Assignment 6

> Designer: ZHU Yueming
>
> Junit Designer: LI Xiaolei
>
> Tester: HUA Wenjia

Introduce

This assignment is to exercise the interface, and implement classes of interface and Comparable or Comparator Interface.

In this assignment, we will continue to use the `Direction`, `Location`, `Shape`, `Circle`, `RightTriangle` classes that have been designed in assignment 5. You can use the classes designed by yourself, or use the classes we provided.

In this assignment, the names class like `ShapeCanvas`, `AvoidConflictShapeCanvas` and `OverLapShapeCanvas` should be the same as requirements and do not modify or remove any fields or methods that have been already defined in document. You can add other classes, fields and methods if you need. You can also override superclass' methods if necessary.

We provided a junit local judge, you can download it from bb website or from the link below:

https://github.com/zhuym1219/assignment5and6_in_24spring/tree/main

What to Submit?

Location.java

Direction.java

Shape.java

Circle.java

RightTriangle.java

ShapeCanvas.java

AvoidConflictShapeCanvas.java

OverLapShapeCanvas.java

Interface ShapeCanvas

It is an interface.

Methods

addShape()

```
public boolean addShape(int x, int y, char pattern, int... params);
```

Implement the method in its implement classes.

getSpaceGridCount()

```
public int getSpaceGridCount();
```

Implement the method in its implement classes.

This method is to return an int value, which represents how many space value in canvas.

getShapeCount()

```
public int getShapeCount();
```

Implement the method in its implement classes.

This method is to return an int value, which represents how many shapes being added into canvas successfully.

getShapesByArea()

```
public List<Shape> getShapesByArea();
```

Implement the method in its implement classes.

This method is to return a List, which contains all shapes in canvas, and sort the shape as :

- Ascending order of the area of shapes.
- If shapes with a same area, sorted by the ascending order of its character value of pattern.

Shapes with a same area and same pattern cannot appear in any test case in this assignment.

getShapesByLocation()

```
public List<Shape> getShapesByLocation();
```

Implement the method in its implement classes.

This method is to return a List, which contains all shapes in canvas, and sort the shape as :

- Ascending order of the x value in location.
- If shapes with a same value of x, sorted by the ascending order of its y value in location.
- If shapes with a same location, sorted by the ascending order of its character value of

pattern.

Shapes with a same location and same pattern cannot appear in any test case in this assignment.

getCanvas()

```
public char[][] getCanvas();
```

Implement the method in its implement classes.

This method is to return a `char[][]` array, which represents a painted canvas. If a cell should be painted, it will be the pattern. If a cell haven' t been paint, it will be a space like

`' '`

AvoidConflictShapeCanvas

It is the concrete implement class of the interface `ShapeCanvas`

Attributes

shapes

```
private List<Shape> shapes;
```

Design a attribute shapes, which is to store the successfully added shapes.

canvas

```
private char[][] canvas;
```

Using a `char[][]` array to represent the canvas. The initial value of each grid in canvas is a space `' '`, which mean an empty grid.

Methods:

You need implement all abstract methods in the interface `ShapeCanvas`

Constructor()

```
public AvoidConflictShapeCanvas(int rows, int cols)
```

Design a constructor with two parameters, rows represents the height of canvas and cols represents the width of canvas. Those are also the rows and columns of the private field `canvas`.

addShape()

```
@Override
    public boolean addShape(int x, int y, char pattern, int... params)
```

This method is to add a shape into canvas, and the parameter means:

- x and y: the location x and y of shape
- pattern: the pattern of shape
- params: The length of params only be 1 or 3 in all test cases in this assignment.
    - **For circle**: the length of params is 1, which represents the radius of circle
    - **For RightTriangle**: the length of params is 3.
        - The first one is the **width** of RightTriangle
        - The second one is the **height** of RightTriangle
        - The third one is the **index of Direction**. For example, if the third value is 0, it means `LEFT_UP` direction.

The **return value** is determined by whether the shape can be added successfully, which means if it has no conflict when adding a shape, the method returns true, otherwise it returns false.

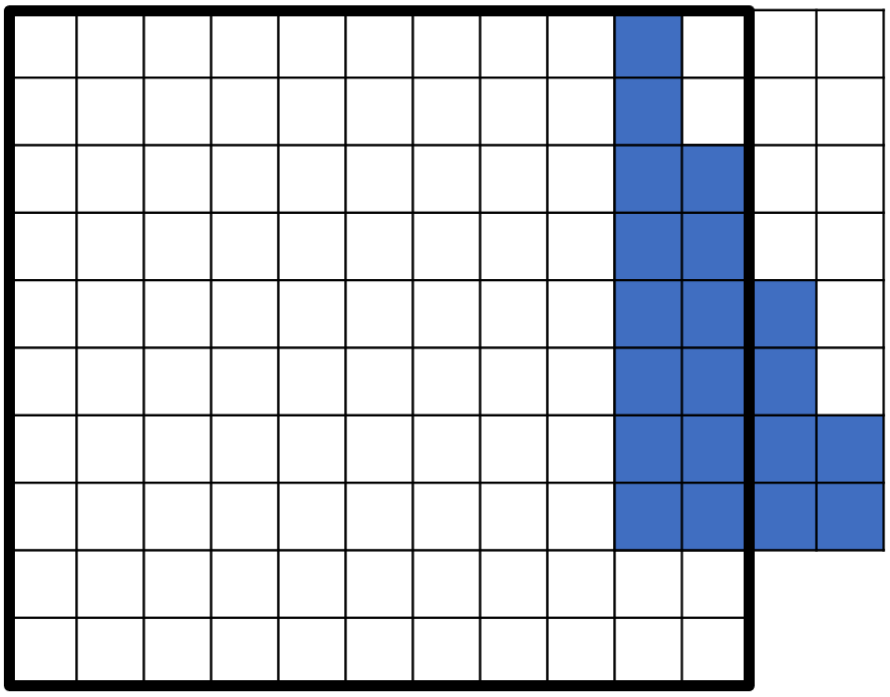
The conflict includes:

- The shape is out of the bound of canvas.
- Overlap Conflict: Has non-space grid of shape in corresponding location of the canvas is also non-space.

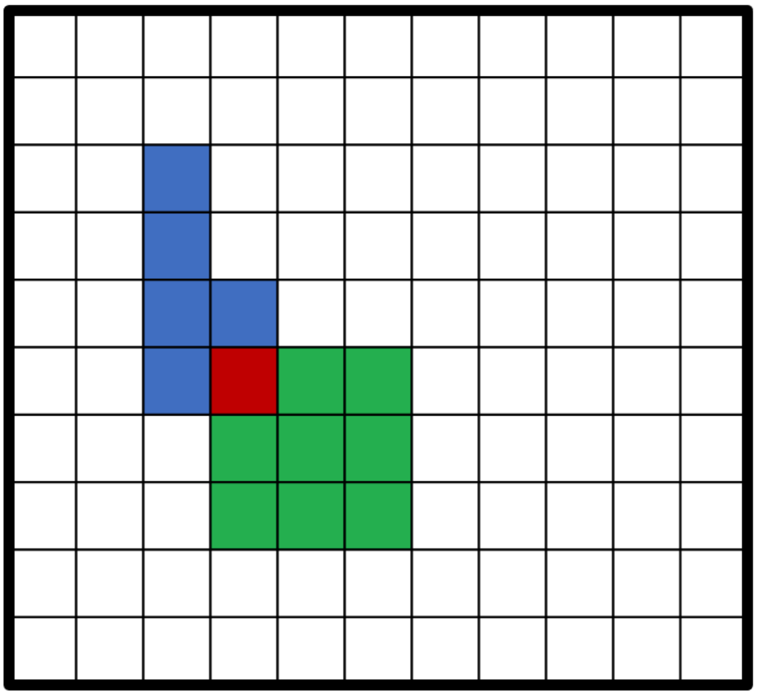
For example:

It is the out of bound conflict

(0, 0) (0, y)



It is the overLap conflict.



Test code:

```
public static void main(String[] args) {
        ShapeCanvas shapeCanvas = new AvoidConflictShapeCanvas(20, 20);
        System.out.println(shapeCanvas.addShape(0, 2, 'A', 5, 3, 1));
        System.out.println(shapeCanvas.addShape(6, 8, 'B', 5, 7, 2));
        System.out.println(shapeCanvas.addShape(8, 12, 'C', 5));
        System.out.println(shapeCanvas.addShape(6,6,'D',5,7,1));
        System.out.println(shapeCanvas.addShape(0,8,'E',3));

        shapeCanvas.getShapesByArea().forEach(System.out::println);
        shapeCanvas.getShapesByLocation().forEach(System.out::println);

        for (char[] line:shapeCanvas.getCanvas()) {
            System.out.println(line);
        }
    }
```

Result:

```
true
true
false
true
true
RightTriangle: (0,2) area=11 pattern=A
RightTriangle: (6,8) area=23 pattern=B
RightTriangle: (6,6) area=23 pattern=D
Circle: (0,8) area=36 pattern=E
RightTriangle: (0,2) area=11 pattern=A
Circle: (0,8) area=36 pattern=E
RightTriangle: (6,6) area=23 pattern=D
RightTriangle: (6,8) area=23 pattern=B
  AA    EEEEEE
  AAAA  EEEEEE
  AAAAA EEEEEE
        EEEEEE
        EEEEEE
        EEEEEE
      D BBBBB
      DDBBBBB
      DDDBBBB
      DDD BBB
      DDDDBBB
      DDDDDBB
      DDDDD B
```

OverLapShapeCanvas

It is the concrete implement class of the interface `ShapeCanvas`

Attributes

shapes

```
private List<Shape> shapes;
```

Design a attribute shapes, which is to store the successfully added shapes.

canvas

```
private char[][] canvas;
```

Using a `char[][]` array to represent the canvas. The initial value of each grid in canvas is a space `' '`, which mean an empty grid.

Methods:

You need implement all abstract methods in the interface `ShapeCanvas`

Constructor()

```
public OverLapShapeCanvas(int rows, int cols)
```

Design a constructor with two parameters, rows represents the height of canvas and cols represents the width of canvas. Those are also the rows and columns of the private field `canvas`.

addShape()

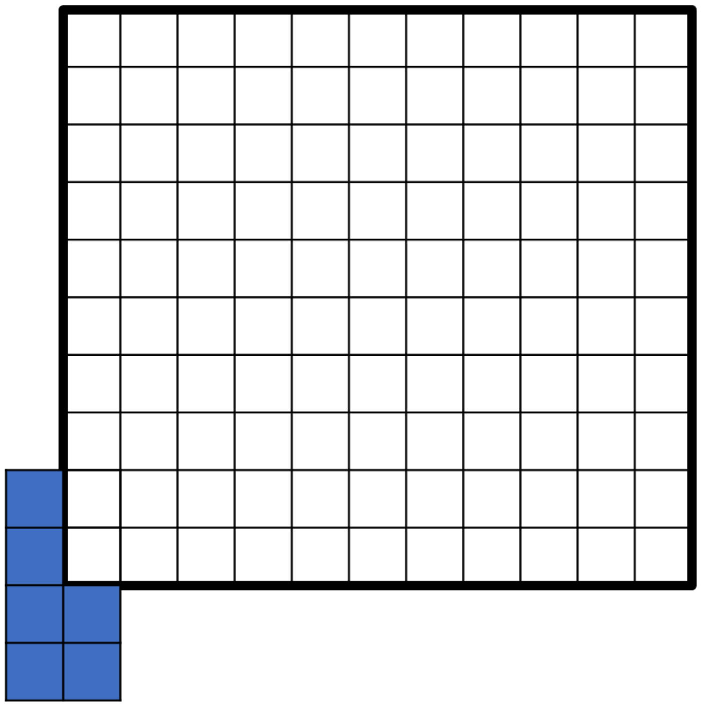
```
@Override
    public boolean addShape(int x, int y, char pattern, int... params)
```

This method is to add a shape into canvas, and the parameter means:

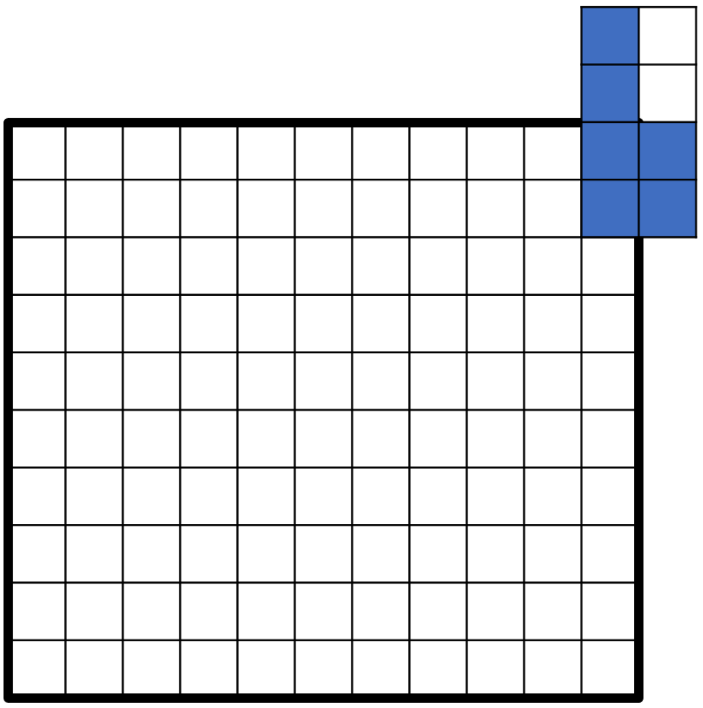
- x and y: the location x and y of shape
- pattern: the pattern of shape
- params:
  - **For circle**: the length of params is 1, which represents the radius of circle
  - **For RightTriangle**: the length of params is 3.
    - The first one is the **width** of RightTriangle
    - The second one is the **height** of RightTriangle
    - The third one is the **index of Direction**. For example, if the third value is 0, it means `LEFT_UP` direction.

Return value: Has one non-space grid of shape in corresponding location of canvas is in the bound, in this case it returns true, otherwise returns false.

It is a false example:



It is a true example:



In this class, we allow shapes to be stored overlappingly, and shapes added later can be placed overlapping on top of those being added.

When placed overlappingly, the **space grid** in shape means empty and it cannot cover the original non-space grid.

Test code:

```java
public static void main(String[] args) {
        ShapeCanvas canvas1 = new OverLapShapeCanvas(15, 15);
        canvas1.addShape(0, 0, 'A', 6);
        canvas1.addShape(1, 1, 'B', 5);
        canvas1.addShape(2, 2, 'C', 4);
        canvas1.addShape(3, 3, 'D', 3);
        canvas1.addShape(10, 5, 'E', 4, 6, 2);
        canvas1.addShape(14, 14, 'F', 4, 6, 3);
        canvas1.addShape(10, 5, '0', 3, 2, 1);
        canvas1.addShape(10, 5, '1', 1, 1, 2);
        for (char[] line : canvas1.getCanvas()) {
            System.out.println(line);
        }
        System.out.println(canvas1.getShapeCount());
        System.out.println(canvas1.getSpaceGridCount());
        canvas1.getShapesByArea().forEach(System.out::println);
        canvas1.getShapesByLocation().forEach(System.out::println);
    }
```

Result:

```
  AAAAAAAA
 AABBBBBBAA
AABCCCCCCBAA
ABCDDDDDDCBA
ABCDDDDDDCBA
ABCDDDDDDCBA
ABCDDDDDDCBA
ABCDDDDDDCBA
ABCDDDDDDCBA
AABCCCCCCBAA
 AABB10EEAA
  AAA000EA
      EEE
       EE
       EE
7
86
RightTriangle: (10,5) area=1 pattern=1
RightTriangle: (10,5) area=5 pattern=0
RightTriangle: (10,5) area=16 pattern=E
Circle: (3,3) area=36 pattern=D
Circle: (2,2) area=60 pattern=C
Circle: (1,1) area=88 pattern=B
Circle: (0,0) area=132 pattern=A
Circle: (0,0) area=132 pattern=A
```

```
Circle: (1,1) area=88 pattern=B
Circle: (2,2) area=60 pattern=C
Circle: (3,3) area=36 pattern=D
RightTriangle: (10,5) area=5 pattern=0
RightTriangle: (10,5) area=1 pattern=1
RightTriangle: (10,5) area=16 pattern=E
```