NAME: MOHAMMAD IDREES

ID: 16285

SEMESTER: 2ND

DEPARTMENT: CS

ANS NO 1:

PART A:

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.

Access Modifiers are like entry gates for other classes i.e. a class can control what information or data can be accessible by other classes.

Java provides a number of access modifiers to help you set the level of access you want for classes as well as the fields, methods and constructors in your classes. A member has package or default accessibility when no accessibility modifier is specified.

1Private:

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

accessed from outside the class. 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. The private access modifier is specified using the keyword private. The methods or data members declared as private are accessible only within the class in which they are declared. Any other class of same package will not be able to access these members.

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 When no access modifier is specified for a class method or data member It is said to be having the default access modifier by default. The data members, class or methods which are not declared using any access modifiers i.e. having default access modifier are accessible only within the same package.

PART B:

Private access modifier:

class Data {

 private String name;

 // getter method

 public String get Name() {

 return this.name;

 }

 // setter method

 public void set Name(String name) {

 this.name= name;

 }

}

public class Main {

 public static void main(String[] main){

 Data d = new Data();

 // access the private variable using the getter and setter

 d. set Name("Program");

 System. out. PRINTLN (d .get Name());

 }

}

Output:

The name is Program.

2 Default access modifier:

package default Package;

class Logger {

 void message(){

 System. out. PRINTLN("This is a message");

 }

}

ANS NO 2:

PART A:

1 Protected:

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 modifier.

Variables, methods, and constructors, which are declared protected in a super class 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.

2 Public:

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

The members, methods and classes that are declared public can be accessed from anywhere. This modifier doesn’t put any restriction on the access.

PART B:

1 PROTECTED ACCESS MODIFIER:

class Animal {

 // protected method

 protected void display() {

 System. out. PRINTLN("I am an animal");

 }

}

class Dog extends Animal {

 public static void main(String[] ARGS ) {

 // create an object of Dog class

 Dog d o g = new Dog();

 // access protected method

 dog. display();

 }

}

Output:

I am an animal

2 PUBLIC ACCESS MODIFIER:

// Animal.java file

// public class

public class Animal {

 // public variable

 public I n t leg Count;

 // public method

 public void display() {

 System. out. PRINTLN("I am an animal.");

 System. out. PRINTLN("I have " + leg Count + " legs.");

 }

}

// Main.java

public class Main {

 public static void main( String[] ARGS ) {

 // accessing the public class

 Animal a n i m a l = new Animal();

 // accessing the public variable

 animal. Leg Count = 4;

 // accessing the public method

 animal. display();

 }

}

Output:

I am an animal.

I have 4 legs.

ANS NO 3:

PART A:

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.

it is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class The subclass can add its own fields and methods in addition to the superclass fields and methods.

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.

PART B:

class Animal{

void eat(){System. out. PRINTLN("eating...");}

}

class Dog extends Animal{

void bark(){System. out. PRNTLN("barking...");}

}

class Test Inheritance{

public static void main(String ARGS[]){

Dog d=new Dog();

d. bark();

d. eat();

}}

Output:

Barking…

eating...

ANS NO 4:

PART A:

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.

Polymorphism in Java is a concept by which we can perform a single action in different ways We can perform polymorphism in java by method overloading and method overriding.

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.

PART B:

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

public class Employee {

 private String name;

 private String address;

 private I n t 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 mail Check() {

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

 }

 public String to String() {

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

 }

 public String get Name() {

 return name;

 }

 public String get Address() {

 return address;

 }

 public void set Address(String new Address) {

 address = new Address;

 }

 public in t get Number() {

 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, in t number, double salary) {

 super(name, address, number);

 set Salary(salary);

 }

 public void mail Check() {

 System. out. PRINTLN("Within mail Check of Salary class ");

 System. out. PRINTLN("Mailing check to " + get Name()

 + " with salary " + salary);

 }

 public double get Salary() {

 return salary;

 }

 public void set Salary(double new Salary) {

 if(new Salary >= 0.0) {

 salary = new Salary;

 }

 }

 public double compute Pay() {

 System. out. PRINTLN("Computing salary pay for " + get Name());

 return salary/52;

 }

}

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

/\* File name : VirtualDemo.java \*/

public class Virtual Demo {

 public static void main(String [] ARGS) {

 Salary s = new Salary("HASHAM", "AMAR, UP", 3, 3600.00);

 Employee e = new Salary("KHUSHAL", "ABDULLAH, MA", 2, 2400.00);

 System. out. PRINTLN("Call mail Check using Salary reference --");

 s. mail Check();

 System. out. PRINTLN("\n Call mail Check using Employee reference--");

 e. mail Check();

 }

}

This will produce the following result

Output

Constructing an Employee

Constructing an Employee

Call mail Check using Salary reference –

Within mail Check of Salary class

Mailing check to HASHAM with salary 3600.0

Call mail Check using Employee reference--

Within mail Check of Salary class Mailing check to KHUSHAL with salary 2400.

ANS NO 5:

PART A:

Abstraction is one of the key concepts of object-oriented programming (OOP) languages. Its main goal is to handle complexity by hiding unnecessary details from the user That's a very generic concept that's not limited to object-oriented programming. You can find it everywhere in the real world image result for why abstraction is important in oops

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

PART B:

// Abstract class

abstract class Animal {

 // Abstract method (does not have a body)

 public abstract void animal Sound();

 // Regular method

 public void sleep() {

 System. out. PRINTLN("ZZZ");

 }

}

// Subclass (inherit from Animal)

class Pig extends Animal {

 public void animal Sound() {

 // The body of animal Sound() is provided here

 System. Out. PRINTLN("The pig says: wee wee");

 }

}

class My Main Class {

 public static void main(String[] ARGS) {

 Pig my Pig = new Pig(); // Create a Pig object

 My Pig .animal Sound();

 My Pig. sleep();

 }

}

OUTPUT

WEE W EE ZZZ