Classes (2)

Han-fen Hu

Tony Gaddis (2019) Starting Out with Java: From Control Structures through Data Structures, 4th Edition

UNLV

Outline

- `this` Reference
- Java built-in Classes: Random Class
- Passing and Returning Objects
- copy Method
- Null Reference
- toString Method
- equals Method

UNLV

`this` Reference

- **`this`** reference is simply a name that an object can use to refer to itself.
- **`this`** reference can be used to overcome shadowing and allow a parameter to have the same name as an instance field.

```
public void setFeet(int ft) {
  this.feet = ft;
  //sets this instance's feet field
  //equal to the parameter ft.
}
```

UNLV

`this` Reference (2)

```
public class Stock {
    private String symbol;      // Trading symbol of stock
    private double sharePrice; // Current price per share
    /**
    Constructor
    @param sym The stock's trading symbol.
    @param price The stock's share price.
     */
    public Stock(String symbol, double sharePrice) {
        this.symbol = symbol;
        this.sharePrice = sharePrice;
    }
```

`symbol` here refers to the parameter.

`this` here refers to the object

`this.symbol` thus refers to the object's **`symbol`** field.

`this` Reference (3)

```
public class Rectangle {
    private double width;
    private double length;

    public double getWidth() {
        return width;
    }
    public void setWidth(double width) {
        this.width = width;
    }
    public double getLength() {
        return length;
    }
    public void setLength(double length) {
        this.length = length;
    }
    public double getArea() {
        return width * length;
    }
}
```

Scope & Shadowing

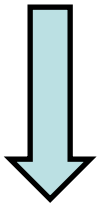
- Variables declared as instance fields in a class can be accessed by any object method in the same class
- A parameter variable is, in effect, a local variable.
  - Within a method, variable names must be unique.
  - ***Shadowing****:* A method may have a local variable with the same name as an Instance Fields.
- Shadowing is discouraged and local variable names should not be the same as instance field names

Avoid Shadowing

parameter, a local variable

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

field, can be accessed by any method in the same class



```
public void setNumber(int num) {
    number = num;
}
```


UNLV

Random Class (1)

- Some applications, such as games and simulations, require the use of randomly generated numbers.
- The Java API has a class, **Random**, for this purpose.
- To use the **Random** class, use the following **import** statement and create an instance of the class.

```
import java.util.Random;

Random randomNumbers = new Random();
```

`Random` Class (2)

Method	Description
`nextDouble()`	Returns the next random number as a double. The number will be within the range of 0.0 and 1.0.
`nextFloat()`	Returns the next random number as a float. The number will be within the range of 0.0 and 1.0.
`nextInt()`	Returns the next random number as an int. The number will be within the range of an int, which is –2,147,483,648 to +2,147,483,648.
`nextInt(int n)`	This method accepts an integer argument, n. It returns a random number as an int. The number will be within the range of 0 (inclusive) to n (exclusive).

See more details at
http://docs.oracle.com/javase/7/docs/api/java/util/Random.html

UNLV

```java
import java.util.Scanner;
import java.util.Random;

/**
   This program simulates the rolling of dice.
*/

public class RollDice {
   public static void main(String[] args)   {
      String again = "y";  // To control the loop
      int die1;            // To hold the value of die #1
      int die2;            // to hold the value of die #2

      // Create a Scanner object to read keyboard input.
      Scanner keyboard = new Scanner(System.in);

      // Create a Random object to generate random numbers.
      Random rand = new Random();

      // Simulate rolling the dice.
      while (again.equalsIgnoreCase("y"))      {
         System.out.println("Rolling the dice...");
         die1 = rand.nextInt(6) + 1;
         die2 = rand.nextInt(6) + 1;
         System.out.println("Their values are:");
         System.out.println(die1 + " " + die2);

         System.out.print("Roll them again (y = yes)? ");
         again = keyboard.nextLine();
      }
   }
}
```

```java
import java.util.Scanner;
import java.util.Random;

/**
   This program simulates the rolling of dice.
*/

public class RollDice {
   public static void main(String[] args)   {
      String again = "y";  // To control the loop
      int die1;            // To hold the value of die #1
      int die2;            // to hold the value of die #2

      // Create a Scanner object to read keyboard input.
      Scanner keyboard = new Scanner(System.in);

      // Create a Random object to generate random numbers.
      Random rand = new Random();

      // Simulate rolling the dice.
      while (again.equalsIgnoreCase("y"))      {
         System.out.println("Rolling the dice...");
         die1 = rand.nextInt(6) + 1;
         die2 = rand.nextInt(6) + 1;
         System.out.println("Their values are:");
         System.out.println(die1 + " " + die2);

         System.out.print("Roll them again (y = yes)? ");
         again = keyboard.nextLine();
      }
   }
}
```

Create an object for random number

Rand.nextInt(6)
Generates numbers between 0 (inclusive) to 6 (exclusive)

UNLV

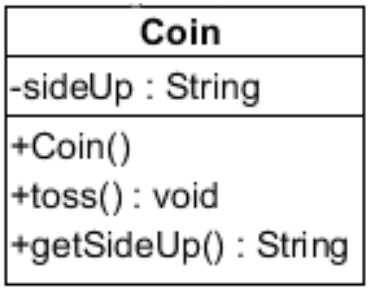
```
import java.util.Scanner;
import java.util.Random;

/**
   This program simulates the rolling of dice.
*/

public class RollDice {
   public static void main(String[] args) {
      String again = "y";  // To control the loop
      int die1;            // To hold the value of die #1
      int die2;            // to hold the value of die #2

      // Create a Scanner object to read keyboard input.
      Scanner keyboard = new Scanner(System.in);

      // Create a Random object to generate random numbers.
      Random rand = new Random();

      // Simulate rolling the dice.
      while (again.equalsIgnoreCase("y"))     {
         System.out.println("Rolling the dice...");
         die1 = rand.nextInt(6) + 1;
         die2 = rand.nextInt(6) + 1;
         System.out.println("Their values are:");
         System.out.println(die1 + " " + die2);

         System.out.print("Roll them again (y = yes)? ");
         again = keyboard.nextLine();
      }
   }
}
```

The dice rolling program can roll the dice multiple times

The loop stops when the user enter anything other than y

UNLV

Lab (1)



- Coin
  - The Coin class simulates a coin
    - **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.

UNLV

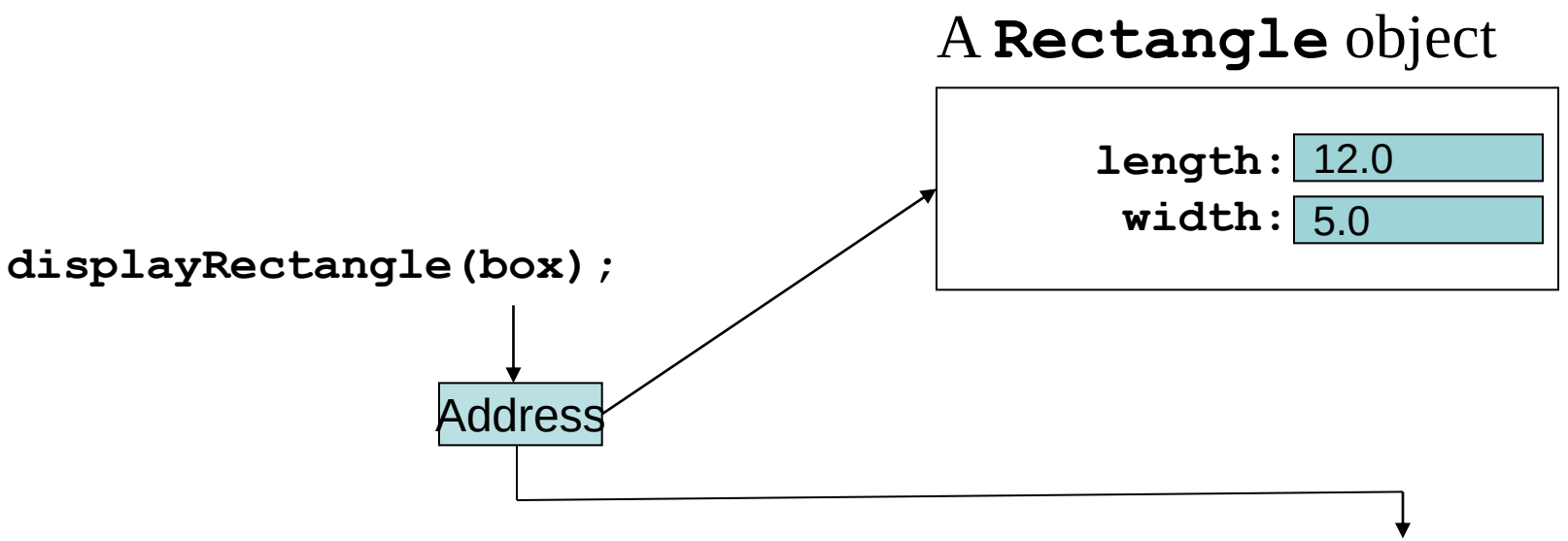
Passing Objects as Arguments (1)

- In Java, all arguments of the primitive data types are passed by value
  - Only a copy of an argument's value is passed into a parameter variable.
  - If a parameter variable is changed inside a method, it has no affect on the original argument.

Passing Objects as Arguments (2)

- Objects can be passed to methods as arguments.
  - When an object is passed as an argument, the value of the reference variable is passed.
  - The value of the reference variable is an address or reference to the object in memory.
  - A copy of the object is not passed, just a pointer to the object.
- When a method receives a reference variable as an argument, it is possible for the method to modify the contents of the object referenced by the variable.

Passing Objects as Arguments (3)




```
public static void displayRectangle(Rectangle r){
  // Display the length and width.
  System.out.println("Length: " + r.getLength() +
                     " Width: " + r.getWidth());
}
```

Pass Object Example

```
public class PassObject {
        public static void main(String[] args) {
           // Create a Rectangle object.
           Coin myCoin = new Coin();

           // Display the object's contents.
           System.out.println(myCoin.getSideUp());

           // Pass a reference to the object to the
           // flipCoin method.
           tossInAMethod(myCoin);

           // Display the object's contents again.
           System.out.println("After tossing the coin outside the scope of the main method: ");
           System.out.println(myCoin.getSideUp());
        }

        public static void tossInAMethod(Coin c) {
            c.toss();
        }
}
```

```
Tails
After tossing the coin outside the scope of the main method:
Heads
```

Lab (2)

- Die.java
  - Die class simulates a die with any number of sides. It also contains the roll() method that will determine the value by a random number.
- DiceGame.java
  - 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.

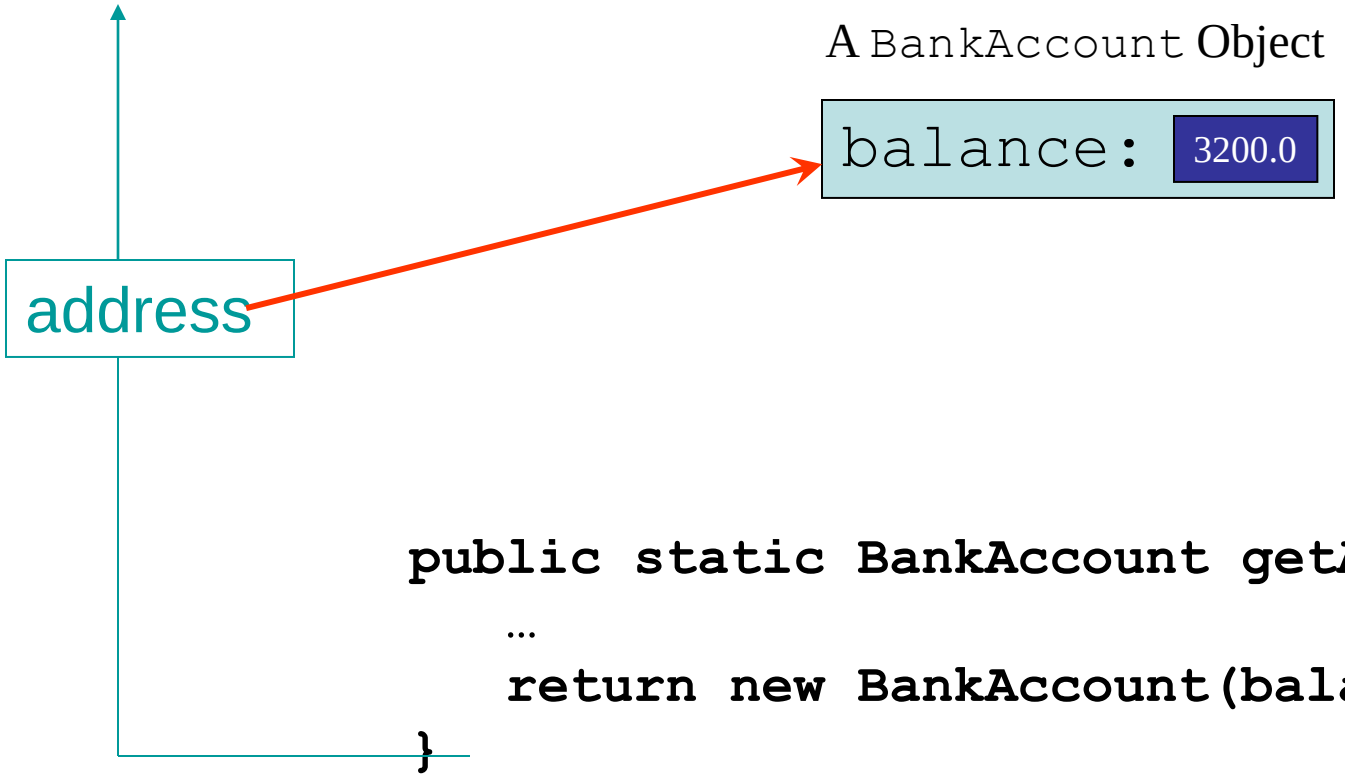
Returning Objects From Methods (1)

- Methods can return references to objects as well.
- Just as with passing arguments, a copy of the object is **not** returned, only its address.
- Method return type:

```
public static BankAccount getAccount(){
        …
      return new BankAccount(balance);
}
```

Returning Objects from Methods (2)

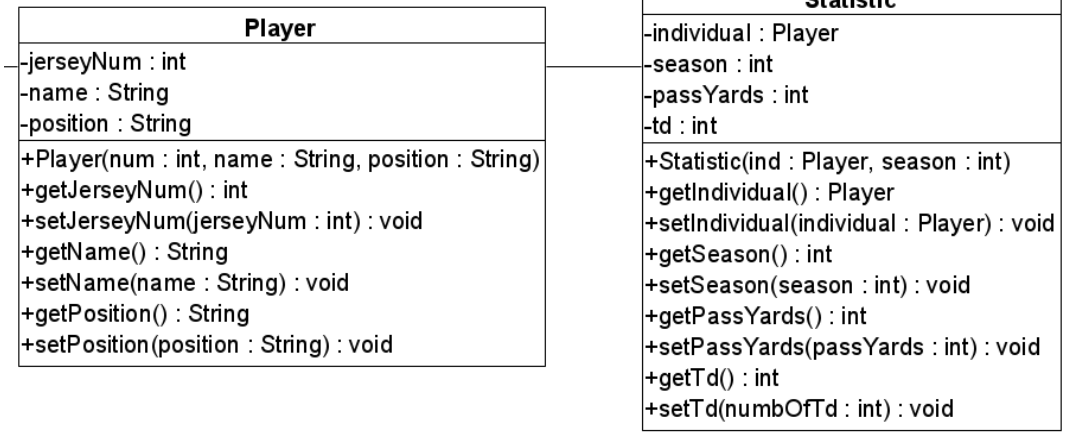
```
account = getAccount();
```




```
public static BankAccount getAccount() {
    …
    return new BankAccount(balance);
}
```


UNLV

Example of Returning an Object




```
3  public class Stat {
4      private int season;
5      private Player player;
6      private int passingYards;
7      private int numOfTD;
8
9
10     public int getSeason() {
11         return season;
12     }
13     public void setSeason(int season) {
14         this.season = season;
15     }
16     public Player getPlayer() {
17         return player;
18     }
19     public void setPlayer(Player player) {
20         this.player = player;
21     }
22     public int getPassingYards() {
23         return passingYards;
24     }
25     public void setPassingYards(int passingYards) {
26         this.passingYards = passingYards;
27     }
28     public int getNumOfTD() {
29         return numOfTD;
30     }
31     public void setNumOfTD(int numOfTD) {
32         this.numOfTD = numOfTD;
33     }
34
35 }
36
```

Object as A Reference Variable

- Objects can be passed to methods as arguments.
  - When an object is passed as an argument, the value of the reference variable is passed.
  - The value of the reference variable is an address or reference to the object in memory.
  - A copy of the object is not passed, just a pointer to the object.
- Methods can return references to objects as well.
  - Just as with passing arguments, a copy of the object is **not** returned, only its address.

Methods That Copy Objects

- Deep copy
  - Create a new instance of the class and copying the values from one object into the new object.

```
public Stock copy()    {
       Stock copyObject = new Stock(symbol, sharePrice);
       // Return a reference to the new object.
       return copyObject;
}
```

```
------------------
Stock company1 = new Stock("XYZ", 9.62);
Stock company2;

company2 = company1.copy();
```

Lab (3)

- Stock.java
  - It simulates a stock on the market and includes the constructor, getters and setters.
- CopyDemo.java
  - It makes 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.

Null References

- A *null reference* is a reference variable that points to nothing.
- If a reference is null, then no operations can be performed on it.
- References can be tested to see if they point to null prior to being used.

```
if(lastName != null){
   ……
}
```

UNLV

Lab (4)

- FullName.java
  - It simulates a full name
- NameTester.java
  - It demonstrates a NullPointerException

`toString` Method

- All objects have a **`toString`** method that returns the class name and a hash of the memory address of the object.
- We can override the default method with our own to print out more useful information.
- The toString method of a class can be called *explicitly*:

```
Stock xyzCompany = new Stock ("XYZ", 9.62);

System.out.println(xyzCompany.toString());
```

`toString` Method (2)

- However, the **`toString`** method does not have to be called explicitly but is called implicitly whenever you pass an object of the class to **`println`** or **`print`**.

```
Stock xyzCompany = new Stock ("XYZ", 9.62);

System.out.println(xyzCompany);
```

- The **`toString`** method is also called implicitly whenever you concatenate an object of the class with a string.

```
Stock xyzCompany = new Stock ("XYZ", 9.62);

System.out.println("The stock data is:\n" +
xyzCompany);
```

toString Method (3)

- Lab
  - Stock.java
  - ObjectCopy.java
    - The class attempts to print an object of the Stock class.
    - We can override the default toString() method to print out more useful information.

`equals` Method (1)

- When the == operator is used with reference variables, the memory address of the objects are compared.
- All objects have an **`equals`** method.
  - The default operation of the equals method is to compare memory addresses of the objects
  - The contents of the objects are not compared.
- We can override the **`equals`** method to do more meaningful comparison

`equals` Method (2): Example

- The **`Stock`** class has an **`equals`** method.
- If we try the following:

```
Stock stock1 = new Stock("GMX", 55.3);
Stock stock2 = new Stock("GMX", 55.3);
if (stock1 == stock2) // This is a mistake.
  System.out.println("The objects are the same.");
else
  System.out.println("The objects are not the same.");
```

only the addresses of the objects are compared.

`equals` Method (3)

- We should use the **`equals`** method to compare to **`Stock`** objects

```
public boolean equals(Stock object2) {
   boolean status;

   if(symbol.equals(Object2.symbol)
      && sharePrice == Object2.sharePrice)
      status = true;
   else
      status = false;
   return status;
}
```

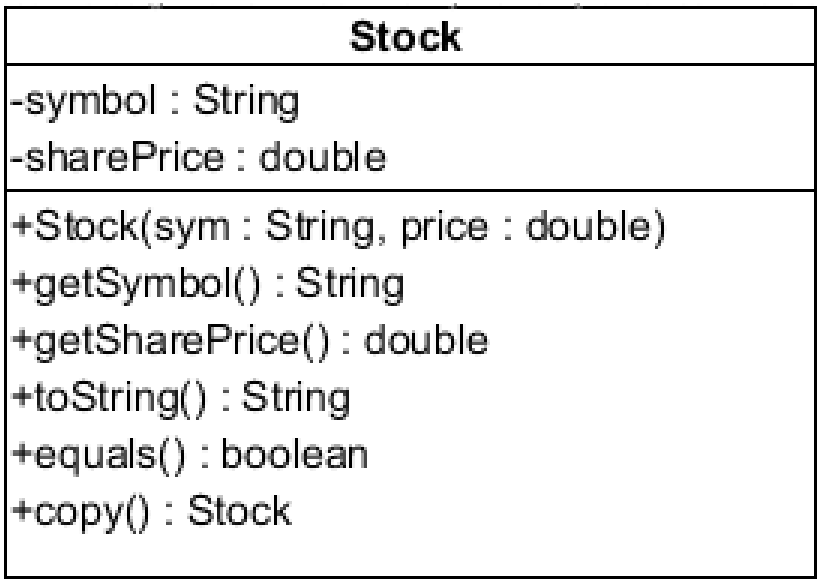
- Now, objects can be compared by their contents rather than by their memory addresses.

Lab (5)

- Stock.java
  - Override the default equals() method
- CopyDemo.java

Good Design of a Class

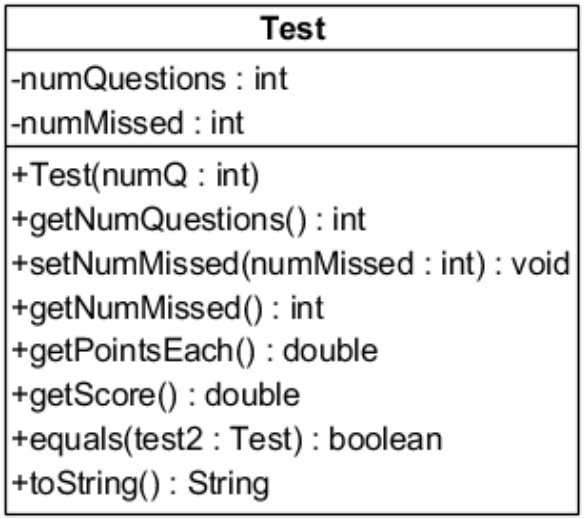
- Now we have a well-designed model class.
  - Fields
  - Constructors
  - Getter and setters
  - toString() method
  - equals() method
  - copy() method



Finding Classes and Their Responsibilities

- Finding the classes
  - Get written description of the problem domain
  - Identify all nouns, each is a potential class
  - Refine list to include only classes relevant to the problem
- Identify the responsibilities
  - Things a class is responsible for knowing
  - Things a class is responsible for doing
  - Refine list to include only classes relevant to the problem

Class Exercise: Test Class (1)



- 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.
  - However, if the value passed to this constructor is not a positive number, make numQuestion as 0.

UNLV

Class Exercise: Test Class (2)

- 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
  - However, when numQuestion is 0, make the points for each question as 0
- The getScore() method should calculate the final score for this test.
  - For example, if there are 40 questions in the test and the test-taker missed 3 questions, the score should be 92.5.

Class Exercise: Test Class (3)

- The equals() method compares 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

Class Exercise: Test Class (4)

- Create a program to demonstrate this Test class.
  - 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.
```