| | | |
|---|---|---|
| **NAME** | **-** | **M.AWAIS** |
| **SECTION** | **-** | **B** |
| **Semester** | **-** | **2nd** |
| **ID** | **-** | **16378** |
| **DEPART** | **-** | **(SE)** |
| **SUBJECT** | **-** | **Oops** |
| **SUBMITTED TO** | **-** | **M Ayub sir** |

## Q3:- A) What is inheritance and why it is used, discuss in detail ?

**(ANSWER)**

## Inheritance:-

Inheritance is a mechanism in which one class acquires the properties of another class. For example, a child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class. Hence, inheritance facilities reusability and is an important concept of oops.

## Why inheritance is used:-

Inheritance is a term for reusing code by a mechanism of passing down information and behavior from a parent class to a child or sub class. It's pretty easy to understand really. Just like a child can inherit various mannerisms and behaviors from his or her biological parents, in software this same concept holds true. By leveraging the power of inheritance and creating child classes that extend their parent, we can make sub classes with super powers that have everything their parent has and more. Let's take a look at how inheritance works in PHP.

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

(ANSWER)

## Program:-

**Cat:-**

```
 public  class Cat extendsAnimal{

 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;

 }
}
```

Let’s write a simple test class to create Cat object and use some of its methods.

```
 public classAnimalInheritanceTest {

 public static voidmain(String[] args) { Cat cat = new Cat(false, "milk",
4, "black");

 System.out.println("Cat is Vegetarian?" +

 cat.isVegetarian());

 System.out.println("Cat eats " +

 cat.getEats());

 System.out.println("Cat has "+

 cat.getNoOfLegs() + " legs.");

 System.out.println("Cat color is " +

 cat.getColor());
}

}
```

## Java Inheritance Program Output

Output:-

Cat is vegetarian? False.

Cat eats milk.

Cat has 4 legs.

Cat colour is black.

Cat class doesn’t have `getEats()`method but still it works because it’s inherited from Animal class.

## Q4:- A) What is polymorphism and why it is used, discuss in detail ?

**(ANSWER)**

### Polymorphism:-

The word polymorphism is used in various contexts and describes situations in which something occurs in several different forms. In computer science, it describes the concept that objects of different types can be accessed through the same interface. Each type can provide its own, independent implementation of this interface. It is one of the core concepts of object-oriented programming (OOP).

If you're wondering if an object is polymorphic, you can perform a simple test. If the object successfully passes multiple is-a or instance of tests, it's polymorphic. As I've described in my post about inheritance, all Java classes extend the class *Object*. Due to this, all objects in Java are polymorphic because they pass at least two instanceof checks.

### Why polymorphism is used:-

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.

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

**(ANSWER)**

**Program:-**

```
/* 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;
   }
}
```

```
/* 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
```

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:-

## Definition:-

Abstraction (from the Latin abs, meaning away from and here, meaning to draw) is the process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics.

## Abstraction is used in oop:-

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**.

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.

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.

## b. Write a program on abstraction in java.

**(ANSWER)**

**Program:-**

```
                    // 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();
}


}
```

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

**(ANSWER)**

## Access Modifiers:-

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. Java access modifiers are also sometimes referred to in daily speech as *Java access specifiers*, but the correct name is Java access modifiers. Classes, fields, constructors and methods can have one of four different Java access modifiers.

## Private access modifiers:-

If a method or variable is marked as `private` (has the `private` access modifier assigned to it), then only code inside the same class can access the variable, or call the method. Code inside subclasses cannot access the variable or method, nor can code from any external class.

Classes cannot be marked with the `private` access modifier. Marking a class with the `private` access modifier would mean that no other class could access it, which means that you could not really use the class at all. Therefore the `private` access modifier is not allowed for classes.

## Default access modifier:-

The default Java access modifier is declared by not writing any access modifier at all. The default access modifier means that code inside the class itself as well as code inside classes in the same package as this class, can access the class, field, constructor or method which the default access modifier is assigned to. Therefore, the `default` access modifier is also sometimes referred to as the `package` access modifier.

Subclasses cannot access methods and member variables (fields) in the superclass, if they these methods and fields are marked with the default access modifier, unless the subclass is located in the same package as the superclass.

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

**(ANSWER)**

## Program for Private access modifiers:-

```
 class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

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

```
    System.out.println(obj.data);//Compile Time Error
    obj.msg();//Compile Time Error
   }
}

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
 }
}
```

## Program for Default access modifier:-

```
//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
  }
  }
```

## Q2:- A) Explain in detail Public and protected access modifiers?

### Public access modifiers:-

The Java access modifier **`public`** means that all code can access the class, field, constructor or method, regardless of where the accessing code is located. The accessing code can be in a different class and different package.

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

### protected access modifiers:-

The **`protected`** access modifier provides the same access as the **`default`** access modifier, with the addition that subclasses can access **`protected`** methods and member variables (fields) of the superclass. This is true even if the subclass is not located in the same package as the superclass.

The protected access modifier is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

It provides more accessibility than the default modifer.

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

**(ANSWER)**

**Program for Public access modifiers:-**

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

**Output:-**

```
                              Hello
```

## Program for protected access modifiers:-

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

**Output:-**

```
                        Hello
```

**(THE END)**