MIS 768: Advanced Software Concepts

Spring 2024

Classes (2)

Purpose

- More exercise on creating model classes from class diagram
- Use objects in applications
- Pass and return objects in methods

1. Preparation

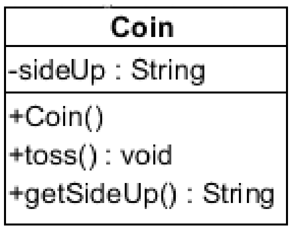
(1) Launch Eclipse, and set the workspace to your personal directory.

(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.labwork9.**

(3) Download **09_lab_files.zip** from WebCampus. Extract the zip file and then import the .java files into **edu.unlv.mis768.labwork9.**

2. Creating the Coin class

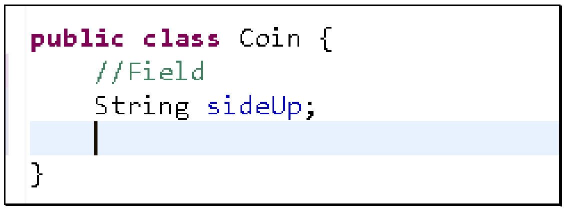
(4) The **Coin** class simulates a coin. It has the following field and attributes:



- **sideUp** will hold either “heads” or “tails” indicating the side of the coin that is facing up
- A no-arg constructor that randomly determines the side of the coin that is facing up and initialized the **sideUp** field accordingly
- When the **toss()** method is called, it randomly determines the side of the coin that is facing up, and sets the **sideUp** field accordingly.
- **getSideUp()** method returns the value of the **sideUp** field.

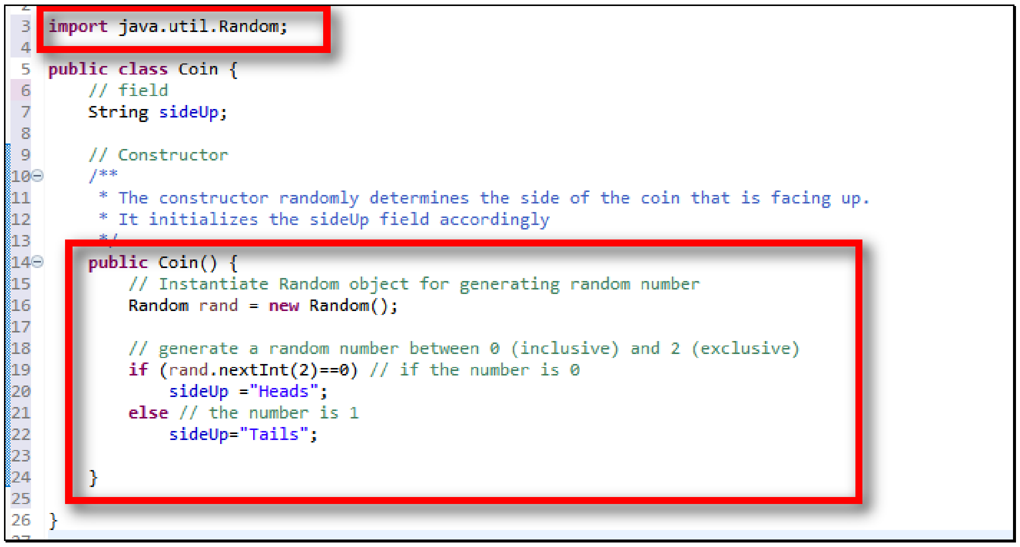
(5) Create a new class and name it as **Coin**.

Enter the following statement for the field for the **`Coin`** class:



```
public class Coin {
    //Field
    String sideUp;

}
```

(6) Create the no-arg constructor:

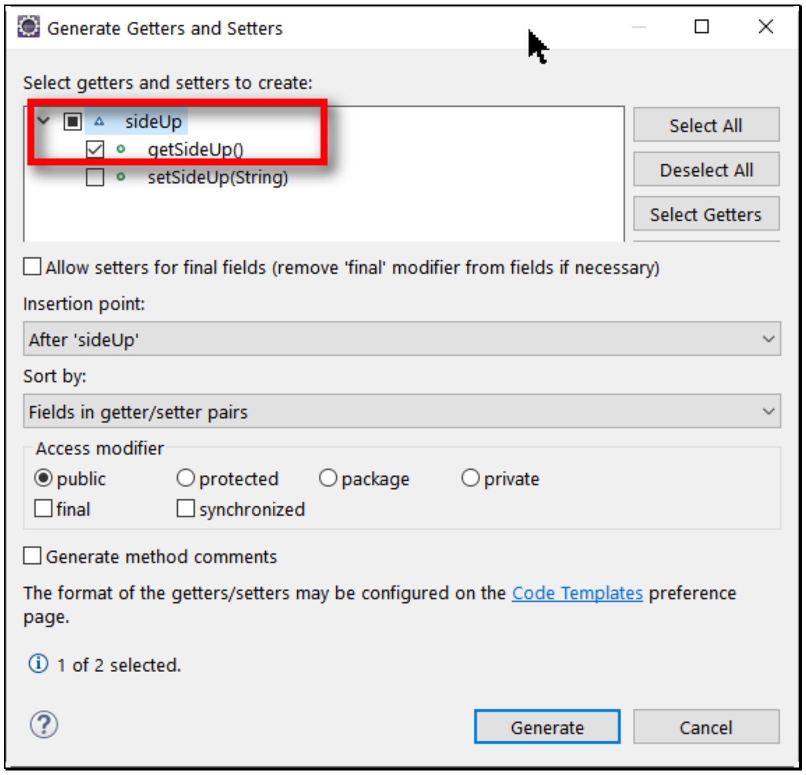


```
import java.util.Random;

public class Coin {
    // field
    String sideUp;

    // Constructor
    /**
     * The constructor randomly determines the side of the coin that is facing up.
     * It initializes the sideUp field accordingly
     */
    public Coin() {
        // Instantiate Random object for generating random number
        Random rand = new Random();

        // generate a random number between 0 (inclusive) and 2 (exclusive)
        if (rand.nextInt(2)==0) // if the number is 0
            sideUp ="Heads";
        else // the number is 1
            sideUp="Tails";

    }

}
```

(7) Now, please create the **toss**() method on your own.

(8) Select **Source \ Generate Getter and Setters** from the menu.

Select **getSideUp**() method in the popup dialogue, and click the **OK** button to generate the get method.



Generate Getters and Setters

Select getters and setters to create:

sideUp
[✓] getSideUp()
[ ] setSideUp(String)

[ ] Allow setters for final fields (remove 'final' modifier from fields if necessary)

Insertion point: After 'sideUp'

Sort by: Fields in getter/setter pairs

Access modifier
[✓ public] protected package private
[ ] final [ ] synchronized

[ ] Generate method comments

The format of the getters/setters may be configured on the Code Templates preference page.

1 of 2 selected.

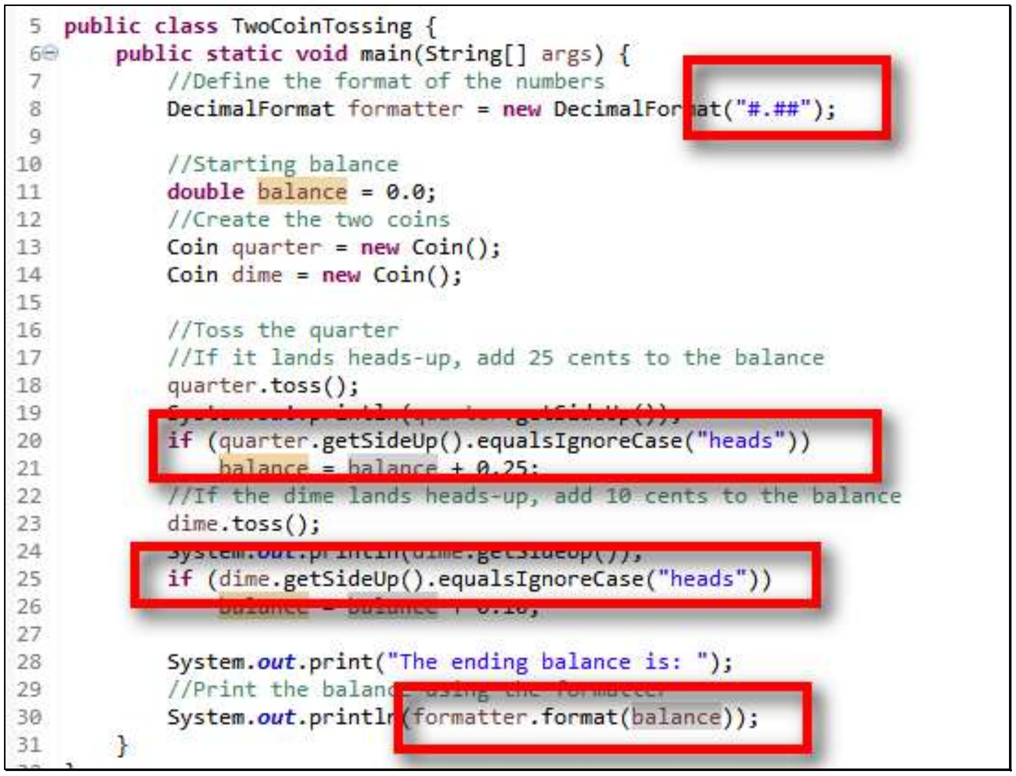
3. **Test the Class: TwoCoinTossing.java**

   (9) Create two instances of the **Coin** class: one representing a quarter and the other representing a dime. When the coins are tossed, the value of the coin is added. (For example, 25 cents are added to the balance if the quarter lands heads-up.)

   Nothing is added to the balance for coins that land tails-up.

   Displays the balance. The format of the balance should be something like **$1.80**

   (10) Please complete the following program to see how the objects work:



```java
public class TwoCoinTossing {
    public static void main(String[] args) {
        //Define the format of the numbers
        DecimalFormat formatter = new DecimalFormat("#.##");

        //Starting balance
        double balance = 0.0;
        //Create the two coins
        Coin quarter = new Coin();
        Coin dime = new Coin();

        //Toss the quarter
        //If it lands heads-up, add 25 cents to the balance
        quarter.toss();
        System.out.println(quarter.getSideUp());
        if (quarter.getSideUp().equalsIgnoreCase("heads"))
            balance = balance + 0.25;
        //If the dime lands heads-up, add 10 cents to the balance
        dime.toss();
        System.out.println(dime.getSideUp());
        if (dime.getSideUp().equalsIgnoreCase("heads"))
            balance = balance + 0.10;

        System.out.print("The ending balance is: ");
        //Print the balance using the formatter
        System.out.println(formatter.format(balance));
    }
```

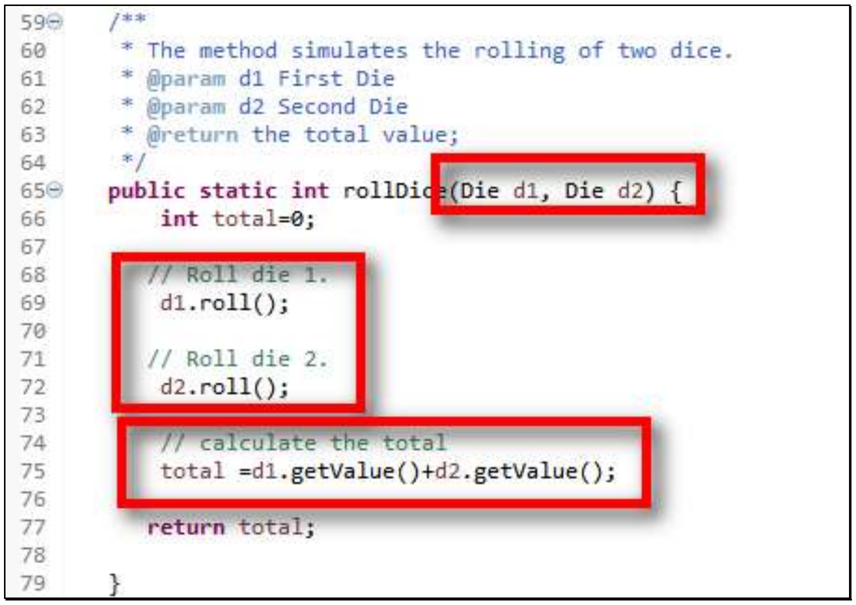
4. **Passing Objects as Arguments: Die and DiceGame**

   (11) Import the **Die** class. This class simulates a die with any number of sides. It also contains the **roll()** method that will determine the value by a random number.

   In the **DiceGame** program, we will allow the user to choose what type of dice (i.e., how many sides). The user will then roll two dice.

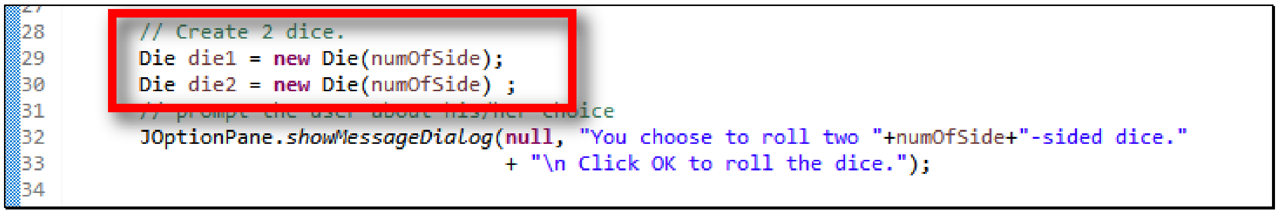
   The computer would then roll the same dice as well. The user wins if the total value of the dice is higher than that of the computer.

(12) In **DiceGame**, please create the **rollDice()** method that has two parameters representing two **Die** objects. This method should roll both dice, and the get the value, and return the total of the values.



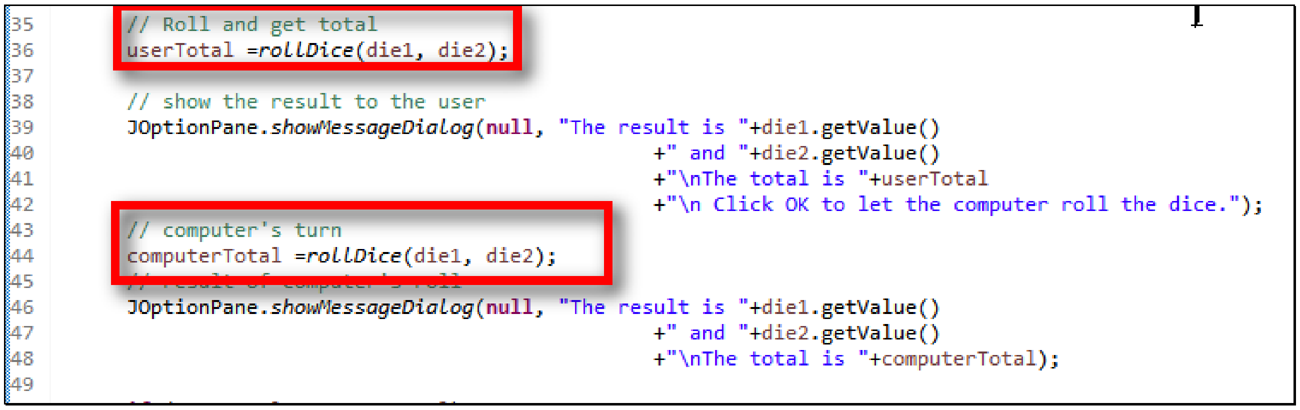
```
    /**
     * The method simulates the rolling of two dice.
     * @param d1 First Die
     * @param d2 Second Die
     * @return the total value;
     */
    public static int rollDice(Die d1, Die d2) {
        int total=0;

        // Roll die 1.
        d1.roll();

        // Roll die 2.
        d2.roll();

        // calculate the total
        total =d1.getValue()+d2.getValue();

        return total;

    }
```

(13) Now in the main method, instantiate two dice.



```
        // Create 2 dice.
        Die die1 = new Die(numOfSide);
        Die die2 = new Die(numOfSide) ;
        // prompt the user about his/her choice
        JOptionPane.showMessageDialog(null, "You choose to roll two "+numOfSide+"-sided dice."
                                      + "\n Click OK to roll the dice.");
```

(14) Call the **rollDice()** method to get the user total and computer total.



```
        // Roll and get total
        userTotal =rollDice(die1, die2);

        // show the result to the user
        JOptionPane.showMessageDialog(null, "The result is "+die1.getValue()
                                            +" and "+die2.getValue()
                                            +"\nThe total is "+userTotal
                                            +"\n Click OK to let the computer roll the dice.");

        // computer's turn
        computerTotal =rollDice(die1, die2);
        // result of computer's roll
        JOptionPane.showMessageDialog(null, "The result is "+die1.getValue()
                                            +" and "+die2.getValue()
                                            +"\nThe total is "+computerTotal);
```

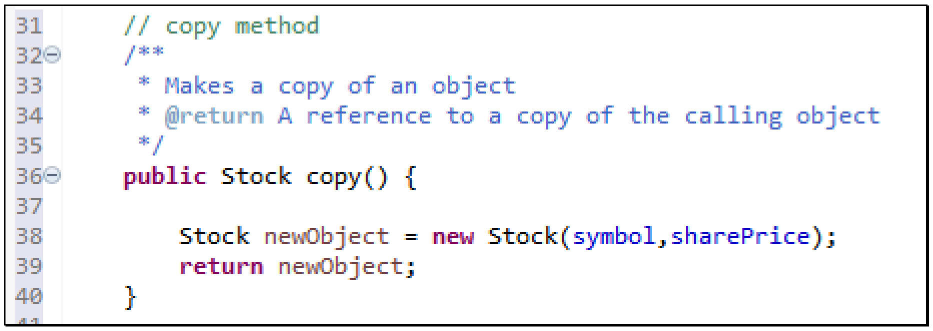
5. Copy Object

(15) The **Stock** class. It simulates a stock on the market and includes the constructor, getters and setters.

(16) The **CopyDemo** class attempts to make a copy of a Stock class. Please run the program and see the default operation when you use = to assign one object to a variable reference.

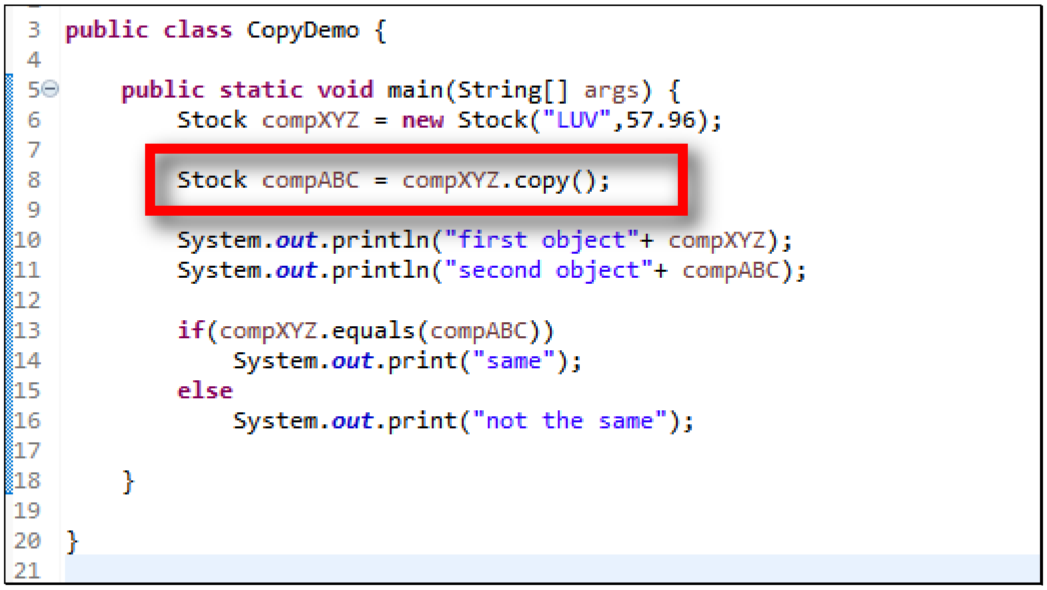
(17) To allow making a copy of the object, we need to add more methods to this class.

Move to the end of the **Stock** class, and add the **copy()** method.



```
    // copy method
    /**
     * Makes a copy of an object
     * @return A reference to a copy of the calling object
     */
    public Stock copy() {

        Stock newObject = new Stock(symbol,sharePrice);
        return newObject;
    }
```

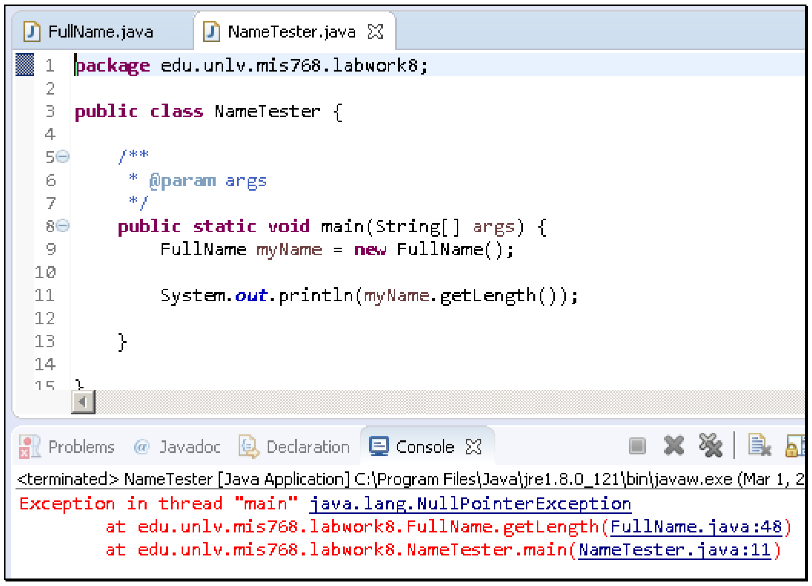
(18) Now, we can revise the **ObjectCopy.java** program to make a "deep" copy of the object.



```
public class CopyDemo {

    public static void main(String[] args) {
        Stock compXYZ = new Stock("LUV",57.96);

        Stock compABC = compXYZ.copy();

        System.out.println("first object"+ compXYZ);
        System.out.println("second object"+ compABC);

        if(compXYZ.equals(compABC))
            System.out.print("same");
        else
            System.out.print("not the same");

    }

}
```

6. Null Reference

(19) Open **NameTester.java**. When you run it, you'll find the following **NullPointerException**:



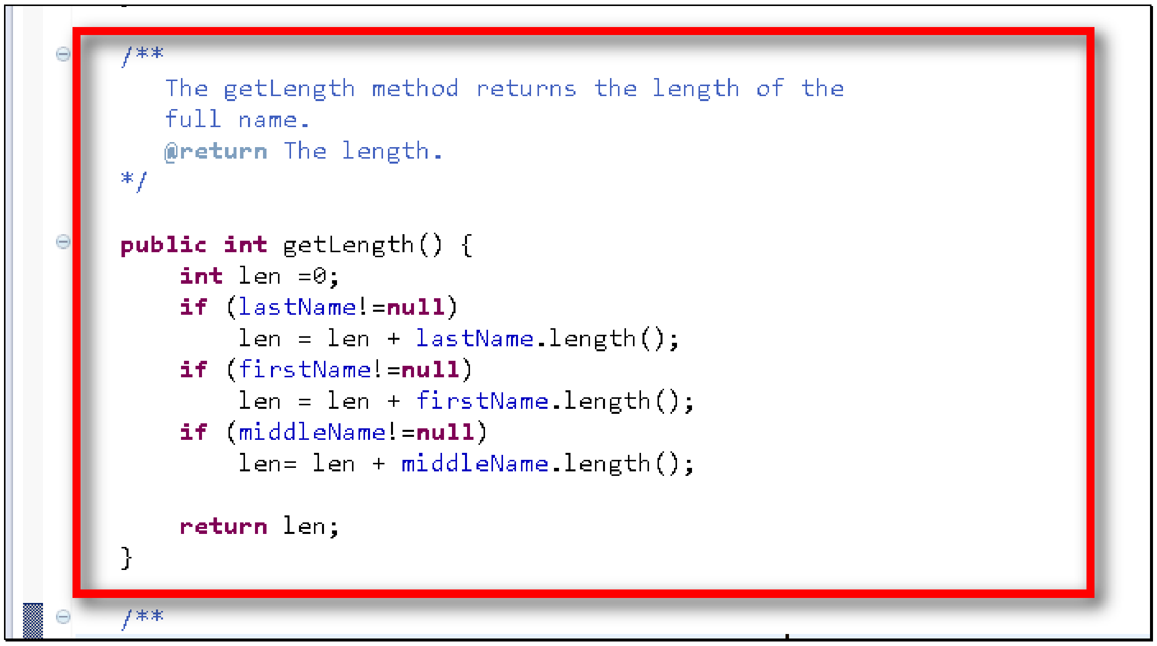
```
package edu.unlv.mis768.labwork8;

public class NameTester {

    /**
     * @param args
     */
    public static void main(String[] args) {
        FullName myName = new FullName();

        System.out.println(myName.getLength());

    }

}
```

```
<terminated> NameTester [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (Mar 1, 2
Exception in thread "main" java.lang.NullPointerException
	at edu.unlv.mis768.labwork8.FullName.getLength(FullName.java:48)
	at edu.unlv.mis768.labwork8.NameTester.main(NameTester.java:11)
```

(20) In order to fix/prevent this problem, we need to revise the code.

Move to **FullName.java**, and find the **getLength()** method.

Revise the method as following:



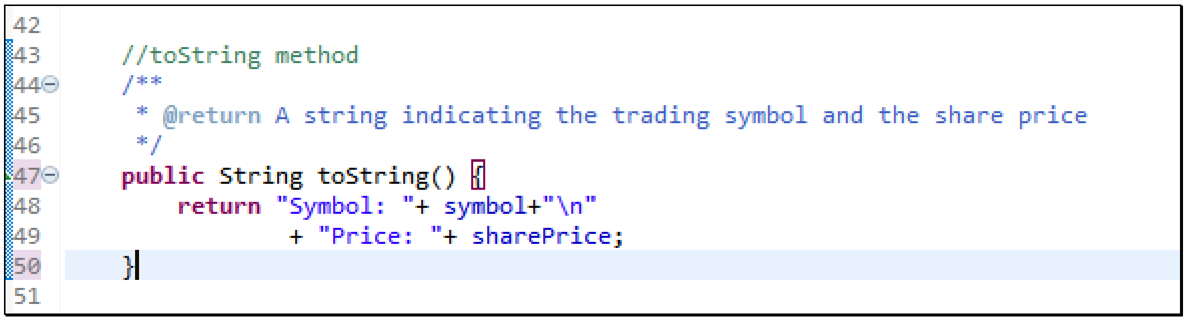
```
/**
   The getLength method returns the length of the
   full name.
   @return The length.
*/

public int getLength() {
    int len =0;
    if (lastName!=null)
        len = len + lastName.length();
    if (firstName!=null)
        len = len + firstName.length();
    if (middleName!=null)
        len= len + middleName.length();

    return len;
}

/**
```

7. toString Method: Stock.java and StockDemo.java

(21) Previously in the **CopyDemo.java** program, we attempt to print the object but only see it prints the class name and a hash of the memory address of the object.

(22) We can override the default **toString()** method to print out more useful information.

Add the following method to the **Stock** class.

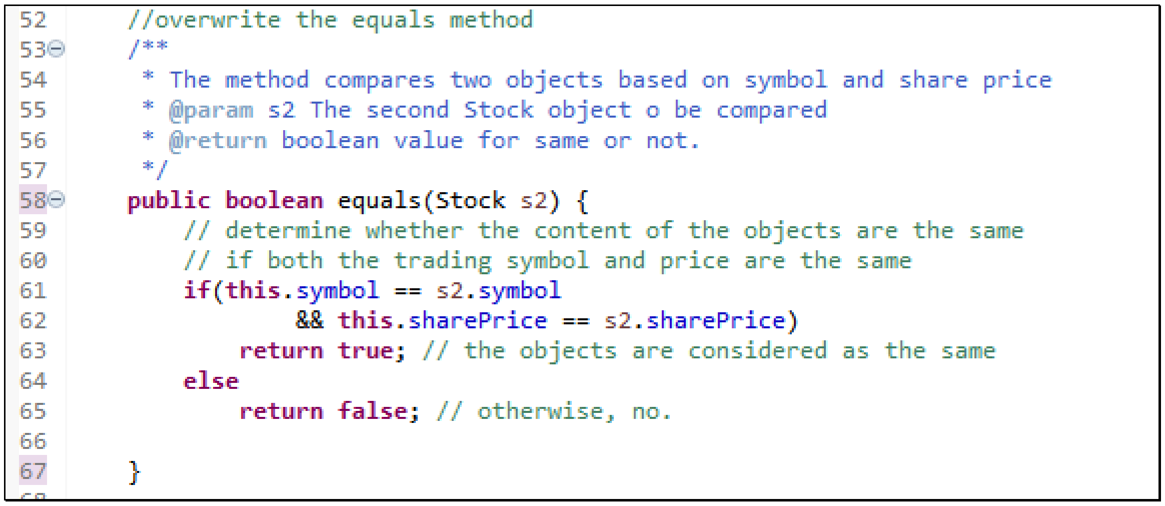


```

    //toString method
    /**
     * @return A string indicating the trading symbol and the share price
     */
    public String toString() {
        return "Symbol: "+ symbol+"\n"
                + "Price: "+ sharePrice;
    }

```

(23) Run **CopyDemo.java** again to see how the new **toString**() method works.

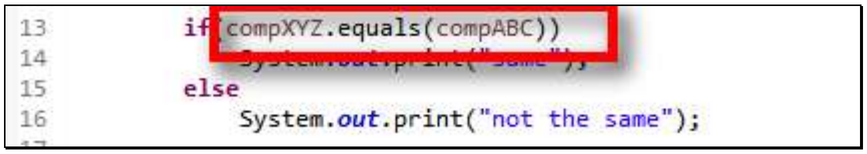
8. equals Method

(24) Add the following method to the **Stock** class.



```
    //overwrite the equals method
    /**
     * The method compares two objects based on symbol and share price
     * @param s2 The second Stock object o be compared
     * @return boolean value for same or not.
     */
    public boolean equals(Stock s2) {
        // determine whether the content of the objects are the same
        // if both the trading symbol and price are the same
        if(this.symbol == s2.symbol
                && this.sharePrice == s2.sharePrice)
            return true; // the objects are considered as the same
        else
            return false; // otherwise, no.

    }
```

(25) Open **CopyDemo.java**. You would notice that the program uses the **equals**() method to compare the objects:

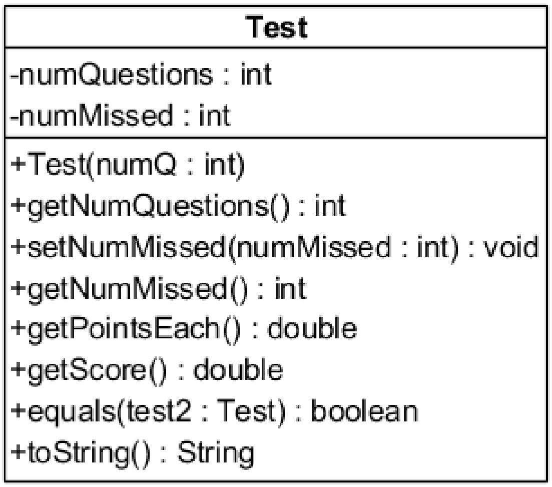


```
        if(compXYZ.equals(compABC))
            System.out.print("same");
        else
            System.out.print("not the same");
```

(26) You can run **CopyDemo.java** to test the **equals()** method.

Exercise

(27) Based on the following class diagram, please implement the class **Test**:



Test
-numQuestions : int -numMissed : int
+Test(numQ : int) +getNumQuestions() : int +setNumMissed(numMissed : int) : void +getNumMissed() : int +getPointsEach() : double +getScore() : double +equals(test2 : Test) : boolean +toString() : String

- Two fields representing the number of questions in this test, and the number of questions missed by a specific test-taker.
- The parameterized constructor accepts an argument representing the number of questions. That is, when a **Test** object is instantiated, the number of questions needs to be specified.
  However, if the value passed to this constructor is not a positive number, make **numQuestion** as 0.
- The get method of **numQuestions** is provided, but not the set method.
- The get and set methods of **numMissed** is provided.

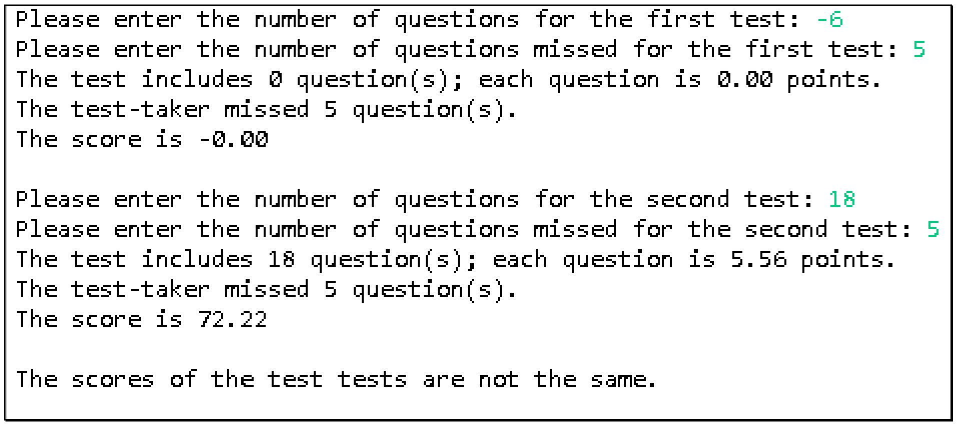
In the **setNumMissed**() method, if the value passed to this method is not a positive number, make **numMissed** as 0.

- The **getPointsEach**() method will determine how many points for each question. Assume the test is 100 points and each question accounts for the same points. Thus, this method will calculate the points for each question, and return a double number.
  However, when **numQuestion** is not a positive number, make the points for each question as 0.
- The **getScore**() method should calculate the final score for this test. This method thus should return a double number. For example, if there are 40 questions in the test and the test-taker missed 3 questions, the score should be 92.5.
- The **equals**() method can be used to compare two Test objects, by the score. It should return true if the scores of the two **Test** objects are the same, and return false if the scores of the two **Test** objects are not the same.
- The **toString**() method can be used to show the content of an object. It should return a String indicating the number of questions, points for each question, number of questions missed, and the score. For example, with 18 questions in the Test and missed 2 questions, the **toString**() method should return the following:

```
The test includes 18 question(s); each question is 5.56 points.
The test-taker missed 2 question(s).
The score is 88.89
```

(28) Create a program to demonstrate this **Test** class.

In this program, ask the user to enter the number of questions for the first test, and also enter the number of questions missed in the first test. After showing the content of object representing the first test, the program asks the user to enter the same information for the second test. The program then shows the content of the second **Test** object. Finally, the program should indicate whether the scores of the two tests are the same.



```
Please enter the number of questions for the first test: -6
Please enter the number of questions missed for the first test: 5
The test includes 0 question(s); each question is 0.00 points.
The test-taker missed 5 question(s).
The score is -0.00

Please enter the number of questions for the second test: 18
Please enter the number of questions missed for the second test: 5
The test includes 18 question(s); each question is 5.56 points.
The test-taker missed 5 question(s).
The score is 72.22

The scores of the test tests are not the same.
```