Assignment Geometric Shapes

Object Orientation

Spring 2025

1 Learning Objectives

After making this exercise you should be able to

- Define new interfaces
- Make classes that implement an interface
- Use interfaces instead of classes as method argument and array elements
- Sort arrays of objects using `Comparator<T>`
- Recognize more complex input with a `Scanner`

We will first give information on all the parts of the assignment, after which we will give concrete instructions on what to implement. We recommend you first read the whole document and then start implementing at Section 6.

2 Geometric Objects

Geometric objects like circles and rectangles have some different fields, but also some similar fields and methods. An interface is a convenient way to model the common 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`.

All these methods should be part of an interface called `Geometric`, and circles and rectangles should implement this interface. A circle is defined by the position of its centre and its radius. A rectangle is characterized by its lower left corner, its width and height. Each of these objects has a tailor-made `toString` method showing its characteristics.

3 Filtering

To filter our array of geometric objects, we can iterate over each geometric object, decide whether it should be removed, and if so, remove it. This means we can separate the procedure for deciding whether an object should be removed and the procedure for doing the actual filtering. We can separate this filtering and criterion logic by creating a `GeometricPredicate` interface, which each filtering criterion will implement. This interface should provide a method `boolean predicate(Geometric shape)`. With this interface, we can write a filter method that takes a `GeometricPredicate` as an argument and uses the provided `predicate` method to remove the geometric objects that satisfy the given criterion.

The usage of an interface allows us to implement all kinds of filters without repeating the filtering logic itself. We see a similar thing used in the Java standard library for sorting arrays.

4 Sorting Arrays in Java

Sorting arrays is used very often in programs. The Java API contains reusable and efficient solutions. The class `Arrays`[1] provides the static sorting method

```
Arrays.sort(Object[] a, Comparator<T> c)
```

This method sorts the elements according to the ordering of the objects using the `Comparator<T>` interface[2]. `Comparator` contains only the method `int compare(T o1, T o2)`.

In this method, `T` is the type of object to be compared. In Java it is called a *generic* argument, which will be covered later in this course. For now replace `T` with the name of the class or interface you want to use as argument in `Comparator<T>`. For example, when you want to compare objects that implement the `Geometric` interface you have to create a class that implements the `Comparator<Geometric>` interface, and the method `int compare(Geometric o1, Geometric o2)`.

When you implement `compare(o1, o2)`, it should return `0` when `o1` and `o2` are equal. It should return a negative number when `o1` is smaller than `o2`. Otherwise, it should return a positive number.

5 Interactive Program

Your interactive program should execute user commands which can create and manipulate geometric objects. Therefore, your program needs to keep track of an array of geometric objects. The array should have a fixed size, for example 10. After each command the system lists the current geometric objects.

The commands to be recognized are:

quit - stops the program.
show - lists the geometric objects.
circle x y r - adds a circle at (x, y) with radius r.
rectangle x y w h - adds a rectangle with (x, y) for the lower left corner, with width w and height h.
move i dx dy - moves object with index i over the specified distance in x and y direction.
remove i - deletes object with index i.

[1] See https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/Arrays.html.
[2] See https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/Comparator.html.

filter *c n* - removes all the objects according to criterion *c n*.

- When *c* is “x”, objects will be removed if their left most is smaller than *n*.
- When *c* is “y”, objects will be removed if their bottom point is smaller than *n*.
- When *c* is “a”, objects will be removed if their area is smaller than *n*.

sort *c* - sorts the objects in the array according to criterion *c*. The argument *c* is optional.

- When *c* is “x”, objects should be sorted by their left most point from small to large.
- When *c* is “y”, objects should be sorted by their bottom point from small to large.
- Without argument the objects are sorted on their area from small to large.

As always you should make a clean separation in your code between input-output handling and the actual logic.

6 Your Tasks

1. Create an interface called `Geometric` for geometric objects and classes `Circle` and `Rectangle` implementing this interface. See Section 2 for details.
2. Introduce the `GeometricPredicate` interface and introduce classes implementing this interface for each of the filter criteria. See Section 3 for details.
3. Ensure that all geometric objects can be sorted based on area, left-most point and bottom-most point. Implement classes implementing the `Comparator<T>` interface for each of the sorting criteria. See Section 4 for more details.
4. Introduce other classes as needed to achieve a well designed program that deals with the I/O of running the interactive program. The classes should provide the functionality discussed in Section 5.

The classes `Arrays` and `Comparator` are part of the Java standard library. You have to use them. It is convenient to construct the parts of this diagram incrementally and to test the finished parts as early as possible, before your entire program in completed.

6.1 Submit Your Project

To submit your project, follow these steps.

1. Find the folder that contains your assignment. In Visual Studio Code, you can find this by going to: File → Open Folder. Add the correct folder to a zip file. At the root level, your zip file should contain only one folder, e.g. `assignment-student-start`, which contain the entire project, i.e. the `app` folder and the Gradle files. *Do not submit only the `.java` files or the `src` folder!*
2. **Submit this zip file on Brightspace**. Do not submit individual Java files. Only one person in your group has to submit it. Submit your project before the deadline, which can be found on Brightspace.
3. **Do not submit any other format.** Do not submit .rar or .tar.gz or .7z files. Only zip files are allowed.