Ex 0: Circle and Point

Basic Information

- **Deadline:** 27 August 2024, Tuesday, 23:59 SGT
- **Difficulty:** ★

Prerequisite

- Familiar with the CS2030S lab guidelines.
- Able to access the CS2030S programming environment via ssh.
- Setup vim and completed basic `vim` lessons.
- Link your PE node account to GitHub.

Files

The link to accept the exercise is posted on Canvas and is not available publicly. You sould not share the link with other people. After accepting the exercise, run the following command

```
~cs2030s/get ex0
```

to retrieve the skeleton code. You should see the following files:

1. **Skeleton Java Files:**
   - `Point.java`: Skeleton file for `Point` class.
   - `RandomPoint.java`: Skeleton file for `RandomPoint` class.
   - `Circle.java`: Skeleton file for `Circle` class.
   - `Ex0.java`: The main program.
2. **Input/Output Files:**
   - `inputs/Ex0.k.in` for the input files for different values of `k`.
   - `outputs/Ex0.k.out` for the output files for different values of `k`.
3. **Bash Script:**
   - `test.sh`: Testing `Ex0` if it estimates π correctly by comparing the output when running `Ex0` on `inputs/Ex0.k.in` with the expected output in `outputs/Ex0.k.out`.
4. **Unit Tests:** `Test1.java` to `Test3.java` to test individual classes for expected behavior.

Overview

The Monte Carlo method for estimating the value of π is as follows. We have a square of width $2r$, and within it, a circle with a radius of r. We randomly generate k points within the square. We count how many points fall within the circle. Suppose n points out of k fall within the circle. Since the area of the square is $4r^2$ and the area of the circle is πr^2, the ratio between them is $\pi/4$. The ratio n/k should therefore be $\pi/4$, and π can be estimated as $4n/k$.

Tasks

A skeleton code has been given. Your task is to complete the implementation of the classes `Point`, `RandomPoint`, `Circle`, and `ex0`, according to the OO principles that were taught: *abstraction*, *encapsulation*, *information hiding*, *inheritance*, *tell-don't-ask*.

Task 1: `Point` Class

Fill in the class `Point` with the constructor and the necessary fields. Add a `toString` method so that a string representation as shown in the examples below is returned.

For instance,

```
new Point(0, 0).toString();
```

should return the string:

```
(0.0, 0.0)
```

You will need to come back to this class and add other methods later. For now, check that your constructor and `toString` methods are correct.

Some simple tests are provided in the file `Test1.java`. Note that these test cases are not exhaustive and you are encouraged to test your `Point` class on your own. Proceed to the next class if you are convinced your `Point` class is correct.

```
user@pe111:~/ex0-github-username$ javac Test1.java
user@pe111:~/ex0-github-username$ java Test1
Point: new at (0, 0).. ok
Point: new at (-3.14, 1.59).. ok
```

Re-Compiling Files that Changed

As an aside, note that we do not need to explicitly compile `Point.java`. Since `Test1.java` refers to the `Point` class, `javac` is smart enough to compile `Point.java` if `Point.class` is not found, or recompile `Point.java` if it is newer than `Point.class`.

However, sometimes Java can get confused (*e.g., if some class files are removed by hand*). It is recommended that students recompile every file that has been edited explicitly, instead of letting Java figure out which file should be recompiled.

A simple, brute-force, way to re-compile all the Java files:

```
user@pe111:~/ex0-github-username$ javac *.java
```

This only works when all the Java files can be compiled without error, of course **including** files that are not being used.

Task #2: `Circle` Class

Most of the `Circle` class has been written for you. You need to complete the method `contains`. The method checks if a given point is contained in the calling `Circle` object. To complete this method according to the tell-don't-ask principle, you will need to add a method in the `Point` class.

Some simple tests are provided in the file `Test2.java`. These test cases are not exhaustive and you are encouraged to test your `Circle` class extensively.

```
user@pe111:~/ex0-github-username$ javac Test2.java
user@pe111:~/ex0-github-username$ java Test2
Circle: new at (0, 0) with radius 4).. ok
Circle centered at (0, 0) with radius 4 contains (0, 0).. ok
Circle centered at (0, 0) with radius 4 does not contain (4, 3).. ok
Circle centered at (0, 0) with radius 4 does not contain (3, 4).. ok
Circle centered at (2, -3) with radius 0.5 contains (1.8, -3.1).. ok
Circle centered at (2, -3) with radius 0.5 does not contain (1.8, -4).. ok
```

Task 3: `RandomPoint` Class

To estimate π using the method above, we need to use a random number generation. A random number generator is an entity that spews up one random number after another. We, however, cannot generate a truly random number algorithmically. We can only generate a pseudo-random number. A pseudo-random number generator can be initialized with a seed. A pseudo-random number generator, when initialized with the same seed, always produces the same sequence of (*seemingly random*) numbers.

Java provides a class `java.util.Random` that encapsulates a pseudo-random number generator. We can create a random number generator with a seed of 1 as follows.

```
Random rng = new Random(1);
```

We can then call `rng.nextDouble()` repeatedly to generate (*pseudo*-)random numbers between 0 and 1.

Impact of Seed

If we re-initialized `rng` again with another random number generator, with a different seed as shown below

```
rng = new Random(2);
```

then calling `rng.nextDouble()` produces a different sequence. But if we re-initialized `rng` with the seed of 1 again as shown below

```
rng = new Random(1);
```

then `rng.nextDouble()` will produce the same sequence as when the seed was 1.

(*Don't take our word for it. Try out the above using* `jshell`)

Using a fixed seed is important for testing since the execution of the program will be deterministic, even when random numbers are involved.

`RandomPoint` is a subclass of `Point` that represents a randomly generated point. The random number generator that generates a random point has a default seed of 1. There is a public method `setSeed()` that we can use to update the seed. Here is how it can be used:

To generate a new point,

```
Point p = new RandomPoint(minX, maxX, minY, maxY);
```

`minX`, `minY`, `maxX`, `maxY` represent the minimum and maximum possible x and y values respectively, for each randomly generated point.

To set the random seed,

```
RandomPoint.setSeed(10);
```

Tips

What are the fields and methods that should be associated with the class `RandomPoint` instead of an instance of `RandomPoint`?

Some simple tests are provided in the file `Test3.java`. These test cases are not exhaustive and you are encouraged to test your `RandomPoint` class extensively.

```
user@pe111:~/ex0-github-username$ javac Test3.java
user@pe111:~/ex0-github-username$ java Test3
RandomPoint: is a subtype of Point.. ok
RandomPoint: generate a new point with default seed.. ok
RandomPoint: generate a new point with seed 10.. ok
RandomPoint: generate a new point with the same seed.. ok
RandomPoint: reset seed to 10 and generate a new point.. ok
```

Task #4: Estimating Pi using Monte Carlo Method

Ex0

`Ex0` is the main program to solve the problem above. The `main` method is provided. It includes the method to read in the number of points and the seed from the standard input and to print the estimated pi value.

The method `estimatePi` is incomplete. Determine how you should declare `estimatePi`, then complete the body by generating random points and count how many fall under the given circle.

Use a circle centred at (0.5, 0.5) with radius 0.5 for this purpose. Use `long` and `double` within `estimatePi` for computation to ensure that you have the right precision.

Tips

In Java and many other languages, using `/` on two integers result in an integer division. Make sure one of the operand of `/` is a floating point number if you intend to use `/` for floating point division.

To compile `Ex0`, run

```
user@pe111:~/ex0-github-username$ javac Ex0.java
```

To run `Ex0` and enter the input manually, run

```
user@pe111:~/ex0-github-username$ java Ex0
```

The program will pause, waiting for inputs from keyboards. Enter two numbers. The first is the number of points. The second is the seed.

To avoid repeatedly entering the same inputs to test, you can enter the two numbers into a text file, say, `TEST`, and then run

```
user@pe111:~/ex0-github-username$ java Ex0 < TEST
```

If you are not sure what `<` means, read more input/output direction here

Sample inputs and outputs have been provided and can be found under the `inputs` and `outputs` directory.

To test your implementation of `Ex0`, automatically against the test data given in `inputs` and `outputs`,

```
user@pe111:~/ex0-github-username$ ./test.sh Ex0
```

Common Mistakes

1. Running a Java File

Symptom

You encounter the following error below.

```
username@pe111:~/ex0-github-username$ java Test1.java
Exception in thread "main" java.lang.IllegalAccessError: failed
to access class CS2030STest from class Test1 (CS2030STest is
in unnamed module of loader 'app'; Test1 is in unnamed module
of loader com.sun.tools.javac.launcher.Main$MemoryClassLoader
@782663d3)
        at Test1.main(Test1.java:5)
```

Why?

Java code needs to be compiled before you run. So the correct sequence is to first compile using `javac` as follows

```
username@pe111:~/ex0-github-username$ javac Test1.java
```

and then run using `java` as follows

```
username@pe111:~/ex0-github-username$ java Test1
```

2. Changes to Code Not Taking Effect

Symptom

You have made changes to your code, but the output or behavior of your program remained unchanged.

Why?

Java code needs to be compiled before you run. You need to compile the files that you have changed first before they can take effect.

After you have made changes to multiple files, the easiest way to recompile everything is:

```
username@pe111:~/ex0-github-username$ javac *.java
```

where `*` is a wildcard that pattern-match any string.

3. Constructor Point Cannot be Applied

Symptom

You encounter the following error below.

```
RandomPoint.java:12: error: constructor Point in class Point
cannot be applied to given types;
```

Why?

The constructor for the subclass should invoke the constructor of the superclass. See the example given in the notes on `ColoredCircle` and `Circle`.

If the constructor of the superclass is not called explicitly, Java tries to call the default constructor of the superclass without any argument. If no such constructor is defined, the error above is generated.

Also note that the call to `super(..)` should be the first line inside the constructor of the subclass. That means, if your call to `super(..)` requires a computed value, the value has to be computed *inline* as the arguments to `super(..)`.

WOOPSIE

Introducing the "Wonderful OOP SanItizEr" also called as WOOPSIE. This is a static analysis tool that will help check some general OOP property of your program. Its outputs are potentially a series of check starting with `[filename]`. These are suggestions to help you not to lose mark.

Please note that the checks that can be performed by WOOPSIE are merely suggestions. In particular, we reduced the precision because a checker that is too eager will give too many checks. Many of these checks will be a false positive. Such checker are not a useful checker and you may not even want to use that.

Additionally, the coverage of the checker may be limited. You can, after all, try to fool it by making your code unnecessarily complicated. We try to hit a sweet spot where if you follow the lab guide, WOOPSIE may give the most optimal benefit with minimal false positives.

To run WOOPSIE on all `.java` file in your current directory, type the following:

```
user@pe111:~/ex0-github-username$ python3 ~cs2030s/WOOPSIE.py
```

To run WOOPSIE on a specific file(s), you can specify the files that you wish to be checked. For instance,

```
user@pe111:~/ex0-github-username$ python3 ~cs2030s/WOOPSIE.py Point.java
Circle.java
```

WOOPSIE may run even if your program cannot compile. The requirement is simply that we it can parse your program. So if the compilation error is due to type issues, WOOPSIE can still check for some common errors. Some messages that you may see includes:

```
[Point.java]: "x" not private
  HINT: can it be made private?
```

javalang

If you encounter a problem related to javalang library, you need to install this first by running the following command

```
user@pe111:~/ex0-github-username$ pip install javalang
```

CS2030STest

If you are running WOOPSIE on all java file, you may see the following message

```
----- WOOPSIE -----
[CS2030STest.java]: "CS2030STest.ANSI_RESET" not private
HINT: can it be made private?
[CS2030STest.java]: "CS2030STest.ANSI_RED" not private
HINT: can it be made private?
[CS2030STest.java]: "CS2030STest.ANSI_GREEN" not private
HINT: can it be made private?
-------------------
```

You do not have to worry about the test files. Focus only on files you edited.