Object Oriented Programming

(Oops)

In java

Made By : Raushan kumar

➔Copy code ➔Past code ➔Compile➔ Do experiment

⇒ Object Oriented Programming System / Structure

⇒Oop is a programming paradigm / methodology

→Object Oriented Paradigm

→Procedural Paradigm

→ Functional Paradigm

→Logical Paradigm

→ Structural Paradigm

6 main Pillars of Oops are

1. Class
2. Object/Methods
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Oop's

1. Class is the collection of objects .
2. Class is not a real world entity . it is just a template or blueprint or prototype
3. Class does not occupy memory

Syntax :

```
Access-modifiers class ClassName
       {
          → Methods

         → Constructors

         → fields

         → Blocks

        → nested class
       }
```

Method Syntax

```
access -modifiers return-type methodName(list of Parameters)
{


}
```

Object

- Object is an instance of class
- Object is real world entity
- Object occupies memory

Object consist of

1. Identity : Name
2. State/Attribute: color , breed ,age
3. Behavior : eat ,run

Ways to create Object

- New() method
- newInstance() method
- Clone() method
- Deserialization
- Factory methods

Phases of Object creation

1. Declaration : Animal buzo ;
2. Instantiation : Animal buzo = new Animal();
3. Initialization :

```
Animal buzo = new Animal();
   buzo.run();
```

Example:

```java
class Test
{
    void show()
    {
        System.out.println("yeah this is calling Test class");
    }
}


public class Demonstration {


    public static void main(String[] args)
    {
        //Creating a object by new method
        Test NewObj= new Test();
        //Here I am calling method name
        NewObj.show();

    }
}
```

Passing the variable in object

1. By reference
2. By methods
3. By Constructor

Example

```java
class Test
{
    int rollNumber;
    String name;
    Test(String name ,int rollNumber)
    {
        this.name= name;
        this.rollNumber=rollNumber;
    }
    void setMetaData(String name ,int rollNumber)
    {
        this.name=name;
        this.rollNumber=rollNumber;
    }
    void show()
    {
        System.out.println("yeah this is calling Test class and Name
is "+ this.name + " rollNumber is "+this.rollNumber);
    }
}


public class Demonstration {
    public static void main(String[] args)
    {
        //Initializing the variable by constructor
        Test NewObj =new Test("raushan",43);


        //Initializing the variable by instance
        NewObj.name="raushan";
        NewObj.rollNumber=43;


        //Initializing the object by method calling
        NewObj.setMetaData("raushan",43);

        //Here I am calling method name
        NewObj.show();

    }
```

Constructor

1. Constructor is a block (similar to method) having same name as that of class Name
2. Constructor does not have any return type not ever void
3. The only modifiers applicable for constructor are public , protected , default and private

Use of Constructor

So constructors are used to assigning values to the class variables at the time of object creation, either explicitly done by the programmer or by Java itself (default constructor).

Types of Constructors in Java

There are two types of constructors in java:

- No-argument constructor
- Parameterized Constructor

Demonstration

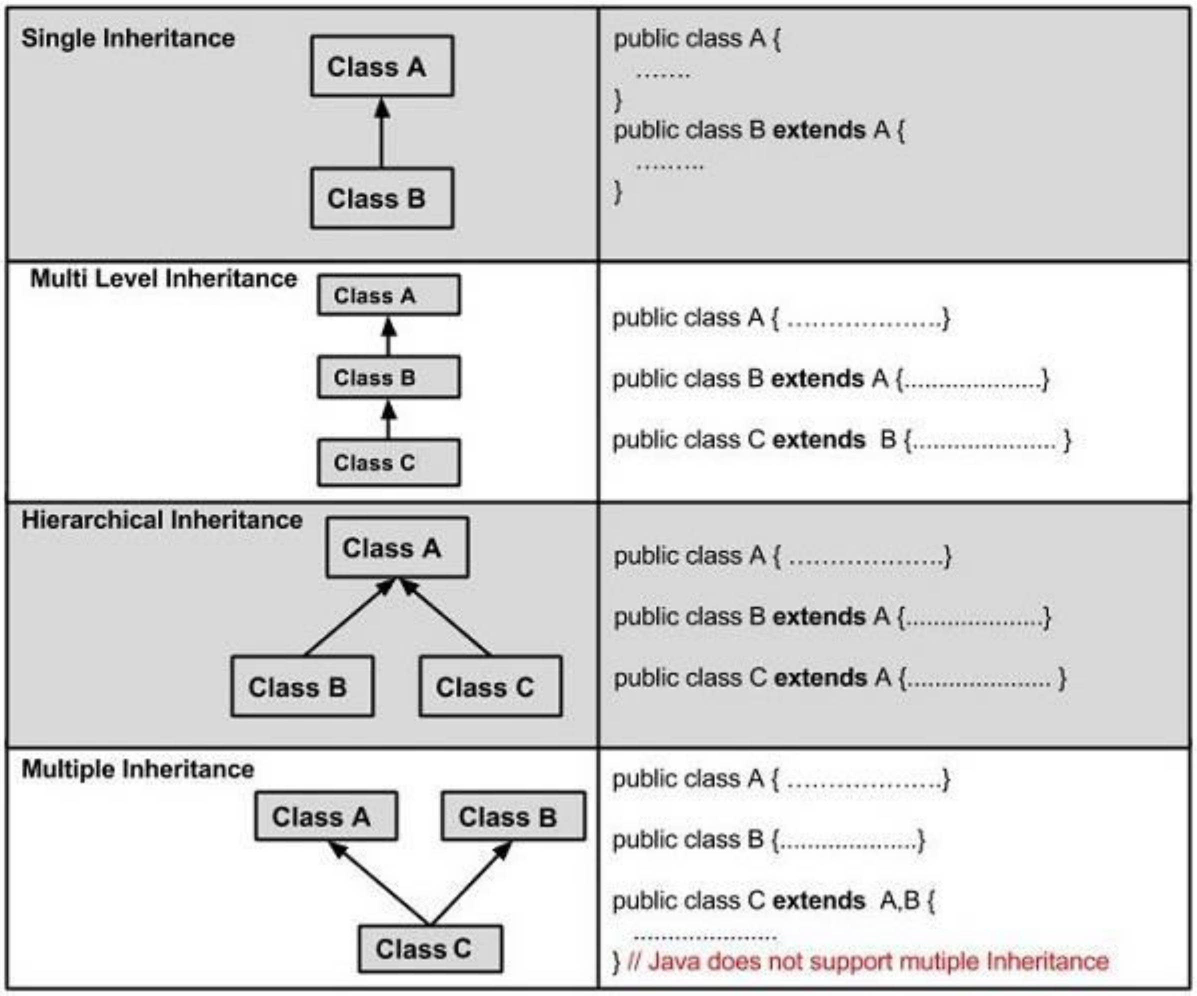
```
class Test
{
    int rollNumber;
    String name;
    Test()
    {
        System.out.println("this is no argument constructor ");
    }
    Test(String name ,int rollNumber)
    {
        System.out.println("this is parameterised constructor");
        this.name= name;
        this.rollNumber=rollNumber;
    }

    void show()
    {
        System.out.println("yeah this is calling Test class and Name
is "+ this.name + " rollNumber is "+this.rollNumber);
    }
}


public class Demonstration {


    public static void main(String[] args)
    {
        //No argument constructor
        Test NewObj= new Test();

        //Parameterized constructor
        Test NewObj1= new Test("raushan",43);

        NewObj.show();
        NewObj1.show();
        }
}
```

Inheritance

⇒ TYPES OF INHERITANCE

1. Single level Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance
5. Hybrid Inheritance

But here Multiple Inheritance and Hybrid Inheritance is not supported by Java but by INTERFACE , multiple Inheritance can be done .


Class A
Class B
Class A
Class B
Class C
Class A
Class B
Class C
Class A
Class B
Class C

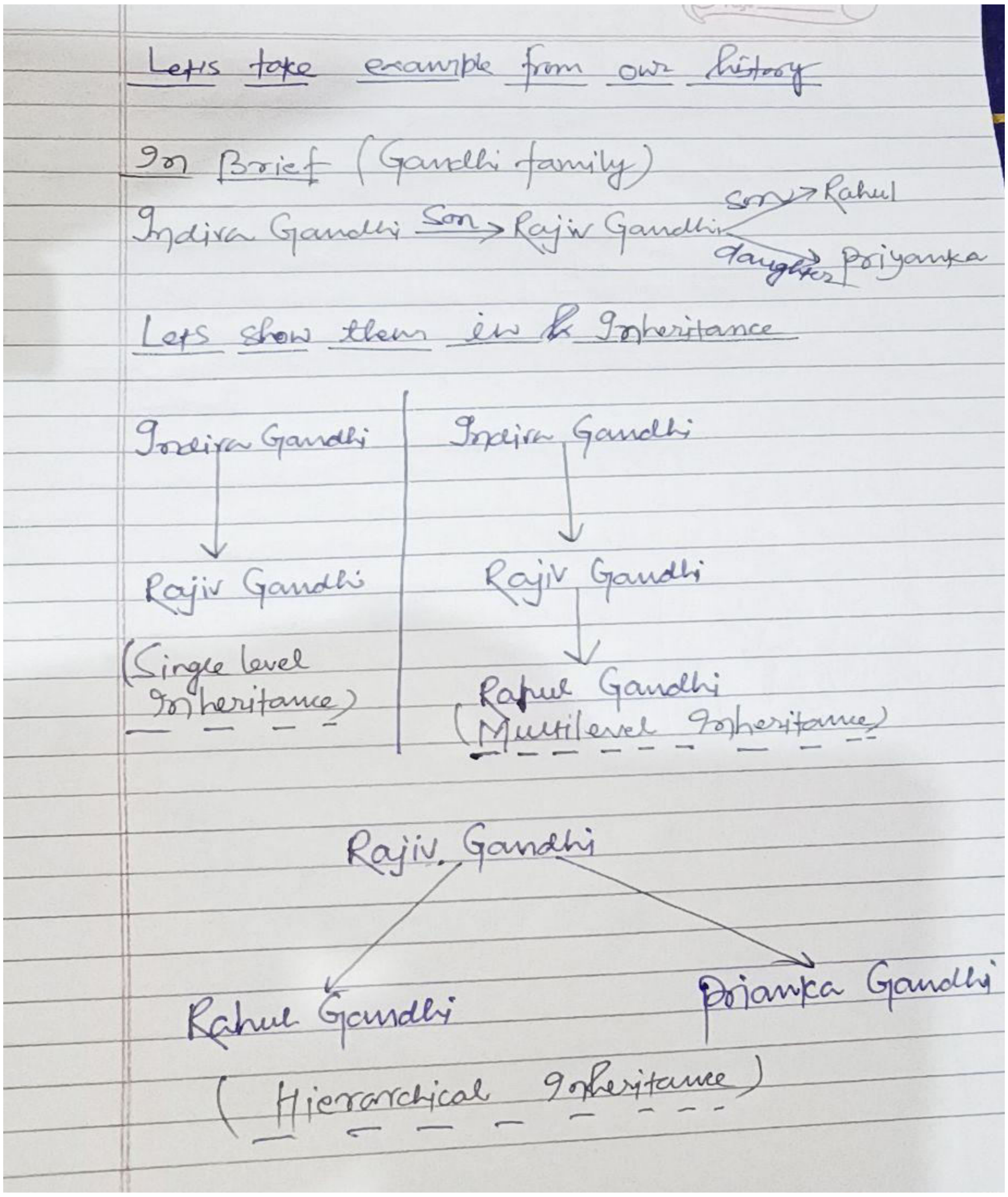
Single Inheritance	public class A {
	……..
	}
	public class B **extends** A {
	………
	}

Multi Level Inheritance	public class A { ……………….}
	public class B **extends** A {……………….}
	public class C **extends** B {…………………. }

Hierarchical Inheritance	public class A { ……………….}
	public class B **extends** A {……………….}
	public class C **extends** A {…………………. }

Multiple Inheritance	public class A { ……………….}
	public class B {……………….}
	public class C **extends** A,B {
	…………………
	} // Java does not support mutiple Inheritance

Lets a example from Gandhi family



Lets take example from our history

In Brief (Gandhi family)

$$\begin{tikzcd}
 & & \text{Rahul} \\
\text{Indira Gandhi} \arrow[r, "\text{Son}"] & \text{Rajiv Gandhi} \arrow[ur, "\text{son}"] \arrow[dr, "\text{daughter}"'] & \\
 & & \text{Priyanka}
\end{tikzcd}$$

Lets show them in ~~k~~ Inheritance

$$\begin{tikzcd}
\text{Indira Gandhi} \arrow[d] \\
\text{Rajiv Gandhi}
\end{tikzcd}$$

(Single level Inheritance)

$$\begin{tikzcd}
\text{Indira Gandhi} \arrow[d] \\
\text{Rajiv Gandhi} \arrow[d] \\
\text{Rahul Gandhi}
\end{tikzcd}$$

(Multilevel Inheritance)

$$\begin{tikzcd}
 & \text{Rajiv Gandhi} \arrow[dl] \arrow[dr] & \\
\text{Rahul Gandhi} & & \text{Priyanka Gandhi}
\end{tikzcd}$$

(Hierarchical Inheritance)

Their code implementation ((single + multi+ hierarchical) inheritance)

```
class IndiraGandhi
{
    /*
    Here I will take height , Profession ,age,
     */
    int height;
    String profession;
    String family;
    int age;
    String gender;
    String name;
    static int count=0;
    IndiraGandhi(String profession,String family)
    {
        /*
        this keyword is to eliminate the confusion between class
attributes and parameters with the same name ( because a class
attributes is shadowed by a method or constructor parameter
         */
        count++;
        this.family=family;
        this.profession=profession;

        System.out.println("This is constructor of IndiraGandhi
class");
    }

    void setDetails(int age,int height,String gender,String name)
    {
        this.age= age;
        this.height=height;
        this.gender=gender;
        this.name= name;
    }
    void showDetails()
    {
        System.out.println("My name is "+this.name + " my profession
is "+ this.profession +" and my family is " +this.family +" my age is
"+this.age + " height "+ this.height + "cm"+" Gender "+ this.gender);
    }
    void showCount()
    {
        System.out.println("count "+this.count);
    }
    void noOfChildren()
    {
        System.out.println("there are 5 children");
    }

}
class RajivGandhi extends IndiraGandhi
{
    String gender;
    int height;
    int age;
    String name;
    RajivGandhi(String profession,String family)
    {
        super(profession,family);
        super.noOfChildren();
        System.out.println("this is the constructor of Rajiv Gandhi
class");
    }




    void setDetails(int age,int height,String gender,String name)
    {
        this.age= age;
        this.height=height;
        this.gender=gender;
        this.name= name;
    }
    void quality()
    {
        System.out.println(this.name +" was a dynamic Prime minister
of Developing India");
    }
    synchronized void showDetails()
    {
        System.out.println("My name is "+this.name + " my profession
is "+ this.profession +" and my family is " +this.family +" my age is
"+this.age + " height "+ this.height + "cm"+" Gender "+ this.gender);
    }
}
class RahulGandhi extends RajivGandhi
{
    String gender;
    int height;
    int age;
    String name;
    RahulGandhi(String profession,String family)
    {
        super(profession,family);
        System.out.println("this is constructor of Rahul Gandhi
class");
    }
    void setDetails(int age,int height,String gender,String name)
```

```java
    {
        this.age= age;
        this.height=height;
        this.gender=gender;
        this.name= name;
    }
    void showDetails()
    {
        System.out.println("My name is "+this.name + " my profession
is "+ this.profession +" and my family is " +this.family +" my age is
"+this.age + " height "+ this.height + "cm"+" Gender "+ this.gender);
    }
}
class PriyankaGandhi extends RajivGandhi
{
    String gender;
    int height;
    int age;
    String name;
    PriyankaGandhi(String profession,String family)
    {
        super(profession,family);
        System.out.println("this Priyanka Gandhi class");
    }
    void setDetails(int age,int height,String gender,String name)
    {
        this.age= age;
        this.height=height;
        this.gender=gender;
        this.name= name;
    }
    void showDetails()
    {
        System.out.println("My name is "+this.name + " my profession
is "+ this.profession +" and my family is " +this.family +" my age is
"+this.age + " height "+ this.height + "cm"+" Gender "+ this.gender);
    }
}

public class Group_discussion {


    public static  void main(String[] args) {

        System.out.println("Yeah we are discussing in the group");

        // initializing state and variables of the respective class
        IndiraGandhi indira =new IndiraGandhi("politician","Gandhi");
        indira.setDetails(80,150,"female","Indira Gandhi");

        RajivGandhi rajiv= new RajivGandhi("politician","Gandhi");
        rajiv.setDetails(60,160,"male","Rajiv Gandhi");
        RahulGandhi rahul= new RahulGandhi("politician and stand up
comedian","Gandhi");
        rahul.setDetails(50,165,"male","Rahul Gandhi");

        PriyankaGandhi priyanka = new
PriyankaGandhi("politician","Gandhi");
        priyanka.setDetails(45,130,"female","Priyanka Gandhi");

        priyanka.showDetails();
        System.out.println("<-----------------------------------------
->");
        rahul.showDetails();
        System.out.println("<-----------------------------------------
->");
        rajiv.showDetails();
        System.out.println("<-----------------------------------------
->");
        indira.showDetails();
        System.out.println("<-----------------------------------------
->");
        priyanka.quality();

        System.out.println("<-----------------------------------------
->");
        priyanka.showCount();

    }



}
```

Polymorphism In Oops

1.Compile time Polymorphism

➔ Static Polymorphism

➔Its achieve by Method Overloading

2.Runtime Polymorphism

➔Dynamic Polymorphism

➔Its achieve by Method Overriding

Method Overloading	Method Overriding
1. Same Name hona chahiye	Same Name
2 . Same Class hona chahiye	Different Class
3 . Different Argument hona chahiye	Same Argument
4 . Here Inheritance according to requirement	Inheritance Necessary

Method Overloading

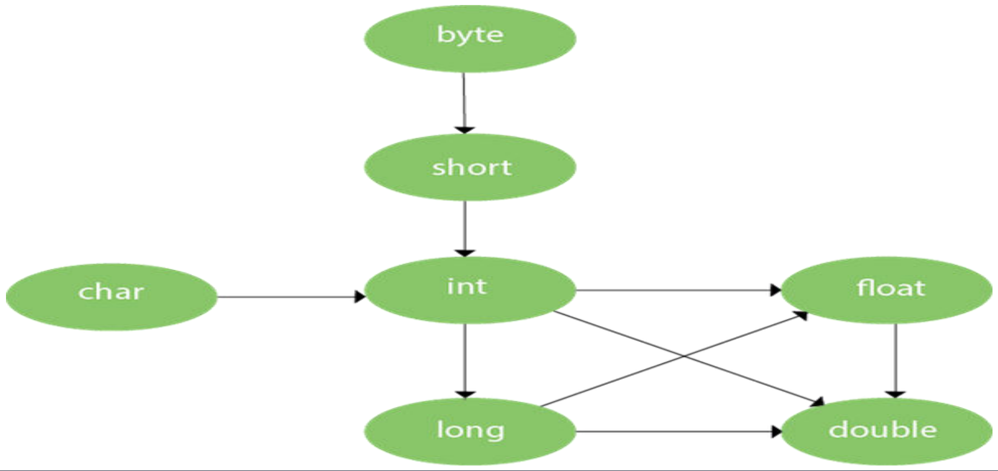
➔Method overloading *increases the readability of the program.*

Different ways to overload the method

There are two ways to overload the method in java

➔By changing number of arguments
➔By changing the data type

Promotion in Overloading



```
class AdditionalArea
{



    int area(int a, int b, int c, int d)
    {
        System.out.println("Yeah this is abstract class of 4 
parameter");
        return (a*b*c*d);
    }
}
class Area extends AdditionalArea
{

    //return type change karne se method overlaoding nahi hoga
    //while resolving Overloaded Methods, Compiler will always give 
precedence for the child type argument than compared with the parent 
type argument

    void show(Object obj)
    {
        System.out.println("I am Parent of all classes in java");
    }
    // String and StringBuffer are at same level so "NULL" cannot be 
referred , if referred , ambiguous error will occur
    void show (StringBuffer obj)
    {
        System.out.println("My name(StringBuffer) "+ obj);
        //show(new StringBuffer("ramlal");
    }
    /* void show(String name)
    {
        System.out.println("My name is "+ name);
    }*/
    void show(int...obj)
    {
        System.out.println("this is class of multiple (triple dot
class)");
        //the varargs allows the method to accept zero or multiple
arguments .
        //if we dont know how many arguments we will have to pass in
the method ,arargs is the better approach .

        System.out.println(obj[0]+ obj[1]);
    }
    void show(char ch)
    {
        System.out.println("the character is "+ch);
    }
    void show(int ch)
    {
        System.out.println("the character is "+  ch);
    }
    int area (int length, int breadth)
    {
        System.out.println("I am two INTEGER argument method named
length and breadth only ");
        return (length*breadth);
    }
    int area(int length,int breadth,int depth)
    {
        System.out.println(" I am three INTEGER argument method named
length , breadth and depth ");
        return (length*breadth*depth);
    }
    int area(int length ,double breadth)
    {
        System.out.println("I am two (INTEGER,Double) argument method
named length and breadth");

        return (int)(length*breadth);
    }
}




public class Group_discussion {


    public static  void main(String[] args) {


        Area area = new Area();
```

```java
        area.area(2,4);
        area.area(2,4.0);
        area.area(2,3,4,5);
        area.show("raushan kumar"); //agar string type ka nahi hai toh
, Oo aapne parent ko call kar lega that is object ;
        area.show(1,2,3,34,235,43);
        area.show('a');


    }
}
```

Question : Can we overload main method ?
➔Yes , we can !

Implementation

```java
public class Demonstration {


    public static void main(String[] args)
    {
        System.out.println("this is main method ");
        Demonstration demo = new Demonstration();
        demo.main(2);

    }
    public static void main(int a)
    {
        System.out.println("this int argument main method ");
    }
}
```

METHOD OVERRIDING

Usage of Java Method Overriding

➔Method overriding is used to provide the specific implementation of a method which is already provided by its super class.

➔Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

➔The method must have the same name as in the parent class

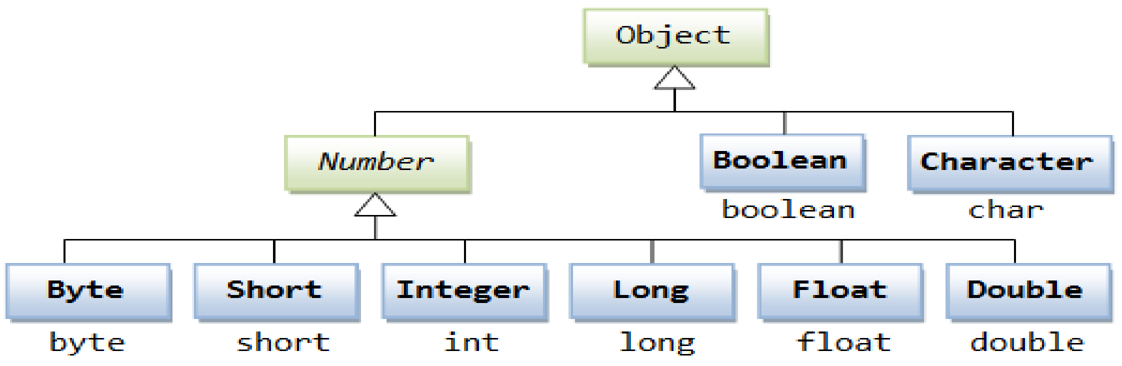
➔The method must have the same parameter as in the parent class.

➔There must be an IS-A relationship (inheritance).

WHICH METHODS CANNOT OVERRIDE?

1 . Final methods cannot be overridden
2 . Static methods cannot be overridden
3 . Private methods cannot be overridden

Wrapper Class in java



Implementation

```
class Test
{
    int rollNumber;
    String name;
    Test()
    {
        System.out.println("this is no argument constructor ");
    }
    void show()
    {
        System.out.println("yeah this is calling Test class and Name
is "+ this.name + " rollNumber is "+this.rollNumber);
    }
    void display()
    {
        System.out.println("this is test class ");
    }

}
class Test1 extends Test
{
    Test1()
    {
        System.out.println("this is constructor class and no argument
");
    }
    //here method riddes happens
    void display()
    {
        System.out.println("This is Test1 class and that is inherited
by test class");
    }

}
```

```
public class Demonstration {


    public static void main(String[] args)
    {
        Test1 test1 = new Test1();
        test1.display();



    }
}
```

Question :
Do overriding method must have same return type (or subtype) ?

➔From java 5.0 onwards it is possible to have different return type for a overriding method in child class but child's return type should be sub-type of parent's return type . This phenomena is known as COVARIANT return type.

OVERRIDING AND ACCESS-MODIFIERS

➔The access modifier for an overriding method can allow more , but not less , access than the overridden method . For example , a protected instance method in the super-class can be made public , but not private , in the subclass . Doing so , will generate compile-time error .

Method overriding by changing the return type

```
class Test
{
    int rollNumber;
    String name;
    Test()
    {
        System.out.println("this is no argument constructor ");
    }
    void show()
    {
        System.out.println("yeah this is calling Test class and
Name is "+ this.name + " rollNumber is "+this.rollNumber);
```

```
    }
    void display()
    {
        System.out.println("this is test class ");
    }
    public Object experimentWithReturnType(int a)
    {
        return a*a;
    }
}
class Test1 extends Test
{
    Test1()
    {
        System.out.println("this is constructor class and no
argument ");
    }

    void display()
    {
        System.out.println("This is Test1 class and that is
inherited by test class");
    }
    //this is working fine means here , we change return type in
overriding ,if we use subclass in child class with respect to
the parent class. Object is parent of all class.
    public String  experimentWithReturnType(int a)
    {
        // return (a*a);

        return "raushan ";
    }
}


public class Demonstration {


    public static void main(String[] args)
    {
        Test1 test1 = new Test1();
        // test1.display();
        String ans= test1.experimentWithReturnType(2);
        System.out.println(ans);
    }
}
```

Encapsulation

➔ Encapsulation in java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit .

Variables + Methods = Encapsulation

➔Steps to achieve Encapsulation

1 . Declare the variable of a class as private.
2 . Provide public setter and getter methods to modify and view the variables values.

→In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class. This concept is known as Data Hiding.

```
class ImplementEncapsulation
{
    private int rollNumber;
    private String name;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public int getRollNumber() {
        return rollNumber;
    }

    public void setRollNumber(int rollNumber) {
        this.rollNumber = rollNumber;
    }
}

public class Encapsulation {
    public static void main(String[] args)
    {
        ImplementEncapsulation Encapsulation= new
ImplementEncapsulation();
        Encapsulation.setName("Abhishek Gupta");
        Encapsulation.setRollNumber(45);
        System.out.println(Encapsulation.getName());
        System.out.println(Encapsulation.getRollNumber());
    }
}
```

Code Reusability

- Inheritance(IS-A relation)
- HAS-A relation
- Polymorphism

Security

- Abstraction
- Data Hiding
- Enscapsulation
- Tightly coupled classes

ABSTRACTION

Abstraction	Encapsulation
1 . Abstraction is detail hiding (implementation hiding)	Encapsulation is data hiding (information hiding)
2 . Data Abstraction deals with exposing the interface to the user and hiding the details of implementation .	Encapsulation groups together data and methods that act upon the data .

➔Abstraction is hiding internal implementation and just highlighting the setup services that we are offering.

➔Abstraction can be achieved by 2 ways

1 . Abstract Class (from 0 to 100% abstraction)

2 . Interface (100% abstraction)

A method without body (no implementation) is known as abstract method.

A method must always be declared in an abstract class or we can say that if a class has an abstract method, it should be declared abstract as well.

If a regular class extends an abstract class , then the class must have to implement all the abstract methods of abstract parent class or it has to be declared abstract as well .

Abstract method in an abstract class is meant to be overridden in derived concrete classes otherwise compile-time error will be thrown.

ABSTRACT CLASS IMPLEMENTATION

```java
//We can not create object of abstract class and interface .
abstract class Parent1
{
    abstract void show2();
    abstract void example();
    void show1()
    {
        System.out.println("this is show1 method of Parent ");
    }
}
class Childs extends Parent1 {
    void show2()
    {
        System.out.println("This is Parent1 methods");
    }
    void example()
    {
        System.out.println("this is example class of Parent1");
    }
    void show3()
    {
        System.out.println("this is show3 method of child's class");
    }
}

public class AbstractClass {
    public static void main(String[] args)
    {
        Childs child1 = new Childs();
        child1.example();
        child1.show3();
        child1.show1();
    }
}
```

INTERFACE (100% ABSTRACTION)

→Interface is a mechanism to achieve abstraction in java.
→Interface is similar to abstract class but having all the methods of abstract type i.e it cannot have a method body.
→Since java 8, we can have default and static methods in an interface .
→Since java 9, we can have private methods in an interface .

Interfaces similar to abstract class but having all the methods of abstract type.

→Loose coupling is a design goal that seeks to reduce the inter-dependencies between components of a system with the goal of reducing the risk that changes in one component will require changes in any other component. Loose coupling is much more generic concept intended to increase the flexibility of system, make it more maintainable and make the entire framework more stable.
→Loose coupling is much more generic concept
➔Through this we can achieve multiple inheritances.

Interface implementations + multi-inheritance

```
interface MultiInheritanceClass1
{
    void multiInheritanceMethod1();

}
interface MultiInheritanceClass2
{
    void multiInheritanceMethod2();
    void multiInheritanceMethod1();

}
interface InterfaceClass
{
    void interfaceShow1();
    void interfaceShow2();// compiler will automatically add public
abstract void interfaceShow2();
    int temp=5;// compiler will add automatically public static final
int temp=5;
    // as interface , we can not create concrete method here but by
writing access modifier (default, static,private) we can write
concrete method here as well.
    static void interfaceShow3()
    {
        System.out.println("This method inside interface ");
    }
    default void interfaceShow4()
    {
        System.out.println("This method inside interface ");
    }
    private void interfaceShow5()
    {
        System.out.println("This method inside interface ");
    }
}
// MULTI-INHERITANCE IN JAVA THROUGH INTERFACE IS HAPPENING
```

```java
class Tests implements InterfaceClass, MultiInheritanceClass2,
MultiInheritanceClass1
{
    public void multiInheritanceMethod2()
    {
        System.out.println("this is method of mult-inheritance of
multiInheritanceMethod2()");
    }
    public void multiInheritanceMethod1()
    {
        System.out.println("this is method of mult-inheritance of
multiInheritanceMethod1()");
    }
    public  void interfaceShow1()
    {
        System.out.println("this is interface  class of method
interfaceShow1()");
    }

    public void interfaceShow2()
    {
        System.out.println("this is interface  class of method
interfaceShow2()");
    }
}

public class InterfaceImplementation {
    public static void main(String[] args)
    {
        Tests test =new Tests();
        test.interfaceShow1();
        System.out.println("<-------------------------------->");
        // Tests.interfaceShow3();// showing error, why ?
        InterfaceClass.interfaceShow3(); //correct
        test.interfaceShow4();
        //  test.interfaceShow5();// showing error , clear because it
private we can not access it directly


    }
}
```

Similarities between Abstract class and Interfaces

1 . Both can contain abstract methods.

2 . We cannot create an instance of abstract class and interfaces .

DIFFERENCE

Abstract class	Interface

Abstract class can have instance method that implements a default constructor.	Methods of a java interface are implicitly abstract and cannot have implementation
An abstract class may contain non-final variables	Interface contains public, static and final variables only.
Methods and variable can have any access-modifiers i.e. public, protected, default and private.	Methods and variable are always public.
Java abstract class should be extended using keyword “extends”	Java interface should be implemented using keyword “ implements”
An abstract class can extend another java class and implement multiple java interface.	An interface can extend another java interface only

USE OF “THIS” KEYWORD

1 . This keyword can be used to refer the current class instance variable .
2 . This keyword can be used to invoke current class method (implicitly).
3 . This() can be used to invoke current class constructor .
4 . this can be used to pass an argument in the method call .
5 . this can be used to pass as an argument in the constructor call .
6 . this can be used to return the current class instance from the method .

Now Implementation of “this” keywords

```
//Demonstration of this keyword
final class Sample1
{

    Sample1(ThisKeyWord temp)
    {
        temp.sampleMethod1();
        System.out.println("this Sample1 method constructor");
    }
}

class ThisKeyWord
```

```
{
    String name;
    String collegeName;

    ThisKeyWord(String name, String collegeName)
    {

        //1. current class instance
        System.out.println("this is current class constructor");
        this.name=name;
        this.collegeName=collegeName;
    }
    ThisKeyWord(int a)
    {
        //3. calling constructor
        this("raushan","IIEST shibpur");
        System.out.println("this is int type constructor ");
    }
    void sampleMethod()
    {
        System.out.println("this is sample Method class");
    }
    void sampleMethod1()
    {
        //2. current class method calling
        this.sampleMethod();

        System.out.println("this is sample method 1 class");
    }
    void example1(ThisKeyWord temp)
    {
        temp.sampleMethod1();
        System.out.println("this calling of example1 method from 
SampleExample1 class");
    }
    void callingThisSame()
    {
        //this keyword can be pass in a argument
        example1(this );
    }
    //5. this argument can be passed in the constructor call
    void callingSample1Constructor()
    {

        Sample1 sample1= new Sample1(this);

    }
    //6.  return the current class instance from the methods
    ThisKeyWord returnSameClass()
    {
        System.out.println("this is return same class method ");
        return this;
    }
}




public class KeyWordDemonstration {
    public static void main(String[] args)
    {
        ThisKeyWord keyword = new ThisKeyWord("raushan","IIEST
shibpur");
        keyword.callingThisSame();
        System.out.println("<-------------------------------->");
        keyword.callingSample1Constructor();
        System.out.println("<-------------------------------->");
        ThisKeyWord temp= keyword.returnSameClass();
        temp.sampleMethod1();
    }
}
```

Super Keyword

"Super" keyword is a reference variable which is used to refer immediate parent class object.

1 . "super" keyword can be used to refer immediate parent class instance variable.

2 . "super" keyword can be used to invoke immediate parent class method.

3 . Super() can be used to invoke immediate parent class constructor .

And their Implementation

```
class Parent
{
    int rollNumber;
    Parent(int rollNumber)
    {
        this.rollNumber=rollNumber;
        System.out.println("this is parent class constructor ");
    }
    void parentMethod()
    {
        System.out.println("this is parentMethod of parent class");
    }
}
class Child extends Parent
{
    Child(int rollNumber)
    {
        //this is parent class constructor
        super(rollNumber);
    }
    void getParentVariable()
    {
        // Here i am getting parent instance variable
        System.out.println("Roll number is "+ super.rollNumber);
    }
    void getParentMethod()
    {
        //Here i am getting parent method through super keyword;
        System.out.println("getting parent method");
        super.parentMethod();
    }
}

public class SuperKeyWordDemonstration
{
    public static void main(String[] args)
    {
        Child child =new Child(43);
        System.out.println("<-------------------------------->");
        child.getParentVariable();
        System.out.println("<-------------------------------->");
        child.getParentMethod();
    }
}
```

FINAL KEYWORD

➔If we create any final variable, it becomes constant. We cannot change the value of the final variable.

➔If we create any final method, we cannot override it.

➔If we create any final class, we cannot extend it or inherit it.

Access Modifiers	Non access Modifiers
Public	Static
Private	Final
Protected	Abstract
Default (No modifier)	Synchronized
	Transient
	Volatile
	Strictfp

STATIC KEYWORD

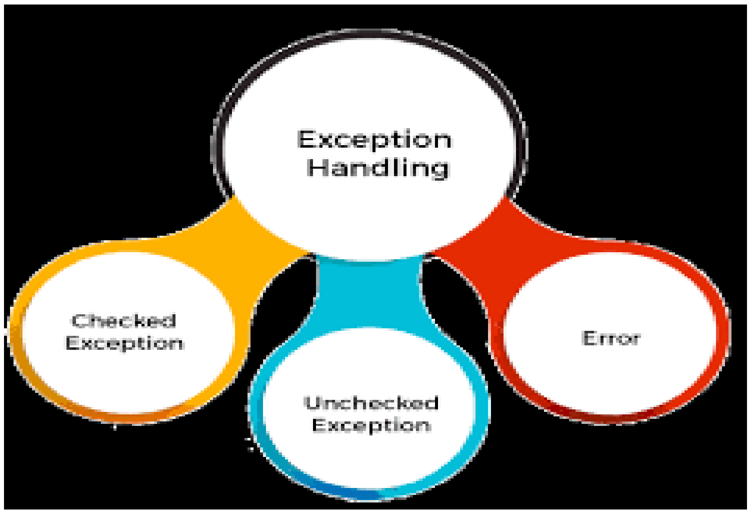
➔Static methods belongs to class , not object.
➔Variable (class level), Methods , block , nested class(only inner class can be static).
➔Static variables are used for memory management .
➔When a variable is declared as static , then a single copy of variable is created and shared among all objects at class level .
➔The static variable gets memory only once in the class area at the time of class loading .
➔A static method can access only static data . It cannot access non-static data i.e. instance data.
➔A static methods can call only other static methods and can not call a not-static method.
➔A static method cannot refer to “this” or “super” keyword in anyway .
➔Static block executes automatically when the class is loaded in the memory .
➔Static block is executed at class loading , hence at the time of class loading if we want to perform any activity , we have to define that inside static block .
➔Static block is used to initialize the static members.

Static keyword Implementation

```
package com.raushan;

class Student
{
    int rollNumber;
    static String collegeName; // Static because for every student
college name will be same .
    static int counter;// this will count the number of student i.e.
total no of object made from student class.
    // Now we will use static block for initialization of static
instance member
    static
    {
        collegeName="Indian Institute of Engineering Science and
Technology, Shibpur";
```

```
        counter=0;
    }
    Student(int rollNumber)
    {
        this.rollNumber=rollNumber;
        counter++;//increasing the counter for every object creation
of student .
    }
    static void method2()
    {

        System.out.println("this is method2 class");

        // method1(); in static method , only static method calls
otherwise it will compile error.
        //  System.out.println(rollNumber);// this will give wrong ,
because in static method only static method or variable are called;
        System.out.println(collegeName);// this is fine
    }
    void method1()
    {
        // static int temp=45; //THIS IS SHOWING WRONG BECAUSE WE CAN
NOT CREATE STATIC VARIABLE INSIDE THE METHODS.
        method2();// we can call static method and instance variable
inside the method
        System.out.println(rollNumber);

        System.out.println("this is method1 class");
    }
}

public class StaticKeyword {

    public static void main(String[] args)
    {
        Student student =new Student(43);
        Student student1 =new Student(45);
        Student student2 =new Student(46);
        Student student3=new Student(47);
        Student student4=new Student(48);
        Student student5 =new Student(49);

        System.out.println(Student.counter);
        //static member is property of class , that why it can be
called directly by Student.counter and same for method as well.



    }
}
```

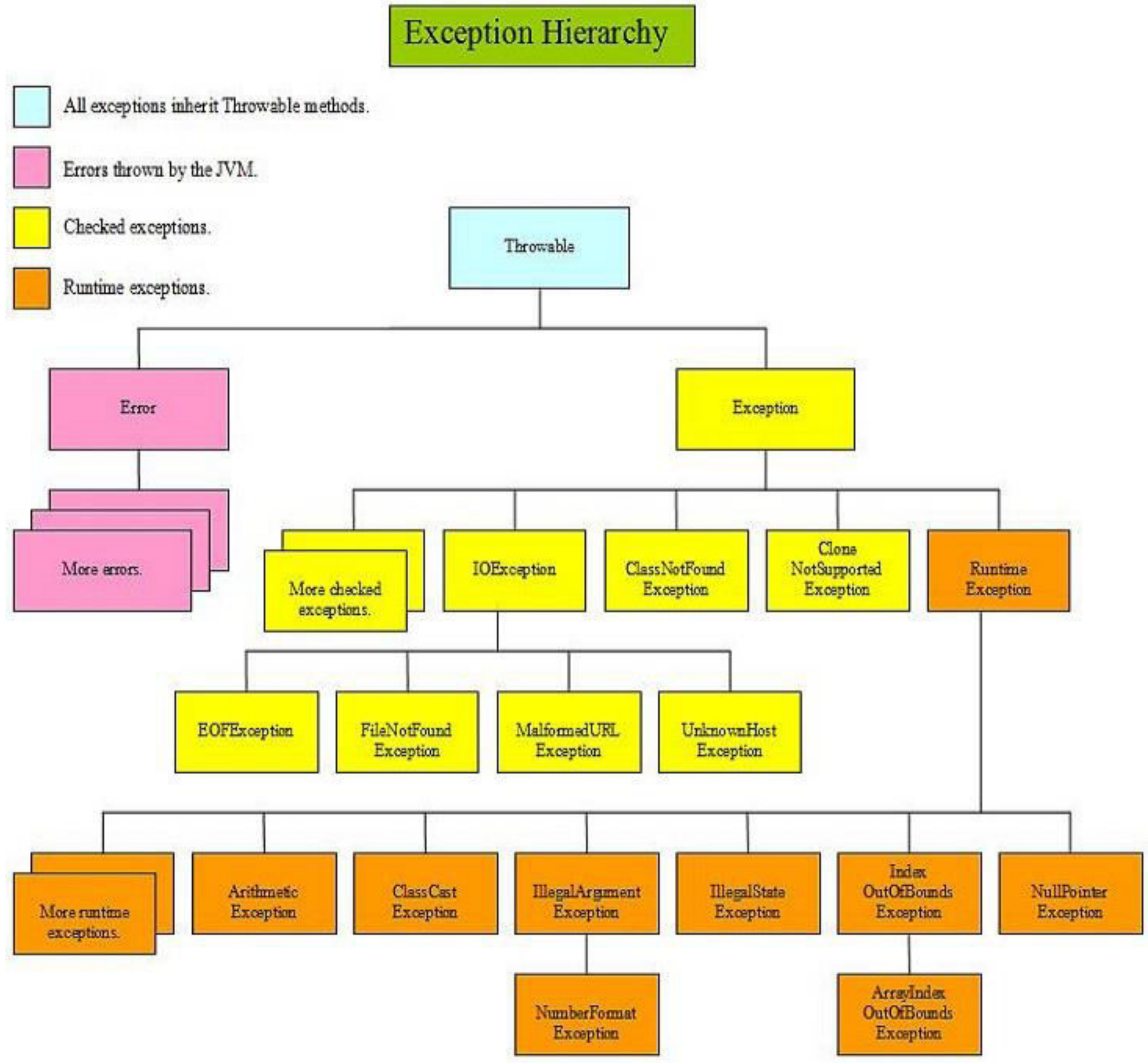
EXCEPTION AND ERROR HANDLING IN JAVA



➔An exception is an unwanted or unexpected event , which occurs during the execution of a program i.e. at run time , that disrupts the normal flow of the program.
➔Errors are occurred because of lack of system Resources; not by our programs and thus , programmer cannot do anything .
➔Throwable is the parent class of Exception class.

DIFFERENCE BETWEEN EXCEPTION AND ERROR

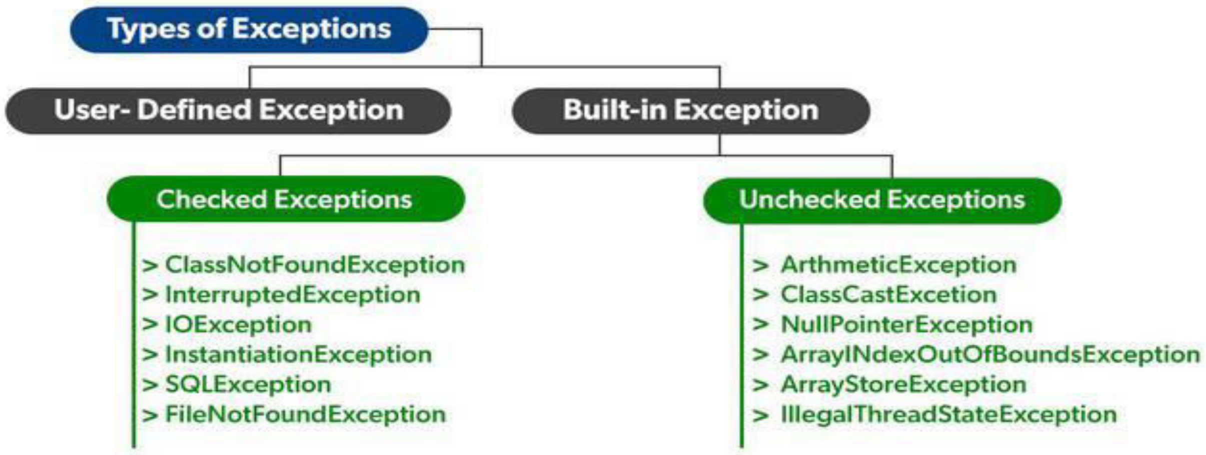
Exception	Error
Exception occurs because of our programs.	Error occurs because of lack of system resources .
Exception are recoverable i.e. programmer can handle them using try-catch block.	Errors are not recoverable i.e. programmer can handle then to their level .
Exceptions are two types: 1.Compile time Exceptions or checked Exceptions 2.Runtime Exceptions or Unchecked Exceptions	Errors are only of one type: →Runtime Exceptions or Unchecked Exceptions



➔Checked Exception == Compile Time Exception

→Unchecked Exception = Runtime Exception

➔All exception occur at runtime .



Implementation

```
package com.raushan.Exception;
import java.io.*;



public class handling {

    public static void main(String[] args) throws Exception {
        System.out.println("I am going to implements Exception
handling ");
        /*
       // compiler can assume this can throw exception  , so we need
to handle the exception as well and this is checked exception that
means compile time exception
        FileInputStream fis= new FileInputStream("d:/abc.txt");
        */


        /*
        //now unchecked exception that means run time exception,
arithmeticException

        int a=100,b=0,c;
        c=a/b;
        System.out.println(c);
        */



    }
}
```

Whenever there is exception, the method in which exception occurs will create an object and that object will store three things;

1. Exception name
2. Description
3. Stack Trace

We can handle the exception using 5 keywords

1 . Try 2 . catch 3 . Finally 4. Throw 5. Throws

TRY AND CATCH

```
package com.raushan.Exception;
import java.io.*;



public class handling {

    public static void main(String[] args)  throws Exception {

// for example you have two option to connect with database mysql or
oracle then you can use try catch if one fail then other work.
        try
        {
            int a=100,b=0,c;
            c=a/b;
            System.out.println(c);
        }
        catch(ArithmeticException e)// or here we can write Exception
e also because Exception is parent of all class.
        {
            System.out.println(e);
            System.out.println("You can not divide any number by 0 ");
        }

        System.out.println("Yeah after try catch block this statement
will also execute ");
        //agar try catch nahi hota to ohi run jata jaha pe exception
aaya tha  but iss case me pura ke pura chalega





    }
}
```

Ways to print Exception

1 . e.printStackTrace() (most effective method to print exception)
- ➔ This will print Exception name , Description , stackTrace

2 . System.out.println(e) , SOUT(e.toString())
- ➔ This will print Exception name , description

3 . System.out.println(e.getMessage());
- ➔ This will print only description

Implementations

```
package com.raushan.Exception;
import java.io.*;



public class handling {

    public static void main(String[] args) throws Exception {

        try
        {
            int a=100,b=0,c;
            c=a/b;
            System.out.println(c);
        }
        catch(Exception e)
        {
         /*
        1. //implementation of stackTrace()
           e.printStackTrace();

           */

           /*
         2.  //implementation of sout
         System.out.println(e);
         System.out.println(e.toString());
         */

      /*
        3. System.out.println(e.getMessage());

       */

            System.out.println("You can not divide any number by 0 ");
        }
    }
}
```

FINALLY KEYWORD

🡪Finally is the block that is always executed whether exception is handled or not .
→In finally block we can use try and catch also.
→We can use multiple catch blocks with one try block but we can only use single finally block with one try block , not multiple.
→The statements present in the finally block execute even if the try block contains control transfer statements (i.e. jump statements) like return , break or continue;

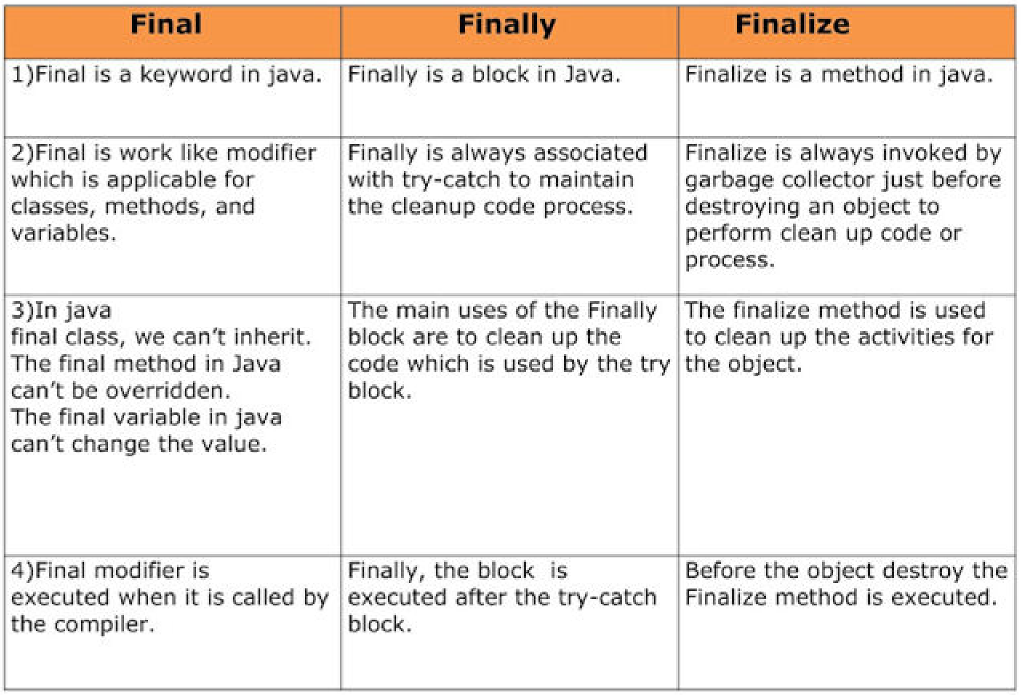
- The possiblility that disturb the execution of the finally block are :
  Case 1 : using of the System.exit() method .
  Case 2: Causing a fatal error that causes the process to abort.
  Case 3: Due to an exception arising in the finally block.
  Case 4: The death of the Thread.

```
package com.raushan.Exception;
import java.io.*;



public class handling {

    public static void main(String[] args) throws Exception {

        try
        {
            //here we generally write risky code
            int a=100,b=0,c;
            c=a/b;
            System.out.println(c);
        }
        catch(Exception e)
        {
            // here we generally write handling code
            System.out.println("you can not divide it by 0");
        }
        finally
        {
            // here we generally write clean up code , like closing
file or close database after use
            System.out.println("Now finally block is under executing
");
        }

        /*
        try
        {
            int a=100,b=0,c;
            c=a/b;
            System.out.println(c);
        }
        finally
        {
            System.out.println("we can write finally keyword without
catch block  but only finally keyword is not allowed it always come
with try and catch block");
        }
     */



    }
}
```

```
package com.raushan.Exception;
import java.io.*;
```

```java
public class handling {
    public static void main(String[] args) throws Exception {
        FileInputStream fis=null;
        try
        {
            fis= new FileInputStream("d:/abc.txt");
        }
        catch(Exception e)
        {
            System.out.println("file is not found");
        }
        finally
        {
            // clean up code
            if(fis!=null)
            {
                fis.close();
            }
            System.out.println("file closed");
        }
    }
}
```

Difference between Final , finally , Finalize



Final	Finally	Finalize
1)Final is a keyword in java.	Finally is a block in Java.	Finalize is a method in java.
2)Final is work like modifier which is applicable for classes, methods, and variables.	Finally is always associated with try-catch to maintain the cleanup code process.	Finalize is always invoked by garbage collector just before destroying an object to perform clean up code or process.
3)In java final class, we can't inherit. The final method in Java can't be overridden. The final variable in java can't change the value.	The main uses of the Finally block are to clean up the code which is used by the try block.	The finalize method is used to clean up the activities for the object.
4)Final modifier is executed when it is called by the compiler.	Finally, the block is executed after the try-catch block.	Before the object destroy the Finalize method is executed.

- Single try , catch , finally wont exist
- We can use multiple catch block provided that should be subset of exception class
- We can use nested try and catch block
- We can also use try catch in catch block
- Try →catch→finally ➔ okay
- Try→finally →catch ➔ give error

```
package com.raushan.Exception;
import java.io.*;



public class handling {

    public static void main(String[] args)  throws Exception
    {
        /*
    //single try block give error
        try
        {
            System.out.println("this single try block ");
        }

        // single catch block error
        catch
        {
            System.out.println("this is catch block");
        }
        //single finally block give error
        finally
        {
            System.out.println("this is finally block");
        }

         */
        /*
        //we can use multiple catch block child-->parent ( this
can be judged by exception hierarchy
        try
        {
            int a=10,b=0,c;
            c=a/b;
        }
        catch(IndexOutOfBoundsException e)
        {
            System.out.println("1");
            System.out.println(e);
        }
        catch(NumberFormatException e)
        {
            System.out.println("2");
            System.out.println(e);
        }
        catch(Exception e)
        {
            System.out.println("3");
            System.out.println(e);
        }

         */
/*
        // we can use nested try catch block and inside catch try
catch or inside finally we can use try catch block
        try
        {
            try
            {
                int a=10,b=0,c;
                c=a/b;
                System.out.println(c);
            }
            catch(Exception e)
            {
                int a=10,b=0,c;
                c=a/b;
            }
        }
```

```
           catch(Exception e)
           {
               //similary for catch and finally block we can use
   nested try catch similiar like to try block
               System.out.println("you can not divide with 0");
           }

    */

       }
   }
```

THROW EXCEPTION

```
class ThrowClass
{
    void method()
    {
        try
        {
            int a=10,b=0,c;
            c= a/b;
        }
        catch (Exception e)
        {
            System.out.println(e);
        }

    }
}



public class handling {

    public static void main(String[] args)
    {
      ThrowClass temp = new ThrowClass();
      // we can either put try catch here or in method itself but
better to put in method itself
      temp.method();

        System.out.println("After the method calling ");
        //throw used for custome exception or user defined exception

        try
        {
            throw new RuntimeException();//custom exception
        }
        catch(Exception e)
        {
            System.out.println(e);
        }


        System.out.println("After runtime exception");
    }
}
```

USER OR CUSTOMIZE DEFINED EXCEPTION

```
package com.raushan.Exception;
import java.io.*;
import java.util.Scanner;

//here i am creating unchecked extension i.e. runtime exception
class YoungerAgeException extends RuntimeException
 {
     YoungerAgeException(String message)
     {
         super(message);
     }
 }





public class handling {

    public static void main(String[] args)
    {
        int age;
        Scanner sc = new Scanner(System.in);
        age= sc.nextInt();

        try
        {
            if(age<19)
            {

                throw new YoungerAgeException("You are not eligible
for voting ");
                //Here throw keyword are used for creating object
            }
            else
            {
                System.out.println("Yeah you are eligible for voting
");
            }
        }
        catch(YoungerAgeException e)
        {
            e.printStackTrace();
        }
        System.out.println("This will run after exception ");

    }
}
```

THROWS KEYWORD

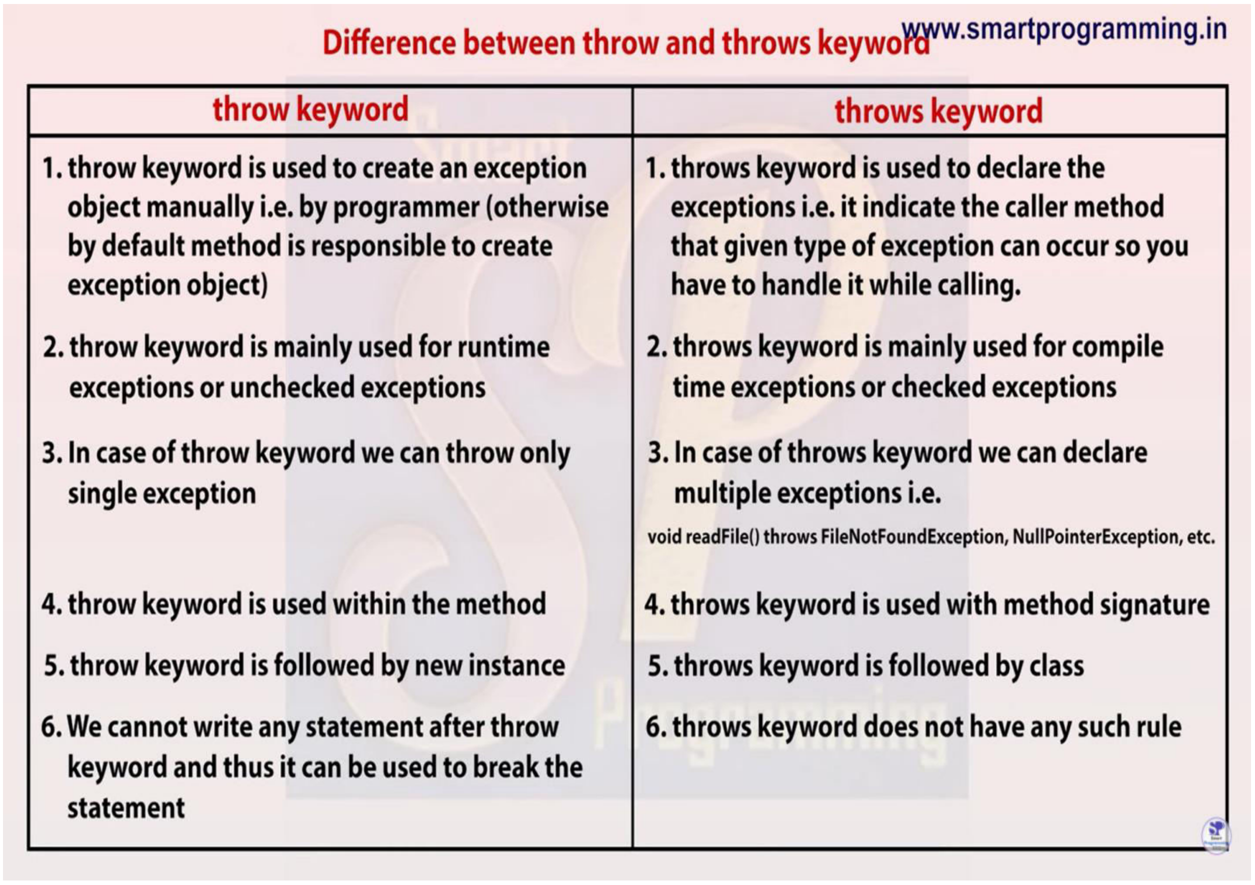
➔Throws keyword is used to declare an exception. It gives information to the caller method that there may occur an exception so it is better for the caller method to provide the exception handling code so that normal flow can be maintained.
→Throw keyword call inside the method
→Throws keyword method pe call hota hai
→Throw keyword always used with checked exception

Implementation of Throws keyword

```
package com.raushan.Exception;
import java.io.*;
import java.util.Scanner;

class First
{
    void method()  throws FileNotFoundException
    {
        //checked exception
```

```
            FileInputStream fis= new FileInputStream("d:/abc.txt");
    }

    void method1() throws FileNotFoundException
    {
        FileInputStream fis= new FileInputStream("d:/abcd.txt");
    }
}


public class handling {

    public static void main(String[] args)  {
       First first= new First();

        try {
            first.method();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

        System.out.println("after method calling");
    }
}
```

DIFFERENCE BETWEEN THROW AND THROWS



Difference between throw and throws keyword www.smartprogramming.in

throw keyword	throws keyword
1. throw keyword is used to create an exception object manually i.e. by programmer (otherwise by default method is responsible to create exception object)	1. throws keyword is used to declare the exceptions i.e. it indicate the caller method that given type of exception can occur so you have to handle it while calling.
2. throw keyword is mainly used for runtime exceptions or unchecked exceptions	2. throws keyword is mainly used for compile time exceptions or checked exceptions
3. In case of throw keyword we can throw only single exception	3. In case of throws keyword we can declare multiple exceptions i.e. void readFile() throws FileNotFoundException, NullPointerException, etc.
4. throw keyword is used within the method	4. throws keyword is used with method signature
5. throw keyword is followed by new instance	5. throws keyword is followed by class
6. We cannot write any statement after throw keyword and thus it can be used to break the statement	6. throws keyword does not have any such rule