Exercise 2

Objectives

The objective of this exercise is to be able to create simple classes. In creating classes, students will understand the concept of object-oriented encapsulation by defining state and behavior. In this exercise, students will learn:

- How to define a class
- How to instantiate an object from the class
- How to access fields and methods of the created objects

A: Creating Objects [1 pt]

Your task is to create the Point class. The Point class has x-coordinate and y-coordinate values as fields and has methods to set and get them respectively.

Test the Point class with the PointCreationApplication class shown below.

```
import java.util.Scanner;

class PointCreationApplication{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);

        Point p1 = new Point();
        Point p2 = new Point();

        p1.setX(sc.nextInt());
        p1.setY(sc.nextInt());
        p2.setX(sc.nextInt());
        p2.setY(sc.nextInt());

        System.out.println("(" + p1.getX() + ", " + p1.getY() + ")");
        System.out.println("(" + p2.getX() + ", " + p2.getY() + ")");
    }
}
```

If PointCreationApplication produces the corresponding output for the following sample inputs, it can be judged that the Point class has been implemented correctly.

Sample Input	**Sample Output**
1 2 3 4	(1, 2) (3, 4)

Submission Files	Types
`Point.java`	`Java Class`
`PointCreationApplication.java`	`Java Class`

B: Manipulating Objects [1 pt]

Your task is to extend the Point class by adding the *move* method. The *move* method takes two integers dx and dy and moves the x-coordinate value of the point by dx and the y-coordinate value by dy.

Test the Point class with the PointMovingApplication class shown below. Note that the initial position of the point is the origin (0, 0).

```
import java.util.Scanner;

class PointMovingApplication{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);

        Point pt = new Point();

        int Q = sc.nextInt();
        for ( int i = 0; i < Q; i++ ){
            int dx = sc.nextInt();
            int dy = sc.nextInt();
            pt.move(dx, dy);
        }

        System.out.println("(" + pt.getX() + ", " + pt.getY() + ")");
    }
}
```

If PointMovingApplication produces the corresponding output for the following sample inputs, it can be judged that the Point class has been implemented correctly.

Sample Input	Sample Output

5
1 2
3 4
5 6
7 8
9 10
	(25, 30)

Submission Files	Types
`Point.java`	`Java Class`
`PointMovingApplication.java`	`Java Class`

* Since `Point.java` satisfies the specification of Problem A, only one `Point.java` needs to be submitted for Problems A and B.

C: Manipulating Objects II [1 pt]

Your task is to extend the Point class by adding a constraint in the *move* method as follows.

The *move* method does not execute the body of the method if the absolute value of the x-coordinate value or the absolute value of the y-coordinate value of the point exceeds 100 as a result of that operation.

Test the Point class with the PointMovingApplication class shown above.

If PointMovingApplication produces the corresponding output for the following sample inputs, it can be judged that the Point class has been implemented correctly.

Sample Input	Sample Output

```
10
20 10
35 65
10 30
0 -15
10 30
-90 10
-80 0
5 -5
-80 2
20 -11
```
	`(-80, 86)`

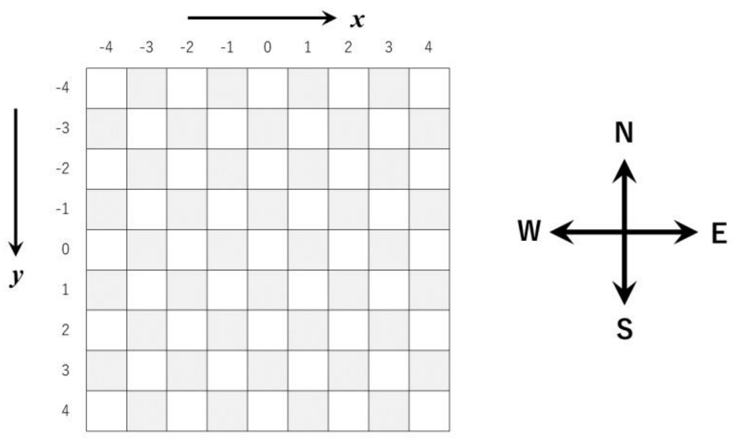
Submission Files	Types
`Point.java`	`Java Class`

* Since `Point.java` satisfies the specification of Problem A and B, only one `Point.java` needs to be submitted for Problems A, B, and C.

D: Robot Simulator [2 pt]

Your task is to create the TwirlingRobot class that meets the following specifications.

The robot moves on a 9 x 9 grid as shown in the figure below. The robot can move from its current square to squares in the north, east, south, and west directions. The (x, y) coordinate system is as shown in the figure.



The TwirlingRobot class must implement the following fields and methods

Field Name	Details
`x`	Integer value to represent x-coordinate of the robot.
`y`	Integer value to represent y-coordinate of the robot.
`dir`	Robot orientation. North, East, South, and West are represented by integer values 0, 1, 2, and 3, respectively.

Method Name	Details
`initialize`	Set the robot's initial position (x, y) and direction dir
`turnLeft`	Rotate 90 degrees to the left
`turnRight`	Rotate 90 degrees to the right
`move`	Moves forward one square toward the current direction. However, if the robot goes outside the grid as a result of the operation, no action is taken.
`printLocation`	Displays the current position of the robot.

An object of TwirlingRobot should be operated by executing RobotSimulatorApplication class shown below.

```
import java.util.Scanner;

class RobotSimulatorApplication{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);

        TwirlingRobot tr = new TwirlingRobot();

        tr.initialize(0, 0, 0);

        String commands = sc.next();
        for ( int i = 0; i < commands.length(); i++ ){
            char command = commands.charAt(i);
            if ( command == 'M' ) tr.move();
            else if ( command == 'R' ) tr.turnRight();
            else if ( command == 'L' ) tr.turnLeft();
        }
        tr.printLocation();
    }
}
```

The above RobotSimulatorApplication creates a robot instance and performs several operations.

If RobotSimulatorApplication produces the corresponding output for the following sample inputs, it can be judged that the TwirlingRobot class has been implemented correctly.

Sample Input	**Sample Output**
`MMMMMM`	`(0, -4)`
`RMMMLMMLMMMMMMLMMRMMM`	`(-4, 0)`
`LLLLRRMMMMMMLMMMMMMMLMMLMMMMRMM`	`(0, 0)`
`MLMRMLMRMLMRMLMMMMRMMM`	`(-4, -4)`

Submission Files	**Types**
`TwirlingRobot.java`	`Java Class`
`RobotSimulatorApplication.java`	`Java Class`

Summary

In this exercise, by implementing the Robot class, we confirmed how to define the class state (fields) and behavior (methods). By manipulating one instance of the created class with another instance, the behavior of the object was verified. In the Robot class, the concept of encapsulation was learned by accessing the fields x, y, and dir through the method. The values of the fields (x, y, dir) could be changed correctly (safely) through the method.