MIS 768: Advanced Software Concepts

Spring 2024

Inheritance

Purpose

- Define subclasses that extend a superclass
- Use inherited methods and constructors in subclasses
- Override methods of superclass

1. Preparation

(1) Launch Eclipse, and set the workspace to your personal directory.

(2) Create a **package** to hold our source file. Select the folder **src** from the package navigator. Right click on the folder, and then select **New \ Package** from the popup menu.

Name the package as **edu.unlv.mis768.labwork11.**

(3) Download **11_lab_files.zip** from WebCampus. Extract the zip file and then import the .java files into **edu.unlv.mis768.labwork11.**

2. Inheritance

(4) Open **GradedActivity.java**. There are one field and three methods in this class.

(5) Open **InClassExercise.java.** There are three fields, one constructor, and the getters and setters in this class.

We need to revise the **setCompleted**() method, so that the score will be determined after an exercise is marked as complete. We need to use the **setScore**() method of **GradedActivity** class to do so.

Please enter the following code:

```
41    /**
42     * When the exercise is completed, check whether it is overdue
43     * set the score accordingly
44     * @param c Completed or not
45     */
46    public void setCompleted(boolean c) {
47        Date today = new Date(); // get the system date
48
49        completed = c; // assign the given value to the field
50
51        // if the exercise is completed before due date
52        if(completed && today.compareTo(dueDate)<=0)
53            // set the score to 100
54            setScore(100);
55        // otherwise set the score as 0
56        else
57            setScore(0);
58    }
```

(6) Please open **InClassExerciseDemo.java.** Once we get the title and due date of an exercise, we can instantiate an **InClassExercise** object to represent the activity.

```
20          // prompt for title of exercise
21          System.out.println("Please enter the name of the exercise: ");
22          title = keyboard.nextLine();
23
24          // prompt for due date of exercise
25          System.out.println("Please enter the due date  (yyyy-mm-dd): ");
26          // use the parse() method of the SimpleDateFormat class to convert a string to date format
27          dueDate = sdf.parse(keyboard.nextLine());
28
29          // instantiate the object using title and due date
30          InClassExercise exercise = new InClassExercise(title, dueDate);
31
```

(7) Next, mark this exercise as complete or not, based on the user's input.

(When the exercise is not completed, we do not need to set it to false, because the value is initialized to false upon creation.)

```
32          // prompt for completion or not
33          System.out.println("Complete? (y/n)");
34          // get the answer, convert to all uppercase letters, and get the first character
35          completion = keyboard.nextLine().toUpperCase().charAt(0);
36
37          // if the answer is Y
38          if(completion=='Y')
39              exercise.setCompleted(true); // mark this exercise as completed
40
```

(8) Then we can print the due date and the score for this exercise:

```
41          // display the score for this exercise
42          System.out.println("The exercise is due on "+exercise.getDueDate());
43          System.out.println("The score for the exercise is: "+exercise.getScore());
44
```

(9) You can now test and run the program.

3. Constructor of superclass and subclass

(10) Please open **Rectangle.java**. There are two fields and several methods.

(11) Create a new class **Cube.java**.

This class extends **Rectangle** class with one additional field. Please enter the following lines of code.

```
/**
   This class holds data about a cube.
*/
public class Cube extends Rectangle {
   private double height;  // The cube's height


}
```

(12) Create the constructor **Cube()**. We'll use the superclass constructor here.

```
3  public class Cube extends Rectangle{
4
5      private double height;  // the cube's height
6
7      /**
8       * The constructor sets the length, width, and height of a cube
9       * @param l Length
10      * @param w Width
11      * @param h Height
12      */
13     public Cube(double l,double w,double h) {
14         // call the super class constructor to set the length and width
15         super(l,w);
16         //  set the height
17         height=h;
18     }
19
```

(13) Generate the **set** and **get** method for **height**.

(14) We'll add one more method **getVolume**(). We can use the **getArea**() method in the superclass to simplify the formula.

```
28     /**
29      * Calculates and returns the volume of a cube
30      * @return The volume of a cube
31      */
32     public double getVolume() {
33         return getArea()*height;
34     }
35
```

(15) In order to test the **Cube** class, please open **CubeDemo.java**.

Once the user enters the length, width and height, we can then create a **Cube** object and calculate the volume.

```
27
28          // Create a Cube object and pass the dimensions to the constructor
29          Cube c = new Cube(length, width, height);
30
31          // Display the cube's properties
32          System.out.println("Length: "+c.getLength());
33          System.out.println("Width: "+c.getWidth());
34          System.out.println("Height:"+c.getHeight());
35          System.out.println("Volume: "+c.getVolume());
36      }
37
```

4. Overriding Superclass Methods

(16) Open **CurvedActivity.java.** We want to override the **setScore**() method so that it curves the score by a certain percentage.

Use the **keyword** super, to refer to the method in the superclass.

```
29⊖   public void setScore(double s) {
30        // the given score will be stored as the raw score
31       rawScore = s;
32       // call the setScore() method of the superclass to set the new score (after curving)
33       super.setScore(rawScore * percentage);
34    }
35
```

(17) Open **CurvedActivityDemo.java.**

After getting the raw score and the curve percentage, create a **CurveActivity** object.

Use the object to set the score.

```
20
21            // Create a CurvedActivity object
22            CurvedActivity act = new CurvedActivity(curvePercent);
23
24            // Set the exam score
25            act.setScore(score);
26
```

(18) Then we can show the score. Note that the getRawScore() method needs to be used to get the original score.

```
26
27            // Display the raw score
28            System.out.println("The raw score is "+act.getRawScore());
29
30            // Display the curved score
31            System.out.println("The curved socre is "+act.getScore());
32
33            // Display the exam grade
34            System.out.println("The curved grade is "+act.getGrade());
35
```

5. Protected Members

(19) Open **GradedActivityProtected.java**. The **score** field is now declared as **protected.** Any class that inherits from this class has direct access to it.

(20) Open **CurvedActivityProtected.java**. Now under the **setScore**() method, we can directly update the score field, without calling the **setScore**() method in the superclass.

```
28
29⊖   public void setScore(double s) {
30        // the given score will be stored as the raw score
31       rawScore = s;
32       // update the score as the new score (after curving)
33       score = rawScore * percentage;
34    }
35
```

6. Chains of Inheritance

(21) The **PassFailActivity** class extends the **GradeActivity** class.

Based on a minimum passing score, this class's **getGrade**() method would determine the grade is P or F. First, let's define the new field representing the minimum passing score:

```
8  public class PassFailActivity extends GradedActivity {
9     // Specialized field Minimum passing score
10    private double minPassingScore;
```

(22) Also, define the constructor of this class that requires a value for **minPassignScore**.

```
12    /**
13       The constructor sets the minimum passing score.
14       @param mps The minimum passing score.
15    */
16    public PassFailActivity (double mps) {
17        minPassingScore = mps;
18    }
```

(23) We also need to override the **getGrade**() method to return P of F, rather than the regular letter grade.

```
20    /**
21       The getGrade method returns a letter grade determined
22       from the score field. This method overrides the
23       superclass method.
24       @return The letter grade.
25    */
26    public char getGrade(){
27        // initialize the grade to P
28        char grade='P';
29
30        // when the score is less than passing score
31        if(getScore()<minPassingScore)
32            // set the grade to F
33               grade = 'F';
34
35        return grade;
36    }
```

(24) The **PassFailExam** class then extends **PassFailActivity.**

It has fields for the number of questions on the exam (numQuestions), the number of points each and the number of questions missed by the student (numMissed).

(25) The **PassFailExamDemo** program is used to demonstrate the **PassFailExam** class.

Please complete the program as following:

```
29        // Create a PassFailExam object.
30        PassFailExam exam = new PassFailExam(questions, missed, minPassing);
31
32        // Display the exam score.
33        System.out.println("The exam score is: "+exam.getScore());
34
35        // Display the exam grade.
36        System.out.println("The grade is: "+exam.getGrade());
37    }
```