AN ONLINE THREE-TIERED INSURANCE SYSTEM:

A WEB-BASED ADVANCED APPLICATION

FOR THE J2EE PLATFORM

By

XINYUE ZHU

Bachelor of Science

Hunan Financial University

ChangSha, P. R. China

1991

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May 2002

AN ONLINE THREE-TIERED INSURANCE SYSTEM:

A WEB-BASED ADVANCED APPLICATION

FOR THE J2EE PLATFORM

Thesis Approved:

M. H. Samadzadeh

Thesis Advisor

D. E. Hall

J Chandler

Timothy J. Pettitone

Dean of the Graduate College

PREFACE

The client/server application model has gained acceptance in the computer industry. The three-tiered software architecture emerged in the 1990s to overcome the limitations of the two-tiered architecture. The foundation of the modern three-tiered architecture is the distributed object system architecture. There are several existing technologies for building modern distributed object and client/server systems, such as Common Object Request Broker Architecture (CORBA) and Distributed Component Object Model (DCOM). To reduce cost and for the purpose of fast-track enterprise application design and development, the Java2 Enterprise Edition (J2EE) technologies provide a multi-tiered innovative distributed application model. In this thesis, the aforementioned application models were briefly reviewed. Then the J2EE technologies and other relevant technologies were investigated in detail through applying them to implement an online three-tiered insurance application system.

Currently, the main technologies in J2EE being adopted by enterprise developers include Java Database Connectivity (JDBC), Java Naming and Directory Interface (JNDI) for accessing services, and Java Remote Method Invocation (RMI) for remote method invocation and distributed objects. At the top of this foundation are servlets, Java Server Pages (JSPs), and Enterprise Java Beans (EJBs). A brief description, a model design, and an actual implementation of these technologies were included in this thesis for an online insurance application system. Besides these enterprise Java technologies,

the selection and configuring of the application server and the deployment tool were also included in this thesis. After the entire application system was developed, the system was tested and evaluated. It was observed that the application system met the standard criteria based on its well-defined development and deployment processes.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my advisor Dr. Mansur H. Samadzadeh. He provided essential guidance through my thesis work. He spent a lot of time reviewing my thesis and providing suggestions for refinement. He is my advisor at Oklahoma State University. He is also my teacher in my whole life.

I would like to thank my other committee members, Dr. G. E. Hedrick and Dr. J. P. Chandler. Their time and support are greatly appreciated.

Sincere thanks are extended to all my professors at Oklahoma State University from whom I have gained so much.

Great thanks also go to my parents, and especially to my husband. It is their love, understanding, encouragement, and expectation that have kept me motivated and escorted me in all my endeavors through the years.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

With the growing use of Internet and intranets for enterprise applications, the three-tiered application models are becoming more popular and more important in computer-based industries. The foundation of modern three-tiered architecture is the distributed object system [Asbury and Weiner 99]. A distributed object system allows objects implemented on one computer to send messages to objects running in other memory address spaces across the network [Asbury and Weiner 99]. To reduce cost and for the purpose of fast-track enterprise application design and development, the Java2 Enterprise Edition (J2EE) platform provides a component-based, multi-tiered distributed application model.

To develop an enterprise application using J2EE, one needs to understand many technologies [Asbury and Weiner 99]. In addition to the Java language, it is important to understand the technologies that support networking, the Web, and database management system. Also, a number of relevant enterprise Java technologies need to be mastered.

The objective of this thesis was to study the J2EE and other relevant technologies, and, from the perspective of a real-world project, to develop an online three-tiered insurance application system using these technologies. In this thesis, Chapter II presents some relevant existing techniques and analyzes the advantages of using J2EE

technologies in distributed applications. As explained in Chapter III, based on a solid knowledge of the J2EE platform, a three-tiered online insurance application system was designed and implemented. Chapter IV discusses the testing and evaluation of the application. Finally, Chapter V presents the conclusions of the work and outlines some areas of future work.

CHAPTER II

LITERATURE REVIEW

In the past several years, the client/server model has started to gain acceptance in the computer industry. The client/server software architecture is a message-based and modular infrastructure that is intended to improve usability, interoperability, and scalability as compared to centralized, mainframe, and timesharing computing [Sadoski 97].

The three-tiered software architecture emerged in the 1990s to overcome the limitations of the two-tiered architecture [Berg 98]. In this model, the third tier (middle tier server) is between the user interface (client) and the data management (server) components. The three-tiered architecture is used when an effective distributed client/server design is needed.

The foundation of modern three-tiered architecture is the distributed object system [Asbury and Weiner 99]. Distributed systems are the entire set of existing hardware and software that allow developers to build multi-tiered solutions [Hoque 99]. For the last two decades, CORBA (Common Object Request Broker Architecture) and DCOM (Distributed Component Object Model) have both emerged as important technologies for building modern distributed object and client/server systems [Rosen and Curtis 98].

2.1 Existing Techniques

CORBA

CORBA was designed as a distributed object infrastructure whose goals were interoperability and integration in distributed systems [Rosen and Curtis 98]. CORBA is a specification and an architecture for creating, distributing, and managing distributed program objects in a network. CORBA relies on an Object Request Broker (ORB) to enable a client object to make a server request without knowing where the server object or component is located and what its interfaces are. To make requests or return replies among ORBs on the Internet, programs use the Internet Inter-ORB Protocol (IIOP) [Pritchard 99]. The IIOP is an object-oriented protocol that makes it possible for distributed programs written in different programming languages to communicate over the Internet.

DCOM

DCOM is Microsoft's approach to a network-wide environment for object communication and services [Asbury and Weiner 99]. It provides a set of interfaces allowing applications to communicate on a single computer running a Windows operating system.

DCOM is aimed at providing capabilities similar to those defined in CORBA but for applications on a single computer.

In an enterprise application, both CORBA and DCOM have their specific flaws [Rosen and Curtis 98] as briefly described below.

In the case of CORBA, a programmer need to create language mappings to an intermediate format called the Interface Definition Language (IDL). IDL is part of a language-neutral strategy provided by CORBA. Every language has a mapping to and from the IDL. This mapping limits the capabilities of distributed applications.

In the case of DCOM, programmers can send messages only to Microsoft platforms. It can be argued that DCOM defeats the purpose of a general distributed object solution, which is to create a seamless process.

2.2 J2EE Technologies

Java has been recognized as the platform (i.e., environment) of choice for creating enterprise solutions, specifically for developing distributed applications [Asbury and Weiner 99]. To reduce cost and for the purpose of fast-track design and development of enterprise applications, the J2EE platform provides a component-based, multi-tiered distributed application model [Pawlan 01]. J2EE has the ability to reuse components, provides a unified security model, and has a flexible transaction control [Asbury and Weiner 99]. As a result, developers can deliver solutions faster, and the solutions can be platform-independent and component-based.

Currently, the main technologies in J2EE being adopted by enterprise developers include Java Database Connectivity (JDBC), Java Naming and Directory Interface (JNDI) for accessing services, and Java Remote Method Invocation (RMI) for Java

remote method invocation and distributed objects [Pawlan 01]. On top of this foundation are servlets, Java Server Pages (JSPs), and Enterprise Java Beans (EJBs). All these combine into a feature-rich tool kit for developing enterprise applications for distributed objects [Pawlan 01].

Because of the increasing demand of e-commerce on information technology and the innovative advantages and widespread use of the J2EE technologies, learning and developing enterprise applications for distributed systems using the J2EE technologies is of particular significance and interest. The thrust of thesis was to design and implement an online three-tiered insurance application system using these relevant technologies to demonstrate and manipulate the major business transaction processes of a typical small insurance company.

CHPATER III

THREE-TIERED ONLINE INSURANCE APPLICATION SYSTEM

3.1 System Overview

The J2EE platform uses a multi-tiered distributed application model [Pawlan 01]. This means that the application logic is divided into components according to specific functions. The J2EE multi-tiered applications are generally considered to be three-tiered applications because they are distributed over three different levels/tiers: client tier at the front end, J2EE server at the middleware, and the database server at the back-end. Three-tiered applications that run in this way extend the standard two-tiered client and server model by placing a multithreaded application server between the client application and the back-end storage as the middle tier. The general online three-tiered application system diagram using J2EE technologies is given below in Figure 1.

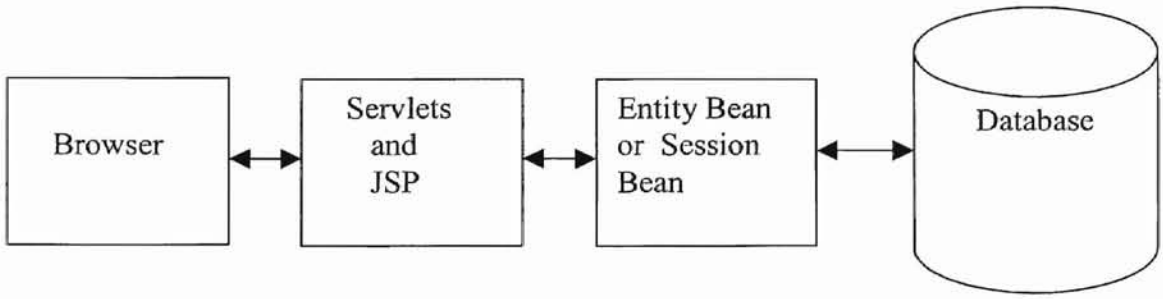


Figure 1. Client/Server Application in J2EE Platform [Pawlan 01]

The HyperText Markup Language (HTML) pages or the application clients get input from and show information to the user in the browser; behind the client side are servlets and JSP that pass data between the browser and the EJB server; and the EJB server handles reading from and writing to the database [Austin and Pawlan 00]. J2EE applications are made up of components. The J2EE specification defines the following components [Pawlan 01]:

- HTML browser or Java applets are components that run on the client.
- Java Servlets and JSPs technology components are web components that run on the web server.
- EJB components (enterprise beans) are business components that run on the application server.
- Enterprise Information Systems are database systems that run on the database server.

Normally, an online insurance order process has the following steps: customers request services/products, the insurance company processes the requests, and the insurance company responds to the customers. To simulate this process, a client/server three-tiered application system was designed and implemented consisting of the client side, the middleware, and the back end as discussed below.

3.1.1 Client Side

The client side is an order-entry online application. It lets the end-users view and search the various insurance products and view product details, and then enter their requests including the services selected and their personal information such as user name and password which will be needed in the registration process. The client side will also

let the employees to retrieve information and to do some analysis. Using J2EE there are two kinds of client: a web client and an application client [Pawlan 01] as explained below.

A web client is also called a thin-client since most of the processing takes place on the server. It consists of dynamic web pages generated by web components running in the web tier using various markup languages (HTML, XML, and so on), and a web browser that renders the pages received from the server.

An application client typically runs on a client machine and provides a way for users to handle tasks that require a richer user interface that can be provided by a markup language [Pawlan 01]. It usually has a graphical user interface created using Java Swing or Java AWT (Abstract Window Toolkit) APIs (Application Program Interface). Application clients directly access enterprise beans running in the business tier, and they can open a Hypertext Transfer Protocol (HTTP) connection to establish communication with a servlet running in the web tier.

3.1.2 Middleware

The middle tier encapsulates business logic [Monson-Haefel 99], it receives requests from the client through the HTTP protocol. According to the type of the requests, the middleware will get information from server, process the request, store the data back to the server, and respond to customers by sending a dynamic page to the client. In the J2EE environment, the middleware consists of the web tier and the business tier [Pawlan 01].

The web tier can consist of servlets, JSP pages, or both. Servlets are Java programming language classes that dynamically process requests and construct responses [Asbury and Weiner 99]. JSP pages are text-based documents that execute as servlets, but allow a more natural approach to creating static content [Pawlan 01].

The business tier consists of the kinds of enterprise beans that represent the business logic. An enterprise bean receives data from client programs, processes the data, and sends the data to the back-end for storage. An enterprise bean also retrieves data from storage, processes it, and sends it back to the client program. In the insurance application system developed in this thesis, there are two kinds of enterprise beans: entity bean and session bean. A session bean represents a transient conversation with a client [Monson-Haefel 99]. When the client finishes executing, the session bean and its data are gone. In contrast, an entity bean represents persistent data stored in one row of a database table [Monson-Haefel 99]. If the client terminates or if the server shuts down, the underlying services ensure the entity bean data is saved.

3.1.3 Database Side

A Database Management System is used on server's side. The database server stores all of the customers' data as well as the server data for internal processing.

3.2 System Specification

The goal of this thesis was to model an insurance company. In the real world, an insurance company consists of many inter-related aspects. However, this thesis study did

not model every aspect of this kind of system. It mainly focused on the business transaction processes of products/services people typically care most about in an insurance industry. By combining various enterprise Java technologies (see Section 2.2), and a specific simplified insurance application model, this thesis project created a component-based, distributed application in which the user interface is displayed on the web browser and the communications between the front end and the back end, as well as the enforcement of the business logic is managed in the server programs.

During the design and implementation of the application system, system requirements (including user requirements and data requirements) were considered.

3.2.1 User Requirements

For an online insurance system, the following key application capabilities are needed with respect to the user-oriented information.

- First, this application system needs users to enter their personal information at the time of registration, it also needs to respond dynamically to the users' input while choosing and buying products and policies.
- Second, this application system needs to categorize the available insurance types (such as Auto, Health, and Life) so that the customer doesn't have to look through long lists. To do this, the application system displays the main page and links to specific product pages.
- Third, the users need to search and view the products, and then choose and buy specific policies.

• Finally, the system needs to provide tracking services. The application system needs to keep track of the selection information and the historical records such as the claim records, so that the employees can retrieve information and do case analyses.

The abstract activity diagram of this application system is depicted below in Figure 2.

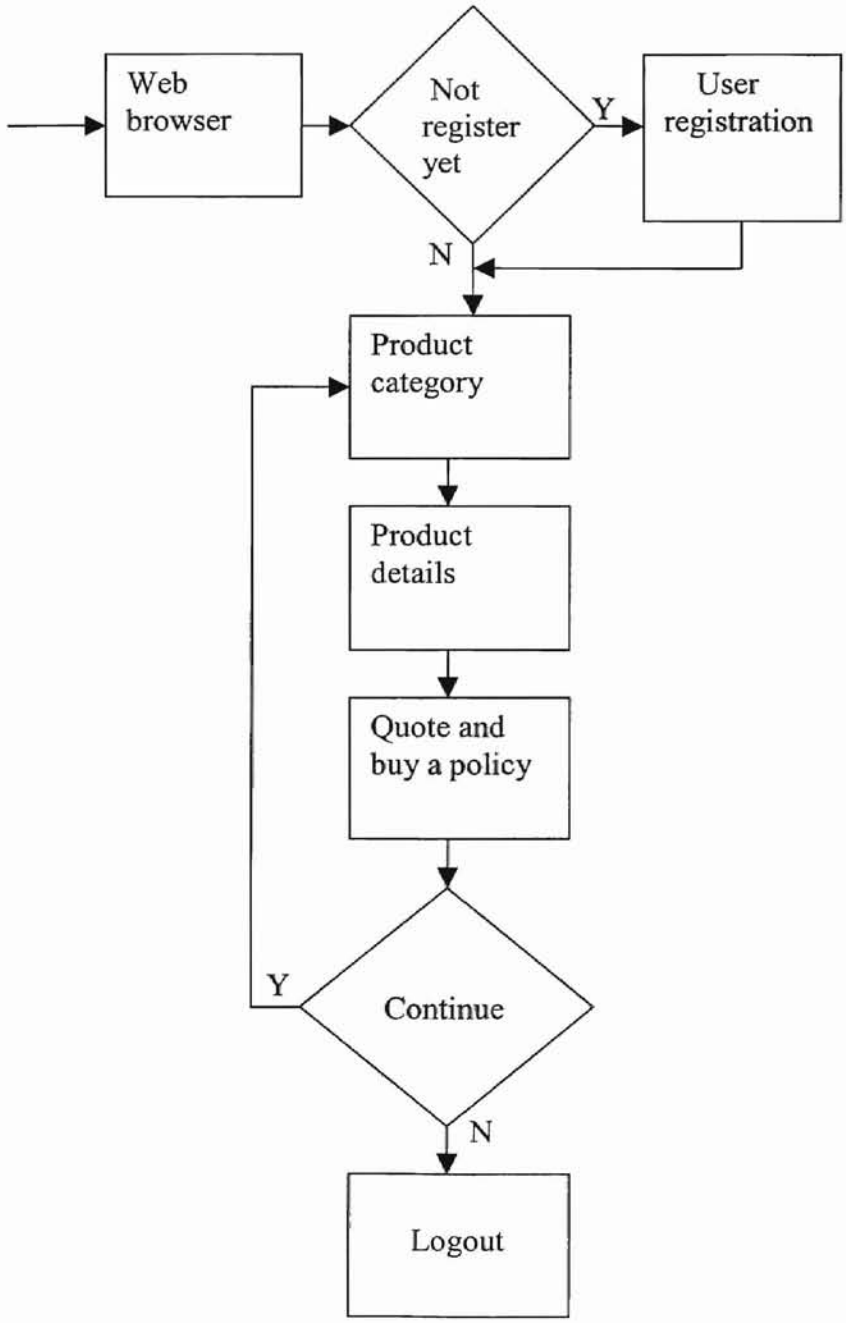


Figure 2. User Activity Diagram

3.2.2 Data Requirements

The data requirements part of the overall system requirements are listed below.

- End Users. Customers are identified by their username and password. The insurance company stores the personal information such as their names and email addresses. Customers can access the specific insurance products' policies.
- Quote Record. After searching and viewing general information about the available products, customers can get a quote for a specific insurance product. There are auto, health, and life insurance quotes. Each quote is assigned a unique quote id. The auto quote stores the customer information and the auto information such as manufacturer, model, and total value. The health and life quotes have the customer's name, age, the plan types and the coverage type, the medical record, beneficiary information and other personal information.
- Products/Services. The insurance company provides three types of insurance products: auto, health, and life. After submitting a specific quote, the server will return the appropriate policy based on the quoted information. Each policy is assigned a unique policy number and also has the insured value, coverage amount, and other relevant information.
- Claim Record. If there is an accident, a customer will file an online claim providing the policy number, the accident time, and the estimated loss. A unique claim number will be assigned by the server.

3.3 System Implementation

3.3.1 Choosing the Right Software and Run-Time Environment

Most of the insurance application in this thesis was created using the following Java APIs: JDBC API for database access, Java multithreading and Java RMI APIs for multiple users to remotely access the system, EJB APIs for business logic (insurance products/services), and Java Servlets and JSP for Web pages to process user input. Also HTML, the extended Markup Language (XML), and JavaScript languages were used for Web application and deployment.

• Tools for Developing and Running the Applicaition

Development tools and an application server are essential to running J2EE applications [Alur and Crupi 01]. In this thesis project, the BEA WebLogic server was used as the application server. For building, deploying, and managing e-business applications and transactions, the BEA WebLogic family of application servers deliver a platform for Web-based applications using the distributed object technologies including J2EE [BEA 01]. The BEA WebLogic server provides a complete set of services and handles many low-level details of application behavior [BEA 01].

The concepts of components and container are fundamental to the benefits of the J2EE standard [Pawlan 01]. An application server provides the underlying services in the form of a container for every component type. A container is responsible for creating new instances of enterprise beans, making sure that the server stores them properly. Tools provided by the server do a lot of work behind the scenes, such as creating the mapping

between entity beans and records in the database, and generating code based on EJB components to handle low-level details.

After each enterprise bean was developed, it was deployed by writing a corresponding deploy descriptor using deploy tools. The deploy descriptor in this thesis is an XML based file, it is used to describe a component's deployment setting.

• How Multi-Tiered Applications Work

The goal in a multi-tiered application is for the client to be able to work on application data without knowing at build time where the data is stored in the third tier [Asbury and Weiner 99]. To make this level of transparency possible, the underlying services in a multi-tiered architecture use lookup services to locate remote server objects. The underlying services also use data communication services to move data from the client, through the remote server object, to its final destination in a storage medium. For this J2EE application in this thesis, the independent and reusable components were put together as depicted below in Figure 3.

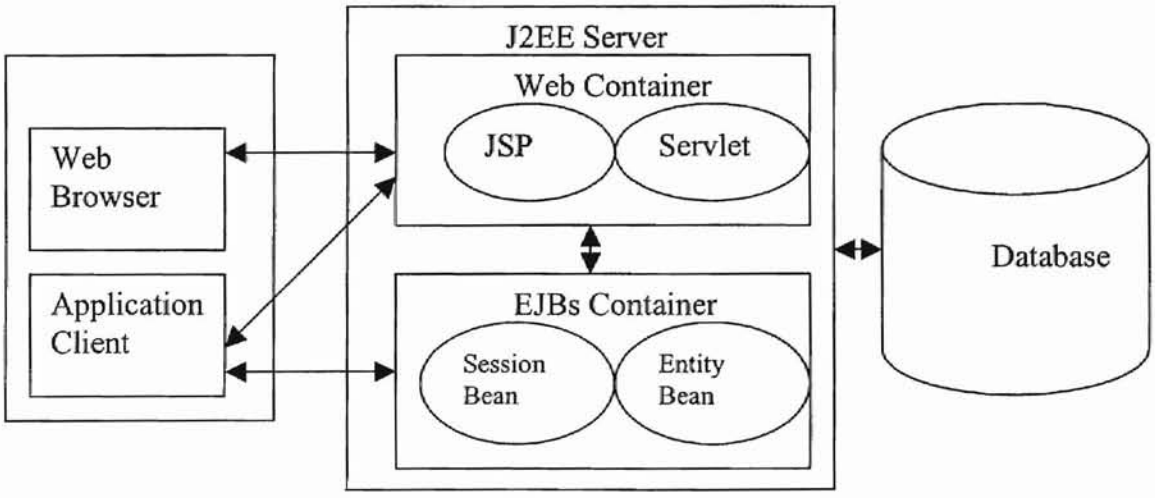


Figure 3. J2EE Server and Container [Pawlan 01]

3.3.2 Creating User Interface and Web Components

3.3.2.1 The Home Page

To make the application work, a main page (or home page) was created for this thesis application. The main page briefly describes the objective of this thesis study and displays some general information about the J2EE environment (Figure 4). It also lists the available insurance products/services in this application (i.e., auto, health, and life insurance). From the main page, end-users can access specific pages titled home, login, profile, search, product information (include product details), claim, and an informational page called "about us". From these pages, the users can carry out the desired specific task. In the home page and every specific page, including the dynamic response Web pages, there is a banner attached at the top. This banner was created using the JSP technology to provide users with a convenient way to randomly access the system.

About Xinyue's Shop

Welocme to Xinyue's Insurance Shop!

Home | Login in | Profile | About Us | Product Information | Claim

The main objective of this thesis was to study the J2EE technologies, and from the perspective of a real world project to implement a three-tiered online insurance application using these technologies. The thesis mainly focuses on the business transaction processes of an insurance company.

The outline of this client/server application follows.

1) The first tier (front tier) is the Graphics User Interface for client access.

2) The second tier (middleware) is the application server for business logic.

3) The third tier (back end) is the database server for the needed information.

This application mainly involves using some mainly J2EE technologies:

JDBC (Java Database Connectivity)	for database access
JNDI (Java Naming and Directory Interface)	for lookup services
EJB (Enterprise Java Bean)	for business logic presentation
Java Servlets and JSP (Java Server Page)	for Web application
Java RMI (Remote Method Invocation)	for accessing objects remotely through the network

It provides three kinds of products/services:

a. Auto Insurance

AUTO

b. Health Insurance

HEALTH

c. Life Insurance

Figure 4. Home Page

3.3.2.2 The Web Components

In the J2EE environment, a Web component consists of servlets and JSP [Pekowsky 00]. In combination with the HTML forms, servlets and JSP provide a mechanism for a true two-way interaction between the client and the server [Pawlan 01]. The Web components servlets, JSP, and JavaBeans are briefly explained below.

Servlet: A servlet is a server-side program written in Java that interacts with clients, and is usually tied to an HTTP server [Callaway 99]. A servlet accepts end-user input from the browser by way of HTTP, passes the input to the appropriate enterprise bean for processing, and displays the processed results to the end user in the browser.

The insurance application in this thesis used the servlets called AutoQuoteServlet, HealthQuoteServlet, LifeQuoteServlet to accept and process end-user input through the browser and to dynamically return product information to the browser. The following figure presents a Unified Modeling Language (UML) class diagram [Austin and Pawlan 00] for these servlet classes.

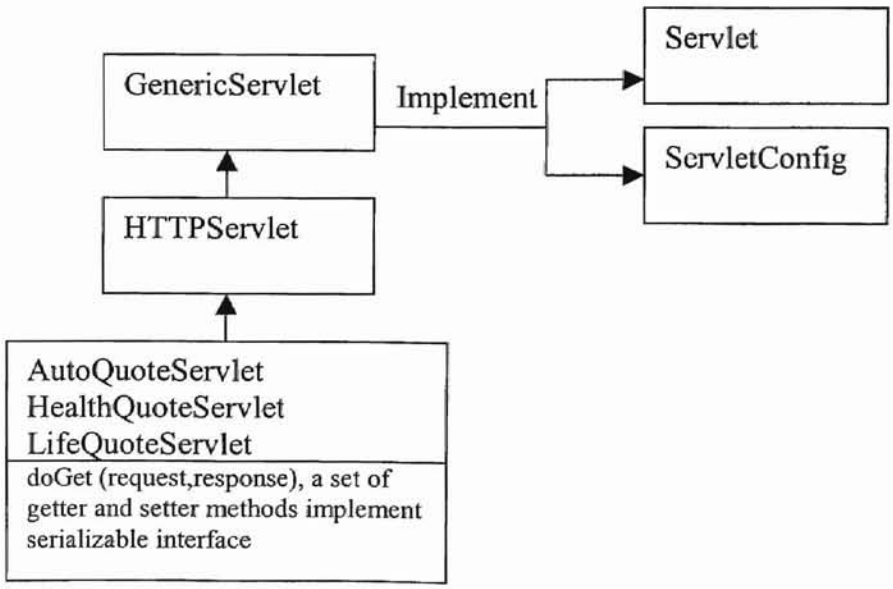


Figure 5. UML Class Diagram for Servlets

JSP: JSP is an extension of the servlet component model that simplifies the process of generating HTML dynamically [Pekowsky 00]. A JSP page is a text-based document that contains two types of text: static template data, which can be expressed in any text-based format such as HTML or XML, and JSP elements, which construct dynamic content [Pawlan 01].

The main job of JSP in the insurance application in this thesis is presentation. A strategy for developing maintainable JSP pages is to minimize the amount of scripting embedded in the pages [Pawlan 01]. In order to achieve this, most dynamic processing tasks of this thesis application were delegated to enterprise beans and JavaBeans components.

JavaBeans: JavaBeans components are Java classes that can be reused and composed together into applications [Casto 98]. JavaBeans are client-side components. JSP technologies directly support using JavaBeans components with JSP language elements. Developers can create and initialize beans, and get and set the values of their properties.

In the insurance application in this thesis, there are several kinds of JSP pages: login, claim, quote, and policy. These JSP pages are briefly discussed below.

The login page (login.jsp) is used for user registration. The claim page (claim.jsp) is used for users filing out an online claim. The product information page links to specific product pages: End-users can view general information about specific products such as auto insurance, health insurance, and life insurance. If end-users are interested, they can go to a quote page to get a specific quote. The required quote information includes the driver information and the vehicle information for auto quotes, and personal information and doctor record for health quotes and life quotes.

For the auto insurance quote process, there are driverBean and vehicleBean to encapsulate the necessary information, and they also correspond to the respective JSP pages. By embedding Java code into the script language, the dynamic pages were created. The format of a JSP file is displayed below in List 1.

List 1. Code segment for the getAutoQuoteDriver.jsp file

```
<jsp:useBean          id="driver"          class="com.web.beans.DriverBean"
scope="session"/>
<HTML><HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE></TITLE></HEAD>
<BODY>
<jsp:include page="banner.jsp" flush="true"/><P>
<TABLE border=1 cellPadding=1 cellSpacing=1 width="75%">
   <TR>
    <TD align=middle><A href="getautoquote_driver.jsp">Driver
    Information</A></TD>
    <TD      align=middle><A      href="getautoquote_vehicle.jsp">Vehicle
Information</A></TD></TR></TABLE></P>
<%
      com.web.beans.DriverBean dr = new com.web.beans.DriverBean();
      String m = request.getParameter("email");
      dr.setEmail(m);
      session.setAttribute("dr", dr); %>

<FORM action="getautoquote_vehicle.jsp" id=FORM1 method=Post name=FORM1>
<P> </P>
<P align=center>Please fill out the information for the applicant.</P>
<P>Email Address; 
<INPUT id=mailid name=email></P>
<P>FirstName; 
<INPUT id=text1 name=firstname></P>
<P>MiddleName;      <INPUT  id=text2  name=middlename  style="HEIGHT:
22px; WIDTH: 75px"></P>
<P>LastName;      <INPUT     id=text3     name=lastname></P><P><FONT
size=1> </p>
```

Figure 6 below depicts the interface for users entering their personal information to get an online quote.

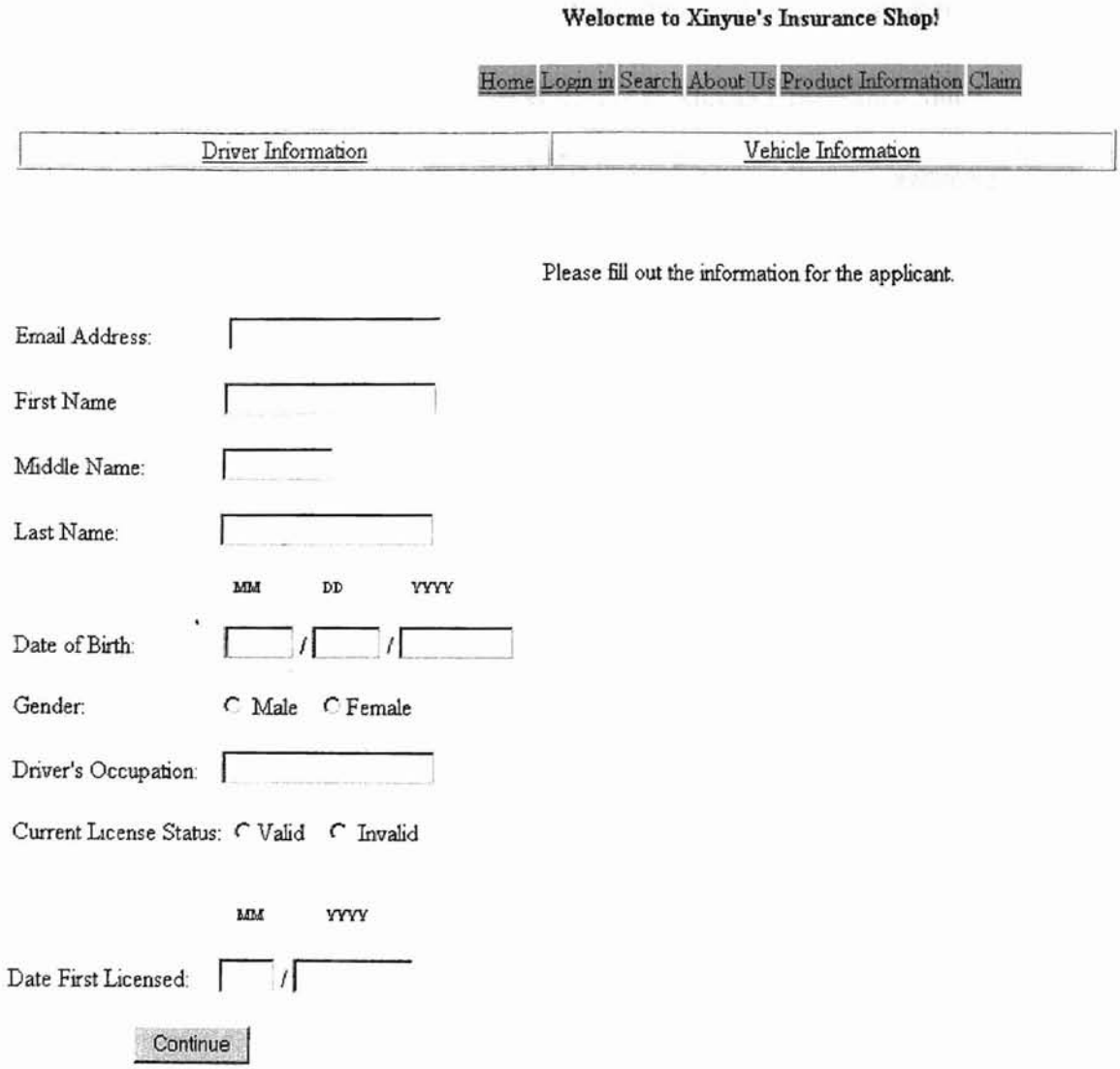


Figure 6. Interface1 for Auto Quote

By pressing the "Continue " button in the interface depicted in Figure 6, the interface depicted in Figure 7 below will appear which will need vehicle information.

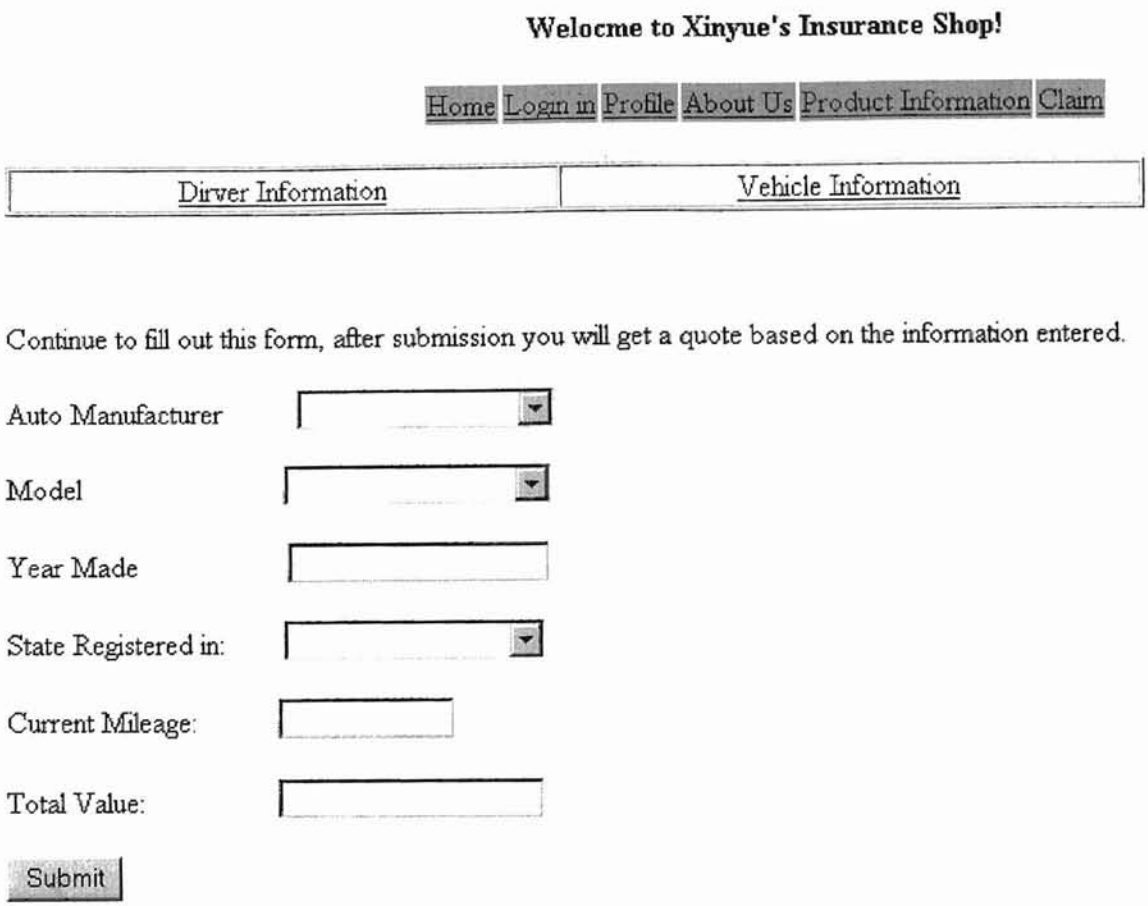


Figure 7. Interface2 for Auto Quotc

After submission of the Auto Quote forms, users will get an appropriate auto quote based on the information entered. At the same time, there are some helper classes for creating a unique quote id, assigning the id to the right quote, and computing the amount of the premium. If an end-user accept a quote, (s)he can obtain a complete online auto policy.

For health insurance and life insurance, there are some different interface or presentation design and implementation issues. First, the users can, if they so desire, view

the types of plan available and their detail, they can choose one of the available plan type, and subsequently fill out an application based on a quote. After the submission of a quote, the server responds to users by returning a specific policy.

3.3.3 Business Logic Design and Implementation

The encapsulation of business logic into business objects has become the most recent focus in the information technology industry [Monson-Haefel 99]. Enterprise beans are server-side component models used on middle-tiered application servers [Monson-Haefel 99]. A component model defines a set of interfaces and classes in the form of Java packages [Pawlan 01]. In the J2EE environment, there are two kinds of enterprise beans: entity beans and session beans [Pawlan 01]. During the implementation of the insurance application in this thesis, entity beans and session beans were used to represent the appropriate insurance products/services.

Entity bean: An entity bean represents a business object in a persistent storage mechanism [Pawlan 01]. Data is at the heart of most business applications. In J2EE applications, entity beans represent the business objects that are stored in a database.

To develop an enterprise bean, developers need to define two interfaces and one classe [Monson-Haefel 99] as explained below.

1) Remote Interface — It defines the bean's business methods [Monson-Haefel 99]. The business methods are the only methods that are visible to the client application; the other methods are visible only to the EJB container.

2) Home Interface — It defines the bean's life cycle methods [Monson-Haefel 99], which are methods for creating new beans, removing beans, and finding beans.

3) Bean Class — It actually implements the bean's business methods [Monso-Haefel 99]. Note that the bean class usually does not implement the bean's home or remote interfaces. However, the bean class must have methods matching the signatures of the methods defined in the remote interface and must have methods corresponding to some of the methods in the home interface.

4) Primary Key—it is a very simple class that provides a pointer into the database. Only entity beans need a primary key.

To understand how to best develop entity beans, it is important to understand how the container manages the entity beans. The EJB specification defines just the major event in an entity bean's life, from the time it is instantiated to the time it is garbage collected [Monson-Haefel 99]. Developers can implement these events using their own business logic. When a new bean is created, a new record must be inserted into the database and a bean instance must be associated with the data. As the bean is used and its state changes, these changes must be synchronized with the data in the database. To illustrate the whole process of an entity bean within an application server, the life cycle of an entity bean is displayed below in Figure 8.

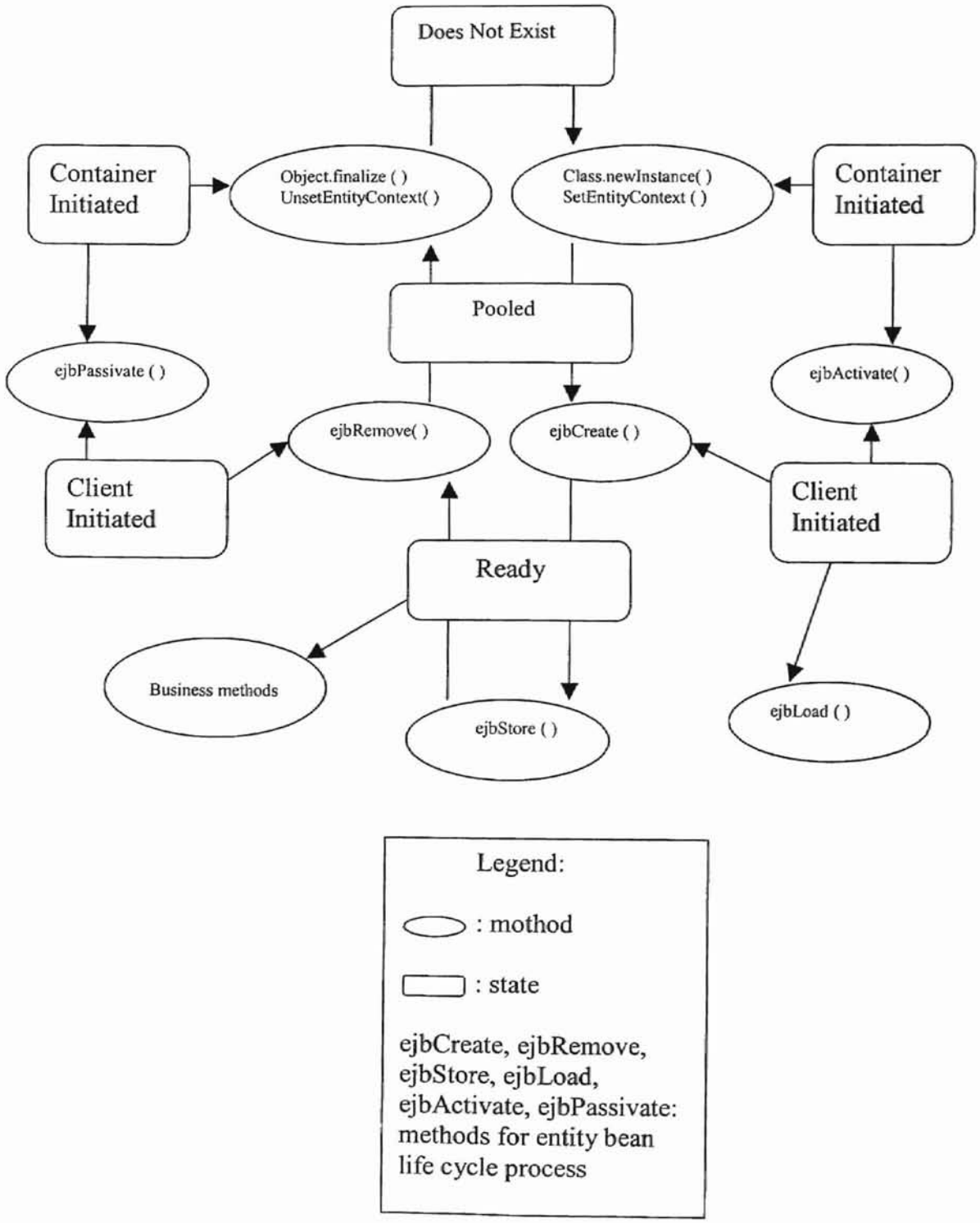


Figure 8. Entity Bean Life Cycle [Valesky 99]

In the insurance application in this thesis, there are the following entity beans: profileBean, autoQuoteBean, healthQuoteBean, lifeQuoteBean, autoPolicyBean, healthPolicyBean, lifePolicyBean, claimBean, and registrationBean, which represent specific persistent items stored in the relational database tables.

The profileBean was used for end-users to enter their personal profiles or change some records later.

For users registration, there is a login interface for users to enter user name and password.

For the three kinds of quote beans (autoQuoteBean, healthQuoteBean, and lifeQuoteBean), there is a specific interface for users to enter their respective information related to auto, health, and life insurance. After the submission of a request, the users can get an appropriate annual premium for the specific quote. If the end-users accept the quote, they may press the "continue" button and obtain an online insurance policy. Otherwise, the end-uses may return to the home page or exit the system.

The following are brief descriptions of the AutoQuoteBean interface and classes.

- AutoQuoteRemote (the remote interface): The AutoQuoteRemote interface describes what the bean does by declaring the user-defined methods that provide the business logic for the AutoQuoteBean, such as getDriverBean( ) and saveToDatabase( ) methods. These methods are the ones used by the client to interact with the bean over the remote connection. The AutoQuoteBean's name maps to the AutoQuote table by using the lookup service.

• AutoQuoteHome (the home interface): The AutoQuoteHome interface describes how the AutoQuoteBean is created, found, and removed from its container (i.e., the methods ejbCreate( ), ejbRemove( ), and ejbFindByPrimaryKey( )).

• AutoQuoteBean (the bean class): The AutoQuoteBean is the enterprise bean class. It implements the EntityBean interface, provides the business logic for the developer-defined methods, and implements the entityBean methods for creating the bean and setting the session context. This is the class that the bean developer needs to implement.

HealthQuoteBean, LifeQuoteBean, AutoPolicyBean, HealthPolicyBean, and LifePolicyBean are enterprise beans consist of the same kind of interfaces, classes, and database tables as the AutoQuoteBean, but the actual business logic implementation, database tables, and primary keys are different (see Appendix A).

Session bean: A session bean represents a single client inside the J2EE server [Pawlan 01]. Unlike the entity beans, session beans do not represent shared data in the database, but they can access the shared data. A session bean performs tasks on behalf of a client and maintains the state related to the client [Pawlan 01]. To access an application that is deployed on the server, the client invokes the session bean's methods. The session bean performs work for its client, shielding the client from complexity by executing business tasks inside the server.

The end-user clients (Web clients and J2EE application clients) access only the session beans [Monson-Haefel 99]. Within the enterprise bean tier, the session beans are clients of the entity beans. On the back-end of the application, the entity beans access the database tables that store the entity states. The Session beans have a home interface, a

remote interface, and bean classes. But the bean classes implement the Session interface in the EJB specification.

In this thesis application, there are three kinds of session beans corresponding to the three quoted entity beans: AutoQuoteControllerBean, HealthQuotecontrollerBean, and LifeQuoteControllerBean. Also there are claimSessionBean and loginSessionBean for end users doing online claim and login. Figure 9 below depicts how the Web components (including the servlets and JSPs) and all enterprise beans interact within the J2EE container.

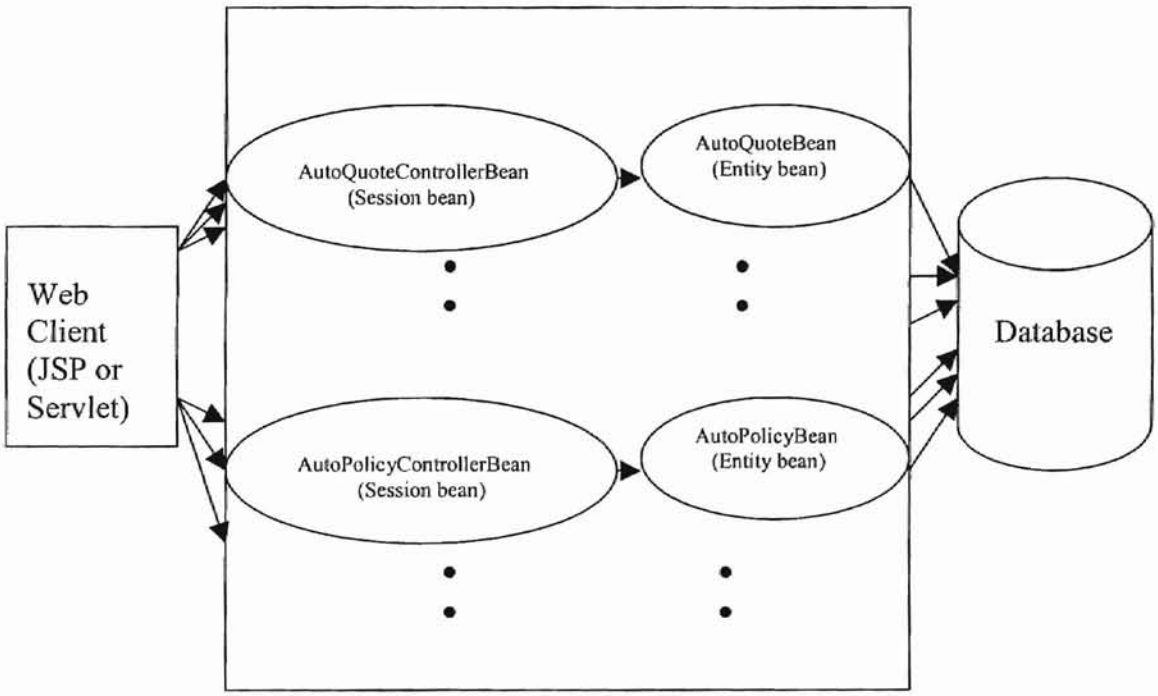


Figure 9. Interaction Between Servlet and EJBs

RMI: Java RMI API enables the client and the server to communicate method invocations over a network using the RMI protocol. In a three-tiered architecture, the

RMI includes three parts: the object server, the skeleton, and the stub [Monson-Haefel 99]. The object server is the business object that resides on the middle-tier [Farley 98]. Every object server class has a matching skeleton and a stub class build specifically for that type of object server [Monson-Haefel 99]. The skeleton is set up on a port and an IP address, and it listens for requests from the stub. The stub resides on the client machine and is connected via the network to the skeleton. The stub acts as the object server's surrogate on the client and is responsible for communicating requests from the client to the object server through the skeleton [Pawlan 01]. In the J2EE environment, the client program can reference the enterprise beans running on the server and access them as if they were running locally on the client program [Monson-Haefel 99]. To transfer objects, the RMI API uses the Serialization API to wrap (marshal) and unwrap (unmarshal) the objects. . To marshal an object, the Serialization API converts the object to a stream of bytes, and to unmarshal an object, the Serialization API converts a stream of bytes into an object [Faley 98]. The data marshaling process in Java RMI is depicted below in Figure 10.

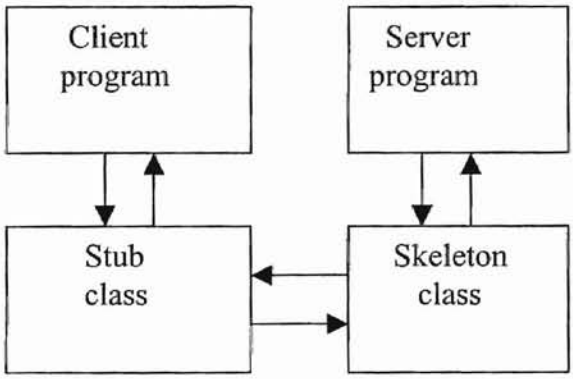


Figure 10. Data Marshaling in RMI

3.3.3 Database Management System Design and Implementation

For the database system of the insurance application in this thesis, a relational database was used to store all the business logic data and the internal data for processing. The Data Definition Language (DDL) was used for establishing all the tables. The following list (List 2) display the schema for AutoQuote.

List 2. DDL for creating AutoQuote tables

```
CREATE table AutoQuote {
        Quoteid             char (12) not null,
        Firstname           varchar (30),
        Middlename          varchar (30),
        Lastname            varchar (30),
        Emailaddress        varchar (50),
        DateOfBirth         varchar (12)
        LicenseStatus       integer ( ),
        DateOfLicensed      varchar (12),
        DriverOccupation    varchar (20),
        Manufacturer        varchar (30),
        Model               varchar (30),
        Madeyear            varchar (12),
        RegisteredState     varchar (30),
        Totalvalue          float ( ),
        Primarykey          quoteid
}
```

In the insurance application in this thesis, there are many tables in the database system to store all the necessary information. Normally, the relationships among these tables are complicated. Among one-to-one, one-to-many, many-to-one, and many-to-many relationships, the one-to-one relationship was adopted in this application because of its simplicity.

University Library

Figure 11 below is the outline of database tables and their relationships (i.e., the relation schemas and their relationships) used in the insurance application in this thesis.

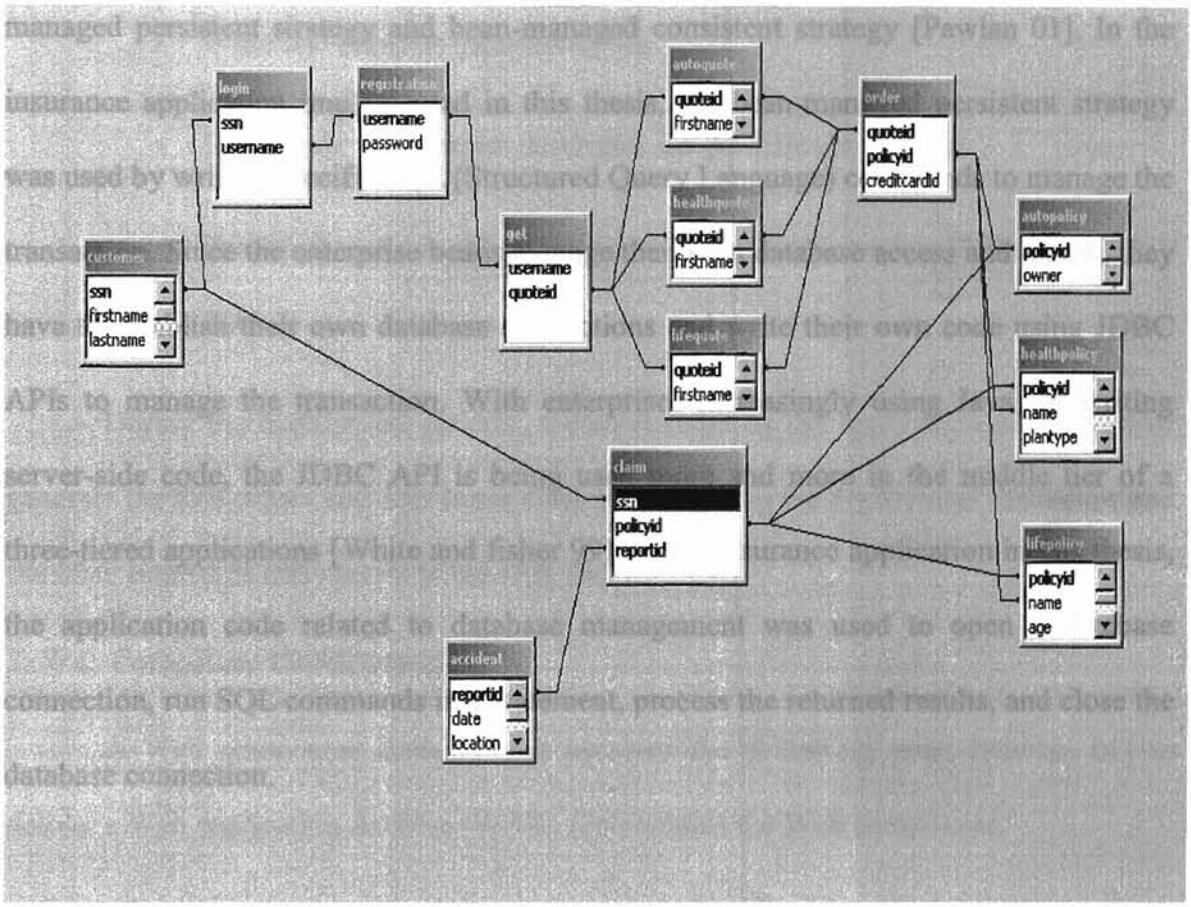


Figure 11. Relational Database Design

According to the data requirements of the insruance system implemented in this thesis, the following entities and relationships are represented by the relational tables. The entities include customer, registration, autoquote, healthquote, lifequote, autopolicy, healthpolicy, lifepolicy, and accident. The relationships include login (used to connect customer and registration), get (used to connect registration and autoquote, healthquote,

University Library

and lifequote), order (used to connect quote and policy), and claim (used to connect customer, policy, and accident).

In the J2EE environment, there are two data management strategies: container-managed persistent strategy and bean-managed consistent strategy [Pawlan 01]. In the insurance application implemented in this thesis, the bean-managed persistent strategy was used by writing specific SQL (Structured Query Language) commands to manage the transaction. Since the enterprise beans manage their own database access and search, they have to establish their own database connections and write their own code using JDBC APIs to manage the transaction. With enterprises increasingly using Java for writing server-side code, the JDBC API is being used more and more in the middle tier of a three-tiered applications [White and fisher 99]. In the insurance application in this thesis, the application code related to database management was used to open a database connection, run SQL commands in a statement, process the returned results, and close the database connection.

3.3.5 Packaging and Deployment

The J2EE components in the insurance application implemented in this thesis were packaged separately and bundled into a J2EE application for deployment. Each component, its related files such as GIF and HTML files or server-side utility classes (helper classes), and a deployment descriptor, were assembled into a module and added to the J2EE application. Each J2EE application is composed of one or more enterprise beans, and Web or application client component modules.

University Library

1) Enterprise Bean Development

An enterprise bean development involves the following specific tasks [Pawlan 01] to deliver an EJB Java Archive (JAR) file that contains the enterprise bean.

- Write and compile the source code.
- Specify the deployment descriptor for the EJB component.
- Bundle the `.class` files and deployment descriptor into EJB JAR files.

During the deployment process, tools provided by the application server vendor generate the EJB object and EJB home class by examining the deployment descriptor and the other interfaces and classes in the JAR file.

2) Web Component Development

A Web component development involves the following tasks [Pawlan 01] to deliver a Web application Archive (WAR) file contains the Web component.

- Write and compile servlet source code.
- Write JSP and HTML files.
- Specify the deployment descriptor for the Web component.
- Bundle the `.class`, `.jsp`, `.html`, and deployment descriptor into the WAR file.

University Library

As a summary, the entire process to develop an enterprise application in the J2EE environment can be outlined as the flow chart in Figure 12 below.

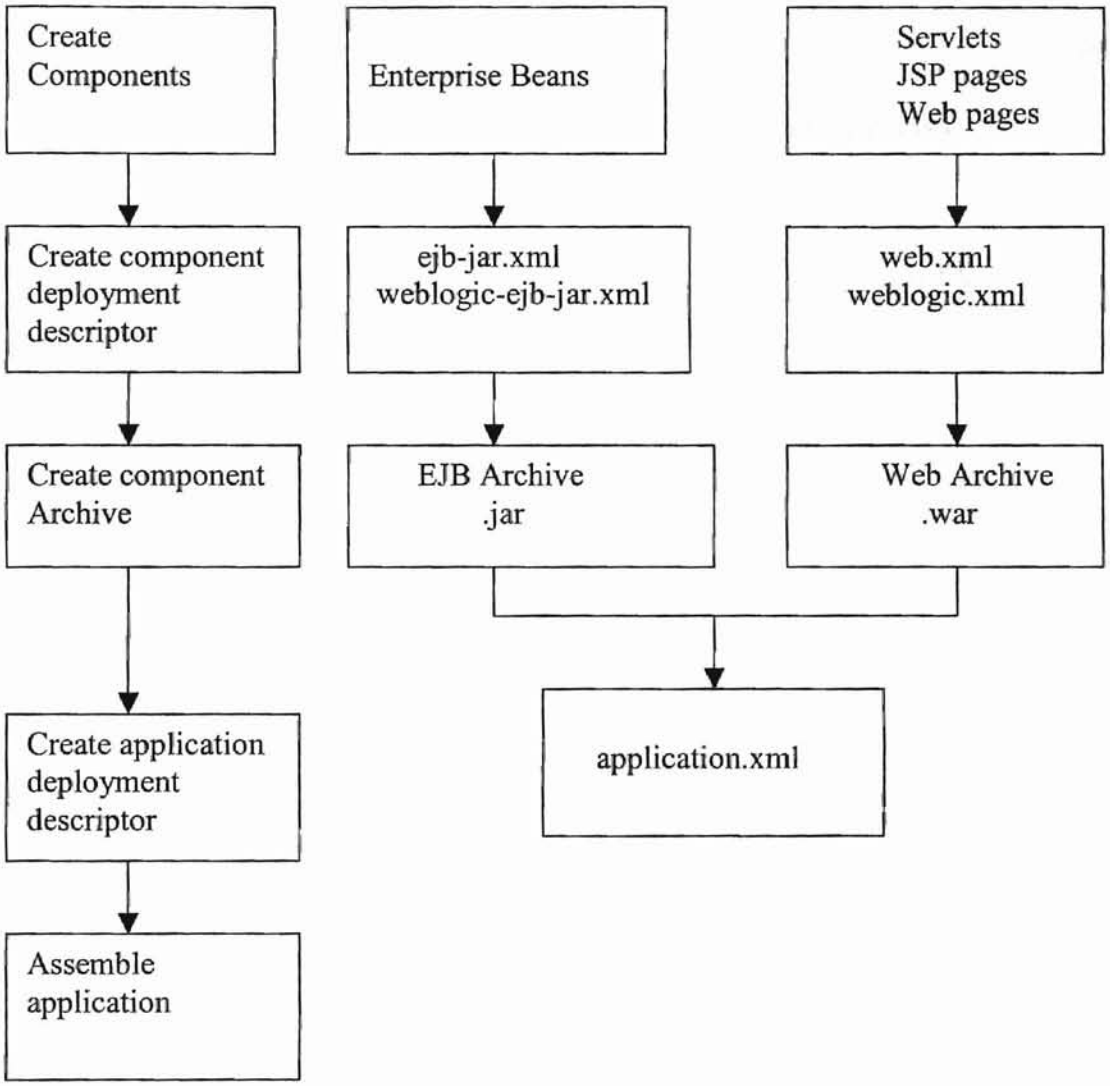


Figure 12. Creating Enterprise Applications [BEA 01]

Oklahoma State University Library

The Apache ant [Jakarta 01] was used as a tool for building and packaging all classes in the insurance application in this thesis. The Apache ant is a build tool that uses the Java technology to perform all built-related tasks. Ant is somewhat like "make" without make's wrinkles [Jakarta 01]. Instead of writing shell commands, the configuration files are XML-based documentation [Jakarta 01]. Developers create XML-based built files that describe the project built process. It has the ability to across platforms. In the insurance application in this thesis, there are four .xml files as listed below.

1) The "build.xml" file was used for compiling all java classes and building .jar files.

2) The "ejb-jar.xml" file was used for describing enterprise beans and deploying them.

3) The "weblogic.xml" file was used for the application server connecting various enterprise beans.

4) The "web.xml" file was used for servlets and JSP mapping.

The format of an XML file is displayed below in List 4.

Oklahoma State University Library

List 4. Sample code for the ejb-jar.xml file

(For one entity bean and one session bean)

```
<?xml version="1.0"?>

<!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>

      <entity>

            <ejb-name>autoquotebean</ejb-name>

            <home>com.ejb.server.quote.AutoQuoteHome</home>

            <remote>com.ejb.server.quote.AutoQuote</remote>

            <ejb-class>com.ejb.server.quote.AutoQuoteBean</ejb-class>

            <persistence-type>Bean</persistence-type>

            <prim-key-class>java.lang.String</prim-key-class>

            <reentrant>False</reentrant>

            <resource-ref>

            <res-ref-name>jdbc/demoPool</res-ref-name>

            <res-type>javax.sql.DataSource</res-type>

            <res-auth>Container</res-auth>

            </resource-ref>

      </entity>

      <session>

            <description>no desc</description>

            <display-name>autoquoteController</display-name>

            <ejb-name>autoquoteController</ejb-name>

            <home>com.ejb.server.quote.AutoQuoteControllerHome</home>

            <remote>com.ejb.server.quote.AutoQuoteController</remote>
```

Oklahoma State University Library

```
        <ejb-
        class>com.ejb.server.quote.AutoQuoteControllerBean</ejb-

        class>

        <session-type>Stateful</session-type>

        <transaction-type>Container</transaction-type>

    </session>

    </ejb-jar>
```

Oklahoma State University Library

CHPATER IV

APPLICATION TESTING AND EVALUATION

All of the components that were developed for the insurance application is this thesis were put together following a well-defined deployment process. The precedure and result of running the three-tiered online application system in the J2EE environment is presented in this chapter.

4.1 Running and Testing the Application

The following steps shortly describe how to run the insurance application implemented in this thesis.

1) Set the CLASSPATH environment variable

In the J2EE environment, the CLASSPATH environment variables are needed to specify the application's own classes only, i.e., not including the utility classes.

2) Start the application server and the database server

The BEA WebLogic server was used as the application server to run and test this project. Before starting the application server, developers need to designate parameters

University Library

like the port number on which the server listens and the directories in which for the client side files.

3) Run the whole application system

The insurance application in this thesis provides three kinds of products/services: auto insurance, health insurance, and life insurance. End-users can use these services. For example, if users want to a health insurance policy quote, they can first compare some listed plan types (Figure 13) and then choose one of them.

After pressing the "Apply" button, you can get a health insurance quote.

Plan Name	Premium (monthly)	Plan Type	Deductable	Coinsurance	Copay	Your Choice
BasicPPO	$28.00	PPO	$1000	20%	N/A	◉
ValuePPO	$31.00	PPO	$1000	25%	N/A	○
PPOSaver	$44.00	PPO	$5000	20%	N/A	○
EPO	$53.00	MSA	$2400	50%	N/A	○
PreferedSavingPlan	$83.00	MSA	$1650	20%	N/A	○
HMO40	$145.00	HMO	$2000	0%	$40	○
HMOSaver	$175.00	HMO	$1500	0%	$10	○
HMOPlan 10	$196.55	HMO	$0	0%	$20	○
POSPlan	$246.12	POS	$500	20%	N/A	○

Apply

Notices:

. The monthly premium amounts shown are subject to change based on your medical histroy, the underwriting practices of the health plan, the optional benefits you selected, if any, and other relevant factors.

. The copayment , deductable, and coinsurance amounts are your share of the costs for the covered benefits.

Figure 13. HealthQuote1

University Library

After choosing a health plan type, the users need enter their personal information through the interface as depicted below in Figure 14.

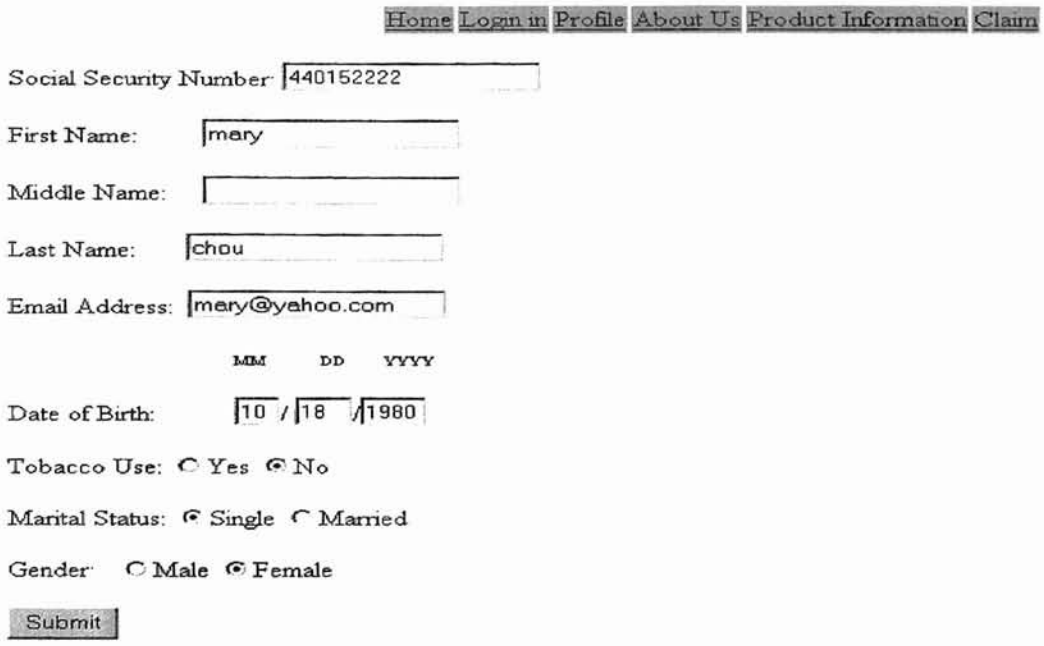


Figure 14. HealthQuote2

After submitting of the forms, the server will return a unique quote number and the monthly premium (Figure 15) for the specific plan type. There is a helper class that is used to determine the actual monthly premium based on a standard one.

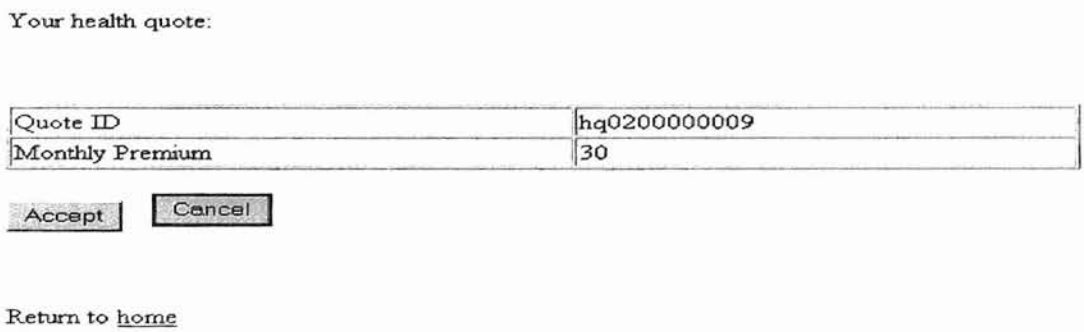


Figure 15. Returned QuoteID and Right Premium

Oklahoma State University Library

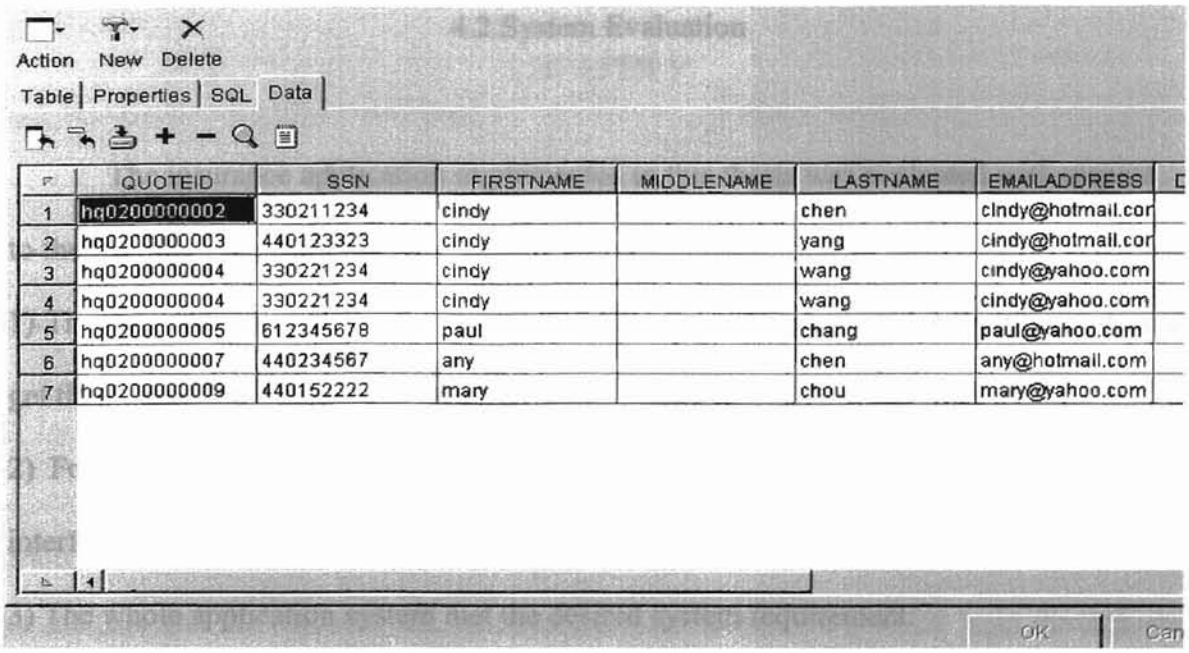
If a user accepts the quote, the server will respond by assigning a unique policy number and returning a health quote policy (Figure 16) based on the information provided.

Home Login in Profile About Us Product Information Claim

Social Security Number	440152222
First Name	mary
Middle Name	
Last Name	chou
Email Address	mary@yahoo.com
Date of Birth	1980-10-18
Gender	F
Tobacco Use	Not a tobacco user
Marital Status	Single
Plan Name	BasicPPO
Standard Monthly Premium	28.0
Quoted Monthly Premium	30
Plan Type	PPO
Deductable	1000
Coinsurance	0%
Copay	N/A
Policy ID	hp0200000014
Quote ID	hq0200000009

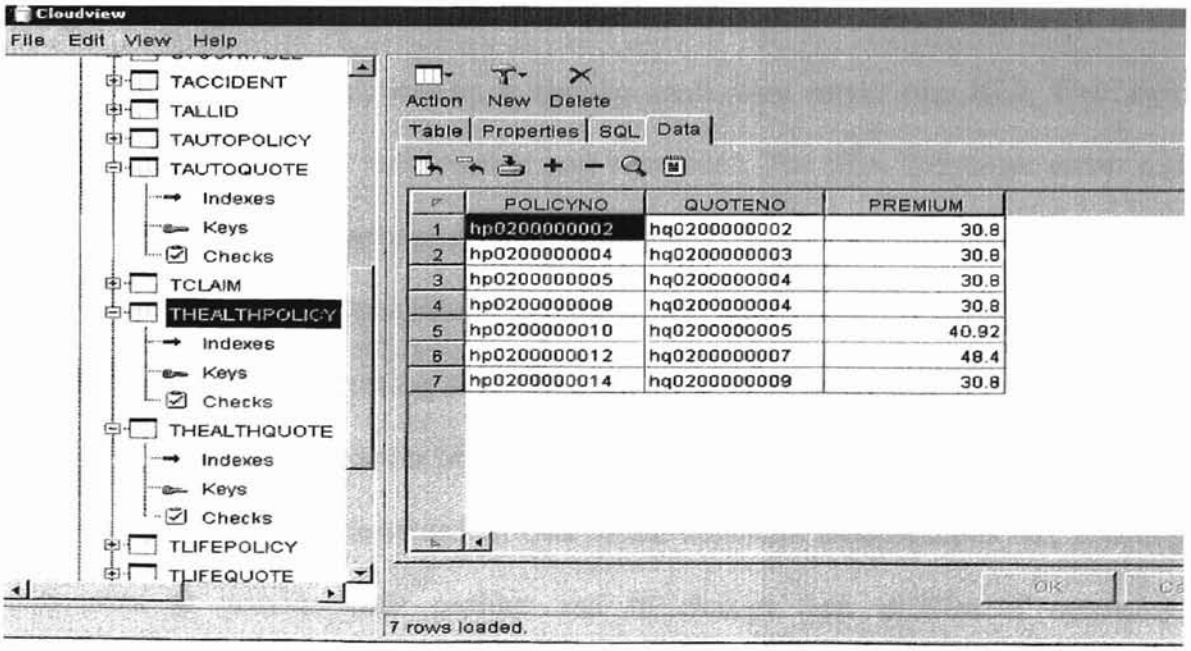
Figure 16. An Online Health Quote Policy

At the same time, all the information is added to the database system. Sample HealthQuote table and HealthPolicy table are given below in Figure 17 and 18.



	QUOTEID	SSN	FIRSTNAME	MIDDLENAME	LASTNAME	EMAILADDRESS
1	hq0200000002	330211234	cindy		chen	cindy@hotmail.cor
2	hq0200000003	440123323	cindy		yang	cindy@hotmail.cor
3	hq0200000004	330221234	cindy		wang	cindy@yahoo.com
4	hq0200000004	330221234	cindy		wang	cindy@yahoo.com
5	hq0200000005	612345678	paul		chang	paul@yahoo.com
6	hq0200000007	440234567	any		chen	any@hotmail.com
7	hq0200000009	440152222	mary		chou	mary@yahoo.com

Figure 17. HealthQuote table



	POLICYNO	QUOTENO	PREMIUM
1	hp0200000002	hq0200000002	30.8
2	hp0200000004	hq0200000003	30.8
3	hp0200000005	hq0200000004	30.8
4	hp0200000008	hq0200000004	30.8
5	hp0200000010	hq0200000005	40.92
6	hp0200000012	hq0200000007	48.4
7	hp0200000014	hq0200000009	30.8

Figure 18. HealthPolicy table

4.2 System Evaluation

The insurance application implemented in this thesis was evaluated with respect to the following aspects.

1) The thin-client side acted as the user interface for end-users to enter their requests and get their responses fast and correctly.

2) For the business logic side and the database tier, they cooperated well and didn't interfere with each other.

3) The whole application system met the desired system requirement.

CHPATER V

SUMMARY AND FUTURE WORK

5.1 Summary

In this thesis work, an online three-tiered insurance application system was developed in the J2EE environment. In Chapter II, a brief review of the existing techniques and the current J2EE technologies were given. Chapter III went through the conventional phases of the software development process based on the three-tiered application model. For every relevant J2EE technology, a brief description, a model design, and an actual implementation were mentioned and discussed in Chapter III. In the implementation of the application, a specific application server (the BEA WebLogic server) and a deployment tool (Apache ant) were used. The BEA WebLogic server and the Apache ant are standard commercial development tools; their configurations consist of a large amount of documentations for the learner to correctly use them step-by-step. Chapter IV discussed the testing and evaluation of the application. Based on the feedback obtained from several instances of system demostration, a number of modifications were made to the original implementation. One of the important modifications was enabling end users to provide their profiles and to change their profiles if necessary.

5.2 Future Work

In the fast moving and changing technology in the J2EE environment, more research and study need to be done. First, a new tag library can be used in JSP to simplify the dynamic process of a client's request and a server's response. Second, a message-driven bean can be used in Java Message Service (JMS) so that messages can be sent to and received from software components in the J2EE environment.

REFERENCES

[Alur and Crupi 01] Depark Alur and John Crupi, *Core J2EE Patterns*, Sun Microsystems, Inc., Palo Alto, CA, 2001.

[Asbury and Weiner 99] Stephen Asbury and Scott R. Weiner, *Developing Java Enterprise Applications*, John Wiley & Sons, Inc., New York, NY, 1999.

[Austin and Pawlan 00] Calvin Austin and Monica Pawlan, *Advanced Programming for the Java2 Platform*, Sun Microsystems, Inc., Palo Alto, CA, 2000.

[BEA 01] "BEA WebLogic Server 6.1 J2EE Compliance", http://www.bea.com/products, and http://e-docs.bea.com/wls/docs61/notes/, Creation Date: July 2001, Access Date: December 2001.

[Berg 98] Clifford J. Berg, *Advanced JAVA Development for Enterprise Applications*, Prentice-Hall, Inc., Upper Saddle River, NJ, 1998.

[Berg and Fritzinger 99] Daniel J. Berg and J. Steven Fritzinger, *Advanced Techniques for Java Developers*, John Wiley & Sons, Inc., New York, NY, 1999.

[Callaway 99] Dustin R. Callaway, *Inside Servlets*, Addison Wesley Longman, Inc., Reading, MA, 1999.

[Casto 98] Jenifer Casto, *JavaBeans Programming*, Brandon A. Nordin, Berkeley, CA, 1998.

[Farley 98] Jim Farley, *Java Distributed Computing*, O'Reilly & Associates, Inc., Sebastopol, CA, 1998.

[Hoque 99] Reaz Hoque, *CORBA for Real Programmers*, Morgan Kaufmann Inc., San Diego, CA, 1999.

[Hunter 98] Jason Hunter, *Java Servlet Programming*, O'Reilly &Associates, Inc., Sebastopol, CA, 1998.

[Jakarta 01] "The Jakarta Project-Apache Ant", http://jakarta.apache.org/builds/jakarta,

Creation Date: October 2001, Access Date: December 2001.

[Learning 00] "Learning Center", http://www.insurance.com/profiles_insights, Creation Date: February 2000, Access Date:September 2001.

[Monson-Haefel 99] Richard Monson-Haefel, *Enterprise JAVA BEANS*, O'Reilly & Associates, Inc., Sebastopol, CA, 1999.

[Moss 98] Karl Moss, *Java Servlets*, McGraw-Hill Companies, Inc., New York, NY, 1998.

[Oaks and Wong 97] Scott Oaks and Henry Wong, *Java Threads*, O'Reilly & Associates, Inc., Sebastopol, CA, 1997.

[Pawlan 01] Monica Pawlan, "The J2EE Tutorial", http://java.sun.com/j2ee/tutorial/overview.html, Creation Date: October 2001, Access Date: November 2001.

[Pekowsky 00] Larne Pekowsky, *Java Server Pages*, Addison Wesley Longman, Inc., Reading, MA, 2000.

[Pritchard 99] Jason Pritchard, *COM and CORBA Side by Side*, Addison Wesley Longman, Inc., Reading, MA, 1999.

[Rosen and Curtis 98] Michael Rosen and David Curtis, *Integrating CORBA and DCOM Applications,* John Wiley & Sons, Inc., New York, NY, 1998.

[Sadoski 97] Darleen Sadoski, "Client/Server Software Architecture", http://www.sei.cmu.edu/str/descriptions/clientserver.html, Creation Date: August 1997, Access Date: December 2001.

[Valesky 99] Tom Valesky, *Enterprise Java Beans,* Addison Wesley Longman, Inc., Reading, MA, 1999.

[White and Fisher 99] Seth White and Maydene Fisher, *JDBC API Tutorial and Reference*, Sun Microsystems, Inc., Palo Alto, CA, 1999.

GLOSSARY

Ant	A free, open-source implementation of Java Servlets and Java Server Page technologies developed under the Jakarta project at the Apache Software Foundation.
API	Application Program Interface, a set of routines, protocols, and tools for building software applications.
AWT	Abstract Window Toolkit, part of the Java foundation classes, the standard API for providing graphical user interfaces for Java programs.
CGI	Common Gateway Interface, programs which enable developers to create separate applications that communicate with a Web server.
COM	Component Object Model, a software architecture that allows applications to be built from software components. COM refers to both specification and implementation developed by Microsoft Corporation which provides a framework for integrating components.
CORBA	Common Object Request Broker Architecture, a standard framework for building distributed object systems developed by members of the Object Management Group (OMG) and their corporate members.
DCOM	Distributed Component Object Model, an extension to COM that allows networking- based component interaction.
DDL	Data Definition Language, a language used to create and destroy databases and database objects.
EJB	Enterprise JavaBean, a standard way to create server-side components in distributed object-oriented applications.
Entity Bean	An entity bean represents a business object in a relational database.
HTML	Hyper Text Markup Language, a markup language which consists of tags embedded in the text of a document.

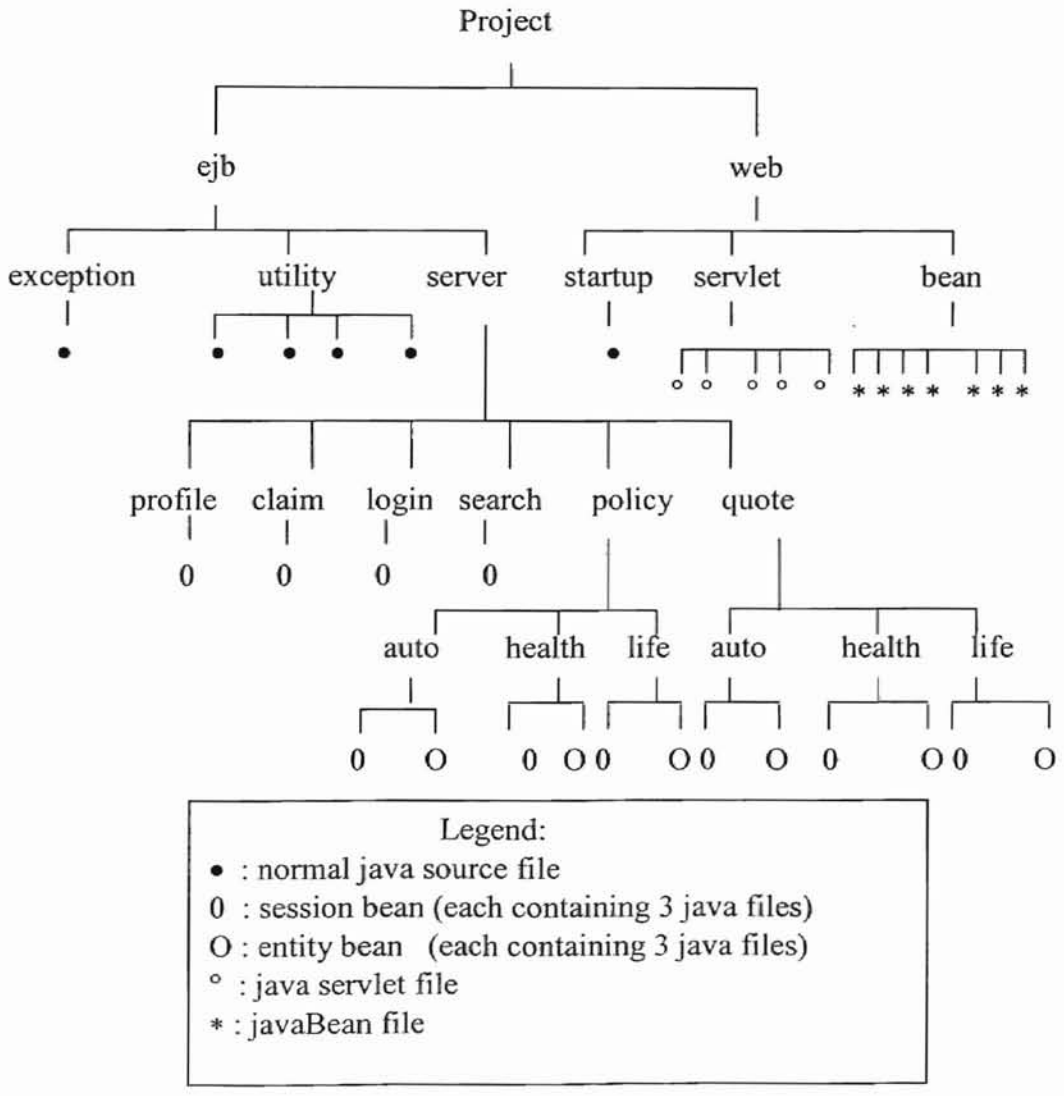
HTTP	Hyper Text Transfer Protocol, an application-level protocol for distributed, collaborative, hypermedia information systems.
IDL	Interface Definition Language, technology-independent syntax for describing object encapsulations. It defines interfaces, thus creating objects.
IIOP	Internet Inter-ORB Protocol, a protocol developed by the Object Management Group to implement CORBA solutions over the World wide Web. IIOP enables browsers and servers to exchange integers, arrays, and more complex objects, unlike HTTP, which only supports transmission of text.
J2EE	Java2 Enterprise Edition, a standard application model for developing multi-tiered enterprise systems developed by Sun Microsystems, Inc.
JAR	Java ARchive, a file format that enables java developer to bundle multiple files into a single archive file.
JavaBeans	Software components that can be used in virtual programming environments.
JDBC	Java DataBase Connectivity, a set of libraries used to connect Java application programs with relational or other databases.
JNDI	Java Naming and Directory Interface, an enabling technology used for accessing and manipulating enterprise resources.
JMS	Java Message Service, an API for accessing enterprise messaging systems.
JSP	Java Server Page, a technique that uses HTML-like tags written in the Java language to encapsulate the logic that generates the content for the page.
Multi-Threading	The ability of a program or a process to manage its use by more than one user at a time and even to manage multiple requests by the same user.
Multiple-Single-Threaded	Multiprocessing, the coordinated processing of programs by more than one computer processor.

ODBC	Open DataBase Connectivity, an Application Programming Interface that allows a programmer to abstract programs from a database.
ORB	Object Request Broker, a component that provides the entire communication infrastructure needed to identify and locate objects in the CORBA technology.
RMI	Remote Method Invocation, a technique for distributed systems which provides remote communication between programs written in Java.
Session Bean	A session bean is an extension of the client application and is responsible for managing processes or tasks.
SQL	Structured Query Language, a databases query language that was adopted as an industry standard.
WAR	Web Archive, a JAR similar to the package used for Java class libraries. A WAR usually contains Web components.
XML	Extensible Markup Language, the universal format for structured document and data on the Web.
UML	Unified Modeling Language, a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and non-software systems.

APPENDIX A

CODE FOR THE BUSINESS LOGIC AND DATABASE TIER

The implementation of the business logic and the database tier was discussed is Chapter III. What follows is the list of all source files implemented in Java. The structure of the implementation is depicted below.



The total number of source files is 66. The total number of lines in these source files is approximately 7,000. The list of the selected program listings in this appendix follows.

EJBs (each EJB contains a remote interface, a home interface, and a bean class.)
ClaimBean
LoginBean
AutoPolicyBean
AutoPolicyControllerBean
AutoQuoteBean
AutoQuoteControllerBean

JavaBeans
ClaimBean.java

Utility classes
DPHelper.java
DateHelper.java
StaticNames.java
EjbGetter.java
Startup.java

```
/**
* @author xinyue zhu
* @date 10/01/2001
*/


/** Code for claimBean, this is a session bean.
* It contains a remote interface, a home interface,
* and a bean class.
*/

package com.ejb.server.claim;

im import java.rmi.*;
import com.web.beans.*;

public interface ClaimSession extends EJBObject {
    public String saveClaimToDB(ClaimBean claimbean)
            throws RemoteException;
}

import javax.ejb.

 /** If there is an accident, the end-user can file an online claim
* through the insurance application system. After submission of the
* claim form,  users will get some response, and the entered
* information will be stored into the database system.
*/
```

```
package com.ejb.server.claim;
import java.sql.*;
import javax.sql.*;
import java.util.*;
import java.math.*;
import javax.ejb.*;
import javax.naming.*;
import java.rmi.RemoteException;

import com.ejb.util.*;
import com.ejb.server.*;
import com.ejb.server.quote.*;
import com.ejb.exception.*;
import com.web.beans.*;



public class ClaimSessionBean implements SessionBean {

    private SessionContext ctx;
    private Connection conn;

    public ClaimSessionBean() {
    }
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbRemove()  { }
    public void setSessionContext(SessionContext context) throws
javax.ejb.EJBException, java.rmi.RemoteException {
        this.ctx = context;}
    public void unsetSessionContext() throws javax.ejb.EJBException,
java.rmi.RemoteException {
        this.ctx = null;
    }
public void ejbCreate() throws CreateException {}

public String saveClaimToDB(ClaimBean claim) throws RemoteException{
        makeConnection();
        StringBuffer buf = new StringBuffer("cr");
        buf.append(DateHelper.getLast2DigitOfYear());
        buf.append(DBHelper.getInstance().getNextReportNo());
        try {
            insertRow(buf.toString(), claim);
        }
        catch (Exception ex) {
            System.out.println("Inserting claim problem: " +
ex.getMessage());
            ex.printStackTrace();
        }
        releaseConnection();

        return buf.toString();
    }
```

```
                    /* Database Connection */

/** There is a claim table in the database server corresponding to the
* claimBean in the middleware, and an interface on the client side.
* There are the following attributes in the claim table:
* claimed, Firstname, Middlename, Lastname, PolicyNo, Emailaddress,
* Accident address (street, city, state, zip), Accident date,
* and  brief desciption of the accident.
*/

private void makeConnection() {
        System.out.println(" Before make connection in claim session
...");
        try {
            InitialContext ic = new InitialContext();
            DataSource ds = (DataSource)
ic.lookup(StaticNames.THE_DATABASE);
            conn =  ds.getConnection();
        } catch (Exception ex) {
             throw new EJBException("Unable to connect to database. " +
                 ex.getMessage());
        }
        System.out.println("End make connection in claim session ");
    }
    private void releaseConnection() {
        System.out.println("Releasing the connection for claim
session... ");
       try {
            conn.close();
        } catch (SQLException ex) {
             throw new EJBException("releaseConnection: " +
ex.getMessage());
        }}
    private void insertRow(String claimno, ClaimBean bean) throws
SQLException {
        System.out.println(bean.getEmail() + "is being claimed");

/** SQL statement: insertion */
        String insertStatement =
            "insert into tclaim (reportid, firstname, middlename,
lastname, " +
                                "policyno, email, acdate, street, " +
                                "city, state, zip, description)" +
                                " values (?,?,?,?,?,?,?,?,?,?,?,?)";
        PreparedStatement prepStmt =
            conn.prepareStatement(insertStatement);

        prepStmt.setString(1, claimno);
        prepStmt.setString(2, bean.getFirstname());
        prepStmt.setString(3, bean.getMiddlename());
        prepStmt.setString(4, bean.getLastname());
        prepStmt.setString(5, bean.getPolicyno());
        prepStmt.setString(6, bean.getEmail());
        prepStmt.setString(8, bean.getStreet());
        prepStmt.setString(9, bean.getCity());
        prepStmt.setString(10, bean.getState());
        prepStmt.setString(11, bean.getZip());
```

```
        prepStmt.setString(12, bean.getDesc());

        java.sql.Date date = DateHelper.getDate(
                                bean.getAcY(),
                                bean.getAcM(),
                                bean.getAcD());
        prepStmt.setDate(7, date);

        prepStmt.executeUpdate();

        System.out.println("OK");

        prepStmt.close();
    }}
package com.ejb.server.claim;

import javax.ejb.*;
import java.rmi.*;

public interface ClaimSessionHome extends EJBHome {
    public ClaimSession create() throws RemoteException,
CreateException}; }



/** Code for login enterprise bean.
*/

/** This EJB was created for end-users to login the insurance
* application system. If users are new to this system, there is
* a registration process. End-users need to enter their userID,
* password and email address. If necessary, users can change their
* passward.
*/


package com.ejb.server.login;

import java.rmi.RemoteException;
import javax.ejb.EJBObject;

public interface Login extends EJBObject {

//public void getEmailAddress(String emailAddr) throws RemoteException;

public String getPassword() throws RemoteException;

//public void changePwd(String pwd) throws RemoteException;

}
package com.ejb.server.login;

import java.io.Serializable;
import java.sql.*;
import java.util.*;

import javax.ejb.*;
```

```
import javax.naming.*;
import javax.sql.*;

public class LoginBean implements EntityBean {

    private final static boolean VERBOSE = true;
    private String myname;
    private EntityContext ctx;
    private String password;
   // private String emailaddr;

    public void setEntityContext(EntityContext ctx) throws
javax.ejb.EJBException, java.rmi.RemoteException {
        this.ctx = ctx;
        log("setEntityContext is being called");
    }


    public void unsetEntityContext() throws javax.ejb.EJBException,
java.rmi.RemoteException {
        log("unsetEntityContext: " + myuser() );
        this.ctx = null;
    }

/** The life cycle methods of an entity bean (see Figure 8)
* These methods mostly handled by the server.
*/

public void ejbActivate() throws javax.ejb.EJBException,
java.rmi.RemoteException {
        log("ejbActivate -- " + myuser() );
    }
    public void ejbPassivate() throws javax.ejb.EJBException,
java.rmi.RemoteException {
        log("ejbPassivate -- " + myuser() );
    }

    public void ejbPostCreate(String uname, String pwd) throws
javax.ejb.EJBException, java.rmi.RemoteException {
        log("ejbPostCreate == " + myuser() );
    }


/**  The server created a bean instance, then handled the database
* connection.
*/

 public String ejbCreate(String uname, String pwd) throws
CreateException{
        log("LoginBean.ejbCreate(username " + myname + " pwd: " +
password);
        this.myname = uname;
        this.password = pwd;

        Connection conn = null;
        PreparedStatement ps = null;
```

```
        try {
            conn = getConnection();
            ps = conn.prepareStatement(
            "Insert into tlogin (username password) values(?,?)");
            ps.setString(1, myname);
            ps.setString(2, password);

            if( ps.executeUpdate() != 1){
                String error = "JDBC did not create any row!";
                log(error);
                throw new CreateException(error);
            }
            return myname;
        }
        catch (SQLException ex) {
            try {
                ejbFindByPrimaryKey(uname);
            }
            catch (ObjectNotFoundException ex2) {
                String error = "SQLException: " + ex2;
                log(error);
                throw new CreateException(error);
            }
            catch (FinderException fe)
            {
                String error = "FInder Exception: " + fe;
                log(error);
                throw new CreateException(error);
            }
            String error = "An user already exist in the database ...";
            log(error);
            throw new DuplicateKeyException(error);
        }
        finally {
            cleanup(conn, ps);
        } }
    public void ejbRemove() throws javax.ejb.RemoveException,
javax.ejb.EJBException, java.rmi.RemoteException {
        log("ejbRemove: " + myuser() );

        Connection conn = null;
        PreparedStatement ps = null;

        try {
            conn = getConnection();
            ps = conn.prepareStatement("delete from tlogin where
username=?");
            ps.setString(1, myname);

            if(!( ps.executeUpdate()>0) ) {
                String error = "Error: LoginBean: " + myname + " Not
found";
                log(error);
                throw new NoSuchEntityException(error);
            }
        }
        catch (SQLException ex) {
```

```
            log("SqlException: " + ex);
            throw new EJBException(ex);
        }
        finally {
            cleanup(conn, ps);
        }
    }


    public void ejbLoad() throws javax.ejb.EJBException,
java.rmi.RemoteException {
        log("ejbLoad() -- " + myuser() );

        Connection conn = null;
        PreparedStatement ps = null;
        myname = (String) ctx.getPrimaryKey();

        try {
            conn = getConnection();
            ps = conn.prepareStatement("SELECT PASSWORD FROM TLOGIN
where username=?");
            ps.setString(1, myname);
            ps.executeQuery();
            ResultSet rs = ps.getResultSet();
            if( rs.next() ) {
                password = rs.getString(1);
               // emailaddr = rs.getString(2);
            }
            else {
                String error = "ejbLoad: LoginBean(" + myname + ") not
found";
                log(error);
                throw new NoSuchEntityException(error);
            }
        }
        catch (SQLException sqe) {
            log("SQLException: " + sqe);
            throw new EJBException(sqe);
        }
        finally {
            cleanup(conn, ps);
        }
    }


    public void ejbStore() throws javax.ejb.EJBException,
java.rmi.RemoteException {
        log("ejbStore: " + myuser() );

        Connection conn = null;
        PreparedStatement ps = null;

        try {
            conn = getConnection();
            ps = conn.prepareStatement(
            "update tlogin set password=? where username=?" );
```

```
            ps.setString(1, password);
            ps.setString(2, myname);

            if( !(ps.executeUpdate() > 0) ) {
                String error = "ejbStore: LoginBean: " + myname + "
failed";
                log(error);
                throw new NoSuchEntityException(error);
            }
        }
        catch (SQLException ex) {
            log("SQLException: " + ex);
            throw new EJBException(ex);
        }
        finally {
            cleanup(conn,ps);
        }
    }


    public String ejbFindByPrimaryKey(String primaryKey) throws
FinderException {
        log("ejbFindByPrimaryKey: " + primaryKey );

        Connection conn = null;
        PreparedStatement ps = null;

        try {
            conn = getConnection();
            ps = conn.prepareStatement("SELECT password from tlogin
where username = ?");
            ps.setString(1, primaryKey);
            //ps.executeQuery();
            ps.executeQuery();
            ResultSet rs = ps.getResultSet();
            if( rs.next() ) {
                password = rs.getString(1);
            }
            else {
                String error = "ejbFindByPrimaryKey: " + primaryKey + "
Not found";
                log(error);
                throw new ObjectNotFoundException(error);
            }
        }
        catch (SQLException ex) {
            log("SqlExcepton: " + ex);
            throw new EJBException(ex);
        }
        finally {
            cleanup(conn, ps);
        }
        log(" ejbfindbyprimarykey: " + primaryKey + " found");
        return primaryKey;
    }
```

```
   private void log(String str){
        if(VERBOSE) System.out.println(str);
    }


    private Connection getConnection() throws SQLException {
        InitialContext initCtx = null;
        try {

     String url = "t3://localhost:7001";

      // Get an InitialContext
      Properties h = new Properties();
      h.put(Context.INITIAL_CONTEXT_FACTORY,
        "weblogic.jndi.WLInitialContextFactory");
      h.put(Context.PROVIDER_URL, url);

            initCtx = new InitialContext(h);
            DataSource ds = (javax.sql.DataSource)
initCtx.lookup("examples-dataSource-demoPool");
            return ds.getConnection();
        }
        catch (NamingException ne) {
          log("Failed to lookup JDBC Datasource. Please double check
that");
          log("the JNDI name defined in the resource-description of the
");
          log("EJB's weblogic-ejb-jar.xml file is the same as the JNDI
name ");
          log("for the Datasource defined in your config.xml.");
          throw new EJBException(ne);
        }
        finally {
        try{
            if(initCtx != null ) initCtx.close();
            }
        catch(NamingException ne)
        {
            log("Error closing context: " + ne);
            throw new EJBException(ne);
        }
      }

    }


    private String myuser() {
        return "PK=" + (String) ctx.getPrimaryKey();
    }

    private void cleanup(Connection conn, PreparedStatement ps){
        try {
            if( ps != null ) ps.close();
        }
        catch (Exception ex) {
            log("Error closing PreparedStatement: " + ex);
            throw new EJBException(ex);
```

```
        }

        try {
            if( conn != null ) conn.close();
        }
        catch (Exception ex) {
            log("Error closing Connection: " + ex);
            throw new EJBException(ex);
        }
    }


    public String getPassword() {
        return password;
    }
}
```

```
package com.ejb.server.login;

import javax.ejb.CreateException;
import javax.ejb.EJBHome;
import javax.ejb.FinderException;
import java.rmi.RemoteException;
import java.util.Collection;


public interface LoginHome extends EJBHome {

    public Login create(String username, String pwd) throws
CreateException, RemoteException;
    public Login findByPrimaryKey(String primaryKey) throws
FinderException, RemoteException;

    //public LoginRemote findUserNameByEmail(String eamilAddr) throws
FinderException, RemoteException;

    //public LoginRemote findPwdByEmail(String emailAddr) throws
FinderException, RemoteException;  //public void}
package com.ejb.server.policy;
import javax.ejb.EJBObject;

public interface AutoPolicy extends EJBObject {
}


/** Code for  policy enterprise beans.
* There are three entity beans: AutoPolicy,
* HealthPolicy, and LifePolicy;
* and three session beans: AutoPolicyController,
* HealthPolicyController, and LifePolicyController.
*/

package com.ejb.server.policy;

import java.io.Serializable;
```

```
import javax.sql.*;
import java.util.*;
import java.sql.*;
import javax.ejb.*;
import javax.naming.*;
import com.web.beans.*;
import com.ejb.util.*;

public class AutoPolicyBean implements EntityBean {

     private EntityContext ctx;
     private String autopolicyno;
     private String autoquoteno;
     private double policyPremium;
     private Connection conn;

/** The life cycle methods of an entity bean (see Figure 8).
* The server initiated a bean instance.
*/

  public void setEntityContext(EntityContext ctx) throws
javax.ejb.EJBException, java.rmi.RemoteException {
        this.ctx = ctx;
    }

    public void unsetEntityContext() throws javax.ejb.EJBException,
java.rmi.RemoteException {
        this.ctx = null;
    }


    public String ejbCreate(String apno, String aqno, double premium)
throws CreateException {
        this.autopolicyno = apno;
        this.autoquoteno = aqno;
        this.policyPremium = premium;

        try {
            insertRow();
        } catch (Exception ex) {
            throw new EJBException("ejbCreate: " + ex.getMessage());
        }

        return apno;
      }

    public void ejbLoad() {
        try {
            selectRow();
         } catch (Exception ex) {
              throw new EJBException("ejbLoad: " + ex.getMessage());
         }
    }

    public void ejbRemove() {
       try {
            deleteRow(autopolicyno);
```

```
        } catch (Exception ex) {
            throw new EJBException("ejbRemove: " +  ex.getMessage());
        }
    }

    public void ejbStore(){
        try {
            updateRow();
        } catch (Exception ex) {
            throw new EJBException("ejbStore: " +  ex.getMessage());
        }
    }

    public String ejbFindByPrimaryKey(String apno){
        System.out.println("calling FindbyPrimaryKey: " + apno);
        try {
            autopolicyno = apno;
            selectRow();
        }
        catch (Exception ex) {
            throw new EJBException("ejb find by primary key: " + apno);
        }
        return autopolicyno;

    }

    public void ejbActivate() throws javax.ejb.EJBException,
java.rmi.RemoteException {
        System.out.println("Activating instance ... ");
    }

    public void ejbPassivate() throws javax.ejb.EJBException,
java.rmi.RemoteException {
        System.out.println("Passivating Instance ... ");
    }

    public void ejbPostCreate(String apno, String aqno, double premium)
        throws EJBException, CreateException {
        System.out.println("Post Creating the instance ... ");
    }



                    /* Database Connection */

/** There is an AutoPolicy table in the database server corresponding
* to the AutopolicyBean on the middleware, and an interface for Auto
* quote. The users' data of an  appropriate  policy returned by the
* server is stored into the database table at the same time.
*/


private void makeConnection() {
        System.out.println(" Before make connection in autopolicybean
...");
        try {
            InitialContext ic = new InitialContext();
```

```
            DataSource ds = (DataSource)
ic.lookup(StaticNames.THE_DATABASE);
            conn =  ds.getConnection();
        } catch (Exception ex) {
             throw new EJBException("Unable to connect to database. " +
                 ex.getMessage());
        }
        System.out.println("End make connection in autopolicybean ");
    }

    private void releaseConnection() {
        System.out.println("Releasing the connection ... ");
       try {
            conn.close();
        } catch (SQLException ex) {
             throw new EJBException("releaseConnection: " +
ex.getMessage());
        }
    }

    private void deleteRow(String apno) throws SQLException {
       makeConnection();
        String deleteStatement =
                "delete from tautopolicy where policyno = ? ";
        PreparedStatement prepStmt =
                conn.prepareStatement(deleteStatement);

/** Using the primary key of the AutoPolicy table
*/

        prepStmt.setString(1, autopolicyno);
        prepStmt.executeUpdate();
        prepStmt.close();
        releaseConnection();
    }

/** SQL statement: insertion
*/

    private void insertRow() throws SQLException {
        makeConnection();
        String insertStatement =
            "insert into tautopolicy (policyno, quoteno, premium)" +
                " values ( ? , ? , ?)";
        PreparedStatement prepStmt =
            conn.prepareStatement(insertStatement);

        prepStmt.setString(1, autopolicyno);
        prepStmt.setString(2, autoquoteno);
        prepStmt.setDouble(3, policyPremium);
        prepStmt.executeUpdate();
        prepStmt.close();
        releaseConnection();
    }

    private void selectRow() throws SQLException {
        makeConnection();
```

```
        String selectStatement =
                "select quoteno, premium " +
                "from tautopolicy where policyno = ? ";
        PreparedStatement prepStmt =
                conn.prepareStatement(selectStatement);

        prepStmt.setString(1, autopolicyno);

        ResultSet rs = prepStmt.executeQuery();

        if (rs.next()) {
            autoquoteno = rs.getString(1);
            policyPremium = rs.getDouble(2);

            prepStmt.close();
            releaseConnection();
        }
        else {
            prepStmt.close();
            releaseConnection();
            throw new NoSuchEntityException("Row for id " +
                autopolicyno + " not found in database.");
        }
    }

    private void updateRow() throws SQLException {
        makeConnection();
        String updateStatement =
                "update tautopolicy set quoteno=?, premium=? " +
                "where policyno = ?";
        PreparedStatement prepStmt =
                conn.prepareStatement(updateStatement);

        prepStmt.setString(1, autoquoteno);
        prepStmt.setDouble(2, policyPremium);
        prepStmt.setString(3, autopolicyno);

        int rowCount = prepStmt.executeUpdate();
        prepStmt.close();
        releaseConnection();

        if (rowCount == 0) {
            throw new EJBException("Storing row for policy id " +
autopolicyno + " failed.");
        }
    }

    /* End of Database Connection*/
}

/** The AutoPolicyController is a session bean. It is used to connected
* the client and the AutoPolicy entity bean within the application
* server (see Figure 9).
*/

/** Like an entity bean, a session bean consists of a remote interface,
* a home interface, and a bean class. But there is no details database
```

```
* connection in a session bean.
*/

package com.ejb.server.policy;

import javax.ejb.*;
import java.rmi.*;
import com.ejb.exception.*;

public interface AutoPolicyController extends EJBObject {
    public void SaveSoldPolicyNumber(String apno, String aqno, double
premium)
                throws RemoteException, InvalidParameterException;
}

package com.ejb.server.policy;

import java.sql.*;
import javax.sql.*;
import java.util.*;
import java.math.*;
import javax.ejb.*;
import javax.naming.*;
import java.rmi.RemoteException;

import com.ejb.util.*;
import com.ejb.server.*;
import com.ejb.server.quote.*;
import com.ejb.exception.*;
import com.web.beans.*;

public class AutoPolicyControllerBean implements SessionBean {

    private AutoPolicyHome autopolicyHome;
    private AutoPolicy  autopolicy;


    public void ejbCreate() throws EJBException, RemoteException {
       try {
            System.out.println("try to Get AutoPolicyHome ... \n");
            autopolicyHome = EJBGetter.getAutoPolicyHome();
            System.out.println("Successfully getting the home interface
.. \n");
        }
        catch (Exception ex) {
            System.out.println("Could not create autoPolicyHome");
            ex.printStackTrace();
            throw new EJBException("ejbCreate: " + ex.getMessage());
        }
    }
   public void SaveSoldPolicyNumber(String apno, String aqno, double
premium) throws RemoteException, InvalidParameterException {
        if( apno == null || apno.indexOf("ap") == -1)
            throw new InvalidParameterException("Policy number is not
correct!");
        if (aqno == null || aqno.indexOf("aq") == -1)
```

```
            throw new InvalidParameterException("The quote number is
not correct!");
        if( premium <=0 )
            throw new InvalidParameterException("The quote premium
should be positive");

        try {
            System.out.println("Trying to create Autopolicy ... ");
            autopolicy = autopolicyHome.create(apno, aqno, premium);
            System.out.println("Added policy and quote number to
database");
        }
        catch (Exception ex) {
            System.out.println("Exception: " + ex.getMessage());
            throw new EJBException("Exception at creating
AutoPolicyBean in Controller");
        }
        finally {
        }
    }


/** The life cycle methods of a session bean
*/

    public AutoPolicyControllerBean() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbRemove(){}
    public void setSessionContext(SessionContext parm1){}

}


package com.ejb.server.policy;

import javax.ejb.*;
import javax.naming.*;
import java.rmi.RemoteException;

public interface AutoPolicyControllerHome extends EJBHome {
    public AutoPolicyController create() throws RemoteException,
CreateException;
}


package com.ejb.server.policy;

import javax.ejb.*;
import java.rmi.*;

public interface AutoPolicyHome extends EJBHome {
    public AutoPolicy create(String apno, String aqno, double premium)
                            throws RemoteException, CreateException;
    public AutoPolicy findByPrimaryKey(String apno)
                        throws RemoteException, FinderException;
}
```

```
package com.ejb.server.policy;
import javax.ejb.EJBObject;

public interface HealthPolicy extends EJBObject {
}



/* Code for quote enterprise beans */

/** There are AutoQuote, HealthQuote,
* and LifeQuote entity beans.
* Also there are AutoQuoteController,
* HealthQuoteController, and LifeQuoteController
* session beans.
*/


/** All the quote EJBs were used for end-users to enter the appropriate
* information to get an online quote, i.e., AutoQuote, HealthQuote, and
* LifeQuote.
*/

package com.ejb.server.quote;


import java.rmi.RemoteException;
import javax.ejb.EJBObject;
import com.web.beans.*;

public interface AutoQuote extends EJBObject {
    public DriverBean getDriver() throws RemoteException;
    public VehicleBean getVehicle() throws RemoteException;
    //public String getDriverLicence()throws RemoteException;
    public double getAutoPremium() throws RemoteException;
}



package com.ejb.server.quote;
import java.io.Serializable;
import javax.sql.*;
import java.util.*;
import java.sql.*;
import javax.ejb.*;
import javax.naming.*;
import com.web.beans.*;
import com.ejb.util.*;

public class AutoQuoteBean implements EntityBean {
    private final static boolean VERBOSE = true;

    private String quoteid;
    private DriverBean driver;
    private VehicleBean vehicle;
    private EntityContext ctx;
```

```
/** The life cycle methods of an entity bean
*/

     public void setEntityContext(EntityContext ctx) throws
javax.ejb.EJBException, java.rmi.RemoteException {
        this.ctx = ctx;
        log("setEntityContext is being called");
}


/** These methods are used by the server to initiate an instance of
* the bean.
*/

public void unsetEntityContext() throws javax.ejb.EJBException,
java.rmi.RemoteException {
        log("unsetEntityContext: " + mycar() );
        this.ctx = null;
    }

    public void ejbActivate() throws javax.ejb.EJBException,
java.rmi.RemoteException {
        log("ejbActivate -- " + mycar() );
    }

     public void ejbPassivate() throws javax.ejb.EJBException,
java.rmi.RemoteException {
        log("ejbPassivate -- " + mycar() );
    }



                    /* Database Connection */

/** There is an AutoQuote table in the database server corresponding
* to the AutoQuoteBean on the middleware. The users' data of an
* appropriate  quote returned by the server is stored into the
* database table at the same time.
*/


/** In the AutoQuote EJB, there are two JavaBeans: DriverBean and
* VehicleBean. These two JavaBeans components were used to set and
* get the specific driver and vehicle information from the Web side.
*/


public String ejbCreate(String qid, DriverBean thedriver, VehicleBean
thevehicle)throws CreateException {

        driver = thedriver;
        vehicle = thevehicle;

        log("AutoQuoteBean.ejbCreate(quoteid " + qid +
                                    "\n firstname: " +
driver.getFirstName() +
```

```
                                        "\n lastname: " +
driver.getLastName());

        Connection con = null;
        PreparedStatement ps = null;
        quoteid = qid;
        try{
        con = getConnection();
        ps = con.prepareStatement(
            "Insert into tAutoQuote (quoteId, firstName, middlename,
lastName, emailAddress, " +
                                        "dobdate, gender, occupation,
licstatus, licdate, " +
                                        "manufacturer, model, madeYear,
mileage, state, totalValue) " +
            "values(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)");


/** setting the attributes of a driverBean
*/

           ps.setString(1,qid);
            ps.setString(2, driver.getFirstName());
            ps.setString(3, driver.getMiddleName());
            ps.setString(4, driver.getLastName());
            ps.setString(5, driver.getEmail());
            ps.setDate(6, driver.getDobDate());

            ps.setInt(7, driver.getGender());
            ps.setString(8, driver.getOccupation());
            ps.setInt(9, driver.getLicstatus()==true?1:0);
            ps.setDate(10, driver.getLicDate());


/** setting the attributes of a vehicleBean
*/


            ps.setString(11, vehicle.getManufacturer());
            ps.setString(12, vehicle.getModel());
            ps.setString(13, vehicle.getMadeyear());
            ps.setLong(14, vehicle.getMileage());
            ps.setString(15, vehicle.getState());
            ps.setDouble(16, vehicle.getTotalvalue());

            System.out.println("OK, after setting values \n");

             if( ps.executeUpdate() != 1){
                String error = "JDBC did not create any row!";
                log(error);
                throw new CreateException(error);
            }
            return quoteid;
        }
        catch(SQLException ex){

        try{
```

```
        ejbFindByPrimaryKey(quoteid);
        }
         catch (ObjectNotFoundException ex2) {
                String error = "SQLException: " + ex2;
                log(error);
                throw new CreateException(error);
            }
           catch (FinderException fe)
            {
                String error = "FInder Exception: " + fe;
                log(error);
                throw new CreateException(error);
            }
            String error = "An user already exist in the database ...";
            log(error);
            throw new DuplicateKeyException(error);
        }

        finally {
            cleanup(con, ps);  }}
        public void ejbPostCreate(String qid, DriverBean driver,
VehicleBean vehicle)throws CreateException {
            log("ejb PostCreate: " + mycar() );
        }
     public void ejbRemove() throws javax.ejb.RemoveException,
javax.ejb.EJBException, java.rmi.RemoteException {
        log("ejbRemove: " + mycar() );

        Connection con = null;
        PreparedStatement ps = null;

        try {
            con = getConnection();
            ps = con.prepareStatement("delete from tAutoQuote where
quoteid=?");
            ps.setString(1, quoteid);

            if(!( ps.executeUpdate()>0) ) {
                String error = "Error: AutoQuoteBean: " + quoteid + "
Not found";
                log(error);
                throw new NoSuchEntityException(error);
            }
        }
        catch (SQLException ex) {
            log("SqlException: " + ex);
            throw new EJBException(ex);
        }
        finally {
            cleanup(con, ps);
        }
    }


    public void ejbLoad() throws javax.ejb.EJBException,
java.rmi.RemoteException {
        log("ejbLoad() -- " + mycar() );
```

```
        Connection con= null;
        PreparedStatement ps = null;
        quoteid = (String) ctx.getPrimaryKey();

        try {
            con = getConnection();
            ps = con.prepareStatement("SELECT firstName, middlename,
lastName, emailAddress, " +
                                      "dobdate, gender, occupation,
licstatus, licdate, " +
                                      "manufacturer, model, madeYear,
mileage, state, totalValue FROM TAUTOQUOTE where quoteid=?");
            ps.setString(1, quoteid);
            ps.executeQuery();
            ResultSet rs = ps.getResultSet();
            if( rs.next() ) {
               driver.setFirstName(rs.getString(1));
               driver.setMiddleName(rs.getString(2));
               driver.setLastName(rs.getString(3));
               driver.setEmail(rs.getString(4));
               driver.setdobDate(rs.getDate(5));
               driver.setGender(rs.getInt(6));
               driver.setOccupation(rs.getString(7));
               driver.setLicstatus(rs.getInt(8)==1?true:false);
               driver.setLicDate(rs.getDate(9));
               vehicle.setManufacturer(rs.getString(10));
               vehicle.setModel(rs.getString(11));
               vehicle.setMadeyear(rs.getString(12));
               vehicle.setMileage(rs.getLong(13));
               vehicle.setState(rs.getString(14));
               vehicle.setTotalValue(rs.getDouble(15));

            }
            else {
                String error = "ejbLoad: AutoQuoteBean(" + quoteid + ")
not found";
                log(error);
                throw new NoSuchEntityException(error);
            }
        }
        catch (SQLException sqe) {
            log("SQLException: " + sqe);
            throw new EJBException(sqe);
        }
        finally {
            cleanup(con, ps);
        }
    }

     public void ejbStore() throws javax.ejb.EJBException,
java.rmi.RemoteException {
        log("ejbStore: " + mycar() );

        Connection con = null;
        PreparedStatement ps = null;

        try {
```

```
            con = getConnection();

            ps = con.prepareStatement("update tAutoQuote set " +
                                      "firstname=?, middlename=?, lastname=?,
" +
                                      "emailaddress=?, dobdate=?, gender=?, "
+
                                      "occupation=?, licstatus=?, licdate=?,
" +
                                      "manufacturer=?, model=?, madeyear=?, "
+
                                      "mileage=?, state=?, totalvalue=? where
quoteid=?");

            ps.setString(16, quoteid);
            ps.setString(1, driver.getFirstName());
            ps.setString(2, driver.getMiddleName());
            ps.setString(3, driver.getLastName());
            ps.setString(4, driver.getEmail());
            ps.setDate(5, driver.getDobDate());
            ps.setInt(6, driver.getGender());
            ps.setString(7, driver.getOccupation());
            ps.setInt(8, driver.getLicstatus()==true?1:0);
            ps.setDate(9, driver.getLicDate());
            ps.setString(10, vehicle.getManufacturer());
            ps.setString(11, vehicle.getModel());
            ps.setString(12, vehicle.getMadeyear());
            ps.setLong(13, vehicle.getMileage());
            ps.setString(14, vehicle.getState());
            ps.setDouble(15, vehicle.getTotalvalue());

            if( !(ps.executeUpdate() > 0) ) {
                String error = "ejbStore: AutoQuoteBean: " + quoteid +
" failed";
                log(error);
                throw new NoSuchEntityException(error);
            }
        }
        catch (SQLException ex) {
            log("SQLException: " + ex);
            throw new EJBException(ex);
        }
        finally {
            cleanup(con,ps);
        }
    }


    public String ejbFindByPrimaryKey(String primaryKey) throws
FinderException {
        log("ejbFindByPrimaryKey: *** " + primaryKey );

        Connection con = null;
        PreparedStatement ps = null;

        try {
```

```
            con = getConnection();
            ps = con.prepareStatement("SELECT firstName, middlename,
lastName, emailAddress, " +
                                      "dobdate, gender, occupation,
licstatus, licdate, " +
                                      "manufacturer, model, madeYear,
mileage, state, totalValue FROM TAUTOQUOTE where quoteid=?");
            ps.setString(1, primaryKey);
            //ps.executeQuery();
            ps.executeQuery();
            ResultSet rs = ps.getResultSet();
            if( rs.next() ) {
               driver.setFirstName(rs.getString(1));
               driver.setMiddleName(rs.getString(2));
               driver.setLastName(rs.getString(3));
               driver.setEmail(rs.getString(4));
               driver.setdobDate(rs.getDate(5));
               driver.setGender(rs.getInt(6));
               driver.setOccupation(rs.getString(7));
               driver.setLicstatus(rs.getInt(8)==1?true:false);
               driver.setLicDate(rs.getDate(9));
               vehicle.setManufacturer(rs.getString(10));
               vehicle.setModel(rs.getString(11));
               vehicle.setMadeyear(rs.getString(12));
               vehicle.setMileage(rs.getLong(13));
               vehicle.setState(rs.getString(14));
               vehicle.setTotalValue(rs.getDouble(15));
            }
            else {
                String error = "ejbFindByPrimaryKey: " + primaryKey + "
Not found";
                log(error);
                throw new ObjectNotFoundException(error);
            }
        }
        catch (SQLException ex) {
            log("SqlExcepton: " + ex);
            throw new EJBException(ex);
        }
        finally {
            cleanup(con, ps);
        }
        log(" ejbfindbyprimarykey: " + primaryKey + " found");
        return primaryKey;
    }


      private Connection getConnection() throws SQLException {
        try {
            InitialContext ic = new InitialContext();
            DataSource ds = (javax.sql.DataSource) ic.lookup("examples-
dataSource-demoPool");
            return ds.getConnection();
        }
        catch (NamingException ne) {
          log("Failed to lookup JDBC Datasource. Please double check
that");
```

```
            log("the JNDI name defined in the resource-description of the
");
            log("EJB's weblogic-ejb-jar.xml file is the same as the JNDI
name ");
            log("for the Datasource defined in your config.xml.");
            throw new EJBException(ne);
          }
          finally {
              System.out.println("Go through the AutoQuoteBean's
getCOnnection\n");
        }
    }
    private String mycar() {
        return "PK=" + (String)ctx.getPrimaryKey();
    }

    private void log (String s){
     if(VERBOSE) System.out.println(s);
    }

    public DriverBean getDriver() {
        return driver;
    }

    public VehicleBean getVehicle() {
        return vehicle;
    }


/** This method was used to compute the auto premium based on the
* information users entered.
*/


public double getAutoPremium() {
        int yeardiff =
DateHelper.getDifferenceOfYear(driver.getDobDate());
        if ( yeardiff < 0 )
            yeardiff = 0; // in case, mistaken input
        double actualvalue = 0;
        double totalvalue = vehicle.getTotalvalue();

        switch(yeardiff) {
            case 0:
                actualvalue = totalvalue;
                break;
            case 1:
            case 2:
                actualvalue = totalvalue * 0.9;
                break;
            case 3:
            case 4:
                actualvalue = totalvalue * 0.7;
                break;
            case 5:
```

```
            case 6:
            case 7:
            case 8:
                actualvalue = totalvalue * 0.5;
                break;
            case 9: case 10: case 11: case 12:
            case 13: case 14: case 15:
                actualvalue = totalvalue * 0.3;
                break;
            default:
                actualvalue = totalvalue * 0.15;
                break;
        }
        return actualvalue * 0.05;
    }

    private void cleanup(Connection conn, PreparedStatement ps){
        try {
            if( ps != null ) ps.close();
        }
        catch (Exception ex) {
            log("Error closing PreparedStatement: " + ex);
            throw new EJBException(ex);
        }

        try {
            if( conn != null ) conn.close();
        }
        catch (Exception ex) {
            log("Error closing Connection: " + ex);
            throw new EJBException(ex);
        }
    }
}
```

```
package com.ejb.server.quote;

import javax.ejb.*;
import java.util.*;
import java.rmi.*;
import com.web.beans.*;

import com.ejb.exception.*;

/** The AutoQuoteController is a session bean.
*/

/**
* The class acts as a client agent on the server side, which links
* the client and a entity bean, and makes all the workload done on
* the server side, thus
* the three-tiered architecture becomes more efficient
*/
```

```
public interface AutoQuoteController extends EJBObject {
    public String CreateAutoQuote( DriverBean driver, VehicleBean
vehicle)
    throws RemoteException, InvalidParameterException ;


     public void testing(String val) throws RemoteException,
InvalidParameterException ;
     public double getAutoPremium() throws RemoteException;

}

package com.ejb.server.quote;

import java.sql.*;
import javax.sql.*;
import java.util.*;
import java.math.*;
import javax.ejb.*;
import javax.naming.*;
import java.rmi.RemoteException;

import com.ejb.util.*;
import com.ejb.server.*;
import com.ejb.server.quote.*;
import com.ejb.exception.*;
import com.web.beans.*;

public class AutoQuoteControllerBean implements SessionBean {

    private String autoQuoteNo;
    private AutoQuoteHome autoquoteHome;
    private AutoQuote autoquote;
    private Connection conn;

/** The life cycle methods of a session bean
*/

    public String CreateAutoQuote( DriverBean driver, VehicleBean
vehicle )
    throws RemoteException, InvalidParameterException {
        if( driver.getFirstName() == null || driver.getLastName() ==
null )
            throw new InvalidParameterException("null first name or
last name");
        if( driver.getEmail() == null )
            throw new InvalidParameterException("null email address" );
        if(  vehicle.getManufacturer()== null )
            throw new InvalidParameterException("null brand for the
quote");
        if( vehicle.getModel() == null )
            throw new InvalidParameterException("null model for the
quote");
        if( vehicle.getMadeyear() == null )
            throw new InvalidParameterException("null year of the auto
for the quote");
        if( vehicle.getTotalvalue() == 0 )
```

```
            throw new InvalidParameterException("no value for the auto
in the quote");

        StringBuffer bufNo = new StringBuffer("aq");
        bufNo.append(DateHelper.getLast2DigitOfYear());
        bufNo.append(DBHelper.getInstance().getNextQuoteNo());
        autoQuoteNo = bufNo.toString();

        System.out.println("Getting autoQuotNumber: " + autoQuoteNo);

        try {
            makeConnection();

            if( autoquoteHome == null )
            {
                System.out.println("AutoQuoteHome is null \n");
                return "";
            }

        autoquote = autoquoteHome.create(autoQuoteNo, driver, vehicle);
        System.out.println("created the remote autoquote, added data to
database\n");

       //insert a Xref table ???
        releaseConnection();
        }
        catch (Exception ex) {
            releaseConnection();
            throw new EJBException("Createquoteaccount: " +
ex.getMessage());
        }

        return autoQuoteNo;
    }


    public void testing(String val) {
        System.out.println("testing controller remote " + val);
    }

    public void ejbCreate() {
        try {
            System.out.println("Get AutoQuoteHome ... \n");
            autoquoteHome = EJBGetter.getAutoQuoteHome();
            System.out.println("Successfully getting the home interface
.. \n");
        }
        catch (Exception ex) {
            System.out.println("Could not create autoquoteHome");
            ex.printStackTrace();
            throw new EJBException("ejbCreate: " + ex.getMessage());
        }
    }


    public AutoQuoteControllerBean() {}
    public void ejbActivate() {}
```

```
    public void ejbPassivate() {}
    public void ejbRemove() {}
    public void setSessionContext(SessionContext parm1){}
    public double getAutoPremium(){
        try{
           return autoquote.getAutoPremium();
        }
        catch(Exception e) {
            System.out.println("return 0, cuz Exception at getting
premium: " + e.getMessage());
            return 0;
        }
    }

    /* Database routines */


    private void makeConnection() {
        System.out.println("AutoQuoteControllerBean Connection");

/*
       InitialContext initCtx = null;
        try {
            initCtx = new InitialContext();
            DataSource ds = (javax.sql.DataSource)
                        initCtx.lookup("java:comp/env/jdbc/demoPool");
            return ds.getConnection();
*/

        try {
     String url = "t3://localhost:7001";

      // Get an InitialContext
      Properties h = new Properties();
      h.put(Context.INITIAL_CONTEXT_FACTORY,
        "weblogic.jndi.WLInitialContextFactory");
      h.put(Context.PROVIDER_URL, url);

            InitialContext ic = new InitialContext(h);
            DataSource ds = (javax.sql.DataSource) ic.lookup("examples-
dataSource-demoPool");
            conn = ds.getConnection();
        }
        catch (Exception ex) {
            System.out.println("Unable to connect to database: " +
ex.getMessage());
            throw new EJBException(ex.getMessage());
        }
    }//end of makeconnection


  private void releaseConnection() {
        System.out.println("AutoquoteCOntrolllerBean connection
release");
        try {

            if( conn !=null && !conn.isClosed() )
```

```
                conn.close();
            }
            catch (Exception ex) {
                System.out.println("Exception: " + ex.getMessage());
                throw new EJBException(ex.getMessage());
            }
        }// end of releaseconnection

       /* End of database connection */


package com.ejb.server.quote;

import javax.ejb.*;
import javax.naming.*;
import java.rmi.RemoteException;

public interface AutoQuoteControllerHome extends EJBHome {
    public AutoQuoteController create() throws RemoteException,
CreateException;
}



package com.ejb.server.quote;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;
import javax.ejb.FinderException;
import java.util.Collection;
import com.web.beans.*;

public interface AutoQuoteHome extends EJBHome {
    public AutoQuote create(String id, DriverBean driver, VehicleBean
vehicle) throws CreateException, RemoteException;
    public AutoQuote findByPrimaryKey (String primaryKey) throws
FinderException, RemoteException;
    }


/** Code for utility files. There are DataHelper.java,
* DBHelper.java, EjbGetter.java, PlanDetails.java,
* and StaticNames.java. These utility classes are used to
* help to get date, create quote number,policy number, claim number and
* computer the monthly premium.
*/



package com.ejb.util;

import java.util.*;
import java.text.*;
```

```
/** The DateHelper class was used to get the date during the online
* quote process and return the right format.
*/


public final class DateHelper {

    public static String getLast2DigitOfYear(){
        Date date = new Date();
        Calendar cal = new GregorianCalendar();
        cal.setTime(date);
        int year = cal.get(Calendar.YEAR);
        String syear = Integer.toString(year);
        return syear.substring(2,4);
    }


    public static int getDifferenceOfYear(java.sql.Date when){
        Date now = new Date();
        Calendar cal = new GregorianCalendar();
        cal.setTime(now);
        int nowyear = cal.get(Calendar.YEAR);
        cal.setTime(when);
        int whenyear = cal.get(Calendar.YEAR);

        return nowyear - whenyear;
    }


   public static final java.sql.Date getDate(String syear, String
smonth, String sday) {

        // returns a Date with the specified time elements,
        // with the hour and minutes both set to 0 (midnight)

        int year=0, month=0, day=0;
        try{
        year = Integer.parseInt(syear);
        month = Integer.parseInt(smonth);
        day = Integer.parseInt(sday);
        }
        catch(NumberFormatException e) {}
        Calendar cal = new GregorianCalendar(year,
intToCalendarMonth(month), day);
        return new java.sql.Date(cal.getTime().getTime());

    } // getDate



    private static int intToCalendarMonth(int month) {

        if (month == 1)
           return Calendar.JANUARY;
        else if (month == 2)
           return Calendar.FEBRUARY;
```

```
        else if (month == 3)
           return Calendar.MARCH;
        else if (month == 4)
           return Calendar.APRIL;
        else if (month == 5)
           return Calendar.MAY;
        else if (month == 6)
           return Calendar.JUNE;
        else if (month == 7)
           return Calendar.JULY;
        else if (month == 8)
           return Calendar.AUGUST;
        else if (month == 9)
           return Calendar.SEPTEMBER;
        else if (month == 10)
           return Calendar.OCTOBER;
        else if (month == 11)
           return Calendar.NOVEMBER;
        else if (month == 12)
           return Calendar.DECEMBER;
        else
           return Calendar.JANUARY;

    } // intToCalendarMonth



    public static void main(String args[]) {
        String y2 = getLast2DigitOfYear();
        System.out.println(y2);
        Date date = getDate("ac01a", "12", "23");
        System.out.println(date);
    }
}



package com.ejb.util;

import COM.cloudscape.database.*;
import COM.cloudscape.core.*;
import java.sql.*;

/**
 * The DBHelper was used to return the quoteNo, policyNo, claimNo.
 */

public class DBHelper {
    private static final String dbUrl =

"jdbc:cloudscape:C:/bea/wlserver6.1/samples/eval/cloudscape/data/demo";
    private static final String dbDriver =
"COM.cloudscape.core.JDBCDriver";

    private static long quoteNo;
    private static long policyNo;
    private static long reportNo;
```

```
    private static long compNo;
    static private DBHelper obj = null;

    private DBHelper() {}

    public static DBHelper getInstance(){
        if( obj == null ){
            obj = new DBHelper();
            getInfoFromDB();
        }
        return obj;
    }


 private static void getInfoFromDB(){
        Connection conn=null;
        PreparedStatement stmt=null;
        try{
            Class.forName(dbDriver);
            conn = DriverManager.getConnection(dbUrl);
            //System.out.println(conn);
            String  sqlString = "SELECT * FROM TALLID WHERE
projectid=?";
            stmt = conn.prepareStatement(sqlString);
            stmt.setInt(1, 10);
            stmt.executeQuery();
            ResultSet rs = stmt.getResultSet();

            if(rs.next()) {
                quoteNo = rs.getInt(2);
                policyNo = rs.getInt(3);
                reportNo = rs.getInt(4);
                compNo = rs.getInt(5);
            }
            rs.close();
            conn.close();
        }
        catch(Exception se){
        try{
            if( stmt != null ) stmt.close();
            if( conn != null && !conn.isClosed())
                conn.close();

            }
        catch(SQLException ex){}
        }
    }


    public String getNextQuoteNo() {
        long temp = quoteNo;
        quoteNo++;
        updateInfoDB();
        return standardNo(temp);
    }
```

```
  public String getNextPolicyNo() {
        long temp = policyNo;
        policyNo++;
        updateInfoDB();
        return standardNo(temp);
    }

 public String getNextReportNo() {
        long temp = reportNo;
        reportNo++;
        updateInfoDB();
        return standardNo(temp);
    }


public String getNextCompNo() {
        long temp = compNo;
        compNo++;
        updateInfoDB();
        return standardNo(temp);
    }



 private void updateInfoDB(){
        Connection conn = null;
        PreparedStatement stmt=null;
        try {
            Class.forName(dbDriver);
            conn = DriverManager.getConnection(dbUrl);
            conn.setAutoCommit(true);
            //System.out.println(conn);
            String  sqlString = "UPDATE tallid SET quoteno=?, " +
                                                   "policyno=?, " +
                                                   "reportno=?, " +
                                                   "compno=? " +
                                "WHERE projectid=?";
            stmt = conn.prepareStatement(sqlString);
            stmt.setLong(1, quoteNo);
            stmt.setLong(2, policyNo);
            stmt.setLong(3, reportNo);
            stmt.setLong(4, compNo);
            stmt.setInt(5, 10);
            stmt.executeUpdate();
            //if( stmt.executeUpdate() >= 0)
             //   System.out.println("storing data error with
cloudscape database");
            stmt.close();
            conn.close();
        }
        catch (Exception ex) {
            System.out.println(ex);
        }
        finally {
            try{
                if( stmt != null ) stmt.close();
                if( !conn.isClosed() )
```

```
                    conn.close();
                }
                catch(Exception e) {}
            }
        }


 private static String standardNo(long ldigits){

     String digits = Long.toString(ldigits);
        StringBuffer sbuf = new StringBuffer(""); //12 0s
        int len = digits.length();
        for(int i=0; i<8-len; i++)
            sbuf.append("0");
        sbuf.append(digits);
        return sbuf.toString();
    }

public static void main(String []args){
        DBHelper u = DBHelper.getInstance();
        String x = u.getNextQuoteNo();
        x = u.getNextQuoteNo();
        System.out.println(x);
    }
}



package com.ejb.util;

import com.ejb.server.quote.*;
import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import com.ejb.server.policy.*;
import com.ejb.server.login.*;
import com.ejb.server.claim.*;


/** The EGBGetter class was used to bind the EJB name defined as the
* staticName.
*/
/** AutoQuote bean binding
*/


public final class EJBGetter {


 public static AutoQuoteHome getAutoQuoteHome() throws NamingException
{
    try{

        InitialContext initial = new InitialContext();
        System.out.println("getting home ... Initial: " + initial);

/** lookup utility
* The client side had the same object reference of the server side.*/
```

```
        Object objref = initial.lookup(StaticNames.AUTOQUOTE_EJBHOME);

        return (AutoQuoteHome)
            PortableRemoteObject.narrow(objref, AutoQuoteHome.class);
        }
    catch(Exception e) {
        System.out.println("Error when lookuping autoquoteHom");
        System.out.println(e);
    }
    return null;
    }

/** AutoPolicy bean binding
*/

 public static AutoPolicyHome getAutoPolicyHome() throws
NamingException {
    try{

        InitialContext initial = new InitialContext();
        System.out.println("getting home ... Initial: " + initial);
        Object objref = initial.lookup(StaticNames.AUTOPOLICY_EJBHOME);

        return (AutoPolicyHome)
            PortableRemoteObject.narrow(objref, AutoPolicyHome.class);
        }
    catch(Exception e) {
        System.out.println("Error when lookuping autopolicyHome");
        System.out.println(e);
    }
        return null;
    }


/** The EJB server uses the defined StaticNames to lookup the
* specific EJB name.
*/

/** AutoQuoteController bean binding */


public static AutoQuoteControllerHome getAutoQuoteControllerHome()
throws NamingException {
        System.out.println("Starting getting aqc home\n");
        InitialContext initial = new InitialContext();
        Object objref =
initial.lookup(StaticNames.AUTOQUOTE_CONTROLLER_EJBHOME);

        System.out.println("objref: " + objref.toString());

        if( objref == null )
            return null;
        AutoQuoteControllerHome home = (AutoQuoteControllerHome)
            PortableRemoteObject.narrow(objref,
AutoQuoteControllerHome.class);
       System.out.println("aqc home: " + home);
       return home;}
```

```
/** AutoPolicyController bean binding
*/


    public static AutoPolicyControllerHome
getAutoPolicyControllerHome() throws NamingException {
        System.out.println("Starting getting apc home\n");
        InitialContext icx = new InitialContext();
        Object obj =
icx.lookup(StaticNames.AUTOPOLICY_CONTROLLER_EJBHOME);
        if( obj == null )
            return null;
       AutoPolicyControllerHome home = (AutoPolicyControllerHome)
            PortableRemoteObject.narrow(obj,
AutoPolicyControllerHome.class);
       System.out.println("apc home: " + home);
       return home;
       }



/** This class was used to adjust the monthly premium based on the
* entered information, i.e., tobacco use and alcohol use.
*/


 public static double getAdjustedHealthPremium(HealthProfileBean
profile, int id){
        HealthPlanDetail detail = getHealthPlanDetail(id);
        double base = 1;
        if( !profile.getMartial() )
            base = base * 1.1;
        if( profile.getTobaccouser() )
            base = base * 1.2;
        return detail.premium * base;
    }


    public static double getAdjustedLifePremium(LifeProfileBean
profile, int id) {
        LifePlanDetail detail = getLifePlanDetail(id);
        double base = 1;
        if( profile.getAlcoholuser() )
            base = base * 1.1;
        if( profile.getTobaccouser() )
            base = base * 1.2;
        if( profile.getRec() )
            base = base * 1.1;

        return detail.annualPremium * base;
    }
}



package com.ejb.util;
```

```
import com.ejb.server.quote.*;

/** Define the specific names for remotely lookup and binding
*/


public interface StaticNames {

    public static final String THE_DATABASE = "examples-dataSource-
demoPool";

    public static final String AUTOQUOTE_EJBHOME = "thesis-autoQuote";
    public static final String AUTOQUOTE_CONTROLLER_EJBHOME = "thesis-
autoQuoteController";
    public static final String AUTOPOLICY_EJBHOME = "thesis-
autoPolicy";

    public static final String AUTOPOLICY_CONTROLLER_EJBHOME = "thesis-
autoPolicyController";

    public static final String HEALTHQUOTE_EJBHOME = "thesis-
healthQuote";

    public static final String HEALTHQUOTE_CONTROLLER_EJBHOME =
"thesis-healthQuoteController";

    public static final String LIFEQUOTE_EJBHOME = "thesis-lifeQuote";

    public static final String LIFEQUOTE_CONTROLLER_EJBHOME = "thesis-
lifeQuoteController";

    public static final String HEALTHPOLICY_EJBHOME = "thesis-
healthPolicy";

    public static final String HEALTHPOLICY_CONTROLLER_EJBHOME =
"thesis-healthPolicyController";

    public static final String LIFEPOLICY_EJBHOME = "thesis-
lifePolicy";

    public static final String LIFEPOLICY_CONTROLLER_EJBHOME = "thesis-
lifePolicyController";

    public static final String CLAIMSESSION_EJBHOME = "thesis-
claimsession";
}

/** Javabean files: these javabean classes are used
* in the JSP files to help to create dynamic web contents.
* There are claimBean, driverBean, vehicleBean,
* healthPlanDetailBean, healthProfileBean,
* lifePlanDetailBean, and lifeProfileBean.
*/

package com.web.beans;

import java.io.*;
```

```
/** ClaimBean was used to set and get the attributes of an online
* claim.
*/

public class ClaimBean implements Serializable {

    private String firstname;
    private String middlename;
    private String lastname;
    private String email;
    private String policyno;
    private String acM;
    private String acD;
    private String acY;
    private java.sql.Date acDate;
    private String street;
    private String city;
    private String state;
    private String zip;
    private String desc;

    public void setFirstname(String fname){
        this.firstname = fname;
    }

    public void setLastname(String lname){
        this.lastname = lname;
    }

    public void setMiddlename(String ml){
        this.middlename = ml;
    }

    public void setAcD(String d) {
        this.acD = d;
    }

    public void setAcM(String m) {
        this.acM = m;
    }

    public void setAcY(String y) {
        this.acY = y;
    }
    public void setDesc(String desc) {
        this.desc = desc;
    }
    public void setPolicyno(String pn) {
        this.policyno = pn;
    }

    public void setState(String st) {
        this.state = st;
    }

    public void setZip(String zip) {
        this.zip = zip }
```

```
public void setStreet(String s) {
    this.street = s;
}

public void setCity(String c) {
    this.city = c;
}

public void setAcDate(java.sql.Date date) {
    this.acDate = date;
}

public void setEmail(String mail) {
    this.email = mail;
}

public String getAcD() {
    return acD;
}

public java.sql.Date getAcDate() {
    return acDate;
}

public String getAcM() {
    return acM;
}

public String getCity() {
    return city;
}

public String getDesc() {
    return desc;
}
public String getEmail() {
    return email;
}

public String getFirstname() {
    return firstname;}
public String getLastname() {
    return lastname;
}


public String getPolicyno() {
    return policyno;
}

public String getState() {
    return state;
}

public String getStreet() {
    return street; }
```

```
    public String getZip() {
        return zip;
    }

    public String getAcY() {
        return acY;
    }

    public String getMiddlename() {
        return middlename;
    }

}


/** startup class
*/



package com.web.startup;

import java.io.*;
import java.net.*;
import java.lang.Runtime;
import java.lang.Thread;
import java.util.*;
import weblogic.common.*;
import java.net.*;

/**
 * This startup class displays a message in the server shell informing
 * the user which URL can be used to access the application.
 * On Windows, a browser will be automatically launched to the
 * appropriate URL.
 * In this thesis application, the URL is http:\\localhost:7001
 */


public class StartBrowser implements Runnable,T3StartupDef

{
  private String port;
  private String host;
  private Socket socket;
  private final static boolean debug   = false;
  private static int SLEEPTIME = 500;
  private T3ServicesDef services;



  /**
   * Constructs a StartBrowser with the specified host and port values.
   *
   * @param h                 hostname
   * @param p                 listen port
   */
```

```
  public StartBrowser(String h, String p) {
    host = h;
    port = p;

  }


  /**
   * Default constructor.
   *
   */

   public StartBrowser() { }
public void setServices(T3ServicesDef services) {
    this.services = services;
  }


  /**
   * Loops indefinitely trying to create a socket to host/port
   * waits sleepTime in between each try.
   * On a successful socket create, start browser.
   */

  public void run() {

      boolean loop = true;
    while (loop) {
      try {
        socket = new Socket(host, new Integer(port).intValue());
        socket.close();
        //launch browser
        String[] cmdArray = new String[3];
        cmdArray[0] = "beaexec.exe";
        cmdArray[1] = "-target:browser";
        cmdArray[2] = "-command:\"http://"+host+":"+port+"\"";
        try {
          Process p = Runtime.getRuntime().exec(cmdArray);
          p.getInputStream().close();
          p.getOutputStream().close();
          p.getErrorStream().close();
        }
        catch (IOException ioe) {
        }
        loop = false;
      } catch (Exception e) {
        try {
          Thread.sleep(SLEEPTIME); // try every 500 ms
        } catch (InterruptedException ie) {}
        finally {
          try {
            socket.close();
          } catch (Exception se) {}
        }
      }
    }
```

```
  /**
   * This is the first method being called.
   */

  public String startup(String name, Hashtable args)
       throws Exception
  {
    String p = (String)args.get("port");
    String h = InetAddress.getLocalHost().getHostName();
    if (h == null)
      h="localhost";
    String os = System.getProperty("os.name");
    if (os.indexOf("Windows") != -1) {
        System.out.println("\nAutomatically launch the welcome-file
\n");
      Thread t = new Thread(new StartBrowser(h, p));
      t.start();
    }
    else {
      System.out.println("
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
      System.out.println(" After the server has booted, point your
browser \n" +
                         " to the URL \"http://"+h+":"+p+"\" \n" +
                         " to view the WebLogic Server Tour running on
this \n" +
                         " server.");
      System.out.println("
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    }
    return "";
  }
}
```

APPENDIX B

CODE FOR THE DYNAMIC WEB APPLICAITON

There are a total of 30 JSP files. The total number of lines in these source files is approximately 3,000. These files are used to create dynamic Web contents for this insurance application system. JSP technology in this application uses XML-like tags, JavaScript language, and JavaBean written in Java to encapsulate the logic that generates the content for all of the Web pages.

What follows is the list of all source files in the implementation of Web application.

AutoQuotePremium.jsp
DuplicatedUser.jsp
HealthPremium.jsp
Login.jsp
Relogin.jsp
ErrorPassword.jsp
ClaimAccepted.jsp
LifePlan.jsp
AutoPremiumAccepted.jsp
LifePremiumAccepted.jsp
Get_a_auto_quote.jsp
Product.jsp
Get_a_health_quote.jsp
Get_a_life_quote.jsp
HealthPlan.jsp
About_us.jsp
LifePremium.jsp
HealthPremiumAccepted.jsp
Get_a_life_quote.jsp
GetAutoQuote_driver.jsp
GetAutoQuote_vehicle.jsp
Register.jsp
Auto.jsp
Health.jsp
Life.jsp
Profile.jsp
PwdChange.jsp
SuccessProfile.jsp
Claim.jsp
Banner.jsp

The list of the selected program files in this appendix follows:

About_us.jsp
AutoPremiumAccepted.jsp
Banner.jsp
Claim.jsp
Get_a_auto_quote.jsp
Get_a_health_quote.jsp
GetAutoQuote_driver.jsp

GetAutoQuote_vehicle.jsp
HealthPlan.jsp
HealthPremium.jsp
HealthPremiumAccepted.jsp

```
/**
* @author xinyue zhu
* @date 11/08/2001
* /

/** about_us.jsp
* It is used to rent the home page.
*/

<html><HEAD><TITLE></TITLE>
<meta http-equiv="Content-Type" content="text/html; charset=unicode">
<meta name="description" content="">
<meta name="keywords" content="">
<meta name="author" content="Unregistered user">
<meta name="GENERATOR" content="MSHTML 5.00.2920.0"></HEAD>
<BODY>
<h2 align="center">About Xinyue's Shop<b> 
            </b>  </h2>

<jsp:include page="banner.jsp" flush="true"/>

<P>
<H2>The objective of this thesis is
to study J2EE technologies, and from the perspective of a real world
project to implement a three-tier online insurance
application using these technologies. It mainly focous on the business
transaction process
of an insurance company.</H2>
<H3><I>The outline of this client/server application
is: </I></H3>
<TABLE align="left" border=0 cellPadding=0 cellSpacing=0 width="40%"
style="BACKGROUND-COLOR: lightblue">

  <TR>
    <TD><h3> 1) the first tier ( front tier) is the Graphics User
Interface.<p>
      for client accessing;</h3></TD></TR>
  <TR>
    <TD><h3> 2) the second tier ( middleware) is the application
server for
      all business logic;</h3></TD></TR>
  <TR>
    <TD><h3> 3) the third tier ( back end) is the database
server for
      all needed information.</h3></TD></TR></TABLE>
<P><h3><i>      </i> </h3>
<H3><I>   <IMG
align=right alt="" src="file://D:\AceHTML\pages\images\mwbanner3.gif"
```

```
style="HEIGHT: 151px; WIDTH: 469px"></I></H3>
<H3><I></I> </H3>
<H3><I></I> </H3>
<H3><I></I> </H3>
<H3><I></I> </H3>
<H3><I>This application involves some mainly J2EE
technologies:</I></H3>
<P></P>
<P> 
<TABLE align="left" border=0 cellPadding=0 cellSpacing=0 width="50%"
style="BACKGROUND-COLOR: lightsteelblue">

  <TR>
    <TD><h3>JDBC(Java Database Connectivity)</h3></TD>
    <TD><h3> for database access</h3></TD></TR>
  <TR>
    <TD><h3>JNDI(Java Naming and Directory Interface)</h3></TD>
    <TD><h3> for lookup services</h3></TD></TR>
  <TR>
    <TD><h3>EJB (Enterprise Java Bean)</h3></TD>
    <TD><h3> for business logic presentation</h3></TD></TR>
  <TR>
    <TD><h3>Java Servlets and JSP (Java Server Page)</h3></TD>
    <TD><h3> for web application</h3></TD></TR>
  <TR>
    <TD><h3>Java RMI (Remote Method Invocation)</h3></TD>
    <TD><h3> for remotely object access through
network</h3></TD></TR></TABLE>     </P><!--<ul
width= "45%" style="BACKGROUND-COLOR: blanchedalmond"> <h3> <li> the
first tier(front end) is the Graphics User Interface for client </h3>
     <h3> <li> the second tier(middleware) is the application server
for business logic</h3>
  <H3> <li> the third (back end)is the database server for storing all
  date<br></H3></LI>
</ul><--
><h3><I>          
It provides three kinds of products/services:</I> </h3>
<P></P>
<P>
<TABLE align="center" border=0 cellPadding=0 cellSpacing=0  width="30%"
style="BACKGROUND-COLOR: lightsteelblue">

  <TR>
    <TD>  <h4>a. Auto
Insurance          &n
bsp;  
         <IMG alt=""
src="file://D:\AceHTML\pages\images\sm_Auto.gif"
      style="HEIGHT: 23px; WIDTH: 114px">   </h4></TD></TR>
  <TR>
    <TD > <h4>b. Health
Insurance       
           <IMG
      alt="" id=IMG1
  src="file://D:\AceHTML\pages\images\sm_health.gif"
      style="HEIGHT: 23px; WIDTH: 114px">  </h4></TD></TR>
  <TR>
```

```
    <TD> <h4>c. Life
Insurance          &n
bsp;
           <IMG alt=""
      src="file://D:\AceHTML\pages\images\sm_life.gif"
      style="HEIGHT: 23px; WIDTH: 120px">  </h4></TD></TR>
  <H3></H3></TABLE></P>

<p><!--<ol style="BACKGROUND-COLOR: white"> <h4><li> Auto Insurance
</h4>
     <h4> <li style="BACKGROUND-COLOR: white"> Health Insurance </h4>
     <h4> <li> Life Insurance </h4></LI></ol><-->
<P> </P>
<P> </P>
<OL></OL>
<OL></OL></BODY>
</html>



/** autopremiumacccepted.jsp
* When submiting an auto quote, the system calculates
* the required auto premium, and returns to client side.
* If the client accept the premium, the system returns
* an auto policy. DriverBean.java and vehicleBean.java are
* embedded in this JSP file.
*/

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE>Your Policy Paper</TITLE>
</HEAD>
<BODY>

<jsp:include page="banner.jsp" flush="true"/>
<%@page import="com.web.beans.*,com.ejb.util.*,
com.ejb.server.policy.*"%>
<%
      DriverBean driver = (DriverBean)
session.getAttribute("quotedriver");
      String aqno = (String) session.getAttribute("quoteid");
      VehicleBean vehicle =
(VehicleBean)session.getAttribute("quotevehicle");
      Double d = (Double)session.getAttribute("quotepremium");
      int premium = d.intValue();
      StringBuffer buf = new StringBuffer("ap");
      buf.append(DateHelper.getLast2DigitOfYear());
      buf.append(DBHelper.getInstance().getNextPolicyNo());
      String apno = buf.toString();

      String quoteid = (String)session.getAttribute("quoteid");
      //call policyController to store apno & aqno and premium to
PolicyBean
      AutoPolicyControllerHome home =
EJBGetter.getAutoPolicyControllerHome();
      AutoPolicyController ctrl = home.create();
```

```
      ctrl.SaveSoldPolicyNumber(apno, quoteid, d.doubleValue());%><P>
<TABLE border=0 cellPadding=1 cellSpacing=1 width="75%" background=""
bgColor=white style="WIDTH: 75%">
    <TR>
    <TD bgColor=#ffffff>Your Policy Number</TD>
    <TD><%=apno%></TD></TR>
 <TR>
    <TD>Your Premium</TD>
    <TD><%=premium%></TD>
  <TR>
    <TD>First Name</TD>
    <TD><%=driver.getFirstName()%></TD></TR>
  <TR>
    <TD>Middle Name</TD>
    <TD><%=driver.getMiddleName()%></TD></TR>
  <TR>
    <TD>Last Name</TD>
    <TD><%=driver.getLastName()%></TD></TR>
  <TR>
    <TD>
      <P>Gender</P></TD>
    <TD><%=(char)driver.getGender()%></TD></TR>
  <TR>
    <TD>Date of Birth</TD>
    <TD><%=driver.getDobDate()%></TD></TR>
  <TR>
    <TD bgColor=#ffffff>Email</TD>
    <TD><%=driver.getEmail()%></TD></TR>
  <TR>
    <TD>Driver Occupation</TD>
    <TD><%=driver.getOccupation()%></TD></TR>
  <TR>
    <TD>Licensed Date</TD>
    <TD><%=driver. getLicDate()%></TD></TR>
  <TR>
    <TD>License Status</TD>
    <TD><%=driver.getLicstatus()==true?"Valid":"Invalid"%></TD></TR>
  <TR>
    <TD>Vehicle Manufacturer</TD>
    <TD><%=vehicle.getManufacturer()%></TD></TR>
  <TR>
    <TD>Vehicle model</TD>
    <TD><%=vehicle.getModel ()%></TD></TR>
  <TR>
    <TD>Vehicle Made Year</TD>
    <TD><%=vehicle.getMadeyear()%></TD></TR>
  <TR>
    <TD>Vehicle Registered State</TD>
    <TD><%=vehicle.getState()%></TD></TR>
  <TR>
    <TD>Vehicle Total Value </TD>
    <TD><%=vehicle.getTotalvalue()%></TD></TR>
<TR>
<TD>Current Mileage</TD>
<TD><%=vehicle.getMileage()%></TD></TR>
  <TR>
    <TD>Your Quote ID</TD>
```

```
    <TD><%=aqno%></TD></TR></TABLE></P>
<P> </P>
<P>Thanks for choosing Xinyue's Insurance Shop.</p></BODY></HTML>


/** banner.jsp
* This file is used as a banner for every Web page.
* So each other JSP files all contain this JSP file.
*/


<html>
<!-- Creation date: 12/19/2001 -->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1">
<title></title>
<meta name="description" content="">
<meta name="keywords" content="">
<meta name="author" content="Unregistered user">
<meta name="generator" content="AceHTML 5 Pro">
</head>
<body>
<form>
<center><b> Welocme to Xinyue's Insurance Shop! </b></center><br>
</form>

<center> <table border="0"><tr><td BGcolor="#00FFFF"><Font
color="#000000"><a href="about_us.jsp">Home </a></Font></td>
<td BGcolor="#00FFFF">
<Font color="#000000"> <a href="login.jsp">Login in</a> </font></td>

<td bgcolor="#00FFFF">
<font color="#000000"><a href="search.jsp">Search</a> </font></td>
<td bgcolor="#00FFFF">
<font color="#000000"> <a href="about_us.jsp">About Us </a></font></td>
<td bgcolor="#00FFFF">
<font color="#000000"><a href="product.jsp">Product Information
</a></font></td>
<td bgcolor="#00FFFF">
<font color="#000000"><a href="claim.jsp">Claim</a></font></td>
</tr>
</table>
</center>
</body>
</html>
```

```
/** claim.jsp
* It is used to rent a claim page,
* and submit the claim to the system.
*/


<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE></TITLE>
</HEAD>
<BODY bgColor=lightblue>
<jsp:include page="banner.jsp" flush="true"/>
<jsp:useBean id="claim" class="com.web.beans.ClaimBean" scope="session"
/>

<P><h2> To report a auto claim, please provide the following
information:</h2>
<P></P>
<FORM action="claimaccepted.jsp" id=FORM1 method=post name=claim>
<P> </P>
<P>Your First Name:        <INPUT
id=text1
name=firstname>    </P>
<P>Your Middle Name:    <INPUT id=text2
name=middlename></P>
<P>Your Last Name:       <INPUT id=text3
name=lastname></P>
<P>Your Email Address:   <INPUT id=text4 name=email></P>
<P>Your Policy Number:    <INPUT id=text5
name=policyno></P>
<P>Accident
Date:          
Month /  Date  /  Year</P>
<P<INPUT id=text6 name=acM style="HEIGHT: 22px; WIDTH: 33px"> /
<INPUT
id=text7 name=acD style="HEIGHT: 22px; WIDTH: 30px"> /<INPUT
id=text8
name=acY style="HEIGHT: 22px; WIDTH: 36px"></P>
<P>Accident Location:  </P>
: <INPUT id=text12 name=street></P>
<PCity :   <INPUT id=text13 name=city
style="HEIGHT: 22px; WIDTH: 94px">   State: <INPUT id=text14
name=state style="HEIGHT: 22px; WIDTH: 64px">  Zip code: 
<INPUT
id=text15 name=zip style="HEIGHT: 22px; WIDTH: 91px"></P>
<P>Accident Brief Description:</P>
<P;<TEXTAREA id=TEXTAREA1 name=desc style="HEIGHT: 82px; WIDTH:
335px"></TEXTAREA>   
</P>
<P> </P>
<P><INPUT id=submit1 name=submit1 type=submit value="Report "></P>
<P> </P>
<P> </P></FORM>
</BODY>
</HTML>
```

```
/** get_a_auto_quote.jsp
* An interface for quoting an auto policy,
* including some javaScript files
* to create the quote page.
*/

<html>
<!-- Creation date: 12/20/2001 -->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1">
<title></title>
<meta name="description" content="">
<meta name="keywords" content="">
<meta name="author" content="Unregistered user">
<meta name="generator" content="AceHTML 5 Pro">
<h3 align="left">Fill out a Quote, after submission, you can get a
policy.</h3>
</head>
<body>
<jsp:include page="banner.jsp" flush="true"/>
<form action="autoquote" method="Get" name="aquote">
<p>
Your First Name:<input type="text" size="30" value="firstName"
maxlength="30"><br><br>
Your Last Nmae:<input type="text" size="30" value="lastName"
maxlength="30"><br><br>
Your Email Address:<input type="text" size="30" value="emailAddress"
maxlength="30"><br><br>
</p>
<p>

/** Listing some Auto Manufacurer options
*/

Auto Manufacturer:<select name="manufacturer" value="manufacturer" >
<option selected label="none" value="none">Manufacturer</option>
<optgroup label="Auto Manufacturer">
<option label="Toyota" value="toyota">Toyota</option>
<option label="Honda" value="honda">Honda</option>
<option label="Mazda" value="mazda">Mazda</option>
</optgroup>
</select>
</p><p>

/** Based on the specific auto manufacturer, there are some model
* options.
*/


Model:<select name="model" value="model">
<option selected label="none" value="none">Model</option>
<optgroup label="model">
<option label="corrla" value="corrola">Corrola</option>
<option label="camera" value="camera">Camera</option>
<option label="fourona" value="fourona">Fourona</option>
<option label="accura" value="accura">Accura</option>
```

```
</optgroup>
</select>
</p><p>
Made Year:<select name="Made Year" value="madeYear" >
<option selected label="none" value="none">Year</option>
<optgroup label="Made Year">
<option label="2001" value="2001">2001</option>
<option label="2000" value="2000">2000</option>
<option label="1999" value="1999">1999</option>
</optgroup></select></p>
<p>
Total Value: <select name="totalValue" value="totalValue" >
<option selected label="none" value="none">Value</option>
<optgroup label="Total Value">
<option label="$20,000" value="$20,000">$20,000-$18,000</option>
<option label="$18,000" value="$18,000">$18,000-$16,000</option>
<option label="$16,000" value="$16,000">$16,000-$14,000</option>
</optgroup>
</select></p>
<input type="submit" name="send" value="Submit">
<p>
Return to home page <br><a href="home1.html">home</p>

</form>
</body>
</html>



/** get_a_health_quote.jsp **/

/** It was used to create an interface for quoting a health policy.
*/


<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1">
<title>Fill out this form, after submission, you may get a health
policy</title>
<meta name="description" content="">
<meta name="keywords" content="">
<meta name="author" content="Unregistered user">
<meta name="generator" content="AceHTML 5 Pro">
<h2 align="left">Fill out a quote, after submission, you will get a
policy</h2>
</head>
<body>
<jsp:include page="banner.jsp" flush="true"/>
<%@page import="com.ejb.util.*, com.web.beans.*"%>

/** plan type options
*/

<%
        String type = request.getParameter("planradio");
```

```
        int itype=0;
        try{
             char c = type.charAt(0);
             switch(c)
             {
                    case '0': itype = 0; break;
                    case '1': itype = 1; break;
                    case '2': itype = 2; break;
                    case '3': itype = 3; break;
                    case '4': itype = 4; break;
                    case '5': itype = 5; break;
                    case '6': itype = 6; break;
                    case '7': itype = 7; break;
                    case '8': itype = 8; break;
                    default: break;
             }
        }
        catch(NumberFormatException ne)
        {
             //using the default package 0
        }

        HealthPlanDetail detail =
PlanDetails.getHealthPlanDetail(itype);
        session.setAttribute("healthplandetail", detail);

%>

<FORM action="healthpremium.jsp" id=FORM1 method=post name=FORM1>
<p></p>

<P>Social Security Number: <INPUT id=text1 name="ssn"></P>
<P>Your First Name:        
<INPUT
id=text2 name="firstname"></P>
<P>Your Middle Name:     <INPUT id=text3
name="middlename"></P>
<P>Your Last Name:       <INPUT id=text4
name="lastname"></P>
<P>Your Email Address:  <INPUT id=text5 name="email"></P>
<FONT size=1>MM         
DD        YY</FONT>
<P>Date Of
Birth:          
<INPUT id=text6 name="dobM" style="HEIGHT: 22px; WIDTH: 25px">/ <INPUT
id=text7
name="dobD" style="HEIGHT: 22px; WIDTH: 32px">/<INPUT id=text8
name="dobY"
style="HEIGHT: 22px; WIDTH: 37px">
</P>

<P>Tobacco Use: <INPUT id=radio5 name="tobacco" value="1"
type=radio>Yes <INPUT id=radio6
name="tobacco" value="0" type=radio>No</P>

<P>Maritial Status: <INPUT id=radio1 name="martial" value="0"
type=radio>Single <INPUT id=radio2
```

```
name="martial" value="1" type=radio>Married</P>

<P>Gender:   <INPUT id=radio3 name="gender" value="M"
type=radio>Male <INPUT
id=radio4 name="gender" value="F" type=radio>Female</P>
<P><INPUT id=submit1 name=submit1 type=submit value=Submit></P>

</FORM></body></html>


/** get_autoquote_driver.jsp
*/


<%@ page errorPage="entry.jsp" %>
<jsp:useBean id="driver" class="com.web.beans.DriverBean"
scope="session"/>

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE></TITLE>
</HEAD>
<BODY>
<jsp:include page="banner.jsp" flush="true"/>

<P>
<TABLE border=1 cellPadding=1 cellSpacing=1 width="75%">

  <TR>
    <TD align=middle><A href="getautoquote_driver.jsp">Driver
    Information</A></TD>
    <TD align=middle><A href="getautoquote_vehicle.jsp">Vehicle
Information</A></TD></TR></TABLE></P>

<%
        com.web.beans.DriverBean dr = new com.web.beans.DriverBean();
        String m = request.getParameter("email");
        dr.setEmail(m);
        session.setAttribute("dr", dr);

%>

<FORM action="getautoquote_vehicle.jsp" id=FORM1 method=Post
name=FORM1>
<P> </P>
<P align=center>Please fill out the information for the applicant.</P>
<P>Email Address<INPUT id=mailid name=email></P>

<P>First Name<INPUT id=text1 name=firstname></P>
<P>Middle Name<INPUT id=text2 name=middlename style="HEIGHT: 22px;
WIDTH: 75px"></P>
<P>Last Name
 <INPUT id=text3 name=lastname></P>
<P><FONT
size=
MM/DDYYYY</FONT></P>
```

```
<P>Date of Birth
<INPUT id=text5 name=dobM style="HEIGHT: 22px; WIDTH: 47px"> /
<INPUT id=text9 name=dobD style="HEIGHT: 22px; WIDTH: 46px"> /
<INPUT id=text10 name=dobY style="HEIGHT: 22px; WIDTH: 77px"></P>
<P>Gender:          &
nbsp;      
     <INPUT id=radio1 name=gender value = "M"
style="LEFT: 85px; TOP: 302px"
type=radio> Male   <INPUT id=radio2 name=gender
value="F"
type=radio>Female</P>
<P>Driver's Occupation:     <INPUT id=text4
name=driveroccupation></P>


<P>Current License Status: <INPUT id=radio3 name=licstatus value="1"
type=radio>Valid   <INPUT id=radio4 name=licstatus value="0"
type=radio> Invalid</P>

<P MMYYYY</FONT></P>

<P>Date First Licensed:      <INPUT id=text6
name=licM
style="HEIGHT: 22px; WIDTH: 38px"> / <INPUT id=text7 name=licY
style="HEIGHT: 22px; WIDTH: 81px"></P>

<P <INPUT id=button1 name=driversubmit type=submit
value=Continue></P></FORM>
<P> </P>
</BODY>
</HTML



/** getautoquote_vehicle.jsp
*/

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE></TITLE>
</HEAD>
<BODY>
<jsp:include page="banner.jsp" flush="true"/>

/** JavaScipt files
*/

<SCRIPT language=JavaScript>
function changemodel()
{
        document.FORM1.selmod.length = 0;

        myfac = document.FORM1.manufacturesel;
        myvalue = myfac[myfac.selectedIndex].text;
        //alert(myvalue);
```

```
/** various auto manufacurer models
*/

        switch(myvalue)
        {
             case 'Toyota':
                   //document.FORM1.selmod.options.length=4;
                   var option0 = new Option("Model");
                   var option1 = new Option("Corola");
                   var option2 = new Option("Camry");
                   var option3 = new Option("Lexus");
                   document.FORM1.selmod.options.add(option0, 0);
                   document.FORM1.selmod.options.add(option1, 1);
                   document.FORM1.selmod.options.add(option2, 2);
                   document.FORM1.selmod.options.add(option3, 3);
                   break;
             case 'Honda':
                   //document.FORM1.selmod.options.length=3;
                   var op0 = new Option("Model");
                   var op1 = new Option("Civic");
                   var op2 = new Option("Accord");
                   var op3 = new Option("Acura");
                   document.FORM1.selmod.options.add(op0, 0);
                   document.FORM1.selmod.options.add(op1, 1);
                   document.FORM1.selmod.options.add(op2, 2);
                   document.FORM1.selmod.options.add(op3, 3);
                   break;
             case 'Mazda':
                   var op0 = new Option("Model");
                   var op1 = new Option("323");
                   var op2 = new Option("Protege");
                   var op3 = new Option("626");
                   var op4 = new Option("929");
                   document.FORM1.selmod.options.add(op0, 0);
                   document.FORM1.selmod.options.add(op1, 1);
                   document.FORM1.selmod.options.add(op2, 2);
                   document.FORM1.selmod.options.add(op3, 3);
                   document.FORM1.selmod.options.add(op4, 4);
                   break;
             case 'Manufacturer':
                   var opc = new Option("Model");
                   document.FORM1.selmod.options.add(opc, 0);
                   break;
             default:
             break;
        }
        document.FORM1.selmod.options.selectedIndex = 0;
}

</SCRIPT>

<P>
<TABLE border=1 cellPadding=1 cellSpacing=1 width="75%">

  <TR>
    <TD align=middle><A href="getautoquote_driver.jsp">Dirver
Information</A></TD>
```

```
    <TD align=middle><A href="getautoquote_vehicle.jsp">Vehicle
Information</A> </TD></TR></TABLE></P>

<%
        String email = request.getParameter("email");
        String fname = request.getParameter("firstname");
        String lname = request.getParameter("lastname");
        String middlename = request.getParameter("middlename");
      String dobM = request.getParameter("dobM");
      String dobD = request.getParameter("dobD");
      String dobY = request.getParameter("dobY");
      java.sql.Date dobdate = com.ejb.util.DateHelper.getDate(dobY,
dobM, dobD);
      String str_g = request.getParameter("gender");
      char gender = str_g.charAt(0);
      String occupation = request.getParameter("driveroccupation");
      String str_lic = request.getParameter("licstatus");
      boolean licstatus = str_lic.charAt(0)=='1'?true:false;
      String licM = request.getParameter("licM");
      String licY = request.getParameter("licY");
      java.sql.Date licdate = com.ejb.util.DateHelper.getDate(licY,
licM, "0");

        com.web.beans.DriverBean dr = new com.web.beans.DriverBean();

        dr.setEmail(email); dr.setFirstName(fname);
dr.setLastName(lname);
        dr.setMiddleName(middlename); dr.setdobDate(dobdate);
dr.setGender(gender);
        dr.setOccupation(occupation); dr.setLicstatus(licstatus);
dr.setLicDate(licdate);
        session.setAttribute("driverInfo", dr);
%>

<FORM action="autoquote" id=FORM1 method=get name=FORM1>
<P>Continue to fill out this form, after submission, you will get a
right quote policy. </P>
<P>Auto
Manufactuer         

<SELECT id=selman name=manufacturesel onchange=changemodel()
style="HEIGHT: 22px; WIDTH: 136px">
<OPTION selected value="none"> Manufacturer</OPTION>
<option value="toyota">Toyota</option>
<option value="honda">Honda</option>
<option value="mazda">Mazda</option>
</SELECT> </P>

<P>Model<SELECT id=selmod name=modelsel style="HEIGHT: 22px; WIDTH:
141px">
<OPTION selected value="none"> Model </OPTION></SELECT></P>

<P>MadeYear
<INPUT id=text1 name=madeyear style="HEIGHT: 22px; WIDTH: 138px"></P>
<P>State Registered in:       
<SELECT id=select3 name=statesel style="HEIGHT: 22px; WIDTH: 138px">
<OPTION selected value="none"> State </OPTION>
```

```
<OPTION value="alabama"> Alabama </OPTION>
<OPTION value="california"> California </OPTION>
<OPTION value="oklahoma"> Oklahoma </OPTION>
<OPTION value="texas"> Texas </OPTION>
</SELECT></P>
<P>Current Mileage
<P>Total Value
<INPUT id=text3 name=totalvalue style="HEIGHT: 22px; WIDTH: 140px"></P>
<P><INPUT id=button1 name=vehiclesubmit type=submit value=Submit></P>

</FORM>

</BODY>
</HTML>



/** healthplan.jsp**/

/** It depicted the details of health plan types.
*/

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE></TITLE>
</HEAD>
<BODY bgColor=silver>
<jsp:include page="banner.jsp" flush="true"/>

<P>Compare the following plan, then make your choice.</P>
<P style="BACKGROUND-COLOR: silver">  After pressing
the "Apply" button, you can get a health insurance quote.
</P>
<FORM action="get_a_health_quote.jsp" id=FORM1 method=post name=FORM1>
<TABLE border=1 cellPadding=1 cellSpacing=1 style="HEIGHT: 243px;
WIDTH: 659px"
width="75%">

  <TR>
    <TD>Plan Name</TD>
    <TD>Premium (monthly)</TD>
    <TD>Plan Type</TD>
    <TD>Deductable</TD>
    <TD>Coinsurance</TD>
    <TD>Copay</TD>
    <TD>Your Choice  </TD></TR>
  <TR>
    <TD>BasicPPO</TD>
    <TD>$28.00</TD>
    <TD>PPO</TD>
    <TD>$1000</TD>
    <TD>20%</TD>
    <TD>N/A</TD>
    <TD><INPUT id=radio1 name=planradio value="0" type=radio
     ></TD></TR>
  <TR>
```

```
  <TD>ValuePPO</TD>
  <TD>$31.00</TD>
  <TD>PPO</TD>
  <TD>$1000</TD>
  <TD>25%</TD>
  <TD>N/A</TD>
  <TD><INPUT id=radio3 name=planradio value="1" type=radio
   ></TD></TR>
<TR>
  <TD>PPOSaver</TD>
  <TD>$44.00</TD>
  <TD>PPO</TD>
  <TD>$5000</TD>
  <TD>20%</TD>
  <TD>N/A</TD>
  <TD><INPUT id=radio2 name=planradio value="2" type=radio
   ></TD></TR>
<TR>
  <TD>EPO</TD>
  <TD>$53.00</TD>
  <TD>MSA</TD>
  <TD>$2400</TD>
  <TD>50%</TD>
  <TD>N/A</TD>
  <TD><INPUT id=radio4 name=planradio value="3" type=radio
   ></TD></TR>
<TR>
  <TD>PreferedSavingPlan</TD>
  <TD>$83.00</TD>
  <TD>MSA</TD>
  <TD>$1650</TD>
  <TD>20%</TD>
  <TD>N/A</TD>
  <TD><INPUT id=radio6 name=planradio value="4" type=radio
   ></TD></TR>
<TR>
  <TD>HMO40</TD>
  <TD>$145.00</TD>
  <TD>HMO</TD>
  <TD>$2000</TD>
  <TD>0%</TD>
  <TD>$40</TD>
  <TD><INPUT id=radio5 name=planradio value="5" type=radio
   ></TD></TR>
<TR>
  <TD>HMOSaver</TD>
  <TD>$175.00</TD>
  <TD>HMO</TD>
  <TD>$1500</TD>
  <TD>0%</TD>
  <TD>$10</TD>
  <TD><INPUT id=radio7 name=planradio value="6" type=radio
   ></TD></TR>
<TR><TD>HMOPlan 10</TD>
  <TD>$196.55</TD>
  <TD>HMO</TD>
  <TD>$0</TD>
```

```
    <TD>0%</TD>
    <TD>$20</TD>
    <TD><INPUT id=radio8 name=planradio value="7" type=radio
     ></TD></TR><TR>
    <TD>POSPlan</TD>
    <TD>$246.12</TD>
    <TD>POS</TD>
    <TD>$500</TD><TD>20%</TD>
    <TD>N/A</TD><TD><INPUT id=radio9 name=planradio value="8"
  type=radio></TD></TR></TABLE>
<P><INPUT id=submit1 name=submit1 type=submit value=Apply></P></FORM>
<P>Notices:</P><P>. The monthly premium amounts shown are subject to
change based on your medical histroy, the underwriting practices of the
health plan, the optional benefits you selected, if any, and other
relevant factors.</P><P>. The copayment , deductable, and coinsurance
amounts are your share of the costs for covered benefits. </P></BODY>
</HTML>


/** healthpremium.jsp **/

/** getting the monthly health premium.
*/

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE></TITLE>
</HEAD>
<BODY>
<%@page
import="com.ejb.util.*,com.web.beans.*,com.ejb.server.quote.*"%>
<%
        HealthPlanDetail detail =
(HealthPlanDetail)session.getAttribute("healthplandetail");
        HealthProfileBean profile = new HealthProfileBean();

        profile.setSsn(request.getParameter("ssn"));
        profile.setFirstname(request.getParameter("firstname"));
        profile.setMiddlename(request.getParameter("middlename"));
        profile.setLastname(request.getParameter("lastname"));
        profile.setEmail(request.getParameter("email"));

        String dobM = request.getParameter("dobM");
        String dobD = request.getParameter("dobD");
        String dobY = request.getParameter("dobY");
        java.sql.Date dobdate = com.ejb.util.DateHelper.getDate(dobY,
dobM, dobD);
        String str_g = request.getParameter("gender");
        char gender = str_g.charAt(0);

        profile.setDob(dobdate);
        profile.setGender(gender);

        String str_t = request.getParameter("tobacco");
        char t = str_t.charAt(0);
        boolean tobacco = t=='1'?true:false;
```

```
        String str_m = request.getParameter("martial");
        char m = str_m.charAt(0);
        boolean martial = m=='1'?true:false;

        profile.setMartial(martial);
        profile.setTobaccouser(tobacco);

        session.setAttribute("healthplanprofile", profile);

        String quoteid="";
        double premium=0;
        try{
            HealthQuoteControllerHome home =
EJBGetter.getHealthQuoteControllerHome();
            HealthQuoteController ctrl = home.create();
            ctrl.saveHealthQuote(profile, detail.id);
            quoteid = ctrl.getQuoteID();
            premium = ctrl.getPremium();
        }
        catch(Exception ex){
            out.println("Exception in connecting with EJBs");
            //can go from to handle exception
        }
        session.setAttribute("hquoteid", quoteid);
        session.setAttribute("hpremium", new Double(premium));
%>


<P>Your health quote:</P>
<FORM action="healthpremiumaccepted.jsp" id=FORM1 method=post
name=FORM1>
<P> </P>
<P>
<TABLE border=1 cellPadding=1 cellSpacing=1 width="75%">

  <TR>
    <TD>Quote ID</TD>
    <TD><%= quoteid%></TD></TR>
  <TR>
    <TD>Monthly Premium</TD>
    <TD><%=premium%></TD></TR></TABLE></P>
<P><INPUT id=submit1 name=submit1 type=submit value=Accept></P></FORM>
<P> </P>
<P>Return to <A href="banner.jsp">home</A>
</P>
</BODY>
</HTML>



/** healthpremiumaccepted**/

/** The monthly premium is accepted by the users.
*/
```

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE></TITLE>
</HEAD>
<BODY>
<jsp:include page="banner.jsp" flush="true"/>
<%@page
import="com.ejb.util.*,com.web.beans.*,com.ejb.server.quote.*,com.ejb.s
erver.policy.*"%>
<%
        HealthPlanDetail detail =
(HealthPlanDetail)session.getAttribute("healthplandetail");
        HealthProfileBean
profile=(HealthProfileBean)session.getAttribute("healthplanprofile");
        String quoteNo = (String)session.getAttribute("hquoteid");
        Double dval = (Double)session.getAttribute("hpremium");
        StringBuffer pno = new StringBuffer("hp");
        pno.append(DateHelper.getLast2DigitOfYear());
        pno.append(DBHelper.getInstance().getNextPolicyNo());

        int percent = (int)detail.copercent *100;
        String strPer = Integer.toString(percent);
        strPer = strPer + "%";

        String copay =
detail.copay==0?"N/A":Integer.toString(detail.copay);

        //tryc, get healthpolicycontroller to send the data to database
        HealthPolicyControllerHome home =
EJBGetter.getHealthPolicyControllerHome();
        HealthPolicyController ctrl = home.create();
        ctrl.SaveSoldPolicyNumber(pno.toString(), quoteNo,
dval.doubleValue());
%>
<P>
<TABLE border=1 cellPadding=1 cellSpacing=1 width="75%">

  <TR>
    <TD>Social Security Number</TD>
    <TD><%=profile.getSsn()%></TD></TR>
  <TR>
    <TD>First Name</TD>
    <TD><%=profile.getFirstname()%></TD></TR>
  <TR>
    <TD>Middle Name</TD>
    <TD><%=profile.getMiddlename()%></TD></TR>
  <TR>
    <TD>Last Name</TD>
    <TD><%=profile.getLastname()%></TD></TR>
  <TR>
    <TD>Email Address</TD>
    <TD><%=profile.getEmail()%></TD></TR>
  <TR>
    <TD>Date of Birth</TD>
    <TD><%=profile.getDob()%></TD></TR>
  <TR>
```

```
    <TD>Gender</TD>
    <TD><%=profile.getGender()%></TD></TR>
  <TR>
    <TD>Tobacco Use</TD>
    <TD><%=profile.getTobaccouser()?"Tobacco User":"Not a tobacco
user"%></TD></TR>
  <TR>
    <TD>Martial Status</TD>
    <TD><%=profile.getMartial()?"Married":"Single"%></TD></TR>
  <TR>
    <TD>Plan Name</TD>
    <TD><%=detail.Name%></TD></TR>
  <TR>
    <TD>Standard Monthly Premium</TD>
    <TD><%=detail.premium%></TD></TR>
  <TR>
    <TD>Quoted Monthly Premium</TD>
    <TD><%=dval.intValue()%></TD></TR>
  <TR>
    <TD>Plan Type</TD>
    <TD><%=detail.type%></TD></TR>
  <TR>
    <TD>Deductable</TD>
    <TD><%=detail.deductable%></TD></TR>
  <TR>
    <TD>Coinsurance</TD>
    <TD><%=strPer%></TD></TR>
  <TR>
    <TD>Copay</TD>
    <TD><%=copay%></TD></TR>
  <TR>
    <TD>Policy ID</TD>
    <TD><%=pno.toString()%></TD></TR>
  <TR>
    <TD>Quote ID</TD>
    <TD><%=quoteNo%></TD></TR></TABLE></P>
<P> </P>
<P>Thanks for shopping Xinyue's shop.</P>

</BODY>
</HTML>
```

APPENDIX C

CODE FOR THE APPLICATION DEPLOYMENT

There are a total of 5 XML files. These files were used to build and deploy the EJB and Servlets components in the J2EE run time environment. XML tags were used to describe the contents and properties. Rather than focusing on the presentation of content, these tags in XML describe the meaning and hierarchical structure of the data.

```
/**
* author @xinyue zhu
* date 12/25/2001
*/


/** code for build.xml
*  For compiling all java classes and building .jar files.
*/


<project name="thesis_firstbean" default="all" basedir=".">
   <!-- set global properties for this build -->
   <property environment="env"/>
   <property name="source" value="."/>
   <property name="build" value="${source}/build"/>
   <property name="dist" value="${source}/dist"/>
   <property name="WL_HOME" value="C:\bea\wlserver6.1"/>
   <property name="APPLICATIONS"
   value="C:\bea\wlserver6.1\config\examples\applications"/>

  <target name="all" depends="clean, init, compile_ejb, jar_ejb, ejbc,
    copy_servlet, copy_util, copy_xml"/>
   <target name="init">
      <!-- Create the time stamp -->
      <tstamp/>
      <!-- Create the build directory structure used by compile  and
       copy the deployment descriptors into it-->
      <mkdir dir="${build}"/>
      <mkdir dir="${build}/META-INF"/>
      <mkdir dir="${dist}"/>
      <copy todir="${build}/META-INF">
         <fileset dir="${source}">
```

```
<!-- Using a target tree for object execution -- >

<include name="*.xml"/>
            <exclude name="build.xml"/>
         </fileset>
      </copy>
   </target>

< --! Copying the servlet classes -- >

   <target name="copy_servlet">
      <tstamp/>
      <copy
todir="C:\bea\wlserver6.1\config\examples\applications\DefaultWebApp\WB
-INF\classes\com\web">
         <fileset dir="D:\Thesis\TheProject\classes\com\web">
            <include name="*.class"/>
            <exclude name="*Bean.class"/>
         </fileset>
      </copy>
   </target>

<!-- Copying the XML files -- >

   <target name="copy_xml">
      <tstamp/>
      <copy
todir="C:\bea\wlserver6.1\config\examples\applications\DefaultWebApp\WE
B-INF">
         <fileset dir="D:\Thesis\TheProject\src">
            <include name="*.xml"/>
            <exclude name="build.xml, ejb-jar.xml, weblogic-ejb-
jar.xml"/>
         </fileset>
      </copy>
   </target>


<!-- Copying the utility Java files -- >

   <target name="copy_util">
      <tstamp/>
      <copy
todir="C:\bea\wlserver6.1\config\examples\applications\DefaultWebApp\WE
B-INF\classes\com\ejb\util">
         <fileset dir="D:\Thesis\TheProject\classes\com\ejb\util">
            <include name="*.class"/>
            <exclude name=""/>
         </fileset>
      </copy>
   </target>
```

```
   <!-- Compile ejb classes into the build directory (jar preparation)
-->

   <target name="compile_ejb" depends="compile_exception, compile_util,
compile_beans, compile_auto, compile_login, compile_policy,
compile_claim, compile_start"/>


   <!-- Make a standard ejb jar file, including XML deployment
descriptors -->

   <target name="compile_exception">
      <javac srcdir="${source}/com/ejb/exception" destdir="${build}"
includes="*.java"/>
   </target>

   <target name="compile_auto">
      <javac srcdir="${source}/com/ejb/server/quote" destdir="${build}"
includes="*.java"/>
   </target>

   <target name="compile_login">
      <javac srcdir="${source}/com/ejb/server/login" destdir="${build}"
includes="*.java"/>
   </target>



   <target name="compile_util">
      <javac srcdir="${source}/com/ejb/util" destdir="${build}"
includes="*.java"/>
   </target>

   <target name="compile_beans">
      <javac srcdir="${source}/com/web/beans" destdir="${build}"
includes="*.java"/>
   </target>

   <target name="compile_policy">
      <javac srcdir="${source}/com/ejb/server/policy"
destdir="${build}" includes="*.java"/>
   </target>

   <target name="compile_claim">
      <javac srcdir="${source}/com/ejb/server/claim" destdir="${build}"
includes="*.java"/>
   </target>

   <target name="compile_start">
      <javac srcdir="${source}/com/web/startup" destdir="${build}"
includes="*.java"/>
   </target>

   <target name="jar_ejb" depends="compile_ejb">
      <jar jarfile="${dist}/std_thesis_bean.jar" basedir="${build}"/>
   </target>
```

```
   <!-- Run ejbc to create the deployable jar file -->


   <target name="ejbc" depends="jar_ejb">
      <delete file="${APPLICATIONS}/thesis_bean.jar"/>
      <java classname="weblogic.ejbc" fork="yes">
         <sysproperty key="weblogic.home" value="${WL_HOME}"/>
         <arg line="-compiler javac ${dist}/std_thesis_bean.jar
${APPLICATIONS}/thesis_bean.jar"/>
         <classpath>
            <pathelement
path="${WL_HOME}/lib/weblogic_sp.jar;${WL_HOME}/lib/weblogic.jar"/>
         </classpath>
      </java>
   </target>


   <!-- Compile handler class into WEB-INF/classes directory of the
Examples WebApplication.  This ensures that the EJB can be accessed by
JSPs of the Examples Web application. -- >


   <target name="compile_webapp">
      <javac srcdir="${source}" destdir="${EX_WEBAPP_CLASSES}"
includes="Servlet.java, Account.java, AccountHome.java,
        ProcessingErrorException.java"/>
   </target>



   <!-- Compile EJB interfaces & client application into the
clientclasses directory -->

   <target name="compile_client">
      <javac srcdir="${source}" destdir="${CLIENT_CLASSES}"
includes="Account.java, AccountHome.java,
ProcessingErrorException.java,
        Client.java"/>
   </target>

   <target name="clean">
      <delete dir="${build}"/>
      <delete dir="${dist}"/>
   </target>

   <!--COPY ALL THE COM DIRECTORY INTO CLASSES DIR TO LET SERVLET
ACCESS THOSE INTERFACE-->
</project>
```

```
/** code for ejb-jar.xml
* For describing enterprise beans (include entity beans
* and session beans) and deploying these beans.
*/


/** It was used to descibe the property of specific enterprise beans,
* also including the database connection .
*/

<?xml version="1.0"?>
<!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>

< !-- Entity beans -- >

      <entity>
         <ejb-name>firstbean</ejb-name>
         <home>com.ejb.server.login.LoginHome</home>
         <remote>com.ejb.server.login.Login</remote>
         <ejb-class>com.ejb.server.login.LoginBean</ejb-class>
         <persistence-type>Bean</persistence-type>
         <prim-key-class>java.lang.String</prim-key-class>
         <reentrant>False</reentrant>
         <resource-ref>
            <res-ref-name>jdbc/demoPool</res-ref-name>
            <res-type>javax.sql.DataSource</res-type>
            <res-auth>Container</res-auth>
         </resource-ref>
      </entity>


      <entity>
         <ejb-name>autoquotebean</ejb-name>
         <home>com.ejb.server.quote.AutoQuoteHome</home>
         <remote>com.ejb.server.quote.AutoQuote</remote>
         <ejb-class>com.ejb.server.quote.AutoQuoteBean</ejb-class>
         <persistence-type>Bean</persistence-type>
         <prim-key-class>java.lang.String</prim-key-class>
         <reentrant>False</reentrant>
         <resource-ref>
            <res-ref-name>jdbc/demoPool</res-ref-name>
            <res-type>javax.sql.DataSource</res-type>
            <res-auth>Container</res-auth>
         </resource-ref>
      </entity>


      <entity>
         <ejb-name>autopolicybean</ejb-name>
         <home>com.ejb.server.policy.AutoPolicyHome</home>
         <remote>com.ejb.server.policy.AutoPolicy</remote>
         <ejb-class>com.ejb.server.policy.AutoPolicyBean</ejb-class>
         <persistence-type>Bean</persistence-type>
         <prim-key-class>java.lang.String</prim-key-class>
```

```
    <reentrant>False</reentrant>
    <resource-ref>
       <res-ref-name>jdbc/demoPool</res-ref-name>
       <res-type>javax.sql.DataSource</res-type>
       <res-auth>Container</res-auth>
    </resource-ref>
</entity>


<entity>
    <ejb-name>healthpolicybean</ejb-name>
    <home>com.ejb.server.policy.HealthPolicyHome</home>
    <remote>com.ejb.server.policy.HealthPolicy</remote>
    <ejb-class>com.ejb.server.policy.HealthPolicyBean</ejb-class>
    <persistence-type>Bean</persistence-type>
    <prim-key-class>java.lang.String</prim-key-class>
    <reentrant>False</reentrant>
    <resource-ref>
       <res-ref-name>jdbc/demoPool</res-ref-name>
       <res-type>javax.sql.DataSource</res-type>
       <res-auth>Container</res-auth>
    </resource-ref>
</entity>


<entity>
    <ejb-name>lifepolicybean</ejb-name>
    <home>com.ejb.server.policy.LifePolicyHome</home>
    <remote>com.ejb.server.policy.LifePolicy</remote>
    <ejb-class>com.ejb.server.policy.LifePolicyBean</ejb-class>
    <persistence-type>Bean</persistence-type>
    <prim-key-class>java.lang.String</prim-key-class>
    <reentrant>False</reentrant>
    <resource-ref>
       <res-ref-name>jdbc/demoPool</res-ref-name>
       <res-type>javax.sql.DataSource</res-type>
       <res-auth>Container</res-auth>
    </resource-ref>
</entity>


<entity>
    <ejb-name>healthquotebean</ejb-name>
    <home>com.ejb.server.quote.HealthQuoteHome</home>
    <remote>com.ejb.server.quote.HealthQuote</remote>
    <ejb-class>com.ejb.server.quote.HealthQuoteBean</ejb-class>
    <persistence-type>Bean</persistence-type>
    <prim-key-class>java.lang.String</prim-key-class>
    <reentrant>False</reentrant>
    <resource-ref>
       <res-ref-name>jdbc/demoPool</res-ref-name>
       <res-type>javax.sql.DataSource</res-type>
       <res-auth>Container</res-auth>
    </resource-ref>
</entity>

<entity>
```

```
      <ejb-name>lifequotebean</ejb-name>
      <home>com.ejb.server.quote.LifeQuoteHome</home>
      <remote>com.ejb.server.quote.LifeQuote</remote>
      <ejb-class>com.ejb.server.quote.LifeQuoteBean</ejb-class>
      <persistence-type>Bean</persistence-type>
      <prim-key-class>java.lang.String</prim-key-class>
      <reentrant>False</reentrant>
      <resource-ref>
         <res-ref-name>jdbc/demoPool</res-ref-name>
         <res-type>javax.sql.DataSource</res-type>
         <res-auth>Container</res-auth>
      </resource-ref>
   </entity>

<!-- Session beans -- >

   <session>
      <description>no desc</description>
      <display-name>autoquoteController</display-name>
      <ejb-name>autoquoteController</ejb-name>
      <home>com.ejb.server.quote.AutoQuoteControllerHome</home>
      <remote>com.ejb.server.quote.AutoQuoteController</remote>
                  <ejb-
         class>com.ejb.server.quote.AutoQuoteControllerBean</ejb-
         class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
   </session>


   <session>
      <description>no desc</description>
      <display-name>autopolicyController</display-name>
      <ejb-name>autopolicyController</ejb-name>
      <home>com.ejb.server.policy.AutoPolicyControllerHome</home>
      <remote>com.ejb.server.policy.AutoPolicyController</remote>
            <ejb-
   class>com.ejb.server.policy.AutoPolicyControllerBean</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
   </session>


   <session>
      <description>no desc</description>
      <display-name>healthpolicyController</display-name>
      <ejb-name>healthpolicyController</ejb-name>
      <home>com.ejb.server.policy.HealthPolicyControllerHome</home>
      <remote>com.ejb.server.policy.HealthPolicyController</remote>
            <ejb-
   class>com.ejb.server.policy.HealthPolicyControllerBean</ejb-
   class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
   </session>
```

```
      <session>
         <description>life policy</description>
         <display-name>lifepolicyController</display-name>
         <ejb-name>lifepolicyController</ejb-name>
         <home>com.ejb.server.policy.LifePolicyControllerHome</home>
         <remote>com.ejb.server.policy.LifePolicyController</remote>
               <ejb-
      class>com.ejb.server.policy.LifePolicyControllerBean</ejb-class>
         <session-type>Stateful</session-type>
         <transaction-type>Container</transaction-type>
      </session>


      <session>
         <description>no desc</description>
         <display-name>healthquoteController</display-name>
         <ejb-name>healthquoteController</ejb-name>
         <home>com.ejb.server.quote.HealthQuoteControllerHome</home>
         <remote>com.ejb.server.quote.HealthQuoteController</remote>
               <ejb-
      class>com.ejb.server.quote.HealthQuoteControllerBean</ejb-class>
         <session-type>Stateful</session-type>
         <transaction-type>Container</transaction-type>
      </session>


      <session>
         <description>no desc</description>
         <display-name>lifequoteController</display-name>
         <ejb-name>lifequoteController</ejb-name>
         <home>com.ejb.server.quote.LifeQuoteControllerHome</home>
         <remote>com.ejb.server.quote.LifeQuoteController</remote>
               <ejb-
      class>com.ejb.server.quote.LifeQuoteControllerBean</ejb-class>
         <session-type>Stateful</session-type>
         <transaction-type>Container</transaction-type>
      </session>


      <session>
         <ejb-name>claimsession</ejb-name>
         <home>com.ejb.server.claim.ClaimSessionHome</home>
         <remote>com.ejb.server.claim.ClaimSession</remote>
         <ejb-class>com.ejb.server.claim.ClaimSessionBean</ejb-class>
         <session-type>Stateless</session-type>
         <transaction-type>Container</transaction-type>
      </session>
   </enterprise-beans>

<!-- Container-managed transaction -- >

   <assembly-descriptor>
      <container-transaction>
         <method>
            <ejb-name>firstbean</ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>*</method-name>
```

```
    </method>
    <trans-attribute>Required</trans-attribute>
</container-transaction>


<container-transaction>
    <method>
        <ejb-name>autoquotebean</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
</container-transaction>


<container-transaction>
    <method>
        <ejb-name>autopolicybean</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
</container-transaction>


<container-transaction>
    <method>
        <ejb-name>healthpolicybean</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
</container-transaction>


<container-transaction>
    <method>
        <ejb-name>lifepolicybean</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
</container-transaction>


<container-transaction>
    <method>
        <ejb-name>healthquotebean</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
</container-transaction>


<container-transaction>
    <method>
```

```
      <ejb-name>autoquoteController</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>*</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>


<container-transaction>
   <method>
      <ejb-name>autopolicyController</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>*</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>


<container-transaction>
   <method>
      <ejb-name>healthpolicyController</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>*</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>


<container-transaction>
   <method>
      <ejb-name>lifepolicyController</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>*</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>


<container-transaction>
   <method>
      <ejb-name>healthquoteController</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>*</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>


<container-transaction>
   <method>
      <ejb-name>lifequotebean</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>*</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
```

```
      <container-transaction>
         <method>
            <ejb-name>lifequoteController</ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>*</method-name>
         </method>
         <trans-attribute>Required</trans-attribute
 </container-transaction>
      <container-transaction>
         <method>
            <ejb-name>claimsession</ejb-name>
            <method-name>*</method-name>
         </method>
         <trans-attribute>Required</trans-attribute>
      </container-transaction>
   </assembly-descriptor>
</ejb-jar>
```

```
/** code for weblogic-ejb-jar.xml
* For the session beans desciptor.
*/


<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic
6.0.0 EJB//EN' 'http://www.bea.com/servers/wls600/dtd/weblogic-ejb-
jar.dtd'>

<weblogic-ejb-jar>

<!-- There are statefull session bean and stateless session bean -- >

   <weblogic-enterprise-bean>
      <ejb-name>autoquoteController</ejb-name>
      <stateful-session-descriptor>
         <stateful-session-clustering>
            <replication-type>InMemory</replication-type>
         </stateful-session-clustering>
      </stateful-session-descriptor>
      <jndi-name>thesis-autoQuoteController</jndi-name>
   </weblogic-enterprise-bean>


   <weblogic-enterprise-bean>
      <ejb-name>autopolicyController</ejb-name>
      <stateful-session-descriptor>
         <stateful-session-clustering>
            <replication-type>InMemory</replication-type>
         </stateful-session-clustering>
      </stateful-session-descriptor>
      <jndi-name>thesis-autoPolicyController</jndi-name>
   </weblogic-enterprise-bean>


   <weblogic-enterprise-bean>
      <ejb-name>healthpolicyController</ejb-name>
      <stateful-session-descriptor>
         <stateful-session-clustering>
            <replication-type>InMemory</replication-type>
         </stateful-session-clustering>
      </stateful-session-descriptor>
      <jndi-name>thesis-healthPolicyController</jndi-name>
   </weblogic-enterprise-bean>


   <weblogic-enterprise-bean>
      <ejb-name>lifepolicyController</ejb-name>
      <stateful-session-descriptor>
         <stateful-session-clustering>
            <replication-type>InMemory</replication-type>
         </stateful-session-clustering>
      </stateful-session-descriptor>
      <jndi-name>thesis-lifePolicyController</jndi-name>
   </weblogic-enterprise-bean>
```

```
<weblogic-enterprise-bean>
   <ejb-name>healthquoteController</ejb-name>
   <stateful-session-descriptor>
      <stateful-session-clustering>
         <replication-type>InMemory</replication-type>
      </stateful-session-clustering>
   </stateful-session-descriptor>
   <jndi-name>thesis-healthQuoteController</jndi-name>
</weblogic-enterprise-bean>


<weblogic-enterprise-bean>
   <ejb-name>lifequoteController</ejb-name>
   <stateful-session-descriptor>
      <stateful-session-clustering>
         <replication-type>InMemory</replication-type>
      </stateful-session-clustering>
   </stateful-session-descriptor>
   <jndi-name>thesis-lifeQuoteController</jndi-name>
</weblogic-enterprise-bean>


<weblogic-enterprise-bean>
   <ejb-name>firstbean   </ejb-name>
   <entity-descriptor>
      <entity-cache>
         <max-beans-in-cache>100</max-beans-in-cache>
      </entity-cache>
   </entity-descriptor>
   <reference-descriptor>
      <resource-description>
         <res-ref-name>jdbc/demoPool</res-ref-name>
         <jndi-name>examples-dataSource-demoPool</jndi-name>
      </resource-description>
   </reference-descriptor>
   <jndi-name>thesis-firstbeanHome</jndi-name>
</weblogic-enterprise-bean>


<weblogic-enterprise-bean>
   <ejb-name>autoquotebean</ejb-name>
   <entity-descriptor>
      <entity-cache>
         <max-beans-in-cache>100</max-beans-in-cache>
      </entity-cache>
   </entity-descriptor>
   <reference-descriptor>
      <resource-description>
         <res-ref-name>jdbc/demoPool</res-ref-name>
         <jndi-name>examples-dataSource-demoPool</jndi-name>
      </resource-description>
   </reference-descriptor>
   <jndi-name>thesis-autoQuote</jndi-name>
</weblogic-enterprise-bean>
```

```
<weblogic-enterprise-bean>
   <ejb-name>autopolicybean</ejb-name>
   <entity-descriptor>
      <entity-cache>
         <max-beans-in-cache>100</max-beans-in-cache>
      </entity-cache>
   </entity-descriptor>
   <reference-descriptor>
      <resource-description>
         <res-ref-name>jdbc/demoPool</res-ref-name>
         <jndi-name>examples-dataSource-demoPool</jndi-name>
      </resource-description>
   </reference-descriptor>
   <jndi-name>thesis-autoPolicy</jndi-name>
</weblogic-enterprise-bean>


<weblogic-enterprise-bean>
   <ejb-name>healthpolicybean</ejb-name>
   <entity-descriptor>
      <entity-cache>
         <max-beans-in-cache>100</max-beans-in-cache>
      </entity-cache>
   </entity-descriptor>
   <reference-descriptor>
      <resource-description>
         <res-ref-name>jdbc/demoPool</res-ref-name>
         <jndi-name>examples-dataSource-demoPool</jndi-name>
      </resource-description>
   </reference-descriptor>
   <jndi-name>thesis-healthPolicy</jndi-name>
</weblogic-enterprise-bean>


<weblogic-enterprise-bean>
   <ejb-name>lifepolicybean</ejb-name>
   <entity-descriptor>
      <entity-cache>
         <max-beans-in-cache>100</max-beans-in-cache>
      </entity-cache>
   </entity-descriptor>
   <reference-descriptor>
      <resource-description>
         <res-ref-name>jdbc/demoPool</res-ref-name>
         <jndi-name>examples-dataSource-demoPool</jndi-name>
      </resource-description>
   </reference-descriptor>
   <jndi-name>thesis-lifePolicy</jndi-name>
</weblogic-enterprise-bean>


<weblogic-enterprise-bean>
   <ejb-name>healthquotebean</ejb-name>
   <entity-descriptor>
      <entity-cache>
         <max-beans-in-cache>100</max-beans-in-cache>
      </entity-cache>
```

```
      </entity-descriptor>
      <reference-descriptor>
         <resource-description>
            <res-ref-name>jdbc/demoPool</res-ref-name>
            <jndi-name>examples-dataSource-demoPool</jndi-name>
         </resource-description>
      </reference-descriptor>
      <jndi-name>thesis-healthQuote</jndi-name>
   </weblogic-enterprise-bean>


   <weblogic-enterprise-bean>
      <ejb-name>lifequotebean</ejb-name>
      <entity-descriptor>
         <entity-cache>
            <max-beans-in-cache>100</max-beans-in-cache>
         </entity-cache>
      </entity-descriptor>
      <reference-descriptor>
         <resource-description>
            <res-ref-name>jdbc/demoPool</res-ref-name>
            <jndi-name>examples-dataSource-demoPool</jndi-name>
         </resource-description>
      </reference-descriptor>
      <jndi-name>thesis-lifeQuote</jndi-name>
   </weblogic-enterprise-bean>


   <weblogic-enterprise-bean>
      <ejb-name>claimsession</ejb-name>
      <jndi-name>thesis-claimsession</jndi-name>
   </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

```
/** code for weblogic.xml
* For application server connecting
* various enterprise beans.
*/


<!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web
Application 6.0//EN" "http://www.bea.com/servers/wls600/dtd/weblogic-
web-jar.dtd">
<weblogic-web-app>
   <description>no desc</description>
   <jsp-descriptor>
      <jsp-param>
         <param-name>
           pageCheckSeconds
         </param-name>
         <param-value>
           1
         </param-value>
      </jsp-param>
      <jsp-param>
         <param-name>
           verbose
         </param-name>
         <param-value>
           true
         </param-value>
      </jsp-param>
   </jsp-descriptor>
</weblogic-web-app>
```

```
/** code for web.xml
* For servlet classes and JSP files mapping.
*/


<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
   <display-name>Examples Web Application</display-name>
   <servlet>
      <servlet-name>thelogin</servlet-name>
      <servlet-class>com.web.LoginServlet</servlet-class>
   </servlet>
   <servlet>
      <servlet-name>autoquote</servlet-name>
      <servlet-class>com.web.AutoQuoteServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>thelogin</servlet-name>
      <url-pattern>/theloginok</url-pattern>
   </servlet-mapping>
   <servlet-mapping>
      <servlet-name>autoquote</servlet-name>
      <url-pattern>/autoquote</url-pattern>
   </servlet-mapping>
   <welcome-file-list>
      <welcome-file>about_us.jsp</welcome-file>
   </welcome-file-list>
</web-app>
```

VITA

Xinyue Zhu

Candidate of the Degree of

Master of Science

Thesis: AN ONLINE THREE-TIERED INSURANCE APPLICATION SYSTEM: A WEB-BASED ADVANCED APPLICATION FOR THE J2EE PLATFORM

Major Field: Computer Science

Biographical:

Education: Received Bachelor of Science in Financial Accounting from Hunan Financial University, ChangSha, P. R. China in July 1991. Completed the requirements for the Master of Science degree in Computer Science at the Computer Science Department at Oklahoma State University in May 2002.

Experience: Employed as an accountant in People's Insurance Company of China, Hunan Branch from July 1991 to April 1999.