Tutorial of Comparator

> Designed by ZHU Yueming in 2023. May. 6th

Objective

- Learn how to use the interface `java.util.Comparator<T>`.

Part 1: Comparator Interface

Before exercise

The original code describes a sorting method in the order that the graphics were added.

If we want to sort by the distance from x, y to the origin point, we can write it into the `Shape` class through the `Comparable` interface, such as:

Design code in Shape class:

```
public abstract class Shape implements Comparable<Shape> {
     @Override
    public int compareTo(Shape s) {
        return Double.compare(this.distance(), s.distance());
    }
}
```

Invoker code in Main class:

```
 Collections.sort(shapes);
 for (Shape s : shapes) {
      System.out.println(s);
 }
```

However, using `Comparable` interface is a hard code indicating that the current `Shape` class can only be sorted by distance.
If the requirements are changeable, sometimes we need to sort by area, sometimes we need to sort by the distance, and sometimes we need to sort by the order of x or y, how to achieve it?

Comparator Interface

> Following description is from official document.

A comparison function, which imposes a total ordering on some collection of objects. Comparators can be passed to a sort method (such as Collections.sort or Arrays.sort) to allow precise control over the sort order. Comparators can also be used to control the order of certain data structures (such as sorted sets or sorted maps), or to provide an ordering for collections of objects that don't have a natural ordering.

If we want to sort by area, we can design a sorted class:

```
class SortByArea implements Comparator<Shape> {
    @Override
    public int compare(Shape o1, Shape o2) {
        return Double.compare(o1.area(), o2.area());
    }
}
```

If we want to sort by distance, we can design a sorted class:

```
class SortByDistance implements Comparator<Shape> {
    @Override
    public int compare(Shape o1, Shape o2) {
        return Double.compare(o1.distance(), o2.distance());
    }
}
```

Then we can invoke it by following code:

```
 public static void printByArea(List<Shape> shapes){
        System.out.println("print by area");
        shapes.sort(new SortByArea());
        for (Shape s : shapes) {
            System.out.println(s);
        }
    }

public static void printByDistance(List<Shape> shapes) {
        System.out.println("print by distance");
        shapes.sort(new SortByDistance());
        for (Shape s : shapes) {
            System.out.println(s);
        }
    }
```

Exercise

Design a class named `Student` below, and add its three parameter constructor, getter and setter method of all private fields, and toString() method.

```
public class Student{
    private String group;
    private String name;
    private int grade;
     //todo: add constructor, getter and setter method, toString() method
}
```

Then create a class named `TestSort`, which contains a `List` that contains several students:

```
public class TestSort {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
        students.add(new Student("01","ZhangSan",90));
        students.add(new Student("01","LiMing",95));
        students.add(new Student("03","XiaoLan",89));
        students.add(new Student("02","Wong",99));
        students.add(new Student("02","LiSi",80));
    }
}
```

If we have two requirements, the one is sorted by name while the other is sorted by grade? How to design?

If we make `Student` class to be a Comparable, it can only be sorted by one field, so that in this case, using Comparator would be more flexible.

Design your code and output as follows:

```
sort by name:
Student{group='01', name='LiMing', grade=95}
Student{group='02', name='LiSi', grade=80}
Student{group='02', name='Wong', grade=99}
Student{group='03', name='XiaoLan', grade=89}
Student{group='01', name='ZhangSan', grade=90}
sort by grade:
Student{group='02', name='LiSi', grade=80}
Student{group='03', name='XiaoLan', grade=89}
Student{group='01', name='ZhangSan', grade=90}
Student{group='01', name='LiMing', grade=95}
Student{group='02', name='Wong', grade=99}
```