CS2030 Programming Methodology
Semester 2 2019/2020

6 February 2020
Problem Set #3

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. Using examples of overriding methods, illustrate why a Java class cannot inherit from multiple parent classes, but can implement multiple interfaces.

3. Consider the following classes: `FormattedText` that adds formatting information to the text. We call `toggleUnderline()` to add or remove underlines from the text. A `URL` is a `FormattedText` that is always underlined.

```
class FormattedText {
    public String text;
    public boolean isUnderlined;

    public void toggleUnderline() {
        isUnderlined = (!isUnderlined);
    }
}

class URL extends FormattedText {
    public URL() {
        isUnderlined = true;
    }

    @Override
    public void toggleUnderline() {
        return;
    }
}
```

Does the above violate Liskov Substitution Principle? Explain.

4. Consider the following program fragment.

```
class A {
    int x;
    A(int x) {
        this.x = x;
    }
    public A method() {
        return new A(x);
    }
}

class B extends A {
    B(int x) {
        super(x);
    }
    @Override
    public B method() {
        return new B(x);
    }
}
```

Does it compile? What happens if we switch the method definitions between class `A` and class `B` instead? Give reasons for your observations.