MIS 768: Advanced Software Concepts

Spring 2024

Classes (1)

1. **Purpose**
   - Create model classes from class diagram
   - Use objects (of model classes) in the program
   - Practice using object-oriented approach of programming
2. **Preparation**

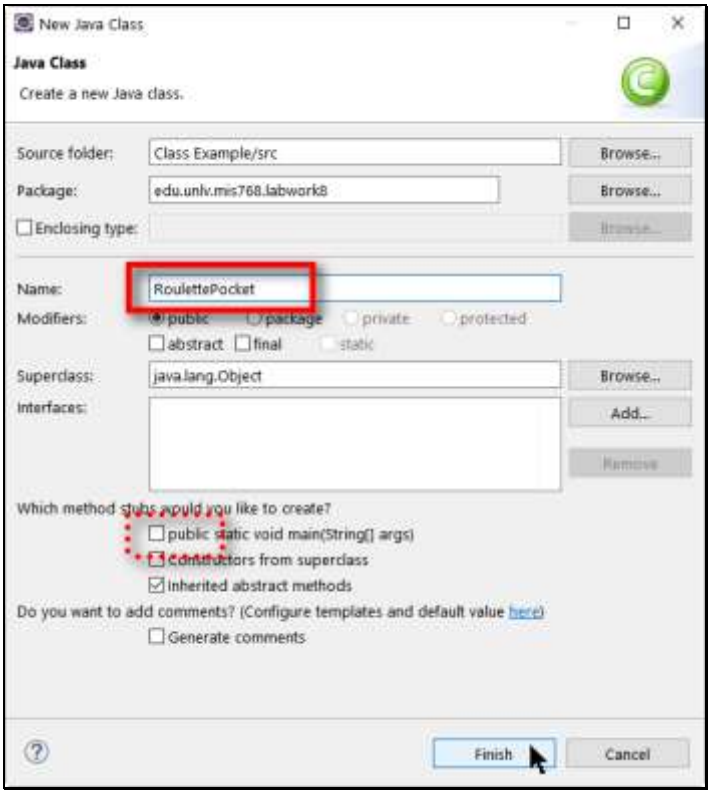
   (1) Launch Eclipse, and set the workspace to your personal directory.

   (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.labwork8.**

   (3) Download **08_lab_files.zip** from WebCampus. Extract the zip file and then import the .java files into **edu.unlv.mis768.labwork8.**
3. **Creating the RoulettePocket Class**

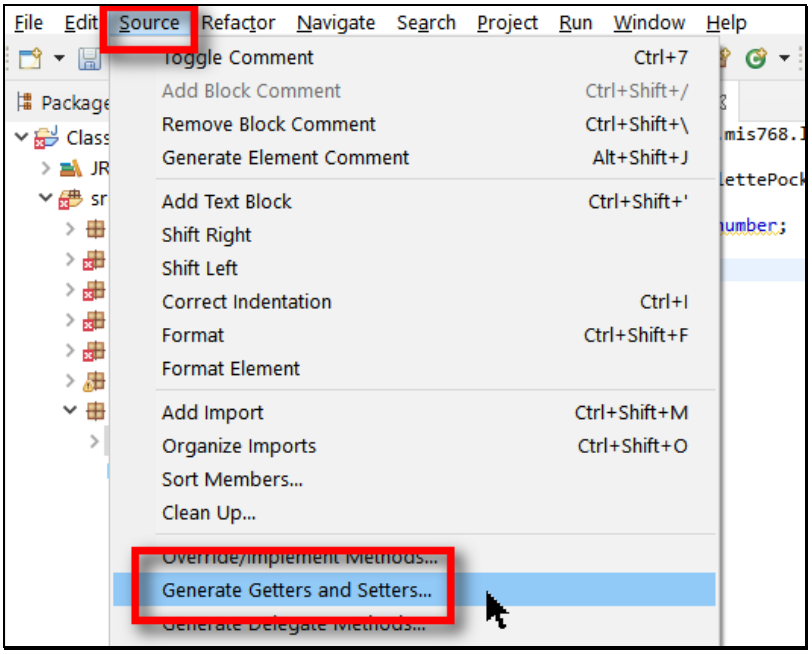
   (4) Create a new class, and name it as **RoulettePocket**. This is NOT an application class, and please do not check the main method option.



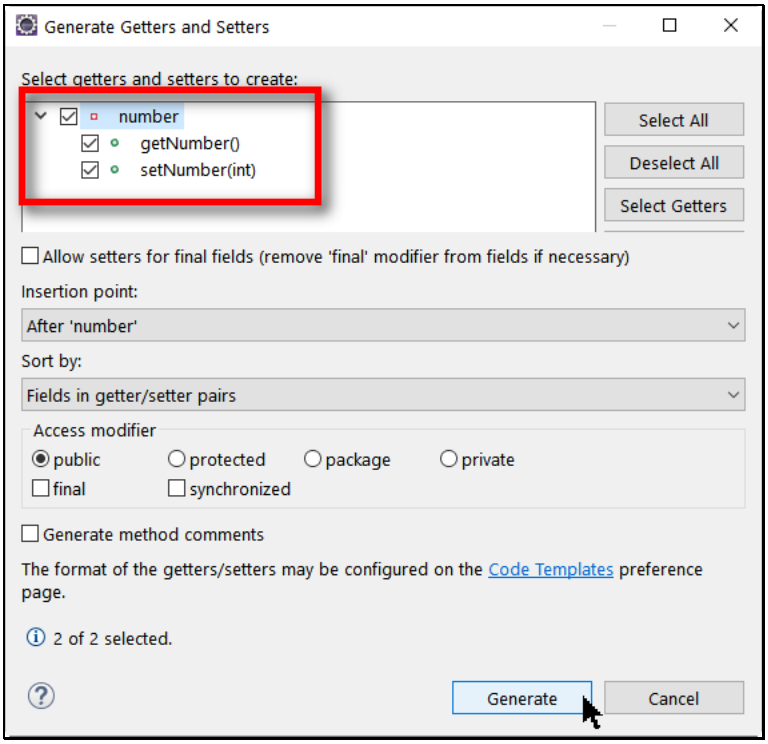
(5) Enter the following statements for the attribute/field:

```
2
3  public class RoulettePocket {
4      // field
5      private int number;
6  }
7
```

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



(7) Select the fields and click the **OK** button.



(8) By now you should have the following code:

```
public class RoulettePocket {
    // field
    private int number;

    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }


}
```

(9) Now we can implement the **getColor()** method to determine the color based on number.

- Pocket 0 is green.
- 1 ~10, the odd numbered pockets are red and the even numbered pockets are black.
- 11 ~ 18, the odd numbered pockets are black and the even numbered pockets are red.
- 19 ~ 28, the odd numbered pockets are red and the even numbered pockets are black.
- 29 ~ 36, the odd numbered pockets are black and the even numbered pockets are red.

```
    /**
     * The method determined the pocket color based on the number
     * @return Color of the pocket
     */
    public String getColor() {
        // declare variable to represent the color
        String color="green"; // make the color default to green

        // 1 ~10, or 19-28
        // the odd numbered pockets are red
        // the even numbered pockets are black.
        if ((number >=1 && number <=10)
             || (number >=19 &&  number <=28)) {
            if(number % 2 == 0) // even number
                color = "black";
            else // odd number
                color = "red";
        }
        // 11 ~ 18, or 29~36
        // the odd numbered pockets are black
        // the even numbered pockets are red.
        else if ((number >=11 && number <=18)
         || (number >=29 && number <=36)){
            if(number % 2 == 0) // even number
                color = "red";
            else // odd number
                color = "black";
        }

        return color;
    }
```

(10) Now we can test the class. Please open **RouletteTicketDemo**.

Lines 9-11 are used to generate a random number between 0 and 36.

```
2
3  import java.util.Random;
4
5  public class RouletteTicketDemo {
6
7      public static void main(String[] args) {
8          // Create a Random object to generate random numbers.
9          Random rand = new Random();
10         // generate a random number between 0 (inclusive) and 37 (exclusive)
11         int num = rand.nextInt(37);
12
13
14
15
16     }
17
18 }
19
```

(11) Please complete the code to create an object named **pocket**.

Set the number as the generated random number.

Print out the number and the color for the pocket.

```
5  public class RouletteTicketDemo {
6      public static void main(String [] args)   {
7          // Create a Random object for generating random numbers
8          Random rand = new Random();
9          // generate a random number between 0 (inclusive) and 37(exclusive)
10         int num = rand.nextInt(37);
11
12         // instantiate a Roulette Pocket object
13         RouletteTicket pocket = new RouletteTicket();
14         // set the number
15         pocket.setNumber(num);
16         // print out the number and color for testing
17         System.out.println(pocket.getNumber()+"  "+pocket.getColor());
18
19     }
20 }
21
```

(12) Run the program. If the result is correct, revise the code to create 10 pockets (with 10 random numbers) to test the program.

4. Constructor

(13) Now move back to the **RoulettePocket** class. Enter the following statements:

In the second constructor, we also added a decision structure to make sure the number will not be negative.

```
3  public class RoulettePocket {
4      // fields
5      private int number;
6
7      // constructors
8      public RoulettePocket() {
9      }
10
11     public RoulettePocket(int num) {
12         if (num<0)
13             number = 0;
14         else
15             number = num;
16     }
17
18     // getter and setter
19     public int getNumber() {
20         return number;
21     }
22
```

(14) It is a good practice to add validation in the constructor and set methods to ensure legal values are assigned. Please revise the **setNumber**() method to validate the value as well.

```
23     public void setNumber(int num) {
24         if (num <0)
25             number = 0;
26         else
27             number = num;
28     }
```

(15) Please revise **RoulettePocketDemo**.java and use the constructor to assign a value to number while instantiate an object.

```
// instantiate a Roulette Pocket object
RoulettePocket pocket = new RoulettePocket(num);
```

5. Revise the No-Arg Constructor

(16) Please revise the no-arg constructor. When this constructor is called, it will generate a random number and assign it to number.

```
3  import java.util.Random;
4
5  public class RoulettePocket {
6      // field
7      private int number;
8
9      // constructors
10     public RoulettePocket(){
11         // Random object to generate random numbers.
12         Random rand = new Random();
13         // generate a random number between 0 (inclusive) and 37 (exclusive)
14         this. number = rand.nextInt(37);
15     }
16
17     public RoulettePocket(int num){
18         this.number = num;
19     }
20
```

(17) Please revise **RoulettePocketDemo**.java once again to use this new constructor.

6. Overloaded methods

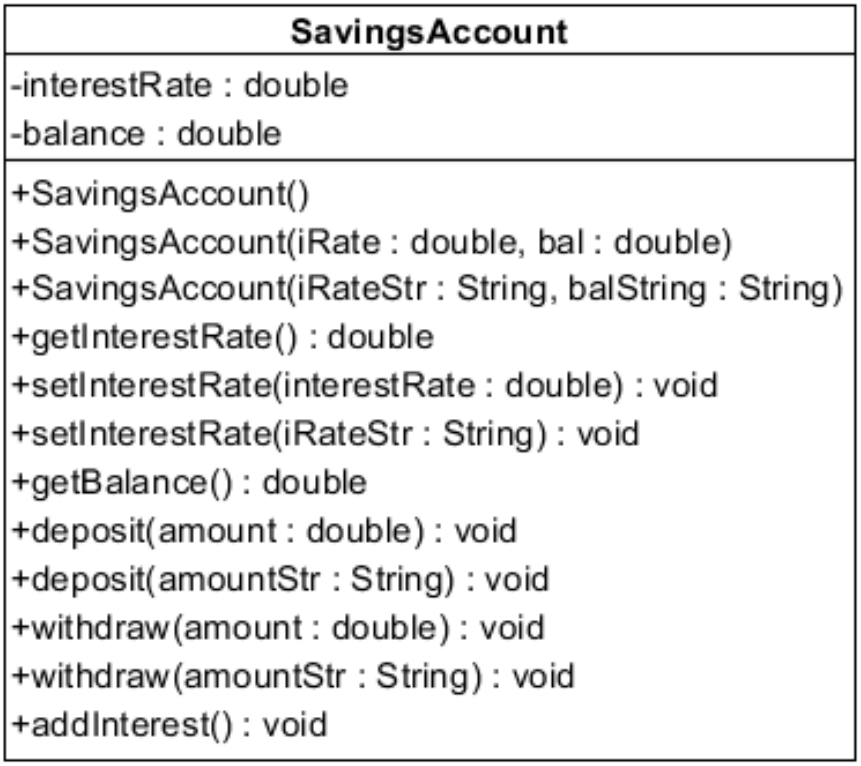
(18) Please create a new method **setNumber**() that accepts a string argument.

```
30     public void setNumber(String numStr) {
31         int num = Integer.parseInt(numStr);
32         if (num<0)
33             number = 0;
34         else
35             number = num;
36     }
37
```

(19) Please complete the program **RoulettePocketDemo2**.java and use a string to call the new **setNumber**() method.

7. A more complete example: SavingsAccount

(20) Based on the class diagram below, we can implement the **SavingsAcount** class.



The class has two fields:

- monthly interest rate: a double number
- balance: a double number

The class should have the following methods:

- No-arg constructor: Set the interest rate and balance to 0.
- Constructor: Accepts the double numbers interest rate and the amount of starting balance.
  If the input starting balance is less than zero, set the field **balance** to zero (0).
  If the input interest rate is less than 0, set the field **interestRate** to zero (0).
  If the input interest rate is greater than 0.01 (i.e., 1%), divide the number by 100 before setting the value to the field **interestRate**.
- Constructor: Accepts two strings as the interest rate and the starting balance. (Note: Convert the String to numbers before setting the values to the fields.
- Get and set method for the interest rate, including the overloaded method with Strings as the parameters.
  If the input interest rate is less than 0, set the field **interestRate** to zero (0).
  If the input interest rate is greater than 0.01 (i.e., 1%), divide the number by 100 before setting the value to the field **interestRate**.

- **deposit** method: Add the amount of a deposit to the balance, including the overloaded method with String as the parameter.

  If the input amount is less than 0, set the amount to zero(0).
- **withdraw** method: Subtract the amount of a withdrawal from the balance, including the overloaded method with String as the parameter.

  If the input amount is less than 0, set the amount to zero(0).
- **addInterest** method: Adding the amount of monthly interest to the balance. To add the monthly interest to the balance, multiply the monthly interest rate by the balance, and add the result to the balance.

(21) Let's create the class and define the fields:

```
3  public class SavingsAccount {
4      // field
5      private double interestRate;
6      private double balance;
7
8  }
9
```

(22) Generate the constructors. The constructor with arguments should also check for the legal values.

```
8      // constructors
9      public SavingsAccount() {
10
11     }
12
13     public SavingsAccount(double iRate, double bal) {
14         // If the input interest rate is less than 0, set the field balance to zero (0).
15         if (bal<0)
16             bal=0;
17         balance = bal;
18
19         // If the input starting balance is less than zero, set the field balance to zero (0).
20         if(iRate <0)
21             iRate = 0;
22         // If the input interest rate is greater than 0.01 (i.e., 1%), divide the number by 100
23         if(iRate <0.01)
24             iRate /= 100;
25
26         interestRate = iRate;
27
28     }
```

(23) Generate the getters and setters. You can write the code on your own or use Eclipse function to generate the code. Revise the set methods to include the validation of values.

```
30    public double getInterestRate() {
31        return interestRate;
32    }
33
34    public void setInterestRate(double iRate) {
35        // If the input starting balance is less than zero, set the field balance to zero (0).
36        if(iRate <0)
37            iRate = 0;
38        // If the input interest rate is greater than 0.01 (i.e., 1%), divide the number by 100
39        if(iRate <0.01)
40            iRate /= 100;
41
42        interestRate = iRate;
43    }
44
45    public double getBalance() {
46        return balance;
47    }
```

(24) Please note there isn't **setBalance**() method in this class. To update the balance, the **deposit()** and **withdraw()** methods are needed. In the two methods, value validation also need to be implemented.

```
49    public void deposit(double amout) {
50        // If the input value is less than 0, set value to zero (0).
51        if (amout<0)
52            amout=0;
53        balance += amout;
54    }
55
56    public void withdraw(double amout) {
57        // If the input value is less than 0, set value to zero (0).
58        if (amout<0)
59            amout=0;
60        balance -= amout;
61    }
```

(25) Finally, the addInterest() method is a simple calculation.

```
63    public void addInterest() {
64        balance+= balance * interestRate;
65    }
```

(26) Please implement the other methods that accepts string arguments.