Exercise 5

Objectives

The aim of this exercise is to learn the concept of inheritance. In this exercise you will learn:

- The concept of inheritance
- How to create a sub-class from a base-class
- How to construct a class hierarchy

A: Overriding and Hiding Methods [1 pt]

Consider the following two classes:

```
public class ClassA {
    public void methodOne(int i) { }
    public void methodTwo(int i) { }
    public static void methodThree(int i) { }
    public static void methodFour(int i) { }
}

public class ClassB extends ClassA {
    public static void methodOne(int i) { }
    public void methodTwo(int i) { }
    public void methodThree(int i) { }
    public static void methodFour(int i) { }
}
```

Answer the following questions (in `OverridingAndHidingMethods.txt`):

- Which method overrides a method in the superclass?
- Which method hides a method in the superclass?
- What do the other methods do?

Submission Files	Types
`OverridingAndHidingMethods.txt`	`Texts in English`

B: Independent Classes [1 pt]

You should create three independent classes: Point, MovingPoint, and TestObjectApplication. Point and MovingPoint should contain instant variables that represent the coordinate (x, y). You should define constructors to accept and initialize the coordinate. Point and MovingPoint classes should include a method that prints its position and a method that calculates the distance between a given point and itself. Furthermore, the MovingPoint class should have integers vx and vy representing the velocity in the x and y directions, respectively, and should also have a move method that moves itself by adding vx and vy to the current location x and y, respectively.

速度

The main method of the TestObjectApplication class should instantiate two Point objects and two MovingPoint objects with specified initial positions. For the MovingPoint object, you should set their velocity and move them. Finally, it should print the coordinate of each object and distance against the specific object. You should use the following template for the TestObjectApplication:

```
class TestObjectApplication{
   public static void main(String[] args){
       Point p1 = new Point(10, 10);
       Point p2 = new Point(10, 20);
       MovingPoint m1 = new MovingPoint(0, 0);
       MovingPoint m2 = new MovingPoint(0, 0);
       m1.setVelocity(2, 3);
       m2.setVelocity(-4, 2);

       for ( int i = 0; i < 10; i++ ) {
           m1.move();
           m2.move();
       }

       p1.print();
       p2.print();
       m1.print();
       m2.print();

       System.out.println(p1.getDistance(p2));
       System.out.println(m1.getDistance(m2));
   }
}
```

Using the template, you can obtain:

```
(10, 10)
(10, 20)
(20, 30)
(-40, 20)
10.0
60.8276253029822
```

When you finish your program, analyze your code and answer the following questions (in `IndependentClasses.txt`):

- How many fields does the Point have?
- How many fields does the MovingPoint have?
- How many methods does the Point have?
- How many methods does the MovingPoint have?
- Are there any fields with the same meaning in the Point and MovingPoint classes?
- Are there any methods in the Point and MovingPoint classes expressing similar behavior?

Submission Files	Types
`Point.java`	`Java Class`
`MovingPoint.java`	`Java Class`
`TestObjectApplication.java`	`Java Class`
`IndependentClasses.txt`	`Texts in English`

C: Inherited Classes [2 pt]

Analyze your solution to the previous problem. Answer the following questions:

- Can you design inherited classes?
- You have two classes: Point and MovingPoint. Which one should be a base class?

You should design inherited classes: Point and MovingPoint. Their functionality should be the same as in the previous problem. The TestObjectApplication class should output the same result.

When you finish your program, analyze your code and answer the following questions (in `InheritedClasses.txt`):

- How many fields does the Point have?
- How many fields does the MovingPoint have?
- How many methods does the Point have?
- How many methods does the MovingPoint have?
- What are the advantages that the solution to Problem C has over the solution to Problem B?
- Can you execute the following operation at the end of TestObjectApplication? And why?

```
System.out.println(p1.getDistance(m1));
System.out.println(p2.getDistance(m2));
```

Submission Files	Types
`Point.java`	`Java Class`
`MovingPoint.java`	`Java Class`
`TestObjectApplication.java`	`Java Class`
`InheritedClasses.txt`	`Texts in English`

D: Testing Cast [2 pt]

Create a class TestCastApplication to test the class MovingPoint implemented in Problem C. The class TestCastApplication must perform the following operations:

1. Instantiate the class MovingPoint and create a MovingPoint object located on (0, 0). Refer to this object by an object reference o1 of type MovingPoint.
2. Instantiate class MovingPoint and create a MovingPoint object located on (3, 4). Reference this object by an object reference o2 of type Point.
3. Cast o2 to the MovingPoint type and assign it to the object reference o3 of the MovingPoint type.
4. For each object referenced by o1, o2, and o3, the coordinate of the object is output by the print method.
5. Print the distance between objects indicated by o1 and o2.
6. Print the distance between objects indicated by o2 and o3.

The class TestCastApplication should be executed and the following output should be obtained.

```
(0, 0)
(3, 4)
(3, 4)
5.0
0.0
```

In addition, consider the following points and explain (in `TestingCast.txt`):

- Which of the above 1. - 6. operations are implicit casts?
- Which of the operations in 1.- 6. above are explicit casts?
- Why can an object reference of type Point refer to an object of type MovingPoint?
- In operation 3., can we assign o2 to o3 without casting and why?

Submission Files	Types
`TestingCast.txt`	`Texts in English`
`TestCastApplication.java`	`Java Class`

Summary

In this exercise, we learned about the concept of inheritance, how to create a sub-class from a base-class as well as how to construct a class hierarchy. We studied some advantages of inheritance through a simple class hierarchy. Also, we checked the availability of method through an object reference through the concept of “type” and casting operation.