Assignment 3

1. Explain polymorphism.

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

2. What is overloading?

Method Overloading is a Compile time polymorphism. In method overloading, more than one method shares the same method name with a different signature in the class.

3. What is overriding?

Method overriding is a run-time polymorphism.

4. What does the final mean in this method: public void doSomething(final Car aCar){}

It means this parameter aCar can't be changed inside the program.

5. Suppose in question 4, the Car class has a method setColor(Color color){…}, inside doSomething method, Can we call aCar.setColor(red);?

Yes.

6. Can we declare a static variable inside a method?

No.

7. What is the difference between interface and abstract class?

An abstract class allows you to create functionality that subclasses can implement or override. An interface only allows you to define functionality, not implement it.

8. Can an abstract class be defined without any abstract methods?

Yes

9. Since there is no way to create an object of abstract class, what's the point of constructors of abstract class?

Constructors is used to initialize fields.

10. What is a native method?

A native method is a Java method (either an instance method or a class method) whose implementation is also written in another programming language such as C/C++

11. What is marker interface?

A marker interface is an interface that has no methods or constants inside it. It provides run-time type information about objects, so the compiler and JVM have additional information about the object.

12. Why to override equals and hashCode methods?

Developers should override both methods in order to achieve a fully working equality mechanism

13. What's the difference beween int and Integer?
Int is primitive type, Integer is object version of int.

14. What is serialization?
Searlization convert an Object to stream that we can send over the network or save it as file or store in DB for later usage.

15. Create List and Map. List A contains 1,2,3,4,10(integer) . Map B contains ("a","1") ("b","2") ("c","10") (key = string, value = string)
Question: get a list which contains all the elements in list A, but not in map B.

```
public static List<Integer> getList(){
      List<Integer> listA = Arrays.asList(1,2,3,4,10);
      Map<String, String> mapB = new HashMap<>();
      mapB.put("a", "1");
      mapB.put("b", "2");
      mapB.put("c", "10");
      List<Integer> list = new LinkedList<>();
      for(Integer elem : listA){
         if(!mapB.containsValue(elem.toString())){
            list.add(elem);
         }
      }
      return list;
   }
```

16. Implement a group of classes that have common behavior/state as Shape. Create Circle, Rectangle and Square for now as later on we may need more shapes. They should have the ability to calculate the area. They should be able to compare using area. Please write a program to demonstrate the classes and comparison. You can use either abstract or interface. Comparator or Comparable interface.

```
interface Shape extends Comparable<Shape>{
   double getArea();
}

class Circle implements Shape{
   private final double radius;
```

```
    public double getRadius() {
        return radius;
    }

    public Circle(double radius){
        this.radius = radius;
    }

    @Override
    public double getArea() {
        return radius * radius * Math.PI;
    }

    @Override
    public int compareTo(Shape o) {
        Double selfArea = this.getArea();
        Double oArea = o.getArea();
        return selfArea.compareTo(oArea);
    }
}

class Rectangle implements Shape{
    private final double width;
    private final double length;

    public double getLength() {
        return length;
    }

    public double getWidth() {
        return width;
    }

    public Rectangle(double width, double length){
        this.width = width;
        this.length = length;
    }
    @Override
    public double getArea() {
        return this.width * this.length;
    }

    @Override
```

```
    public int compareTo(Shape o) {
        Double selfArea = this.getArea();
        Double oArea = o.getArea();
        return selfArea.compareTo(oArea);
    }
}

class Square implements Shape{
    private final double side;

    public Square(double side) {
        this.side = side;
    }

    public double getSide() {
        return side;
    }

    @Override
    public double getArea() {
        return side * side;
    }

    @Override
    public int compareTo(Shape o) {
        Double selfArea = this.getArea();
        Double oArea = o.getArea();
        return selfArea.compareTo(oArea);
    }
}
```