IQRA National
University, Peshawar

NAME: SHAHID ABBAS

ID: 16694

SEMESTER: 2nd SPRING BS(CS)

SUBJECT: OOPS

INSTRUCTOR: M. Ayub Khan

ASSIGNMENT: FINAL TERM #1

DATE: 29/06/2020

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

Java Access Modifiers:

You must have seen public, private and protected keywords while practicing java programs, these are called access modifiers. An access modifier restricts the access of a class, constructor, data member and method in another class.

Default access modifier:

When we do not mention any access modifier, it is called default access modifier. The scope of this modifier is limited to the package only. This means that if we have a class with the default access modifier in a package, only those classes that are in this package can access this class. No other class outside this package can access this class. Similarly, if we have a default method or data member in a class, it would not be visible in the class of another package. Let's see an example to understand this:

Example:

Variables and methods can be declared without any modifiers, as in the following

```
String version = "1.6.1";

boolean processOrder() {

  return true;

}
```

Private Access Modifier:

Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.

Private access modifier is the most restrictive access level. Class and interfaces cannot be private.

Variables that are declared private can be accessed outside the class, if public getter methods are present in the class.

Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.

Example:

The following class uses private access control −

```
public class Logger {
   private String format;
  public String getFormat() {
      return this.format;
   }
  public void setFormat(String format) {
      this.format = format;
   }
}
```

Here, the *format* variable of the Logger class is private, so there's no way for other classes to retrieve or set its value directly.

So, to make this variable available to the outside world, we defined two public methods: *getFormat()*, which returns the value of format, and *setFormat(String)*, which sets its value

Q1. b. Write a specific program of the above mentioned access modifiers in java.

Private Access modifier Example:

```
class A{
private A(){}
void msg(){
System.out.println("Hello java");
}
}

public class Simple{
```

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

Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
package pack;
class A{
  void msg(){System.out.println("Hello");}
}
//////////////
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package

Q2. a. Explain in detail Public and Protected access modifiers?

Public Access Modifier:

A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.

However, if the public class we are trying to access is in a different package, then the public class still needs to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

Example:

The following function uses public access control.

```
public static void main(String[] arguments) {

  // ...

}
```

The main() method of an application has to be public. Otherwise, it could not be called by a Java interpreter (such as java) to run the class.

Protected Access Modifier:

Variables, methods, and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.

The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.

Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

Example:

The following parent class uses protected access control, to allow its child class override *openSpeaker()* method –

```
class AudioPlayer {

  protected boolean openSpeaker(Speaker sp) {

    // implementation details

  }

}
```

```
class StreamingAudioPlayer extends AudioPlayer {

  boolean openSpeaker(Speaker sp) {

    // implementation details

  }

}
```

Here, if we define openSpeaker() method as private, then it would not be accessible from any other class other than *AudioPlayer*. If we define it as public, then it would become accessible to all the outside world. But our intention is to expose this method to its subclass only, that's why we have used protected modifier.

Q2.b. Write a specific program of the above mentioned access modifiers in java.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
package pack;

public class A{

protected void msg(){

System.out.println("Hello");

}

}

//////////////////

package mypack;

import pack.*;


class B extends A{

  public static void main(String args[]){

   B obj = new B();
```

```
   obj.msg();
  }
}
```

Example of public access modifier

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

//////////////////////

```
package mypack;
import pack.*;

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

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

Inheritance:

It is a process by which one class acquires the properties (data members) and functionalities(methods) of another class is called inheritance. The aim of inheritance is to provide the reusability of code so that a class has to write only the unique features and rest of the common properties and functionalities can be extended from the another class.

Child Class:
The class that extends the features of another class is known as child class, sub class or derived class.

Parent Class:
The class whose properties and functionalities are used(inherited) by another class is known as parent class, super class or Base class.

Syntax: Inheritance in Java:

To inherit a class, we use extends keyword. Here class XYZ is child class and class ABC is parent class. The class XYZ is inheriting the properties and methods of ABC class.

```
class XYZ extends ABC
{
}
```

Use of inheritance:

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

Types of inheritance:

Single Inheritance:

refers to a child and parent class relationship where a class extends the another class.

Multilevel Inheritance:

refers to a child and parent class relationship where a class extends the child class. For example, class C extends class B and class B extends class A.

Hierarchical Inheritance:

refers to a child and parent class relationship where more than one classes extend the same class. For example, classes B, C & D extends the same class A.

Multiple Inheritance:

refers to the concept of one class extending more than one classes, which means a child class has two parent classes. For example, class C extends both classes A and B. Java doesn't support multiple inheritance,

Q3.b. Write a program using Inheritance class on Animal in java.

```
public class Animal {
    public Animal() {
        System.out.println("A new animal has been created!");
    }

    public void sleep() {
        System.out.println("An animal sleeps...");
    }

    public void eat() {
        System.out.println("An animal eats...");
    }
}

public class Bird extends Animal {
    public Bird() {
```

```
        super();
        System.out.println("A new bird has been created!");
    }

    @Override
    public void sleep() {
        System.out.println("A bird sleeps...");
    }

    @Override
    public void eat() {
        System.out.println("A bird eats...");
    }
}

public class Dog extends Animal {
    public Dog() {
        super();
        System.out.println("A new dog has been created!");
    }

    @Override
    public void sleep() {
        System.out.println("A dog sleeps...");
    }

    @Override
    public void eat() {
```

```
        System.out.println("A dog eats...");
    }
}
```

```
public class MainClass {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Bird bird = new Bird();
        Dog dog = new Dog();

        System.out.println();

        animal.sleep();
        animal.eat();

        bird.sleep();
        bird.eat();

        dog.sleep();
```

```
    dog.eat();
  }
```

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

Polymorphism:

Polymorphism literally means "having multiple forms." In computer science, polymorphism describes a code element (method, class, or data type) that can have or transform into multiple forms. ... Method overloading (that is, when multiple methods have the same name) is an example of overloading polymorphism.

In a programming language that exhibits polymorphism, objects of classes belonging to the same hierarchical tree (inherited from a common base class) may possess functions bearing the same name, but each having different behaviors.

Types of polymorphism:

There are 2 types of polymorphism which are commonly mentioned.

Dynamic Polymorphism:

This is also mentioned as Run-Time polymorphism, Dynamic binding, Run-Time binding, Late binding and Method overriding. Here having many forms is happening in different classes.

Static Polymorphism:

This is also mentioned as Compile-Time polymorphism, Static binding, Compile-Time binding, Early binding and Method overloading. Here having many forms is happening in the same class.

Importance of polymorphism in java:

The good reason for why **Polymorphism** is **need** in **java** is because the concept is extensively used in implementing inheritance. It plays an important role in allowing objects having different internal structures to share the same external interface.

b. Write a program using polymorphism in a class on Employee in java

Class on Employee:

```
// Employee abstract superclass.


  public abstract class Employee
  {
    private String firstName;
     private String lastName;
     private String socialSecurityNumber;


   // three-argument constructor
    public Employee( String first, String last, String ssn )
    {
      firstName = first;
      lastName = last;
      socialSecurityNumber = ssn;
    } // end three-argument Employee constructor


    // set first name
    public void setFirstName( String first )
    {
      firstName = first; // should validate
    } // end method setFirstName


    // return first name
    public String getFirstName()
    {
```

```
   return firstName;
} // end method getFirstName

// set last name
public void setLastName( String last )
{
   lastName = last; // should validate
} // end method setLastName
// return last name
public String getLastName()
{
   return lastName;
} // end method getLastName

// set social security number
public void setSocialSecurityNumber( String ssn )
{
   socialSecurityNumber = ssn; // should validate
} // end method setSocialSecurityNumber

// return social security number
public String getSocialSecurityNumber()
{
   return socialSecurityNumber;
} // end method getSocialSecurityNumber

// return String representation of Employee object
@Override
public String toString()
```

```
    {
      return String.format( "%s %s\nsocial security number: %s",
        getFirstName(), getLastName(), getSocialSecurityNumber() );
    } // end method toString

    // abstract method overridden by concrete subclasses
    public abstract double earnings(); // no implementation here
  } // end abstract class Employee
```

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

Abstraction:

Abstraction is the process of hiding the internal details of an application from the outer world. Abstraction is used to describe things in simple terms. It's used to create a boundary between the application and the client programs.

Abstraction in Real Life:

Abstraction is present in almost all the real life machines.

- Your car is a great example of abstraction. You can start a car by turning the key or pressing the start button. You don't need to know how the engine is getting started, what all components your car has. The car internal implementation and complex logic is completely hidden from the user.
- We can heat our food in Microwave. We press some buttons to set the timer and type of food. Finally, we get a hot and delicious meal. The microwave internal details are hidden from us. We have been given access to the functionality in a very simple manner.

Abstraction in OOPS:

Objects are the building blocks of Object-Oriented Programming. An object contains some properties and methods. We can hide them from the outer world through access modifiers. We can provide access only for required functions and properties to the other programs. This is the general procedure to implement abstraction in OOPS.

Types of abstraction:

There are two types of abstraction.

1. Data Abstraction
2. Process Abstraction

Data Abstraction:

When the object data is not visible to the outer world, it creates data abstraction. If needed, access to the Objects' data is provided through some methods.

Process Abstraction:

We don't need to provide details about all the functions of an object. When we hide the internal implementation of the different functions involved in a user operation, it creates process abstraction.

Abstraction in Java:

Abstraction in Java is implemented through interface and abstract classes. They are used to create a base implementation or contract for the actual implementation classes.

(b) Write a program on abstraction in java.

program where the Car functions will be used.

```
package com.journaldev.oops.abstraction;


public class CarTest {


	public static void main(String[] args) {
		Car car1 = new ManualCar();
		Car car2 = new AutomaticCar();


		car1.turnOnCar();
		car1.turnOffCar();
		System.out.println(car1.getCarType());


		car2.turnOnCar();
		car2.turnOffCar();
		System.out.println(car2.getCarType());


	}


}
```