##### **Which keyword is used to refer current object of a class in Java?**

Ans: this

1. **Which of these can be used to fully abstract a class from its implementation?**

Ans: Interfaces

1. **Which of the following is the correct way of importing an entire package ‘pkg’?**

Ans: import pkg.\*

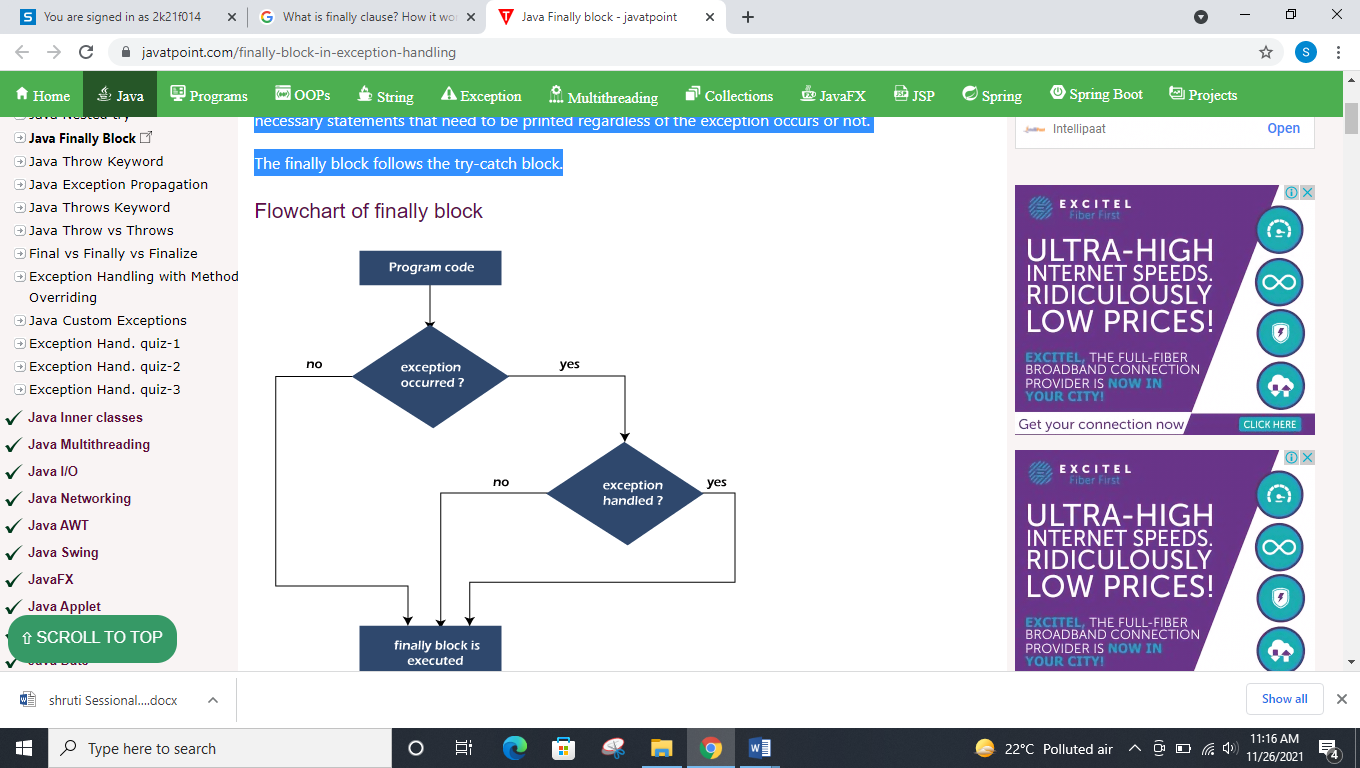
1. **Which of these keywords is not a part of exception handling?**

Ans: thrown

1. **What is finally clause? How it works?**

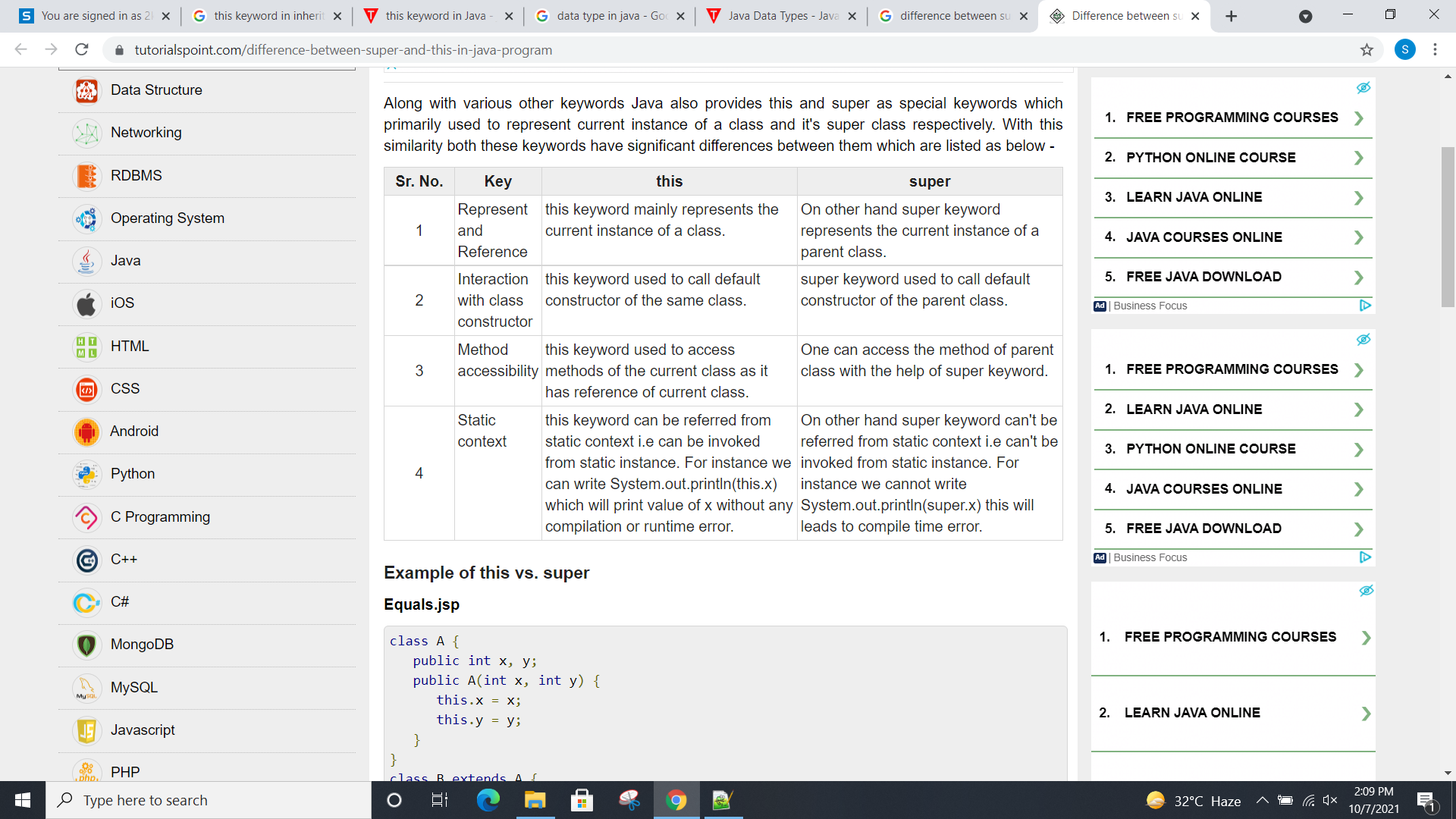
Ans: Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.

The finally block follows the try-catch block.



1. **Difference between This and Super with example.**

Ans:



**Example:**

class Person{

int id;

String name;

Person(int id,String name){

this.id=id;

this.name=name;

} }

class Emp extends Person{

float salary;

Emp(int id,String name,float salary){

super(id,name);//reusing parent constructor

this.salary=salary; }

void display(){System.out.println(id+" "+name+" "+salary);} }

class TestSuper5{

public static void main(String[] args){

Emp e1=new Emp(1,"ankit",45000f);

e1.display(); } }

1. **How JAVA support multiple inheritance? Differentiate between abstract class and interface**

**Ans:** The Java programming language supports multiple inheritance of type, which is the ability of a class to implement more than one interface. As with multiple inheritance of implementation, a class can inherit different implementations of a method defined (as default or static) in the interfaces that it extends.

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 7) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 8)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

1. **What is array? How do you declare Single dimensional array in JAVA? Explain with example**

**Ans:**

An array is a collection of similar data elements stored at contiguous memory locations. It is the simplest data structure where each data element can be accessed directly by only using its index number.

**Declare Single dimensional array**

import java.io.\*;

class Test {

public static void main(String[] args)

{ int[] a; // valid declaration

int b[]; // valid declaration

int[] c; // valid declaration

} }

**Java program to demonstrate working of one dimensional arrays**

class oneDimensionalArray {

  public static void main(String args[])

    {

        int[] a; // one dimensional array declaration

        a = new int[3]; // creating array of size 3

        for (int i = 0; i < 3; i++) {

            a[i] = 100;

            System.out.println(a[i]);

        }

    }

}

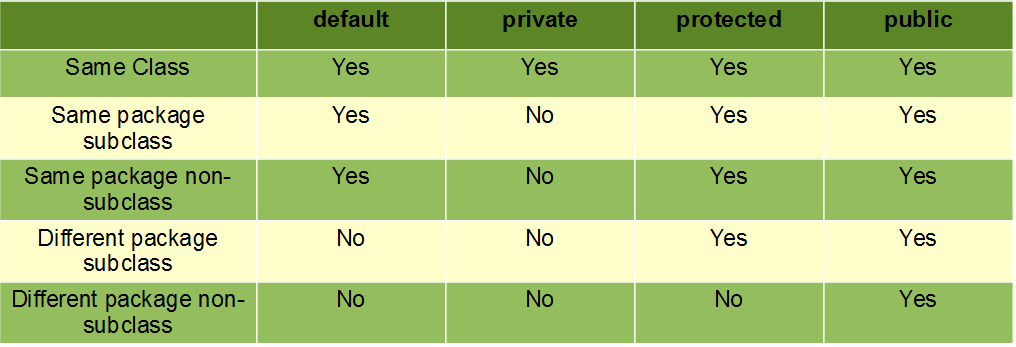
1. **A class inside package contains following class member declarations. Describe visibility of these members within or outside the package.**

**int a;**

**public int b;**

**protected int c;**

**private int d;**



**int a**:-

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.

**public int b;-**

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

**protected int c;**

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.

**private int d;**

The private access modifier is accessible only within the class.

1. **What is method overriding? Write a JAVA program to implement method overloading.**

**Expected Output:**

Enter three numbers:2 3 4

Multiplication of two numbers: 6

Multiplication of three numbers: 24

**Ans:**

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**. In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding. Method overriding is used to provide the specific implementation of a method which is already provided by its superclass. Method overriding is used for runtime polymorphism

import java.util.Scanner;

class Multiplication

{

public void mul(int a, int b)

{

System.out.println("Multiplication of two numbers: "+(a+b));

}

public void mul(int a, int b, int c)

{

System.out.println("Multiplication of three numbers: "+(a+b+c));

}

}

class Overloading

{

public static void main(String args[])

{

Scanner sc=new Scanner(System.in);

System.out.print("Enter three numbers: ");

int x=sc.nextInt();

int y=sc.nextInt();

int z=sc.nextInt();

Multiplication m=new Multiplication();

m.mul(x,y);

m.mul(x,y,z);

}

}

1. **Create a try block that is likely to generate three types of exception and then incorporate necessary catch blocks to catch and handle them appropriately.**

public class NestedTryBlock{

 public static void main(String args[]){

  try{

    try{

     System.out.println("going to divide by 0");

     int b =39/0;

   }

    catch(ArithmeticException e)

    {

      System.out.println(e);

    }

 try{

    int a[]=new int[5];

     a[5]=4;

     }

catch(ArrayIndexOutOfBoundsException e)

    {

       System.out.println(e);

    }

    System.out.println("other statement");

  }

    catch(Exception e)

  {

    System.out.println("handled the exception (outer catch)");

  }

     try {

            String a = null;

            System.out.println(a.charAt(0));

}

catch(NullPointerException e) {

            System.out.println("NullPointerException..");

         }

System.out.println("normal flow..");

 }

}

1. **Write a Java program to handle an ArithmeticException divide by zero using try, catch and finally blocks.**

import java.io.\*;

class Test {

public static void main(String[] args)

{

int a = 5;

int b = 0;

try {

System.out.println(a / b); // throw Exception

}

catch (ArithmeticException e) {

// Exception handler

System.out.println("Divided by zero operation cannot possible");

}

finally

{

System.out.println("finally block is always executed");

}

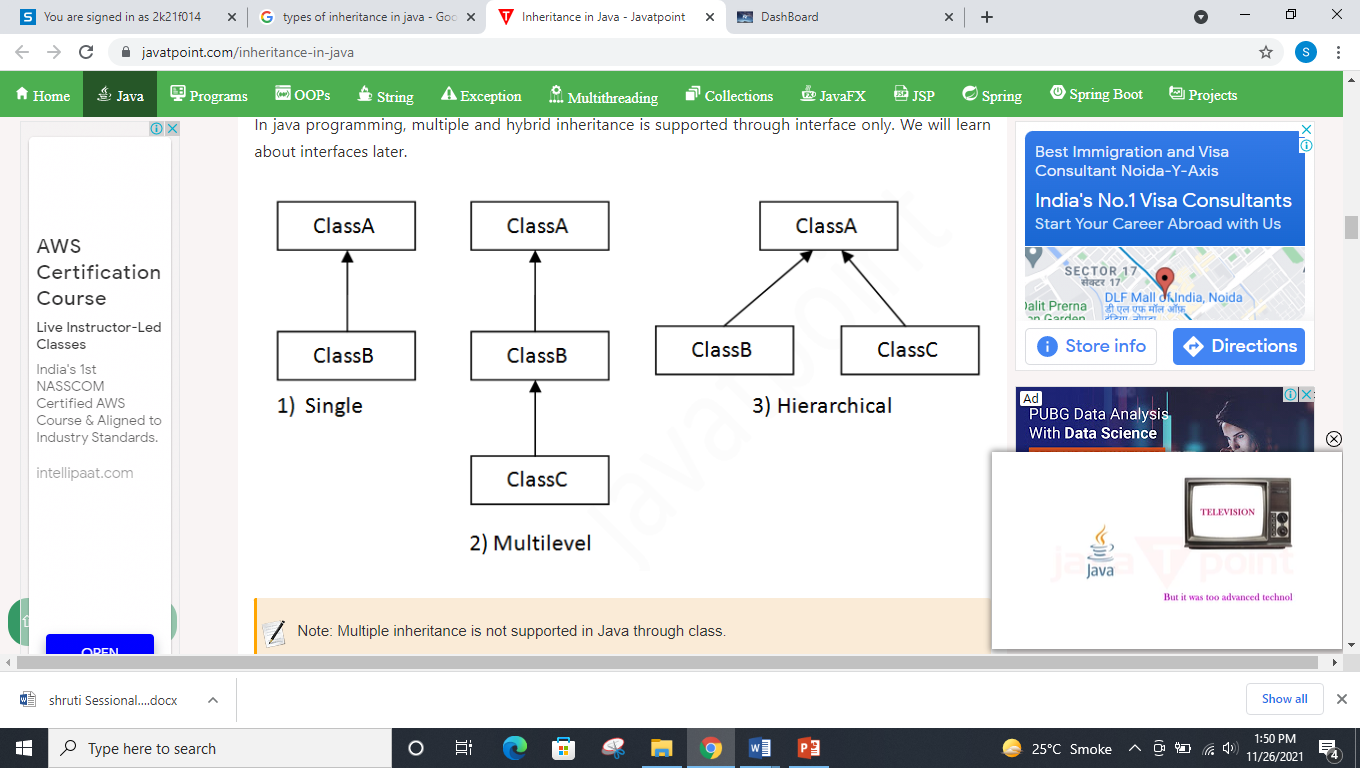
}

}

1. **What is inheritance? What are its types explaining with example**

Inheritance in Java is **a** mechanism in which one object acquires all the properties and behaviors of a parent object. The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class.

## **Types of inheritance in java**



## **1. Single Inheritance**

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

class Animal{

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

}

class Dog extends Animal{

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

}

class TestInheritance{

public static void main(String args[]){

Dog d=new Dog();

d.bark();

d.eat();

}}

## **2. Multilevel Inheritance**

When there is a chain of inheritance, it is known as *multilevel inheritance*. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

class Animal{

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

}

class Dog extends Animal{

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

}

class BabyDog extends Dog{

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

}

class TestInheritance2{

public static void main(String args[]){

BabyDog d=new BabyDog();

d.weep();

d.bark();

d.eat();

}}

## **3. Hierarchical Inheritance**

When two or more classes inherits a single class, it is known as *hierarchical inheritance*. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

class Animal{

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

}

class Dog extends Animal{

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

}

class Cat extends Animal{

void meow(){System.out.println("meowing...");}

}

class TestInheritance3{

public static void main(String args[]){

Cat c=new Cat();

c.meow();

c.eat();

//c.bark();//C.T.Error

}}