Tutorial of Constructor and toString

> 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

Objective

- Learn to declare constructors and use them to construct objects
- Learn to declare and use of static data field and `toString( )` method.
- Learn to use composition of class.

Part 1: Constructors and instance methods

The Circle class defined in the previous lab does not contain any explicitly declared constructor. The Java compiler will provide a default constructor that would initialize all three fields (radius, x, y) to 0.0 when called. If we want to create a circle object, of which the three fields have the following values: radius = 2.0, x = 1.0, y = 1.0, we can write a main method like the one below.

```java
public static void main(String[] args) {
   Circle c = new Circle();
   c.setRadius(2.0);
   c.setX(1.0);
   c.setY(1.0);
}
```

However, this is quite troublesome. A better solution is to declare constructors so that they can be called to construct objects with certain radius values and center positions. The following code declares two such constructors. The first constructor takes one argument to initialize the radius field (the fields x and y will be initialized to 0.0). The second constructor takes three arguments to initialize all three fields.

```java
public Circle(double radius) {
    this.radius = radius;
}

public Circle(double radius, double x, double y) {
    this.radius = radius;
    this.x = x;
    this.y = y;
}

```

Note that in the constructors, “this” keyword is needed to differentiate the field access from method argument access. Now we can simply create the circle (radius = 2.0, x = 1.0, y = 1.0) with a constructor call. Much easier, right?

```
public static void main(String[] args) {
   Circle c = new Circle(2.0, 1.0, 1.0);
}
```

Continue to type the following code in the main method and see what happens.

```
Circle c = new Circle();
```

The code would not compile. Do you know why?

Exercise 1

Add a public method distanceToOrigin() to the Circle class. The method returns the distance between the circle’s center point and the origin point `(0.0, 0.0)`. Then write a Java program to perform the following tasks:

1. Generate a random number N in the range `[5, 10)`.
2. Create N circles. Each circle has a random radius in the range [1.0, 3.0) and a random center position: x and y are in the range `[2.0, 5.0)`.
3. Among the generated circles, find the one with the smallest area and the one whose center is the farthest from the origin point.

For random number generation, you may use the following two methods of the Random class:

- `public int nextInt(int bound)`
- `public double nextDouble()`

See https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Random.html for more details.

A sample run:

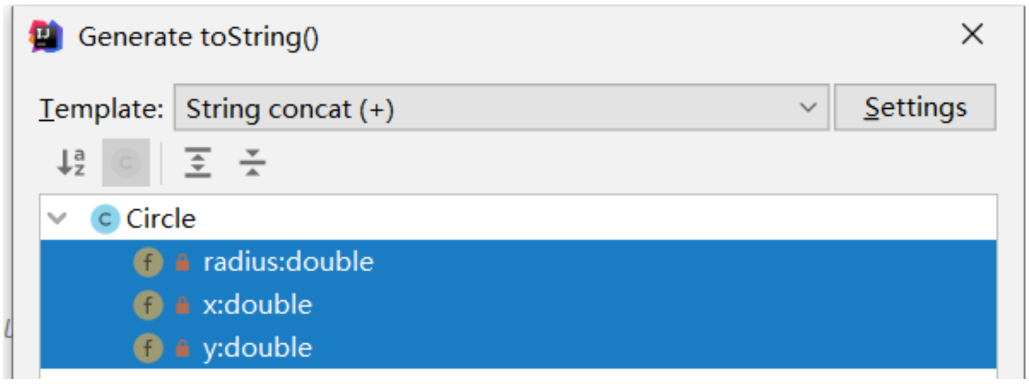
```
Circle #1: radius = 1.49, x = 4.01, y = 3.55
Circle #2: radius = 1.58, x = 3.92, y = 3.77
Circle #3: radius = 1.81, x = 2.89, y = 3.60
Circle #4: radius = 2.47, x = 3.90, y = 3.87
Circle #5: radius = 1.34, x = 2.94, y = 4.05
Circle #5 is the smallest circle, area = 5.62
Circle #4 is the farthest circle, distance to origin = 5.50
```

Part 2: method toString()

A special instance method **`toString( )`** which return a string related to the current object.

While the object is used as a string, the `toString( )` is invoked by default. For example:

- Create a Circle object c : `Circle c = new Circle(1.0, 1.0, 1.0);`
- Use `System.out.print(c)` to print the c, what's the output?
- Use `System.out.print(c.toString())` instead of `System.out.print(c)`, what's the output?
- Use IDEA to generated the `toString()` of the current class.



The following instance method `toString()` is generated, all the instance data field is added into the String:

```
public String toString() {
      return "Circle{" +
            "radius=" + radius +
            ", x=" + x +
            ", y=" + y +
            '}';
 }
```

- Invoke `System.out.print(c)` and `System.out.print(c.toString())` again, what's the output?
- While change the `toString( )` as follow code, invoke `System.out.print(c)`. what's the output, why?

```
public String toString(char z) {
      return "Circle{" +
            "radius=" + radius +
            ", x=" + x +
            ", y=" + y +
      ", z=" + z +
            '}';
 }
```

Exercise 2： Food

Continue to design te class Food.

The class contains:

- Private data fields:

int **id**;

String **name;**

String **type**;

int **size**;

double **price;**

- Implement the **getter** and **setter** method for each private field of Food.
- Implement a five arguments **constructor** to initialize the five private data fields.
- Impement a **toString()** method to return a String, which contains all the information of this food object like:

  ```
  Seafood pizza: (11 Inches) 120.00 $
  ```

In `FoodTest` class, create four objects of Food by using its **constructor** as follows:

Object Name	id	name	type	size	price
pizza1	1	pizza	Seafood	11	12
pizza2	2	pizza	Beef	9	10
Fried rice	3	fried rice	Seafood	5	12
Noodles	4	noodles	Beef	6	14

Create an `ArrayList<Food>` to add those four Food objects, and then print its **toString()** method of them as follows by iterating the `ArrayList<Food>` we created.

```
Seafood pizza: (11 Inches) 120.00 $
Beef pizza: (9 Inches) 100.00 $
Seafood fried rice: (5 Inches) 40.00 $
Beef noodle: (6 Inches) 35.00 $
```

Exercise 3: Composition

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 two private data fields:
- You can add other fields or methods that you think are necessary.

Design a class named **Preson**

- Private data fields:

Direction[] **directions**;// represents the directions the preson can go

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

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

int **index**;// represents the index of current directions

- 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 preson talk steps in current direction, after that the i and j would be changed accordingly.
- Design a **toString()** method, which returns String describe the current location of the preson, 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 4: 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 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 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
```