CS2030 Programming Methodology II

Semester 1 2022/2023

1 & 2 February 2023

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;
    }
}
```

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)`

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

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

This method returns true if and only if x and y refer to the same object (x == y has the value true)

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

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

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
```

compile-time type Object, obj can only call Object equals() method. Overloading does not work here, only overriding can

Why is the outcome of the following test `false`?

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

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

obj is compile-time type Object
while run-time type Circle
As there is no @Override declared in equals method in circle class, the equals() invoked is Object method, returning false as not pointing to the same reference

(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;
    }
}
```

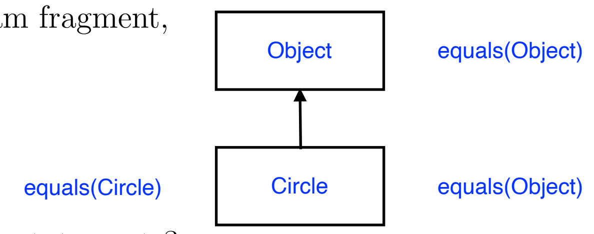
Why does the same test case in question 1a now produce the correct expected outcome?

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

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

(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) `o1.equals(o2);` Running equals(Object) method
true

(b) `o1.equals(c2);` Running equals(Object) method
true

(c) `o1.equals(c1);` Running equals(Object) method
true

(d) `c1.equals(o2);` Running equals(Object) method
true

(e) `c1.equals(c2);` Running equals(Circle) method
true

(f) `c1.equals(o1);` Running equals(Object) method
true

2. We would like to design a class `Square` that inherits from `Rectangle`.

```
class Rectangle {
    private final int width;
    private final int height;

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

    @Override
    public String toString() {
        return this.width + " x " + this.height;
    }
}
```

As an example of constructing a rectangle,

```
jshell> new Rectangle(3, 4) // width = 3 and height = 4
$.. ==> 3 x 4
```

(a) A square has the constraint that the four sides are of the same length. Keeping in mind the *abstraction principle*, how should `Square` be implemented to obtain the following evaluation from `JShell`?

```
jshell> new Square(5)
$.. ==> 5 x 5
```

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

```
Rectangle setWidth(double width) { ... }
Rectangle setHeight(double height) { ... }

jshell> new Rectangle(3, 4).setHeight(2)
$.. ==> 3 x 2
```

SetWidth() or SetHeight would change it to a rectangle as only the width or length changes

Moreover, return type is rectangle -> leading to a compilation error

(c) What happens if `Square` inherits the methods `setWidth` and `setHeight` from `Rectangle`?

Change the methods such that it “returns new square(width)”

(d) How would you override the methods `setWidth` and `setHeight` in the `Square` class?

(e) Do you think that it is now sensible to have `Square` inherit from `Rectangle`? No

(f) Should `Rectangle` inherit from `Square`? Or maybe they should not inherit from each other at all? None at all

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

Decide by name and parameter list!

(a)
```
class A1 {
    void f(int x) {}
    void f(boolean y) {}
}
```
Different parameter list:
one int , one boolean
can overload

(b)
```
class A2 {
    void f(int x) {}
    void f(int y) {}
}
```
same name, same parameter list:
cannot overload
-> compilation error

(c)
```
class A3 {
    private void f(int x) {}
    void f(int y) {}
}
```
same name, same parameter list:
cannot overload
-> compilation error

(d)
```
class A4 {
    int f(int x) {
        return x;
    }
    void f(int y) {}
}
```
same name, same parameter list:
cannot overload
-> compilation error

(e)
```
class A5 {
    void f(int x, String s) {}
    void f(String s, int y) {}
}
```
Different parameter list:
(int, string), vs (String, int)
can overload