Assignment 3 – Wei Sun, May 6th

1. Explain polymorphism.

Polymorphism refers to the ability of a class to provide different implementations of a method, depending on the type of object that is passed to the method.

2. What is overloading?

Overloading is one type of Polymorphism, normally happened in the same class. For the same method, different implementations can be provided with different types of parameters, and different return types.

3. What is overriding?

Overriding is one type of Polymorphism, normally happened in the sub-class. For the same method, the sub-class can provide different implementation from super-class, the method name, parameters, and return type should all be the same.

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

The keyword final can **stop the reassignment** of the parameter object, in this case, "aCar" cannot be reassigned.

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

Yes (if red is a valid object of class Color), as the setColor() method only changed the field variables of aCar without reassigning the aCar to another object.

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

No, it is **not allowed** to declare static variable inside a method. Inside method all variables are local variables that has no existence outside this. Only field variables and methods can be static.

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

Abstract class is a class with keyword "abstract" before the class name. An abstract class can have abstract methods and concrete methods. Abstract methods are methods declared in the class without implementations, the method name should have the keyword "abstract" either.

Interface is a completely abstract class. It only declares methods in it without implementations.

Besides, a class can only extend one class, while it can implement many interfaces.

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

Yes, abstract class can have no abstract method. But we cannot instantiate objects from abstract class, even it has no abstract classes.

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

In abstract class, we have instance variables, abstract methods, and non-abstract methods. We need to initialize the non-abstract methods and instance variable, so abstract classes have a constructor.

10. What is a native method?

Native methods are Java methods that start in a language other than Java. Native methods can access system-specific functions and APIs that are in other programming languages such as C, and C++. The JNI (Java Native Interface) framework lets native methods use Java objects in the same way that Java code used these objects.

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?

We must override hashCode() in every class that overrides equals(). Failure to do so will result in a violation of the general contract for Object.hashCode(), which will prevent your class from functioning properly in conjunction with all hash-based collections, including HashMap, HashSet, and Hashtable.

13. What's the difference between int and Integer?

Integer and int are both used to store Integer data type.

-The major difference between int and Integer is that int is a primitive data type while Integer is a Wrapper class.

-An Integer can be used as an argument to a method that requires an object, whereas int can be used as an argument to a method that requires an integer value, that can be used for arithmetic expression.

-An int datatype helps to store integer values in memory whereas Integer helps to convert int into an object and to convert an object into an int.

-The variable of int type is mutable unless it is marked as final and the Integer class contains one int value and is immutable.

14. What is serialization?

Serialization is a mechanism of converting the state of an object into a byte stream. Similarly, deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

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
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> eleInListNotInMap = solve(list, map);
    System.out.println(eleInListNotInMap);
}

private List<Integer> solve(List<Integer> list, Map<String, String> map) {
    Set<Integer> inMap = new HashSet<>();
    for(String key : map.keySet()) {
        inMap.add(Integer.parseInt(map.get(key)));
    }
    List<Integer> res = new ArrayList<>();
    for(Integer ele : list) {
        if(!inMap.contains(ele)) {
            res.add(ele);
        }
    }
    return res;
}
```

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.

```
public interface Shape extends Comparable<Shape> {
    int calArea();
}
```

```
class Circle implements Shape {
    private int radius;
    public Circle(int r) {
        this.radius = r;
    }

    @Override
    public int calArea() {
        return this.radius * this.radius;
    }

    @Override
    public int compareTo(Shape o) {
        return this.calArea() - o.calArea();
    }
}
```

```
public class Rectangle implements Shape {
    private int length, height;

    public Rectangle(int length, int height) {
        this.length = length;
        this.height = height;
    }

    @Override
    public int calArea() {
        return this.length * this.height;
    }

    @Override
    public int compareTo(Shape o) {
        return this.calArea() - o.calArea();
    }
}
```

```java
public class Square implements Shape {
    private int length;

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

    @Override
    public int calArea() {
        return this.length * this.length;
    }

    @Override
    public int compareTo(Shape o) {
        return this.calArea() - o.calArea();
    }
}
```