Tutorial of Composition & Package

> Based on the tutorial of "2020S-Java-A" designed by teaching group in SUSTech
>
> Modified (only change to markdown file) by ZHU Yueming in 2021. April. 6th
>
> Add Exercise 3 and Exercise 4 by ZHU Yueming in 2022.Nov.7th
>
> Remove contents of part 1 & part 2 and add instructions of package by WANG Qing in 2023. April. 7th

Objective

- Learn to use composition of classes.
- Learn to create packages to organize classes.

Part 1: Composition of class

We have learned how to define a Java class and create its object, and we also have acquainted ourselves with constructor, setter, getter methods and static data field of class. If we wanna use other objects in the definition of a new class, is it workable in Java? A class can have references to objects of other classes as members. This is called **composition** and is sometimes referred as a **has-a relation**.

Exercise 1

Design a class named **Direction**, the objects of which represents directions:

- Private data fields:

  int **row**; // represents the direction of row

  int **col**; // represents the direction of col
- Design a **constructor** to initialize the two private data fields.
- You can add other fields or methods that you think are necessary.

Design a class named **Person**:

- Private data fields:

  Direction[] **directions**;// represents all the directions the person can go

  int **i**;// represents the current vertical position

  int **j**;// represents the current horizontal position

  int **index**;// represents the index of current direction in the `directions` field
- Design a **constructor** with three parameters to initialize three private data fields includes i, j and index. In constructor, initialize the `directions` fields with 8 directions objects as the table below:

Direction Name	row	col
Right	0	1
Right up	-1	1
Up	-1	0
Left up	-1	-1
Left	0	-1
Left down	1	-1
Down	1	0
Right down	1	1

- Design the **getter** and **setter** methods of those three fields: i, j and index.
- Design a method **changeDirection()** to increase the index of current direction to the next direction.

  ```
  Person p = new Person(0,0,6);
  System.out.println(p.getIndex());
  p.changeDirection();
  System.out.println(p.getIndex());
  p.changeDirection();
  System.out.println(p.getIndex());
  ```

  result:

  ```
  6
  7
  0
  ```

- Design a method **walk(int step)** with an int parameter step, which indicates the person walk steps in current direction, after that the i and j would be changed accordingly.
- Design a **toString()** method, which returns a string describes the current location of the person, like:

  ```
  (1,1)
  ```

Sample Main method:

```
public static void main(String[] args) {
        Person p = new Person(0,-1,0);
        p.walk(3);
        p.changeDirection();
        System.out.println(p);
        p.walk(2);
        p.changeDirection();
        System.out.println(p);
        p.walk(5);
        p.changeDirection();
        System.out.println(p);
    }
```

Sample output:

```
(0,2)
(-2,4)
(-7,4)
```

Exercise 2: Octagon

According to the input and output requirements, draw and print a octagon.

Input description:

Enter an integer n, representing the variable length of the rectangle in which the octagon is located. The number n-1 must be a multiple of positive integer 3, so that the value of n can be `4, 7, 10, 13.....`

Output description:

Print a $n * n$ matrix, in which 1 represents the edge of octagon, 0 represents other.

Sample Input:

```
4
```

Sample output:

```
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
```

Sample Input:

```
7
```

Sample Output

```
0 0 1 1 1 0 0
0 1 0 0 0 1 0
1 0 0 0 0 0 1
1 0 0 0 0 0 1
1 0 0 0 0 0 1
0 1 0 0 0 1 0
0 0 1 1 1 0 0
```

Sample Input:

```
10
```

Sample Output

```
0 0 0 1 1 1 1 0 0 0
0 0 1 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 1 0 0
0 0 0 1 1 1 1 0 0 0
```

Part 2: Package of class

We've seen `import ...` statements in almost every examples in previous labs. Before a class can be imported into other Java programs, it must be placed in a package to make it reusable. The steps for creating a reusable class are:

Step 1

Declare a public class as we did in previous labs.

We will take the file `Circle.java` as an example.

Step 2

Add a **package declaration** statement at the beginning of the `Circle.java` file.

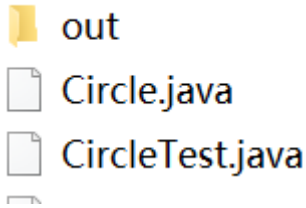
```
package sustech.cs109.lab8;
public class Circle {
        ...
    }
```

Step 3

Compile and run Java file in command line.

Try following command:

- check files in current path. The out folder is an empty folder



- compile `Circle.java` into out folder.

  ```
  javac -d out Circle.java
  ```

  A `Circle.class` file would be created in the path of `./out/sustech/cs109/lab8`

Step 4

Use `import ...` statement to import packaged class into other programs.

```
package sustech.cs109.lab9;
import sustech.cs109.lab8.Circle;

public class CircleTest {
    ...
}
```

- compile `CircleTest.java` in current folder and use the `Circle.class` in `./out/sustech/cs109/lab8`

  ```
  javac -cp out -d . CircleTest.java
  ```

  A `CircleTest.class` file would be created in the path of `./sustech/cs109/lab9`

- Run `CircleTest`

  ```
  java -cp out; sustech.cs109.lab9.CircleTest
  ```

  or

  ```
  java -cp out: sustech.cs109.lab9.CircleTest
  ```

  result:

  ```
  Position of the circle is (0.0, 0.0)
  ```