CS2030 Programming Methodology
Semester 1 2018/2019

07 September 2018
Tutorial 2
Inheritance and Polymorphism

1. Given the following interfaces.

```
public interface Shape {
    public double getArea();
}

public interface Printable {
    public void print();
}
```

   (a) Suppose class `Circle` implements both interfaces above. Given the following program fragment,

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

   Are the following statements allowed? Why do you think Java does not allow some of the following statements?

   i. s.print();
   ii. p.print();
   iii. s.getArea();
   iv. p.getArea();

   (b) Someone proposes to re-implement `Shape` and `Printable` as abstract classes instead? Would this work?

   (c) Can we define another interface `PrintableShape` as

```
public interface PrintableShape extends Printable, Shape {
}
```

   and let class `Circle` implement `PrintableShape` instead?

2. Write a class `Rectangle` that implements the two interfaces in question 1. You should make use of two diagonally-opposite points (bottom-left and top-right) to define the rectangle. How do you handle the case that the two points do not define a proper rectangle?

   Assume that the sides of the rectangles are parallel with the x- and y-axes (in other words, the sides are either horizontal or vertical).

3. Let's now extend our shapes from two-dimensional to three dimensional.

   (a) Write an interface called `Shape3D` that supports a method `getVolume`. Write a class called `Cuboid` that implements `Shape3D` and has three private `double` fields `length`, `height`, and `breadth`. The method `getVolume()` should return the volume of the `Cuboid` object. The constructor for `Cuboid` should allow the client to create a `Cuboid` object by specifying the three fields length, height and breadth.

   (b) Write a new interface `Solid3D` that inherits from interface `Shape3D` that supports two methods: `getDensity()` and `getMass()`.

   (c) Now, write a new class called `SolidCuboid` with an additional private `double` field `density`. The implementation of `getDensity()` should return this field while `getMass()` should return the mass of the cuboid. The `SolidCuboid` should call the constructor of `Cuboid` via `super` and provides two constructors: one constructor that allows the client to specify the density, while the other does not and just sets the default density to 1.0.

   (d) Test your implementation with by writing a suitable client class.

4. Write each of the following program fragments using `jshell`. Will it result in a compilation or runtime error? If not, what is the output?

(a)
```
class A {
    void f() {
        System.out.println("A f");
    }
}

class B extends A {
}

B b = new B();
b.f();
A a = b;
a.f();
```

(b)
```
class A {
    void f() {
        System.out.println("A f");
    }
}

class B extends A {
    void f() {
        System.out.println("B f");
    }
}

B b = new B();
b.f();
A a = b;
a.f();
a = new A();
a.f();
```

(c)
```
class A {
    void f() {
        System.out.println("A f");
    }
}

class B extends A {
    void f() {
        super.f();
        System.out.println("B f");
    }
}

B b = new B();
b.f();
A a = b;
a.f();
```

(d)
```
class A {
    void f() {
        System.out.println("A f");
    }
}

class B extends A {
    void f() {
        this.f();
        System.out.println("B f");
    }
}

B b = new B();
b.f();
A a = b;
a.f();
```

(e)
```
class A {
    void f() {
        System.out.println("A f");
    }
}

class B extends A {
    int f() {
        System.out.println("B f");
        return 0;
    }
}

B b = new B();
b.f();
A a = b;
a.f();
```

(f)
```
class A {
    void f() {
        System.out.println("A f");
    }
}

class B extends A {
    void f(int x) {
        System.out.println("B f");
    }
}

B b = new B();
b.f();
b.f(0);
A a = b;
a.f();
a.f(0);
```

(g)
```
class A {
    public void f() {
        System.out.println("A f");
    }
}

class B extends A {
    public void f() {
        System.out.println("B f");
    }
}

B b = new B();
A a = b;
a.f();
b.f();
```

(h)
```
class A {
    private void f() {
        System.out.println("A f");
    }
}

class B extends A {
    public void f() {
        System.out.println("B f");
    }
}

class Main {
    public static void main(String[] args) {
        B b = new B();
        A a = b;
        a.f();
        b.f();

    }
}
```

(i)
```
class A {
    static void f() {
        System.out.println("A f");
    }
}

class B extends A {
    public void f() {
        System.out.println("B f");
    }
}

B b = new B();
A a = b;
a.f();
b.f();
```

(j)
```
class A {
    static void f() {
        System.out.println("A f");
    }
}

class B extends A {
    static void f() {
        System.out.println("B f");
    }
}

B b = new B();
A a = b;
A.f();
B.f();
a.f();
b.f();
```

(k)
```
class A {
    private int x = 0;
}

class B extends A {
    public void f() {
        System.out.println(x);
    }
}

B b = new B();
b.f();
```

(l)
```
class A {
    private int x = 0;
}

class B extends A {
    public void f() {
        System.out.println(super.x);
    }
}

B b = new B();
b.f();
```

(m)
```
class A {
    protected int x = 0;
}

class B extends A {
    public void f() {
        System.out.println(x);
    }
}

B b = new B();
b.f();
```

(n)
```
class A {
    protected int x = 0;
}

class B extends A {
    public int x = 1;
    public void f() {
        System.out.println(x);
    }
}

B b = new B();
b.f();
```

(o)
```
class A {
    protected int x = 0;
}

class B extends A {
    public int x = 1;
    public void f() {
        System.out.println(super.x);
    }
}

B b = new B();
b.f();
```