Department of Computer Science

Final Term Exam Spring 2020

Subject: **Object Oriented Programming**

BS (CS,SE) Instructor: M.Ayub Khan

Name : Haris Ur Rehman (ID # 16043) Max Marks: 50

Q1. a. Why access modifiers are used in java, explain in detail Private and Default access modifiers?

Ans

Access modifier hide its visibility where it is not required to be visible and shows where it is required for instance, we make function private which can only be used inside that function.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Private:

The private access modifier is accessible only within the class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

Default:

If you don't use any modifier, it is treated as default by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But it is more restrictive than protected, and public.

b. Write a specific program of the above-mentioned access modifiers in java.

```
//save by A.java
package pack;
class A{
void msg(){
System.out.println("Hello");
}
}

//save by B.java
package mypack;
import pack.*;
class B{
public static void main(String args[]){
A obj = new A();//Compile Time Error
obj.msg();//Compile Time Error
}
}

//save by B.java
package mypack;
import pack.*;
class B{
public static void main(String args[]){
A obj = new A();//Compile Time Error
        obj.msg();//Compile Time Error
}
}
```

Q2.

a. Explain in detail Public and Protected access modifiers?

Ans.

1. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.
2. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package

   Protected data member and method are only accessible by the classes of the same package and the subclasses present in any package. You can also say that the protected access modifier is similar to default access modifier with one exception that it has visibility in sub classes.
   Classes cannot be declared protected. This access modifier is generally used in a parent child relationship.

b. Write a specific program of the above-mentioned access modifiers in java.

Public:

```
//save by A.java

package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
```

```
        //save by B.java

package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Protected:

//save by A.java

```
package pack;
public class A{
protected void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;

class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

Q3. a. What is inheritance and why it is used, discuss in detail?

Inheritance:

Inheritance is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

Its usage

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

b. Write a program using Inheritance class on Animal in java.

```
public class Animal {

    private boolean vegetarian;

    private String eats;

    private int noOfLegs;

    public Animal(){}

    public Animal(boolean veg, String food, int legs){

            this.vegetarian = veg;

            this.eats = food;

            this.noOfLegs = legs;

    }

    public boolean isVegetarian() {

            return vegetarian;

    }

    public void setVegetarian(boolean vegetarian) {

            this.vegetarian = vegetarian;

    }
```

```
	public String getEats() {
		return eats;
	}
	public void setEats(String eats) {
		this.eats = eats;
	}
	public int getNoOfLegs() {
		return noOfLegs;
	}
	public void setNoOfLegs(int noOfLegs) {
		this.noOfLegs = noOfLegs;
	}

}
public class Cat extends Animal{
	private String color;
	public Cat(boolean veg, String food, int legs) {
		super(veg, food, legs);
		this.color="White";
	}
	public Cat(boolean veg, String food, int legs, String color){
		super(veg, food, legs);
		this.color=color;
	}
	public String getColor() {
		return color;
	}
```

```
public void setColor(String color) {
        this.color = color;
    }
}
```

Q4. a. What is polymorphism and why it is used, discuss in detail?

Ans:

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

Example

Let us look at an example.

```
public interface Vegetarian{}

public class Animal{}

public class Deer extends Animal implements Vegetarian{}
```

Now, the Deer class is considered to be polymorphic since this has multiple inheritance. Following are true for the above examples −

- A Deer is an Animal
- A Deer is a Vegetarian
- A Deer is a Deer
- A Deer is an Object

When we apply the reference variable facts to a Deer object reference, the following declarations are legal −

Example

```
Deer d = new Deer();

Animal a = d;

Vegetarian v = d;

Object o = d;
```

All the reference variables d, a, v, o refer to the same Deer object in the heap.

b. **Write a program using polymorphism in a class on Employee in java.**
Example of Polymorphism

I will show you how the behavior of overridden methods in Java allows you to take advantage of polymorphism when designing your classes.

We already have discussed method overriding, where a child class can override a method in its parent. An overridden method is essentially hidden in the parent class, and is not invoked unless the child class uses the super keyword within the overriding method.

```
/* File name : Employee.java */

public class Employee {

  private String name;

  private String address;

  private int number;

  public Employee(String name, String address, int number) {

    System.out.println("Constructing an Employee");

    this.name = name;

    this.address = address;

    this.number = number;

  }

  public void mailCheck() {

    System.out.println("Mailing a check to " + this.name + " " + this.address);

  }

  public String toString() {

    return name + " " + address + " " + number;

  }
```

```
   public String getName() {
     return name;
   }
   public String getAddress() {
     return address;
   }
   public void setAddress(String newAddress) {
     address = newAddress;
   }
   public int getNumber() {
     return number;
   }
}

/* File name : Salary.java */
public class Salary extends Employee {
   private double salary; // Annual salary

   public Salary(String name, String address, int number, double salary) {
     super(name, address, number);
     setSalary(salary);
   }

   public void mailCheck() {
     System.out.println("Within mailCheck of Salary class ");
     System.out.println("Mailing check to " + getName()
     + " with salary " + salary);
   }
```

```
      public double getSalary() {
        return salary;
      }
      public void setSalary(double newSalary) {
        if(newSalary >= 0.0) {
          salary = newSalary;
        }
      }

  public double computePay() {
    System.out.println("Computing salary pay for " + getName());
    return salary/52;
  }
}
      /* File name : VirtualDemo.java */
      public class VirtualDemo {

        public static void main(String [] args) {
          Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);
          Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);
          System.out.println("Call mailCheck using Salary reference --");
          s.mailCheck();
          System.out.println("\n Call mailCheck using Employee reference--");
          e.mailCheck();
        }
}
```

Q5. a. Why abstraction is used in OOP, discuss in detail?

Abstraction:

abstraction is the process of hiding certain details and showing only essential information to the user.

Abstraction can be achieved with either abstract classes or interfaces

The abstract keyword is a non-access modifier, used for classes and methods:

- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods:

```
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");
  }
}
```

From the example above, it is not possible to create an object of the Animal class:

```
Animal myObj = new Animal(); // will generate an error
```

To access the abstract class, it must be inherited from another class. Let's convert the Animal class we used in the Polymorphism chapter to an abstract class:

b. Write a program on abstraction in java.

```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
```

```
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class MyMainClass {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Output

```
The pig says: wee wee
Zzz
```