CS2030 Programming Methodology
Semester 1 2018/2019

07 September 2018
Tutorial 2 Suggested Guidance
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();

   Only `s.getArea()` and `p.print()` are premissible. Suppose `Shape s` references an array of objects that implements the `Shape` interface, so each object is guaranteed to implement the `getArea` method. Other than that, each object may or may not implement other interfaces (such as `Printable`), so `s.print()` may or may not be applicable.

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

   No, you cannot inherit from multiple parent classes.

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

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

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

   Yes, it is allowed. Interfaces can inherit from multiple parent interfaces.

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).

```
public class Rectangle implements Shape, Printable {
    Point bottomLeft;
    Point topRight;

    private Rectangle(Point bottomLeft, Point topRight) {
            this.bottomLeft = bottomLeft;
            this.topRight = topRight;
    }

    public static Rectangle getRectangle(Point bottomLeft, Point topRigth) {
        if (getLength(bottomLeft, topRight) > 0 &&
                getHeight(bottomLeft, topRight) > 0) {
            return new Rectangel(bottomLeft, topRight);
        } else {
            return null;
        }
    }

    private static double getLength(Point bottomLeft, Point topRight) {
        return topRight.getX() - bottomLeft.getX();
    }

    private static double getHeight(Point bottomLeft, Point topRight) {
        return topRight.getY() - bottomLeft.getY();
    }

    private double getLength() {
        return getLength(this.bottomLeft, this.topRight);
    }

    private double getHeight() {
        return getHeight(this.bottomLeft, this.topRight);
    }

    public double getArea() {
        return getLength() * getHeight();
    }

    public void print() {
        System.out.println("Printable...");
    }
}
```

   Alternatively, one can throw an exception as demonstrated during the lecture.

```
public class Rectangle implements Shape, Printable {
    Point bottomLeft;
    Point topRight;

    public Rectangle(Point bottomLeft, Point topRight) {
        if (getLength(bottomLeft, topRight) > 0 &&
                getHeight(bottomLeft, topRight) > 0) {
            this.bottomLeft = bottomLeft;
            this.topRight = topRight;
        } else {
            throw new IllegalArgumentException(bottomLeft + " " +
                    topRight);
        }
    }

    private static double getLength(Point bottomLeft, Point topRight) {
        return topRight.getX() - bottomLeft.getX();
    }

    private static double getHeight(Point bottomLeft, Point topRight) {
        return topRight.getY() - bottomLeft.getY();
    }

    private double getLength() {
        return getLength(this.bottomLeft, this.topRight);
    }

    private double getHeight() {
        return getHeight(this.bottomLeft, this.topRight);
    }

    public double getArea() {
        return getLength() * getHeight();
    }

    public void print() {
        System.out.println("Printable...");
    }
}

import java.util.Scanner;
class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextDouble()) {
            Point bottomLeft = new Point(sc.nextDouble(), sc.nextDouble());
            Point topRight = new Point(sc.nextDouble(), sc.nextDouble());
            try {
                Rectangle r = new Rectangle(bottomLeft, topRight);
                System.out.println(r.getArea());
            } catch (IllegalArgumentException ex) {
                System.err.println(ex);
            }
        }
    }
}
```

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.

```
public interface Shape3D {
    double getVolume();
}

public interface Solid3D {
    double getDensity();
    double getMass();
}

public class Cuboid implements Shape3D {
    private double length;
    private double height;
    private double breadth;

    public Cuboid(double length, double breadth, double height) {
        this.length = length;
        this.breadth = breadth;
        this.height = height;
    }

    public double getVolume() {
        return length * height * breadth;
    }
}

public class SolidCuboid extends Cuboid implements Solid3D {
    private double density;

    public SolidCuboid(double length, double height, double breadth,
            double density) {
        super(length, height, breadth);
        this.density = density;
    }

    public SolidCuboid(double length, double height, double breadth) {
        this(length, breadth, height, 1.0);
    }

    public double getDensity() {
        return density;
    }

    public double getMass() {
        return getVolume() * density;
    }
}
```

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();
```

Students are encouraged try these out themselves. Just some noteworthy mention below:

- (d) results in an infinite recursion leading to stack overflow
- (e) is a compilation error as method `f()` has the same method signature, which implies the method in `B` should override that of `A`, but the return type is different.
- In (f), `a.f(0)` is not accessible, only `a.f()` is ok.
- In (h), `a.f()` has private access.a
- (i) is a compilation error as `f()` in `B` cannot override `f()` in `A` which is declared `static`. `static` methods cannot be overridden.
- (k) is a compilation error as `x` has private access; likewise for (l)

More detailed output below:

```
AB1.java
A f
A f
AB2.java
B f
B f
A f
AB3.java
A f
B f
A f
B f
AB4.java
|  java.lang.StackOverflowError thrown:
|        at B.f (#2:3)
:
|        at B.f (#2:3)
AB5.java
|  Error:
|  f() in B cannot override f() in A
|    return type int is not compatible with void
|      int f() {
|      ^--------...
AB6.java
A f
B f
A f
|  Error:
|  method f in class A cannot be applied to given types;
|    required: no arguments
|    found: int
|    reason: actual and formal argument lists differ in length
|  a.f(0);
|  ^-^
```

```
AB7.java
B f
B f
AB8.java
|  Error:
|  f() has private access in A
|  a.f();
|  ^-^
B f
AB9.java
|  Error:
|  f() in B cannot override f() in A
|    overridden method is static
|      public void f() {
|      ^-----------------...
AB10.java
A f
B f
A f
B f
AB11.java
|  Error:
|  x has private access in A
|          System.out.println(x);
|                             ^
AB12.java
|  Error:
|  x has private access in A
|          System.out.println(super.x);
|                             ^-----^
AB13.java
0
AB14.java
1
AB15.java
```