Lab 1: Comments

Here are some common mistakes made, in decreasing order of seriousness.

Most Serious Offences

Design: Breaking abstraction barrier

Some students took the liberty to change the access modifier of instance fields from `private` to `public`.

In doing so, you are violating the fundamental principles of object oriented programming and demonstrating that you do not understand the concept and importance of abstraction barriers and encapsulation.

Design: Getter / Setter

If you use getter or setter in your code when you could have easily move the associated tasks to within the encapsulation, you score a major minus in your lab. Again, using getter and setter to expose the internals is only slightly better than just changing the members to `public`.

There are cases where the use of getters and setters are justified. For instance, a couple of you use complex equations to determine the center and it involves x and y, instead of using `move`, `angleTo`, and `distanceTo`. One could argue that these solutions heavily rely on the knowledge of x and y.

For all other uses of x and y in the `Circle` constructor, such as to calculate gradient, to copy a point, etc, they can all be delegated to the `Point` class.

Remember, we want the `Point` class to be the only one who knows the internal representation of a point: whether Euclidean coords or polar coords are used.

E.g.,

```java
Point p = new Point(q.getX(), q.getY());
```

can be

```java
Point p = q.copy();
```

or

```java
double gradient = (p.getY()-q.getY())/(p.getX()-q.getX())
```

can be

```java
double gradient = p.gradientTo(q);
```

Design: Checking for Circle validity

Along the same line of mistakes, some of you do this in `MaxDiscCover.java`:

```java
    if (Double.isNaN(circle.getRadius()) { .. }
```

or

```java
    if (circle.radiusIsNaN()) { .. }
```

Suppose now we change an invalid circle to one that has `-1` radius, or one that has a boolean `isValid` flag, then this code wouldn't work or has to be renamed.

A better way is for `Circle` to decide itself whether it is valid or not, by providing a method `isValid()`. Then, `MaxDiscCover.java` can just call

```java
    if (circle.isValid()) { .. }
```

If Circle decides to change its implementation later, it is none of `MaxDiscCover` 's business.

Correctness: Object References

A few students did the following in `Circle` constructor:

```java
public Circle(Point p, Point q, double r, boolean centerOnLeft) {
    Point m = p;
    :
    m.move(..)
    :
}
```

Remember from the figures drawn in class that `m` and `p` are just references to objects. Now, you are pointing `m` and `p` to the same object. When you move `m` , `p` is moved as well, and as a result, one input point from the `points[]` array gets moved.

Performance: Double Loops

Some of you did this in `solve()` :

```java
for (Point p: points) {
    for (Point q: points) {
        :
    }
}
```

What happened is that, now, you loop through each pair of points twice, (first with, say, `p` as `points[0]` , `q` as `points[1]` , and later `p` as `points[1]` , `q` as `points[0]`).

This also meant that for each point, you tried to construct a circle through two copies of itself, a wasted effort since the resulting circle is invalid.

Here's a better way to iterate through all possible pairs of points:

```java
for (int i = 0; i < points.length; i++) {
    for (int j = i + 1; j < points.length; j++) {
        Point p = points[i];
        Point q = points[j];
        :
    }
}
```

Style: Indentation

It is important to keep your code properly and consistently indented. If you use a source code editor like `vim` (with `autoindent` on and `smartindent` on), then there is no reason for you to have crooked indentation.

My favourite example of bad indentation bug: Apple `goto fail` bug [https://www.synopsys.com/blogs/software-security/understanding-apple-goto-fail-vulnerability-2/]

Style: Naming Convention

Java has a certain naming convention for class names, variable names, method names, constant variables, etc. Please follow them. At the beginning of the lab, we ask all of those who attended to follow, but there are still some who does not.

From Lab 3 onwards, we will enforce indentation and naming style.

Less Serious Mistakes

Style: `this`

When referring to the fields of the current object, I prefer to use `this` reference. It is redundant, but makes your program much less bug prone and make it more explicit which variable you are referring to.

Let's suppose I give you a method of about 100 lines and 10 parameters and variables to read, one of the lines said:

```
    center = p;
```

How can you be absolutely sure that this line is updating a member called `center` of the object?

To be sure, you have to scan through 100 lines to double check that there is no local variable declared with the same name, and there is no method argument with the same name.

Now, let's suppose I give you a method of about 100 lines and 10 parameters and variables to read, one of the lines said:

```
    this.center = p;
```

How can you be absolutely sure that this line is updating a member called `center` of the object?

I rest my case :)

Style: Too Many Nested Blocks

In the `solve` method, some of you have FIVE levels of nested blocks: two double loops to go through each pair of points, one to check for circle validity, another loop to go through all points, one to check for containment.

These nested blocks make the code long and hard to read. One of you have to label the `{` and `}` to help with bracket matching.

As a guideline, if you have more than two nested blocks, it is time to think about breaking down the method into smaller / shorter methods. For instance, for every pair of points, find out how many points are in the two circles that pass throught this pair.

Shorter methods are easier to read, understand, and debug. So do your future self a favour.

Style: `if (x == true)`

`x` can be a variable or a function. Some of you wrote as `if (x == true)` in the condition, some wrote it as `if (x)`.

`if (x)` is more succinct, and if you name `x` properly, it is more readable and understandable than `if (x == true)`.

For instance,

```
if (circle.isValid()) {..}
```

is perfectly clear that we are checking if the circle is valid.

```
if (circle.isValid() == true)) {..}
```

is redundant.

What if we call the function `x` as something else, say, `checkValidity()`? First of all, DON'T. Second, yes, it is then OK to write:

```
if (circle.checkValidity() == true)) {..}
```

but can you tell by reading this line of code, whether `true` means the circle is valid or not? So, DON'T. Keep your code short, english-like, and choose a proper name for your variables and methods.