* Exception Handling ⇒

An exception is unexpected /unwanted/ abnormal situation that occured at runtime called as Exception

What is Exception Handling →

In exception handling, we should have an alternate source through which we can handle the exception.

The object Orientation mechanism has provided the following techniques to work with exception →

i) try ii) Catch iii) Throw iv) Throws v) finally

Example →

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

    System.out.println("main method started");

    int a=10, b=0, c; Arithmetic
    c = a/b;          // Exception
    System.out.println(c);

    System.out.println("main method Ends");
}
```

Output →

main method started
java.lang.ArithmeticException: / by zero

To handle this exception with the help of (try, catch) we can write code as follows.

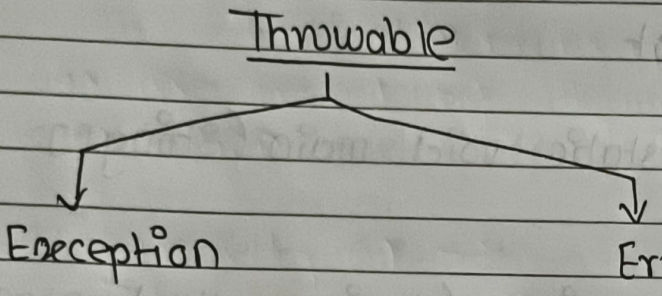
```
Class Test
{
  public static void main(String[] args)
  {

System.out.println("main method started");

int a=10, b=0, c;

try {
      c=a/b;
      System.out.println(c);
    }
catch (ArithmeticException e)
    {
      System.out.println("can't divide by zero");
    }
catch (Exception e)
    {
      System.out.println(e);
    }

System.out.println("main method ended");
}
```

output →

main method started
can't divide by zero
main method ended

Exception Hierarchy ⇒

Throwable class is super class of java exception hierarchy which contains two sub classes that is
i) Exception
ii) Error.



i) RunTimeException
ii) IOException
iii) SQLException
iv) InterruptedException
v) ClassNotFoundException

i) StackOverflowErr
ii) Out of MememoryErr
iii) IO Error
iv) Linkage Error

(1) RunTime Exception →

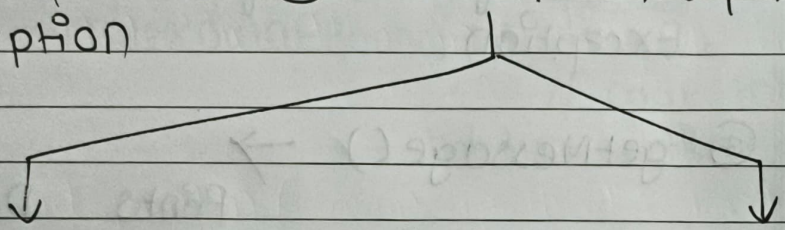
i) Arithmetic Exception
ii) Nullpointer Exception
iii) Number format Exception
iv) Index Out of Bound Exception
- Array Index Out of Bound Exception
- String Index out of Bound Exception

② IO Exception →

i) EOF Exception (End of file)
ii) File Not found Exception

* Types of Exceptions →

① User Defined Exception ② Built in Exception



Checked Exception | Unchecked Excep?

i) ClassNotfound Exception
ii) Interrupted Exception
iii) IO Exception
iv) Instantiation Exception
v) SQL Exception
vi) File Not found Exception

i) Arithmetic Excep?
ii) Class Cast Excep?
iii) Null pointer Excep?
iv) Array Index Out Of Bound Exception
v) Array Store Excep?
vi) Illegal Thread State Exception

① User Defined Exception :→

Build in Exception in Java not able to describe in certain situ? in case user can create own Exception

② Build-in Exception :→

Pre defined Exception classes provided by Java

* Methods to print Exception Info →

① PrintStackTrace() →

Prints the full stack trace of the exception, including the name message and location of the error

② toString() →

Prints exception information in the format of the name of the Exception.

③ getMessage() →

Prints Description of the Exception.

* Try - Catch Block →

A try-catch block in java is a mechanism to handle exception. The try block contains code that might thrown an exception and the catch block is used to handles the exceptions if it occurs.

Syntax →

```
try{
    // code that may throw exception
}
catch (Exception e){
    // code to handle Exception
}
```

We can handle multiple type of exceptions in Java by using multiple catch blocks, each catching a different type of exception.

```
try {
    // code that may throw an exception.
}
catch (ArithmeticException e) {
                    // code to handle
}
catch (ArrayIndexOutOfBound e) {
                    // code to
                handle another excep?
}
catch (NumberFormatException e) {
                    // code to
                handle another excep?
}
```

* Try block → Try is a block that contains code which may have Exception.

* Catch block → It is a block that used to handle the exception.

finally block is always execute Example → if we write normal print statement after the try catch block it will be always execute but suppose if we write code of function and it returns or (terminate the function) then also finally block will be executed in this case normal print statement not execute but finally will

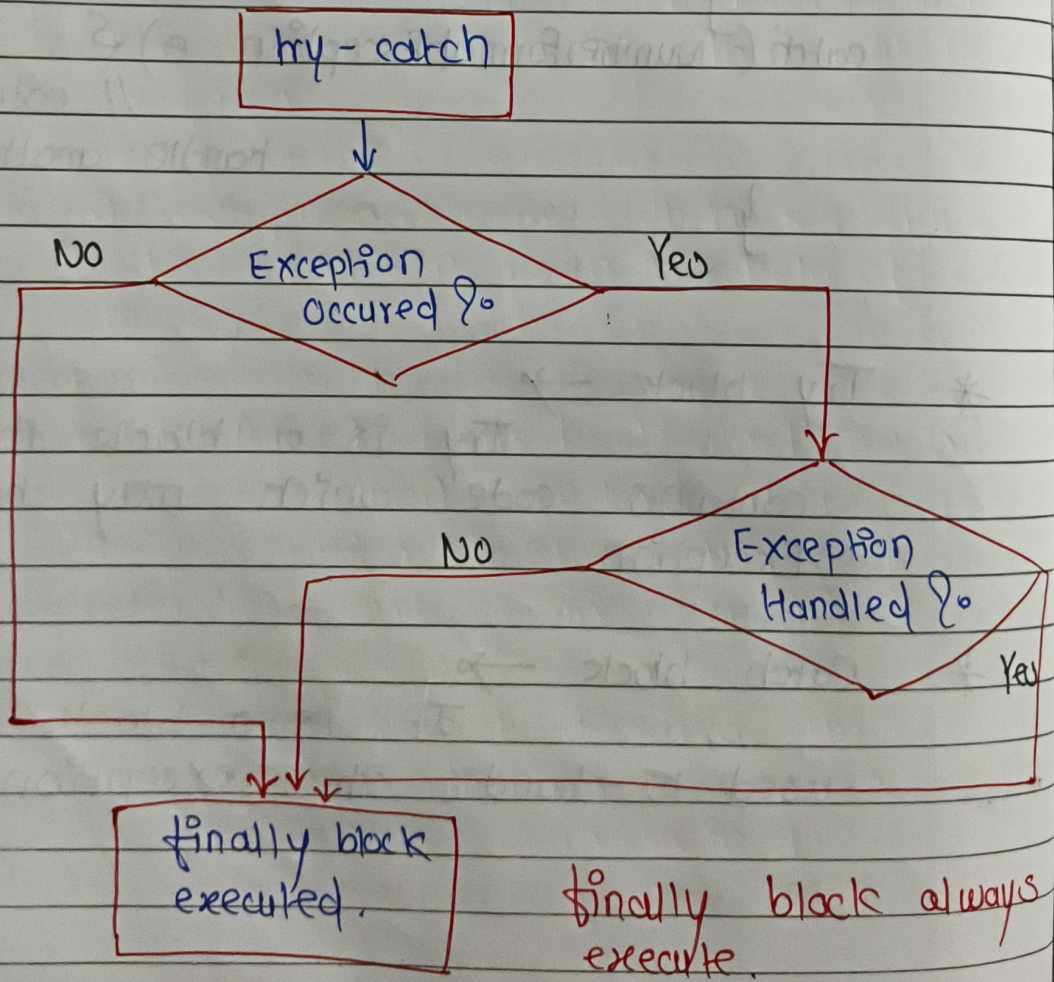
* Finally block →

It is used to execute important code such as closing the connection etc.

Finally block is always executed whether an exception is handled or not by try-catch block.

Finally block always follows try-catch block.

Flow chart →



finally block always execute.

Why finally block used ?

Finally block used to "cleanup" code such as closing a file, closing connection, etc.

Important statements to be printed can be placed in finally block because its always executes.

* Throw keyword →

Throw keyword is used to throw the user defined or Customised Exception object to JVM explicitly

```
public class CustomException extends Exception
{
    public CustomException (String message)
    {
        super(message);
    }
}
```

// custom Exception Class

```
public class CustomExceptionDemo {

    public static void validateAge (int age)
                throws CustomException {

        if(age<18) {

            throw new CustomException("Age is less than 18");
        }
```

```
Else {
        System.out.println("Age is valid." + age);
    }

public static void main(String[] args)
                    throws CustomException
{
    validateAge(16);
}
}
```

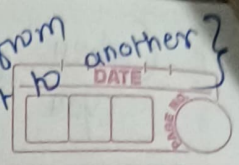
Here in this code when if age is not less than 18 then their will be Exception

if exception occurs then we throw that exception with the help of 'throws'

validateAge() method does not catch that exception, Instead it propagate the exception to the caller of validateAge (i.e Main method)

validateAge
Here ~~main~~ method declares throw CustomException, therefore exception can propagate to the main method.

{ Passing from one part to another }



Since main method also declared throw custom Exception, it propagate the exception further insted of handling it.

Neither validateAge or main catches the exception
Hence, JVM ultimately catched the unhandled exception.

* Summary →

i) validateAge() detects invalid input and throws CustomException.

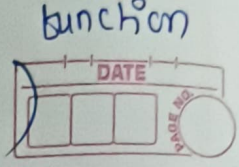
ii) The exception propagates to the main method because validateAge declared throws CustomException.

iii) The main method does not handle it so the JVM catched the exception & terminates the program.

we can also handle this Exception using try-catch

```
public static void main (String[] args)
    try {
        validateAge (16);
    }
    catch (CustomException e) {
        System.out.println("Enter age greater than 18");
    }
}
```

(when your function is a library function you should always use throws



* Throws can declare multiple Exceptions

Example →

```
public class ExceptionExample
{
  // method that may throw exception
  public static void checkExceptions
          (int index, int divisor) throws
    ArrayIndexOutOfBoundsException, ArithmeticException
  {
    int [] number = {10, 20, 30};

    System.out.println("Accessing element at index" + index);
    System.out.println("Element is : " + number[index]);
    // ArrayIndexOutofBound Exception example

    System.out.println("Dividing by" + divisor);

    int result = 10 / divisor;

    System.out.println("Result : " + result);
    // ArithmeticException example.
  }

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

```
try {
    checkException (5,0);
}
catch (ArithmeticArrayIndexOutOfBoundEx e)
{
    System.out.println("Array Index out of Bound
                        caught:" + e);
}
catch (ArithmeticException e) {
    System.out.println("Arithmetic Exception
                        caught" + e);
}
}
}
```

Throw	Throws.
i) Used to explicitly throw an exception	i) Used to declare exceptions a method may throws
ii) Used inside a method or block of code	ii) Used in the method signature.
iii) Can throw only one exception at a time	iii) Can declare multiple exceptions separated by commas (,).

iv) throw is for actually throwing an exception	iv) throws is for informing the caller of the method about possible exceptions.
v) Example →	v) Example →
throw new ArithmeticException("Divide by zero");	void myMethod() throws IOException, ArithmeticException .....

Checked Exception	Unchecked Exception.
i) Checked Exceptions are checked at compile time.	i) Unchecked Exceptions are checked at run time.
ii) Derived from Exception Class	ii) Derived from RuntimeException. Class
iii) External factors like file I/O and database connection cause the checked Exception.	iii) Programming bugs like logical Error cause the unchecked Exception.
iv) Checked exception Examples are IOException, SQLExcep, FileNotFoundExcep	iv) NullPointerException, ArrayIndexOutOfBounds Exception.