Inheritance → Inheritance can be defined as the procedure or mechanism of acquiring all the properties & behaviour of one class to another. i.e. acquiring the properties and behaviour of child class from the parent class.

→ when an object acquires all the properties & behaviour of another object it's known as inheritance

→ It represent IS-A relationship, also known as parent-child relationship.

→ You can inherit parent class data with help of 'extends' keyword.

Parent class / super class / base class

child class / sub class / Derived class

Uses of Inheritance

① Code Reusability:

② Method overriding (run-time polymorphism can be achieved)

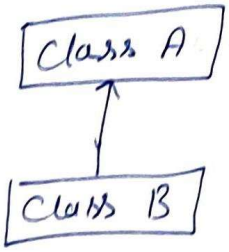
Types of Inheritance

There are 4 types of inheritance in java.

1. single inheritance
2. Multilevel inheritance
3. Hierarchical inheritance
4. Hybrid inheritance.

(1) Single Inheritance

Syntax:




```
Public class A
  {
  ------
  }
Public class B extend A
{
}
```

→ In single inheritance a class can inherit from only one superclass.

or

- A subclass extended only one parent class; gaining access to its field & Methods.

Through Method

Example:-

```
public class A
  {public void disPA()
  {System.out.println("disPA() is executed");
  }}
public class B extends A
  { public void disPB()
  {
  System.out.println("disPA() is executed");
  }
```

```
public static void main(String[] args)
{
    B b = new B();
    b.dispA();   // calling method
    b.dispB();   // calling second method
}}
```

(B → class name; b → object name)

output: dispA() is executed
dispB() is executed.

Through constructor :- constructor executed automatically no need to call method.

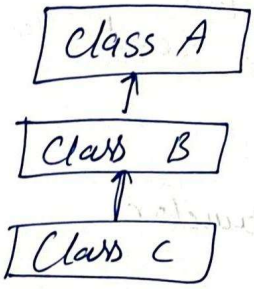
```
class Demo
{
    Demo() {          // constructor
        System.out.println("Demo is created");
    }}

class Demo1 extends Demo
{
    Demo1()
    {
        super();      // call the parent class constructor using super()
        System.out.println("Demo1 is created");
    }}

class Test
{
    public static void main(String[] args)
    {
```

```
Demo1 d = new Demo1();
}}
```

Output: Demo is created
Demo1 is created.

(2) Multilevel Inheritance :

Syntax:




```
class A {
    ----
}
class B extends A
  { ----
  }
class C extends B
  { ----
  }
```

- In multilevel Inheritance, one class can inherit from a derived class. Hence, the derived class becomes the base class ~~from~~ for the ~~it~~ new class.

Through Method

```
public class classA
{ public void displA()
{ System.out.println (" class A is executed");
}}
public class classB extends classA
  {public void displB()
   {System.out.print (" class B is executed");
   }
  }
public class classC extends classB
{ public void displC()
 {System.out.println (" class C is executed");
 }
 public static void main (String[] args)
 {
    classC  c = new classC ();
     c.displA();
     c.displB();
     c.displC();
  }}
```

Output: Class A is executed
Class B is executed
Class C is executed.

~~Through~~ ~~constructor~~

Q write a Program to create a Simple Calculator using multilevel inheritance

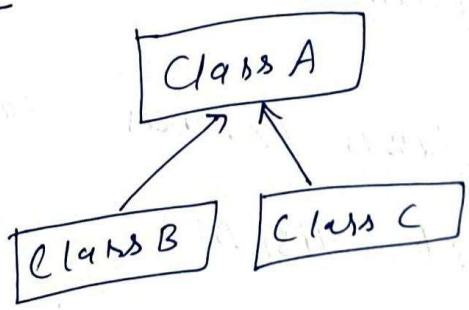
```
Class Calculator
{
    Void add (int a, int b)
    {
      System.out.println("Addition" + (a+b));
    }}

Class Sub extends Calculator
  { Void substraction (int a , int b)
  {System.out.println("substraction" + (a-b));
  }}

Class Mul extends Sub
  { Void multiply (int a, int b)
  { System.out.println("Multiplication" + (a*b));
  }}

Class div extends Mul
  { Void divide (int a , int b)
  { System.out.println("Divide is" + (a/b));
  }
```

```
public static void main (String args[])
{
    div d = new div();
    d.add (6, 7);
    d.sub (7, 3);
    d.Mul (7, 3);
    d.div (10, 2);
}}
```

(3) Hierarchical inheritance

It is type of inheritance in which same class is inherited by more than one class.

Syntax




```
Class A {
---
}
class B extend A
  {
  ----
  }
Class C extends A
  {---
  }.
```

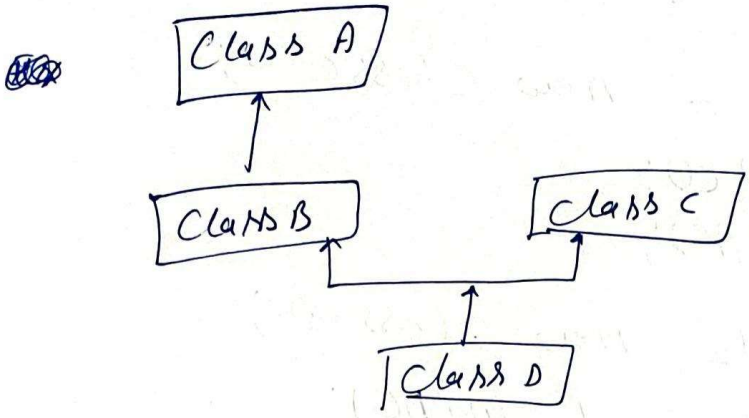
```
public class ClassA
{ public void dispA()
  { System.out.println("class A is executed");
  }}
public class classB extends ClassA
  { public void dispB()
    { System.out.println("class B is executed");
    }}
public class ClassC extends classA
  { public void dispC()
    { System.out.println("class C is executed");
    }}
public class classD extends ClassA
    { public void dispD()
      { System.out.println("D is executed");
      }}
public Class Heirearchical
    {
    public static void main(String[] args)
      {
```

```
ClassB b = new classB()
    b.dispB();
    b.dispA();

ClassC c = new classC();
    c.dispC();
    c.dispA();

ClassD d = new classD();
    d.dispD();
    d.dispA();

}}
```

output: class B is executed
class A is executed
class C is executed
class A is executed
class D is executed
class A is executed.

④ Hybrid Inheritance: It is a combination of two or more types of inheritance (such as single, multiple, multilevel, etc)
- But Java does not support multiple Inheritance with classes due to diamond problem.

- But Java provides a way to achieve Multiple inheritance using Interface.



Syntax:

```
Class A
{ ----
};
Class B extend A
  { ----
  }
Class C
  {----
  }
class D  extends B., C
    {
     ---
    }
```

```
Interface Animal
{
   void eat();  // abstract Method
}
Interface Mammal
   {
      void walk();
   }

Class Dog implements Animal, Mammal
      {
         public void eat()
         {
         System.out.println("Dog eats");
         }
         public void walk()
           {
           System.out.println("Dog walk");
           } }

Class Dog1 extends Dog
   {
      public void sound()
        {
        System.out.println("Dog barks");
        }}

   Class Test
        {
```

```
public static void main (String [] args)
{
    Dog1 d = new Dog1();
    d.eat(); //from interface
    d.walk(); //  "  "  "  "
    d.sound(); //from Dog1 class
}}
```

Output: Dog eat
Dog walk
Dog bark.

Question 1:- use of super keyword inheritance.

Ans: Super keyword use to Access the Parent class member.

→ It will call parent class constructor

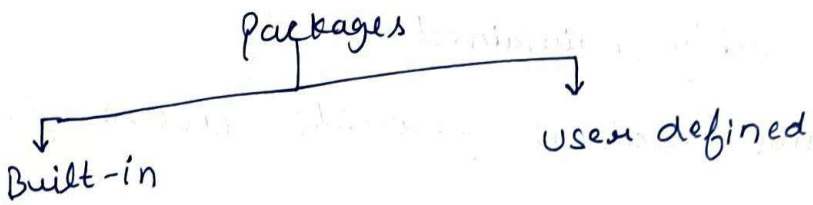
Question 2:- Role of final keyword in inheritance.

→ when you declare a final keyword class so it will not inherit

→ when you declare a final keyword Method so it will not ~~inherit~~ override.

Packages

→ Packages in Java is a mechanism to encapsulate a group of classes, sub packages & Interface



→ A Java package is a group of similar types of classes, interface & sub-package.

→ The package keyword is used to create package.

Built-in Packages :- In Java, we already have various pre-defined packages and these packages contain large number of classes and interface that we used in java are known as Built-in Package.

Example: java.sql, java.lang, java.util java.net, java.io, java.awt

User-defined Package :- As the name suggest user-defined packages are a package that is defined by the user or programmer.

Syntax: package Package_name;

Advantage of Package

1) Java package is used to categorize the classes and interface so that they can be easily maintained.

2) Java package provide access protection.

3) Java package removes naming collision.

Create user defined package

(1) Create a package in java class. The format is very simple and easy.

```
package example1;
```

(2) Include class in java package. But remember that class only has one package declaration

```
package example1;

class Test
    {
    public static void main (String[] args)
{
    // code.
}}
```

(3) Now the user defined package is successfully created we can import it on to other packages and use it.

Example 1. In this example we will create a user-defined package and function to print a message for the user.

```
package example;
Public Class Test
        {
        public void show()
        { System.out.println("Import successful");
        }
```

Example 2:. In this example, we will import the user defined package "example".

```
import example.Test;
public class Test1
   {
    public static void main(String[] args)
    {
```

```
    Test t1 = new Test();
      t1.show();
  }}
```

Output: Import successfull.

Built-in- Package:

- java.sql: Provide the classes for accessing and processing data stored in database.
- java.lang → contain classes & interface that are fundamental to the design of the java Programming language. eg classes
- java.util → Contains the collection framework, some internationalization support classes, properties, random number generation classes eg → LinkedList, HashMap
- java.net → Provide classes for implementing networking applications
  eg → HTTP cookie, Socket, URL
- java.awt → contains classes for creating user interface and for painting graphics and image. Classes like Button, Color, Events.

A Java package is a group of similar types of classes, interface and sub-packages

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

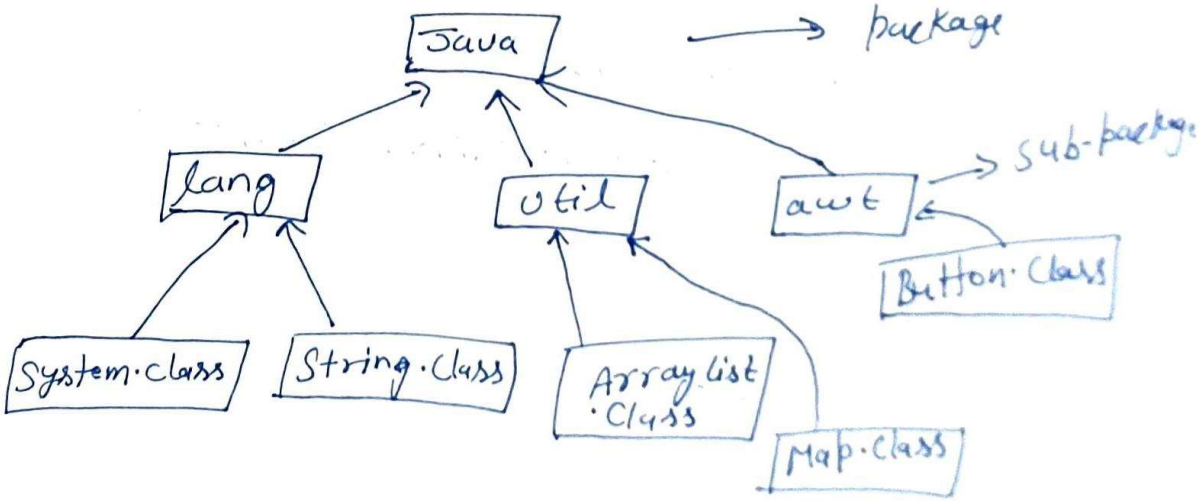
There are many built in packages such as java, lang, awt, javax, swing, net, io, util, sql etc

Advantages of Java package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provide access protection.

3) Java package removes naming collision



Access a package

There are three way to access the package from outside the package

1. import package.*;
2. import package.classname;
3. fully qualified name;

1) Using packagename.*;

→ If you see package.* then all the classes and interface of this package will be accessible but not subpackage.

→ import keyword is used.

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();
      }
  }
```

output: Hello

2) using packagename.classname

If you import package.classname then only declared class of this package will be accessible

```
// save by A.java
package pack;
public class A
  {
    public void msg()
      { System.out.println("Hello");
      }
  }

// save by B.java
package mypack;
import pack.A
  Class A
    {
      public static void main(String[] args)
```

```
{
    A obj = new A();
    obj.msg();
  }
}
```

output: Hello

3) Using Fully qualified name

If you use fully qualified name then only declare class of this package will be accessible. There is no need to import. But you need to use fully qualified name every time when you can accessing the class or interface.

Example

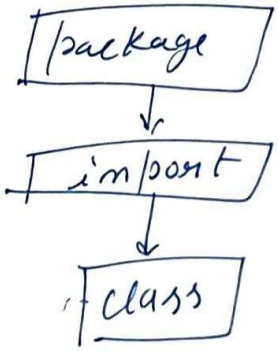
```
//save by A.java
package pack;
public class A
{
  public void msg()
  {
    System.out.println("Hello");
  }
}
```

// save by B.java

```
package mypack;
class B
  {
  public static void main (String [] args)
    {
     pack.A   obj = new pack.A ();
        obj.msg();
    }
}
```

output: Hello

Note: Sequence of the program must be package the import then class



Exception

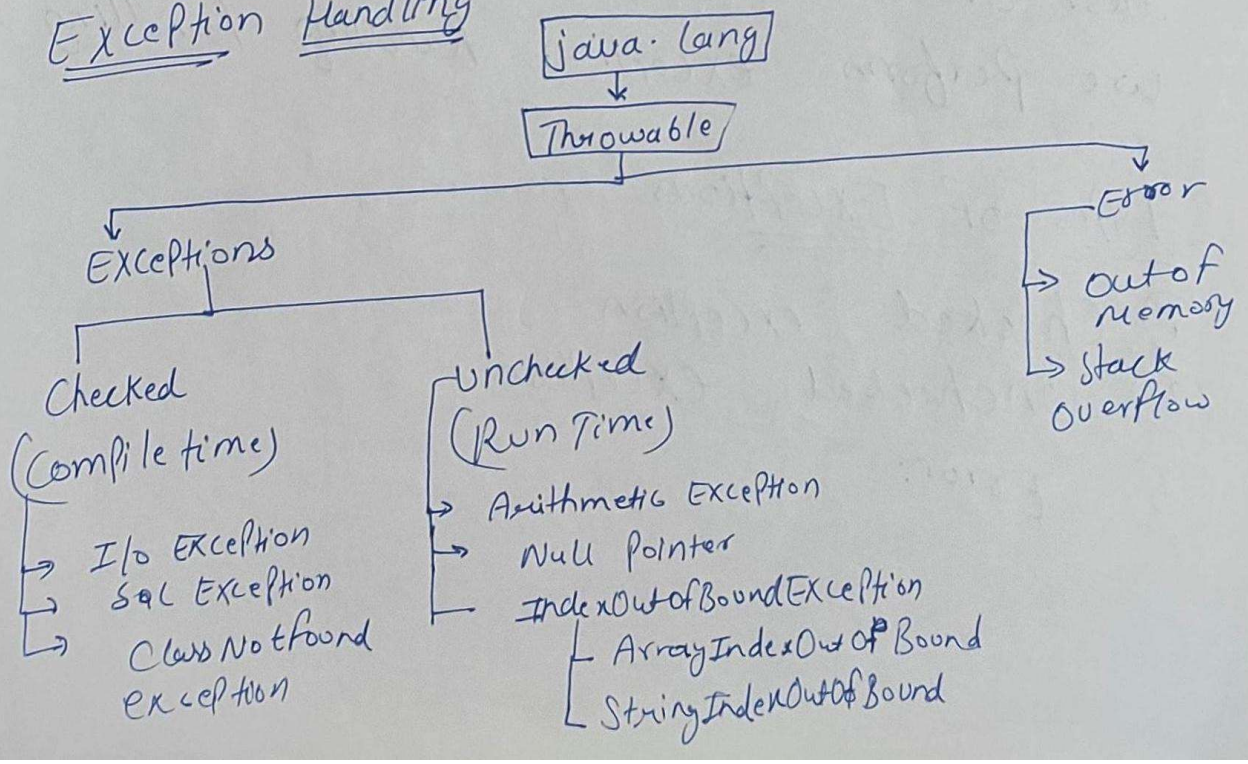
Exception is an error event that can be happen during the executing of program & distrub its normal flow.

example

```
class Test
{
public static void main(String[] args)
{
System.out.println(10/0);
System.out.println("Hello world");
}}
```

Output: Exception in thread "main" java.lang.
Arithmetic exception : - - -

Exception Handling



Exception

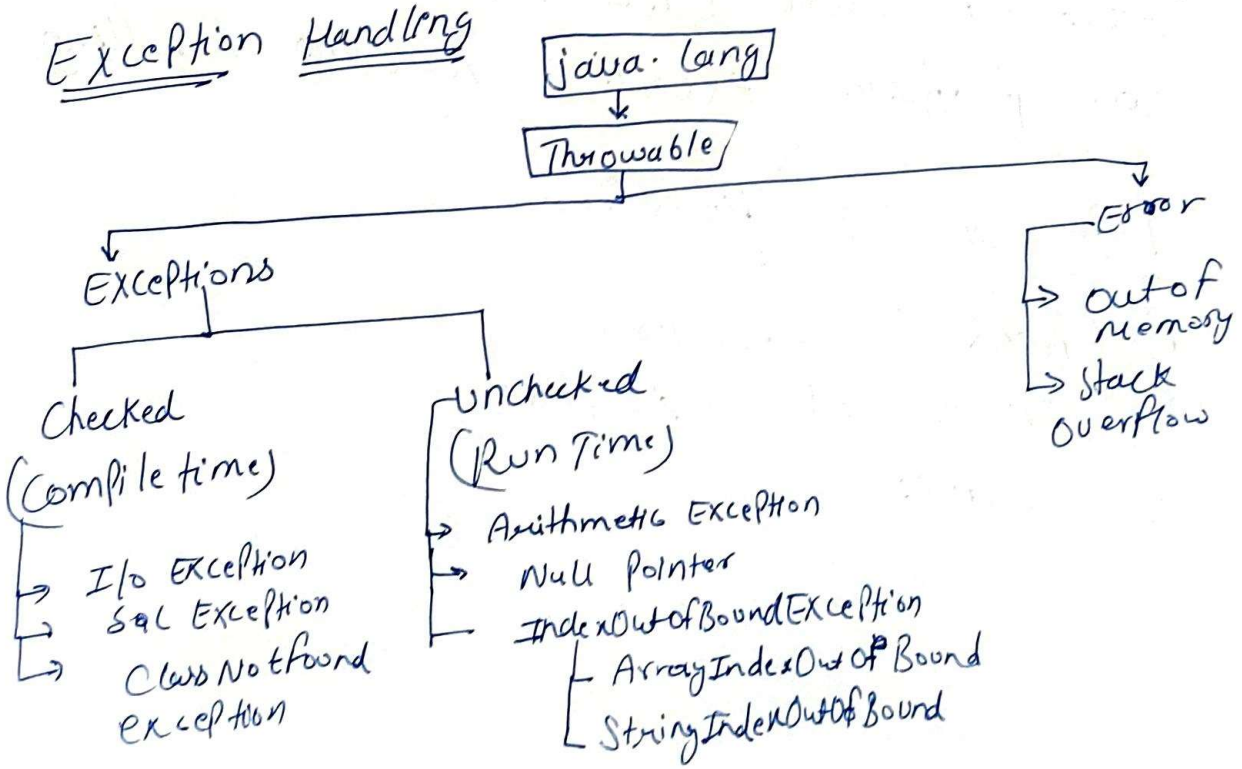
Exception is an error event that can be happen during the executing of program & distrub it's normal flow.

example

```
class Test
{
public static void main(string[] args)
{
System.out.println(10/0);
System.out.println("Hello world");
}}
```

Output: Exception in thread "main" java.lang.
Arithmetic exception: - - -

Exception Handling



Benefits of Exception handling

The main advantage of exception handling is to maintain the normal flow of the program or Application. Exception normally disturb the normal flow of program that is why we use exception handling

```
Statement 1;
Statement 2;  // exception occur.
Statement 3;
Statement 4;
```

Suppose there is 4 statements in your program & exception occur in statement 2 rest of code will not executed. If we perform exception handling handling

Types of Exceptions

(1) Checked exception
(2) Unchecked exception
(3) Error:

Checked Exception :- A checked exception is an exception that occurs at a compile time these are also called compile time exception.
Eg→ IO Exceptions, SQL Exception etc

UnChecked Exceptions. An unchecked Exception is an exception that occurs the time of execution. These are also called Runtime Exceptions. These include programming bugs, such as logic errors, improper use of API

Error : Indicate Serious problems & abnormal condition that most applications should not try to handle.

or

Error is irrecoverable eg→ out of Memory, JVM Error etc

There are 5 keywords for Java Exception handling

1. Try
2. catch
3. finally
4. throw
5. throws

1. Try :- The try statement allows you to define a block of code to be tested for error while it is being executed.

```
try {
    // error occured
}
```

2. catch :- The catch statement allows you to define a block of code to be executed if an error occured in try block.

```
catch ( ExceptionType e )
{

}
```

Note: The try and catch keywords come in pairs:

```
try {
    // Block of code
    }
catch (ExceptionType e)
{   // Block of code to handle error
}
```

example

```
Class Test
{
Public static void main (String[] args)
{ System.out.println ("Hello world");

try
  { System.out.println (10/0);
  }
Catch (ArithmeticException e)
{ System.out.println (e);
}
System.out.println ("Rest of code");
}}
```

output:- Hello world
Arithmetic exception occur
Rest of code

Try with multiple catch blocks

```
Class Test
 {
Public Static void main (String[] args)
{
try
 {
 System.out.println (10/0);
 System.out.println ("Ajay".charAt(80));
 }
catch (ArithmeticException e)
 {
 System.out.println(e);
 }
catch (StringIndexOutofBoondsException ee)
 { System.out.println(ee);
 }
}}
```

output: error : Division by zero is not allowed

if this statement is comment

// System.out.println (10/0);

Then output is: Error: String Index out of bounds:

Note:

Pipe :- If a catch block handle multiple exceptions, you can seprate them using Pipe (|)

eg

```
Class Test
{
public static void main (String [] args)
{
try
{
System.out.println (10/2);
System.out.println ("Ajay".charAt(2));
}
catch (ArithmeticException | StringIndexOutOfBoundException
                                                    e)

{ System.out.println (e);

}
}}
```

Output 5
A

⇒ There are 3 methods/ways to print Exception message

1) toString()
2) getMessage()
3) printStackTrace()

① toString() :- It print the name & description of exception. This method is overridden in Throwable class.

```
Catch(Exception e)
{
System.out.println(e.toString());
}
```

② getMessage() :- getMessage() Method is used to return the detail description of exception.
- It is non static Method
- It show only Message of exceptions.

③ printStackTrace() :- print the name of the exception description & complete stack trace including the line where exception occured.

```
Catch( Exception e)
{ e.printStackTrace();
}
```

eg→
```
public class Test
{
public static void main (String [] args)
{
  try
  {
    int a= 20/0;
  }
catch (Exception e)
  {
    e.printStackTrace();
    System.out.println(e); // Print what exception has been thrown
  }}
```

Output:- java.lang.ArithmeticException: / by zero at Test.main (Test.java:9)

output of getMessage():
/ by zero

Out Put of toString():
java.lang.ArithmeticException: / by zero.

(3) Finally block:- It is a block to execute important code such as closing a connection

→ It always executed whether exception occur or not.

→ finally is not mandatory, but if you use it, will executed always.

Syntax:

```
try {
    // code
}
catch (ExceptionType e)
{   // code to handle the exception
}
finally {
        // code always executed whether exception occurs or not.
    }
```

Example: 1

Case I

```
public class Example {
public static void main (String[] args)
{
try
{   // Risk code that may throw an exception.
  int data = 10/0;
System.out.println (data);
}
```

```
catch (ArithmeticException e)
  {
   System.out.println(e.getMessage());
  }
 finally
  {
   System.out.println("finally block executed");
  }
 }}
```

output: caught an exception: / by zero
finally block executed.

Case 2 when finally block occur Exceptions

```
class Test
{
public static void main(String[] args)
{
 try
  {
   System.out.println(10/0);
  }
 catch (ArithmeticException e)
  {
   e.printStackTrace();
  }
 finally {
   System.out.println("50/0");
  }
 }}
```

Output: java.lang.ArithmeticException: / by zero
at Test.main(filename)

Exception in thread "main" java.lang.
ArithmeticException: / by zero at Test.
main(filename)

Case 3 when try & finally block is executed both. but try occur exception. So finally block first & run after try block will executed

Example

```
class Test
{public static void main(String[] args)
{
  try
  { System.out.println(6/0);
  }
  finally
        { System.out.println("Hello");
        }
  }}
```

output: → Hello
→ exception in thread "main" java.lang.
Arithmetic exception: / by zero

(4) Throw :- The throw keyword in java is used to explicitly throw an exception from a method or any block of code, we can throw either Checked & unchecked exceptions

- Throw keyword is mainly used to throw custom exception.

Syntax

```
throw new ExceptionType("Error Message");
```

Example

```
class Test
{
public static void main(String[] args)
{
    int age = 13;
    if (age < 18)
    {
    throw new ArithmeticException("Invalid age");
    }
else {
    System.out.println("valid Age to vote");
    }
}}
```

Output : Invalid Age

⑤ Throws :- Throws keyword is used to specify the type of exceptions that can be thrown by a method.

- It gives an indication to the developer about the exception that may occur. It is better for developer to provide the exception handling code. So that normal flow of program is maintained.

→ It comes compile time.

Syntax: return-type method_name() throws exception_class_name
{ // Method code }

```
import java.i.FileInputStream;
import java.i.FileNotFoundException;
import java.i.o.FileOutputStream;
class ReadWrite
{
    void readFile() throws FileNotFoundException
    {
        FileInputStream fis = new FileInputStream("d:/abc.txt");
    }
    void saveFile() throws FileNotFoundException
    {
        String text = "This is throws";
        FileOutputStream fos = new FileOutputStream("d:/xyz.txt");
    }
}
class Test
{
    public static void main(String[] args)
    {
        ReadWrite rw = new ReadWrite();
```

```
try
{
   rw.readFile();
}
catch (FileNotFoundException e)
{
   e.printStackTrace();
} System.out.println("Hello");
}}
```

output: java.io.FileNotFoundException
Hello

Note1: In that case we use try catch block execute a program successfully. ~~after~~ or a normal terminate.

Case2 If we use throws keyword in main method program will not executed properly. or abnormal terminate

```
Class Test
  {
  public static void main(String[] args) throws
   FileNotFoundException
  {  ReadWrite rw = new ReadWrite();
       rw.readfile();
       System.out.println("Hello");
  }}
```

output: java.io FileNotFoundException

Difference between throw & throws Keyword

throw	throws
① Java throw keyword is used throw an exception explicitly in the code, inside the function on the block of code	① throws keyword is used in the method signature to declare an exception which might be thrown by the function while execution of code.
② Type of exception using throw keyword, we can only propagate unchecked exceptions.	② using throws keyword we can declare both checked and unchecked exceptions. However it can be used to propagated checked exceptions only.
③ The throw keyword is followed by an instances of exception to be thrown.	③ The throws keyword is followed by class name of exceptions to be thrown.
④ throw is used with in the method	④ throws keyword is used with the method signature
⑤ ~~throw~~ we can not throw multiple exception	⑤ we can declare multiple exception using throws keyword that can throws by the method.

Common Scenarios where Exception occur

Some scenarios where unchecked exception can occur.

1) ArithmeticException occur

If we divide number by zero

```
int = 50/0;
```

2) NullPointerException occur

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

```
String s= null;
System.out.println(s.length);
```

3) ArrayIndexOutOfBoundException occur

If we are inserting any value in the wrong index

```
int a[] =new int[5];
a[10]= 50;
```

User Defined Exception

- we can create our exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user defined exception.

Basically, java custom exception are used to customize the exception according to user need.

user defind exception needs to inherit (extends) Exception class.

```
// represent custom exception class
class InvalidAgeException extends Exception
{
  public InvalidAgeException (String str)
  {
      super(str); // call parent class constructor
  }}

public class TestCustom // Create class to use
                        custom exception is
                        use.
{
  static void validate (int age) throws InvalidAgeException
    {
      if (age<18)
      {   // user defined object
      throw new InvalidAgeException ("age is not valid");
    }
    else
      { System.out.println("welcome to vote");
    }}
```

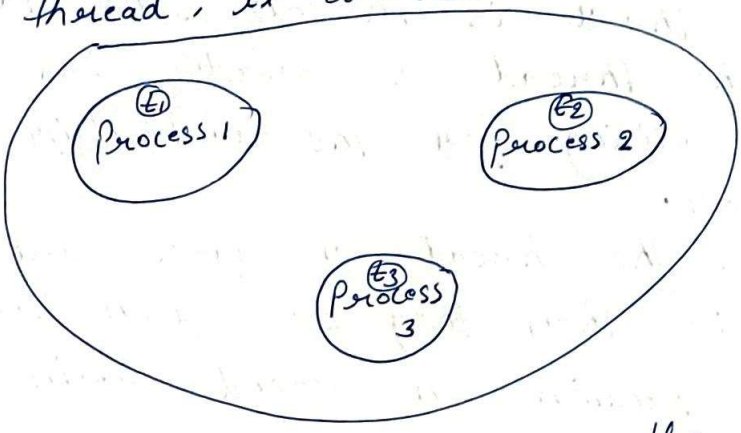
```
public static void main (String[] args)
{
  try
  {
                    //call a method
      validate(13);
  }
catch (InvalidAgeException  e)
  {
     System.out.println ("caught the exception");
     System.out.println (e);
  }
  System.out.println ("rest of code");
}}
```

output

caught the exception
Exception occured: InvalidAgeException : age is not vote
rest of code

Multithread Programming

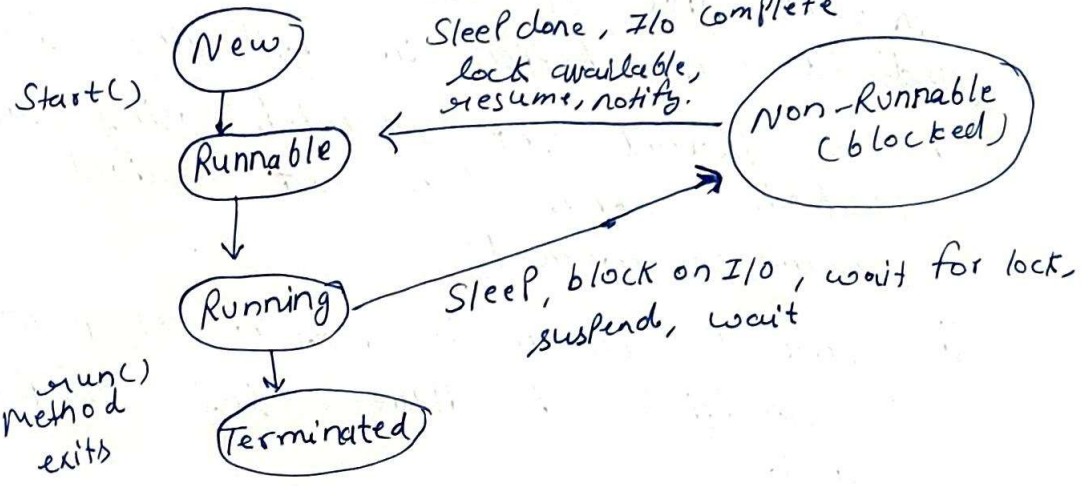
Thread :→ A thread is a lightweight subprocess, the smallest unit of processing. It is separate path of execution.
Thread are Independent. If there occur exception in one thread, it does not affect other thread



→ A thread is executed inside the process

Multithreading :- Multithreading in java is a process of executing multiple thread simultaneously.

Thread Life cycle :-



(1) New born State or New

when a thread object is created a new thread is born & said to be new born state.

(2) Runnable State:

If a thread is in this state it means that the thread is ready for execution and waiting for the availability of the processor.

If all the thread in queue are of same priority then they are given time slot for execution in round robin fashion.

(3) Running state:

It means that the processor has given its time to the thread for execution.

A thread keeps running until following conditions occurs.

(a) Thread give up its control on its own and it can happen in following situation

(i) A thread get suspended using suspend() Method which can ~~not~~ revived with resume() method.

(ii) Thread is made ~~for~~ to sleep for a specified of time using sleep(time) method

(iii) A thread is made to wait for some event to occur using wait() method. And is scheduled to run using notify() method.

(b) A thread is pre-empted by higher priority thread.

(4) Blocked State:- If a thread is prevented from entering in to runnable state & subsequently running state, then a thread is said to be in Blocked state.

(5) Dead State: A runnable thread enters the Dead or terminated state when it completes its task.

How to create a Thread in java programming:

There are two ways to create thread in java

(1) By extending Thread class

(2) By implementing Runnable Interface.

Create a thread by Implementing Runnable Interface:-

Step 1 → We need to implement a run() method provide by Runnable interface. This method provide an entry point for the thread and we put our complete business logic inside this method.

```
Public void run()
```

Step 2 : In second step, we will instantiate a thread object using following constructor

```
Thread (Runnable threadObj, String threadName);
```

→ threadObj is an instances of class that implements Runnable Interface

→ threadName is the given to new Thread.

Step-3 :- Once a thread object is created we, can start it by calling start() method, which executed a call to run() method

```
void start();
```

```
class classA implements Runnable
{
public void run()
 {
  for (int i=1 ; i<=5 ; i++)
    {
     System.out.println(" My child Thread");
    }
 }}

class ClassB
 {
 public static void main (String[] args)
 {
   classA  a = new classA();
  Thread  t = new Thread (a);
    t.start();
  for (int i=1 ; i<=5; i++)
  {
    System.out.println ("main Thread");
  }
 }}
```

output:
```
main thread
main thread
main thread
main thread
main thread
My child Thread
My child Thread
My child Thread
My child Thread.
My child Thread.
```

(2) Create a Thread by Extending a Thread class:

The second way to create a thread is to create a new class that extends Thread class using the following two steps. This approach provides more flexibility in handling multiple thread created using available methods in Thread class

(a) we need to override run() method available in Thread class. This method provides an entry point for the thread and we put our complete business logic inside this method.

```
public void run()
```

(b) once Thread object is created, we can start it by calling start() method, which executed a call to run() method.

```
void start();
```

```
Class A extends Thread
{
 public void run()
 {
  for (int i=1 ; i<=5 ; i++)
   {
    System.out.println ("child thread");
  }}}

Class B
 {
 public static void main (String[] args)
 {
    A r= new A();
    t.start();
  for (int i=1; i<=5; i++)
  {
    System.out.println ("main Thread");
  }
 }}
```

Output:

```
main Thread
main Thread
main Thread
main Thread
main Thread
child Thread
child Thread
child Thread
child Thread
child Thread.
```

Thread Methods

1) public void run() :- It is used to perform an action for a thread.

2) Public void start() :- Starts the execution of the thread. JVM calls the run() method on the thread.

3) public void sleep() :- causes the currently thread to sleep for the specified number of millisecond.

4) Join() :- wait for thread to die.

5) public int getPriority() :- return the Priority of the thread.

Priority of of a Thread

Each thread has a priority. Priorities are represented by a number between 1 to 10

In most cases, the thread Scheduler schedule the thread according to their Priority

But it is not guaranteed because it depends on JVM specification that which Scheduling it choose.

3 Constants Defined in Thread class:

```
Public Static int MIN_PRIORITY
Public Static int NORM_PRIORITY
Public Static int MAX_PRIORITY
```

Default Priority 5 NORM_PRIORITY
Minimum Priority 1 MIN_PRIORITY
MAXimum Priority 10 MAX_PRIORITY

```
class Test extends Thread
{
  public void run()
  {
system.out.println(Thread.currentThread().getName()
                   +"with Priority"+ Thread.currentThread
                   ().getPriority());
  }
  public static void main(String[] args)
  {
      Test t1 = new Test();
      Test t2 = new Test();
      Test t3 = new Test();

      t1.setPriority(Thread.MIN_PRIORITY);
      t2.setPriority(Thread.NORM_PRIORITY);
      t3.setPriority(Thread.MAX_PRIORITY);
```

```
        t1.start();
        t2.start();
        t3.start();
    }
}
```

output:

```
Thread-0 with Priority 1
Thread-1 with Priority 5
Thread-2 with Priority 10
```

Thread synchronization in Java

When we start two or more threads within a program, there may be situation when multiple thread try to access same resource and finally they can produce unforseen result due to concurrency issues.

General Syntax :-

```
Synchronized (object)
{
  // statement to be synchronized
}
```

Why use Synchronization

① To prevent thread interference.

② To prevent consistency problem.

Synchronized Method

If we declare any method as synchronised it is known as synchronized Method.
A Synchronized method is used to lock an object for any shared resource.

```
class Test
{
 synchronized void PrintTab(int n)
 {
   for(int i=1; i<=5; i++)
     { System.out.println(n*i);
   try {
    Thread.sleep(400); }
   catch(Exception e)
    { System.out.println(e); }
    }}}
class Trve extends Thread
 {
   Table t;
   Trve(Table t) {
      this.t=t; }
    public void run() {
       t.PrintTab(5); }}
class Trve1 extends Thread{
    Table t;
    Trve1(Table t) {
       this.t=t; }
    public void run()
    {
       t.PrintTab(100); }}
```

```
public class TestSyn {
 public static void main(String[]
                          args)
{
 Table obj = new Table();
 Trve t1 = new Trve(obj);
 Trve1 t2 = new Trve1(obj);
  t1.start();
  t2.start();
 }}
```

output:

```
5
10
15
20
25
100
200
300
400
500
```

Synchronized Block

Synchronized block can be used to perform synchronization on any specific resource of the method. It is smaller than synchronized method.

eg→ Suppose you have 50 line of code in our method, but we want to synch-ronize only 5 lines, then we can use synchronized block.

→ If we put all the codes of the method in synchronized block, it will work same as synchronized method

Syntax:

```
synchronized (object ref. expression)
{ // code
}
```

Create a two thread in Java

```java
class MyThread extends Thread
{ public void run()
{
  for (int i = 1; i<=5; i++)
  { System.out.println(Thread.currentThread().
                         getName() + "working"+i);

  try
  {   Thread.sleep(1000);
  }
  catch (InterruptedException e)
  {
        e.printStackTrace();
  }
}}}

public class MultiThreadDemo
{ public static void main(String[] args)
{  MyThread t1 = new MyThread();
   My_Thread t2 = new MyThread();

   t1.setName("Thread 1");    // Name of
   t2.setName ("Thread 2");   threads.

   t1.start();
   t2.start();
}}
```

Output:

Thread 1 working : 1
Thread 2 working : 2

Thread 1 working : 1
Thread 2 working : 2

Thread 1 working : 1
Thread 2 working : 2

Thread 1 working : 1
Thread 2 working : 2

Thread 1 working : 1
Thread 2 working : 2

Difference Multithreading VS Multitasking

Multithreading	Multitasking
i> In multithreading many thread are created from a process through which computer power increased	i> In Multitasking users are allowed to perform many task by CPU.
ii> Multi~~tasking~~ threading also CPU switching is often involved between thread	ii> Multitasking involves often CPU switching between the tasks.
iii> Multithread component does not involves multi-processing	iii> Multitasking component involves multiprocessing.
iv> Multithreading process allocate same memory	iv> In Multitasking the ~~component~~ Process share seprate memory.
v) Multithreading termin-ation of threads less time	v) Multitasking termination of a process takes more time.

Example: Executing Two Threads (Even and Odd Numbers)

We will create two threads: one for printing even numbers and another for printing odd numbers from 1 to 5.

Complete Program

Here is the complete code combining both steps:

```
classEvenNumbersimplementsRunnable {
publicvoidrun() {
for (inti=2; i<= 5; i += 2) {
System.out.println("Even: " + i);
try {
Thread.sleep(100); // Sleep for a while to simulate work
        } catch (InterruptedException e) {
System.out.println(e);
        }
```

```
        }
    }
}

classOddNumbersimplementsRunnable {
publicvoidrun() {
for (inti=1; i<= 5; i += 2) {
System.out.println("Odd: " + i);
try {
Thread.sleep(100); // Sleep for a while to simulate work
        } catch (InterruptedException e) {
System.out.println(e);
        }
    }
  }
}

publicclassThreadExample {
publicstaticvoidmain(String[] args) {
ThreadevenThread=newThread(newEvenNumbers());
ThreadoddThread=newThread(newOddNumbers());

evenThread.start(); // Start the even numbers thread
oddThread.start();  // Start the odd numbers thread
  }
}
```

Explanation

1. **Runnable Interface**: The EvenNumbers and OddNumbers classes implement the Runnable interface and override the run() method to define the behavior for each thread.
2. **Thread Creation**: In the main method, two threads are created by instantiating the Thread class with EvenNumbers and OddNumbers as the target.
3. **Starting Threads**: The start() method is called on each thread to begin execution. This triggers the run() method in each thread concurrently.
4. **Sleep**: The Thread.sleep(100) method is used to pause the execution for 100 milliseconds. This simulates some processing time and allows us to see the output more clearly.

Output

The output of the program will look something like this (the order may vary due to the concurrent nature of threads):

```
Odd: 1
Even: 2
Odd: 3
Even: 4
Odd: 5
```