CS284: Exercise Booklet 1 - Java

Exercise 1

See the first video (`01-java-classes`) and implement a class `Person` with at least the following data fields: name, age, address, and the corresponding getters and setters. Try out your class by adding a main method, as seen in the video, that creates a couple of instances of your class and then prints out the name, age and address.

Exercise 2

Add JavaDoc comments to your code after viewing video 2 (`02-comments`). After doing so, place your cursor on the class declaration to verify that they have been processed by Eclipse (you should get a popup with information regarding the class, following the description that you inserted as a JavaDoc comment).

Exercise 3

Define classes Student, Undergraduate, Graduate, Faculty, Non-Tenured and Tenured, and organize them into a hierarchy whose root is Person from the previous exercise. Use video 3 (`03-inheritence`) as a guideline. Try your classes out by instantiating some of them and then printing out information on them.

Exercise 4

Add a method `public String getCredentials()` to Student, Undergraduate, Graduate, Faculty, Non-Tenured and Tenured classes. It should return the appropriate credentials. For a student it should be her CWID. Undergrads should also list their current GPA. Grad students should list their major. Think of similar credentials for the other classes. This is illustrates method override, assuming you organized your hierarchy in the previous exercise as expected. See video 4 (`04-method-override`) before solving this exercise.

Exercise 5

Define a class School that includes two array data fields, faculty and students. Provide operations to print out all the students and similarly for faculty.

Exercise 6

Watch video 5 (`05-subtype-polymorphism`) and then answer the following questions. Suppose you have the following two classes:

```
public class Circle {
   // private instance variable
   private double radius;
   // Constructor
   public Circle(double radius) {
      this.radius = radius;
   }
   // Getter
   public double getRadius() {
      return this.radius;
   }
```

```
    // Return the area of this circle
    public double getArea() {
        return radius * radius * Math.PI;
    }
    // Describe itself
    public String toString() {
        return "Circle[radius=" + radius + "]";
    }
}
```

```
public class Cylinder extends Circle {
    // private instance variable
    private double height;
    // Constructor
    public Cylinder(double height, double radius) {
        super(radius);
        this.height = height;
    }
    // Getter
    public double getHeight() {
        return this.height;
    }
    // Return the volume of this cylinder
    public double getVolumne() {
        return super.getArea() * height;
    }
    // Override the inherited method to return the surface area
    @Override
    public double getArea() {
        return 2.0 * Math.PI * getRadius() * height;
    }
    // Override the inherited method to describe itself
    @Override
    public String toString() {
        return "Cylinder[height=" + height + "," + super.toString() + "]";
    }
}
```

Does the following declaration produce an error?

```
Circle c1 = new Cylinder(1.1, 2.2);
```

Does the following call produce an error?

```
c1.getRadius();
```

What about these?

```
c1.getHeight();
c1.getVolume();
```

And these? Do they produce an error? If not, what method is executed in each case?

```
c1.toString();
c1.getArea();
```

Exercise 7

Which of the classes in the exercise 3 would you declare as abstract? Why? See video 6 (`06-abstract-classes`) before answering.

Exercise 8

Read, discuss and research the following operation for determining equality of rectangles.

```
public boolean equals(Object o) {
    // self check
    if (this == o)
        return true;
```

```
    // null check
    if (o == null)
        return false;
    // type check and cast
    if (!(o instanceof Rectangle))
        return false;
    Rectangle r = (Rectangle) o;
    // field comparison
    return Objects.equals(base, r.getBase())
            && Objects.equals(width, r.getWidth());
}
```

Exercise 9

Define an interface Measurable that includes a method signature `double measure()` that allows any object of an implementing class to be measured. Then define classes Orange and Building that implement the interface. The measure of an Orange is its weight and that of a building its height. Define a class PlayingWithMeasurableObjects that has a static method minimum for computing the minimum of an array of measurable objects. See video `07-Interfaces` in Blackboard Collaborate, in Canvas, before solving this exercise.

Exercise 10

Using generics, create a class Triple that holds three values of arbitrary types. Include the constructor, getters, setters and `toString` operations. Before solving this exercise, see the video `08-Generics` in Blackboard Collaborate.

Exercise 11

Complete the definition of the following class for dictionaries:

```
public class Dict<K,V> {
    private Pair<K,V>[] d;
    static final int SIZE=10;
    // Constructor
    Dict() {
        d = (Pair<K,V>[]) (new Object [SIZE]);
    }
    // Methods
    public boolean found(K key) {
        return 0;
    }

    public V getValue(K key) {
        return 0;
    }
}
```

Exercise 12

Read, discuss and research the following operation for determining whether two pairs are equal. Compare with the same operation given above for Rectangle.

```
public boolean equals(Object o) {
    // self check
    if (this == o)
        return true;
    // null check
    if (o == null)
        return false;
    // type check and cast
    if (!(o instanceof Pair<?,?>))
        return false;
    Pair<E,F>p = (Pair<E,F>) o;
    // field comparison
```

```
    return Objects.equals(fst, p.getFst())
14              && Objects.equals(snd, p.getSnd());
}
```