FINAL TERM EXAMINATION

2020

NAME	MUHAMMAD JUNAID
ID#	16027
SECTION#	A
DEPARTEMENT	BS (SE)
PAPER	OBJECT ORIENTED PROGRAMMING
SEMESTER	2TH
DATE	29/06/2020

INSTRUCTOR

MR SIR M AYUB KHAN

QUESTION#1

a)

Why access modifiers are used in java, explain in detail Private and Default Access modifiers?

ANSWER

- 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.

- A Java access modifier specifies which classes can access a given class and its fields, constructors and methods. Access modifiers can be specified separately for a class, its constructors, fields and methods.

1) PRIVATE ACCESS MODIFIERS

- The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N

- 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.

```
1.  class A{
2.  private int data=40;
3.  private void msg(){System.out.println("Hello java");}
4.  }
5.
6.  public class Simple{
7.   public static void main(String args[]){
8.     A obj=new A();
9.     System.out.println(obj.data);//Compile Time Error
10.    obj.msg();//Compile Time Error
11.   }
12. }
```

- **Role of Private Constructor**

- If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

```
1. class A{
2. private A(){}//private constructor
3. void msg(){System.out.println("Hello java");}
4. }
5. public class Simple{
6.  public static void main(String args[]){
7.    A obj=new A();//Compile Time Error
8.  }
9. }
```

- **Note:** **A class cannot be private or protected except nested class.**

2) DEFAULT ACCESS MODIFIER

- 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.

- **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.

```
1. //save by A.java
2. package pack;
3. class A{
4.   void msg(){System.out.println("Hello");}
5. }
```

```
1. //save by B.java
2. package mypack;
3. import pack.*;
4. class B{
5.   public static void main(String args[]){
6.    A obj = new A();//Compile Time Error
7.    obj.msg();//Compile Time Error
8.   }
9. }
```

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

b) Write a specific program of the above mentioned access modifiers in java?

ANSWER

- **PROGRAM FOR PRIVATE**

```
//private modifier
   package p1;

     class A
  {
private void display()
     {
 System.out.println("GeeksforGeeks");
 }
 }

 class B
 {
 public static void main(String args[])
 {
 A obj = new A();
//trying to access private method of another class
  obj.display();
 }
 }
```

- **PROGRAM FOR DEFAULT**

```
//Class is having Default access modifier
class Geek
{
    void display()
       {
           System.out.println("Hello World!");
       }
}


//Java program to illustrate error while
//using class from different package with
//default modifier
package p2;
import p1.*;

//This class is having default access modifier
class GeekNew
{
    public static void main(String args[])
       {
          //accessing class Geek from package p1
          Geeks obj = new Geek();

          obj.display();
       }
}
```

QUESTION#2

a) Explain in detail Public and Protected access modifiers?

ANSWER

- **PROTECTED ACCESS MODIFIER**: The protected access modifier is specified using the keyword **protected**.

- The methods or data members declared as protected are **accessible within same package or sub classes in different package.**

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

```
1. //save by A.java
2. package pack;
3. public class A{
4. protected void msg(){System.out.println("Hello");}
5. }
```

```
1. //save by B.java
2. package mypack;
3. import pack.*;
4.
5. class B extends A{
6.   public static void main(String args[]){
7.    B obj = new B();
8.    obj.msg();
9.   }
10. }
```

- **PUBLIC ACCESS MODIFIER**

The public access modifier is specified using the keyword **public**.

- The public access modifier has the **widest scope** among all other access modifiers.
- Classes, methods or data members which are declared as public are **accessible from everywhere**

In the program. There is no restriction on the scope of a public data members.

- The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.
- **Example of public access modifier**

```
1. //save by A.java
2.
3. package pack;
4. public class A{
5. public void msg(){System.out.println("Hello");}
6. }
```

```
1. //save by B.java
2.
3. package mypack;
4. import pack.*;
5.
6. class B{
7.   public static void main(String args[]){
8.    A obj = new A();
```

b) Write a specific program of the above mentioned access modifiers in java?

ANSWER

- **PROGRAM FOR PROTECTED A MODEFIER**

In this example, we will create two packages p1 and p2. Class A in p1 is made public, to access it in p2. The method display in class A is protected and class B is inherited from class A and this protected method is then accessed by creating an object of class B.

```
//protected modifier
package p1;

//Class A
public class A
{
   protected void display()
    {
        System.out.println("GeeksforGeeks");
    }
}
```

```
//Java program to illustrate
//protected modifier
package p2;
import p1.*; //importing all classes in package p1

//Class B is subclass of A
class B extends A
{
   public static void main(String args[])
   {
       B obj = new B();
       obj.display();
   }

}
```

- **PROGRAM FOR PUBLIC A MODEFIER**

In this example, we will create two packages p1 and p2. Class A in p1 is made public, to access it in p2. The method display in class A is protected and class B is inherited from class A and this protected method is then accessed by creating an object of class B.

```
//Java program to illustrate
//protected modifier
package p1;

//Class A
public class A
{
   protected void display()
    {
        System.out.println("GeeksforGeeks");
    }
}
```

```
//Java program to illustrate
//protected modifier
package p2;
import p1.*; //importing all classes in package p1

//Class B is subclass of A
class B extends A
{
   public static void main(String args[])
   {
       B obj = new B();
       obj.display();
   }

}
```

QUESTION#3

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

ANSWER

- Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.
- The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

- ## extends Keyword

Extends is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

Syntax

```
class Super {
   .....
   .....
}
class Sub extends Super {
   .....
   .....
}
```

- ## Sample Code

Following is an example demonstrating Java inheritance. In this example, you can observe two classes namely Calculation and My Calculation.

Using extends keyword, the My Calculation inherits the methods addition() and Subtraction() of Calculation class.

Copy and paste the following program in a file with name My_Calculation.java.

- **Example**

```
class Calculation {
   int z;

   public void addition(int x, int y) {
      z = x + y;
      System.out.println("The sum of the given numbers:"+z);
   }

   public void Subtraction(int x, int y) {
      z = x - y;
      System.out.println("The difference between the given numbers:"+z);
   }
}

public class My_Calculation extends Calculation {
   public void multiplication(int x, int y) {
      z = x * y;
      System.out.println("The product of the given numbers:"+z);
   }

   public static void main(String args[]) {
      int a = 20, b = 10;
      My_Calculation demo = new My_Calculation();
      demo.addition(a, b);
      demo.Subtraction(a, b);
      demo.multiplication(a, b);
   }
}
```

- The Superclass reference variable can hold the subclass object, but using that variable you can access only the members of the superclass, so to access the members of both classes it is recommended to always create reference variable to the subclass.

- If you consider the above program, you can instantiate the class as given below. But using the superclass reference variable (**cal** in this case) you cannot call the method **multiplication()**, which belongs to the subclass My_Calculation.

```
Calculation demo = new My_Calculation();
       demo.addition(a, b);
       demo.Subtraction(a, b);
```

b) Write a program using Inheritance class on Animal in java?

ANSWER

```
class Animal {

   public void eat() {
      System.out.println("I can eat");
   }

   public void sleep() {
      System.out.println("I can sleep");
   }
}

class Dog extends Animal {
   public void bark() {
      System.out.println("I can bark");
   }
}

class Main {
   public static void main(String[] args) {

      Dog dog1 = new Dog();

      dog1.eat();
      dog1.sleep();

      dog1.bark();
   }
}
```

QUESTION#4

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

ANSWER

- 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-A Animal
- A Deer IS-A Vegetarian
- A Deer IS-A Deer
- A Deer IS-A 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.

- # Virtual Methods

- In this section, 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.

- ## Example

```
/* 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;
   }
}
```

Now suppose we extend Employee class as follows −

```
/* 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;
   }
}
```

Now, you study the following program carefully and try to determine its output −

```
/* 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();
   }
}
```

b) Write a program using polymorphism in a class on Employee in java?

ANSWER

- **POLYMORPHISM CLASS ON EMPLOYEE**

```
/* 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;
   }
}
```

Now suppose we extend Employee class as follows −

```
/* 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;
   }
}
```

Now, you study the following program carefully and try to determine its output −

```
/* 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();
   }
}
```

QUESTIN#5

a) Why abstraction is used in OOP, discuss in detail?

ANSWER

- ## ABSTRACTION

✓ **Abstraction** is the quality of dealing with ideas rather than events. For example, when you consider the case of e-mail, complex details such as what happens as soon as you send an e-mail, the protocol your e-mail server uses are hidden from the user. Therefore, to send an e-mail you just need to type the content, mention the address of the receiver, and click send.

✓ Likewise in Object-oriented programming, abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.

- ## Abstract Class

A class which contains the **abstract** keyword in its declaration is known as abstract class.

- Abstract classes may or may not contain *abstract methods*, i.e., methods without body (public void get();)
- But, if a class has at least one abstract method, then the class **must** be declared abstract.
- If a class is declared abstract, it cannot be instantiated.
- To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

- ### Example

This section provides you an example of the abstract class. To create an abstract class, just use the **abstract** keyword before the class keyword, in the class declaration.

```
/* File name : Employee.java */
   public abstract 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;

   }
```

- # Inheriting the Abstract Class

We can inherit the properties of Employee class just like concrete class in the following way −

- ## Example

```
/* 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;
}
}
```

- # Abstract Methods

If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as an abstract.

- **Abstract** keyword is used to declare the method as abstract.
- You have to place the **abstract** keyword before the method name in the method declaration.
- An abstract method contains a method signature, but no method body.
- Instead of curly braces, an abstract method will have a semi colon (;) at the end.

Following is an example of the abstract method.

- ## Example

```
   public abstract class Employee {
   private String name;
   private String address;
   private int number;

   public abstract double computePay();
// Remainder of class definition
}
```

- Declaring a method as abstract has two consequences −

- The class containing it must be declared as abstract.
- Any class inheriting the current class must either override the abstract method or declare itself as abstract.

Note − eventually, a descendant class has to implement the abstract method; otherwise, you would have a hierarchy of abstract classes that cannot be instantiated.

Suppose Salary class inherits the Employee class, then it should implement the **compute Pay()** method as shown below −

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

   public double computePay() {
      System.out.println("Computing salary pay for " + getName());
      return salary/52;
   }
   // Remainder of class definition
}
```

b) Write a program on abstraction in java?

ANSWER

- ## PROGRAM ON ABSTRACTION IN JAVA

```
abstract class Bike{

Bike(){System.out.println("bike is created");}

 abstract void run();

 void changeGear(){System.out.println("gear changed");}
 }

//Creating a Child class which inherits Abstract class

class Honda extends Bike{

void run(){System.out.println("running safely..");}
}



//Creating a Test class which calls abstract and non-abstract methods

class TestAbstraction2{

public static void main(String args[]){

Bike obj = new Honda();

obj.run();

obj.changeGear();
}
}
```


THANK YOU SIR