Object Orientation – Spring 2018 **Assignment 3**

Learning Objectives

After making this exercise you should be able to:

- define new interfaces;
- make classes that implement one or more interfaces;
- use interface objects instead of class objects as method argument and array elements;
- sort arrays of objects using `compareTo<T>` en `Comparable<T>`,
- recognizing more complex input with a `Scanner` (this does **not** imply `patterns`).

Geometric Objects

Geometric objects like circles and rectangles have different properties. Hence, they generally belong to a different class. Apart from these differences geometric objects can also have similar properties and we can apply similar actions to them. An interface is a convenient way to model the common and operations on geometric objects in Java. In this way we can use geometric objects, without bothering whether they are circles or rectangles.

For the geometric objects in this exercise the common operations are:

- four methods that return the left-, right-, bottom- and top-border of the smallest surrounding rectangle as a `double`. For example, a circle that has its centre at the coordinates (1,2) and radius 1, the left-, right-, bottom- and top-border will be 0, 2, 1 and 3 respectively.
- a method that yields the area of the object as a `double`.
- a method to move the object over the given distances dx and dy, both of type `double`.
- the method `compareTo` from the Java `Comparable<T>` interface that compares the area of geometric objects[1].

In this exercise, there are two kinds of objects implementing this interface: circles and rectangles. A circle is defined by the position of its centre and its radius. A rectangle is characterized by its left under corner, its width and height. Each of these objects has a tailor-made `toString` method showing its characteristics.

Your interactive program executes user commands which can create and manipulate geometric objects. Therefore, you need to keep track of an array of geometric objects. The array has some fixed size, e.g. 10. After each command the system lists the current geometric objects.

The commands to be recognized are:

quit stop the program.

show list the geometric objects.

circle x y r adds a circle at (x, y) with radius r if the array is not full.

rectangle x y h w adds a rectangle at (x, y) with height h and width w.

move i dx dy moves the i^{th} object over the specified distance in x and y direction.

remove i remove the i^{th} object.

sort the objects in the array are sorted. This command has one optional argument. The argument **x** indicates sorting on the left most point from small to large. With argument **y** the objects are sorted such that the one with a smaller bottom point preceeds objects that are higher in a drawing. Without argument the objects are sorted on their area from small to large.

[1]See `docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html`.

As always, you should make a clean separation in your program between the input-output handling and the actual logic of your program. Here, this logic is the object store and its manipulations.

Sorting Arrays in Java

Sorting arrays is used very often in programs. The Java API contains reusable and efficient solutions. The class `Arrays`[2], provides the static sorting methods

```
sort(Object[] a)
sort(Object[] a, int fromIndex, int toIndex)
```

These methods sort the elements according to the ordering of the objects using the `Comparable<T>` interface[3]. Use this to sort an array `a` as `Arrays.sort(a)`. This interface contains only the method `int compareTo(T o)`. That means your geometrical object interface should extend this interface, so that the rectangle and circle classes provide an actual implementation for the `int compareTo(T o)` method.

In this method, the `T` is the type of object you want to compare. In Java it is called *generic* argument, which will be covered later in this course. For now replace the `T` with the name of the class or interface you want to use as argument in `Interface<T>`. For instance, when you want to compare objects that implement the `Geometric` interface you have to implement `int compareTo(Geometric o)`. When class `C` implements the comparable interface you write `public class C implements Comparable<C>` as header of the class definition.

The result of `o1.compareTo(o2)` should be `0` when the objects are equal. The expected result is negative when `o1` is smaller than `o2`. Otherwise, the result is positive. Implement this interface to compare geometric objects based on their area.

If you want to sort also using another ordering relation you cannot use the `Comparable<T>` interface again since a class can implement an interface only once. The class `Arrays` offers static sorting methods that use an external object implementing the comparator interface[4] for this purpose. This interface provides the method `int compare(T o1, T o2)` . The integer result obeys the same rules as in the `Comparable<T>` interface.

```
sort(T [] a, Comparator<T> c)
sort(T [] a, int fromIndex, int toIndex, Comparator<T> c)
```

These sorting methods take an additional object argument (the comparator) which is used to compare the objects in the array `a`.

Deliverables

Make the interface for geometric objects and the classes `Circle` and `Rectangle` implementing this interface. Ensure that all geometric objects can be sorted by implementing the `Comparable<T>` interface. Introduce other classes by need to achieve a well designed program, i.e., separating input–output and logic.

It might be helpful to draw a class diagram before you start coding, but it is not required to hand in this diagram. It is convenient to construct the parts of this diagram incrementally and to test the finished parts as early as possible (i.e. long before your entire program in completed).

Do not forget to apply the layout rules and include JavaDoc specifying your names and student numbers.

Hand in all `.java` files from the package via Blackboard. **Deadline Sunday February 25, 23:59**.

[2]See `docs.oracle.com/javase/8/docs/api/java/util/Arrays.html`.
[3]See `docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html`.
[4]See `docs.oracle.com/javase/8/docs/api/java/util/Comparator.html`.