Lab 0: Circle and Point

- Deadline: 24 August 2022, Wednesday, 23:59 SST
- Marks: 0

> **Important**
>
> The deadline shown on CodeCrunch will be different as we allow late submissions.

Prerequisite:

- Familiar with the CS2030S lab guidelines
- Able to access the Sunfire student account (stu.comp.nus.edu.sg) via ssh
- Completed basic `vim` lessons

Estimating Pi using Monte Carlo Method

The Monte Carlo method for estimating the value of pi 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 pi r^2, the ratio between them is pi/4. The ratio n/k should therefore be pi/4, and pi can be estimated as 4n/k.

Background: Random Number Generator

To estimate pi 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:

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

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

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

Files Provided

Inside the directory `Lab0`, you will see the following files:

- Skeleton Java files: `Point.java`, `RandomPoint.java`, `Circle.java`, `Lab0.java`
- Inputs and outputs for `Lab0`: `inputs/Lab0.k.in` and `outputs/Lab0.k.out` for different values of k.
- Bash script: `test.sh` for testing `Lab0` if it estimates pi correctly, by comparing the output when running `Lab0` on `inputs/Lab0.k.in` to the expected output in `outputs/Lab0.k.out`
- Unit tests for Java classes: `Test1.java` to `Test3.java`. These files test individual classes to check if they have the expected behavior.

Your Task

A skeleton code has been given. Your task is to complete the implementation of the classes `Point`, `RandomPoint`, `Circle`, and `Lab0`, according to the OO principles that were taught: abstraction, encapsulation, information hiding, inheritance, tell-don't-ask.

The `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.

```
cs2030s@stu1:~Labs/Lab0$ javac Test1.java
cs2030s@stu1:~Labs/Lab0$ java Test1
Point: new at (0, 0).. ok
Point: new at (-3.14, 1.59).. ok
```

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`.

The `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.

```
cs2030s@stu1:~Labs/Lab0$ javac Test2.java
cs2030s@stu1:~Labs/Lab0$ 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
```

The `RandomPoint` class

`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);
```

Tip: 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.

```
cs2030s@stu1:~Labs/Lab0$ javac Test3.java
cs2030s@stu1:~Labs/Lab0$ 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
```

Lab0

`Lab0` 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 centered 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.

Tip: In Java, 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 run `Lab0` and enter the input manually, run

```
java Lab0
```

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

You can enter the two numbers into a text file, say, `TEST`, and then run

```
java Lab0 < TEST
```

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

To test your implementation of `Lab0`, automatically against the test data given in `inputs` and `outputs`,

```
./test.sh Lab0
```

Submission

Upload the following files to CodeCrunch:

1. `Circle.java`
2. `Lab0.java`
3. `Point.java`
4. `RandomPoint.java`

> **Important**
>
> The grade initially shown on CodeCrunch is an autograding for correctness. It may change after manual grading.