

CS2030 Programming Methodology

Semester 2 2018/2019

30 January – 01 February 2019

Tutorial 1 Suggested Guidance

Abstraction, Encapsulation and Inheritance

1. Given the following program fragment.

```
class A {  
    public int x = 5;  
    public static int y = 1;  
  
    public A() {  
        x = x + 1;  
        y = y + 1;  
    }  
}
```

By either creating a `main` method or using `JShell`, invoke the following:

```
A a1 = new A();  
A a2 = new A();
```

- (a) After executing `a1.x = 10`, what is the value of `a2.x`?
- (b) After executing `a1.y = 10`, what is the value of `a2.y`?
- (c) What is the significance of the `static` keyword used during instance variable declaration? How is it useful?
- (d) Is `A.x = 3` a valid statement? How about `a1.x = 3` and `A.y = 3`?

Students are expected to try it and infer the differences between an instance variable, and a class (static) variable used for aggregation purposes. Even though `a1.y` is valid, it is considered a bad programming practice to access a class field through an instance variable; the proper way is to use the class name `A.y`.

2. Consider the following two classes:

<pre>public class P { private int x; public void changeSelf() { x = 1; } public void changeAnother(P p) { p.x = 1; } }</pre>	<pre>public class Q { public void changeAnother(P p) { p.x = 1; } }</pre>
--	---

- (a) Which line(s) above violate the private access modifier of `x`?
- (b) What does this say about the concept of an “abstraction barrier”?

The abstraction barrier sits between the client and the implementer. Here class `P` is the implementer, and `Q` is the client that makes use of the `p`, an object of `P`.

The barrier is not broken when one object of type `P` accesses the instance variables of another type `P` object, since `P` is the sole implementer.

3. Given the following class `Circle`.

```
public class Circle {

    Point centre;
    double radius;

    public Circle(Point centre, double 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;
    }
}
```

In the following 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) <code>o1.equals(o2);</code> | (e) <code>c1.equals(o2);</code> |
| (b) <code>o1.equals((Circle) o2);</code> | (f) <code>c1.equals((Circle) o2);</code> |
| (c) <code>o1.equals(c2);</code> | (g) <code>c1.equals(c2);</code> |
| (d) <code>o1.equals(c1);</code> | (h) <code>c1.equals(o1);</code> |

```
jshell> o1.equals(o2)
equals(Object) called
...

jshell> o1.equals((Circle) o2)
equals(Object) called
...

jshell> o1.equals(c2)
equals(Object) called
...

jshell> o1.equals(c1)
equals(Object) called
...

jshell> c1.equals(o2)
equals(Object) called
...

jshell> c1.equals((Circle) o2);
equals(Circle) called
...

jshell> c1.equals(c2)
equals(Circle) called
...

jshell> c1.equals(o1)
equals(Object) called
...
```

Calling the `equals` method though a reference of type `Object` would invoke the `toString` method of `Object`, but which is overridden by the same method of the sub-class `Circle`.

The only time that the overloaded method `equals(Circle circle)` can be called is when the method is invoked through an object of `Circle` type, and the argument is an object of `Circle` type also.

The final print out of `true` or `false` depends on how `equals` method of `Point` is defined.

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

(a)

```
class A {  
    public void f(int x) {}  
    public void f(boolean y) {}  
}
```

(b)

```
class A {  
    public void f(int x) {}  
    public void f(int y) {}  
}
```

(c)

```
class A {  
    private void f(int x) {}  
    public void f(int y) {}  
}
```

(d)

```
class A {  
    public int f(int x) {  
        return x;  
    }  
    public void f(int y) {}  
}
```

(e)

```
class A {  
    public void f(int x, String s) {}  
    public void f(String s, int y) {}  
}
```

Method overloading supports a class to have more than one method of the same name (or constructor) with different argument lists (number/type/order of parameters).

(a) Compilable

(b)

```
A.java:3: error: method f(int) is already defined in class A  
    public void f(int y) {}  
                ^
```

1 error

(c)

```
A.java:3: error: method f(int) is already defined in class A  
    public void f(int y) {}  
                ^
```

1 error

(d)

```
A.java:5: error: method f(int) is already defined in class A  
    public void f(int y) {}  
                ^
```

1 error

(e) Compilable

5. In Lecture #3, we designed the class `Rectangle` that inherits from the class `Shape`. Now we want 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 such that we obtain the following using JShell?

```
jshell Shape.java Rectangle.java Square.
java
| Welcome to JShell -- Version 9.0.4
| For an introduction type: /help intro

jshell> Square s = new Square(5)
s ==> Square with area 25.00 and perimeter 20.00

jshell>
```

- (b) Do you think `Square` should inherit from `Rectangle`? Or should it be the other way around? Or maybe they should not inherit from each other?

Suppose `Square` inherits from `Rectangle`, so methods in `Rectangle` gets inherited by `Square`.

One method that can be included into class `Rectangle` is the

```
resize(int width, int height)
```

method that resizes the rectangle object.

Now, suppose a client of the rectangle class receives `Rectangle` objects (some of which could be of type `Square`). By applying `resize` on these objects, one can turn a square into a rectangle since the method `resize` is inherited from `Rectangle` to a `Square`. Even `setsize` can be overridden in `Square`, which of the two parameters, `height` or `width` would it use?