main(String [] args)
  {
   CompanySessionFacadeRemoteClient companySessionFacadeRemoteClient =
                 new CompanySessionFacadeRemoteClient();
   try
   {
    Context context = getInitialContext();
    CompanySessionFacadeRemoteHome companySessionFacadeRemoteHome =
    (CompanySessionFacadeRemoteHome)PortableRemoteObject.narrow
        (context.lookup("CompanySessionFacadeEJB"),
                     CompanySessionFacadeRemoteHome.class);
    CompanySessionFacadeRemote companySessionFacadeRemote =
     companySessionFacadeRemoteHome.create();

    // The id is ignored as pk will be generated automatically
    CompanyTO to = new CompanyTO(123, "AAA", "AAAA", 4.5, 4.6, 4.4) ;
    System.out.println("Create new company(AAA, AAA, 4.5, 4.6, 4.4)") ;
    //  long pk = companySessionFacadeRemote.addCompany(to) ;
    long pk = companySessionFacadeRemote.addCompany("AAA", "AAAA", 4.5,
                         4.6, 4.4) ;
    System.out.println("Primary key of new company is " + pk) ;
    System.out.println("Now get company(" + pk + ")") ;
    to = companySessionFacadeRemote.getCompany(pk) ;
    System.out.println(to.getCompanyId() + "\t" + to.getSymbol() + "\t" +
                    to.getName() + "\t" + to.getSharePrice() + "\t" +
                    to.getHigh() + "\t" + to.getLow() ) ;
    System.out.println("Now get company(AAA)") ;
    to = companySessionFacadeRemote.getCompanyBySymbol("AAA") ;
    System.out.println(to.getCompanyId() + "\t" + to.getSymbol() + "\t" +
                    to.getName() + "\t" + to.getSharePrice() + "\t" +
                    to.getHigh() + "\t" + to.getLow() ) ;
    System.out.println("Now amend company(" + pk + " BBB, BBBB, 5.5, 5.6, 5.4)") ;
    to = new CompanyTO(pk, "BBB", "BBBB", 5.5, 5.6, 5.4) ;
    companySessionFacadeRemote.amendCompany(to) ;
    System.out.println("Now get company amended company(" + pk + ")") ;
    to = companySessionFacadeRemote.getCompany(pk) ;
    System.out.println(to.getCompanyId() + "\t" + to.getSymbol() + "\t" +
                    to.getName() + "\t" + to.getSharePrice() + "\t" +
                    to.getHigh() + "\t" + to.getLow() ) ;
    System.out.println("Now delete company(" + pk + ")") ;
    companySessionFacadeRemote.deleteCompany(to) ;
    System.out.println("Now attempt to get company deleted company(" + pk + ")") ;
    to = companySessionFacadeRemote.getCompany(pk) ;
   }
   catch(/* Exception handling not shown */    { /* etc. */   }

  }

  private static Context getInitialContext() throws NamingException
  {
   return new InitialContext();
  }
}
```

```
package sk.sharesapp.utils ;
import            // imports not shown

/*
    This client is used by local clients to lookup
    local home interfaces.  Exercise for students:
    1. Modify to allow other lookups, e.g. data sources,
       Queue connections etc.
     2. Write Service Locator for remote look ups
    The class is implemented as Singleton.
 */

 public class ServiceLocator
 {
  private static ServiceLocator serviceLocator = null ;
  private Context context = null ;
  private Map cache = null ;

  private ServiceLocator() throws ServiceLocatorException
  {
    try
    {
         context = new InitialContext() ;
         cache = new HashMap() ;
    }
    catch(NamingException e)
    {
         throw new ServiceLocatorException(e.getMessage() ) ;
    }
 }
public static ServiceLocator getInstance() throws ServiceLocatorException
{
    if (serviceLocator == null)
         serviceLocator = new ServiceLocator() ;

    return serviceLocator ;
}
```

```
public EJBLocalHome getLocalHome(String ejbName)
                              throws ServiceLocatorException
{
     EJBLocalHome localHome = null;
     try
     {
          localHome = (EJBLocalHome) cache.get(ejbName) ;
          if (localHome == null)              // not already in cache
          {
            localHome = (EJBLocalHome)
               context.lookup("java:comp/env/ejb/local/" + ejbName);
            cache.put(ejbName, localHome) ;                // cache it
          }
     }
     catch (NamingException e)
     {
          throw new ServiceLocatorException(e.getMessage() ) ;
     }
     catch (Exception e)
     {
          throw new ServiceLocatorException(e.getMessage() ) ;
     }
     return localHome;
 }
}

package sk.sharesapp.ejb.mdb;
import            // imports not shown

public class SimpleMessageDrivenEJBBean
           implements MessageDrivenBean, MessageListener
{
  private MessageDrivenContext context;
  QueueConnection queueConnection ;
  QueueSession queueSession ;
  QueueReceiver queueReceiver ;
```

```
public void ejbCreate()
 {
  try
  {
   System.out.println("MDB ejbCreate()") ;
   InitialContext ctx = new InitialContext();
   System.out.println("Looking up Queue connection factory...") ;
   QueueConnectionFactory queueConnectionFactory =
      (QueueConnectionFactory) ctx.lookup("jms/QueueConnectionFactory");
   System.out.println("Looking up Queue...") ;
   Queue queue =       (Queue) ctx.lookup("jms/simpleQueue") ;
   System.out.println("Creating Queue connection...") ;
   queueConnection = queueConnectionFactory.createQueueConnection() ;
   queueConnection.start() ;
   System.out.println("Getting queue session...") ;
   queueSession = queueConnection.createQueueSession(false,
                              Session.AUTO_ACKNOWLEDGE);
   System.out.println("Creating queue receiver (i.e. a queue consumer)...") ;
   queueReceiver = queueSession.createReceiver(queue) ;
  }
  catch(  /* Exception handling not shown */ )  { /* etc. */ }
}
public void onMessage(Message msg)
 {
  try
  {
   if (msg instanceof TextMessage)
   {
    TextMessage txtMessage = (TextMessage) msg ;
   }
   else if (msg instanceof MapMessage)
   {
    MapMessage mapMessage = (MapMessage) msg ;
    String one = mapMessage.getString("one") ;
    int two =  mapMessage.getInt("two") ;
    double three = mapMessage.getDouble("three") ;
    System.out.println("Message received one: " + one  + " " + two + " " + three) ;
   }
```

```
    if (msg instanceof ObjectMessage)
    {
      ObjectMessage objMessage = (ObjectMessage) msg ;
      ArrayList a = (ArrayList) objMessage.getObject() ;
      for (int i = 0 ; i < a.size() ; i++)
      {
       String s = (String) a.get(i) ;
       System.out.println(s) ;
      }
    }
  }
  catch( /* Exception handling not shown * / )   { /* etc.  */ }
 }
 public void ejbRemove()
 {
  try
  {
    queueReceiver.close();
    queueSession.close();
    queueConnection.stop();
  }
  catch( /* Exception handling not shown * / )   { /* etc.  */ }
 }
 public void setMessageDrivenContext(MessageDrivenContext ctx)
 {
  this.context = ctx;
 }
}

package sk.sharesapp.utils;
import            // imports not shown

public class SimpleMessageDrivenSender
{
 Context context = null ;
 QueueConnection queueConnection ;
 QueueSession queueSession ;
 QueueSender queueSender ;
```

```
public SimpleMessageDrivenSender()
{
 context = getInitialContext();
}
public void run()
{
 try
 {
   System.out.println("Starting queue...") ;
   startQueue() ;
   System.out.println("Sending messages...") ;
   sendMessages() ;
   System.out.println("Stop queue") ;
   stopQueue() ;
 }
 catch( /* Exception handling not shown * / )   { /* etc.  */ }
}

public void startQueue()
{
 try
 {
   System.out.println("Looking up Queue connection factory...") ;
   QueueConnectionFactory queueConnectionFactory =
      (QueueConnectionFactory) context.lookup("jms/QueueConnectionFactory");
   System.out.println("Looking up Queue...") ;
   Queue queue = (Queue) context.lookup("jms/simpleQueue") ;
   System.out.println("Creating Queue connection...") ;
   queueConnection = queueConnectionFactory.createQueueConnection() ;
   System.out.println("Starting queueConnection...") ;
   queueConnection.start() ;
   System.out.println("Getting queue session...") ;
   queueSession = queueConnection.createQueueSession(false,
                                         Session.AUTO_ACKNOWLEDGE);
   System.out.println("Creating queue sender (i.e. a queue producer)...") ;
   queueSender = queueSession.createSender(queue) ;
 }
 catch( /* Exception handling not shown * / )   { /* etc.  */ }
}
```

```
 public void sendMessages() throws RemoteException,
         NamingException, JMSException
 {
   System.out.println("Sending text message...") ;
   TextMessage txtMessage = queueSession.createTextMessage() ;
   txtMessage.setText("This is my text message") ;
   queueSender.send(txtMessage) ;
   System.out.println("Sending map message...") ;
   MapMessage mapMessage = queueSession.createMapMessage() ;
   mapMessage.setStringProperty("MessageFormat", "Version 3.4") ;
   mapMessage.setString("one", "This is map message one") ;
   mapMessage.setInt("two", 2 ) ;
   mapMessage.setDouble("three", 3.0) ;
   queueSender.send(mapMessage) ;

   ArrayList a = new ArrayList() ;
   a.add("one") ; a.add("two") ; a.add("three") ;
   ObjectMessage objectMessage = queueSession.createObjectMessage() ;
   objectMessage.setObject(a) ;
   queueSender.send(objectMessage) ;
}
public void stopQueue()
{
   try
   {
     System.out.println("Closing queue connection...") ;
     queueConnection.close() ;
     System.out.println("Closing queue connection completed...") ;
   }
   catch( /* Exception handling not shown * / )   { /* etc.  */ }     }
}
public static void main(String [] args)
{
   SimpleMessageDrivenSender SimpleMessageDrivenSender =
                              new SimpleMessageDrivenSender();
   try   {
     SimpleMessageDrivenSender.run() ;
   }
   catch( /* Exception handling not shown * / )   { /* etc.  */ }     }
}
}

private Context getInitialContext()
{
  try
  {
   context = new InitialContext();
  }
 catch( /* Exception handling not shown * / )   { /* etc.  */ }     }

 return context ;
}
}
```

Appendix B: Chapter 5 code

```
package sk.sharesapp.utils ;
import sk.sharesapp.exceptions.RandomGeneratorException;

public class MyRandomGenerator
{
    public MyRandomGenerator()
    {
    }
    //    generate random number between 1 and end

    public static int generateNumber(int end) throws RandomGeneratorException
    {
        if (end < 2)
        {
         throw new RandomGeneratorException("end value must be greater >= 2") ;
        }
        double x = Math.random() ;      // 0.0 <= x < 1.0
        int number = (int) ( (x * end) + 1) ;

        return number ;
    }
    //    generate n random numbers between 1 and end, inclusive

    public static int[] generateNumbers(int n, int end)
                         throws RandomGeneratorException
    {
        if (n < 2)  {
                throw new RandomGeneratorException("n must be >= 2") ;
        }
        else if (end < 2)  {
          throw new RandomGeneratorException("end value must be >= 2") ;
        }
        else if (n > end)  {
          throw new RandomGeneratorException("n cannot be greater than end") ;
        }
     int[] vals = new int[end + 1] ;
     for (int i = 0 ; i <= end ; i++)
     {
       vals[i] = i ;
     }

     int[] numbers = new int[n] ;
     for (int i = 0 ; i < n ; i++)
     {
       double x = Math.random() ;          // 0.0 <= x < 1.0
       int number = (int) ( (x * (end - i) ) + 1) ; // 1 <= number <= end, end reducing by 1
       numbers[i] = vals[number] ;               // use numberas index into array
      // now swap selected number out of array to avoid repetition
       vals[number] = vals[end - i] ;
       vals[end - i] = numbers[i] ;
     }
     return numbers ;
    }

    public static void main(String[] args) throws RandomGeneratorException
    {
      for (int i = 0 ; i < 100 ; i++)
      {
         int n = MyRandomGenerator.generateNumber(6) ;
         if (n < 2)
           n = 2 ;

           int[] numbers = MyRandomGenerator.generateNumbers(n, 42) ;
           for (int j = 0 ; j < n ; j++)
           {
             System.out.print(numbers[j] + "\t") ;
           }
           System.out.println() ;
         }
      }
}
```

```
package sk.sharesapp.utils;
import             // imports not shown

/* Author:   Seamus Kelly
 * Project:  M.Sc. in Computing
 * Date:     September 2005
 *
 * SharePriceNews.java
 *
 * This application is the message producer of the main
 * application.  This application uses CompanyListerEJB
 * to read the list of companies from the database.  It
 * continuously selects at random a number of these companies
 * to update the share price.  It randomly selects the % change
 * in the share price and sends a list of theese to a
 * message queue.
 */

public class SharePriceNews
{
  private static final int minNumUpdates = 3 ;  // i.e. number of updates
  private static final int maxNumUpdates = 7 ;  //      in trend
  private static final int minNumCompanies = 2 ;
  private static final double modifier = 20.0 ;
  //  Modifier = 20 means that max. percentage change = 100 / 20, i.e. 5%
  private int numberUpdates = 0 ;
  private int[] companiesForUpdate ;
  private int[] trend ;
  Context context = null ;
  ArrayList companies = null ;
  ArrayList updateCompaniesList = null ;

  QueueConnection queueConnection ;
  QueueSession queueSession ;
  QueueSender queueSender ;
  ObjectMessage objectMessage ;

  public SharePriceNews()
  {
    context = getInitialContext();
  }
```

```
public void run()
{
 try
 {
   companies = readCompanies() ;
   //showCompanies() ;
   //selectCompaniesForUpdate() ;
   //showCompaniesForUpdate() ;
   //updateCompanies() ;
   //sendPriceChanges(updateCompaniesList) ;
   //showCompaniesForUpdate() ;

   System.out.println("Starting queue...") ;
   startQueue() ;
   System.out.println("Started queue...") ;

   for (int i = 0 ; i < 100 ; i++)
   {
     System.out.println("i = " + i) ;
     updateCompanies() ;
     while (numberUpdates > 0)
     {
       System.out.println("number updates = " + numberUpdates) ;
       sendPriceChanges(updateCompaniesList) ;
       //companies = readCompanies() ;
       //showCompanies() ;
       Thread.sleep(1000) ;
       updateCompanies() ;
     }
   }
   System.out.println("Stop queue") ;
   stopQueue() ;
  }
  catch( /*  Exception handling not shown */ )    { /* etc. */  }
}
```

```
public ArrayList readCompanies()
{
 ArrayList a = null ;
 try
 {
  Context context = getInitialContext();
  CompanyListerRemoteHome companyListerRemoteHome =
        (CompanyListerRemoteHome)
            PortableRemoteObject.narrow(context.lookup
                ("CompanyListerEJB"), CompanyListerRemoteHome.class);
  CompanyListerRemote companyListerRemote =
                                companyListerRemoteHome.create();
  a = (ArrayList) companyListerRemote.getCompanies() ;
 }
 catch( /* Exception handling not shown */ ) { /* etc. */ }
 return a ;
}

public void showCompanies()
{
 if (companies == null)
  return ;

 for (int i = 0 ; i < companies.size() ; i++)
 {
  CompanyTO to = (CompanyTO) companies.get(i) ;
  System.out.println(to.getCompanyId() + "\t" + to.getSymbol() + "\t" +
        to.getName() + "\t" + to.getSharePrice() + "\t" +
        to.getHigh() + "\t" + to.getLow() ) ;
 }

}

public void startQueue()
{
 try
 {
  System.out.println("Looking up Queue connection factory...") ;
  QueueConnectionFactory queueConnectionFactory =
        (QueueConnectionFactory) context.lookup("jms/QueueConnectionFactory");
  System.out.println("Looking up Queue...") ;
  Queue queue = (Queue) context.lookup("jms/myQueue") ;
  System.out.println("Creating Queue connection...") ;
  queueConnection = queueConnectionFactory.createQueueConnection() ;
  System.out.println("Starting queueConnection...") ;
  queueConnection.start() ;
  System.out.println("Getting queue session...") ;
  QueueSession queueSession = queueConnection.createQueueSession(false,
                                        Session.AUTO_ACKNOWLEDGE);
  System.out.println("Creating queue sender (i.e. a queue producer)...") ;
  queueSender = queueSession.createSender(queue) ;
  objectMessage = queueSession.createObjectMessage() ;
 }
 catch( /* Exception handling not shown */ ) { /* etc. */ }
}
public void stopQueue()
{
 try
 {
  System.out.println("Closing queue connection...") ;
  queueConnection.close() ;
 }
 catch( /* Exception handling not shown */ ) { /* etc. */ }
}
public void showCompaniesForUpdate()
 {
  for (int k = 0 ; k < companiesForUpdate.length ; k++)
  {
   System.out.println(companiesForUpdate[k] + "\t" + trend[k]) ;
  }
  for (int i = 0 ; i < updateCompaniesList.size() ; i++)
  {
   ShareChange sc = (ShareChange) updateCompaniesList.get(i) ;
   System.out.println(sc.getCompanyId() + "\t" + sc.getPercentChange() ) ;
   System.out.println(trend[i]) ;
  }
}
```

```
 public void updateCompanies() throws RandomGeneratorException
 {
    if (numberUpdates == 0)
      numberUpdates = selectCompaniesForUpdate() ;

   // Generates updateCompaniesList.size() numbers between 1 and 100
   // Must not be more than 100 companies
  int[] perCentChange =
      MyRandomGenerator.generateNumbers(updateCompaniesList.size(), 100) ;

  // (3) For each company, randomly generate % change in share price
  for (int i = 0 ; i < updateCompaniesList.size() ; i++)
  {
    double change =  ( (double) ( trend[i] * perCentChange[i]) ) / modifier ;
    System.out.println(change) ;
    ShareChange sc = (ShareChange) updateCompaniesList.get(i) ;
    sc.setPercentChange(change);
  }
  numberUpdates-- ;
 }
 public int selectCompaniesForUpdate()  throws RandomGeneratorException
 {
  // Determine number of updates to apply to estabish trend (3 to 7)
  numberUpdates = MyRandomGenerator.generateNumber(maxNumUpdates) ;
  if (numberUpdates < minNumUpdates)
     numberUpdates = minNumUpdates ;

 // (1)   Generate random number n, the number of companies to update
 // keys.length is the total number of companies

 int n = MyRandomGenerator.generateNumber(companies.size()) ;
 if (n < minNumCompanies)
     n = minNumCompanies ;

// (2)    Randomly select these n companies

companiesForUpdate =
MyRandomGenerator.generateNumbers(n, companies.size()) ;
updateCompaniesList = new ArrayList(companiesForUpdate.length) ;
  for (int i = 0 ; i < companiesForUpdate.length ; i++)
  {
    CompanyTO to = (CompanyTO) companies.get(companiesForUpdate[i] - 1) ;
    ShareChange sc = new ShareChange() ;
    sc.setCompanyId(to.getCompanyId());
  //  System.out.println(sc.getCompanyId()) ;
    updateCompaniesList.add(sc) ;
  }
  trend = new int[n] ;
  for (int i = 0 ; i < n ; i++)
  {
     int direction = MyRandomGenerator.generateNumber(2) ;
     if (direction == 1)
         trend[i] = -1 ;
    else
         trend[i] = 1 ;
   }
   return numberUpdates ;
 }
 public void sendPriceChanges(ArrayList list) throws
               RemoteException, NamingException, JMSException
 {
  objectMessage.setObject((ArrayList) list) ;
  queueSender.send(objectMessage) ;
  System.out.println("Object message sent...") ;
 }
 public static void main(String [] args)
 {
  SharePriceNews sharePriceNews = new SharePriceNews();
  try   {
   sharePriceNews.run() ;
  }
  catch(  /* Exception handling not shown */ )  {  /* etc. */    }
 }
 private Context getInitialContext()
 {
   try    {      context = new InitialContext();    }
  catch(  /* Exception handling not shown */ )  {  /* etc. */    }
  return context ;
 }
}
```

```
package sk.sharesapp.ejb.mdb;

import java.util.ArrayList;
package sk.sharesapp.ejb.mdb;

import             // imports not shown

public class PriceWatchMessageDrivenBean
                   implements MessageDrivenBean, MessageListener
{
  private MessageDrivenContext context;
  QueueConnection queueConnection ;
  QueueSession queueSession ;
  QueueReceiver queueReceiver ;

  public void ejbCreate()
  {
   try
   {
     System.out.println("MDB ejbCreate()") ;
     InitialContext ctx = new InitialContext();
     System.out.println("Got initial context...");
     System.out.println("Looking up Queue connection factory...") ;
     QueueConnectionFactory queueConnectionFactory =
           (QueueConnectionFactory) ctx.lookup("jms/QueueConnectionFactory");
     System.out.println("Looking up Queue...") ;
     Queue queue =       (Queue) ctx.lookup("jms/myQueue") ;
     System.out.println("Creating Queue connection...") ;
     queueConnection = queueConnectionFactory.createQueueConnection() ;
     System.out.print("Starting queueConnection" ;
     queueConnection.start() ;
     System.out.println("Getting queue session...") ;
     queueSession = queueConnection.createQueueSession(false,
                                    Session.AUTO_ACKNOWLEDGE);
     System.out.println("Creating queue receiver (i.e. a queue consumer)...") ;
     queueReceiver = queueSession.createReceiver(queue) ;
   }
   catch(  /* Exception handling not shown */ ) { /* etc. */ }
  }
  public void onMessage(Message msg)
  {
   try
   {
     CompanyLocalHome home = getCompanyLocalHome() ;
     if (msg instanceof ObjectMessage)
     {
       ObjectMessage objMessage = (ObjectMessage) msg ;
       ArrayList a = (ArrayList) objMessage.getObject() ;
       for (int i = 0 ; i < a.size() ; i++)
       {
         ShareChange sc = (ShareChange) a.get(i) ;
         CompanyLocal local =
             home.findByPrimaryKey(new Long(sc.getCompanyId())) ;
         double oldSharePrice = local.getShareprice().doubleValue() ;
         double high = local.getHigh().doubleValue() ;
         double low = local.getLow().doubleValue() ;
         double newSharePrice = oldSharePrice * (1 + sc.getPercentChange()/ 100.0) ;
         local.setShareprice(new Double(newSharePrice) );
         if (newSharePrice > high)          local.setHigh(new Double(newSharePrice) );
         if (newSharePrice < low)           local.setLow(new Double(newSharePrice) );
       }
     }
   }
   catch(  /* Exception handling not shown */ ) { /* etc. */ }   catch(FinderException e)
  }
  public void ejbRemove()
  {
   try   {
     queueReceiver.close();
     queueSession.close();
     queueConnection.stop();
   }
   catch(  /* Exception handling not shown */ ) { /* etc. */ }   catch(FinderException e)
  }
```

```
 public void setMessageDrivenContext(MessageDrivenContext ctx)
 {
  this.context = ctx;
 }
 private CompanyLocalHome getCompanyLocalHome() throws NamingException
 {
  final InitialContext context = new InitialContext();
  return   (CompanyLocalHome)
        context.lookup("java:comp/env/ejb/local/CompanyEJB");
 }
}

package sk.sharesapp.ejb.entity.cmp;
import            // imports not shown
```

public interface PlayerLocalHome extends EJBLocalHome

```
{
 PlayerLocal create() throws CreateException;
 PlayerLocal findByPrimaryKey(Long primaryKey) throws FinderException;
 Collection findAll() throws FinderException;
 PlayerLocal create(Long playerid, String codename, String password,
                    String lastname, String firstname, String department)
             throws CreateException;
 PlayerLocal findByCodeNamePassword(String codename, String password)
             throws FinderException;
 PlayerLocal findByCodeName(String codename) throws FinderException;
}

package sk.sharesapp.ejb.entity.cmp;
import            // imports not shown
```

public interface PlayerLocal extends EJBLocalObject

```
{
 Long getPlayerid();
 String getCodename();
 void setCodename(String codename);
 String getPassword();
 void setPassword(String password);
 String getLastname();
 void setLastname(String lastname);
 String getFirstname();
 void setFirstname(String firstname);
 String getDepartment();
 void setDepartment(String department);
 Collection getGameEJB_playerid();
 void setGameEJB_playerid(Collection gameEJB_playerid);
}

package sk.sharesapp.ejb.entity.cmp;
import            // imports not shown

public abstract class PlayerBean implements EntityBean
{
 private EntityContext context;

 public Long ejbCreate() {   return null;  }
 public void ejbPostCreate()  { }

 public Long ejbCreate(Long playerid, String codename, String password,
                       String lastname, String firstname, String department)
 {
  setPlayerid(playerid);
  setCodename(codename);
  setPassword(password);
  setLastname(lastname);
  setFirstname(firstname);
  setDepartment(department);
  return playerid;
 }
 public void ejbPostCreate(Long playerid, String codename, String password,
                       String lastname, String firstname, String department)
 { }
```

```
  public void ejbActivate()  { }
  public void ejbLoad()  { }
  public void ejbPassivate()  { }
  public void ejbRemove()  { }
  public void ejbStore()  { }
  public void setEntityContext(EntityContext ctx)
  {
   this.context = ctx;
  }
  public void unsetEntityContext()
  {
   this.context = null;
  }

  public abstract Long getPlayerid();
  public abstract void setPlayerid(Long playerid);
  public abstract String getCodename();
  public abstract void setCodename(String codename);
  public abstract String getPassword();
  public abstract void setPassword(String password);
  public abstract String getLastname();
  public abstract void setLastname(String lastname);
  public abstract String getFirstname();
  public abstract void setFirstname(String firstname);
  public abstract String getDepartment();
  public abstract void setDepartment(String department);
  public abstract Collection getGameEJB_playerid();
  public abstract void setGameEJB_playerid(Collection gameEJB_playerid);
}

package sk.sharesapp.utils;
import java.io.Serializable;
```

public class PlayerTO implements Serializable

```
{
  private long playerId;
  private String codeName;
  private String password;
  private String lastName;
  private String firstName;
  private String department;

  public PlayerTO()
  {
   this.playerId = 0L ;
   this.codeName = "" ;
   this.password = "" ;
   this.lastName = "" ;
   this.firstName = "" ;
   this.department = "" ;
  }
  public PlayerTO(String codeName, String password, String lastName,
                          String firstName, String department)
  {
     this.playerId = playerId ;
     this.codeName = codeName ;
     this.password = password ;
     this.lastName = lastName ;
     this.firstName = firstName ;
     this.department = department ;
  }
  public long getPlayerId()  {    return playerId;  }
  public void setPlayerId(long playerId)  {    this.playerId = playerId;  }

 // Other set / get methods not shown
}

package sk.sharesapp.ejb.entity.cmp;
import               // imports not shown
```

public class PlayerLocalDTO implements Serializable

```
{
  private Long playerid;
  private String codename;
  private String password;
  private String lastname;
  private String firstname;
  private String department;
  private Collection gameEJB_playeridDTO;
  public PlayerLocalDTO()
  {
  }
public PlayerLocalDTO(PlayerLocal playerLocal)
{
 if (playerLocal != null)
 {
   playerid = playerLocal.getPlayerid();
   codename = playerLocal.getCodename();
   password = playerLocal.getPassword();
   lastname = playerLocal.getLastname();
   firstname = playerLocal.getFirstname();
   department = playerLocal.getDepartment();
 }
}
public Long getPlayerid()
{
 return playerid;
}

// Other set / get methods not shown

 public Collection getGameEJB_playeridDTO()
{
 return gameEJB_playeridDTO;
}

private void _loadGameEJB_playeridDTO(Collection gameEJB_playerid)
{
 final int len = (gameEJB_playerid == null ? 0 : gameEJB_playerid.size());
 gameEJB_playeridDTO = new ArrayList(len);
 if (len > 0)
 {
   for (Iterator iter = gameEJB_playerid.iterator();iter.hasNext();)
   {
     gameEJB_playeridDTO.add(new GameLocalDTO((GameLocal)iter.next()));
   }
 }
}
public void setGameEJB_playeridDTO(Collection gameEJB_playeridDTO)
{
 this.gameEJB_playeridDTO = gameEJB_playeridDTO;
}
}
```

```
package sk.sharesapp.ejb.entity.cmp;
import          // imports not shown

public interface GameLocalHome extends EJBLocalHome
{
  GameLocal create() throws CreateException;
  GameLocal findByPrimaryKey(Long primaryKey) throws FinderException;
  Collection findAll() throws FinderException;
  GameLocal create(Long gameid) throws CreateException;
}

package sk.sharesapp.ejb.entity.cmp;
import          // imports not shown

public interface GameLocal extends EJBLocalObject
{
  Long getGameid();
  Double getBalance();
  void setBalance(Double balance);
  Timestamp getPeriod();
  void setPeriod(Timestamp period);
  PlayerLocal getPlayerEJB_playerid();
  void setPlayerEJB_playerid(PlayerLocal playerEJB_playerid);
}

package sk.sharesapp.ejb.entity.cmp;
import          // imports not shown

public abstract class GameBean implements EntityBean
{
  private EntityContext context;

  public Long ejbCreate() {   return null;  }
  public void ejbPostCreate()  { }
  public Long ejbCreate(Long gameid)
  {
    setGameid(gameid);
    return gameid;
  }
  public void ejbPostCreate(Long gameid)  { }
  public void ejbActivate()  { }
  public void ejbLoad()  { }
  public void ejbPassivate()  { }
  public void ejbRemove()  { }
  public void ejbStore()  { }
  public void setEntityContext(EntityContext ctx)
  {
     this.context = ctx;
  }
  public void unsetEntityContext()
  {
   this.context = null;
  }
  public abstract Long getGameid();
  public abstract void setGameid(Long gameid);
  public abstract Double getBalance();
  public abstract void setBalance(Double balance);
  public abstract Timestamp getPeriod();
  public abstract void setPeriod(Timestamp period);
  public abstract PlayerLocal getPlayerEJB_playerid();
  public abstract void setPlayerEJB_playerid(PlayerLocal playerEJB_playerid);
}

package sk.sharesapp.utils;
import              // imports not shown
```

```
public class GameTO implements Serializable, Comparable
{
  private long gameid;
  String codename;
  private double balance;
  private Timestamp period;
  public GameTO()  { }
  public GameTO(long gameId, String codename, double balance, Timestamp period)
  {
   this.gameId = gameId;
   this.codename = codename ;
   this.balance = balance;
   this.period = period;
  }
  public long getGameid()  {    return gameId;  }
  public void setGameid(long gameId)  {    this.gameId = gameid;  }

  // sort in descending order, needed as EJB QL has no "ORDER BY"
  public int compareTo(Object obj)
  {
   GameTO other = (GameTO) obj ;
    return (int) (other.balance - balance) ;
  }
}

package sk.sharesapp.ejb.entity.cmp;
import              // imports not shown
```

```
public class GameLocalDTO implements Serializable
{
  private Long gameid;
  private Double balance;
  private Timestamp period;
  private PlayerLocalDTO playerEJB_playeridDTO;

  public GameLocalDTO()  { }

  public GameLocalDTO(GameLocal gameLocal)
  {
   if (gameLocal != null)
   {
     gameid = gameLocal.getGameid();
     balance = gameLocal.getBalance();
     period = gameLocal.getPeriod();
   }
  }

  public Long getGameid()
  {
   return gameid;
  }

  // other set / get methods not shown
```

```
 public PlayerLocalDTO getPlayerEJB_playeridDTO()
 {
  return playerEJB_playeridDTO;
 }
 public void setPlayerEJB_playeridDTO(PlayerLocalDTO playerEJB_playeridDTO)
 {
  this.playerEJB_playeridDTO = playerEJB_playeridDTO;
 }
}

package sk.sharesapp.ejb.session.stateless;
import          // imports not shown

public interface PlayerGameSessionFacadeRemoteHome extends EJBHome
{
 PlayerGameSessionFacadeRemote create()
        throws RemoteException, CreateException;
}

package sk.sharesapp.ejb.session.stateless;
import          // imports not shown

public interface PlayerGameSessionFacadeRemote extends EJBObject
{
 Long addPlayer(PlayerLocalDTO dto) throws  CreateException,
   ServiceLocatorException, NamingException, RemoteException, DaoException;

 Long addPlayer(String codeName, String password, String lastName,
             String firstName, String department)
   throws CreateException, RemoteException, NamingException,
              FinderException,DaoException, ServiceLocatorException;
 void amendPlayer(PlayerLocalDTO dto) throws   ServiceLocatorException,
                    NamingException, FinderException, RemoteException;
 void deletePlayer(PlayerLocalDTO dto) throws  FinderException,
        RemoveException,  ServiceLocatorException, NamingException,
                    RemoteException;
 Long addGame(GameLocalDTO dto) throws RemoteException, CreateException,
    FinderException, NamingException, DaoException, ServiceLocatorException;
  Long addGame(Double balance, Timestamp period, PlayerLocal playerLocal)
    throws RemoteException, CreateException, FinderException,
               NamingException, DaoException, ServiceLocatorException;
  Long addGame(Double balance, Timestamp period, Long playerId)
       throws  DaoException,  FinderException,  NamingException,
             ServiceLocatorException, CreateException, RemoteException;
 void deleteGame(GameLocalDTO dto) throws RemoteException, FinderException,
      RemoveException, NamingException, ServiceLocatorException;
 PlayerLocalDTO getPlayer(String codeName, String password)
       throws  NamingException,  ServiceLocatorException, FinderException,
                  RemoteException;
 PlayerLocalDTO getPlayer(String codename) throws RemoteException,
      NamingException, FinderException, ServiceLocatorException;
 PlayerLocalDTO getPlayer(Long primaryKey) throws RemoteException,
      NamingException, FinderException, ServiceLocatorException;
 GameLocalDTO getGame(Long primaryKey) throws  NamingException,
       ServiceLocatorException, FinderException, RemoteException;
 PlayerLocalDTO getPlayerGames(Long primaryKey) throws  NamingException,
             ServiceLocatorException, FinderException, RemoteException;
 Collection getSortedGames() throws  NamingException,  ServiceLocatorException,
             FinderException, RemoteException;
}

package sk.sharesapp.ejb.session.stateless;
import          // imports not shown

publicblic class PlayerGameSessionFacadeEJBBean implements SessionBean
{
 public void ejbCreate()  { }
 public void ejbActivate()  { }
 public void ejbPassivate()  { }
 public void ejbRemove()  { }
 public void setSessionContext(SessionContext ctx)  { }
public Long addPlayer(String codeName, String password, String lastName,
                              String firstName, String department)
          throws NamingException, FinderException, DaoException,
                              CreateException, ServiceLocatorException
{
  PrimaryKeyGenerator pkGenerator = PrimaryKeyGenerator.getInstance() ;
  long id = pkGenerator.getNextPK() ;
  Long pk = new Long(id) ;
  PlayerLocalHome localHome = getPlayerLocalHome() ;
  // * test if codename already exists */
  PlayerLocal local = localHome.create(pk, codeName, password,
                   lastName, firstName, department ) ;
  return pk ;
}

public Long addPlayer(PlayerLocalDTO dto) throws  CreateException,
               ServiceLocatorException, NamingException, DaoException
{
  PrimaryKeyGenerator pkGenerator = PrimaryKeyGenerator.getInstance() ;
  long id = pkGenerator.getNextPK() ;
  Long pk = new Long(id) ;

  PlayerLocalHome localHome = getPlayerLocalHome() ;
  PlayerLocal local = localHome.create(pk, dto.getCodename(), dto.getPassword(),
                   dto.getLastname(), dto.getFirstname(), dto.getDepartment() ) ;
  return pk;
}

public void amendPlayer(PlayerLocalDTO dto) throws  FinderException,
                        ServiceLocatorException, NamingException
{
  PlayerLocalHome localHome = getPlayerLocalHome() ;
  PlayerLocal local = localHome.findByPrimaryKey(dto.getPlayerid()) ;
  local.setCodename(dto.getCodename() );
  local.setPassword(dto.getPassword() );
  local.setLastname(dto.getLastname() );
  local.setFirstname(dto.getFirstname() );
  local.setDepartment(dto.getDepartment() );
}
```

```
public void deletePlayer(PlayerLocalDTO dto) throws  FinderException,
        RemoveException,  ServiceLocatorException, NamingException
 {
  PlayerLocalHome localHome = getPlayerLocalHome() ;
  PlayerLocal local = localHome.findByPrimaryKey(dto.getPlayerid() ) ;
  local.remove();
 }

 public PlayerLocalDTO getPlayer(String codeName, String password) throws
                NamingException,  FinderException, ServiceLocatorException
 {
  PlayerLocalHome localHome = getPlayerLocalHome() ;
  PlayerLocal local = localHome.findByCodeNamePassword(codeName, password) ;
  PlayerLocalDTO dto = new PlayerLocalDTO() ;
  dto.setPlayerid(local.getPlayerid() ) ;
  dto.setCodename(local.getCodename());
  dto.setPassword(local.getPassword());
  dto.setLastname(local.getLastname());
  dto.setFirstname(local.getFirstname());
  dto.setDepartment(local.getDepartment());
  // Not necessary to return games hanging from this player
  return dto ;
 }
 public PlayerLocalDTO getPlayer(String codeName) throws FinderException,
                NamingException, ServiceLocatorException
 {
  PlayerLocalHome localHome = getPlayerLocalHome() ;
  PlayerLocal local = localHome.findByCodeName(codeName) ;
  PlayerLocalDTO dto = new PlayerLocalDTO() ;
  dto.setPlayerid(local.getPlayerid() ) ;
  dto.setCodename(local.getCodename());
  dto.setPassword(local.getPassword());
  dto.setLastname(local.getLastname());
  dto.setFirstname(local.getFirstname());
  dto.setDepartment(local.getDepartment());

  // Not necessary to return games hanging from this player
  return dto;
 }
```

```
public PlayerLocalDTO getPlayer(Long primaryKey) throws NamingException,
            FinderException, ServiceLocatorException
{
 PlayerLocalHome localHome = getPlayerLocalHome() ;
 PlayerLocal local = localHome.findByPrimaryKey(primaryKey) ;
 PlayerLocalDTO dto = new PlayerLocalDTO() ;
 dto.setPlayerid(local.getPlayerid() ) ;
 dto.setCodename(local.getCodename());
 dto.setPassword(local.getPassword());
 dto.setLastname(local.getLastname());
 dto.setFirstname(local.getFirstname());
 dto.setDepartment(local.getDepartment());
 return dto;
}
public PlayerLocalDTO getPlayerGames(Long primaryKey) throws
      NamingException,  FinderException, ServiceLocatorException
{
 PlayerLocalHome localHome = getPlayerLocalHome() ;
 PlayerLocal local = localHome.findByPrimaryKey(primaryKey) ;
 PlayerLocalDTO dto = new PlayerLocalDTO() ;
 dto.setPlayerid(local.getPlayerid() ) ;
 dto.setCodename(local.getCodename());
 dto.setPassword(local.getPassword());
 dto.setLastname(local.getLastname());
 dto.setFirstname(local.getFirstname());
 dto.setDepartment(local.getDepartment());
 //  Now hang games onto player
 Collection colLocal = local.getGameEJB_playerid() ;
 ArrayList a = new ArrayList() ;
 Iterator it = colLocal.iterator() ;   int i = 0 ;
 while (it.hasNext() )   {
  GameLocal gameLocal = (GameLocal) it.next() ;
  GameLocalDTO gameDTO = new GameLocalDTO() ;
  gameDTO.setGameid(gameLocal.getGameid());
  gameDTO.setBalance(gameLocal.getBalance()) ;
  gameDTO.setPeriod(gameLocal.getPeriod());
  a.add(gameDTO) ;
 }
 dto.setGameEJB_playeridDTO(a);
 return dto;
}
```

```
public GameLocalDTO getGame(Long gamePK) throws  NamingException,
            FinderException, ServiceLocatorException
{
 GameLocalHome gameLocalHome = getGameLocalHome() ;
 GameLocal gameLocal = gameLocalHome.findByPrimaryKey(gamePK) ;

 GameLocalDTO gameDTO = new GameLocalDTO() ;
 gameDTO.setGameid(gameLocal.getGameid());
 gameDTO.setBalance(gameLocal.getBalance());
 gameDTO.setPeriod(gameLocal.getPeriod());

 // Now find player for this game
 PlayerLocal playerLocal = gameLocal.getPlayerEJB_playerid() ;
 // Transfer player details to DTO
 PlayerLocalDTO playerDTO = new PlayerLocalDTO() ;
 playerDTO.setPlayerid(playerLocal.getPlayerid());
 playerDTO.setCodename(playerLocal.getCodename());
 playerDTO.setPassword(playerLocal.getPassword());
 playerDTO.setLastname(playerLocal.getLastname());
 playerDTO.setFirstname(playerLocal.getFirstname());
 playerDTO.setDepartment(playerLocal.getDepartment());

 gameDTO.setPlayerEJB_playeridDTO(playerDTO);

 return gameDTO ;
}

public Collection getSortedGames() throws  FinderException, NamingException,
                  ServiceLocatorException
{
 GameLocalHome gameLocalHome = getGameLocalHome() ;
 Collection col = gameLocalHome.findAll() ;
 ArrayList games = new ArrayList() ;
 Iterator it = col.iterator() ;
 while (it.hasNext() )
 {
  GameLocal gameLocal = (GameLocal) it.next() ;
  PlayerLocal playerLocal = gameLocal.getPlayerEJB_playerid() ;
  GameTO to = new GameTO() ;
  to.setGameid(gameLocal.getGameid().longValue());
  to.setCodename(playerLocal.getCodename());
```

```
    to.setBalance(gameLocal.getBalance().doubleValue());
    to.setPeriod(gameLocal.getPeriod()) ;
    games.add(to) ;
  }
  Collections.sort(games);  // No "ORDER BY" in EJB QL
  return games ;
}

public Long addGame(GameLocalDTO dto) throws NamingException,
    CreateException, FinderException,  DaoException, ServiceLocatorException
{
  PrimaryKeyGenerator pkGenerator = PrimaryKeyGenerator.getInstance() ;
  long id = pkGenerator.getNextPK() ;
  Long pk = new Long(id) ;

  GameLocalHome gameLocalHome = getGameLocalHome() ;
  GameLocal gameLocal = gameLocalHome.create(pk) ;
  gameLocal.setBalance(dto.getBalance());
  gameLocal.setPeriod(dto.getPeriod());
  // Find player associated with this game
  PlayerLocalHome playerHome = (PlayerLocalHome)getPlayerLocalHome() ;
  PlayerLocal playerLocal = (PlayerLocal)
    playerHome.findByPrimaryKey(dto.getPlayerEJB_playeridDTO().getPlayerid()) ;

  gameLocal.setPlayerEJB_playerid(playerLocal);

  return pk;
}

public Long addGame(Double balance, Timestamp period, PlayerLocal playerLocal)
                    throws NamingException, CreateException,
                                    DaoException, ServiceLocatorException
{
  PrimaryKeyGenerator pkGenerator = PrimaryKeyGenerator.getInstance() ;
  long id = pkGenerator.getNextPK() ;
  Long pk = new Long(id) ;

  GameLocalHome gameLocalHome = getGameLocalHome() ;
  GameLocal gameLocal = gameLocalHome.create(pk) ;
  gameLocal.setBalance(balance);
  gameLocal.setPeriod(period);
```

```
  gameLocal.setPlayerEJB_playerid(playerLocal);

  return pk;
}

public Long addGame(Double balance, Timestamp period, Long playerId)
        throws  CreateException, DaoException,  FinderException,
                        NamingException,  ServiceLocatorException
{
  PrimaryKeyGenerator pkGenerator = PrimaryKeyGenerator.getInstance() ;
  long id = pkGenerator.getNextPK() ;
  Long pk = new Long(id) ;

  GameLocalHome gameLocalHome = getGameLocalHome() ;
  GameLocal gameLocal = gameLocalHome.create(pk) ;
  gameLocal.setBalance(balance);
  gameLocal.setPeriod(period);

  // Find player associated with this game
  PlayerLocalHome playerHome = (PlayerLocalHome)getPlayerLocalHome() ;
  PlayerLocal playerLocal = (PlayerLocal)
    playerHome.findByPrimaryKey(playerId) ;

  gameLocal.setPlayerEJB_playerid(playerLocal);

  return pk;
}

public void deleteGame(GameLocalDTO dto) throws NamingException,
        FinderException, RemoveException, ServiceLocatorException
{
  GameLocalHome localHome = getGameLocalHome() ;
  GameLocal local = localHome.findByPrimaryKey(dto.getGameid() ) ;

  local.remove();;
}
```

```
  private PlayerLocalHome getPlayerLocalHome()
         throws NamingException, ServiceLocatorException
 {
  //final InitialContext context = new InitialContext();
  //return (PlayerLocalHome)context.lookup("java:comp/env/ejb/local/PlayerEJB");

  ServiceLocator serviceLocator = ServiceLocator.getInstance() ;
  PlayerLocalHome localHome =
         (PlayerLocalHome) serviceLocator.getLocalHome("PlayerEJB") ;
  return localHome ;
 }

 private GameLocalHome getGameLocalHome()
         throws NamingException, ServiceLocatorException
 {
  ServiceLocator serviceLocator = ServiceLocator.getInstance() ;
  GameLocalHome localHome =
                 (GameLocalHome) serviceLocator.getLocalHome("GameEJB") ;
  return localHome ;
 }
}

package sk.sharesapp.ejb.session.stateless;
import          // imports not shown

public class PlayerGameSessionFacadeRemoteClient
{
   PlayerGameSessionFacadeRemote playerGameSessionFacadeRemote = null;

   public PlayerGameSessionFacadeRemoteClient()
         throws RemoteException, NamingException, CreateException
   {
    Context context = getInitialContext();
    PlayerGameSessionFacadeRemoteHome
        playerGameSessionFacadeRemoteHome =
          (PlayerGameSessionFacadeRemoteHome)PortableRemoteObject.narrow
          (context.lookup("PlayerGameSessionFacadeEJB"),
                              PlayerGameSessionFacadeRemoteHome.class);
    playerGameSessionFacadeRemote =
                              playerGameSessionFacadeRemoteHome.create();
 }

public void testPlayer() throws RemoteException, CreateException,
    ServiceLocatorException,  RemoveException, DaoException,
                                  FinderException, NamingException
 {
  System.out.println("addPlayer( abcdef, abcdef, Smith, John, Science" );
  // Long pk = playerGameSessionFacadeRemote.addPlayer( "abcdef", "abcdef",
  //                  "Smith", "John", "Science" );
  PlayerLocalDTO to = new PlayerLocalDTO() ;
  to.setPlayerid(new Long(111));  // Will be auto generated, so value irrevalent
  to.setCodename("abcdef");
  to.setPassword("abcdef");
  to.setLastname("Smith");
  to.setFirstname("John");
  to.setDepartment("Science");
  Long pk = playerGameSessionFacadeRemote.addPlayer(to) ;

  System.out.println("Now find by primary key " + pk );
  PlayerLocalDTO dto = playerGameSessionFacadeRemote.getPlayer(pk) ;
  display(dto) ;
  System.out.println("Now find by codename, password: abcdef abcdef");
  dto = playerGameSessionFacadeRemote.getPlayer("abcdef", "abcdef") ;
  display(dto) ;
  System.out.println("Now find by codename abcdef");
  dto = playerGameSessionFacadeRemote.getPlayer(pk) ;
  display(dto) ;
  System.out.println("Now amend, ghijkl ghijkl jones ann eng");
  dto.setCodename("ghijkl") ;   dto.setPassword("ghijkl");
  dto.setLastname("jones");    dto.setFirstname("ann");
  dto.setDepartment("eng");
  playerGameSessionFacadeRemote.amendPlayer(dto);
  display(dto) ;
  System.out.println("Now find by primary key " + pk );
  dto = playerGameSessionFacadeRemote.getPlayer(pk) ;
  display(dto) ;
  System.out.println("Now delete " );
  playerGameSessionFacadeRemote.deletePlayer(dto);
  System.out.println("Now attempt to find by primary key " + pk );
  dto = playerGameSessionFacadeRemote.getPlayer(pk) ;
 }
```

```
public void testPlayerGames() throws RemoteException, CreateException,
        ServiceLocatorException, RemoveException, DaoException,
                                        FinderException, NamingException
 {
  System.out.println("Try getPlayer() first" );
  PlayerLocalDTO dto =
        playerGameSessionFacadeRemote.getPlayer(new Long(10001)) ;
  display(dto) ;
  System.out.println("Now try getPlayerGames()" );
  dto = playerGameSessionFacadeRemote.getPlayerGames(new Long(10001)) ;
  System.out.println("and display" );
  displayPlayerGames(dto) ;
 }

public void testGame() throws RemoteException, CreateException,
        ServiceLocatorException,  RemoveException, DaoException,
                                        FinderException, NamingException
 {
  GregorianCalendar gc = new GregorianCalendar() ;
  long millisecs = gc.getTimeInMillis() ;
  ArrayList a = new ArrayList() ;
  System.out.println("addPlayer( abcdef, abcdef, Smith, John, Science" );
  PlayerLocalDTO playerDTO = new PlayerLocalDTO() ;
  playerDTO.setPlayerid(new Long(111)); // auto generated, so value irrevalent
  playerDTO.setCodename("abcdef");
  playerDTO.setPassword("abcdef");
  playerDTO.setLastname("Smith");
  playerDTO.setFirstname("John");
  playerDTO.setDepartment("Science");
  Long playerPK = playerGameSessionFacadeRemote.addPlayer(playerDTO) ;
  playerDTO.setPlayerid(playerPK); // ensure TO has correct pk, needed below
  System.out.println("Now add game for this player" );
  GameLocalDTO gameDTO1 = new GameLocalDTO() ;
  gameDTO1.setGameid(new Long(1)) ; //  pk generated automatically
  gameDTO1.setBalance((new Double(1000)) );
  gameDTO1.setPeriod(new Timestamp(millisecs));

  gameDTO1.setPlayerEJB_playeridDTO(playerDTO);

  Long gamePK = playerGameSessionFacadeRemote.addGame(gameDTO1) ;
  System.out.println("PK for this game: " + gamePK + " now get this game") ;
  gameDTO1 = playerGameSessionFacadeRemote.getGame(gamePK) ;
  display(gameDTO1) ;
  System.out.println("Adding game " + gameDTO1.getGameid() + " to ArrayList a") ;
  a.add(gameDTO1) ;

  System.out.println("Now add another game for this player" );
  GameLocalDTO gameDTO2 = new GameLocalDTO() ;
  gameDTO2.setGameid(new Long(1)) ; //  pk generated automatically
  gameDTO2.setBalance((new Double(2000)) );
  gameDTO2.setPeriod(new Timestamp(millisecs));
  gameDTO2.setPlayerEJB_playeridDTO(playerDTO);
  gamePK = playerGameSessionFacadeRemote.addGame(gameDTO2) ;
  System.out.println("PK for this game: " + gamePK + " now get this game") ;
  gameDTO2 = playerGameSessionFacadeRemote.getGame(gamePK) ;
  display(gameDTO2) ;
  System.out.println("Adding game " + gameDTO1.getGameid() + " to ArrayList a") ;
  a.add(gameDTO2) ;

  System.out.println("Now add another game for this player" );
  GameLocalDTO gameDTO3 = new GameLocalDTO() ;
  gameDTO3.setGameid(new Long(1)) ;  pk generated automatically
  gameDTO3.setBalance((new Double(2000)) );
  gameDTO3.setPeriod(new Timestamp(millisecs));
  gameDTO3.setPlayerEJB_playeridDTO(playerDTO);
  gamePK = playerGameSessionFacadeRemote.addGame(gameDTO3) ;
  System.out.println("PK for this game: " + gamePK + " now get this game") ;
  gameDTO3 = playerGameSessionFacadeRemote.getGame(gamePK) ;
  display(gameDTO3) ;
  System.out.println("Adding game " + gameDTO1.getGameid() + " to ArrayList a") ;
  a.add(gameDTO3) ;

  System.out.println("Add the Arraylist of games to player") ;
  playerDTO.setGameEJB_playeridDTO(a);
  System.out.println("Now display player's games") ;
  Collection col = playerDTO.getGameEJB_playeridDTO() ;
  if (col == null)
    System.out.println("col is null") ;
  else
  {
   Iterator it = col.iterator() ;
   while (it.hasNext() )
   {
    GameLocalDTO game = (GameLocalDTO) it.next() ;
    display(game) ;
   }
  }
  System.out.println("Now perform cascade delete") ;
  playerGameSessionFacadeRemote.deletePlayer(playerDTO);

  System.out.println("Now attempt to get player " + playerPK) ;
  playerGameSessionFacadeRemote.getPlayer(playerPK) ;
}
public void display(PlayerLocalDTO dto)
{
  System.out.println(dto.getPlayerid() + "\t" + dto.getCodename() + "\t" +
                   dto.getPassword() + "\t" + dto.getLastname() + "\t" +
                   dto.getFirstname() + "\t" + dto.getDepartment() ) ;
}
public void displayPlayerGames(PlayerLocalDTO dto)
{
  System.out.println(dto.getPlayerid() + "\t" + dto.getCodename() + "\t" +
                   dto.getPassword() + "\t" + dto.getLastname() + "\t" +
                   dto.getFirstname() + "\t" + dto.getDepartment() ) ;
  Collection col = dto.getGameEJB_playeridDTO() ;
  if (col == null || col.size() == 0)
   System.out.println("No games") ;
  else
  {
   Iterator it = col.iterator() ;
   while (it.hasNext() )
   {
    GameLocalDTO gameDTO = (GameLocalDTO) it.next() ;
    System.out.print("Game details: ") ;
    System.out.println(gameDTO.getGameid() + "\t" +
                   gameDTO.getBalance() + "\t" + gameDTO.getPeriod() ) ;
   }
  }
}
```

```
public void display(GameLocalDTO dto)
 {
  System.out.print("Game details: ") ;
  System.out.println(dto.getGameid() +"\t"+ dto.getBalance() +"\t"+ dto.getPeriod() ) ;
  System.out.print("Played by: ") ;
  display(dto.getPlayerEJB_playeridDTO() ) ;
 }
 public static void testTimestamp()
 {
  GameLocalDTO dto = new GameLocalDTO() ;
  dto.setGameid(new Long(1000));
  dto.setBalance(new Double(100));
  GregorianCalendar gc = new GregorianCalendar() ;
  long millisecs = gc.getTimeInMillis() ;
  dto.setPeriod(new Timestamp(millisecs));
  System.out.println(dto.getGameid() + "\t" +
                   dto.getBalance() + "\t" + dto.getPeriod() ) ;
 }
 public static void main(String [] args)
 {
  try
  {
   PlayerGameSessionFacadeRemoteClient client =
              new PlayerGameSessionFacadeRemoteClient() ;
   //client.testPlayer();
   client.testGame();
   //client.testPlayerGames();
   //client.testTimestamp() ;
  }
  catch(Exception e)
  {
   System.out.println(e.getMessage() ) ;
  }
 }
private static Context getInitialContext() throws NamingException
 {
  return new InitialContext();
 }
}
```

```
package sk.sharesapp.ejb.session.stateful;
import              // imports not shown

public interface PortfolioRemoteHome extends EJBHome
{
 PortfolioRemote create() throws RemoteException, CreateException;
 PortfolioRemote create(Long playerid, double balance)
                                  throws RemoteException, CreateException;
}

package sk.sharesapp.ejb.session.stateful;
import             // imports not shown

public interface PortfolioRemote extends EJBObject
{
 double getBalance() throws RemoteException;
 void setBalance(double balance) throws RemoteException;
 void purchaseShares(FolioTO folio)
          throws RemoteException, StockExchangeException;
 void sellShares(int folioId, long quantity) throws RemoteException,
          StockExchangeException, FinderException, ServiceLocatorException;
 void sellAllShares() throws RemoteException, CreateException, FinderException,
                   StockExchangeException, ServiceLocatorException, DaoException;
 Long getPlayerId() throws RemoteException;
 void setPlayerId(Long playerId) throws RemoteException;
 public Collection getFolios() throws RemoteException;
}

package sk.sharesapp.ejb.session.stateful;
import             // imports not shown;

public class PortfolioBean implements SessionBean
{
 public Long playerId;
 public ArrayList folios;
 public double balance;
 private SessionContext context;
```

```
 public void ejbCreate()
{
 playerId = new Long(0L) ;
 balance = 50000 ;              // Default balance
 folios = new ArrayList() ;
}
public void ejbCreate(Long playerid, double balance)
{
 this.playerId = playerId ;
 this.balance = balance ;
 folios = new ArrayList() ;
}
public void ejbActivate()  { }
public void ejbPassivate()  { }
public void ejbRemove()  { }
public void setSessionContext(SessionContext ctx)
{
 this.context = ctx;
}
public double getBalance()  {
 return balance;
}
public void setBalance(double balance)  {
 this.balance = balance;
}
public Collection getFolios()
{
 return folios;
}
public void setFolios(ArrayList folios)
{
 this.folios = folios;
}
public void purchaseShares(FolioTO folio) throws StockExchangeException
{
 double cost = folio.getPurchasePrice() * folio.getQuantity() ;
 double newBalance = balance - cost ;
  if (newBalance < 0)
   throw new StockExchangeException("Insufficient funds.  Balance "
                                            + balance + " Cost " + cost) ;
  folios.add(folio) ;
  balance = newBalance ;
  System.out.println("After purchase shares") ;
  showPortfolio() ;
}
public void sellShares(int folioId, long quantity) throws FinderException,
                                StockExchangeException, ServiceLocatorException
{
  boolean found = false ;
  for (int i = 0 ; i < folios.size() ; i++)
  {
    FolioTO folio = (FolioTO) folios.get(i) ;
    if (folioId == folio.getFolioId() )
    {
      found = true ;
      long sharesRemaining = folio.getQuantity() - quantity ;
      if (sharesRemaining < 0)
      {
        throw new StockExchangeException("ERROR! Insufficient shares, owns " +
                            folio.getQuantity() + " attempting to sell " + quantity) ;
      }
      String symbol = folio.getSymbol() ;
      CompanyLocalHome localHome = getCompanyLocalHome() ;
      CompanyLocal local =
          localHome.findByPrimaryKey(new Long(folio.getCompanyId())) ;
      double sharePrice = local.getShareprice().doubleValue() ;
      if (sharesRemaining > 0)
      {
        folio.setQuantity(sharesRemaining) ;  folio.display() ;
      }
      else
      {
        folios.remove(i) ;
      }
      balance += sharePrice * quantity ;
    }
  }
  if (found == false)
    throw new StockExchangeException("This folio does not exist") ;
  System.out.println("After selling shares") ;   showPortfolio() ;
}
```

```
public void sellAllShares() throws CreateException, DaoException, FinderException,
                                   StockExchangeException, ServiceLocatorException
{
  for (int i = 0 ; i < folios.size()  ; i++)
  {
    FolioTO folio = (FolioTO) folios.get(i) ;
    String symbol = folio.getSymbol() ;
    CompanyLocalHome companyLocalHome = getCompanyLocalHome() ;
    CompanyLocal companyLocal =
          companyLocalHome.findByPrimaryKey(new Long(folio.getCompanyId())) ;
    double sharePrice = companyLocal.getShareprice().doubleValue() ;
    System.out.println("xxxShareprice " + sharePrice) ;
    balance += sharePrice * folio.getQuantity() ;
  }
  PrimaryKeyGenerator pkGenerator = PrimaryKeyGenerator.getInstance() ;
  long id = pkGenerator.getNextPK() ;
  Long pk = new Long(id) ;
  GameLocalHome gameLocalHome = getGameLocalHome() ;
  GameLocal gameLocal = gameLocalHome.create(pk) ;
  GregorianCalendar gc = new GregorianCalendar() ;
  Timestamp period = new Timestamp(gc.getTimeInMillis()) ;
  gameLocal.setBalance(new Double(balance) );
  gameLocal.setPeriod(period);
  PlayerLocalHome playerHome = (PlayerLocalHome) getPlayerLocalHome() ;
  PlayerLocal playerLocal = (PlayerLocal) playerHome.findByPrimaryKey(playerId) ;
  gameLocal.setPlayerEJB_playerid(playerLocal);
  folios.clear();
  System.out.println("After selling all shares") ;
  showPortfolio() ;
}
private void showPortfolio()
{
  System.out.println(playerId + "'s portfolio " + "Balance " + balance ) ;
  for (int i = 0 ; i < folios.size() ; i++)    {
    FolioTO folio = (FolioTO) folios.get(i) ;
    System.out.println(folio.getFolioId() + "\t" + folio.getCompanyId() + "\t" +
                  folio.getSymbol() + "\t" + + folio.getPurchasePrice() + "\t" +
                  folio.getQuantity()) ;
  }
}
```

```
 private CompanyLocalHome getCompanyLocalHome()
         throws ServiceLocatorException
 {
  ServiceLocator serviceLocator = ServiceLocator.getInstance() ;
  CompanyLocalHome localHome =
         (CompanyLocalHome) serviceLocator.getLocalHome("CompanyEJB") ;
  return localHome ;
 }

 private GameLocalHome getGameLocalHome() throws ServiceLocatorException
 {
  ServiceLocator serviceLocator = ServiceLocator.getInstance() ;
  GameLocalHome localHome =
         (GameLocalHome) serviceLocator.getLocalHome("GameEJB") ;
  return localHome ;
 }

 private PlayerLocalHome getPlayerLocalHome() throws ServiceLocatorException
 {
  ServiceLocator serviceLocator = ServiceLocator.getInstance() ;
  PlayerLocalHome localHome =
                (PlayerLocalHome) serviceLocator.getLocalHome("PlayerEJB") ;
  return localHome ;
 }
 public Long getPlayerId()
 {
  return playerid;
 }
 public void setPlayerId(Long playerId)
 {
  this.playerid = playerid;
 }
}

package sk.sharesapp.utils;
import java.io.Serializable;

public class FolioTO implements Serializable
{
 static long nextId = 10001L ;
 private long folioId ;
 private long companyId ;
 private String symbol ;
 private double purchasePrice ;
 private long quantity ;

 public FolioTO()
 {
  folioId = nextId++ ;
  companyId = 0L ;   symbol = "" ;
  purchasePrice = 0.0 ;   quantity = 0 ;
 }
 public FolioTO(long companyId, String symbol, double purchasePrice, long quantity)
 {
  this.folioId = nextId++ ;
  this.companyId = companyId ;
  this.symbol = symbol ;
  this.purchasePrice = purchasePrice ;
  this.quantity = quantity ;
 }

 //  get / set methods not shown

 public void display()
 {
    System.out.println(getCompanyId() + "\t" + getSymbol() + "\t" +
              getPurchasePrice() + "\t" + getQuantity()) ;
 }
 public static void main(String[] args)
 {
    FolioTO folio = new FolioTO() ;
    folio.display() ;
    folio.setCompanyId(10001);
    folio.setSymbol("AIB") ;
    folio.setPurchasePrice(200.5) ;
    folio.setQuantity(10) ;
    folio.display() ;
    FolioTO folio2 = new FolioTO(10001, "Iona", 78.9, 200) ;
    folio2.display() ;
 }
}
```

```
package sk.sharesapp.ejb.session.stateful;
import              // imports not shown

public class PortfolioRemoteClient
{
  public static void main(String [] args)
  {
    PortfolioRemoteClient portfolioRemoteClient = new PortfolioRemoteClient();
    try
    {
      Context context = getInitialContext();
/*    PortfolioRemoteHome portfolioRemoteHome =
          (PortfolioRemoteHome)PortableRemoteObject.narrow
             (context.lookup("PortfolioEJB"), PortfolioRemoteHome.class);
      PortfolioRemote portfolioRemote;

      Long playerid = new Long(10001) ;
      portfolioRemote = portfolioRemoteHome.create(playerId, 50000);
      portfolioRemote.setPlayerId(playerId);
      FolioTO folio = new FolioTO(10001, "AIB", 18.2, 100) ;
      long id1 = folio.getFolioId() ;
      portfolioRemote.purchaseShares(folio);
      folio = new FolioTO(10001, "AIB", 18.2, 200) ;
      long id2 = folio.getFolioId() ;
      portfolioRemote.purchaseShares(folio);
      folio = new FolioTO(10002, "Iona", 2.7, 100) ;
      long id3 = folio.getFolioId() ;
      portfolioRemote.purchaseShares(folio);
      folio = new FolioTO(10002, "AIB", 2.7, 200) ;
      long id4 = folio.getFolioId() ;
      portfolioRemote.purchaseShares(folio);
      //Testing for overflow
      //folio = new FolioTO(10001, "AIB", 18.2, 5000) ;
      //Long id5 = folio.getFolioId() ;
      //portfolioRemote.purchaseShares(folio);

      //portfolioRemote.sellShares(id1, 100);
      //portfolioRemote.sellShares(id1, 50);
      //portfolioRemote.sellShares(id1, 200);
      portfolioRemote.sellAllShares();
*/
      PlayerGameSessionFacadeRemoteHome
         playerGameSessionFacadeRemoteHome =
            (PlayerGameSessionFacadeRemoteHome)PortableRemoteObject.narrow
                 (context.lookup("PlayerGameSessionFacadeEJB"),
                              PlayerGameSessionFacadeRemoteHome.class);
      PlayerGameSessionFacadeRemote playerGameSessionFacadeRemote =
                              playerGameSessionFacadeRemoteHome.create();
      Collection col = playerGameSessionFacadeRemote.getSortedGames() ;
      Iterator it = col.iterator() ;
      while (it.hasNext() )
      {
        GameTO to = (GameTO) it.next() ;
        System.out.println(to.getGameid() + "\t"+ to.getBalance() +"\t" + to.getPeriod() ) ;
      }
    }
    catch( /*  Exception handling not shown */ ( { /* etc. */
  }

  private static Context getInitialContext() throws NamingException
  {
    return new InitialContext();
  }
}

package sk.controller.servlet ;
import              // imports not shown;

public class FrontController extends HttpServlet
{
  public void doGet(HttpServletRequest request, HttpServletResponse response)
                 throws IOException, ServletException
  {
    Command commandHandler = null ;
    String command = request.getParameter("command") ;
     HttpSession session = request.getSession(true) ;
    String state = (String) session.getAttribute("state") ;
if (command == null || state == null)              // First time controller is called
{
  command = "Home" ;
  state = "NotRegistered" ;
  session.setAttribute("state", state) ;
  Locale locale = request.getLocale() ;
  ResourceBundle messages ;
  String header = request.getHeader("User-Agent") ;
  if (header.indexOf("Mozilla") == -1)        // use wireless bundle
  {
    session.setAttribute("device", "wireless") ;
   messages = ResourceBundle.getBundle
                    ("sk.sharesapp.resources.w_MyResources", locale) ;
  }
  else
  {
    session.setAttribute("device", "pc") ;
   messages = ResourceBundle.getBundle
                    ("sk.sharesapp.resources.MyResources", locale) ;
  }
  session.setAttribute("messages", messages);
}
ResourceBundle messages =
                          (ResourceBundle) session.getAttribute("messages") ;
// Check if ValueList handler needed
if ( !(command.equals("ViewLeagueTable")
                              || command.equals("ViewSectionTable")) )
{
  try
  {
    ValueListHandlerSessionFacadeRemote remoteListHandler =
           (ValueListHandlerSessionFacadeRemote)
                              session.getAttribute("remoteListHandler") ;
    if (remoteListHandler != null)
    {
      remoteListHandler.remove();
      session.removeAttribute("remoteListHandler") ;
    }
  }
  catch(RemoveException e)
  {
    throw new ServletException("FrontController " + e.getMessage() ) ;
  }
}
CommandFactory factory = CommandFactory.getInstance() ;
try
{
  commandHandler = factory.createCommand(command) ;
}
catch(CommandCreationException e)
{
  throw new ServletException("FrontController " + e.getMessage() ) ;
}
if (commandHandler == null)
  throw new ServletException("Illegal command handler " + command) ;
else
{
  String page = commandHandler.execute(request, response) ;
  String device = (String) session.getAttribute("device") ;
   if (device.equals("wireless") )
     page = "w_" + page ;        // different JSP for wireless devices
  gotoPage(page, request, response) ;
}
return ;
}
  public void doPost(HttpServletRequest request, HttpServletResponse response)
                 throws IOException, ServletException   {
                 doGet(request, response) ;
  }
  public void gotoPage(String page, HttpServletRequest request,
              HttpServletResponse response) throws IOException, ServletException
  {
       Token.saveToken(request) ; // To prevent duplicate submissions
       page = "/WEB-INF/private/" + page ;
       String encodedURL = response.encodeURL(page) ;
       RequestDispatcher dispatcher =
                 getServletContext().getRequestDispatcher(encodedURL) ;
       dispatcher.forward(request, response) ;
  }
}
```

```
package sk.sharesapp.utils ;
import              // imports not shown

public class Token      // Used to prevent duplicate submissions
{
 public Token()
 {
 }
 public static void saveToken(HttpServletRequest request)
 {
  HttpSession session = request.getSession() ;
  String token = generateToken(request) ;
  if (token != null )
  {
   session.setAttribute("token", token) ;
  }
 }
 private static String generateToken(HttpServletRequest request)
 {
  HttpSession session = request.getSession() ;
  GregorianCalendar d = new GregorianCalendar() ;
  long t = d.getTimeInMillis() ;              // generate string based on time
  String token = session.getId() + t ;       // and on session id
  return token ;
 }
 public static boolean isTokenValid(HttpServletRequest request)
 {
  HttpSession session = request.getSession(false) ;
  if (session == null)
   return (false) ;
  String saved = (String) session.getAttribute("token") ;
  if (saved == null)
   return (false) ;
  String token = request.getParameter("token") ;
  if (token == null)
   return (false) ;
  return (saved.equals(token)) ;
 }
}
```

```
package sk.controller.command ;
import              // imports not shown

public interface Command
{
     String execute(HttpServletRequest request, HttpServletResponse response) ;
}
```

```
package sk.controller.command ;
import              // imports not shown

public class CommandFactory
{
     private static final String packageName = "sk.controller.command." ;
     private static CommandFactory factory = null ;
     private CommandFactory()
     {
     }
     public synchronized static CommandFactory getInstance()
     {
          if (factory == null)                                        // first time
                    factory = new CommandFactory() ;
          return factory ;
     }
  public synchronized Command createCommand(String command)
                                          throws CommandCreationException
  {
   Command commandHandler = null ;
   try
   {
    command = packageName + command + "Command" ;
    Class theClass = Class.forName(command) ;
    Object theObject = theClass.newInstance() ;
     commandHandler = (Command) theObject ;
   }
   catch (  /* Exception handling not shown  */

   return commandHandler ;
  }
}
```

```
package sk.controller.command ;
import              // imports not shown

public class BuySharesCommand implements Command
{
 public BuySharesCommand()
 {
 }
 public String execute(HttpServletRequest request, HttpServletResponse response)
 {
  String page = null ;
  try
  {
   HttpSession session = request.getSession(true) ;
   String state = (String) session.getAttribute("state") ;
   if (!state.equals("Registered"))
   {
    request.setAttribute("errorMessage", NOT_LOGGED_IN) ;
    page = errorPage ;
   }
   else
   {
    String symbol = request.getParameter("symbol") ;
    String quantity = request.getParameter("quantity") ;
    if (symbol == null || symbol.length() == 0 ||
                                   quantity == null || quantity.length() == 0)
    {
     page = "BuyShares.jsp" ;
    }
    else
    {
     if (Token.isTokenValid(request) )  // avoid duplicate submission
     {
      long numShares = Long.parseLong(quantity) ;
      Context context = new InitialContext();
      CompanySessionFacadeRemoteHome remoteHome =
         (CompanySessionFacadeRemoteHome)PortableRemoteObject.narrow
              (context.lookup("CompanySessionFacadeEJB"),
                                   CompanySessionFacadeRemoteHome.class);
      CompanySessionFacadeRemote remote = remoteHome.create() ;
      CompanyTO to = remote.getCompanyBySymbol(symbol) ;
      FolioTO folio = new FolioTO(to.getCompanyId(), symbol,
      to.getSharePrice(), numShares) ;
      PortfolioRemote portfolioRemote =
         (PortfolioRemote) session.getAttribute("portfolioRemote") ;
      portfolioRemote.purchaseShares(folio);
      request.setAttribute("message", SUCCESSFUL_TRANSACTION) ;
     }
     page = messagePage ;
    }
   }
  }
  catch(  /* Exception handling not shown  */ ) { /* etc.  */ }
  return page ;
 }
}
```

```
<!-- BuyShares.jsp -->

<HTML>
<TITLE>Buy Shares</TITLE>

<jsp:include page = "/WEB-INF/private/Banner.jsp" flush="true" />

<%@ include file = "Header.jsp" %>

<FORM METHOD="POST" ACTION="<%= controller %>" >
   <INPUT TYPE = "HIDDEN" NAME="command" VALUE="BuyShares">
   <INPUT TYPE = "HIDDEN" NAME="token"
                  VALUE=<%= (String) session.getAttribute("token") %> >
<Table  WIDTH = "100%" CELLPADDING = "10">
<COLGROUP>
  <COL WIDTH = "30%" VALIGN = "TOP" ALIGN = "RIGHT" >
  <COL ALIGN = "CENTER">
  <COL WIDTH = "40%" ALIGN = "LEFT">
 </COLGROUP>
 <TR>
   <TD></TD>
   <TD><H2><%= messages.getString("PurchaseShares") %><H2></TD>

 </TR>
 <TR>   <!—Using Resource Bundle →
  <TD><%= messages.getString("Symbol") %></TD>
  <TD><INPUT TYPE= "TEXT" NAME= "symbol" ></TD>
 </TR>
 <TR>
  <TD><%= messages.getString("Quantity") %></TD>
  <TD><INPUT TYPE= "TEXT" NAME= "quantity" ></TD>
 </TR>
 <TR>
  <TD></TD>
    <TD><INPUT TYPE="SUBMIT"
          VALUE = "<%= messages.getString("SubmitYourEntries") %>"></TD>
    <TD></TD>
 </TR>
</TABLE>
</FORM>

<jsp:include page = "/WEB-INF/private/Footer.jsp" flush="true" />

</HTML>
```

<!-- Header.jsp This is not a complete page. This is the start of a page. -->

```
<!--
        Every page includes this header.  The String controller
        is therefore available to every page.
-->

<%@ page import="java.util.*" %>

<%
    String controller = response.encodeURL("controller") ;
  ResourceBundle messages = (ResourceBundle)session.getAttribute("messages");
%>
```

```
<TABLE VALIGN = "TOP" cellpadding="10" cellspacing="0" ALIGN = "CENTER"
WIDTH = "100%">
 <TBODY>
  <COLGROUP>
   <COL WIDTH = "20%" VALIGN = "TOP"  ALIGN = "LEFT">
   <COL VALIGN = "TOP"  ALIGN = "CENTER">
  </COLGROUP>
 <TR >
   <TD class = "menu">
    <B><%= messages.getString("TradersCorner") %></B><BR><BR>
    <A HREF= "<%= controller%>?command=ViewCompanies">
                    <%= messages.getString("ViewCompanies") %></A><BR>
    <A HREF= "<%= controller%>?command=BuyShares">
                    <%= messages.getString("BuyShares") %></A><BR>
    <A HREF= "<%= controller%>?command=SellShares">
                    <%= messages.getString("SellShares") %></A><BR>
    <A HREF= "<%= controller%>?command=SellAllShares"><%=
messages.getString("SellAllShares") %></A><BR>
    <A HREF= "<%= controller%>?command=ViewPortfolio">
                    <%= messages.getString("ViewPortfolio") %></A><BR>
    <A HREF= "<%= controller%>?command=ViewYourGames"><%=
messages.getString("ViewYourGames") %></A><BR>
    <A HREF= "<%= controller%>?command=ViewLeagueTable">
                    <%= messages.getString("ViewLeagueTable") %></A><BR>
    <A HREF= "<%= controller%>?command=ViewSectionTable">
                    <%= messages.getString("ViewSectionTable") %></A><BR><BR>
    <B><%= messages.getString("PlayersCorner") %></B><BR><BR>
    <A HREF= "<%= controller%>?command=Login">
                    <%= messages.getString("Login") %></A><BR>
    <A HREF= "<%= controller%>?command=Register"><%=
messages.getString("Register") %></A><BR>
    <A HREF= "<%= controller%>?command=AmendPlayer">
                    <%= messages.getString("Amendyourdetails") %></A><BR>
    <A HREF= "<%= controller%>?command=DeletePlayer">
                    <%= messages.getString("Deleteyourdetails") %></A><BR>
     </TD>
   <TD>
```

```
package sk.sharesapp.resources ;
/*
   Default language English
*/
import java.util.* ;

public class MyResources extends ListResourceBundle
{
 public Object[][] getContents()
 {
  return contents ;
 }
 static final Object[][] contents =
 {
  {"messagePage",          "Message.jsp"} ,
  {"TradersCorner",        "Traders' Corner"} ,
  {"Home",                 "Home"} ,
  {"ViewCompanies",        "View Companies"} ,
  {"BuyShares",            "Buy Shares"} ,
  //  etc.
  {"ALREADY_LOGGED_IN",    "You are already logged in. "} ,
  {"PLAYER_EXISTS",        "You are already registered.  You should login."} ,
  {"NO_SUCH_PLAYER",       "Sorry, but you are not a registered member."} ,
  {"NOT_LOGGED_IN",        "Sorry, but you are not logged in."} ,
  // etc.
 };
}

public class w_MyResources extends ListResourceBundle
{
   public Object[][] getContents()
   {
      return contents ;
   }
   static final Object[][] contents =
   {
      {"WELCOME_PLAYER",        "Welcome "} ,
      // etc.
   };
}
```

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">

<%@ page import="java.io.*,java.util.*" %>
<%
 response.setContentType("text/vnd.wap.wml");
 String message = (String) request.getAttribute("message") ;
 String target =  response.encodeURL
          ("http://localhost:8988/SharesApp-Project-context-root/controller") ;
%>

<wml>
   <card id = "home" title = "Home">

      <p>
<%    if (message != null)
      {
%>
            <%= message %>
<%
      }
%>
      </p>
      <p>
       codename: <input name = "codename" type = "text"  />
      </p>
      <p>
       password:  <input name = "password" type = "password"  />
      </p>
      <do type= "accept" label="Go" optional = "false" name = "Go">
       <go method = "get" href="<%= target %>" >
        <postfield name = "codename" value="$(codename)" />
        <postfield name = "password" value="$(password)" />
        <postfield name = "device" value="mobile" />
        <postfield name = "command" value="Login" />
       </go>
      </do>
   </card>
</wml>
```

Appendix C: Configuration files

ejb-jar.xml

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
 <enterprise-beans>
  <session>
   <description>Session Bean ( Stateless )</description>
   <display-name>CompanyViewEJB</display-name>
   <ejb-name>CompanyViewEJB</ejb-name>
   <home>sk.sharesapp.ejb.session.stateless.CompanyViewEJBHome</home>
   <remote>sk.sharesapp.ejb.session.stateless.CompanyViewEJB</remote>
   <local-
home>sk.sharesapp.ejb.session.stateless.CompanyViewEJBLocalHome</local-
home>
   <local>sk.sharesapp.ejb.session.stateless.CompanyViewEJBLocal</local>
   <ejb-class>sk.sharesapp.ejb.session.stateless.CompanyViewBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
  </session>
  <session>
   <description>Session Bean ( Stateful )</description>
   <display-name>AccountEJB</display-name>
   <ejb-name>AccountEJB</ejb-name>
   <home>sk.sharesapp.ejb.session.stateful.AccountRemoteHome</home>
   <remote>sk.sharesapp.ejb.session.stateful.AccountRemote</remote>
   <ejb-class>sk.sharesapp.ejb.session.stateful.AccountBean</ejb-class>
   <session-type>Stateful</session-type>
   <transaction-type>Container</transaction-type>
  </session>
  <session>
   <description>Session Bean ( Stateless )</description>
   <display-name>CompanyListerEJB</display-name>
   <ejb-name>CompanyListerEJB</ejb-name>
   <home>sk.sharesapp.ejb.session.stateless.CompanyListerRemoteHome</home>
   <remote>sk.sharesapp.ejb.session.stateless.CompanyListerRemote</remote>
   <local-
home>sk.sharesapp.ejb.session.stateless.CompanyListerLocalHome</local-home>
   <local>sk.sharesapp.ejb.session.stateless.CompanyListerLocal</local>
   <ejb-class>sk.sharesapp.ejb.session.stateless.CompanyListerBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
  </session>
  <session>
   <description>Session Bean ( Stateless )</description>
   <display-name>CompanySessionFacadeEJB</display-name>
   <ejb-name>CompanySessionFacadeEJB</ejb-name>

<home>sk.sharesapp.ejb.session.stateless.CompanySessionFacadeRemoteHome</
home>

<remote>sk.sharesapp.ejb.session.stateless.CompanySessionFacadeRemote</remo
te>
   <ejb-
class>sk.sharesapp.ejb.session.stateless.CompanySessionFacadeBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
   <ejb-local-ref>
    <ejb-ref-name>ejb/local/CompanyEJB</ejb-ref-name>
    <ejb-ref-type>Entity</ejb-ref-type>
    <local-home>sk.sharesapp.ejb.entity.cmp.CompanyLocalHome</local-home>
    <local>sk.sharesapp.ejb.entity.cmp.CompanyLocal</local>
    <ejb-link>CompanyEJB</ejb-link>
   </ejb-local-ref>
  </session>
  <session>
   <description>Session Bean ( Stateless )</description>
   <display-name>PlayerGameSessionFacadeEJB</display-name>
   <ejb-name>PlayerGameSessionFacadeEJB</ejb-name>

<home>sk.sharesapp.ejb.session.stateless.PlayerGameSessionFacadeRemoteHome
</home>

<remote>sk.sharesapp.ejb.session.stateless.PlayerGameSessionFacadeRemote</re
mote>
```

```
    <ejb-
class>sk.sharesapp.ejb.session.stateless.PlayerGameSessionFacadeEJBBean</ejb-
class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
    <ejb-local-ref>
      <ejb-ref-name>ejb/local/PlayerEJB</ejb-ref-name>
      <ejb-ref-type>Entity</ejb-ref-type>
      <local-home>sk.sharesapp.ejb.entity.cmp.PlayerLocalHome</local-home>
      <local>sk.sharesapp.ejb.entity.cmp.PlayerLocal</local>
      <ejb-link>PlayerEJB</ejb-link>
    </ejb-local-ref>
    <ejb-local-ref>
      <ejb-ref-name>ejb/local/GameEJB</ejb-ref-name>
      <ejb-ref-type>Entity</ejb-ref-type>
      <local-home>sk.sharesapp.ejb.entity.cmp.GameLocalHome</local-home>
      <local>sk.sharesapp.ejb.entity.cmp.GameLocal</local>
      <ejb-link>GameEJB</ejb-link>
    </ejb-local-ref>
  </session>
  <session>
    <description>Session Bean ( Stateful )</description>
    <display-name>PortfolioEJB</display-name>
    <ejb-name>PortfolioEJB</ejb-name>
    <home>sk.sharesapp.ejb.session.stateful.PortfolioRemoteHome</home>
    <remote>sk.sharesapp.ejb.session.stateful.PortfolioRemote</remote>
    <ejb-class>sk.sharesapp.ejb.session.stateful.PortfolioBean</ejb-class>
    <session-type>Stateful</session-type>
    <transaction-type>Container</transaction-type>
    <ejb-local-ref>
      <ejb-ref-name>ejb/local/CompanyEJB</ejb-ref-name>
      <ejb-ref-type>Entity</ejb-ref-type>
      <local-home>sk.sharesapp.ejb.entity.cmp.CompanyLocalHome</local-home>
      <local>sk.sharesapp.ejb.entity.cmp.CompanyLocal</local>
      <ejb-link>CompanyEJB</ejb-link>
    </ejb-local-ref>
    <ejb-local-ref>
      <ejb-ref-name>ejb/local/GameEJB</ejb-ref-name>
      <ejb-ref-type>Entity</ejb-ref-type>
      <local-home>sk.sharesapp.ejb.entity.cmp.GameLocalHome</local-home>
      <local>sk.sharesapp.ejb.entity.cmp.GameLocal</local>
      <ejb-link>GameEJB</ejb-link>
    </ejb-local-ref>
    <ejb-local-ref>
      <ejb-ref-name>ejb/local/PlayerEJB</ejb-ref-name>
      <ejb-ref-type>Entity</ejb-ref-type>
      <local-home>sk.sharesapp.ejb.entity.cmp.PlayerLocalHome</local-home>
      <local>sk.sharesapp.ejb.entity.cmp.PlayerLocal</local>
      <ejb-link>PlayerEJB</ejb-link>
    </ejb-local-ref>
  </session>
  <session>
    <description>Session Bean ( Stateful )</description>
    <display-name>ValueListHandlerSessionFacadeEJB</display-name>
    <ejb-name>ValueListHandlerSessionFacadeEJB</ejb-name>

<home>sk.sharesapp.ejb.session.stateful.ValueListHandlerSessionFacadeRemoteHo
me</home>

<remote>sk.sharesapp.ejb.session.stateful.ValueListHandlerSessionFacadeRemote<
/remote>
    <ejb-
class>sk.sharesapp.ejb.session.stateful.ValueListHandlerSessionFacadeBean</ejb-
class>
    <session-type>Stateful</session-type>
    <transaction-type>Container</transaction-type>
  </session>
  <entity>
    <description>Entity Bean ( BMP )</description>
    <display-name>BmpCompanyEJB</display-name>
    <ejb-name>BmpCompanyEJB</ejb-name>
    <home>sk.sharesapp.ejb.entity.bmp.BmpCompanyRemoteHome</home>
    <remote>sk.sharesapp.ejb.entity.bmp.BmpCompanyRemote</remote>
    <local-home>sk.sharesapp.ejb.entity.bmp.BmpCompanyLocalHome</local-
home>
    <local>sk.sharesapp.ejb.entity.bmp.BmpCompanyLocal</local>
    <ejb-class>sk.sharesapp.ejb.entity.bmp.BmpCompanyBean</ejb-class>
    <persistence-type>Bean</persistence-type>
    <prim-key-class>java.lang.Long</prim-key-class>
    <reentrant>False</reentrant>
  </entity>
  <entity>
```

```
      <description>Entity Bean ( BMP )</description>
      <display-name>BmpCompanyPatternsEJB</display-name>
      <ejb-name>BmpCompanyPatternsEJB</ejb-name>

<home>sk.sharesapp.ejb.entity.bmp.BmpCompanyPatternsRemoteHome</home>
      <remote>sk.sharesapp.ejb.entity.bmp.BmpCompanyPatternsRemote</remote>
      <local-
home>sk.sharesapp.ejb.entity.bmp.BmpCompanyPatternsLocalHome</local-home>
      <local>sk.sharesapp.ejb.entity.bmp.BmpCompanyPatternsLocal</local>
      <ejb-class>sk.sharesapp.ejb.entity.bmp.BmpCompanyPatternsBean</ejb-class>
      <persistence-type>Bean</persistence-type>
      <prim-key-class>java.lang.Long</prim-key-class>
      <reentrant>False</reentrant>
    </entity>
    <entity>
      <description>Entity Bean ( CMP )</description>
      <display-name>CompanyEJB</display-name>
      <ejb-name>CompanyEJB</ejb-name>
      <home>sk.sharesapp.ejb.entity.cmp.CompanyRemoteHome</home>
      <remote>sk.sharesapp.ejb.entity.cmp.CompanyRemote</remote>
      <local-home>sk.sharesapp.ejb.entity.cmp.CompanyLocalHome</local-home>
      <local>sk.sharesapp.ejb.entity.cmp.CompanyLocal</local>
      <ejb-class>sk.sharesapp.ejb.entity.cmp.CompanyBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.Long</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>CompanyEJB</abstract-schema-name>
      <cmp-field>
        <field-name>companyid</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>symbol</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>name</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>shareprice</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>high</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>low</field-name>
      </cmp-field>
      <primkey-field>companyid</primkey-field>
      <query>
        <query-method>
          <method-name>findAll</method-name>
          <method-params/>
        </query-method>
        <ejb-ql>select object(o) from CompanyEJB o</ejb-ql>
      </query>
      <query>
        <query-method>
          <method-name>findCompaniesWithPriceLessThan</method-name>
          <method-params>
            <method-param>java.lang.Double</method-param>
          </method-params>
        </query-method>
        <ejb-ql>select object(c) from CompanyEJB c where c.shareprice &lt; ?1</ejb-ql>
      </query>
      <query>
        <query-method>
          <method-name>findCompaniesWithNameLike</method-name>
          <method-params/>
        </query-method>
        <ejb-ql>select object(c) from CompanyEJB c where c.name like '%Bank%'</ejb-
ql>
      </query>
      <query>
        <query-method>
          <method-name>findCompanyBySymbol</method-name>
          <method-params>
            <method-param>java.lang.String</method-param>
          </method-params>
        </query-method>
        <ejb-ql>select object (c) from CompanyEJB c where c.symbol = ?1</ejb-ql>
      </query>
    </entity>
    <entity>
```

```
  <description>Entity Bean ( CMP )</description>
  <display-name>GameEJB</display-name>
  <ejb-name>GameEJB</ejb-name>
  <local-home>sk.sharesapp.ejb.entity.cmp.GameLocalHome</local-home>
  <local>sk.sharesapp.ejb.entity.cmp.GameLocal</local>
  <ejb-class>sk.sharesapp.ejb.entity.cmp.GameBean</ejb-class>
  <persistence-type>Container</persistence-type>
  <prim-key-class>java.lang.Long</prim-key-class>
  <reentrant>False</reentrant>
  <cmp-version>2.x</cmp-version>
  <abstract-schema-name>GameEJB</abstract-schema-name>
  <cmp-field>
   <field-name>gameid</field-name>
  </cmp-field>
  <cmp-field>
   <field-name>balance</field-name>
  </cmp-field>
  <cmp-field>
   <field-name>period</field-name>
  </cmp-field>
  <primkey-field>gameid</primkey-field>
  <query>
   <query-method>
    <method-name>findAll</method-name>
    <method-params/>
   </query-method>
   <ejb-ql>select object(o) from GameEJB o</ejb-ql>
  </query>
 </entity>
 <entity>
  <description>Entity Bean ( CMP )</description>
  <display-name>PlayerEJB</display-name>
  <ejb-name>PlayerEJB</ejb-name>
  <local-home>sk.sharesapp.ejb.entity.cmp.PlayerLocalHome</local-home>
  <local>sk.sharesapp.ejb.entity.cmp.PlayerLocal</local>
  <ejb-class>sk.sharesapp.ejb.entity.cmp.PlayerBean</ejb-class>
  <persistence-type>Container</persistence-type>
  <prim-key-class>java.lang.Long</prim-key-class>
  <reentrant>False</reentrant>
  <cmp-version>2.x</cmp-version>
  <abstract-schema-name>PlayerEJB</abstract-schema-name>
  <cmp-field>
   <field-name>playerid</field-name>
  </cmp-field>
  <cmp-field>
   <field-name>codename</field-name>
  </cmp-field>
  <cmp-field>
   <field-name>password</field-name>
  </cmp-field>
  <cmp-field>
   <field-name>lastname</field-name>
  </cmp-field>
  <cmp-field>
   <field-name>firstname</field-name>
  </cmp-field>
  <cmp-field>
   <field-name>department</field-name>
  </cmp-field>
  <primkey-field>playerid</primkey-field>
  <query>
   <query-method>
    <method-name>findAll</method-name>
    <method-params/>
   </query-method>
   <ejb-ql>select object(o) from PlayerEJB o</ejb-ql>
  </query>
  <query>
   <query-method>
    <method-name>findByCodeNamePassword</method-name>
    <method-params>
     <method-param>java.lang.String</method-param>
     <method-param>java.lang.String</method-param>
    </method-params>
   </query-method>
   <ejb-ql>select object (p) from PlayerEJB p where p.codename = ?1 and
p.password = ?2</ejb-ql>
  </query>
  <query>
   <query-method>
    <method-name>findByCodeName</method-name>
    <method-params>
```

```
        <method-param>java.lang.String</method-param>
      </method-params>
    </query-method>
    <ejb-ql>select object (p) from PlayerEJB p where p.codename = ?1</ejb-ql>
  </query>
</entity>
<message-driven>
  <description>Message Driven Bean</description>
  <display-name>PriceWatchMessageDrivenEJB</display-name>
  <ejb-name>PriceWatchMessageDrivenEJB</ejb-name>
  <ejb-class>sk.sharesapp.ejb.mdb.PriceWatchMessageDrivenBean</ejb-class>
  <transaction-type>Container</transaction-type>
  <acknowledge-mode>Auto-acknowledge</acknowledge-mode>
  <message-driven-destination>
    <destination-type>javax.jms.Queue</destination-type>
    <subscription-durability>Durable</subscription-durability>
  </message-driven-destination>
  <ejb-local-ref>
    <ejb-ref-name>ejb/local/CompanyEJB</ejb-ref-name>
    <ejb-ref-type>Entity</ejb-ref-type>
    <local-home>sk.sharesapp.ejb.entity.cmp.CompanyLocalHome</local-home>
    <local>sk.sharesapp.ejb.entity.cmp.CompanyLocal</local>
    <ejb-link>CompanyEJB</ejb-link>
  </ejb-local-ref>
</message-driven>
<message-driven>
  <description>Message Driven Bean</description>
  <display-name>SimpleMessageDrivenEJB</display-name>
  <ejb-name>SimpleMessageDrivenEJB</ejb-name>
  <ejb-class>sk.sharesapp.ejb.mdb.SimpleMessageDrivenEJBBean</ejb-class>
  <transaction-type>Container</transaction-type>
  <acknowledge-mode>Auto-acknowledge</acknowledge-mode>
  <message-driven-destination>
    <destination-type>javax.jms.Queue</destination-type>
    <subscription-durability>Durable</subscription-durability>
  </message-driven-destination>
</message-driven>
</enterprise-beans>
<relationships>
  <ejb-relation>
    <ejb-relation-name>GameEJB - PlayerEJB</ejb-relation-name>
      <ejb-relationship-role>
        <ejb-relationship-role-name>GameEJB may have one PlayerEJB</ejb-
relationship-role-name>
        <multiplicity>Many</multiplicity>
        <cascade-delete/>
        <relationship-role-source>
          <ejb-name>GameEJB</ejb-name>
        </relationship-role-source>
        <cmr-field>
          <cmr-field-name>playerEJB_playerid</cmr-field-name>
        </cmr-field>
      </ejb-relationship-role>
      <ejb-relationship-role>
        <ejb-relationship-role-name>PlayerEJB may have many GameEJB</ejb-
relationship-role-name>
        <multiplicity>One</multiplicity>
        <relationship-role-source>
          <ejb-name>PlayerEJB</ejb-name>
        </relationship-role-source>
        <cmr-field>
          <cmr-field-name>gameEJB_playerid</cmr-field-name>
          <cmr-field-type>java.util.Collection</cmr-field-type>
        </cmr-field>
      </ejb-relationship-role>
    </ejb-relation>
  </relationships>
  <assembly-descriptor>
    <container-transaction>
      <method>
        <ejb-name>CompanyViewEJB</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
    <container-transaction>
      <method>
        <ejb-name>AccountEJB</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
```

```
<container-transaction>
 <method>
  <ejb-name>CompanyListerEJB</ejb-name>
  <method-name>*</method-name>
 </method>
 <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
 <method>
  <ejb-name>BmpCompanyEJB</ejb-name>
  <method-name>*</method-name>
 </method>
 <trans-attribute>Supports</trans-attribute>
</container-transaction>
<container-transaction>
 <method>
  <ejb-name>BmpCompanyPatternsEJB</ejb-name>
  <method-name>*</method-name>
 </method>
 <trans-attribute>Supports</trans-attribute>
</container-transaction>
<container-transaction>
 <method>
  <ejb-name>CompanyEJB</ejb-name>
  <method-name>*</method-name>
 </method>
 <trans-attribute>Supports</trans-attribute>
</container-transaction>
<container-transaction>
 <method>
  <ejb-name>CompanySessionFacadeEJB</ejb-name>
  <method-name>*</method-name>
 </method>
 <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
 <method>
  <ejb-name>GameEJB</ejb-name>
  <method-name>*</method-name>
 </method>
 <trans-attribute>Supports</trans-attribute>
  </container-transaction>
  <container-transaction>
   <method>
    <ejb-name>PlayerEJB</ejb-name>
    <method-name>*</method-name>
   </method>
   <trans-attribute>Supports</trans-attribute>
  </container-transaction>
  <container-transaction>
   <method>
    <ejb-name>PlayerGameSessionFacadeEJB</ejb-name>
    <method-name>*</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
  </container-transaction>
  <container-transaction>
   <method>
    <ejb-name>PortfolioEJB</ejb-name>
    <method-name>*</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
  </container-transaction>
  <container-transaction>
   <method>
    <ejb-name>ValueListHandlerSessionFacadeEJB</ejb-name>
    <method-name>*</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
  </container-transaction>
 </assembly-descriptor>
</ejb-jar>
```

orion-ejb-jar.xml

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<!DOCTYPE orion-ejb-jar PUBLIC "-//Evermind//DTD Enterprise JavaBeans 1.1
runtime//EN" "http://xmlns.oracle.com/ias/dtds/orion-ejb-jar.dtd">
<orion-ejb-jar>
 <enterprise-beans>
  <session-deployment name="CompanyViewEJB"/>
```

```
    <session-deployment name="AccountEJB"/>
    <session-deployment name="CompanyListerEJB"/>
    <session-deployment name="CompanySessionFacadeEJB"/>
    <session-deployment name="PlayerGameSessionFacadeEJB"/>
    <session-deployment name="PortfolioEJB"/>
    <session-deployment name="ValueListHandlerSessionFacadeEJB"/>
    <entity-deployment name="BmpCompanyEJB"/>
    <entity-deployment name="BmpCompanyPatternsEJB"/>
    <entity-deployment name="CompanyEJB" data-source="jdbc/sharesDS"
table="SHARES.COMPANY">
      <primkey-mapping>
        <cmp-field-mapping name="companyid" persistence-name="COMPANYID"
persistence-type="NUMBER(6)"/>
      </primkey-mapping>
      <cmp-field-mapping name="companyid" persistence-name="COMPANYID"
persistence-type="NUMBER(6)"/>
      <cmp-field-mapping name="symbol" persistence-name="SYMBOL" persistence-
type="VARCHAR2(20)"/>
      <cmp-field-mapping name="name" persistence-name="NAME" persistence-
type="VARCHAR2(80)"/>
      <cmp-field-mapping name="shareprice" persistence-name="SHAREPRICE"
persistence-type="NUMBER(10,2)"/>
      <cmp-field-mapping name="high" persistence-name="HIGH" persistence-
type="NUMBER(10,2)"/>
      <cmp-field-mapping name="low" persistence-name="LOW" persistence-
type="NUMBER(10,2)"/>
    </entity-deployment>
    <entity-deployment name="GameEJB" data-source="jdbc/sharesDS"
table="SHARES.GAME">
      <primkey-mapping>
        <cmp-field-mapping name="gameid" persistence-name="GAMEID" persistence-
type="NUMBER(6)"/>
      </primkey-mapping>
      <cmp-field-mapping name="gameid" persistence-name="GAMEID" persistence-
type="NUMBER(6)"/>
      <cmp-field-mapping name="balance" persistence-name="BALANCE" persistence-
type="NUMBER(10,2)"/>
      <cmp-field-mapping name="period" persistence-name="PERIOD" persistence-
type="DATE"/>
      <cmp-field-mapping name="playerEJB_playerid" persistence-
name="PLAYERID">
        <entity-ref home="PlayerEJB">
          <cmp-field-mapping persistence-name="PLAYERID" persistence-
type="NUMBER(6)"/>
        </entity-ref>
      </cmp-field-mapping>
    </entity-deployment>
    <entity-deployment name="PlayerEJB" data-source="jdbc/sharesDS"
table="SHARES.PLAYER">
      <primkey-mapping>
        <cmp-field-mapping name="playerid" persistence-name="PLAYERID"
persistence-type="NUMBER(6)"/>
      </primkey-mapping>
      <cmp-field-mapping name="playerid" persistence-name="PLAYERID"
persistence-type="NUMBER(6)"/>
      <cmp-field-mapping name="codename" persistence-name="CODENAME"
persistence-type="VARCHAR2(20)"/>
      <cmp-field-mapping name="password" persistence-name="PASSWORD"
persistence-type="VARCHAR2(10)"/>
      <cmp-field-mapping name="lastname" persistence-name="LASTNAME"
persistence-type="VARCHAR2(30)"/>
      <cmp-field-mapping name="firstname" persistence-name="FIRSTNAME"
persistence-type="VARCHAR2(30)"/>
      <cmp-field-mapping name="department" persistence-name="DEPARTMENT"
persistence-type="VARCHAR2(30)"/>
      <cmp-field-mapping name="gameEJB_playerid">
        <collection-mapping table="SHARES.GAME">
          <primkey-mapping>
            <cmp-field-mapping>
              <entity-ref home="PlayerEJB">
                <cmp-field-mapping name="playerEJB_playerid_playerid" persistence-
name="PLAYERID" persistence-type="NUMBER(6)"/>
              </entity-ref>
            </cmp-field-mapping>
          </primkey-mapping>
          <value-mapping type="sk.sharesapp.ejb.entity.cmp.GameLocal">
            <cmp-field-mapping>
              <entity-ref home="GameEJB">
                <cmp-field-mapping name="gameEJB_playerid_gameid" persistence-
name="GAMEID" persistence-type="NUMBER(6)"/>
              </entity-ref>
            </cmp-field-mapping>
```