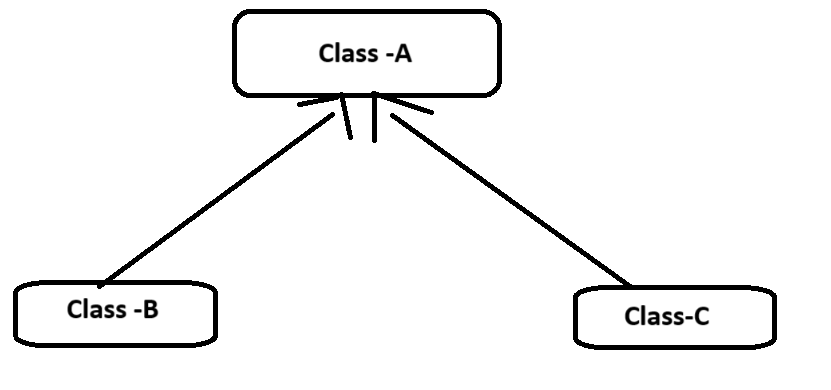
1. Explain Hierarchical inheritance with example.

Ans:=

Hierarchical inheritance in Java involves creating a class hierarchy where one superclass (parent class) is inherited by multiple subclasses (child classes), forming a hierarchical structure. Each subclass inherits the properties and behaviors of the superclass and can also have its own unique properties and behaviors.

Diagram of Hierarchical Inheritance:=



2. What is package? How to create and import package in java? Explain it with

suitable example.

**Or**

14.What is package? Explain the uses of the keywords “package” and “import”.

Ans:=

What is package:=

Package in Java is a collection of classes, sub-packages, and interfaces.It helps organize your classes into a folder structure and make it easy to locate and use them.

**Syntax:-**

package nameOfPackage;

## **How to Create a package?**

Creating a package is a simple task as follows

* Choose the name of the package
* Include the package command as the first line of code in your Java Source File.
* The Source file contains the classes, interfaces, etc you want to include in the package.
* Compile to create the Java packages

example:=following package program

package p1;

class c1()

{

public void m1()

{

System.out.println("m1 of c1");

}

public static void main(string args[])

{

c1 obj = new c1();

obj.m1();

}

}

## **How to Import Package**

To create an object of a class (bundled in a package), in your code, you have to use its fully qualified name.

**Example:**

java.awt.event.actionListner object = new java.awt.event.actionListner();

But, it could become tedious to type the long dot-separated package path name for every class you want to use. Instead, it is recommended you use the import statement.

**Syntax:=**

import packageName;

Once imported, you can use the class without mentioning its fully qualified name.

import java.awt.event.\*; // \* signifies all classes in this package are imported

import javax.swing.JFrame // here only the JFrame class is imported

//Usage

JFrame f = new JFrame; // without fully qualified name.

**Example**: To import package

package p3;

import p1.\*; //imports classes only in package p1 and NOT in the sub-package p2

class c3{

public void m3(){

System.out.println("Method m3 of Class c3");

}

public static void main(String args[]){

c1 obj1 = new c1();

obj1.m1();

}

}

3. Define interface .How does it overcome the problem of java’s multiple inheritance.?

**OR**

6. What is inheritance? Explain multilevel inheritance with appropriate example.

Ans:=

**What is inheritance?**

Inheritance in Java is a mechanism by which a new class (subclass) can acquire the properties and behaviors (methods and fields) of an existing class (superclass).

### **What is Interface in Java?**

A Java interface is a group of abstract methods that specify the behavior that implementing classes must follow.

**Systax:=**

**Interface.interface\_name{**

**//declare contains field**

**//declare method that abstract**

**//by default**

**}**

### **Why Java does not Support Multiple Inheritance?**

Multiple inheritance is supported by C++, Python, and a few other languages, but not by Java. To prevent the ambiguity produced by multiple inheritance, Java does not allow it. The diamond problem, which happens in multiple inheritance, is one example of such a problem.

There are two reasons given below that explain why Java does not support multiple inheritance.

Example:=

interface Dog

{

void bark();

}

interface Cat

{

void meow();

}

class Animal implements Dog, Cat

{

public void bark()

{

System.out.println("Dog is barking");

}

public void meow()

{

System.out.println("Cat is meowing");

}

}

class Main

{

public static void main(String args[])

{

Animal a = new Animal();

a.bark();

a.meow();

}

}

**Output of the above code:**

Dog is barking

Cat is meowing

4. Differentiate: Abstract class V/s Interface.

Ans:=

Abstract Class:

1.->An abstract class can have both abstract (methods without implementation) and concrete (methods with implementation) methods.

2.->An abstract class can have instance variables.

3.->An abstract class cannot be instantiated directly; it needs to be subclassed.

4.->A subclass of an abstract class must either provide implementations for all abstract methods or be declared abstract itself.

5.->An abstract class can have constructors.

6.->It supports the concept of inheritance.

Example:

abstract class Shape {

abstract void draw();

void commonMethod() {

// implementation

}

}

Interface:

1.->An interface can only have abstract methods (methods without implementation) and constant variables.

2.->An interface cannot have instance variables (fields).

3.->An interface cannot contain constructors.

4->A class can implement multiple interfaces but can extend only one class.

5->All methods in an interface are public by default.

6->Interfaces support the concept of multiple inheritance.

Example:

interface Drawable {

void draw();

// More methods...

}

8. What is package? Explain package with simple example.

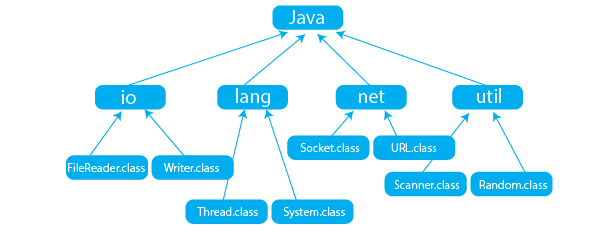
Ans:=

**PACKAGE in Java** is a collection of classes, sub-packages, and interfaces.It helps organize your classes into a folder structure and make it easy to locate and use them.

### **What are the advantages of a java package?**

* Java packages are used to proved access protection
* Java packages are used to eliminate naming collision
* Java packages are used to classify the classes and the interfaces into categories

**Diagram of Java Package:=**



## **Simple package program in java:**

import java.util.Scanner;

class PrepBytes {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

int num=sc.nextInt();

System.out.println("The Number is "+num);

}

}

Output: The Number is 10

10.Write the difference between class, abstract class and interface with java

Program.

Ans:=

In Java, class, abstract class, and interface are three fundamental concepts used in object-oriented programming. Here's a brief explanation of each along with examples:

Class:

->A class is a blueprint for creating objects. It encapsulates data for the object and provides methods to operate on that data.

->It can contain fields, constructors, and methods.

->Objects are instances of classes.

Example:=

public class Car {

private String make;

private String model;

public Car(String make, String model) {

this.make = make;

this.model = model;

}

public void start() {

System.out.println("The " + make + " " + model + " is starting.");

}

}

Abstract Class:

->An abstract class is a class that cannot be instantiated on its own and is meant to be subclassed.

->It can contain abstract methods, which are declared but not implemented in the abstract class.Subclasses must provide implementations for these methods.

->Abstract classes can also have concrete methods.

Example:=

public abstract class Animal {

private String name;

public Animal(String name) {

this.name = name;

}

public abstract void sound();

}

Interface:

->An interface is a reference type in Java that is similar to a class. It is a collection of abstract methods.

->It can contain only method declarations and constant variables.

->Classes can implement one or more interfaces, and it enforces the implementing class to provide implementations for all the methods declared in the interface.

Example:=

public interface Shape {

double area();

double perimeter();

}

11.Demonstrate use of Top level nested classes and Inner classes with suitable example.

Ans:=

Sure, here's a demonstration of top-level nested classes and inner classes in Java with suitable examples:

Top-Level Nested Class Example:

public class Outer {

private int outerField;

// Top-level nested class

static class Nested {

private int nestedField;

public void nestedMethod() {

System.out.println("Inside nested method");

}

}

public static void main(String[] args) {

Outer.Nested nestedObject = new Outer.Nested();

nestedObject.nestedMethod();

}

}

Explanation:

->Nested class is nested inside the Outer class.

->It's a static class, so it can be accessed using Outer.Nested.

->In the main method, we create an instance of the Nested class and call its method.

Inner Class Example:

public class Outer {

private int outerField;

// Inner class

class Inner {

private int innerField;

public void innerMethod() {

System.out.println("Inside inner method");

}

}

public static void main(String[] args) {

Outer outerObject = new Outer();

Outer.Inner innerObject = outerObject.new Inner();

innerObject.innerMethod();

}

}

Explanation:

->Inner class is an inner class inside the Outer class.

->Inner classes have access to members of the enclosing class.

->In the main method, we create an instance of the Outer class and then use it to create an instance of the Inner class. Then, we call the method of the Inner class.

Both top-level nested classes and inner classes provide a way to logically group classes that are only used in one place, thus increasing encapsulation and creating more readable and maintainable code.

12.What is class? Explain any one wrapper class with example.

Ans:=

Class:

In Java, a class is a blueprint for creating objects. It defines the structure and behavior of objects by specifying fields to store data and methods to operate on that data.

The wrapper classes in Java are used to convert primitive types (int, char, float, etc) into corresponding objects.

Each of the 8 primitive types has corresponding wrapper classes.

| Primitive Type | Wrapper Class |
| --- | --- |
| byte | Byte |
| boolean | Boolean |
| char | Character |
| double | Double |
| float | Float |
| int | Integer |
| long | Long |
| short | Short |

## **Convert Primitive Type to Wrapper Objects**

We can also use the valueOf() method to convert primitive types into corresponding objects.

Example:=

class Main {

public static void main(String[] args) {

// create primitive types

int a = 5;

double b = 5.65;

//converts into wrapper objects

Integer aObj = Integer.valueOf(a);

Double bObj = Double.valueOf(b);

if(aObj instanceof Integer) {

System.out.println("An object of Integer is created.");

}

if(bObj instanceof Double) {

System.out.println("An object of Double is created.");

}

}

}

Output:=

**An object of Integer is created.**

**An object of Double is created.**

13.What is an interface? How it is important in java? Explain with example.

Ans:=

Interface:

In Java, an interface is a reference type that defines a set of abstract methods that a class must implement.

Importance of Interface in Java:

1.Achieving Abstraction: Interfaces allow programmers to define a set of methods that must be implemented by implementing classes, without providing the implementation details. This promotes abstraction, making the code more modular and easier to maintain.

2.Enforcing Contracts: Interfaces define contracts that implementing classes must adhere to. This ensures consistency and interoperability among different implementations, facilitating code integration and collaboration.

3.Facilitating Polymorphism: Interfaces support polymorphism, allowing objects of different classes that implement the same interface to be treated interchangeably. This promotes flexibility and code reusability.

4.Decoupling Components: By programming to interfaces rather than concrete implementations, components become loosely coupled, making the code more flexible and easier to refactor or extend.

Example:

// Interface defining the behavior of a shape

interface Shape {

double calculateArea(); // Abstract method to calculate the area

double calculatePerimeter(); // Abstract method to calculate the perimeter

}

// Implementing class for a Rectangle

class Rectangle implements Shape {

private double length;

private double width;

public Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

@Override

public double calculateArea() {

return length \* width;

}

@Override

public double calculatePerimeter() {

return 2 \* (length + width);

}

}

// Implementing class for a Circle

class Circle implements Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

@Override

public double calculateArea() {

return Math.PI \* radius \* radius;

}

@Override

public double calculatePerimeter() {

return 2 \* Math.PI \* radius;

}

}

// Main class to demonstrate the usage of interfaces

public class InterfaceExample {

public static void main(String[] args) {

Shape rectangle = new Rectangle(5, 3);

Shape circle = new Circle(4);

System.out.println("Area of rectangle: " + rectangle.calculateArea());

System.out.println("Perimeter of rectangle:" + rectangle.calculatePerimeter());

System.out.println("Area of circle: " + circle.calculateArea());

System.out.println("Circumference of circle: " + circle.calculatePerimeter());

}

}

//nathi lakhvana(5,6,7,15)

5. Explain java’s Access Modifiers with appropriate example.

6. What is inheritance? Explain multilevel inheritance with appropriate example.

7. How Overridden Methods Support Polymorphism? Explain with appropriate example.

15.What is Polymorphism? Explain its type and give appropriate example.