Name------------------- Hidayatullah

ID------------------------ 16495

Section---------------- (B)

Department--------- BS Software Engineering

Semester------------- 2nd

Subject---------------- Object Oriented Programming

Examination--------- Final-Term

Date------------------- June 29th, 2020

Teacher--------------- M.Ayub Khan

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

Answer: Modifiers

As the name suggests access modifiers in Java helps to restrict the scope of a class, constructor, variable, and method or data member. There are four types of access modifiers available in java:

Default – No keyword required

Private

Protected

Public

Public

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.

```
//Java program to illustrate
//public modifier
package p1;
public class A
{
  public void display()
    {
       System.out.println("GeeksforGeeks");
    }
}
package p2;
import p1.*;
class B
{
   public static void main(String args[])
    {
       A obj = new A;
       obj.display();
    }
}
```

Output:

```
GeeksforGeeks
```

Important Points:

If other programmers use your class, try to use the most restrictive access level that makes sense for a particular member. Use private unless you have a good reason not to.

Avoid public fields except for constants.

Protected

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 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();
  }

}
```

Output:

```
GeeksforGeeks
```

---

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

1. **PROGRAM:**

```
package h2;

public class hidayatullah {

	protected void display()
	{
		System.out.println("hello world");
	}
	public void message()
	{
		System.out.println("Hello hidayatullah");
	}

}
package p4;

import h2.hidayatullah;

class yousafzai extends hidayatullah {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		yousafzai obj=new yousafzai();
		obj.display();
		yousafzai mss=new yousafzai();
		mss.message();
	}

}

```

OUTPUT

```
Hello world

Hello hidayatullah yousafzai
```

EXPLANATION

The above program how to use protected and access modifier. For this I created a java project which name is h2. After this I create two class one is hidayatullah and other is yousafzai with different packages. In sami class I used two method one method is protected access modifier and the other is public access modifier one name is display and the other is message and both of them is void type which not return anything else. These method access in other class which name is khan by simple calling the method. In the result Hello world and Hello hidayatullah yousafzai is print on the screen.

Entually I run my above program and I got correct result .

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

Answer: Inheritance

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.

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.

Inheritance is a process of defining a new class based on an existing class by extending its common data members and methods.
Inheritance allows us to reuse of code, it improves reusability in your java application.

Note: The biggest advantage of Inheritance is that the code that is already present in base class need not be rewritten in the child class.

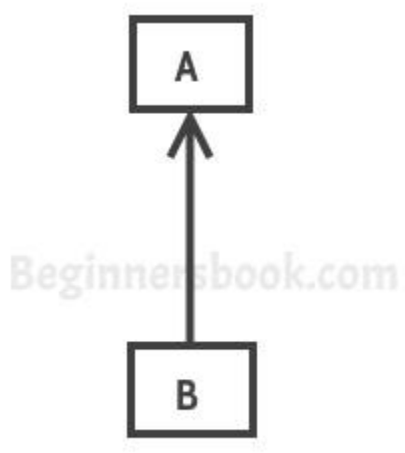
This means that the data members (instance variables) and methods of the parent class can be used in the child class as.

Syntax: Inheritance in Java

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

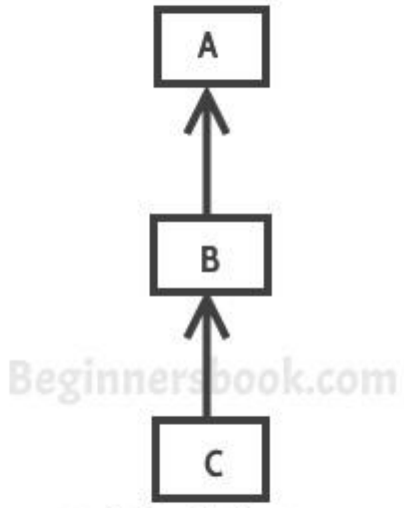
Types of inheritance

To learn types of inheritance in detail, refer: Types of Inheritance in Java.
Single Inheritance**: refers to a child and parent class relationship where a class extends the another class.**



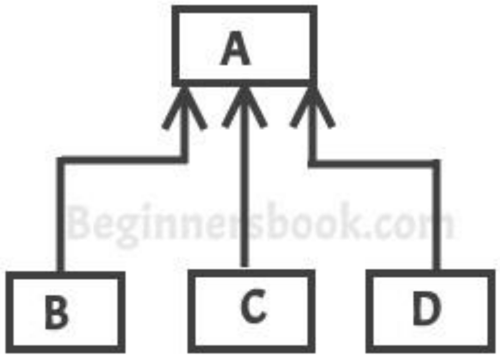
Single Inheritance

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.



Multilevel Inheritance

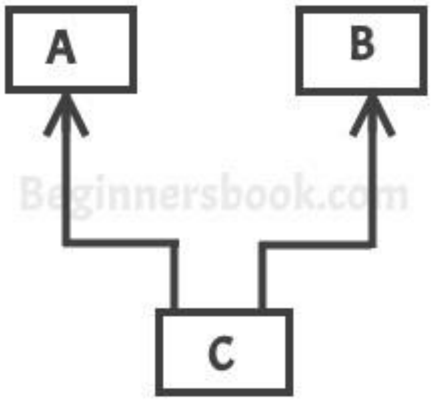
Hierarchical inheritance: refers to a child and parent class relationship where more than one classes extends the same class. For example, classes B, C & D extends the same class A.



Hierarchical Inheritance

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, read more about it here.



Multiple Inheritance

Hybrid inheritance: Combination of more than one types of inheritance in a single program. For example class A & B extends class C and another class D extends class A then this is a hybrid inheritance example because it is a combination of single and hierarchical inheritance.

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

Answer (b) **: Program**

```
class Animal{
void eat(){System.out.println("eating….");}
}
class Lion extends Animal{
void roar(){System.out.println("Roaring.....");}
}
class Babylion extends Lion{
void weep(){System.out.println("weeping….");}
}
class TestInheritance2{
public static void main(String args[]){
Babylion l=new Babylion ();
l.weep();
l.bark();
l.eat();
47. }}
```

Output:

```
weeping.....
Roaring.....
eating.....
```

Explanation

In this program, we have a base class Animal and a sub class lion, baby lion and Testinheritance2 . Then I used output function like

```
void roar(){System.out.println("Roaring.....");}
```

After this I mentioned sub class by the name of baby lion like this

```
class Babylion extends Lion
```

After that again I mentioned output function to display weeping like this

```
void weep(){System.out.println("weeping.....");}
```

Then I mentioned third class and also this I wrote public static void main (String args[])

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

Finally I run my program , I got a correct output

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

Answer: Modifiers

There are two types of modifiers in Java: access modifiers and non-access modifiers. 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.

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

Understanding Java Access Modifiers

Let's understand the access modifiers in Java by a simple table.

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

Private

The private access modifier is accessible only within the class.

Simple example of private access modifier

In this example, we will create two classes A and B within same package p1. We will declare a method in class A as private and try to access this method from class B and see the result.

```
//Java program to illustrate error while
//using class from different package with
//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();
    }
}
```

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:

```
class A{
private A(){}//private constructor
void msg(){System.out.println("Hello java");}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();//Compile Time Error
 }
}
```

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

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.

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.

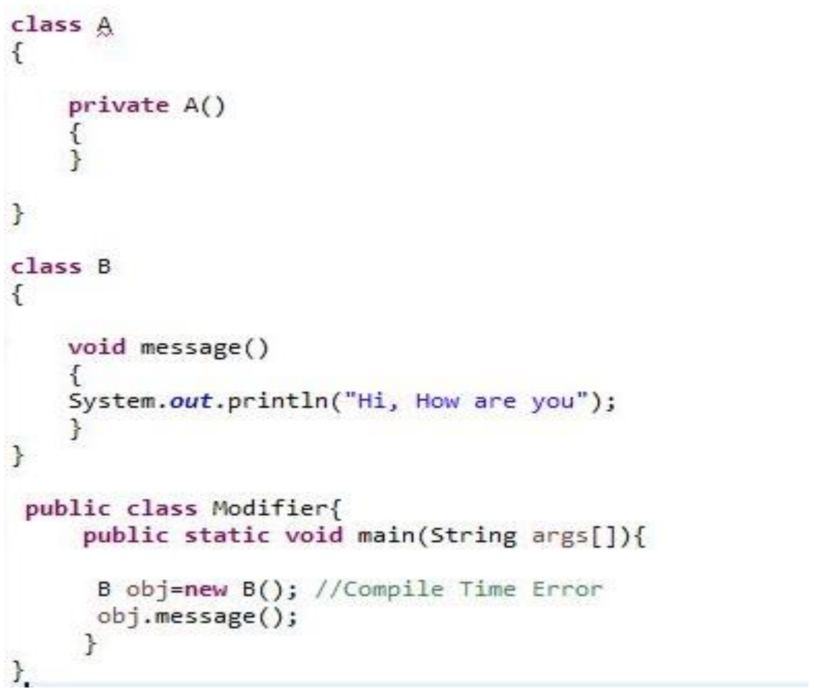
```
//save by A.java
package pack;
class A{
  void msg(){System.out.println("Hello");}
}
```

```
//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
  }
}
```

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

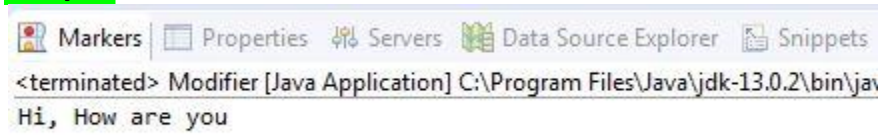
b. Write a specific program of the above mentioned access modifiers in java.

Programs of Private Access Modifier



```
class A
{

    private A()
    {
    }

}

class B
{

    void message()
    {
    System.out.println("Hi, How are you");
    }
}

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

      B obj=new B(); //Compile Time Error
      obj.message();
     }
}.
```

Output



Markers | Properties | Servers | Data Source Explorer | Snippets

```
<terminated> Modifier [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\jav
Hi, How are you
```

Explanation

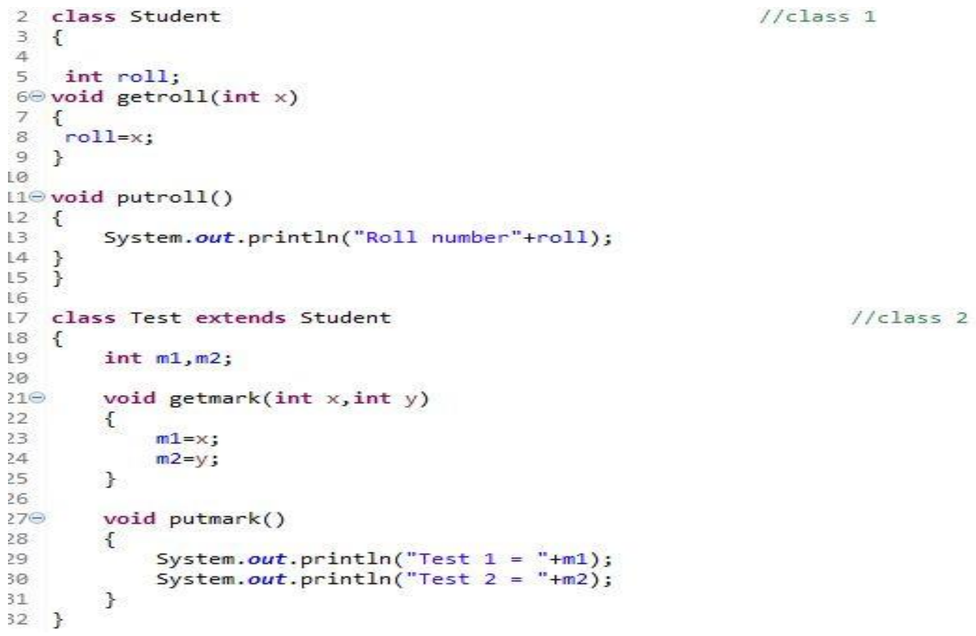
First of all I opened eclipse, then I went to file > new > Java project and I gave the name > finish. After this I clicked on file which I created then new > I clicked on class.

First of I mention main class A then I wrote private a after this then I mentioned class b and also I mentioned void message () then I used output function in order to display " Hi , how are you ".

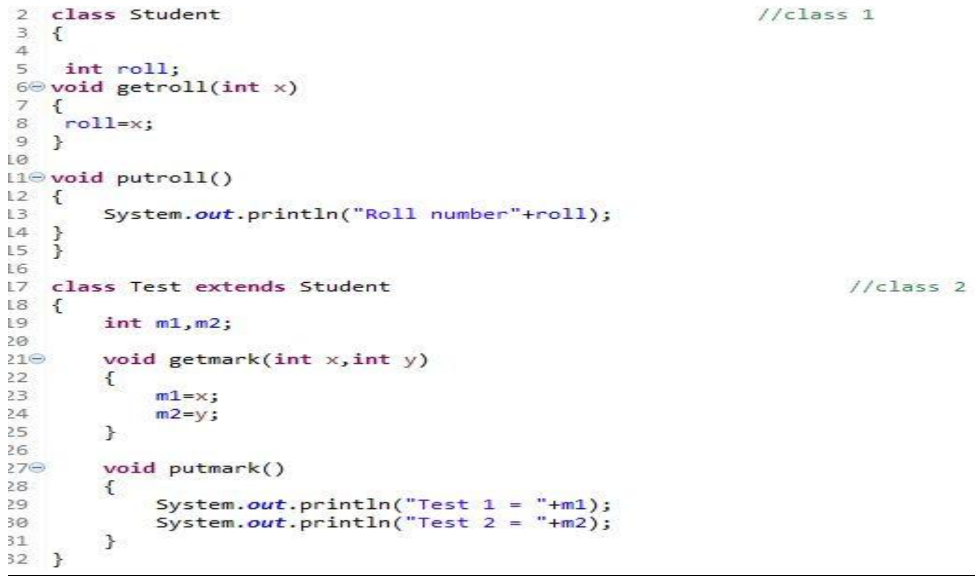
Then I used public class modifier and took an object as by name b.

Finally I run my program and I got correct output.

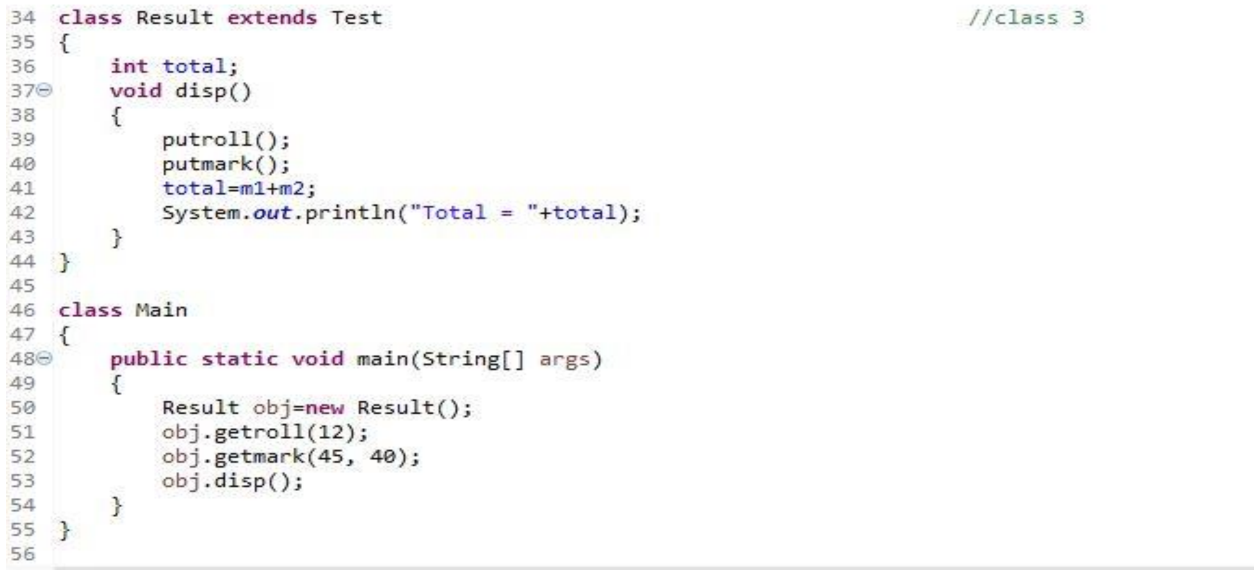
Program of Default Access modifiers:



```
class Student                                                   //class 1
{

 int roll;
void getroll(int x)
{
 roll=x;
}

void putroll()
{
    System.out.println("Roll number"+roll);
}
}

class Test extends Student                                      //class 2
{
    int m1,m2;

    void getmark(int x,int y)
    {
        m1=x;
        m2=y;
    }

    void putmark()
    {
        System.out.println("Test 1 = "+m1);
        System.out.println("Test 2 = "+m2);
    }
}
```

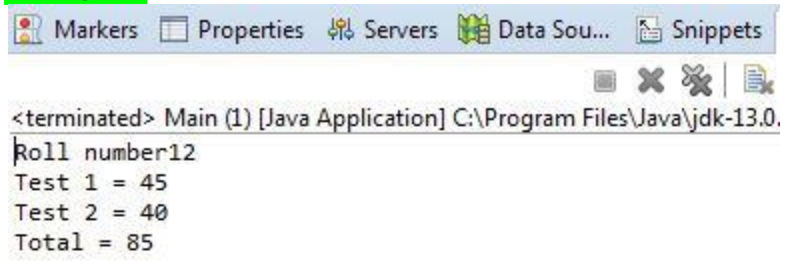


```
class Student                                                   //class 1
{

 int roll;
void getroll(int x)
{
 roll=x;
}

void putroll()
{
    System.out.println("Roll number"+roll);
}
}

class Test extends Student                                      //class 2
{
    int m1,m2;

    void getmark(int x,int y)
    {
        m1=x;
        m2=y;
    }

    void putmark()
    {
        System.out.println("Test 1 = "+m1);
        System.out.println("Test 2 = "+m2);
    }
}
```



```java
class Result extends Test                                           //class 3
{
    int total;
    void disp()
    {
        putroll();
        putmark();
        total=m1+m2;
        System.out.println("Total = "+total);
    }
}

class Main
{
    public static void main(String[] args)
    {
        Result obj=new Result();
        obj.getroll(12);
        obj.getmark(45, 40);
        obj.disp();
    }
}
```

Output



```
<terminated> Main (1) [Java Application] C:\Program Files\Java\jdk-13.0.
Roll number12
Test 1 = 45
Test 2 = 40
Total = 85
```

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

Answer: Polymorphism

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.

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

Program Using Polymorphism

```
package paper2;
/* 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);
}

private void setSalary(double salary2) {
// TODO Auto-generated method stub

}

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 setSalary1(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();
}
}
```

Output

```
Constructing an Employee
Constructing an Employee

Call mailCheck using Salary reference --
Within mailCheck of Salary class
Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--
Within mailCheck of Salary class
Mailing check to John Adams with salary 2400.0
```

Explanation

Here, we instantiate two Salary objects. One using a Salary reference s, and the other using an Employee reference e.
While invoking s.mailCheck(), the compiler sees mailCheck() in the Salary class at compile time, and the JVM invokes mailCheck() in the Salary class at run time.
mailCheck() on e is quite different because e is an Employee reference. When the compiler sees e.mailCheck(), the compiler sees the mailCheck() method in the Employee class.
Here, at compile time, the compiler used mailCheck() in Employee to validate this statement. At run time, however, the JVM invokes mailCheck() in the Salary class.
This behavior is referred to as virtual method invocation, and these methods are referred to as virtual methods. An overridden method is invoked at run time, no matter what data type the reference is that was used in the source code at compile time.

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

Answer: Abstraction

As per dictionary, 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.
In Java, abstraction is achieved using Abstract classes and interfaces.

Or

Abstraction is selecting data from a larger pool to show only the relevant details of the object to the user. Abstraction “shows” only the essential attributes and “hides” unnecessary information. It helps to reduce programming complexity and effort. It is one of the most important concepts of OOPs.

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.**

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 semoi 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 computePay() 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.

Program on Abstraction

```

abstract class A
{
abstract void disp();
void display()
{
System.out.print.ln("Method from A class");
}
class B extends A
{
void disp()
System.out.println("Method difine in B class");
}
}
pbulic class Main {
pbulic static void main(string[] args)
{
B obj=new B();
obj.disp();
obj.display();
}
```

Output

```
Method define in B class
Method from A class
```

Explanation

First of all I mentioned main class by the name of A then I used this abstract void disp(); void display() after I used output function in order to display a message like " Method from A class after that then I used second class by the name of B like this class B extends A

Then I mentioned void disp() after this I used output function again to display second message like “Method define in B class”
At the end I used public class and I mentioned some objects then I run my program so I got an accurate result.

Thank You My Respectable Teacher…!