

CS2030 Programming Methodology II
Semester 1 2022/2023

31 August & 1 September 2022

Problem Set #2

Inheritance and Polymorphism

1. Study the following Circle class.

```
class Circle {  
    private final int radius;  
  
    Circle(int radius) {  
        this.radius = radius;  
    }  
  
    @Override  
    public String toString() {  
        return "Circle with radius " + this.radius;  
    }  
}
```

toString() We have seen how the `toString` method can be defined in the `Circle` class that overrides the same method in its parent `java.lang.Object` class. There is another `equals(Object obj)` method defined in the `Object` class which returns `true` only if the object from which `equals` is called, and the argument object is the same.

[https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Object.html#equals\(java.lang.Object\)](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Object.html#equals(java.lang.Object))

```
jshell> Circle c = new Circle(10)  
c ==> Circle with radius 1
```

```
jshell> c.equals(c)  
$.. ==> true
```

```
jshell> c.equals("10")  
$.. ==> false
```

```
jshell> c.equals(new Circle(10))  
$.. ==> false
```

→ becos c and new Circle are referencin different objects.

In particular for the latter test, since both `c` and `new Circle(10)` have radius of `10 units`, we would like the `equals` method to return `true` instead.

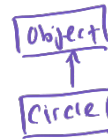
- (a) We define an overloaded method `equals(Circle other)` in the `Circle` class:

```
boolean equals(Circle circle) {  
    System.out.println("Running equals(Circle) method");  
    return circle.radius == radius;  
}
```

such that

```
jshell> new Circle(10).equals(new Circle(10))  
Running equals(Circle) method  
$.. ==> true
```

```
jshell> new Circle(10).equals("10")  
$.. ==> false
```



Why is the outcome of the following test false?

```
jshell> Object obj = new Circle(10)  
obj ==> Circle with radius 1
```

```
jshell> obj.equals(new Circle(10))  
$.. ==> false
```

We are assigning the new circle object (which is the child class) to be of type object (which is the parent class). We cannot use the child's methods in the parent class. So in the end, the equals method of object class is used. ∴ false.

- (b) Instead of an overloaded method, we now define an overriding method.

```
@Override  
public boolean equals(Object obj) {  
    System.out.println("Running equals(Object) method");  
    if (obj == this) { // trivially true since it's the same object  
        return true;  
    } else if (obj instanceof Circle circle) { // is obj a Circle?  
        return circle.radius == this.radius;  
    } else {  
        return false;  
    }  
}
```

Why does the same test case in question 1a now produce the correct expected outcome?

```
jshell> Object obj = new Circle(10)  
obj ==> Circle with radius 1
```

```
jshell> obj.equals(new Circle(10))  
Running equals(Object) method  
$.. ==> true
```

This new overriding equals method takes in an object which is of the object class. This method can be used by obj, which is of type object. This overriding method is used instead of the original equals method.

- (c) With both the overloaded and overriding `equals` method in questions 1a and 1b defined, given the following program fragment,

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

what is the output of the following statements?

- | | |
|--|--------------------------------------|
| (a) ^{obj} o1.equals(^{obj} o2); ^{True} ✓ | (d) c1.equals(o2); ^{True} ✓ |
| (b) ^{obj} o1.equals(^{Circle} c2); ^{True} ✓ | (e) c1.equals(c2); ^{True} ✓ |
| (c) o1.equals(c1); ^{True} ✓ | (f) c1.equals(o1); ^{True} ✓ |

2. A unit-circle is a special type of circle with radius 1. We would like to design a class `UnitCircle` that inherits from `Circle`. As an example of constructing a `Circle`,

```
jshell> new Circle(new Point(1.0, 2.0), 3.0)
$.. ==> circle at (1.0, 2.0) with radius 3.0
```

- (a) By adhering to the abstraction principle, how should `UnitCircle` be implemented to obtain the following evaluation from `JShell`?

```
jshell> new UnitCircle(new Point(1.0, 2.0))
$.. ==> circle at (1.0, 2.0) with radius 1.0
```

- (b) Now implement the method to scale the circle:

```
Circle scale(double factor) { ... }
```

```
jshell> new Circle(new Point(1.0, 2.0), 3.0).scale(2.0)
$.. ==> circle at (1.0, 2.0) with radius 6.0
```

- (c) What happens when `UnitCircle` inherits the method `scale` from `Circle`?
 (d) Should we override the `scale` method in the `UnitCircle` class?
 (e) Do you think that it is sensible to have `UnitCircle` inherit from `Circle`?
 (f) Should `Circle` inherit from `UnitCircle`? Or maybe they should not inherit from each other at all?

3. Which of the following program fragments will result in a compilation error?

- ~~(a)~~

```
class A1 {
    void f(int x) {}
    void f(boolean y) {}
}
```
- ~~(b)~~

```
class A2 {
    void f(int x) {}
    void f(int y) {}
}
```

?

(c) class A3 {
 private void f(int x) {}
 void f(int y) {}
 }

(d) class A4 {
 int f(int x) {
 return x;
 }
 void f(int y) {}
 }

(e) class A5 {
 void f(int x, String s) {}
 void f(String s, int y) {}
 }

2a)

```
48 class UnitCircle extends Circle {
49
50     UnitCircle(Point centre) {
51         super(centre);
52     }
53
54     private getCentre() {
55         super.getCentre();
56     }
57
58     @Override
59     public String toString() {
60         return "Circle at " + this.getCentre().toString() + " with radius 1.0";
61     }
62 }
```

2b)

```
17 @Override
18 public String toString() {
19     return "Circle at " + this.getCentre().toString() + " with radius " + this.getRad();
20 }
21
22 Circle scale(double factor) {
23     return new Circle(this.getCentre(), factor * this.getRad());
24 }
25 }
```

- 2c) Compilation error. No radius parameter needed for a circle object, since radius is fixed at 1. Method scale which requires a reference to the radius of itself does not work.
- 2d) No. makes no sense. Should not scale the radius of a unit circle.
- 2e) Yes, makes use of the Point parameters. Unit circle is a circle
- 2f) No. Circle is not a unit circle.