NUS | Computing
National University of Singapore

CodeCrunch

Home | My Courses | Browse Tutorials | Browse Tasks | Search | My Submissions | Logout | Logged in as: **e1120988**

CS2030 (2410) Exercise #1: Maximum Disc Coverage

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

Maximum Disc Coverage

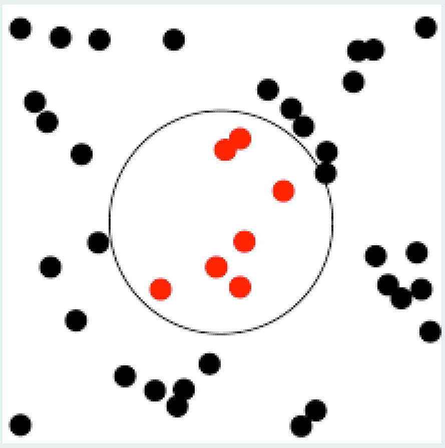
Topic Coverage

- Basic Java syntax and semantics
- Object-oriented principles: abstraction and encapsulation
- Java Version 21 API Specification
- CS2030 Java Style Guide

Problem Description

Given a set of points on a 2D plane, we would like to place a unit disc (i.e., a circle of radius `1.0`) so that it covers as many points as possible.

Note that it is not necessary that one of the points must be at the centre of the disc.

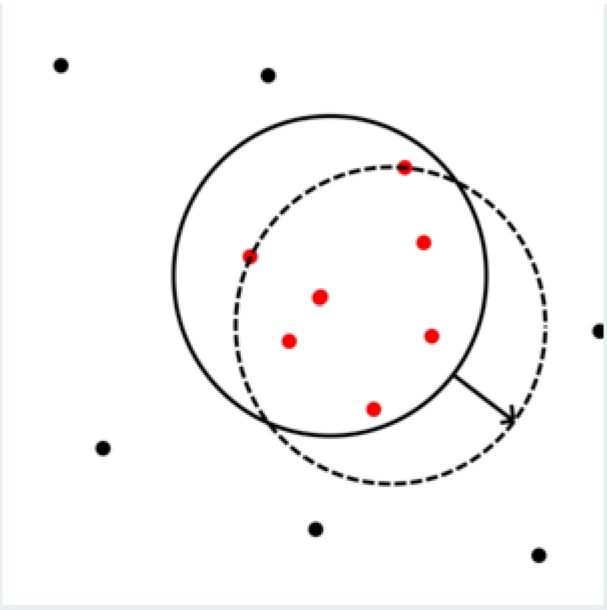


What is the maximum number of points that we can cover with the disc at any one time?

Here are some interesting observations:

- A circle that covers the maximum number of points must pass through at least two points.

  Suppose we found a circle containing the maximum number of points (at least two). We can translate this circle such that two points lie on its perimeter, while the number of points it contains remain unchanged. If translating this circle increases the coverage, this contradicts the premise that the the circle contains the maximum number of points in the first place.



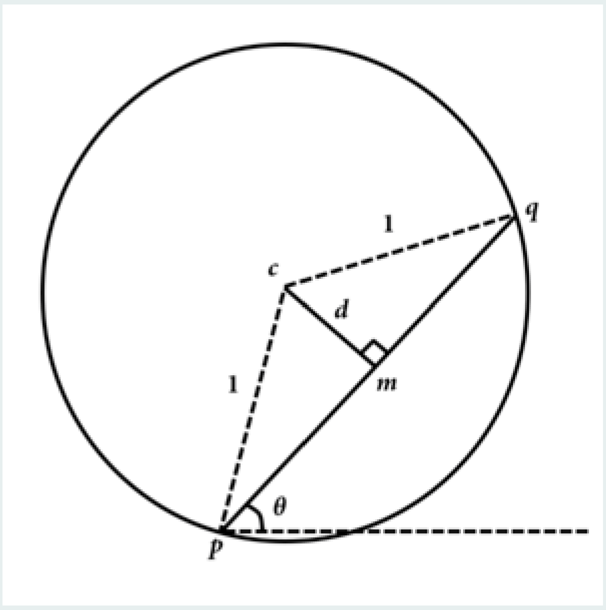
- For every pair of points that are no more than distance `2.0` away from each other, there will be at most two unit circles with their perimeters passing through the two points.

The algorithm

The high level algorithm for finding the maximum disc coverage is as follows:

1. For every pair of distinct points of distance no more than `2.0`, create a unit circle such that the pair of points lie on that circle's perimeter.
2. For each unit circle created in the previous step, count the number of points that it covers.
3. Return the maximum coverage.

The most interesting part of this task lies in creating the unit circle with two given points, such that they lie on the perimeter of the circle. Some elementary trigonometry can guide us as to how to do this:



Given points `p` and `q`, we can

1. Find the midpoint of `p` and `q`, denoted as point `m`.
2. Find the angle of the line `pq` with respect to the `x` axis, denoted as `θ`
3. Find the length of line `mc`, denoted as `d`, which can be obtained using the Pythagorean Theorem
4. Move point `m` by distance `d`, at the angle of line `mc` with respect to the `x`-axis, which is `θ + π/2`. The new location of point `m` is the centre of the circle, `c`.

In this exercise, we only consider "left" unit circles with respect to traversing from point `p` to point `q`.

The Task

Given a list of points, go through every pair of points, and for each circle that passes through them, count how many points are covered by each circle. Finally, return the maximum count.

You may assume that there are always at least two points with a positive distance less than `2.0` between them.

A working program is provided for you comprising of `Point` class, `Circle` class and `Main.java`.

Here is a sample run using a list of two points:

```
$ jshell
|  Welcome to JShell -- Version 21.0.4
|  For an introduction type: /help intro

jshell> /open Point.java

jshell> /open Circle.java

jshell> /open Main.java

jshell> List<Point> points = List.of(new Point(0.0, 0.0), new Point(1.0, 0.0))
points ==> [Point@..., Point@...]

jshell> findMaxDiscCoverage(points)
$.. ==> 2

jshell> /exit
|  Goodbye
```

By applying the object-oriented principles of *abstraction* and *encapsulation*, re-design the program as a object-oriented solution to the problem.

You may choose to either retain the imperative style of the solution using loops, or re-implement them declaratively using Java streams. Take it as an opportunity to find a programming style that suits you.

Level 1

The `Point` class represents a point object with each pair of x- and y- coordinates.

Define a `toString` method to output each point. The output of each `double` value, say `d`, is to be formatted with `String.format("%.3f", d);`

In addition, define the `distanceTo` method that takes in another point, and returns the Euclidean distance between the point and itself.

```
jshell> /open Point.java

jshell> new Point(0.0, 1.0)
$.. ==> point (0.000, 1.000)

shell> new Point(0.0, 1.0).distanceTo(new Point(1.0, 1.0))
$.. ==> 1.0

jshell> /exit
```

Using an editor, modify and save the `Point` class in `Point.java`, and upload to CodeCrunch for testing.

Level 2

Find the mid-point between two consecutive points `p` and `q`, as well as to find the angle (in radians) of line `pq` using the `atan` or `atan2` method of the `Math` class. You may refer to the Java API for details.

Define the `midPoint` and `angleTo` methods in the `Point` class. You may define other `private` helper methods to further abstract away lower-level functionalities.

```
jshell> /open Point.java

jshell> new Point(0.0, 0.0).midPoint(new Point(1.0, 1.0))
$.. ==> point (0.500, 0.500)

jshell> new Point(0.0, 0.0).angleTo(new Point(1.0, 1.0))
$.. ==> 0.7853981633974483

jshell> new Point(0, 0).angleTo(new Point(-1.0, -1.0))
$.. ==> -2.356194490192345

jshell> Point p = new Point(1.0, 1.0)
```

```
p ==> point (1.000, 1.000)

jshell> p.midPoint(new Point(2.0,2.0))
$.. ==> point (1.500, 1.500)

jshell> p
p ==> point (1.000, 1.000)

jshell> /exit
```

Inconsistencies between your output and the actual output involving `-0.000` and `0.000` can be ignored.

Level 3

Now we need to move a point by an angle `θ` and a distance `d`. In the spirit of immutability, rather than moving the original point, a new point is returned instead.

Specifically, a point at `(x, y)` that is moved at an angle `θ` and distance `d`, would result in the new position having the coordinates `(x + d cosθ, y + d sinθ)`

```
jshell> /open Point.java

jshell> Point p = new Point(0.0, 0.0)
p ==> point (0.000, 0.000)

jshell> p.moveTo(Math.PI / 2, 1.0)
$.. ==> point (0.000, 1.000)

jshell> p
p ==> point (0.000, 0.000)

jshell> /exit
```

Level 4

Define the `Circle` class so that circles can be constructed with a given centre and radius. You will also need to include an appropriate `toString` method. You may assume that all circles created have positive radii.

```
jshell> /open Point.java

jshell> /open Circle.java

jshell> new Circle(new Point(0.0, 0.0), 1.0)
$.. ==> circle of radius 1.0 centred at point (0.000, 0.000)

jshell> new Circle(new Point(1.0, 1.0), 2.0)
$.. ==> circle of radius 2.0 centred at point (1.000, 1.000)
```

Modify the `createUnitCircle` method in `Main.java` so as to make use of the `distanceTo`, `midPoint`, `angleTo` and `moveTo` methods of the `Point` class.

It is worth noting that if the distance between points `p` and `q` is larger than `2.0` units, then there is no unit circle whose perimeter coincides with the points. In this level, you may assume that the two points will always form a unit circle.

```
jshell> /open Main.java

jshell> createUnitCircle(new Point(0.0, 0.0), new Point(1.0, 0.0))
$.. ==> circle of radius 1.0 centred at point (0.500, 0.866)

jshell> createUnitCircle(new Point(0.0, 0.0), new Point(2.0, 0.0))
$.. ==> circle of radius 1.0 centred at point (1.000, 0.000)

jshell> /exit
```

Level 5

You are now ready to find the maximum unit disc coverage.

Within `Main.java`, modify the method `findMaxDiscCoverage` that takes in a list of `Point` objects and finds the maximum coverage among them. You may make use of other helper methods.

Keep in mind that if the distance between points p and q is larger than 2.0 units, then there is no unit circle whose perimeter coincides with the points.

```
jshell> /open Point.java

jshell> /open Circle.java

jshell> /open Main.java

jshell> List<Point> points = List.of(new Point(0.0, 0.0), new Point(1.0, 0.0))
points ==> [point (0.000, 0.000), point (1.000, 0.000)]

jshell> findMaxDiscCoverage(points)
$.. ==> 2

jshell> points = List.of(new Point(0.0, -1.0), new Point(1.0, 0.0),
   ...> new Point(0.0, 1.0), new Point(-1.0, 0.0))
points ==> [point (0.000, -1.000), point (1.000, 0.000), poi ... 0), point (-1.000, 0.000)]

jshell> findMaxDiscCoverage(points)
$.. ==> 4

jshell> /exit
```

Submission (Course)

Select course: CS2030 (2024/2025 Sem 1) - Programming Methodology II

Your Files:

BROWSE

SUBMIT (only .java, .c, .cpp, .h, .jsh, and .py extensions allowed)

To submit multiple files, click on the Browse button, then select one or more files. The selected file(s) will be added to the upload queue. You can repeat this step to add more files. Check that you have all the files needed for your submission. Then click on the Submit button to upload your submission.