Assignment 1: Hit Detection

2IP90, Programming - Q1 (2023)

1 | Hit Detection

Problem Description

We say that a point hits a shape if that point is inside the shape. If a point is on the border of the shape, that also counts as a hit. Checking if a point hits a shape is called *hit detection* or *hit-testing*.
Given two circles and a point in a 2D Cartesian coordinate system, write a program to detect which of these circles is hit by that point.
Your program reads eight floating point numbers on input. The first two numbers represent the center of the first circle, first its x-coordinate and then its y-coordinate. The third number represents this circle's radius. The next three numbers represent the second circle in exactly the same fashion. Finally, the last two numbers represent the point. Again, its x-coordinate is followed by its y-coordinate.
The program behaves as follows:

- When the point does not hit any of the two circles, the program outputs:
  `The point does not hit either circle`
- When the point hits just a single circle, depending on which of the circles, the program outputs:
  - `The point hits the first circle`
  - `The point hits the second circle`
- When the point hits both circles, the program outputs:
  `The point hits both circles`
- Finally, when either circle has a negative radius, the program outputs:
  `input error`

The output should be exactly as described above. Deviations, like asking `Please gimme input`, or `Warning, negative radius`, aren't allowed.

Example Run

```
0 0 3.1 0.25 0.13 1 2.1 2.03
```
The point hits the first circle

Startup

1. Create a new directory `hitting_shapes` and open it in VSCode.
2. Download files `minimal_test_set.txt` and `HitDetection.java` and put them in this folder.
3. Open both files and fill in your names and student IDs in each file.

Understanding the Problem

Minimal Test Set

4. Before you start programming, create a minimal test set in the file `minimal_test_set.txt`. Add at least seven different test cases to cover the behavior of this program. We've already put the example run as the first test case in the file. You have to fill in the other six test cases. Motivate each test case. For each test case, give the exact input and the exact output you expect. Don't add multiple test cases that test the same behavior. For example, adding a test case with input `0 0 100 0 0 1 50 50` alongside the given example run test case would be superfluous.

Manual

5. Once you've created the minimal test set, you should have a good understanding of the program's behavior. Use that understanding to write a short, but complete user manual for the `HitDetection` program. Use your own words. In the file `HitDetection.java`, include documentation.

Programming

6. Write the program.
   In the file `HitDetection.java`, include the hit detection logic.

Hints

- Use Boolean variables.
- Adhere to our coding standard. In particular, use descriptive variable names that are distinctive. Variable names like `x1` don't provide the reader of your source code with enough information. After all, going by this name alone, how would the reader of your program know to which shape or point this x-coordinate belongs?

7. Test your program using the minimal test set you created.
   Note: Due to the way Java handles your operating system's locale settings, you might need to use a comma instead of a period as the decimal point when inputting floating point numbers on the console.
8. If you're confident that your program is correct, submit both files to Canvas.
9. Check Momotor Output. If Momotor reports any issues, fix them and resubmit. Ask yourself why your tests didn't catch these issues: Do you need to update your minimal test set or manual as well? You can submit to Canvas as often as you want before the deadline.