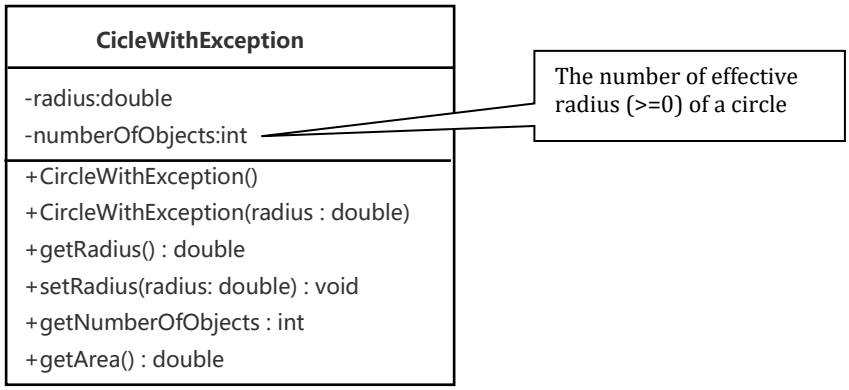
1. (25 Points) Declaring, Throwing, and Catching Exceptions

Define a new circle class named CircleWithException, in which the setRadius(double newRadius) mathod throws an IllegalArgumentException if the augument newRadius is negative.




```
public class CircleWithException {
  // The raidus of the circle
  // The numberoOfObjects created
  //Construct a circle with a specified radius
  //return radius
  //set a new radius {
        //Declare exception
        //Throw exception
  //}
  //return numberOfObjects
  //return the area of this circle
}
```

```
public class Main{
    public static void main(String[] args) {
        try {
            CircleWithException c1 = new CircleWithException(5);
            CircleWithException c2 = new CircleWithException(-5);
            CircleWithException c3 = new CircleWithException(0);
        }
        catch (IllegalArgumentException ex) {
            System.out.println(ex);
        }
        System.out.println("Number of objects created: " + CircleWithException.getNumberOfObjects() );
    }
}
```

Output:

```
java.lang.IllegalArgumentException: Radius cannot be negative
Number of objects created: 1
```

2. (30 Points) Interface Movable and its implementations

Suppose that we have a set of objects with some common behaviors: they could move up, down, left or right. The exact behaviors (such as how to move and how far to move)

depend on the objects themselves. One common way to model these common behaviors is to define an interface called Movable, with abstract methods moveUp(), moveDown(), moveLeft() and moveRight(). The classes that implement the Movable interface will provide actual implementation to these abstract methods.

(1) Let's write two concrete classes - MovablePoint and MovableCircle - that implement the Movable interface. The code for the interface Movable is straight forward.

```
public interface Movable {  // saved as "Movable.java"
   public void moveUp();
   ......
}
```

(2) For the MovablePoint class, declare the instance variable x, y, xSpeed and ySpeed with package access as shown with '~' in the class diagram in Fig.1 (i.e., classes in the same package can access these variables directly). For the MovableCircle class, use a MovablePoint to represent its center (which contains four variable x, y, xSpeed and ySpeed). In other words, the MovableCircle composes a MovablePoint, and its radius.

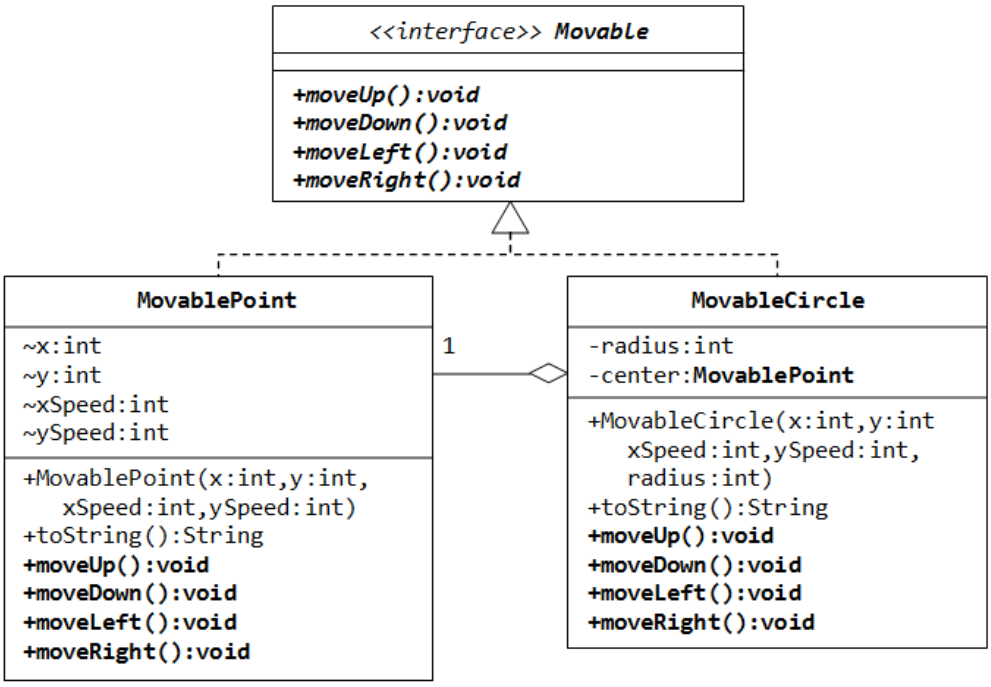


Fig. 1: Interface Movable and its implementations

```
public class MovablePoint implements Movable { // saved as "MovablePoint.java"
   // instance variables
   int x, y, xSpeed, ySpeed;    // package access
   // Constructor
   public MovablePoint(int x, int y, int xSpeed, int ySpeed) {
     this.x = x;
     ......
  }
  ......
  // Implement abstract methods declared in the interface Movable
```

```
    @Override
    public void moveUp() {
        y += ySpeed;   // y-axis pointing down for 2D graphics
    }
    ......
}
public class MovableCircle implements Movable { // saved as "MovableCircle.java"
    // instance variables
    private MovablePoint center;   // can use center.x, center.y directly
                                   //  because they are package accessible
    private int radius;
    // Constructor
    public MovableCircle(int x, int y, int xSpeed, int ySpeed, int radius) {
    // Call the MovablePoint's constructor to allocate the center instance.
    center = new MovablePoint(x, y, xSpeed, ySpeed);
     ......
    }
    // Implement abstract methods declared in the interface Movable
    @Override
    public void moveUp() {
        center.y += center.ySpeed;
    }
    ......
}
```

```
public class Main{
    public static void main(String[] args) {
        Movable m1 = new MovablePoint(5, 6, 10, 15); // upcast
        System.out.println(m1);
        m1.moveLeft();
        System.out.println(m1);
        m1.moveUp();
        System.out.println(m1);

        Movable m2 = new MovableCircle(1, 2, 3, 4, 20); // upcast
        System.out.println(m2);
        m2.moveRight();
        System.out.println(m2);
        m2.moveDown();
        System.out.println(m2);
    }
}
```

OutPut:

```
MovablePoint (5, 6) with xSpeed = 10 and ySpeed = 15
MovablePoint (-5, 6) with xSpeed = 10 and ySpeed = 15
MovablePoint (-5, 21) with xSpeed = 10 and ySpeed = 15
MovableCircle at point MovablePoint (1, 2) with xSpeed = 3 and ySpeed = 4 with radius = 20
MovableCircle at point MovablePoint (4, 2) with xSpeed = 3 and ySpeed = 4 with radius = 20
MovableCircle at point MovablePoint (4, -2) with xSpeed = 3 and ySpeed = 4 with radius = 20
```

3. (45 Points) Superclass Shape and its subclasses Circle, Rectangle and Square

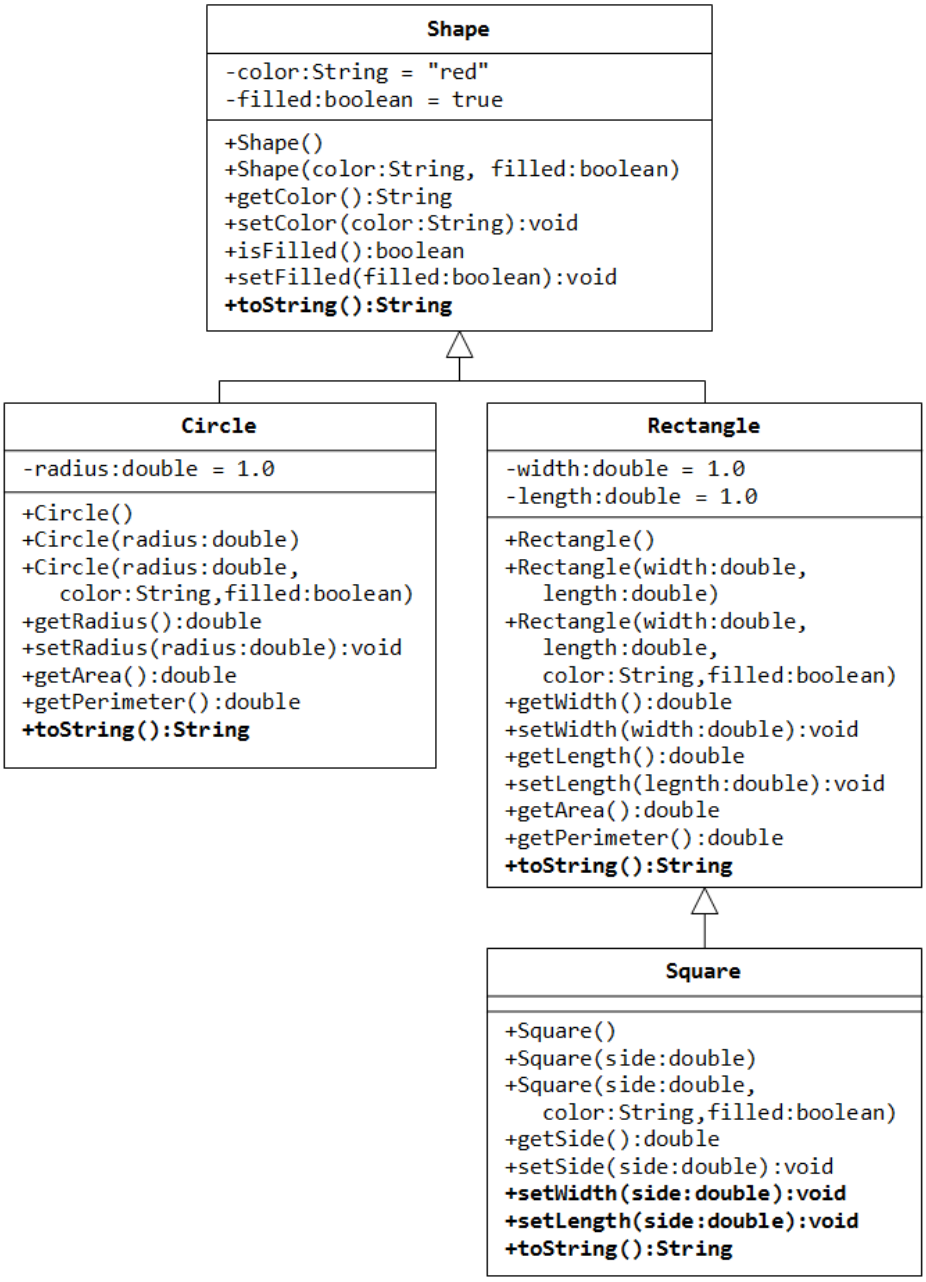


Fig. 2: Superclass Shape and its subclasses Circle, Rectangle and Square

3.1 Superclass Shape and its subclasses Circle, Rectangle and Square

1) Write a superclass called Shape (as shown in Fig. 2), which contains:
   - Two instance variables color (String) and filled (boolean).
   - Two constructors: a no-arg (no-argument) constructor that initializes the color to "red" and filled to true, and a constructor that initializes the color and filled to the given values.
   - Getter and setter for all the instance variables. By convention, the getter for a boolean variable xxx is called isXXX() (instead of getXxx() for all the other types).
   - A toString() method that returns "A Shape with color of xxx and filled/Not filled".
2) Write two subclasses of Shape called Circle and Rectangle, as shown in Fig. 2.

①The Circle class contains:

- An instance variable radius (double).
- Three constructors as shown. The no-arg constructor initializes the radius to 1.0.
- Getter and setter for the instance variable radius.
- Methods getArea() and getPerimeter().
- Override the toString() method inherited, to return "A Circle with radius=xxx, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

②The Rectangle class contains:

- Two instance variables width (double) and length (double).
- Three constructors as shown. The no-arg constructor initializes the width and length to 1.0.
- Getter and setter for all the instance variables.
- Methods getArea() and getPerimeter().
- Override the toString() method inherited, to return "A Rectangle with width=xxx and length=zzz, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

3) Write a class called Square, as a subclass of Rectangle. Convince yourself that Square can be modeled as a subclass of Rectangle. Square has no instance variable, but inherits the instance variables width and length from its superclass Rectangle.

- Provide the appropriate constructors (as shown in Fig. 2). Hint:

```
public Square(double side) {
    super(side, side);  // Call superclass Rectangle(double, double)
}
```

- Override the toString() method to return "A Square with side=xxx, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.
- Do you need to override the getArea() and getPerimeter()? Try them out.
- Override the setLength() and setWidth() to change both the width and length, so as to maintain the square geometry.

```
public class Main {
    public static void main(String[] args) {
        // Upcast Circle to Shape
        Shape s1 = new Circle(5.5, "green", false);
        System.out.println(s1);
        System.out.println(s1.getColor());
        System.out.println(s1.isFilled());

        // Downcast back to Circle
        Circle c1 = (Circle)s1;
        System.out.println(c1);
        System.out.println(c1.getColor());
        System.out.println(c1.isFilled());

        // Upcast
        Shape s2 = new Rectangle(1.0, 2.0, "green", false);
        System.out.println(s2);
        System.out.println(s2.getColor());

        //Downcast
        Rectangle r1 =  (Rectangle)s2;
        System.out.println(r1);
        System.out.println(r1.getArea());
        System.out.println(r1.getColor());
        System.out.println(r1.getLength());
```

```
        // Upcast
        Shape s3 = new Square(6.6);
        System.out.println(s3);
        System.out.println(s3.getColor());

        Rectangle r2 =  (Rectangle)s3;
        System.out.println(r2);
        java.text.DecimalFormat df =new java.text.DecimalFormat("#.00");
        double area1 = r2.getArea();
        System.out.println(df.format(area1));
        System.out.println(r2.getColor());
        System.out.println(r2.getLength());

        // Downcast Rectangle r2 to Square
        Square sq1 = (Square)r2;
        System.out.println(sq1);
        area1 = sq1.getArea();
        System.out.println(df.format(area1));
        System.out.println(sq1.getColor());
        System.out.println(sq1.getSide());
        System.out.println(sq1.getLength());
    }
}
```

Output：

```
    A Circle with radius = 5.5, which is a subclass of A Shape with color of green and NOT filled
    green
    false
    A Circle with radius = 5.5, which is a subclass of A Shape with color of green and NOT filled
    green
    false
    A Rectangle with width = 1.0 and length = 2.0, which is a subclass of A Shape with color of green and
NOT filled
    green
    A Rectangle with width = 1.0 and length = 2.0, which is a subclass of A Shape with color of green and
NOT filled
    2.0
    green
    2.0
    A Square with side = 6.6, which is a subclass of A Rectangle with width = 6.6 and length = 6.6, which is a
subclass of A Shape with color of red and filled
    red
    A Square with side = 6.6, which is a subclass of A Rectangle with width = 6.6 and length = 6.6, which is a
subclass of A Shape with color of red and filled
    43.56
    red
    6.6
    A Square with side = 6.6, which is a subclass of A Rectangle with width = 6.6 and length = 6.6, which is a
subclass of A Shape with color of red and filled
    43.56
    red
    6.6
    6.6
```

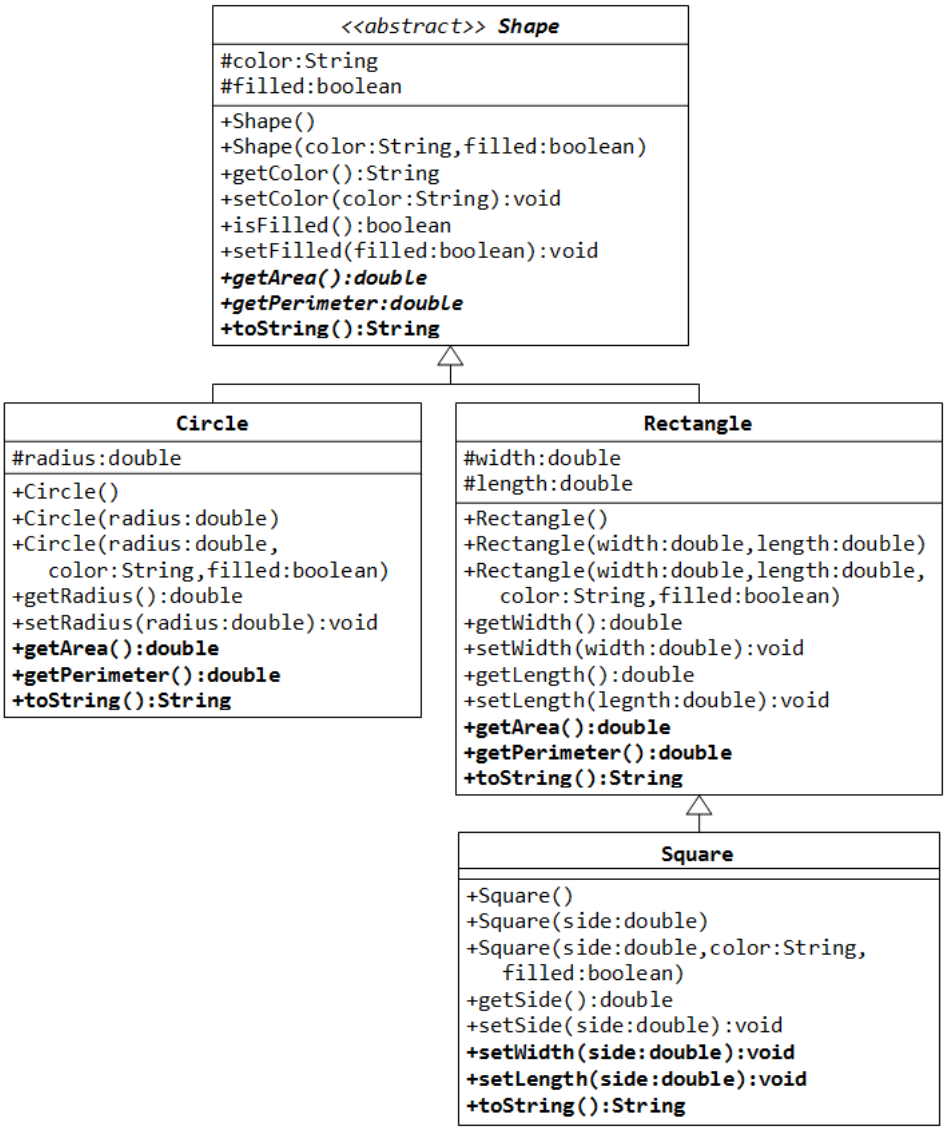


Fig. 3: Abstract Shape and its subclasses Circle, Rectangle and Square

3.2 Abstract Shape and its subclasses Circle, Rectangle and Square

1) Rewrite the superclass Shape and its subclasses Circle, Rectangle and Square, as shown in Fig. 3. In this exercise, Shape shall be defined as an abstract class, which contains:
   - Two protected instance variables color(String) and filled(boolean).
     The protected variables can be accessed by its subclasses and classes in the same package. They are denoted with a '#' sign in Fig. 3.
   - Getter and setter for all the instance variables, and toString().
   - Two abstract methods getArea() and getPerimeter() (shown in italics in Fig.3).

2) The subclass Circle and Rectangle shall override the abstract methods getArea() and getPerimeter() and provide the proper implementation. They also override the toString().

The Main class is used to test these statements involving polymorphism. Some statements may trigger compilation errors. Explain the errors, if any.

```
public class Main{
```

```
        public static void main(String[] args) {
                java.text.DecimalFormat df =new java.text.DecimalFormat("#.00");
                Shape s1 = new Circle(5.5, "green", false); // Upcast Circle to Shape
                System.out.println(s1);  // which version?
                double Area1 = s1.getArea();
                double Perimeter1 = s1.getPerimeter();
                System.out.println(df.format(Area1)); // which version?
                System.out.println(df.format(Perimeter1));
                System.out.println(s1.getColor());
                System.out.println(s1.isFilled());
//Shape.test

                Circle c1 = (Circle)s1; // Downcast back to Circle
                System.out.println(c1);
                double Area2 = c1.getArea();
                double Perimeter2 = c1.getPerimeter();
                System.out.println(df.format(Area2));
                System.out.println(df.format(Perimeter2));
                System.out.println(c1.getColor());
                System.out.println(c1.isFilled());
                System.out.println(c1.getRadius());

               Shape s2 = new Shape(); // Why wrong?

//Circle.test

                Shape s3 = new Rectangle(1.0, 2.0, "green", false); // Upcast
                System.out.println(s3);
                double Area3 = s3.getArea();
                double Perimeter3 = s3.getPerimeter();
                System.out.println(df.format(Area3));
                System.out.println(df.format(Perimeter3));
                System.out.println(s3.getColor());


                Rectangle r1 = (Rectangle)s3; // downcast
                System.out.println(r1);
                double Area4 = r1.getArea();
                System.out.println(df.format(Area4));
                System.out.println(r1.getColor());
                System.out.println(r1.getLength());
//Rectangle.test

                Shape s4 = new Square(6.6); // Upcast
                System.out.println(s4);
                double Area5 = s4.getArea();
                System.out.println(df.format(Area5));
                System.out.println(s4.getColor());

                Rectangle r2 = (Rectangle)s4;
                System.out.println(r2);
                double Area6 = r2.getArea();
                System.out.println(df.format(Area6));
                System.out.println(r2.getColor());
                System.out.println(r2.getLength());

                Square sq1 = (Square)r2;
                System.out.println(sq1);
                double Area7 = sq1.getArea();
                System.out.println(df.format(Area7));
                System.out.println(sq1.getColor());
                System.out.println(sq1.getSide());
```

```
            System.out.println(sq1.getLength());
//Square.test
        }
}
```

Output：

```
        A Circle with radius = 5.5, which is a subclass of A Shape with color of green and NOT filled
        95.03
        34.56
        green
        false
        A Circle with radius = 5.5, which is a subclass of A Shape with color of green and NOT filled
        95.03
        34.56
        green
        false
        5.5
        A Rectangle with width = 1.0 and length = 2.0, which is a subclass of A Shape with color of green and
NOT filled
        2.00
        6.00
        green
        A Rectangle with width = 1.0 and length = 2.0, which is a subclass of A Shape with color of green and
NOT filled
        2.00
        green
        2.0
        A Square with side = 6.6, which is a subclass of A Rectangle with width = 6.6 and length = 6.6, which is a
subclass of A Shape with color of red and filled
        43.56
        red
        A Square with side = 6.6, which is a subclass of A Rectangle with width = 6.6 and length = 6.6, which is a
subclass of A Shape with color of red and filled
        43.56
        red
        6.6
        A Square with side = 6.6, which is a subclass of A Rectangle with width = 6.6 and length = 6.6, which is a
subclass of A Shape with color of red and filled
        43.56
        red
        6.6
        6.6
```

Bonus Question: Chess (20 Points)

In this part, you need to complete the Chess game in command line. First, design a chessboard UI in commend line. Second, complete the logics of chess and other game features. Next, we also introduce the design patterns used in Chess. Finally, we provide some edition of the previous code.

1. Design a chessboard UI:

The following chessboard image is the screenshot from Eclipse, you can design your own chessboard output but need to explain in your document.

```
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P 6P 6P 6P 6P 6P
6
5
4
3
2 P6 P6 P6 P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
```

The first column and the first row is the index of the Chessboard.

The combination of number+character such as `3R` means black chess, and the character+number such as `R3` means white chess.

Here is the meaning of the character and the number:

chess	King	Queen	Rook	Bishop	Knight	Pawn
Character	K	Q	R	B	N	P
Number	1	2	3	4	5	6

As we can only use the keyboard to control the chess, thus we use the following instruction to move the chess. `Caution: You need to handle the invalid instruction to guarantee the robustness of the program`

Examples:

Instruction	Meaning
23 43	chess at (Row 2, Column 3) move to (Row 4, Column 3)
74 54	chess at (Row 7, Column 4) move to (Row 5, Column 4)
7 61	Invalid
`LoadBoard` when there is no saved board	Invalid
91 71	Invalid

Chessboard output example:

```
Enter name of user 1 as white player: Alice
Enter name of user 2 as black player: Bob
Enter "StartGame" to start a new game or "LoadBoard" to load from file: StartGame
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P 6P 6P 6P 6P 6P
6
5
4
3
2 P6 P6 P6 P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
Enter the action of white player: 23 43
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P 6P 6P 6P 6P 6P
6
5
4       P6
3
2 P6 P6    P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
Enter the action of black player: 74 54
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P    6P 6P 6P 6P
6
5          6P
4       P6
3
2 P6 P6    P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
Enter the action of white player: 43 54
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P    6P 6P 6P 6P
6
5          P6
4
3
2 P6 P6    P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
Enter the action of black player: 84 54
  1  2  3  4  5  6  7  8
8 3R 5N 4B    1K 4B 5N 3R
7 6P 6P 6P    6P 6P 6P 6P
6
5          2Q
4
3
2 P6 P6    P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
```

Invalid Example:

```
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P 6P 6P 6P 6P 6P
6
5
4       P6
3
2 P6 P6    P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
Enter the action of black player: 75 54
Invalid
Enter the action of black player: 91 71
Invalid
Enter the action of black player: 7 61
Invalid
Enter the action of black player: 84 64
Invalid
```

2. Game features to be completed:

- Game rules to be completed based on Exercise 1:

Chess	Rules
King	King cannot move to the square which is under control of any opponent's pieces.
Queen	Cannot leap over any pieces
Rook	Cannot leap over any pieces
Bishop	Cannot leap over any pieces
Knight	-
Pawn Rule 1	For the first step, it can move two squares and cannot leap over any pieces. Otherwise, only one square at a time.
Pawn Rule 2	`En passant` : It can capture an opponent's piece on a square diagonally in front of it on an adjacent file, by moving to that square.
Pawn Rule 3	Promotion: When a pawn advances to the eighth rank, as a part of the move it is *promoted* and must be exchanged for the player's choice of queen, rook, bishop, or knight of the same color.

  Some hard rules not included in the table such as castling and check is not required in this homework, you can also complete it if you want, if you did please write a document to tell us how well you have done.

- Exit:

  Player can type `Exit` to exit the game.

- Undo:

  Player can type `Undo` to undo.

  Examples:

```
Enter name of user 1 as white player: Alice
Enter name of user 2 as black player: Bob
Enter "StartGame" to start a new game or "LoadBoard" to load from file: StartGame
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P 6P 6P 6P 6P 6P
6
5
4
3
2 P6 P6 P6 P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
Enter the action of white player: 23 43
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P 6P 6P 6P 6P 6P
6
5
4       P6
3
2 P6 P6    P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
Enter the action of black player: 74 54
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P    6P 6P 6P 6P
6
5          6P
4       P6
3
2 P6 P6    P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
Enter the action of white player: Undo
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P 6P 6P 6P 6P 6P
6
5
4       P6
3
2 P6 P6    P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
Enter the action of black player: Undo
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P 6P 6P 6P 6P 6P
6
5
4
3
2 P6 P6 P6 P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
```

- Save and LoadBoard:

  Player can type `Save` to save the chessboard, or type `LoadBoard` to load the chessboard of last game record from file.

  Example:

  Enter `Save` to save current chessboard.

```
Enter name of user 1 as white player: Alice
Enter name of user 2 as black player: Bob
Enter "StartGame" to start a new game or "LoadBoard" to load from file: StartGame
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P 6P 6P 6P 6P 6P
6
5
4
3
2 P6 P6 P6 P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
Enter the action of white player: 23 43
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P 6P 6P 6P 6P 6P
6
5
4       P6
3
2 P6 P6    P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
Enter the action of black player: 74 54
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P    6P 6P 6P 6P
6
5          6P
4       P6
3
2 P6 P6    P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
Enter the action of white player: Save
```

Enter `LoadBoard` to load chessboard from file:

```
Enter name of user 1 as white player: Alice
Enter name of user 2 as black player: Bob
Enter "StartGame" to start a new game or "LoadBoard" to load from file: LoadBoard
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P    6P 6P 6P 6P
6
5          6P
4       P6
3
2 P6 P6    P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
Enter the action of white player: 43 54
  1  2  3  4  5  6  7  8
8 3R 5N 4B 2Q 1K 4B 5N 3R
7 6P 6P 6P    6P 6P 6P 6P
6
5          P6
4
3
2 P6 P6    P6 P6 P6 P6 P6
1 R3 N5 B4 Q2 K1 B4 N5 R3
```

3. Design Patterns in Chess:

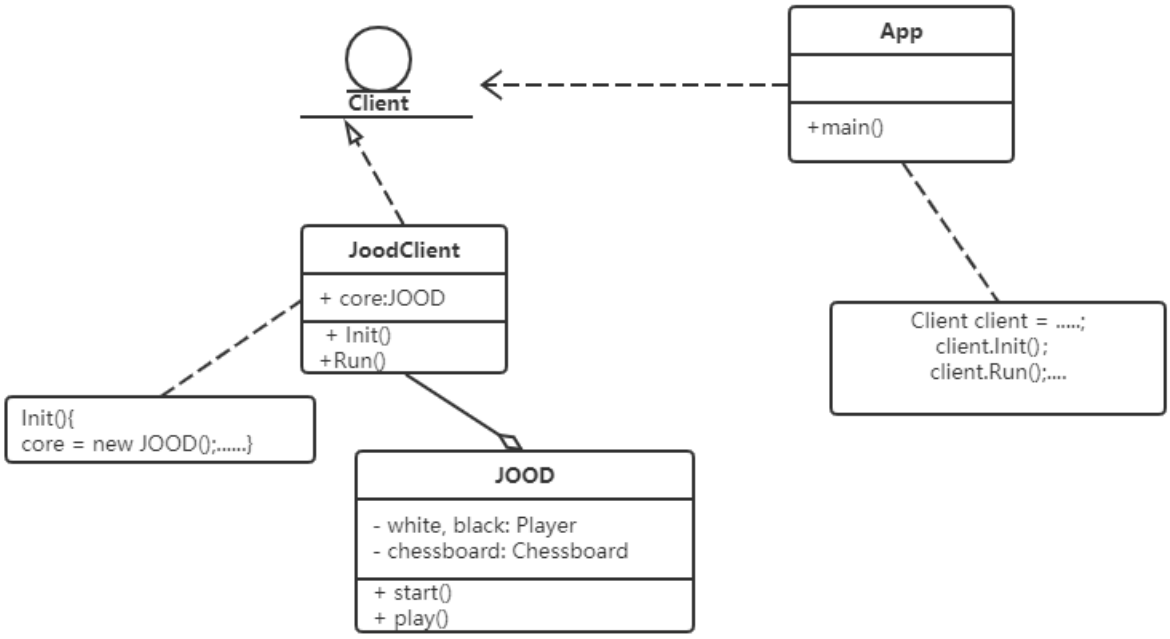
- Simple factory pattern

As we have told you in Exercise 1, simple factory pattern is used to create each type of chess in `Chessman.create()`. You can regard the Chessman class as a factory, it will create corresponding chess for chessboard according to the parameter which means `new a needed chess and return it in the create function`.

- Proxy Pattern

Why we need Proxy Pattern in Chess?

We will have two UI, one is the commend line UI, the other one is the graphical interfaces. As we use the proxy pattern, it easy to change.



Thus, in this Exercise, you need to complete `cli/JoodClient.java,JOOD.java, Chessboard.java, Chessman.java` at least, you also need to do some addition or deletion in chessman class to complete the rules above. Of course, you can also define your own classes or interfaces as you like, the code we provided is only as a reference.

4. Some Edition we did in the code

The whole code of the framework we design is on the website.

- `Chessman.java`

`_color, chess_status` are changed to non static parameter.

`Promote` , `can_promote` and `canCross` are moved from `Chessman.java` to `Chessboard.java`