CS2030S
Problem Set 2

AY22/23 S2 | 1-2 February 2023

1. Consider the `Rectangle` class below:

```
public class Rectangle {
  private double width;
  private double height;

  public Rectangle(double width, double height) {
    this.width = width;
    this.height = height;
  }

  public double getArea() {
    return this.width * this.height;
  }

  @Override
  public String toString() {
    return "Width: " + this.width +  " Height: " + this.height;
  }
}
```

The method `Rectangle::getArea` is expected to return the product of its width and height.

```
jshell> new Rectangle(5, 8)
$.. ==> Width: 5.0 Height: 8.0
jshell> new Rectangle(5, 8).getArea()
$.. ==> 40.0
```

We would like to design a class `Square` that inherits from `Rectangle`. A `Square` instance must satisfy the constraint that the four sides are always of the same length.

(a) Create a class called `Square` with a single constructor method such that we have the following output from `JShell`.

```
jshell> new Square(5)
$.. ==> Width: 5.0 Height: 5.0
jshell> new Square(5).getArea()
$.. ==> 25.0
```

Calling `getArea` on a `Square` instance returns the square of its width (or equivalently, its height).

(b) Now, implement two separate methods to set the width and height of the rectangle in the `Rectangle` class.

```
public void setHeight(double height) {
  this.height = height;
}

public void setWidth(double width) {
  this.width = width;
}
```

The `Rectangle` class is expected to behave as follows:

```
jshell> Rectangle r = new Rectangle(5, 5)
jshell> r.setHeight(5)
jshell> r.setWidth(9)
jshell> r.getArea()
$.. ==> 45.0
```

Specifically, if the most recent call to `setWidth` is `setWidth(w)` and the most recent call to `setHeight` is `setHeight(h)`, then `getArea()` must return `w * h`.

Explain how the behavior of the `Square` class is undesirably affected by this addition.

(c) Now implement two overriding methods in the `Square` class:

```
@Override
public void setHeight(double height) {
  super.setHeight(height);
  super.setWidth(height);
}

@Override
public void setWidth(double width) {
  super.setHeight(width);
  super.setWidth(width);
}
```

These methods overrides the methods in `Rectangle` so that the width and height can no longer be set independently.

Do you think that it is still sensible to have `Square` inherit from `Rectangle`?

(d) Is it sensible for `Rectangle` to inherit from `Square` instead?

2. Consider 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 the Java compiler 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?

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

Past Year Questions

These questions are provided here for discussion among yourselves (e.g., on Piazza). We will not discuss these during the recitations.

3. **Midterm 2020/21 Semester 2.**

   Consider the following four classes:

```
class A {
  void foo(A a) {
    System.out.println("class: A, parameter: A");
  }
}

class B extends A {
  @Override
  void foo(A a) {
    System.out.println("class: B, parameter: A");
  }

  void foo(B a) {
    System.out.println("class: B, parameter: B");
  }
}

class C extends B {
  void foo(C a) {
    System.out.println("class: C, parameter: C");
  }
}

class D extends C {
  @Override
  void foo(B a) {
    System.out.println("class: D, parameter: B");
  }
}
```

   We initialize four variables as follows:

```
A a = new D();
B b = new D();
C c = new D();
D d = new D();
```

   What will be printed if we call:

   (a) `a.foo(d);`

   (b) `b.foo(d);`

   (c) `c.foo(d);`

   (d) `d.foo(d);`

4. **Midterm 2020/21 Semester 2.**

   Consider the following classes.

   The class `Time` encapsulates a time measurement in units of seconds and milliseconds.

```
class Time {
  public int second;
  public int millisecond;

  public Time(int second, int millisecond) {
    this.second = second;
    this.millisecond = millisecond;
  }
}
```

The class `Interval` encapsulates a time interval and consists of a starting time (begin) and an ending time (end).

```
class Interval {
  private Time begin;
  private Time end;

  public Interval(Time begin, Time end) {
    this.begin = begin;
    this.end = end;
  }

  public int durationInMs() {
    return (end.second * 1000 + end.millisecond)
      - (begin.second * 1000 + begin.millisecond);
  }
}
```

A `Jiffy` is an interval with a fixed duration of 10ms.

```
class Jiffy extends Interval {
  public Jiffy() {
    super(new Time(0, 0), new Time(0, 10));
  }

  @Override
  public int durationInMs() {
    return 10;
  }
}
```

(a) The code above violates information hiding. True or false? Why?

(b) The code above violates the ”tell, don’t ask” principle. True or false? Why?

(c) The code above violates the Liskov substitution principle. True or false? Why?