

Lab Assignment (#2)

Introduction to Eclipse IDE

Points: 50

Instructions:

- Submit the following:
YourASUID-Lab2.zip This compressed folder should contain the following files:
 1. The complete Workspace after DELETING ALL THE JAR FILES. You will receive a penalty of 10 points if jar files are not deleted
 2. Optional readme.txt containing any instructions you want to provide to the instructor.
-

Install Eclipse:

1. Download Eclipse from <http://www.eclipse.org/downloads/index.php> (Pick the appropriate Operating System).
 2. Make sure you pick Package solutions - Eclipse IDE for Java EE Developers
 3. Extract all files from the downloaded compressed folder.
 4. Place the extracted directory at a location of your choice (e.g. C:\eclipse-jee-kepler-R-win32).
 5. Double-click eclipse.exe under eclipse directory to launch Eclipse IDE (You can create a shortcut to eclipse.exe on your desktop).
-

Objectives:

1. Create an Eclipse project from an existing source directory
2. Understand how Eclipse manages libraries for building source code.
3. Observe the Eclipse builder executing when project state changes.
4. Demonstrate several of the Eclipse “intentional programming” features that increase developer productivity.

Most Java projects consist of one or more source directories, include multiple jar libraries, and have complex compilation and deployment scripts that create files such as jar, war, ear, sar, etc. This lab walks you through the creation of an Eclipse Java project with these types of characteristics.

The steps to this exercise are to:

1. Create the project from an existing directory
2. Examine the project
3. Configure project libraries
4. Run the project
5. Test the project
6. Debug the project
7. Navigating and Understanding the Code

Step 1: Create the project from an existing directory

- Download Lab-Code.zip from blackboard. Uncompress it and place it under C:\SER215Labs\EclipseOverview.
- Open Eclipse IDE and select a workspace (e.g. C:\SER215Labs\EclipseOverview).
- If a Welcome screen is prompted, click on Workbench icon on top right corner of the screen.
- Right-click in the Project Explorer or Package Explorer, select New > Project > Java Project, and click Next. You will see a dialog similar to Figure 1.
 - Enter a project name (e.g. Banking).
 - Uncheck “Use Default Location”.
 - Click on “Browse” and navigate to the project directory included with the course materials (e.g. Lab-Code\project) and click Next.
 - Change the default output folder to “Banking/build” as that is the directory the project’s build script (an Ant build.xml file) expects. (Example screen shown in Figure 2)
 - DON’T CLICK FINISH YET.
- Notice Java recognized the src directory as Java source and the jar files in the lib directory as libraries (click the Libraries tab to see jar files).
- Click the Libraries tab.
 - These are the libraries the builder uses for compilation (CLASSPATH) and they were discovered during the import (we will add libraries to this list later in the lab).
 - Click “Order and Export” to show the search order used during compilation.
 - Accept the defaults by clicking “Finish”.
 - If you see a message about “Open Associated Perspective?”, click “Yes”. Notice there are error markers; you will fix them soon.

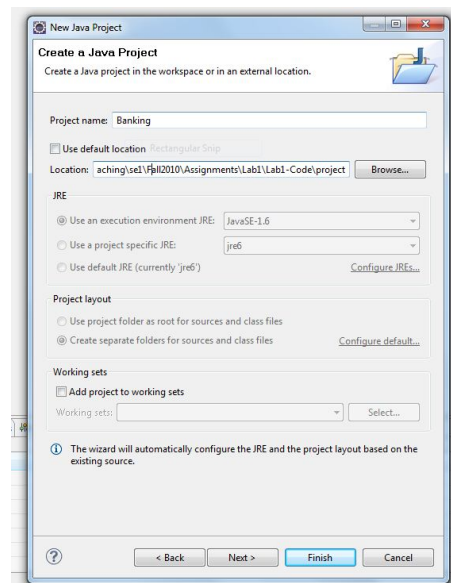


Figure 1: New Java Project

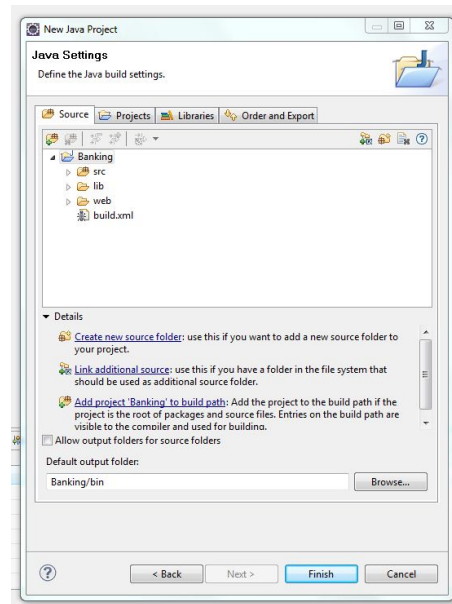


Figure 2: Project Settings

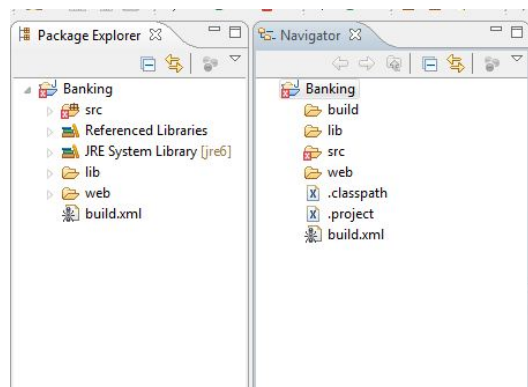


Figure 3: Package and Navigator Explorer

Step 2: Examine the Project

- Open the “Navigator” View (Window > Show View > Navigator) and look at the project’s file structure.
- Drill into the build directory and notice Eclipse already compiled the classes.
- Compare other differences between the physical structure in the Navigator view and the Java-specific Package Explorer view (hint - drag one view to the side to show them side-by-side) - See Figure 3:
 - Jar files are pulled up and brought to top level under Referenced Libraries
 - Java source directories are rolled up and only show directories containing classes
 - The build directory does not exist in Package Explorer
 - Markers indicate Java-specific information such as errors and warnings
- Close the Navigator view

Step 3: Configure project libraries

Next we'll fix the errors by adding libraries and instructing Eclipse to use them - See Figure 4 for sample Error window.

- Using Window's Explorer (or Finder for Mac), copy both servlet-api.jar and jsp-api.jar (located in C:\SER215Labs\EclipseOverview\Lab-Code) to the project's lib directory (C:\SER215Labs\EclipseOverview\Lab-Code\project\lib). In the Eclipse Package Explorer, right-click on the project, select "Refresh", then open the lib directory and verify the jar files are there.
- While the library files are now part of the project and would be used for building with tools like ant, the Eclipse Java builder is not aware of them (notice there are still errors in banking.gui.servlet). Tell the Eclipse Java Builder to include those jar files by doing the following:
 - Open project properties (right-click on the project)
 - Navigate to: BuildPath > Configure Build Path, click "Libraries" tab and select "Add Jars"
 - Drill into the lib directory and add both the Servlet and JSP API libraries. If you don't see any libraries, you did not "Refresh" in the previous step.
 - Click OK and notice Eclipse adds those libraries to the Referenced Libraries and rebuilds the project.
- The remaining errors are unresolved references to JUnit classes. JUnit is a popular Java unit testing facility and Eclipse comes bundled with JUnit libraries.
 - Open the Problems view in the lower view area (or Window > Show View > Problems) and open the error listing.
 - Double click the first error; results in opening the file at the error line in the Java editor.
 - Hover the mouse over the error marker and observe the error.
 - Right click the marker on the first error and select "Quick Fix".
 - Eclipse lists several options to repair this error - select "Add JUnit3 or JUnit4 library to build path".
 - Eclipse rebuilds the project and the errors are fixed.
 - Open the project properties (right-click Properties) and observe JUnit was added to Java Build Path > Libraries. Junit could have also been added here via the "Add Libraries" button.

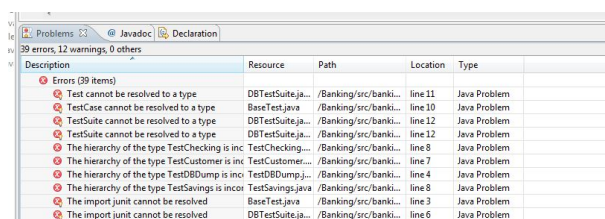
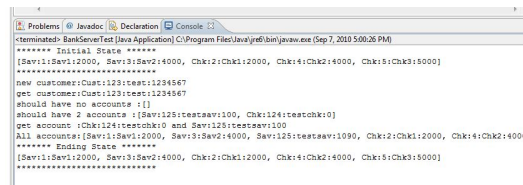


Figure 4: Problems Window

Step 4: Run the Project

- Simple program execution. In the Package Explorer, right click on src > banking.server > BankServerTest and select Run As > Java Application.
- Observe the execution in the console. This simple Eclipse program execution uses the project build path set up in Eclipse, including referenced jar files.
- Complex program execution. Launch configurations provide more complex execution directions. In the menubar, pull down on Run > “Run Configurations” (screenshot of dialog shown in Figure 6).
- This dialog allows complete control of the execution environment for the Java application.
 - Right click on Java Application and select “New”.
 - Set the name to “Test Run”.
 - Click on the “Search” button in the “Main class” pane and notice it lists all classes with a main() method.
 - Select the BankServerTest class.
 - Click the Arguments tab and enter the string “-verbose:class” in the “VM Arguments” text box.
 - Click Run and observe this execution has turned on verbose class loading.
- Redirecting Console output.
 - Open the “Run configuration” used previously, click the Common tab.
 - Check the “File” box under “Standard input and output” and enter a path to a file (e.g. C:\SER215Labs\EclipseOverview\output.txt).
 - Click Apply and Ok. Rerun the application and view the file output (example shown in Figure 5).



```

***** TestLab State *****
[Serv1:Serv12000, Serv3:Serv2:4000, Chk1:2:Chk1:2000, Chk1:4:Chk2:4000, Chk1:5:Chk3:8000]
*****
new customer: Cust:123: test:1234567
get customer: Cust:123: test:1234567
should have no accounts: {}
should have 2 accounts: {Serv125:test:av100, Chk124:test:chk10}
get account: {Chk124:test:chk10 and Serv125:test:av100}
All accounts: {Serv1:Serv12000, Serv3:Serv2:4000, Serv125:test:av1090, Chk1:2:Chk1:2000, Chk1:4:Chk2:4000,
***** Ending State *****
[Serv1:Serv12000, Serv3:Serv2:4000, Chk1:2:Chk1:2000, Chk1:4:Chk2:4000, Chk1:5:Chk3:8000]
*****
  
```

Figure 5: Output

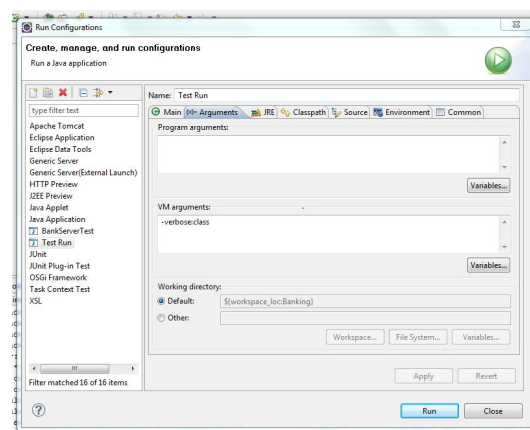


Figure 6: Run Configuration

Step 5: Working with JUnit inside Eclipse

The JUnit framework is a popular method for testing code. Eclipse has built-in support for JUnit including a dashboard for running tests and test suites.

- Expand the package `banking.db.test`, right-click the `DBTestSuite`, and click `Run As > JUnit Test`.
- Observe the output in the Console view and the results (13 tests in 4 files) are all successful (as shown in Figure 7).
- Force an error by double clicking `testRead` under `banking.db.test.TestChecking`
- Open the file and change the `c.getId()` assertion to 3 (on line 13).
- Save the file and re-execute the program by typing `<ctrl>F11` (last launched).
- `TestChecking.testRead` now fails.
- Change the code back to a “2” and rerun the test suite to verify it works.

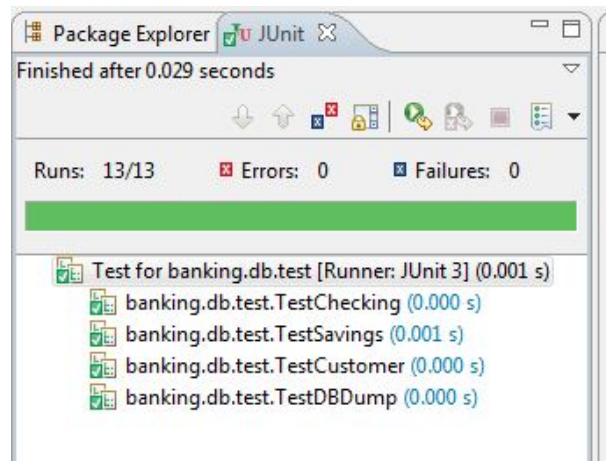


Figure 7: JUnit Test

Step 6: Debugging the project

- Add a breakpoint to the program. Back in the Package Explorer, open the file `banking.server.BankServerTest.java`.
- Right click on the grey bar next to the first statement in `main()` and select “Toggle Breakpoint”.
- Debug the program by right-clicking `BankServerTest` in the Package Explorer view and selecting “Debug As” > Java Application.
 - If prompted to pick a Run Configuration, pick any one.
 - If prompted to change to the Debug Perspective, select “Yes”.
- Use the Step Over button to step over the first line. (Debug tool bar shown in Figure 8)
- Notice “server” now exists in the “Variables” view and can be expanded to view it’s attributes.
- Step over 3 more times, stopping on the `newCustomer()` line.
- Step into `newCustomer()` which will take you into the call to the method.
- Notice the “Variables” view contains the parameters, name and phone, as well as a reference `this` (Example in Figure 9).
- Also notice the debug view contains the stack trace of all called methods for each thread in the execution.
- Click on any of the items in that stack trace and observe the “Variables” view update with the local variables from the selected stack frame and the editor changes to highlight to the location in the code.
- Modify the value of the phone by clicking the *Value* column of the “Variables” view. Change the value to “999999” and press the Enter key.
- The Eclipse debugger allows us to modify the state of the running program.
- Step into the next line (the call to `create()` in the return statement).
- Observe the stack trace showing the call hierarchy for the `main()` thread in the `BankServerTest` process has added another stack frame (Example shown in Figure 10).
- Click resume and let the program run to completion. Since there are no more breakpoints, the program terminates normally.
- Observe the output that the new customer’s phone number is indeed “999999”.



Figure 8: Debug Buttons

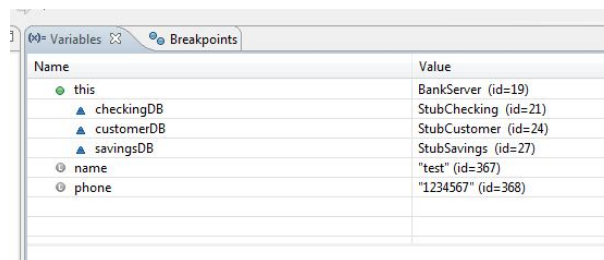


Figure 9: Debug Variables

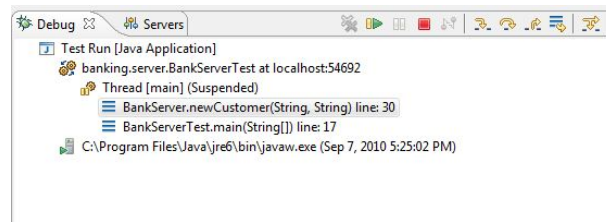


Figure 10: Stack Trace

Step 7: Navigating and Understanding the code

Eclipse contains many tools and features that provide software developers the ability to better understand and to efficiently navigate their system artifacts. This lab walks through several interesting tools to help developers navigate the system's code.



Figure 11: Hierarchy

- **Type Hierarchy:**
Open `banking.entity.core.Account.java`. Right-click on `Account` and select “Open Type Hierarchy”. Observe the classes in the Hierarchy View. Select classes in the view and observe that the pane below updates reflecting with annotations of the abstract methods, overridden methods, constructors, etc. This pane is similar to the Outline view for a class. Open the Outline view if it is not already open (Window > Show View) to observe. (Example in Figure 11)
- **Navigating in the Java Editor:**
Select the attribute “id” and notice it as well as all uses become highlighted. In the editor’s left column, click one of the highlight markers to move to the various uses. Select the “id” parameter of the `Account` constructor and notice the highlight is indeed context sensitive.

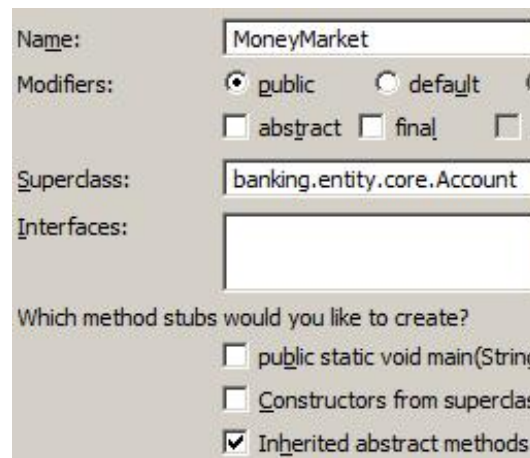


Figure 12: Create Class

Select “Asset” in the Account declaration at the top of the file. Notice the editor highlights the overridden methods defined in Asset. If not already open, open the Outline View (Window > Show View) and use the outline to navigate the file in the editor. Select a method or attribute to navigate.

- Editing code using the Java Editor:

Eclipse editor has many features to automate common activities of a developer. First, make a new class.

- Right-click banking.entity.core and select New > Class (Example in Figure 12).
- In the New Class dialog set the name to “MoneyMarket”.
- Click the Browse button next to Superclass and in the “Choose a type” field enter “Account” which will list all classes with the name account.
- Select Account from banking.entity.core.
- Ensure “Inherit abstract methods” is checked and click Finish.
- Notice there is one error - the parent class, Account, constructor requires parameters which requires MoneyMarket’s constructor to call it - and one warning - MoneyMarket is serializable and should have an id.
- Right click the error marker, select Quick Fix, and select “add constructor” (it does not matter which one for this lab) which will fix the error.
- Save the file (ctrl-S), select the warning marker as before for the error and use Quick Fix to add a generated serial version ID.
- The generated code is shown below.

```
private static final long serialVersionUID = 7940718811556954792L;
```

```
public MoneyMarket(int id, int custId, String n, int b, int x, int y, int z) {  
    super(id, custId, n, b);  
}
```

- Next, add three attributes x, y, and z (manually - finally entering some code!)

```
public class MoneyMarket extends Account {
```

```
int x;  
int y;  
int z;
```

- Right-click anywhere in the Java Editor to bring up the editor's context menu and select Source > Generate Getters and Setters.
- Select x/y/z, choose the insertion point and your sort by criteria, and click OK.
- Next make a constructor for these fields using the same context menu and selecting Source > Generate Constructor using Fields.
- Select x, y, z, select your insertion point and click OK. Notice the constructor also includes all properties required by the parent.

```
public MoneyMarket(int id, int custId, String n, int b, int x, int y, int z) {  
    super(id, custId, n, b);  
    this.x = x; this.y = y; this.z = z; }
```

- Next we decide x, y, and z, should be properties of all Accounts, not just MoneyMarket.
- Use the same context menu and select Refactor > Pull Up.
- Select banking.entity.core.Account as the destination type and check all members related to x, y, and z.
- Click Next twice to the Refactoring dialog and observe the changes being made to the files.
- Click Finish and observe the code changes in MoneyMarket and Account classes.
- Eclipse can undo Refactor changes no matter how many files are involved.

- Understanding call structure with Call Hierarchy and References:

Call Hierarchy shows all the calling sequences for a given method. In Account.java, right-click on getId() and select "Open Call Hierarchy". Notice the Call Hierarchy View shows all calls to this method. Expand any of the calls and observe call hierarchy until you locate the originating main() or Thread.run() method. Navigate to the locations in the code by double clicking on any of the method calls. References show all methods that refer to a selected variable or method, but organized by the classes that contain those methods instead of a call hierarchy. Right-click again on Account.getId() again and select References > Project. Expand the references in the Search window and observe the same methods from Call Hierarchy, but organized by classes that refer to the method. Select other items for referencing, for example, "Account" and "String". Open one of the files by double-clicking the link and observe the editor is highlighting occurrences of the search item. Notice also that, since this was a search result, the editor marks the matches in the right column.

- Adjusting warnings and errors:

There are several warnings in the project. Open the Problems View to observe them. We decide these warnings are allowed in our project and we would like to tell Eclipse to stop warning us. Open the preferences in Window > Preferences > Java > Compiler > Errors/Warnings. Under "Potential programming problems", turn "Serializable class without serialVersionUID to Ignore". Under "Generic Types" ignore "Usage of a raw type" and "Unchecked generic type operation". Click OK and allow Eclipse to rebuild the projects. All problems should be eliminated.

- Refactoring:

Changing the names and locations of code in a project is typically painful due to references from other parts of the system. One of the Eclipse features for programming by intention is refactoring. In the Package Explorer, show that there are 22 references to Account.java (right-click on Account.java > References > Workspace). What is the impact of changing this class name or the class location? Use refactoring to change the name first by right-clicking on the Account and selecting Refactor > Rename. Set the new name to “MyAccount” and click Next. The preview shows all changes that will be made to source files. Select a source file and navigate to the changes by selecting the markers in the right column. Either select cancel at this point or if you perform the rename, you can rename the class back to the name “Account” or use the Eclipse undo mechanism (ctrl-Z). Also try Refactor > Move and change the package location for Account and select Preview. Again, either click cancel or move Account back to banking.primitive.core.

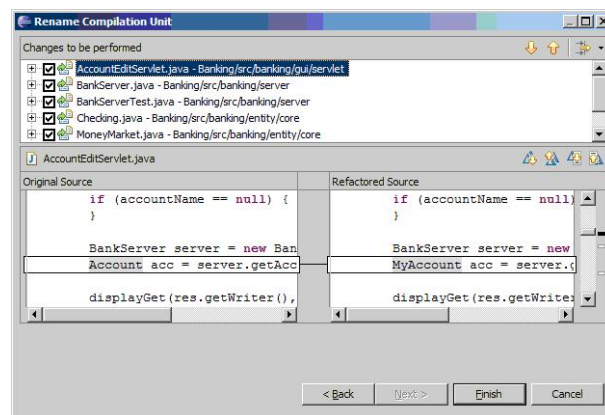


Figure 13: Refactor