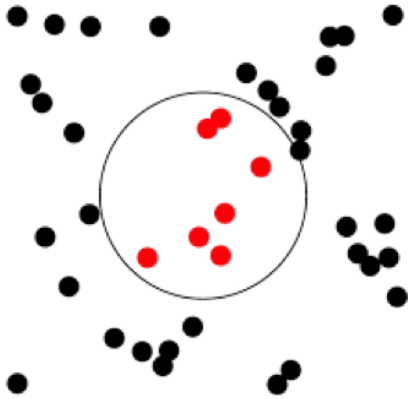
Maximum Disc Coverage

Topic Coverage

- Basic Java syntax and semantics
- Object-oriented principles: abstraction and encapsulation
- Java Version 11 API Specification
- CS2030 Java Style Guide

Problem Description

In this problem, you are given a set of points on a 2D plane. We want to place a unit disc (i.e., a circle of radius 1) so that it covers as many points as possible. *Note that this is different from previous exercises where one of the points must be at the centre of the disc.*

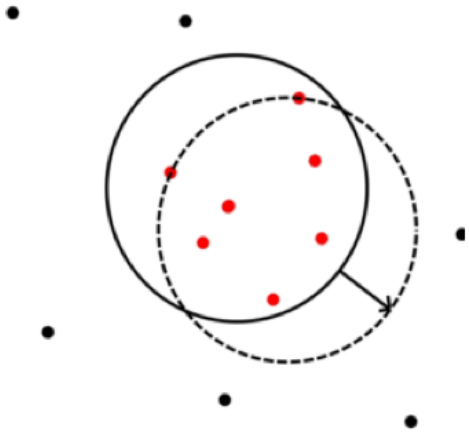


What is the maximum number of points that we can cover with the disc at any one time?

To help you along, here are some 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.



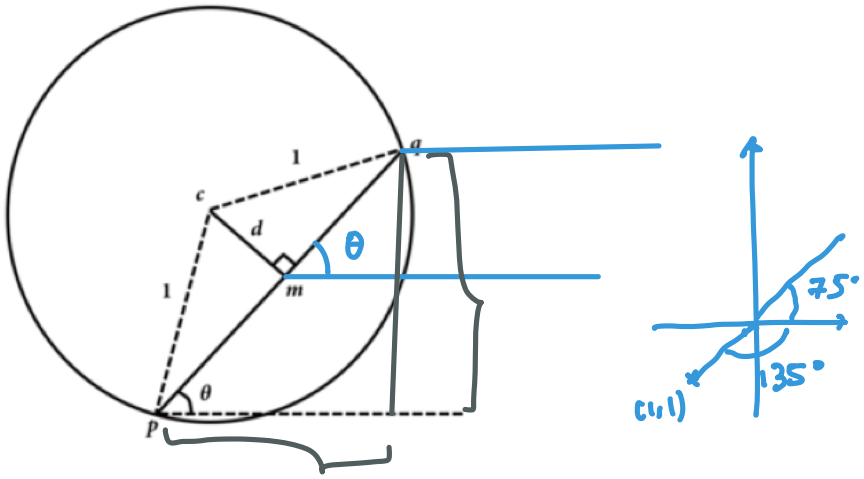
- For every pair of points that are no more than distance `2` away from each other, there will be at most two unit circles with their perimeters passing through the two points.

The algorithm

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

1. For every pair of distinct points of distance no more than `2`, 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`.

The Task

Given a set 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` between them.

This task is divided into five levels. You may submit as soon as you complete one level. There is no restriction on the number of attempts.

In the spirit of incremental coding and testing, you will need to complete ALL levels to get full credit for this lab.

Level 1

Represent a Point

The first thing we can do is to group `x` and `y` coordinates into a single data structure or class.

Design a class `Point` to represent a point object with each pair of `x`- and `y`- coordinates.

```
class Point {
    // include the properties of a Point

    // complete the constructor
    Point(double x, double y) {
    }

    // include a toString method
}
```

Define an overriding `toString` method to output each point. In addition, the output of each `double` value, say `d`, is to be `String.format("%.3f", d);`

```
jshell> /open Point.java

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

jshell> /exit
```

Level 2

Find the mid-point and angle of line `pq`

Find the mid-point between two consecutive points `p` and `q`. In addition, find the angle (in radians) of line `pq` using the `atan` or `atan2` Math function (refer to the Java API specifications).

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

```
class Point {
    ...

    Point midPoint(Point q) {
        ...
    }

    double angleTo(Point q) {
        ...
    }
}
```

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

```
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, -1))
$.. ==> -2.356194490192345

jshell> Point p = new Point(1, 1)
p ==> point (1.000, 1.000)

jshell> p.midPoint(new Point(2,2))
$.. ==> point (1.500, 1.500)

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

jshell> /exit
```

Level 3

Moving the point

Now we need to allow points to be "moved", so that we can move point `m` to `c`. However, in the spirit of immutability, the original point is not moved. Instead, a new point is returned, which reflects the new position of the original point if it has been moved by angle θ and distance `d`.

Hint: if a point, say `m`, is at `(x, y)`, then moving `m` at an angle θ and distance `d`, would result in the new position having the coordinates `(x + d cosθ, y + d sinθ)`

```
class Point {
    ...

    Point moveTo(double theta, double d) {
        ...
        return new Point...;
    }
}
```

```
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

Creating the Circle

Define the `Circle` class to allow for the creation of circle objects with a given centre and radius. You may assume that all circles created are valid.

```
jshell> /open Point.java

jshell> /open Circle.java

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

jshell> new Circle(new Point(1, 1), 2)
$.. ==> circle of radius 2.0 centered at point (1.000, 1.000)
```

We are now ready to construct unit circles whose perimeter passes through two given points.

By using the mid-point and angle of line `pq`, move the mid-point to the centre of the circle of radius `r` whose perimeter coincides with points `p` and `q`. You will need to work out the respective angle and distance values.

Create a jshell script `maxDiscCoverage.jsh` and define a `createUnitCircle` method that takes in two points `p` and `q` and returns the unit circle created.

```
Circle createUnitCircle(Point p, Point q) {
    ...
    return new Circle...;
}
```

Notice that if the distance between points `p` and `q` is larger than `2` units, then there is no unit circle whose perimeter coincides with the points. In this level, you may assume that the two points will form a unit circle.

```
jshell> /open maxDiscCoverage.jsh

jshell> createUnitCircle(new Point(0, 0), new Point(1, 0))
$.. ==> circle of radius 1.0 centered at point (0.500, 0.866)

jshell> createUnitCircle(new Point(0, 0), new Point(2, 0))
$.. ==> circle of radius 1.0 centered at point (1.000, 0.000)

jshell> /exit
```

Level 5

Maximum Disc Coverage

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

Within `maxDiscCoverage.jsh`, further define a method `findMaxDiscCoverage` that takes in an array of `Point` objects and finds the maximum coverage among them. You may make use of other helper methods.

```
Circle createUnitCircle(Point p, Point q) {
    ...
}

int findMaxDiscCoverage(Point[] points) {
    int maxDiscCoverage = 0;

    for (int i = 0; i < points.length - 1; i++) {
        for (int j = i + 1; j < points.length; j++) {
            // find coverage with (points[i], points[j])
            ...
        }
    }
    return maxDiscCoverage;
}
```

Once again, keep in mind that if the distance between points `p` and `q` is larger than `2` units, then there is no unit circle whose perimeter coincides with the points.

```
jshell> /open Point.java

jshell> /open Circle.java

jshell> /open maxDiscCoverage.jsh

jshell> Point[] points = new Point[]{new Point(0, 0), new Point(1, 0)}
points ==> Point[2] { point (0.000, 0.000), point (1.000, 0.000) }

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

jshell> points = new Point[]{new Point(0, -1), new Point(1, 0),
   ...> new Point(0, 1), new Point(-1, 0)}
points ==> Point[4] { point (0.000, -1.000), point (1.000, 0 ... ), point (-1.000, 0.000) }

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

jshell> /exit
```