Homework 6: Design Patterns, Java-UML conversion and UML-Java Conversion using Rational Software Development Platform

Name : _____

Student Number : _____

Laboratory Time : _____

Objectives

- Create a Class Diagram using the Reverse Engineering feature in Rational Software Development Platform
- Create a Class Diagrams using the Design Patterns feaure in Rational Software Development Platform
- Generate Java code from a Class Diagram in Rational Software Development Platform
- Modify and run a Java application in Rational Software Development Platform

Preamble

In this lab, we will learn how to create a UML Class diagram from existing Java source code. We will also learn how to create Class Diagrams including Design Patterns in Rational Software Development Platform. Specifically, we will be using the Observer Design Pattern. Finally, we will learn how to generate Java code from a UML Diagram and how to run Java applications using the Rational tool.

Grading Checklist (30 points)

By the end of the laboratory session, you need to demonstrate to the TA that you can do the following tasks. The TA will check off the items below that you have completed and collect this cover page from you.

- ☐ Create a Class Diagram from Lunar Lander code
- ☐ Create a class Diagram showing the Observer Design Pattern, and all attributes and methods for the concrete classes
- ☐ Generate Java code from the Class Diagram
- ☐ Add Java code and run an application using the Observer pattern
- ☐ Implement the update method for the Observer that prints the employee's names

TA Initials: ____

Instructions for the Laboratory

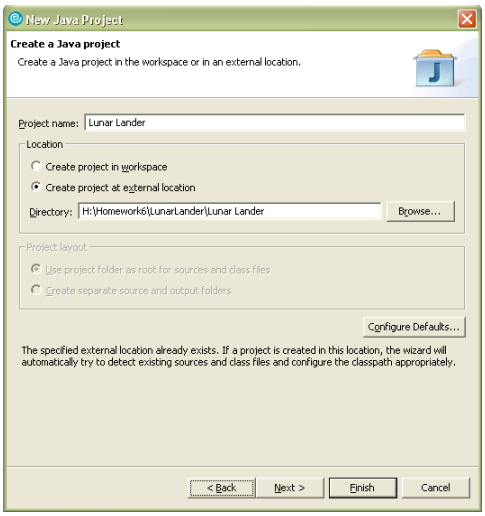
Task 1: Import Lunar Lander source code used in Lab 1.

Download the source file LunarLander.zip used in Homework 2 from the course webpage. Unzip the file into "H:\Homework6\"

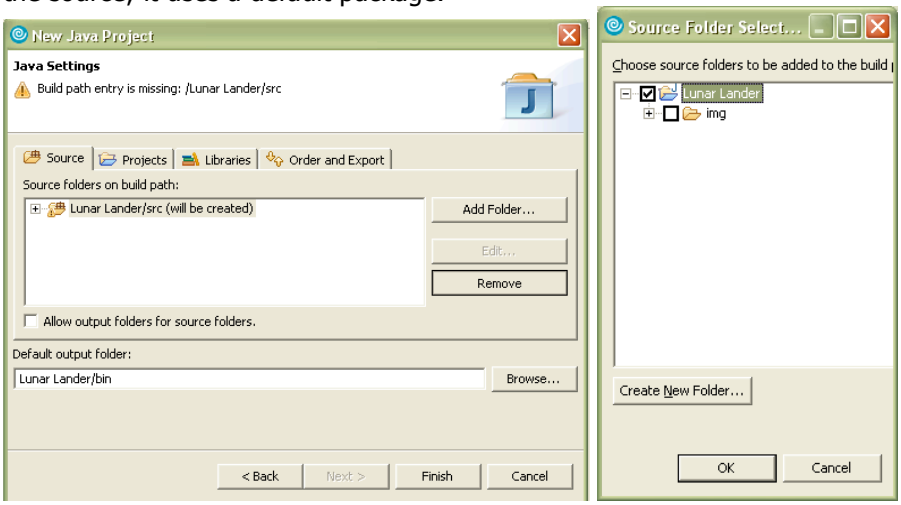
Open Rational Software Development Platform and perform the following actions:

a) Create a Java project from an existing source code in Rational Software Development Platform using the menu: File -> New ->Project. Then select Java Project from the new dialog box. This is very similar to creating a Java project in Eclipse. Click Next. If you are prompted with a dialog box asking for Confirmation Enablement of "Java Development", click OK.

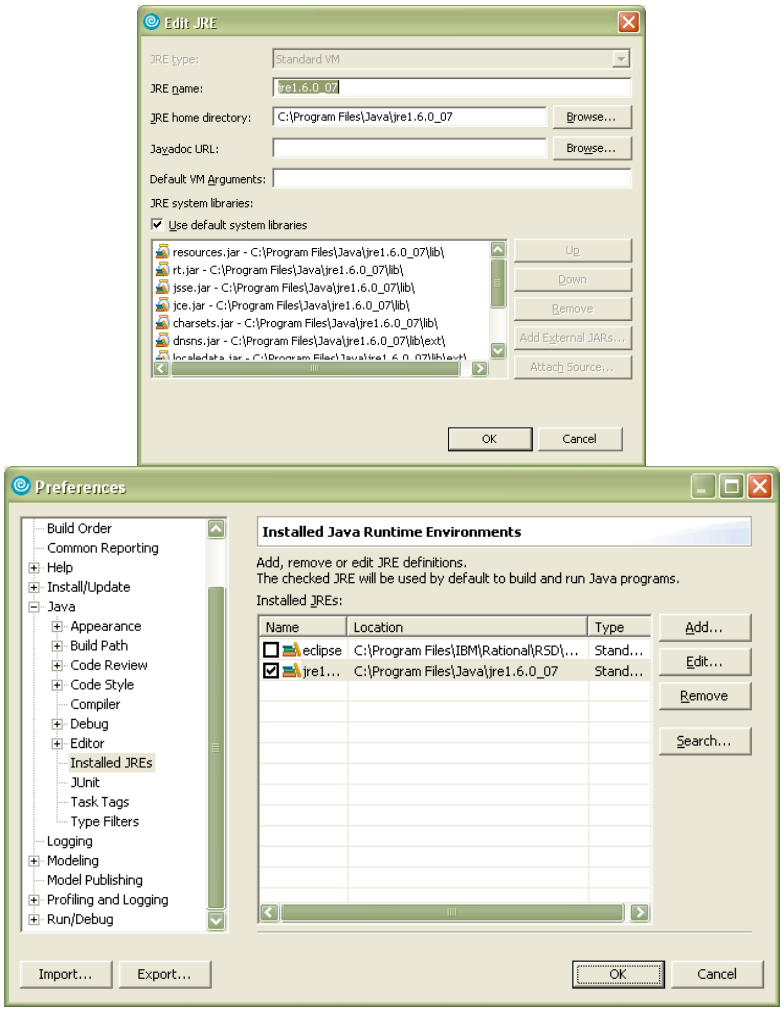
b) Name the project Lunar Lander and select the radio button "create project from existing source". Browse to the directory containing Lunar Lander code and select the Next button.



c) Configure the build path of the project by removing the "Lunar Lander/src" and add the project folder instead. Click Finish. We need to do this because Lunar Lander does not use src to keep the source; it uses a default package.

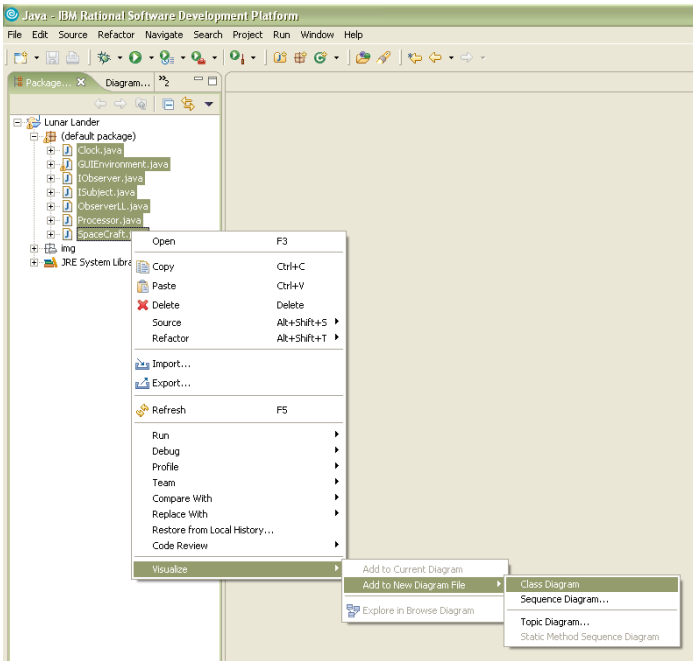


d) If you receive a notification to switch to the Java Perspective, select Yes.
e) You will have an error due to JRE incompatibility. Fix it by adding new installed JRE. Select Windows->Preference->Java->Installed JREs. Uncheck the current JRE and Add a new one. Name the new JRE, jre1.6.0_07 and use C:\Program Files\Java\jre1.6.0_07 as JRE home directory. Click OK. Once a new JRE is added, select it and click OK.

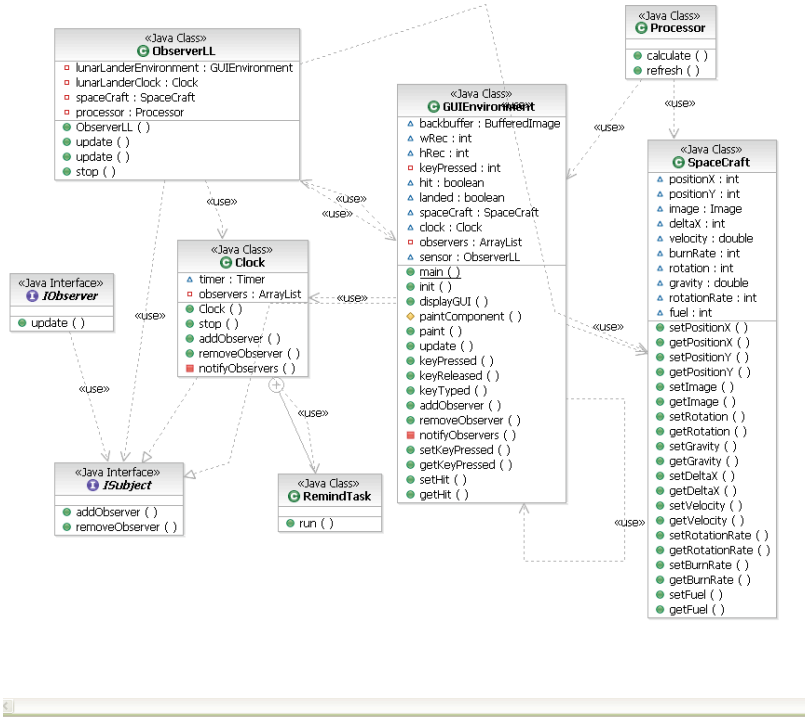


Task 2: Create a Class Diagram from Lunar Lander code

a) In the package explorer view, select all classes in the default package. Then, Right Click ->Visualize->Add to New Diagram File->Class Diagram.



b) When the new class diagram is shown in the editor, rearrange classes so that the classes are not overlapping and their relationships are not tangled.



Task 3: Familiarize yourself with the scenario.

INF111 Software Inc, a company which develops software for grocery stores, has recently started to implement an emergency preparedness plan for protecting its employees from emergencies such as fires or earthquakes. INF111 Software Inc needs to know the list of employees who are in the

building at any moment, so that this list could be used to alert employees and take emergency actions accordingly.

INF111 Software Inc has provided a badge, which has an RFID tag located on the back, to each employee. RFID readers have been installed in the two entrances of the company. Every time an employee enters to the building the RFID reader will read the employee's RFID tag and it will add the employee's first and last name in the list of people who came to work.

Your task is to create an application that will receive the employee's name and add it to a list. Each time and employee's name is added, you should print the number of employees and the list of employee's names that are in the building.

While thinking about the design of your implementation, you have determined that you should keep track of the list of people and trigger two events when a new element is added. You have decided that you can use the Observer Pattern to create this application, where the list of employees will have the Subject role and the classes that handle the counting of employees and the printing of names will have the Observer role.

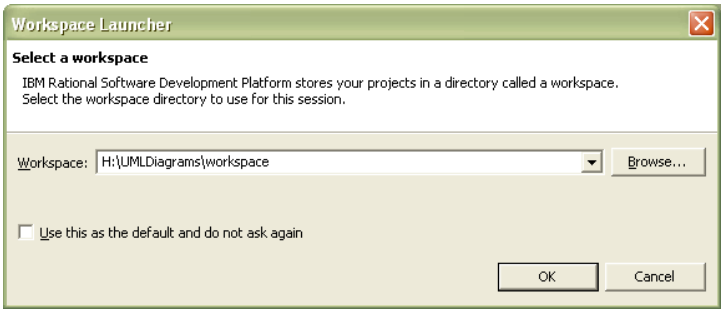
Knowing that Rational Software Development Platform provides support to design applications using Design Patterns and also provides support to generate code, you decided to model your system with this tool and then add your customized code to meet the requirements.

Task 4: Set up a Project in Rational Software Development Platform

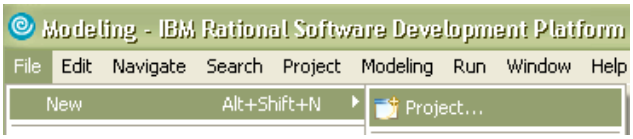
For this task, you will create a class diagram to include the Observer Design Pattern.

a) Make sure you have the directory "UMLDiagrams" in your H directory. If not, create it.
b) Start Rational Software Development Platform.

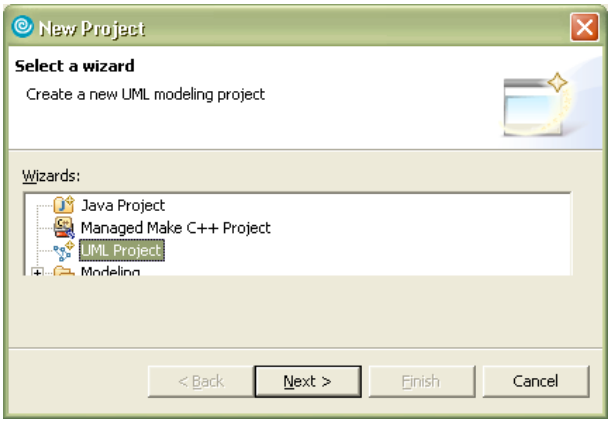
c) Change the Workspace location to H:\UMLDiagrams\workspace



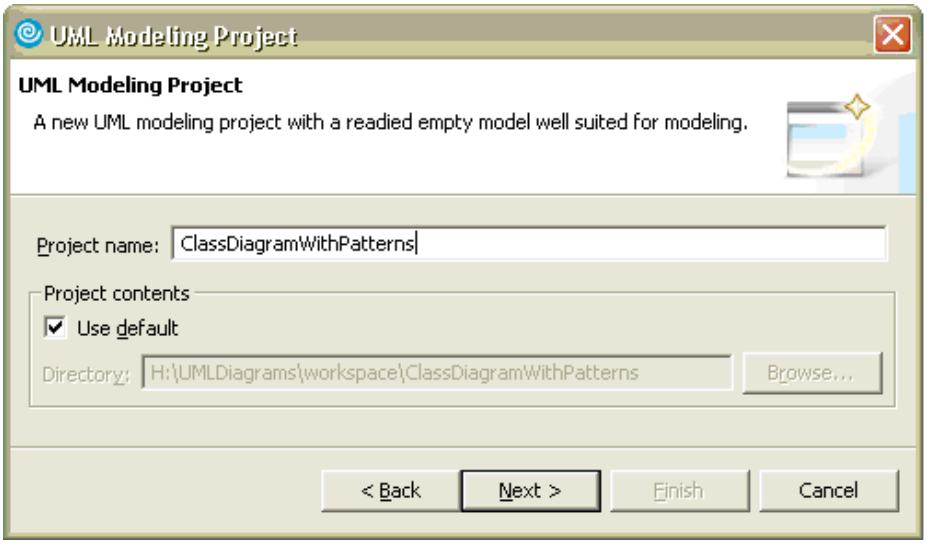
d) Create a new Project. Go to File -> New -> Project



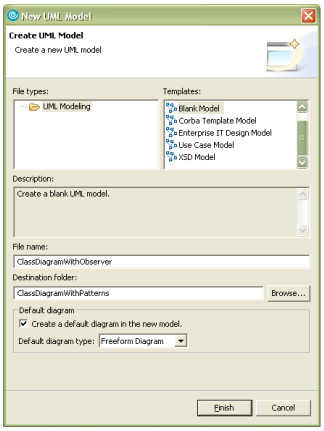
e) Select the UML Project wizard. Click Next.



f) In the Project name field, enter "ClassDiagramWithPatterns". Click Next.

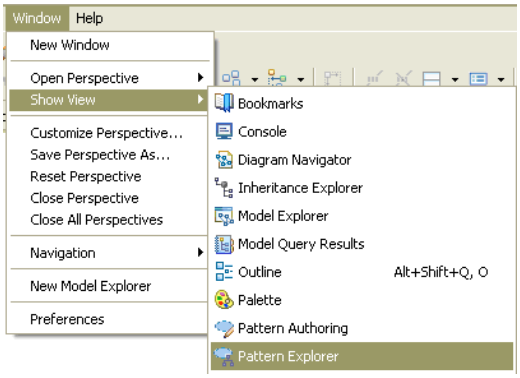


g) In the File name field, enter "ClassDiagramWithObserver". Click Finish.

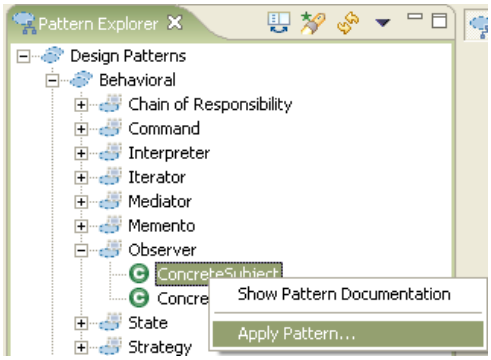


Task 5: Create a Design Pattern in Rational Software Development Platform

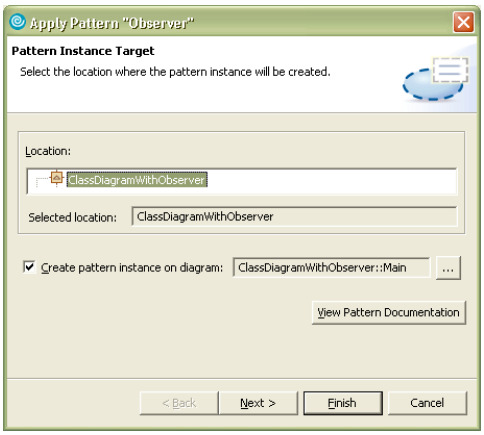
a) Open the Pattern Explorer View. To do that, go to Window -> Show View -> Pattern Explorer.



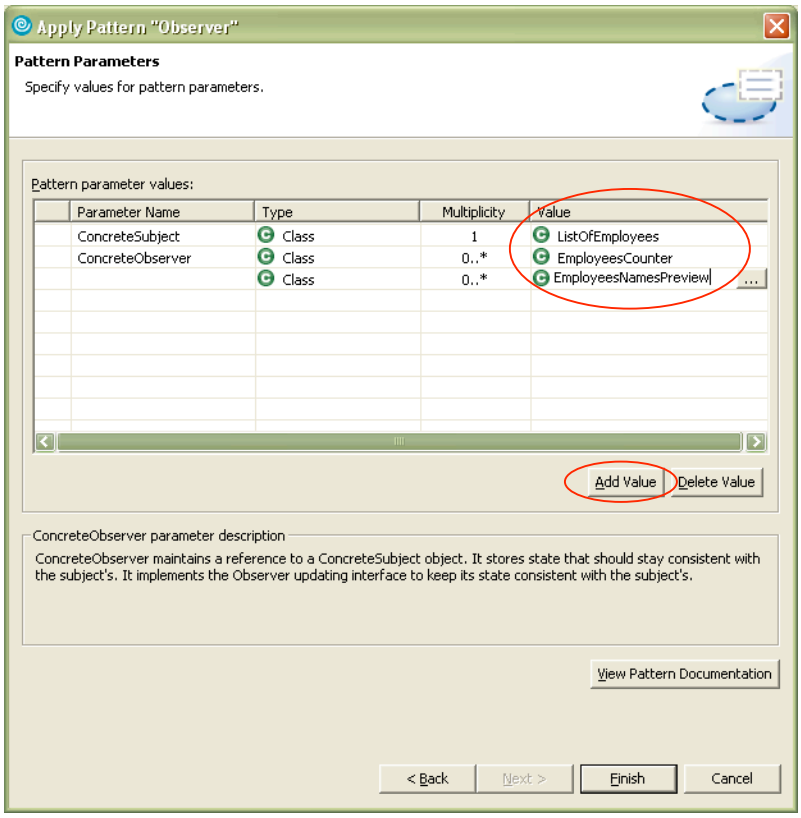
b) In the Pattern Explorer, expand Design Patterns, expand Behavioral, and expand Observer. Select ConcreteSubject. Right click on it. Select Apply Pattern.



c) In the Pattern Instance Target, select ClassDiagramWithObserver as the Location. Click Next.

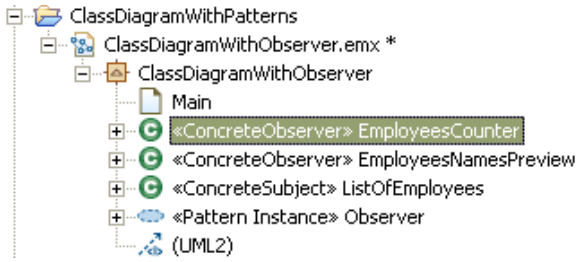


d) In the Pattern Parameters window, specify the Value "ListOfEmployees" for the ConreteSubject parameter and the Value "EmployeesCounter" for the ConcreteObserver parameter. Click on Add Value and specify the Value "EmployeesNamesPreview." Click Finish.

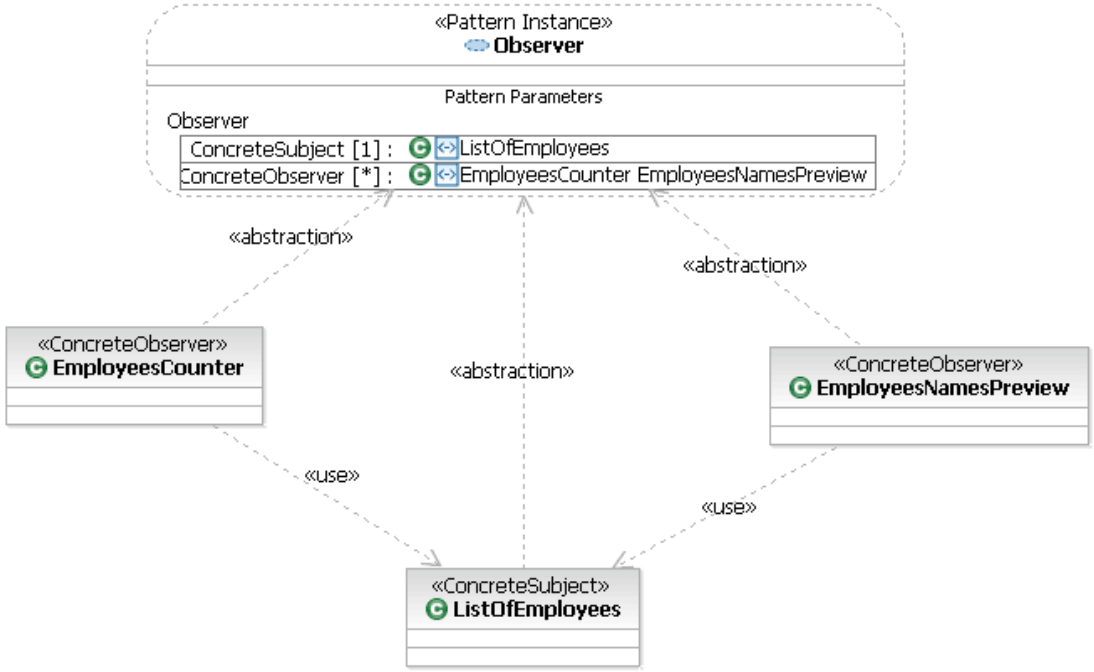


e) You will see the following pattern in your class diagram.

f) In the Model Explorer, you will see that 3 classes have been created. Drag and drop each of the three classes to your diagram.



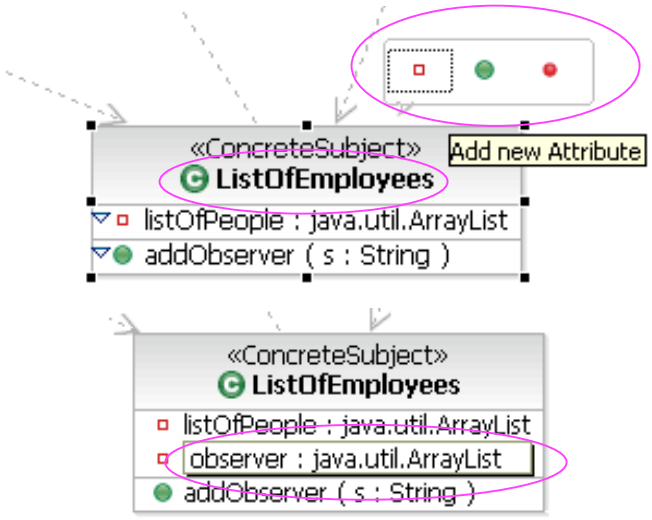
g) Your will see the Observer Pattern Instance and also one ConcreteSubject and two ConcreteObserver classes in your diagram.



h) Add all the attributes and methods to each class as shown in the following diagram.

Note: To see the complete signature of the methods. Select one class. Right click. Select Filters -> Show Signature.

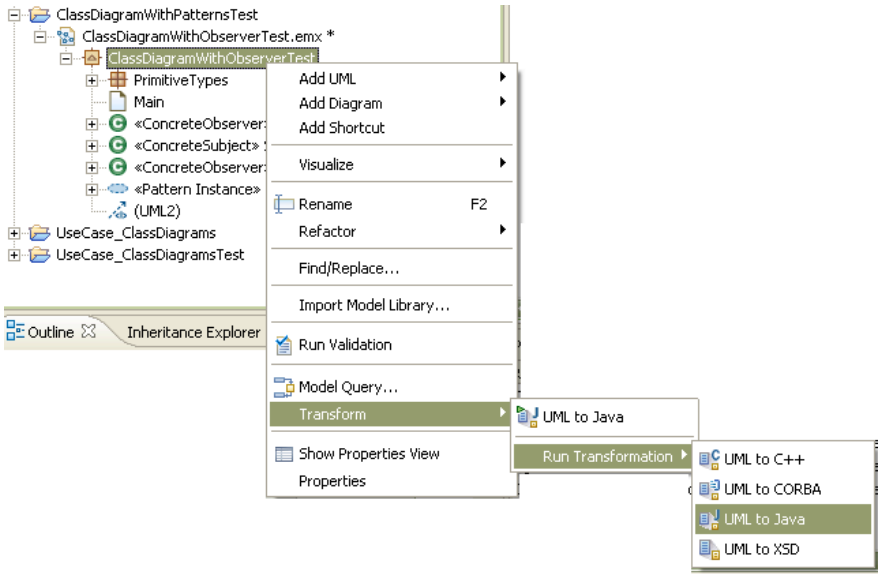
** **NOTE:** Notice that in ListOfEmployees class, you have two variables of type java.util.ArrayList. To create an attribute of a primitive type ArrayList, click on the Class name until you see a pop up menu to add new Attribute. Click on "Add mew Attribute" button. Once a textbox appears for you, type in the attribute, keep typing both name and the type as seen in the picture below. Once you do this, you will have an ArrayList as a type in a primitive type group which you can use when creating attributes with other methods. Do the same for java.util.Observer.



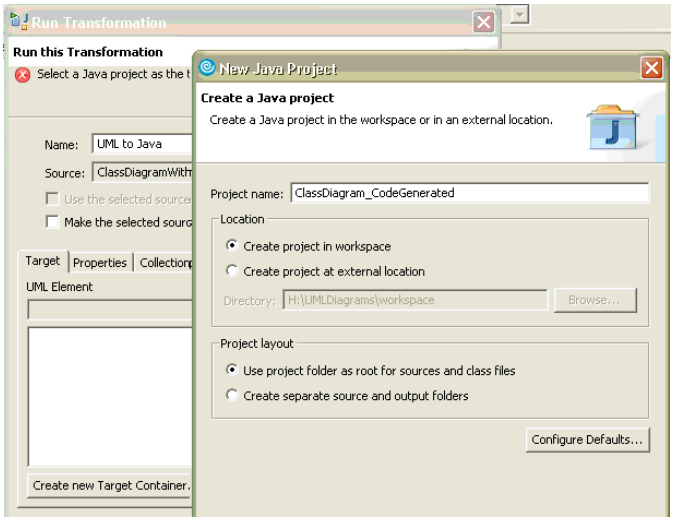
Task 6: Generate Java code from the Class Diagram using Design Patterns in Rational Software Development Platform

Now that you designed your application, you can generate the Java code for it using the Rational tool.

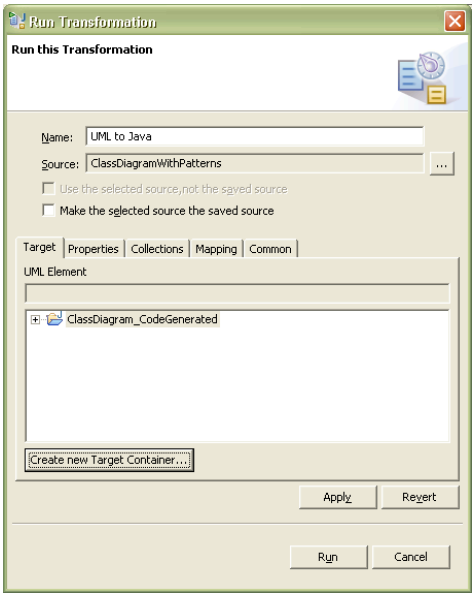
a) Select the model ClassDiagramWithObserver. Right click and select Transform -> Run Transformation -> UML to Java.



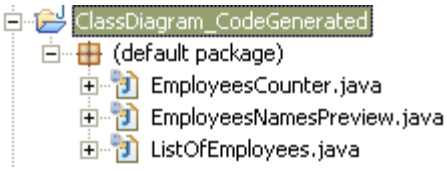
b) Click on "Create new Target Container." In the Create a Java project window, enter **ClassDiagram_CodeGenerated**. Click Finish.



c) In the Run Transformation window, click Apply and Run.



d) Notice that the Java code has been generated for the classes you had in your diagram. Inspect the code that has been generated in each class.



Task 7: Add the Java code to run an application using the Observer Pattern

As you can see in each class, the generated Java code matches with the properties and method signatures you specified in you class diagram. The code generation is a very helpful feature because you already have the code with all the structure of your classes. However, you need to add the

specific functionality that you want each method to execute. Now, you will add the code that is needed to run the application based on the created structure.

Add source code to Classes

Using the Model explorer make the following changes in the Java classes.

a) Class: ListOfEmployees.java

- The declaration of the private properties listOfPeople and observers is already in the source code. You should modify this declaration to add the initialization of the ArrayList in each case. The properties should look like:

```
private ArrayList listOfPeople = new ArrayList();
……
private ArrayList observers = new ArrayList();
```

- Method: add. Add the following lines:

```
listOfPeople.add( s );
notifyObservers();
```

- Method: iterator. Replace return null for:

```
return listOfPeople.iterator();
```

- Method: addObserver. Add the following line:

```
observers.add( o );
```

- Method: removeObserver. Add the following lines:

```
observers.remove( o );
```

- Method: notifyObservers. Add the following lines:

```
// loop through and notify each observer
Iterator i = observers.iterator();
while( i.hasNext() ) {
    Observer o = ( Observer ) i.next();
    o.update( this,o );
}
```

b) Class: EmployeesCounter.java.

- Import java.util.Iterator in the class.

- Method: EmployeesCounter. Add the following lines:

```
this.list = list;
list.addObserver( this );
```

- Method: update. Add the following lines:

```
if( o == list ) {
    System.out.println( "The contents of the list of employees have changed." );
    int counter = 0;
    Iterator i = list.iterator();
    while( i.hasNext() ) {
        String line = ( String ) i.next();
        counter++;
    }
    System.out.println( "The total number of employees in the building is: " + counter);
}
```

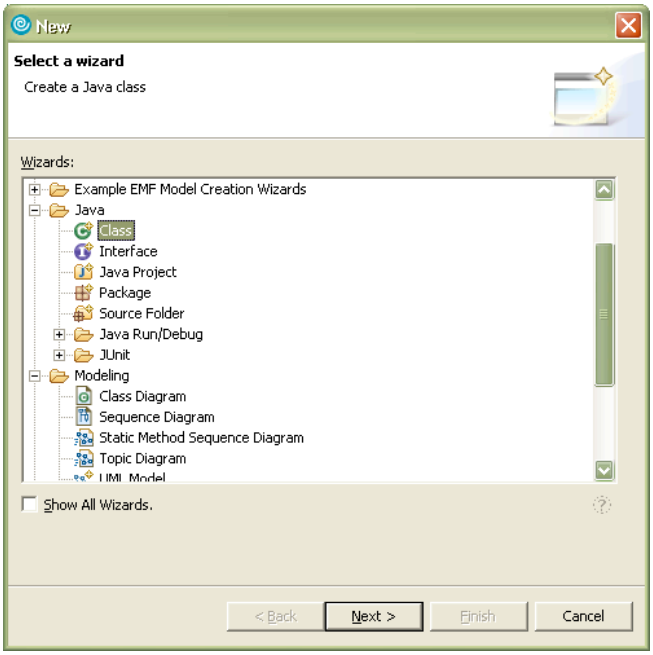
c) Class: EmployeesNamesPreview.java

- Method: EmployeesNamesPreview. Add the following lines:

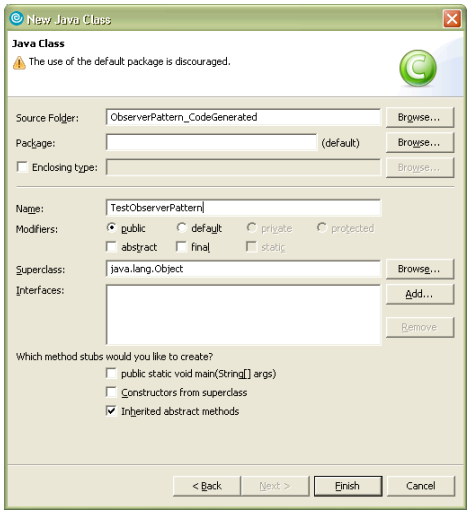
```
this.list = list;
list.addObserver( this );
```

Create class to run the application using the Observer Pattern

d) Create a Class called TestObserverPattern. Right click on the project ClassDiagram_CodeGenerated. Select New -> Other. Select Java -> Class. Click Next.



e) Enter the name of the Class: TestObserverPattern. Click Finish.

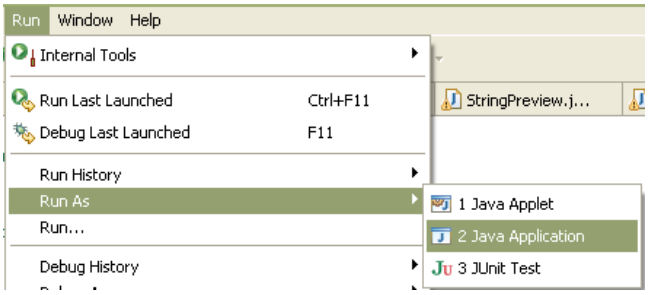


f) Add the following method to the class.

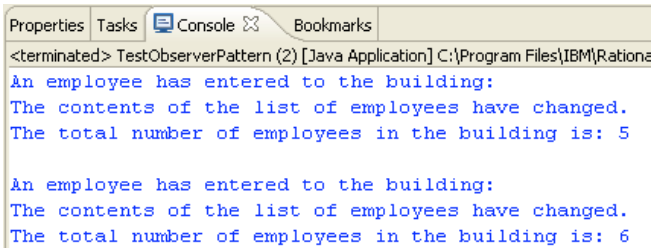
```
public static void main( String [] args ) {
    String employee1 = "Peter Anteater";
    String employee2 = "Bucky Badger";
    String employee3 = "Buster Bronco";
    String employee4 = "Donald Duck";
    String employee5 = "Herbie Husker";
    String employee6 = "Blizzard Husky";
    ListOfEmployees list = new ListOfEmployees();
    list.add( employee1 ); list.add( employee2 ); list.add( employee3 );
    list.add( employee4 );
    //counter and preview add themselves to the listOfPeople
    EmployeesCounter counter = new EmployeesCounter( list );
    EmployeesNamesPreview preview = new EmployeesNamesPreview ( list );

    System.out.println( "An employee has entered to the building:" );
    list.add( employee5 );
    System.out.println("");

    System.out.println( "An employee has entered to the building:");
    list.add( employee6 );
    System.out.println("");
}
```

Run the application using the Observer Pattern

g) Select the class TestObserverPattern and go to Run -> Run as -> Java Application



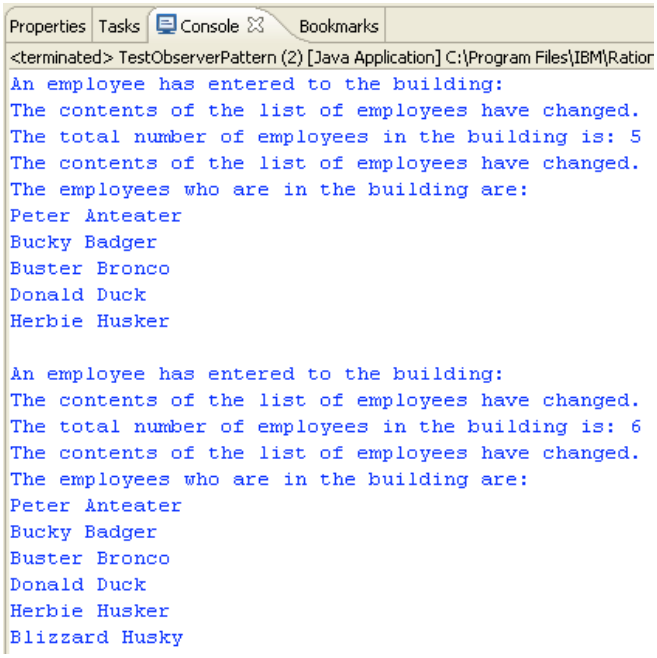
h) You should see the following output:



Task 8: Implement the update method for the Observer EmployeesNamesPreview

a) Implement the update method in the EmployeesNamesPreview Class, so that you will print in the screen the names of the people who are in the building each time one employee is added to the list. Hint: You should use the method update in EmployeesCounter.java as a reference.

b) After you finished with the implementation, run the program. You should have an output similar to the one showed in the following screen.



Take Home

1. (20 points) Reverse engineer the code of the following 12 classes you included in Homework 5 to create a class diagram using Rational Software Development Platform.

- **DVDVendor**
- **DVDDispenser**
- **DVDDispenser Event**
- **DVDDispenser Listener**
- **CheckOutCart**
- **Product**
- **DVD**
- **Transaction**
- **RentTransaction**
- **ReturnTransaction**
- **ProductDB**
- **An exception class of your choice**

You do not have to include their packages this time. You can include a specific class to the diagram by selecting the class -> Visualize -> Add to Current Diagram/Add to New Diagram File -> Class Diagram. Try to rearrange your diagram so that the classes are not overlapped and relationships are untangled.

i) Include the generated class diagram in your report.
ii) Identify 5 differences (beside from the package) between the diagram you created manually in Homework 5 and the one automatically generated by Rational Software Development Platform.

2. (40 points) Identify two design patterns used in DVDVendor. For each design pattern:
i) Create a class diagram containing classes and interfaces involved in the pattern.

- **Use a class diagram of each design pattern in the lecture slides as a template for arranging the classes and relationships.**
- **Use either Note or Stereotype to show the role of each class or template in the design pattern.**
- **For each classes and interface, you don't need to include all attributes and methods. However you need to include attributes and methods that are involved in the pattern.**

ii) Explain how the pattern in the DVDVendor software works. What system or user event causes them to be triggered?