



Application Developers'
Intro School
Specifications Binder
Segment 3

Table of Contents

Activity Instructions	3
Overview	3
Activity Steps.....	3
Key Learning Points	3
Learning Materials	3
Step 1 – Prepare for this Activity.....	5
Step 2 – Prepare the Environment.....	5
Step 3 – Code the Java Objects	8
Step 4 – Execute Scripts, Verify, Debug and Fix.....	8
Step 5 – Save Project Files	9
Step 6 – Reflect on Key Learning Points	9
Technical Specifications.....	10
Purpose.....	10
Project Description	10
Architectural Overview.....	10
Database Tables	11
Database Architecture	11
Calling Sequence	11
Systems Integration.....	12
Call Pattern References.....	13
Pseudo Code.....	13
Unit Testing.....	16

Activity Instructions

Overview

In this segment, you are expected to create Java classes that participate in the JMVC framework. These Java objects use other Java objects, make SQL calls to the database, return information it retrieves to other Java objects, and provides information to be displayed in JSPs.

The JSP and HTML pages that you coded in the previous segments will be used in testing the Java objects that you will create here.

This activity is designed to introduce you to the JMVC framework, review SQL concepts, and apply object-oriented programming concepts.

Activity Steps



To complete the activity you need to follow the steps above. These steps are explained later in greater detail in this section of the binder.

Key Learning Points

At the end of the segment, you need to learn and understand:

- How to develop Java objects
- How your developed object interact with existing components in the system
- How to use SQL to extract information from a database
- How Java and SQL interact to pass information to a Web page
- Unit Testing and Debugging

Learning Materials

The following information may be found within this binder, the Reference Guide or the Performance Support Tool. Because you have used the materials in the previous segments you should be somewhat familiar with what the resources contain and how they will be used. This section describes the purpose and location of each document.

Input

What	Why	Where
Activity Instructions	Guides you through the process of completing the activity	Activity Instructions section in this binder
Technical Specifications	Used as a blueprint for coding java logic	Technical Specifications section in this binder

Reference Material

What	Why	Where
Compiling and Debugging with IntelliJ IDEA	Provides basic background and "how to" compile your project and debug it	IntelliJ Help Step 4 - Execute Test Scripts, Verify, Debug and Fix activity step in this Binder
Java Overview and Java Glossary	Provides information on Java syntax	Reference Guide
Java Packaged Classes	Provides information on packaged classes relevant to the case	c:\appdevschool\references\
MVC Overview	Provides an overview of the model-view-controller design pattern	Reference Guide
SQL Overview and SQL Syntax Reference	Provides information on SQL syntax	Reference Guide
Levels of Testing	Provides information on Unit Testing procedures and documentation	SDLC in the Performance Support Tool
Documentation Standards	Provides information on naming conventions and standards within the company and the community that uses Java	Documentation Standards in the Performance Support Tool
Frequently Asked Questions	Provides answers to questions found throughout the Activity Instructions	Frequently Asked Questions in the Performance Support Tool

Outputs

What	Why	Where
SubscriberManager.java	Connects to the database and returns subscriber information as a String	Server Web Host
SubscriberUsageManager.java	Connects to the database and returns either subscriber usage or subscriber monthly usage information as a String	Server Web Host

Step 1 – Prepare for this Activity

1. Review the information contained in each tab of the Activity Binder.

Briefly review the Activity Overview. Browse the sections of the technical specifications. Get an overview of the information they contain. You will use these sections extensively when you are coding and debugging so it is helpful to gain a general understanding of these materials.

2. Mark your Reference Guide and Performance Support Tool.

Throughout this binder you will encounter questions inside text boxes. These are relevant to the tasks at hand. You will find the answers to these questions from the either the Reference Guide or Performance Support Tool provided to you.

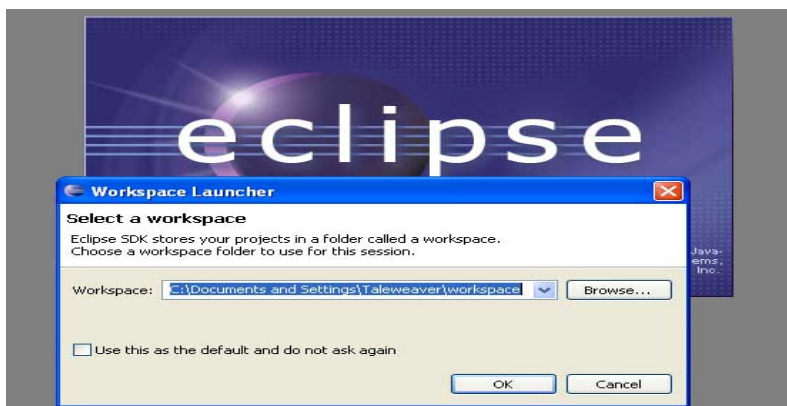
3. Confirm your understanding of the application logic.

To confirm your understanding of the application logic, try to answer the questions below:

- a. What java objects are passed between the SubscriberManager and the SubscriberInfo.jsp?
- b. Which elements in your web project depend on the java objects you will be coding?
- c. With which elements in your web project are your java objects dependent on?

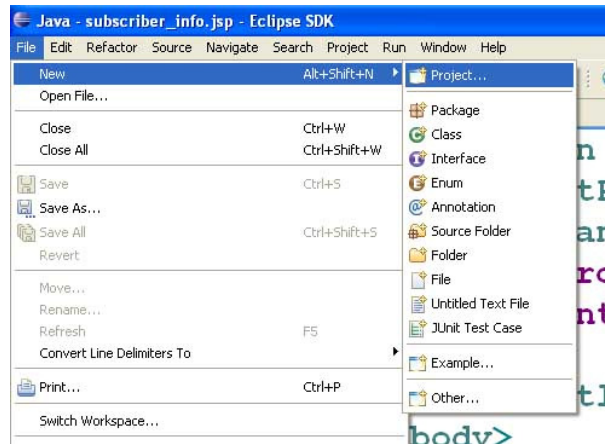
Step 2 – Prepare the Environment

1. Open Eclipse.

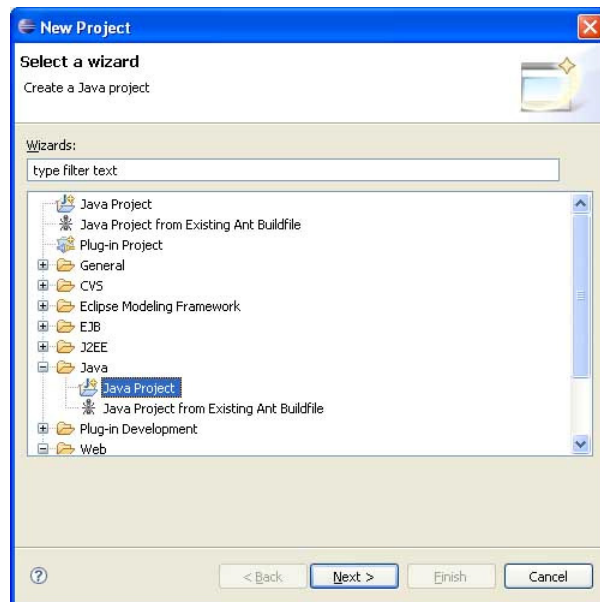


Note: Remember Workspaces? They are physical folders on your PC that hold Eclipse project files. Each PC can have multiple Workspaces. Each new project you create has one subfolder inside the Workspace.

2. Create a new Java project.
 - a. Click on **File – New – Project**.

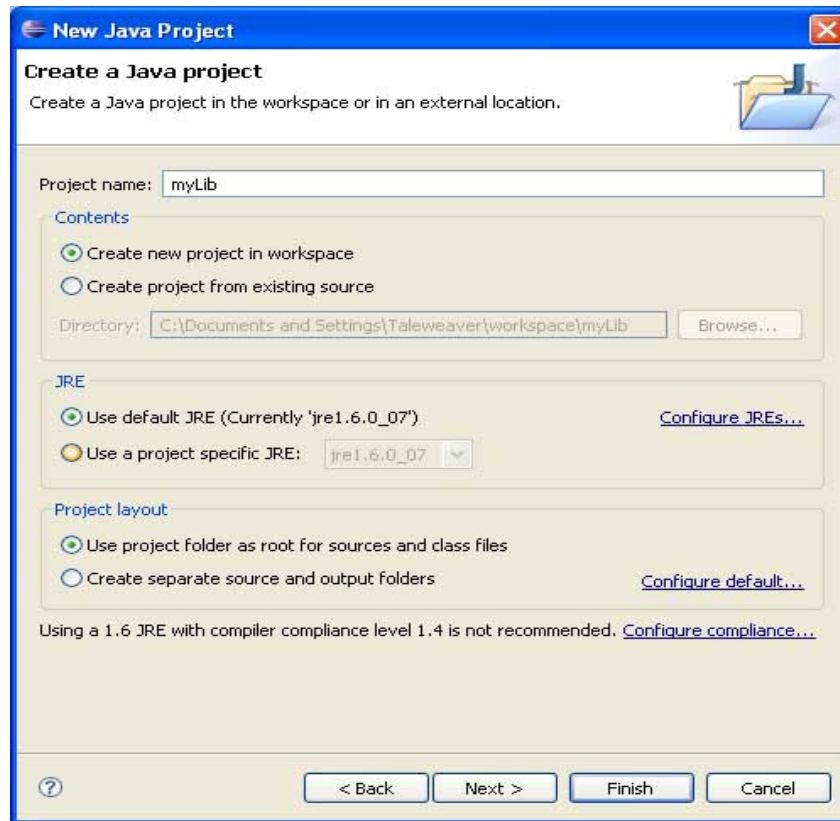


The **New Project Wizard** dialog box appears. Select Java Project.



- b. Click **Next>**.

- c. Type in the Project Name **myLib**. Leave all other options as is. Click **Finish**.



3. Import the previous packages you used in Segment 2 except for **Managers**.
4. Add **Servlet.Jar** to your project's build path under the Project Libraries tab.
5. Create the java files.
 - a. **Right Click** on your project and create a **new package** called **Managers**.
 - b. **Right Click** on the **Managers** package and select **New File – Java**.
 - c. **Type** the name of your new java file including the .java extension.

Step 3 – Code the Java Objects

- What are the SQL classes in Java?
- How can I reference custom Java objects in my code?

You will now code your Java objects. To perform this step, refer to the Systems Integration, Call Pattern References and Pseudo Code sections in the Technical Specifications tab.

1. Using the abovementioned Technical Specification sections, code the `getSubscriber` method in `SubscriberManager.java`.
2. Using the abovementioned Technical Specification sections, code the following methods in the `SubscriberUsageManager.java`:
 - a. `getSubscriberUsage`
 - b. `getSubscriberUsagePerMonth`

Step 4 – Execute Scripts, Verify, Debug and Fix

Refer to the Unit Testing tab of this segment binder for the Unit Test documents you will use. To perform unit testing on the coded java objects you must go through the following steps:

1. Compile the project. Eclipse auto-compiles your project every time you save.
 - a. To compile manually, select **Build Project** from the **Project** menu.
 - b. By default Eclipse auto-compiles your work. You can turn off this feature by going to the **Project** menu and removing the check from **Build Automatically**.
2. Debug the code.
 - a. Go back to your **Segment 2** project.
 - b. Replace the Managers classes in Segment 2 with your output from Segment 3.
 - c. Re-run the Segment 2 test script to see if everything still works.
3. Execute steps for each cycle of your **Unit Test Script** for **Segment 3** and verify your actual results against the expected results.
4. If you find discrepancies, analyze your code to determine the cause of the error. Correct the error and rerun your test scripts to retest your code.
5. Create a Unit Test Report Form using the template provided at the `c:\appdevschool\reports` folder.
6. Save the file as `Segment 3 - Unit Test Report.doc` to the same folder.

Step 5 – Save Project Files

▪ [Saving Files](#)

1. After you have completed the coding and debugging of your JSP files locally, commit the changes you have made by saving both projects.

Step 6 – Reflect on Key Learning Points

Inform your instructor that you have completed the coding and testing activities of this segment. He/she will review your work.

Review the key learning points for this segment. Answer the following questions and discuss them with your co-participants and faculty.

General

1. What did you learn from yesterday's experience that helped you today?
2. What was difficult about today's assignment?
3. What will you do differently if you had to repeat today's activity?
4. What have you learned from the last two days that will help you orient yourself in your next project?

Specific

1. What are the benefits of deploying a Java object using the JMVC framework?
2. What is the value of having a servlet retrieve information from the database instead of calling it directly on your JSP page?
3. What is the purpose of the Statement class?
4. How do you perform error handling in Java?

Technical Specifications

Purpose

This document details the processing logic for the java manager objects. Sufficient information is provided to write the code. This technical specifications document is a hybrid of sections derived from the full-length Functional Specifications and DB Design Documents. In the future (depending on the scope of your project), you may encounter shorter or longer versions of these documents.

▪ [What are functional specifications? Why do we need it?](#)

Project Description

Users of the Smart Online facility for subscribers have been clamoring for an additional feature that would enable them to look up their usage history to track their calls and manage their subscription plan.

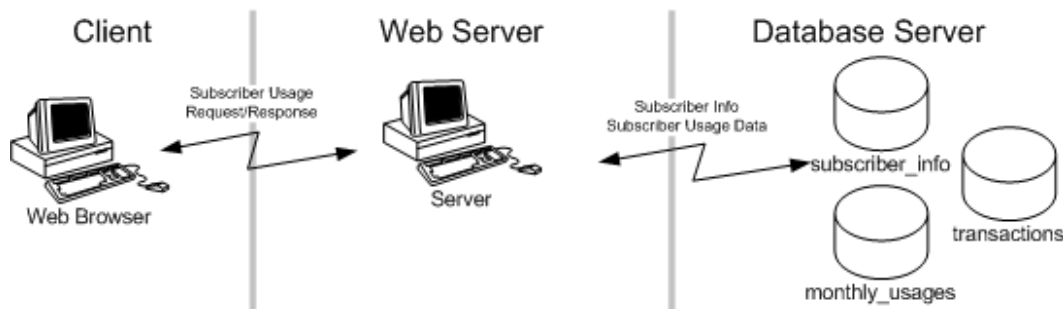
Management decided to pilot test this feature by adding the usage inquiry module to the current suite of applications available to our customer service representatives.

It will be used in the following customer service interaction:

1. The customer calls up the customer service hotline and gives his/her account number to the CSR.
2. The CSR inputs the account number into the system. If entry is successful, will be shown the account information of the subscriber for verifying the caller's identity.
3. Once the caller's identity has been verified manually, the CSR can look up the subscriber's current month's usage or the year-to-date usage history.

Architectural Overview

The usage inquiry module that you will be developing is a web application. The diagram below illustrates the system components involved in the proposed system architecture.



Database Tables

The SubscriberManager and SubscriberUsageManager are servlets that retrieve information from these three tables in the database. Refer to the Table Definitions below to aid you in making SQL calls to the database.

SUBSCRIBER_INFO

SI_IMSI	SI_NAME	SI_MSISDN	SI_IMSI	SI_ACC_NUM	SI_BIRTH DAY	SI_MAIDEN NAME
Primary Key	varchar 30	char 20	char 20	char 20	varchar2 14	varchar 20

TRANSACTIONS

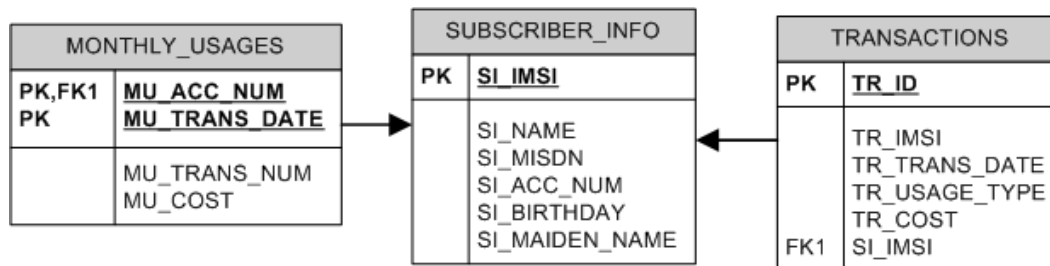
TR_ID	TR_IMSI	TR_TS	TR_UT	TR_ID	TR_COST
Primary Key	char 20	char 14	char 5	char 20	decimal 10,2

MONTHLY_USAGES

MU_ACC_NUM	MU_TS	MU_ACC_NUM	MU_TS	MU_TRANS_NUM	MU_COST
Primary Key		char 20	char 6	int	decimal 10,2

Database Architecture

The tables used in the project are related to each other thus:



Calling Sequence

The diagram below illustrates how page request and response are fulfilled within the web server. In the past segments, you were concerned with the presentation layer. The two servlets you will code today reside in the model layer.

Call Pattern References

The following is the list of SQL that will be executed by the SubscriberManager and SubscriberUsageManager classes.

Call #1

```
SELECT si_name, si_msisdn, si_imsi, si_acc_num, si_birthday, si_maiden_name,  
       si_ir_flag, si_ir_edate  
FROM subscriber_info  
WHERE si_acc_num= <account_number>
```

Call #2

```
SELECT tr_imsi, tr_trans_date, tr_usage_type, tr_id, tr_cost  
FROM transactions  
WHERE tr_imsi = <imsi>
```

Call #3

```
SELECT mu_acc_num, mu_trans_date, mu_trans_num, mu_cost  
FROM monthly_usages  
WHERE mu_acc_num = <account_number>
```

Pseudo Code

subscriberManager.java

The subscriberManager retrieves subscriber information from the Subscriber_Info table.

1. Import the following packaged classes
 - a. dataObjects.Subscriber
 - b. import connections.Database
2. Import the native java class
 - a. java.sql
3. Create the SubscriberManager class
4. Inside the SubscriberManager class,
 - a. Create a getSubscriber method that accepts an Subscriber object (containing the account number, returns the Subscriber data object and throws an Exception.

```
Subscriber getSubscriber(Subscriber subscriber) throws Exception  
{...}
```

- i. If the subscriber parameter passed is null throw an exception
 - b. Create a database connection using the Database class
 - c. Create an SQL call to the subscriber info table and pass this to a String variable
 - i. If you cannot open a connection, throw an exception
 - d. Inside a java error handler,
 - i. Execute the sql query
 - ii. Wrap the resultset in a statement class to make it read only and forward only
 - iii. Set the properties of the Subscriber object to the values returned in the resultset
 - iv. If no data is returned, throw an exception and print the error message in the stack
 - v. Whether successful or not, close the resultset, statement and connection
 5. Return the Subscriber data object containing the subscriber information as a String

subscriberUsageManager.java

The subscriberManager retrieves YTD usage information from the Transactions table and monthly subscriber usage information from the Monthly_Usages table.

1. Import the following packaged classes
 - a. dataObjects.SubscriberUsage
 - b. connections.Database
2. Import the following native java classes
 - a. java.util.Vector
 - b. java.sql.Connection
 - c. java.sql.Statement
 - d. java.sql.ResultSet
 - e. java.sql.SQLException
3. Create the SubscriberUsageManager class
4. Inside the subscriberManager class:
 - a. Create a getSubscriberUsage method that accepts an IMSI String parameter, returns a Vector, and throws an Exception

```
Vector getSubscriberUsage(String imsi)  
throws Exception{...}
```

 - i. If the String parameter passed is null throw an exception
 - b. Create and initialize a Vector and a SubscriberUsage object
 - c. Create a database connection using the Database class
 - d. Create an SQL call to the transactions table and pass this to a String variable
 - i. If you cannot open a connection, throw an exception
 - e. Inside a java error handler,
 - i. Execute the sql query
 - ii. Wrap the resultset in a statement class to make it read only and forward only

- iii. Set the properties of the SubscriberUsage object to the values returned in the resultset
 - iv. If no data is returned, throw an exception and print the error message in the stack
 - v. Whether successful or not, close the resultset, statement and connection
5. Return the SubscriberUsages vector object containing the subscriber usage records
6. Still inside the subscriberManager class:
- a. Create a getSubscriberUsagePerMonth method that accepts an account_number String parameter, returns a Vector, and throws an Exception

Vector getSubscriberUsagePerMonth(String account_number) throws Exception{...}

- i. If the String parameter passed is null throw an exception
 - b. Create and initialize a Vector and a SubscriberUsage object
 - c. Create a database connection using the Database class
 - d. Create an SQL call to the monthly usages table and pass this to a String variable
 - i. If you cannot open a connection, throw an exception
 - f. Inside a java error handler,
 - i. Execute the sql query
 - ii. Wrap the resultset in a statement class to make it read only and forward only
 - iii. Set the properties of the SubscriberUsage object to the values returned in the resultset
 - iv. If no data is returned, throw an exception and print the error message in the stack
 - v. Whether successful or not, close the resultset, statement and connection
7. Return the SubscriberUsages vector object containing the subscriber usage records



Unit Testing

Replace this page with the Unit Test Script and Unit Test Data documents.

