CS2030 Programming Methodology

Semester 1 2019/2020

6 September 2019

Problem Set #2

Inheritance, Interfaces 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. 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.

3. Give practical reasons as to why a Java class cannot inherit from multiple parent classes, but can implement multiple interfaces.

4. A solid cuboid is a box-shaped solid object made up of a specific material, and having six flat sides with all right-angled corners. From the three sides of the solid cuboid, one can compute the volume. In addition, using the density of the material, the mass can be found. By using object-oriented modeling and applying the *abstraction principle*, design a Java program to faciliate the creation of a solid cuboid object. Show how you can obtain its volume, density as well as mass.

   Tip: When designing the classes and interfaces, keep in mind the SOLID principles, as well as possible extensions you might want to model to generalize your program.

5. For each of the following program fragments, 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();
```