MIS 768: Advanced Software Concepts

Spring 2024

Polymorphism

Purpose

- Practice the usage of polymorphic variables.
- Implement abstract method in a subclass.
- Use Interface in the programs.

1. Preparation

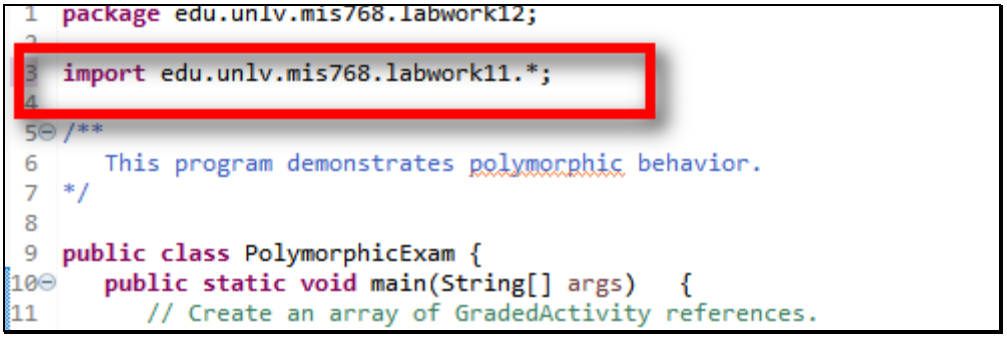
(1) Launch Eclipse. Create a new package to hold our source file. Name the package as **edu.unlv.mis768.labwork12.**

(2) Download **12_lab_files.zip** from WebCampus. Extract the zip file and then import the .java files into the package.

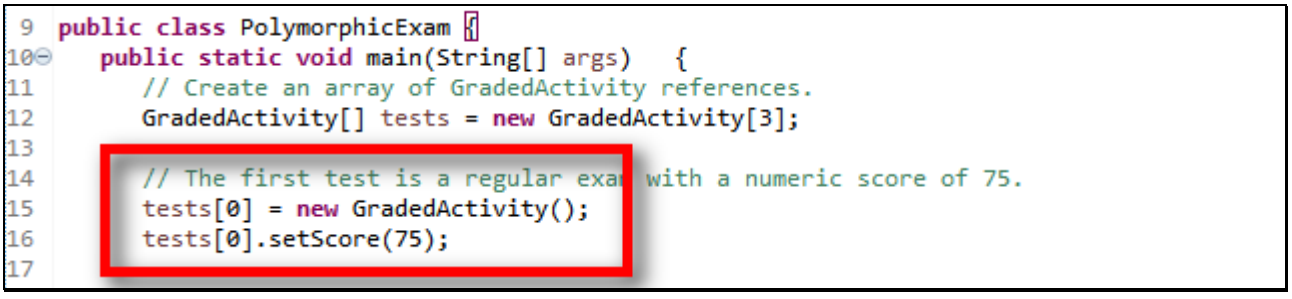
2. Polymorphic Variables: PolymorphismExam

(3) Open **PolymorphicExam.java**. It contains an array that represents three exams.

First, we need to import the classes we created in the last lab.



(4) Based on the exam type, please declare three objects and complete the program. The first is a regular exam. Then set the score to 75.



(5) The second one is a pass/fail exam. At instantiation, please also provide the number of questions, missed questions, and the minimum passing score.

```
18        // The second test is a pass/fail test.
19        // The student missed 5 out of 20 questions, and the minimum passing grade is 60.
20        tests[1] = new PassFailExam(20,5,60);
21
```

(6) The third one is an exam that will be curved by 1.25. The original score should also be provided.

```
22        // The third test is an curved exam. It will be curved by 1.25
23        // The original score is 62
24        tests[2] = new CurvedActivity(1.25);
25        tests[2].setScore(62);
26
27        // Display the grades.
28        for (int i = 0; i < tests.length; i++) {
29           System.out.println("Test " + (i + 1) + ": " +
30                       "score " + tests[i].getScore() +
31                       ", grade " + tests[i].getGrade());
32        }
```

(7) Although the three objects refer to a **GradedActivity**, a **PassFailExam**, and a **CurvedActivity** object respectively, they are all under the same inheritance hierarchy. Thus we can use a loop to traverse the array to print the score and grade.

```
25        tests[2].setScore(62);
26
27        // Display the grades.
28        for (int i = 0; i < tests.length; i++) {
29           System.out.println("Test " + (i + 1) + ": " +
30                       "score " + tests[i].getScore() +
31                       ", grade " + tests[i].getGrade());
32        }
```

(8) Please run and test the program.

3. Dynamic Binding: Payroll

(9) Please declare an ArrayList of **Staff**. We later can add **Staff** and **Manager** objects to this ArrayList.

```
16        int count = 0; // a counter for the number of employee
17
18        // declare an ArrayList to store the employee data
19        // the elements in this ArrayList can be either Staff or Manager
20        ArrayList<Staff> empList =new ArrayList<Staff>();
21
22        // print the purpose of the program
```

(10) If the data entered is for a manger, declare a Manager object and assign values for the fields. Then add this object to the ArrayList.

Please note that we need to declare the object as Manager, so that we can call the **setLevel**() method.

```
52              // for a manager
53              if (manager=='Y') {
54                  // get the level
55                  System.out.print("What level (1-5)?");
56                  level=kb.nextInt();
57
58                  // consume the additional newline char after nextInt()
59                  kb.nextLine();
60
61                  // instantiate a manager object
62                  Manager mgr = new Manager();
63                  mgr.setName(name); // assign the name, method fro the super class
64                  mgr.setPayRate(rate); // assign pay rate, method rom the super class
65                  mgr.setHours(hours); // assign working hours, met od from the super class
66                  mgr.setLevel(level); // assign the level
67
68                  // add the Manager object to the ArrayList
69                  empList.add(mgr);
70              }
```

(11) Similarly, instantiate a Staff object for someone who is not a manager. Set the values, and then add the object to the ArrayList.

```
71              // for a staff
72              else {
73                  // instantiate a staff object
74                  Staff sf = new Staff();
75                  sf.setName(name); // assign the name
76                  sf.setPayRate(rate); // assign the pay rate
77                  sf.setHours(hours); // assign the working hours
78
79                  // add the Staff object to the ArrayList
80                  empList.add(sf);
81              }
82
```

(12) Once all the employees are entered, we can use a loop to print all the data within the ArrayList.

At this step, the program will dynamically determine which **calcSalary**() method to call based on the type of the object.

```
91          // Traverse the ArrayList to print the salary for every employee
92          for(int i=0; i<empList.size(); i++) {
93              // empList.get(i) represents one employee
94              // user getName() to print name, use calcSalary() to show salary
95              System.out.println(empList.get(i).getName()+": $"+empList.get(i).calcSalary());
96          }
97
```

(13) Run and test the program now.

4. Abstract Class

(14) The **Student.java** program is an abstract class.

In this class, the **getRemainingHours()** method is an abstract method with only header and no body. This method needs to be overridden by subclasses.

(15) Open **BusinessStudent.java.** You can see error message showing there are unimplemented methods. Add the following method to the end of this class.

```
45    /**
46       The getRemainingHours method returns the
47       the number of hours remaining to be taken.
48       @return The hours remaining for the student.
49    */
50    @Override
51    public int getRemainingHours() {
52      return (BIZ_HOURS+GEN_ED_HOURS)-bizHours-genEdHours;
53    }
54
```

(16) Open **BusinessStudentDemo** and complete the program to test the **BusinessStudent** class.

Please note that you cannot instantiate **Student** object because it is an abstract class.

```
19        // Create a BusinessStudent object.
20        BusinessStudent std = new BusinessStudent(sName,sID,year);
21
22        // Get the student's business hours
23        System.out.print("Please enter the student's business hours fulfilled: ");
24        int hour = kb.nextInt();
25
26        // Set the business hours
27        std.setBizHours(hour);
28
29        // Get the student's General Ed hours
30        System.out.print("Please enter the student's general ed hours fulfilled: ");
31        hour = kb.nextInt();
32
33        // Set the gen ed hours
34        std.setGenEdHours(hour);
35
36        // Display the number of remaining hours.
37        System.out.println("Hours remaining: "+std.getRemainingHours());
```

5. Interface

(17) **Relatable.java** is an interface. It defines the methods that needs to be implemented.

```
1  package edu.unlv.mis768.labwork12;
2
3  /**
4     Relatable interface
5     It defines the methods needed for comparing Staff objects
6  */
7
8  public interface Relatable {
9     boolean equals(Staff s);
10    boolean isGreater(Staff s);
11    boolean isLess(Staff s);
12 }
```

(18) Open **Manager.java**. Add the **implements** keyword to the header of the class and specify that it implements **Relatable**.

```
1  package edu.unlv.mis768.labwork12;
2
3  public class Manager extends Staff implements Relatable {
4      // the additional field for Manager
5      private int level;
6
7      // constant to represent the stipend for manager
8      public final double STIPEND = 150;
9
```

(19) We now need to implement the three methods defined in the interface. Move toward the end of the class, add the following method **equals().** We will compare the objects by their salary.

```
36     @Override
37     public boolean equals(Staff s) {
38         if (this.calcSalary()== s.calcSalary())
39             return true;
40         else
41             return false;
42     }
```

(20) Then implement **isGreater**() method:

```
44     @Override
45     public boolean isGreater(Staff s) {
46         if (this.calcSalary() > s.calcSalary())
47             return true;
48         else
49             return false;
50     }
```

(21) Finally the **isless**() method should be implemented as well:

```
52     @Override
53     public boolean isLess(Staff s) {
54         if (this.calcSalary() < s.calcSalary())
55             return true;
56         else
57             return false;
58     }
```

(22) In **ManagerComparisonDemo**, we can use the newly implemented methods to compare the objects. Enter the following code:

```
22         // compare whether the two objects
23         if (mgr.isGreater(sf))
24             System.out.println("The manager gets higher pay.");
25         else
26             System.out.println("The manager does not get higher pay.");
27
```

6. Polymorphism with Interfaces

(23) In the example, **RetailItem.java** is an interface. Both **CompactDisc** and **StreamingMovie** Class implement this interface. **getRetailPrice**() method is defined in **RetailItem,** and therefore has to be implemented in **CompactDisc** and **StreamingMovie.**

(24) Please open **PolymorphicInterfaceDemo.java**. We can declare a variable of the Interface type

```
11    public static void main(String[] args)  {
12        // Declare a variable of a RetailItem  (interface) type
13        // It can later be used t reference objects that implements RetailItem interface
14        RetailItem item = null;
15
```

(25) The variable is used later to reference either a **CompactDisc** or **StreamingMovie** object.

```
18        if (type.equalsIgnoreCase("CD")){
19              // Get the title the title
20              String title = JOptionPane.showInputDialog("Please enter the title:");
21
22              // Get the artist
23              String artist = JOptionPane.showInputDialog("Please enter the artist:");
24
25              // Get the retail price
26              double price = Double.parseDouble(JOptionPane.showInputDialog("Please enter the retail price"));
27
28              // Create a CompactDisc object with the informatio entered by the user
29              item = new CompactDisc(title, artist, price);
30
31        } else if (type.equalsIgnoreCase("Movie")){
32              // Get the title the title
33              String title = JOptionPane.showInputDialog("Please enter the title:");
34
35              // Get the running time
36              int time = Integer.parseInt(JOptionPane.showInputDialog("Please enter the running time:"));
37
38              // Get the retail price
39              double price = Double.parseDouble(JOptionPane.showInputDialog("Please enter the retail price"));
40
41              // Create a DvdMovie object with the information ntered by the user
42              item = new StreamingMovie(title, time, price);
43        }
```

(26) In calling the **getRetailPrice()** method, we can just use the same variable, no matter it is a **CompactDisc** or **StreamingMovie** object. The program will dynamically determine which method to call.

```
44
45        // Display the product's price .
46        if (item!=null) {
47            JOptionPane.showMessageDialog(null, "The price is $"+item.getRetailPrice());
48        }
```