# CS2030 Programming Methodology

Semester 2 2019/2020

30 January 2020

Problem Set #2 Suggested Guidance

**Method Overriding/Overloading and Inheritance**

1. Study the following `Point` and `Circle` classes.

```
public class Point {
    private final double x;
    private final double y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}


public class Circle {

    private final Point centre;
    private final int radius;

    public Circle(Point centre, int radius) {
        this.centre = centre;
        this.radius = radius;
    }

    @Override
    public boolean equals(Object obj) {
        System.out.println("equals(Object) called");
        if (obj == this) {
            return true;
        }
        if (obj instanceof Circle) {
            Circle circle = (Circle) obj;
            return (circle.centre.equals(centre) && circle.radius == radius);
        } else {
            return false;
        }
    }

    public boolean equals(Circle circle) {
        System.out.println("equals(Circle) called");
        return circle.centre.equals(centre) && circle.radius == radius;
    }
}
```

Given the following program fragment,

```
Circle c1 = new Circle(new Point(0, 0), 10);
Circle c2 = new Circle(new Point(0, 0), 10);
Object o1 = c1;
Object o2 = c2;
```

what is the output of the following statements?

(a) `o1.equals(o2);`

(b) `o1.equals((Circle) o2);`

(c) `o1.equals(c2);`

(d) `o1.equals(c1);`

(e) `c1.equals(o2);`

(f) `c1.equals((Circle) o2);`

(g) `c1.equals(c2);`

(h) `c1.equals(o1);`

```
jshell> o1.equals(o2)
equals(Object) called
$.. ==> false

jshell> o1.equals((Circle) o2)
equals(Object) called
$.. ==> false

jshell> o1.equals(c2)
equals(Object) called
$.. ==> false

jshell> o1.equals(c1)
equals(Object) called
$.. ==> true

jshell> c1.equals(o2)
equals(Object) called
$.. ==> false

jshell> c1.equals((Circle) o2);
equals(Circle) called
$.. ==> false

jshell> c1.equals(c2)
equals(Circle) called
$.. ==> false

jshell> c1.equals(o1)
equals(Object) called
$.. ==> true
```

*Calling the* `equals` *method through a variable of compile-time type* `Object` *would invoke the* `equals(Object)` *method of* `Object`*. This method can be overridden by the overriding method of the same name in the sub-class* `Circle`*.*

*The only time that the overloaded method* `equals(Circle)` *can be called is when the method is invoked through a variable of compile-time type* `Circle`*, and the run-time type is also* `Circle` *(as can be seen in the output of the code excerpt* `c1.equals(c2)`*).*

*The output of* `true` *or* `false` *largely depends on the presence of an overriding* `equals` *method in the* `Point` *class.*

2. We would like to design a class `Square` that inherits from `Rectangle`. A square has the constraint that the four sides are of the same length.

   (a) How should `Square` be implemented to obtain the following output from `JShell`?

```
jshell> new Square(5)
$3 ==> area 25.00 and perimeter 20.00

public class Rectangle {
    private final double width;
    private final double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    public double getArea() {
        return width * height;
    }

    public double getPerimeter() {
        return 2 * (width + height);
    }

    @Override
    public String toString() {
        return "area " + String.format("%.2f", getArea()) +
            " and perimeter " + String.format("%.2f", getPerimeter());
    }
}

public class Square extends Rectangle {
    public Square(double length) {
        super(length, length);
    }
}
```

(b) Now implement two separate methods to set the width and height of the rectangle:

```
public Rectangle setHeight(double height) {
    return new Rectangle(this.width, height);
}

public Rectangle setWidth(double width) {
    return new Rectangle(width, this.height);
}
```

What undesirable design issues would this present?

`Square` *inherits the* `setHeight` *and* `setWidth` *methods from* `Rectangle`*. As a consequence, a square can be changed to a rectangle.*

```
jshell> new Square(5.0).setHeight(10.0)
$3 ==> area 50.00 and perimeter 30.00
```

(c) Now implement two overriding methods in the `Square` class

```
@Override
public Square setHeight(double height) {
    return new Square(height);
}

@Override
public Square setWidth(double width) {
    return new Square(width);
}
```

Do you think that it is now sensible for to have `Square` inherit from `Rectangle`? Or should it be the other way around? Or maybe they should not inherit from each other?

*Based on the substitutability principle, if* `Square` *inherits from* `Rectangle`*, then anywhere we expect a* `Rectangle`*, we can always substitute it with a* `Square`*.*

*Consider the following example,*

```
jshell> Rectangle[] rects = {new Rectangle(3.0, 5.0), new Square(5.0)}
rects ==> Rectangle[2] { area 15.00 and perimeter 16.00, area 25.00
and perimeter 20.00 }

jshell> rects[0].setHeight(4.0).setWidth(8.0)
$4 ==> area 32.00 and perimeter 24.00

jshell> rects[1].setHeight(4.0).setWidth(8.0)
$6 ==> area 64.00 and perimeter 32.00
```

*Notice that setting* `rects[1]` *(of type* `Rectangle`*) to a height of* 4.0 *and a width of* 8.0 *does not produce the desired rectangle.*