JAVA PROGRAMMING

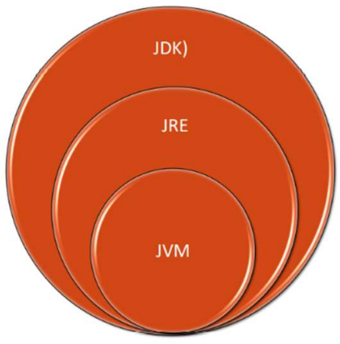
IMPORTANT QUESTIONS AND ANSWERS

KRUTHIK B

1DA19CS077

B SECTION

1)



JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

The JVM performs the following main tasks:

1)Loads code

2)Verifies code

3)Executes code

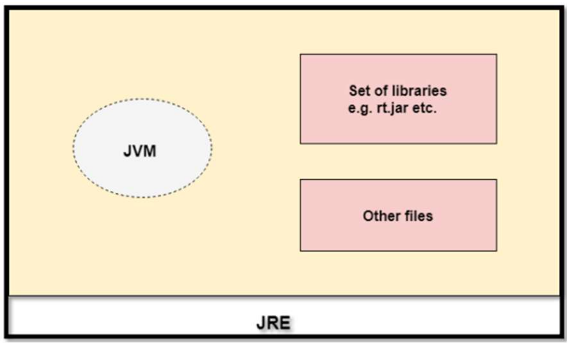
4)Provides runtime environment(JRE)

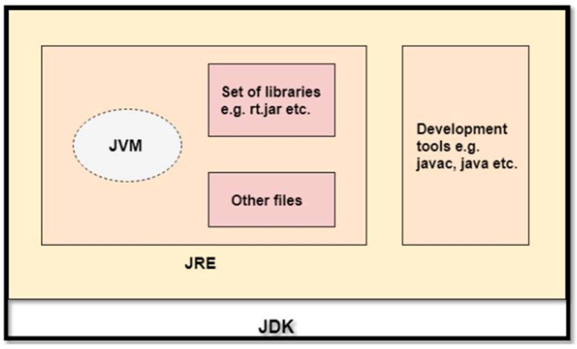
JRE

JRE(Java Runtime Environment). It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

JDK

JDK(Java Development Kit). The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools. The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.




JVM
Set of libraries e.g. rt.jar etc.
Other files
JRE
Development tools e.g. javac, java etc.
JDK

(Draw the first diagram after JRE and second one after JDK)

KRUTHIK B(1DA19CS077 B SECTION)

2)

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

class Animal{

String color="white";

```
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}
}
```

KRUTHIK B(1DA19CS077 B SECTION)

3)

Dynamic method dispatch is a mechanism by which a call to an overridden method is resolved at runtime. This is how java implements runtime polymorphism. When an overridden method is called by a reference, java determines which version of that method to execute based on the type of object it refer to.

Runtime polymorphism is achieved in java by method overriding in which a child class overrides the method described in parent class . Since method invocation is determined by the JVM and not by the complier it is known as run time polymorphism

```
class Student {
    public void show(){
        System.out.println("Student details.");
```

```
      }
}
public class CollegeStudent extends Student {
      public void show(){
            System.out.println("College Student details.");
      }
      public static void main(String args[]){
          Student obj = new CollegeStudent();
          obj.show();
     }
}
```

OUTPUT

```
College student details
```

KRUTHIK B(1DA19CS077 B SECTION)

4)and 15)

Method Overloading:

Method Overloading is a Compile time polymorphism. In method overloading, more than one method shares the same method name with different signature in the class. In method overloading, return type can or can not be be same, but we must have to change the parameter because in java, we can not achieve the method overloading by changing only the return type of the method.

Example of method overloading:

```
class MethodOverloadingEx{
   static int add(int a, int b){return a+b;}
   static int add(int a, int b, int c){return a+b+c;}
    public static void main(String args[]) {
      System.out.println(add(4, 6));
      System.out.println(add(4, 6, 7));
    }
}
```

Output:

```
//Here, add with two parameter method runs
10
// While here, add with three parameter method runs
```

KRUTHIK B(1DA19CS077 B SECTION)

17)

Method Overriding:

Method Overriding is a Run time polymorphism. In method overriding, derived class provides the specific implementation of the method that is already provided by the base class or parent class. In method overriding, return type must be same or co-variant (return type may vary in same direction as the derived class).

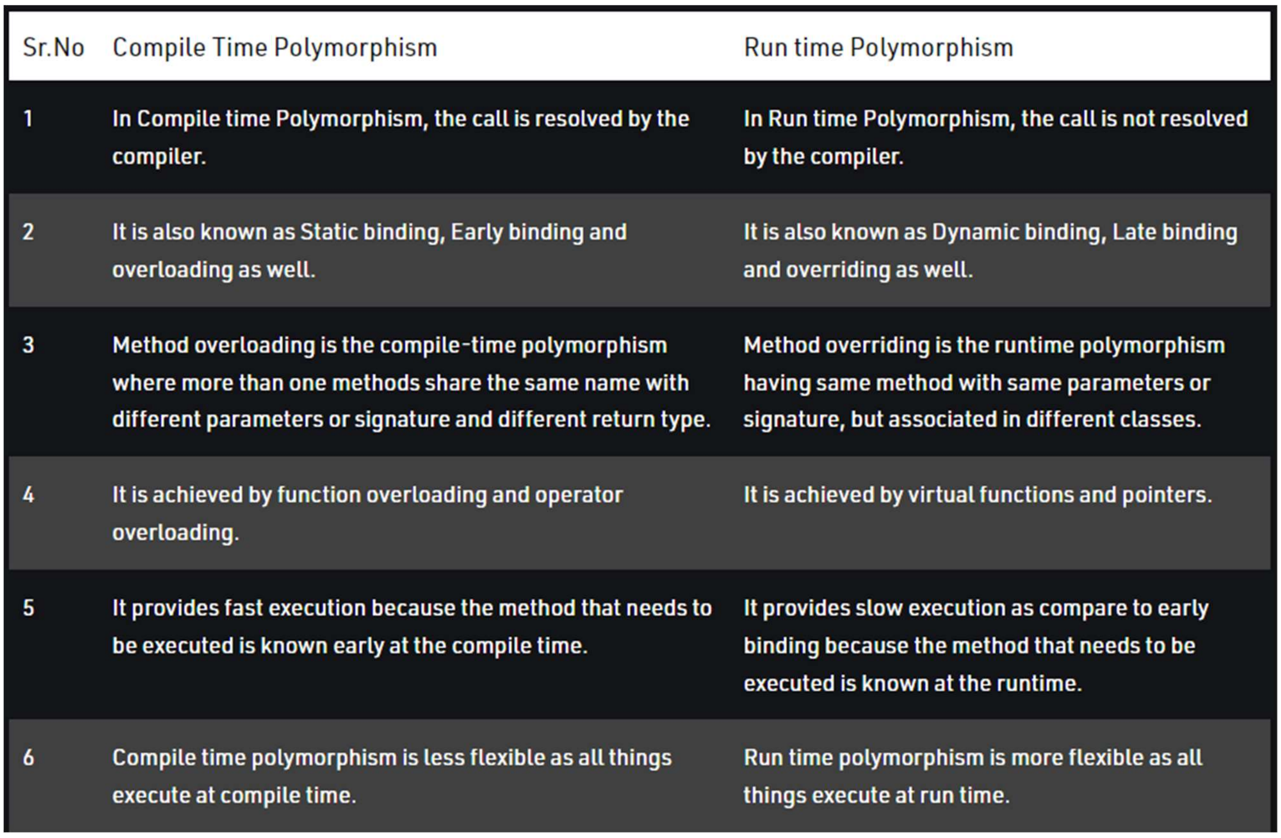
Example of method overriding:

```
class Animal{
    void eat(){System.out.println("eating.");}
    }
  class Dog extends Animal{
  void eat(){System.out.println("Dog is eating.");}
   }
 class MethodOverridingEx{
  public static void main(String args[]) {
    Dog d1=new Dog();
    Animal a1=new Animal();
    d1.eat();
```

```
      a1.eat();
   }
}
```

Output:

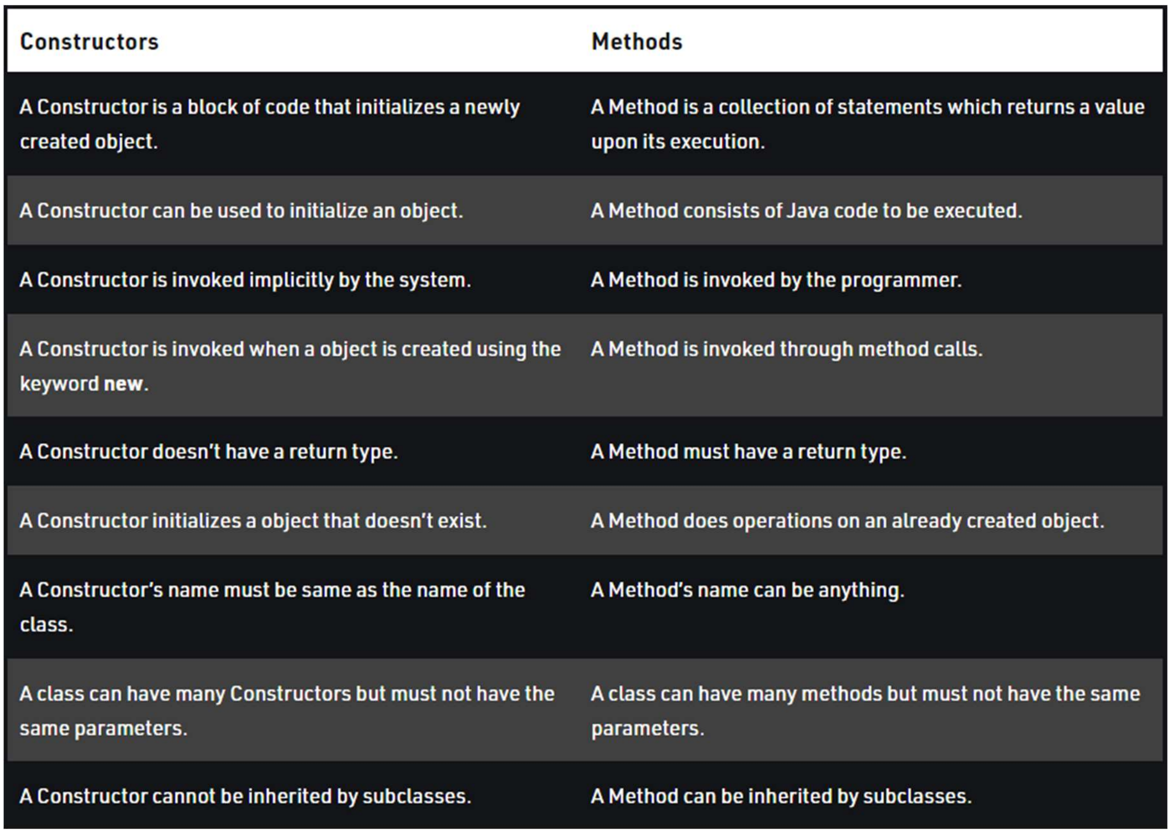
```
// Derived class method eat() runs
Dog is eating
// Base class method eat() runs
Eating
```



Sr.No	Compile Time Polymorphism	Run time Polymorphism
1	In Compile time Polymorphism, the call is resolved by the compiler.	In Run time Polymorphism, the call is not resolved by the compiler.
2	It is also known as Static binding, Early binding and overloading as well.	It is also known as Dynamic binding, Late binding and overriding as well.
3	Method overloading is the compile-time polymorphism where more than one methods share the same name with different parameters or signature and different return type.	Method overriding is the runtime polymorphism having same method with same parameters or signature, but associated in different classes.
4	It is achieved by function overloading and operator overloading.	It is achieved by virtual functions and pointers.
5	It provides fast execution because the method that needs to be executed is known early at the compile time.	It provides slow execution as compare to early binding because the method that needs to be executed is known at the runtime.
6	Compile time polymorphism is less flexible as all things execute at compile time.	Run time polymorphism is more flexible as all things execute at run time.

KRUTHIK B(1DA19CS077 B SECTION)

6)



Constructors	Methods
A Constructor is a block of code that initializes a newly created object.	A Method is a collection of statements which returns a value upon its execution.
A Constructor can be used to initialize an object.	A Method consists of Java code to be executed.
A Constructor is invoked implicitly by the system.	A Method is invoked by the programmer.
A Constructor is invoked when a object is created using the keyword **new**.	A Method is invoked through method calls.
A Constructor doesn't have a return type.	A Method must have a return type.
A Constructor initializes a object that doesn't exist.	A Method does operations on an already created object.
A Constructor's name must be same as the name of the class.	A Method's name can be anything.
A class can have many Constructors but must not have the same parameters.	A class can have many methods but must not have the same parameters.
A Constructor cannot be inherited by subclasses.	A Method can be inherited by subclasses.

Example

```
public class JavaTester {
  int num;
  JavaTester(){
     num = 3;
     System.out.println("Constructor invoked. num: " + num);
  }
  public void init(){
     num = 5;
     System.out.println("Method invoked. num: " + num);
  }
  public static void main(String args[]) {
     JavaTester tester = new JavaTester();
     tester.init();
  }
}
```

Output

```
Constructor invoked. num: 3
Method invoked. num: 5
```

KRUTHIK B(1DA19CS077 B SECTION)

9)

THE ANSWER IS AGGREGATION

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Consider a situation, Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

```
class Employee{
int id;
String name;
Address address;//Address is a class
...
}
```

Aggregation Uses

1)Code reuse is also best achieved by aggregation when there is no is-a relationship.

2)Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

In this example, Employee has an object of Address, address object contains its own informations such as city, state, country etc. In such case relationship is Employee HAS-A address.

Address.java

```
public class Address {
String city,state,country;
public Address(String city, String state, String country) {
   this.city = city;
   this.state = state;
   this.country = country;
}
}
```

Emp.java

```
public class Emp {
int id;
String name;
Address address;
public Emp(int id, String name,Address address) {
   this.id = id;
   this.name = name;
   this.address=address;
}
void display(){
System.out.println(id+" "+name);
System.out.println(address.city+" "+address.state+" "+address.country);
}
public static void main(String[] args) {
Address address1=new Address("gzb","UP","india");
Address address2=new Address("gno","UP","india");


Emp e=new Emp(111,"varun",address1);
Emp e2=new Emp(112,"arun",address2);
e.display();
e2.display();
}
}
```

Output:

```
111 varun
gzb UP india
112 arun
gno UP india
```

KRUTHIK B(1DA19CS077 B SECTION)

10)

Exception Handling in Java

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

Exception in Java

Dictionary Meaning: Exception is an abnormal condition.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Exception Handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Different type of exceptions

1**) Checked Exception**

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

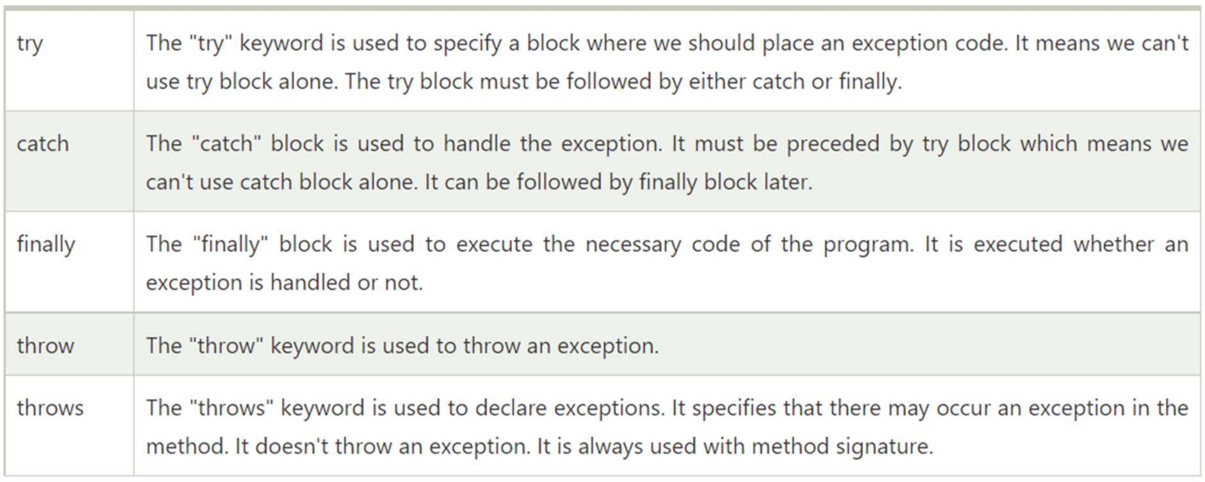
2) **Unchecked Exception**

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) **Error**

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

KEYWORDS USED IN EXCEPTION HANDLING MECHANISMS



try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

KRUTHIK B(1DA19CS077 B SECTION)

13)

Simple

Object-Oriented

Portable

Platform independent

Secured

Robust

Architecture neutral

Interpreted

High Performance

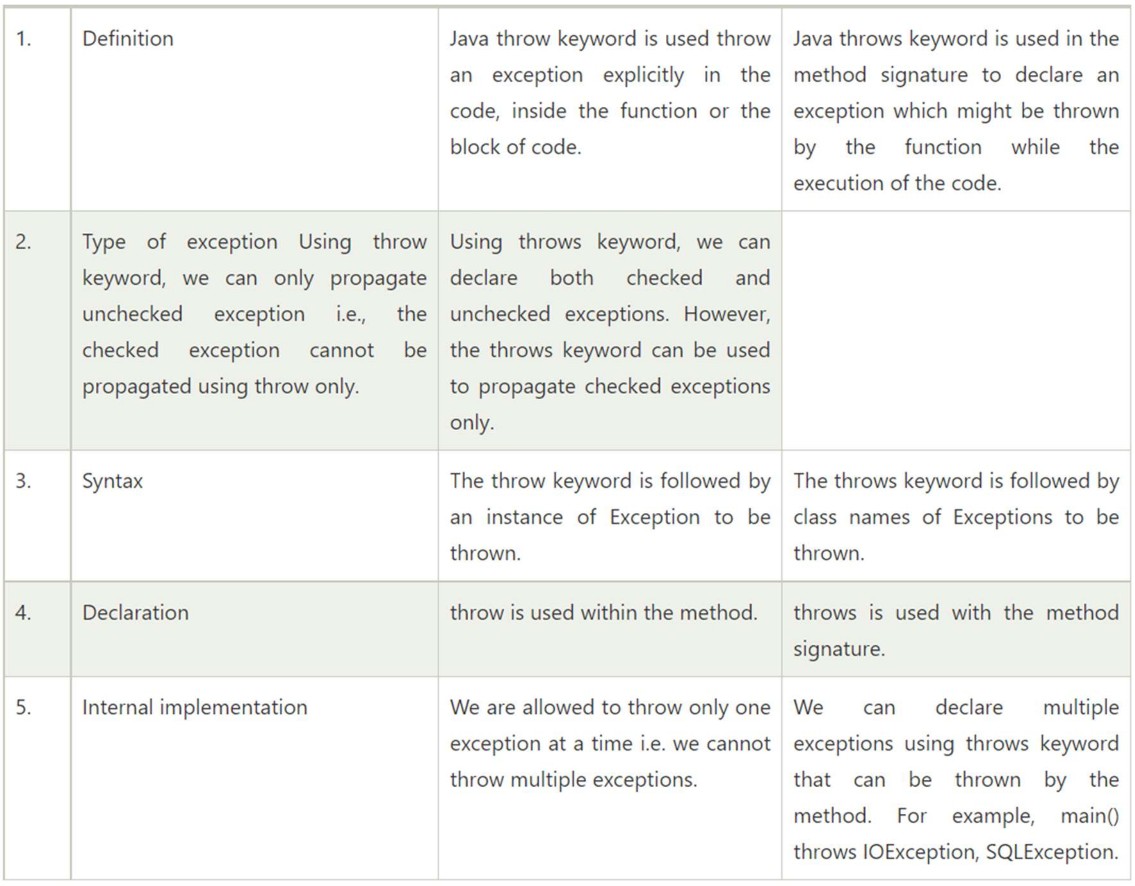
Multithreaded

Distributed

Dynamic

KRUTHIK B(1DA19CS077 B SECTION)

16)



1.	Definition	Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.	Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.
2.	Type of exception Using throw keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only.	Using throws keyword, we can declare both checked and unchecked exceptions. However, the throws keyword can be used to propagate checked exceptions only.	
3.	Syntax	The throw keyword is followed by an instance of Exception to be thrown.	The throws keyword is followed by class names of Exceptions to be thrown.
4.	Declaration	throw is used within the method.	throws is used with the method signature.
5.	Internal implementation	We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions.	We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException.

Java throw Example

TestThrow.java

```
public class TestThrow {
    //defining a method
    public static void checkNum(int num) {
        if (num < 1) {
            throw new ArithmeticException("\nNumber is negative, cannot
calculate square");
        }
        else {
            System.out.println("Square of " + num + " is " + (num*num));
        }
    }
    //main method
    public static void main(String[] args) {
            TestThrow obj = new TestThrow();
            obj.checkNum(-3);
            System.out.println("Rest of the code..");
    }
}
```

OUTPUT

```
Number is negative, cannot calculate square
```

Java throws Example

TestThrows.java

```
public class TestThrows {
    //defining a method
    public static int divideNum(int m, int n) throws ArithmeticException {
        int div = m / n;
        return div;
    }
    //main method
    public static void main(String[] args) {
        TestThrows obj = new TestThrows();
        try {
           System.out.println(obj.divideNum(45, 0));
        }
        catch (ArithmeticException e){
           System.out.println("\nNumber cannot be divided by 0");
        }
        System.out.println("Rest of the code..");
    }
}
```

OUTPUT

```
Number cannot be divided by 0
```

KRUTHIK B(1DA19CS077 B SECTION)

14)

<u>NESTED CLASSES</u>

Writing a class within another is allowed in Java. The class written within is called the nested class, and the class that holds the inner class is called the outer class.

Nested classes are divided into two types

1)Non-static nested classes (Inner Classes) − These are the non-static members of a class.

2)Static nested classes − These are the static members of a class.

```
// outer class
class OuterClass
{
    // static member
    static int outer_x = 10
    // instance(non-static) member
    int outer_y = 20;
    // private member
    private int outer_private = 30;
    // inner class
    class InnerClass
    {
        void display()
        {
            // can access static member of outer class
            System.out.println("outer_x = " + outer_x);

            // can also access non-static member of outer class
            System.out.println("outer_y = " + outer_y);

            // can also access a private member of the outer class
            System.out.println("outer_private = " + outer_private);
        }
    }
}
public class InnerClassDemo
{
    public static void main(String[] args)
    {
        // accessing an inner class
        OuterClass outerObject = new OuterClass();
        OuterClass.InnerClass innerObject = outerObject.new InnerClass();
        innerObject.display();
    }
}
```

(Explain that the methods and variables in the inner classes cannot be accessed by the outer class but can happen vice versa)

KRUTHIK B(1DA19CS077 B SECTION)

12)

<u>TRY-CATCH</u>

Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

If an exception occurs at the particular statement in the try block, the rest of the block code will not execute. So, it is recommended not to keep the code in try block that will not throw an exception.

Java try block must be followed by either catch or finally block.

Java catch block

Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception (i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.

Syntax of Java try-catch

```
try{
//code that may throw an exception
}catch(Exception_class_Name ref){}
```

EXAMPLE

```
public class TryCatchExample{
  public static void main(String[] args) {
    try
    {
    int data=50/0; //may throw exception
    }
      //handling the exception
    catch(ArithmeticException e)
    {
      System.out.println(e);
    }
    System.out.println("rest of the code");
  }
}
```

NESTED TRY-CATCH

Java Nested try block

In Java, using a try block inside another try block is permitted. It is called as nested try block. Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.

For example, the inner try block can be used to handle ArrayIndexOutOfBoundsException while the outer try block can handle the ArithemeticException (division by zero).

```
class NestingDemo{
  public static void main(String args[]){
    //main try-block
    try {
        //try-block2
      try {
         //try-block3
         try {
                  int arr[]= {1,2,3,4};
                  System.out.println(arr[10]);
           }
         catch(ArithmeticException e){
                  System.out.print("Arithmetic Exception");
                  System.out.println(" handled in try-block3");
           }
         }
      catch(ArithmeticException e){
         System.out.print("Arithmetic Exception");
         System.out.println(" handled in try-block2");
         }
      }
    catch(ArithmeticException e3){
        System.out.print("Arithmetic Exception");
        System.out.println(" handled in main try-block");
  }
  catch(ArrayIndexOutOfBoundsException e4){
        System.out.print("ArrayIndexOutOfBoundsException");
        System.out.println(" handled in main try-block");
  }
  catch(Exception e5){
          System.out.print("Exception");
          System.out.println(" handled in main try-block");
     }
   }
}
```

Output:

```
ArrayIndexOutOfBoundsException handled in main try-block
```

TRY-CATCH and FINALLY

Java finally block

Java finally block is a block used to execute important code such as closing the connection, etc.

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.

```
class TestFinallyBlock {
          public static void main(String args[]){
       try{
              //below code do not throw any exception
              int data=25/5;
              System.out.println(data);
           }
       //catch won't be executed
       catch(NullPointerException e){
              System.out.println(e);
           }
```

```
        //executed regardless of exception occurred or not
        finally {
                System.out.println("finally block is always executed");
                }
        System.out.println("rest of phe code...");
    }
}
```

OUTPUT

```
5
finally block is always executed
rest of phe code...
```

KRUTHIK B(1DA19CS077 B SECTION)

11)

SUPER / THIS / FINAL

(this) keyword in Java

There can be a lot of usage of Java this keyword. In Java, this is a reference variable that refers to the current object.

Usage of Java this keyword

Here is given the **6 usage of java this keyword**.

1)this can be used to refer current class instance variable.

2)this can be used to invoke current class method (implicitly)

3)this() can be used to invoke current class constructor.

4)this can be passed as an argument in the method call.

5)this can be passed as argument in the constructor call.

6)this can be used to return the current class instance from the method.

<u>EXAMPLE</u>

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}
}
```

FINAL KEYWORD

The ***final keyword*** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1)Variable

2)method

3)class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only.

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike();
 obj.run();
 }
}//end of class
```

SUPER COVERED IN 2 QUESTION

KRUTHIK B(1DA19CS077 B SECTION)

8)

Interface in Java

An interface in Java is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also represents the IS-A relationship.

There are mainly three reasons to use interface.

1)It is used to achieve abstraction.

2)By interface, we can support the functionality of multiple inheritance.

3)It can be used to achieve loose coupling.

The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a class implements an interface.

EXAMPLE

```
interface Bank{
float rateOfInterest();
}
class SBI implements Bank{
public float rateOfInterest(){return 9.15f;}
}
class PNB implements Bank{
public float rateOfInterest(){return 9.7f;}
}
class TestInterface2{
public static void main(String[] args){
Bank b=new SBI();
System.out.println("ROI: "+b.rateOfInterest());
}
}
```

(Explain the prgms in ur own words)

KRUTHIK B(1DA19CS077 B SECTION)

7)

Java Package

A java package is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Creating a Package

1)Choose a package name according to the naming convention.

2)Write the package name at the top of every source file (classes, interface, enumeration, and annotations).

3)Remember that there must be only one package statement in each source file.

Importing a Package

If we want to use a package in Java program it is necessary to import that package at the top of the program by using the import keyword before the package name.

Syntax:

```
import packageName;
```

EXAMPLE

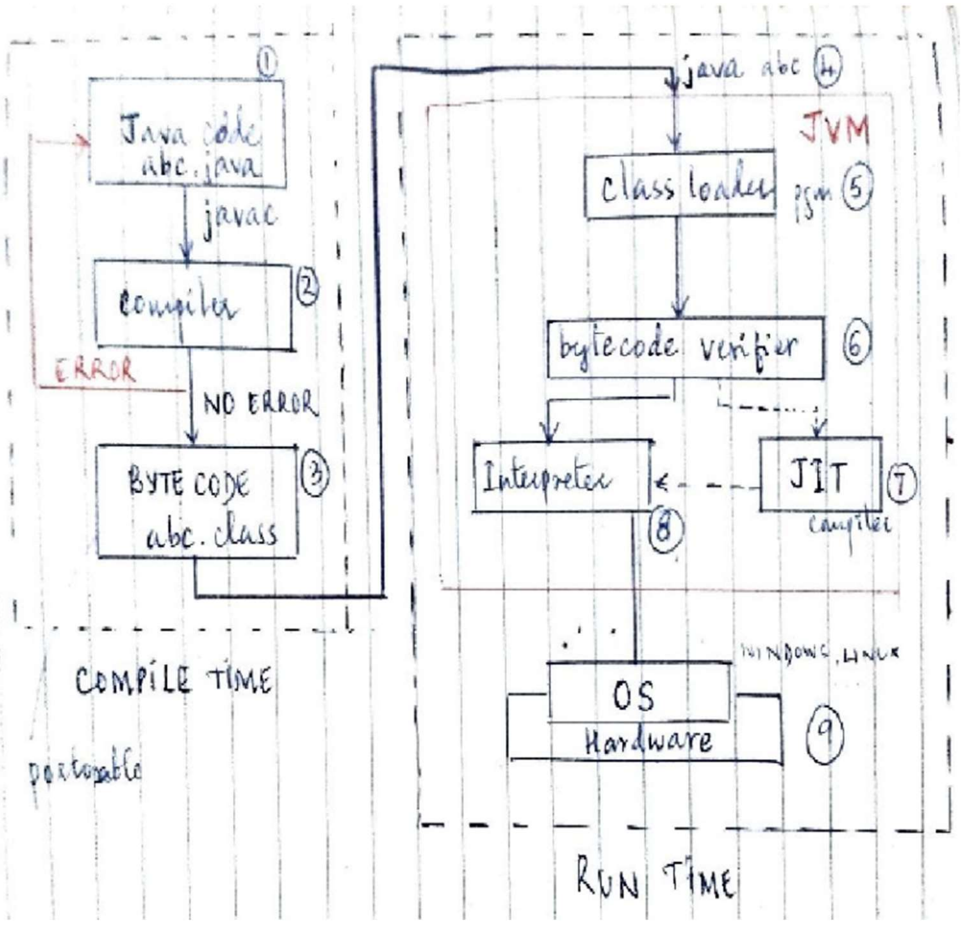
```
//save by A.java

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();
  }
}
```

KRUTHIK B(1DA19CS077 B SECTION)

5)

```
public class Helloworld {
                public static void main(String[] args){
                                System.out.println("HELLO WORLD");
                }//End of main
}//End of FirstJavaProgram Class
```



(Just change the name of prgm from abc.java to Hello.java) and explain the block diagram in ur own words .

->To compile the Example program, execute the compiler, javac, specifying the name of the source file on the ***command line***, as shown here:

```
C:\>javac Helloworld.java
```

1)The javac compiler creates a file called Example.class that contains the bytecode version of the program.

2)The Java bytecode is the intermediate representation of your program that contains instructions the Java Virtual Machine will execute. Thus, the output of javac is not code that can be directly executed.

3)To actually run the program, you must use the Java application launcher called java.

4)To do so, pass the class name Example as a command-line argument, as shown here:

```
 C:\>java Helloworld
```

5)When the program is run, the following output is displayed:

```
 HELLO WORLD
```

->To run java program in ***eclipse***

First we have to create a java project

1)Open eclipse and click File>New>Java Project

2)Provide the project name(Helloworld) and click on finish button , new project is created

3)In the package explorer select the project which we have created

4)Right-click on the src folder , select New>class from the submenu . Provide the class name and click on Finish button .

5)Write the program and save it.

6)Click on the run meny and select run

7)Output

```
HELLO WORLD
```

KRUTHIK B(1DA19CS077 B SECTION)