**Assignment 3**

1. Explain polymorphism.
   Polymorphism is the ability of an object to take many forms. For example, class A is inherited from class B, then we can override the methods in B so that the methods perform differently. Also, we can overload functions, which means we can create functions with same name and different signatures.

2. What is overloading?
   Overloading means different methods can have the same name, but different signatures, where the signature can differ by the number of input parameters or type of input parameters or both.

3. What is overriding?
   Overriding means a subclass can provide a specified implementation of a method that is already provided by one of its super-classes.

4. What does the final mean in this method: public void doSomething(**final** Car aCar){}
   It means that the variable cannot be reassigned inside the method.
   You can change the arribution of the aCar but cannot reassign it.

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

6. Can we declare a static variable inside a method?
   No. static means it is a variable or method of a class that belongs to the class but not an object of the class type.

7. What is the difference between interface and abstract class?
   An interface can only have abstract methods, and default methods.
   An abstract class can have both abstract methods and concrete methods.
   A class can implement multiple interfaces but can only extend one abstract class.

8. Can an abstract class be defined without any abstract methods?
   Yes. But this way, you cannot initiate it.

9. Since there is no way to create an object of abstract class, what's the point of constructors of abstract class?
   The main purpose of the constructor is to initialize the newly created object. In abstract class, we have an instance variable, abstract methods, and non-abstract methods. We need to initialize the non-abstract methods and instance variables, therefore abstract classes have a constructor.

10. What is a native method?

Native methods are the methods that start in a language other than Java. They can access system-specific functions and APIs that are not available directly in Java.

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.
    Ex: Deletable

12. Why to override equals and hashCode methods?
    If we don't, it will be a violation of the general contract of Object.hashCode(), which will prevent your class from functioning properly in conjunction with all hash-based collections, including HashMap, HashSet, and Hashtable.
    We use hashCode() to find the right bucket, and use equals() to search bucket for the right element.

13. What's the difference between int and Integer?
    int is a primitive type and Integer is the wrapper class for int. Integer gives us more flexibility in storing, converting and manipulating an int data.
    We cannot cast a String containing an integer only to an int variable directly.

14. What is serialization?
    Serialization is the process of converting an object into a byte stream.

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.

```java
package Day4.Homework;

import java.util.*;

public class ListMap {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(10);
        Map<String, String> map = new HashMap<>();
        map.put("a", "1");
        map.put("b", "2");
        map.put("c", "10");
        List<Integer> result = new ArrayList<>();
        result = inListNotInMap(list, map);
        System.out.println(result.toString());
    }

    public static List<Integer> inListNotInMap(List list, Map map) {
```

```
        if (list == null || list.isEmpty()) {
            return null;
        }
        List<Integer> result = new ArrayList<>();
        for (int i = 0; i < list.size(); i++) {
            Integer cur = (Integer) list.get(i);
            if (!map.containsValue(cur.toString())) {
                result.add((Integer) list.get(i));
            }
        }
        return result;
    }
}
```

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.

```
package Day4.Homework;

public abstract class Shape implements Comparable<Shape> {
    protected double area;

    public double getArea() {
        return area;
    }

    @Override
    public int compareTo(Shape s) {
        if (area == s.area) {
            return 0;
        }
        return area > s.area ? 1 : -1;
    }
}

class Circle extends Shape {
    double radius;

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

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

class Rectangle extends Shape {
```

```
    private double length;
    private double width;

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

    public double getLength() {
        return length;
    }

    public void setLength(double length) {
        this.length = length;
    }

    public double getWidth() {
        return width;
    }

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

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

class Square extends Shape {
    private double length;

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

    public double getLength() {
        return length;
    }

    public void setLength(double length) {
        this.length = length;
    }

    @Override
    public double getArea() {
        this.area = length * length;
        return area;
    }
}
```