* Exception Handling :-

The exception handling is one of the power-ful Mechanism to handle the runtime errors so that normal flow of the application can be maintained.

* Exception :-

In general, Exception is an abnormal condition. In java, exception is an event that disrupts the normal flow of the program. (or)

An exception is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and The program terminates abnormally.

An exception can occur for many different reasons. The following are some scenarios where an exception occurs.

→ A user has entered an invalid data.

→ A file that needs to be opened cannot be found.

→ A network connection has been lost in the middle of communication.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Based on these, The exceptions are categories into three types, Those are

→ checked exceptions

→ unchecked exceptions

→ Errors.

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception.

→ Checked exceptions:

A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of com- -pilaton, the programmer should take care of these exceptions.

Examples:

File Not Found Exception
Class Not Found Exception
NoSuchFieldException
EOFException, etc.

The checked exceptions can be $\overset{\text{(checked)}}{\text{handle}}$ at the time of compilation.

→ Unchecked exceptions:

An unchecked exceptions is an exception that occurs at the time of execution. These are also called as Runtime exceptions. These include programming bugs, such as logic errors or improper use of API. Runtime exceptions are ignored at the time of compilation.

Examples:

AriithmeticException
ArrayIndexoutOfBoundsException
NullPointerException
NegativeArraySizeException, etc.

The unchecked exceptions can be checked at run-time.

→ Errors:

These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Error defines problems that are not excepted by programmer or program.

Examples: Memory Error,
Hardware errör, JVM error etc.

→ An exception is nothing but runtime error. It can be hand-led to provide a suitable message to user.

→ Java uses a mechanism or model to handle exceptions, This mechanism is known as Exception Handling Mechanism.

→ Exception Handling is a mechanism to handle runtime errors such as classNotFound, ArithmeticException, IO, etc.

→ The core advantage of exception handling is to maintain the normal flow of the application. Normally exception interrupts the normal flow of the application That is why we use exception handling. Let's take a scenario:

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;// exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of code will not be executed i.e statement 6 to 10 will not run.

If we perform exception handling, rest of the statements will be executed. That is why we use exception handling in java.

→ There are 5 Keywords used in java exception handling.

- Try
- catch
- finally
- throw
- Throws

* Exception class :-

The exception class can handle any kind of exceptions by using built-in exception classes. There are many built-in classes in Exception class. Some of them are.

• ArithmeticException	: Arithmetic error
• ArrayIndexOutOfBoundsException	: Array index out of bounds
• NullPointerException	: Accessing an object through null pointer
• NumberFormatException	: Given no. is not number
• ClassNotFoundException	: class not found
• IO Exception	: Input/output exception
• FileNotFoundException	: unable to locate a file
• SQLException	: SQL statement is not well.

→ Exception Handling requires the following four steps

1. Finding the problem (Identify the statements whose execution may result in exception. put all those statements in a Try {...} block.)
2. Inform that an exception is thrown (Throw the exception)
3. Receive the exception (Catch the exception using catch{..} block).
4. provide exception handling code in catch block.

* Try and catch block :-

• **Try block :**

It is used to enclose the code that might throw an exception. i.e all statements that are likely to raise an exception at run time are kept in try block. This block will detect an exception and throws it. It will be handled by catch block.

Try block must be followed by either catch or finally block.

Syntax:- Try - catch

```
Try
{
 // code that may throw exception
}
catch (Exception_className ref)
{
 //statements
}
```

• **catch block :**

It is used to handle the exception. It must be used after the try block only. It provides a suitable message to the user, then user will be able to proceed with the appli-cation.

→ A try block can follow multiple catch blocks, i.e we can use multiple catch blocks with a single try.

→ try { } block may have one or multiple statements.

→ try { } block may throw a single type of exception or multiple exceptions. But at a time it can throw only single type of exception.

Syntax:-

```
try
{
 statements that may throw exceptions
}
catch (ExceptionType 1  e1) {...}
catch (ExceptionType 2  e2) {...}
   :
catch (ExceptionType n  en) {...}
```

Example:-

SampleException.java

```
class SampleException
{
public static void main (String args [])
{
 int a=0;
 int b=10;
 try
 {
  int c = b/a;
  System.out.println(" The result is " + c);
 }
 catch (ArithmeticException e)
 {
  System.out.println("Divided by zero");
                    (or)
                 e.toString()
 }
  int sum= a+b;
  System.out.println("sum is" + sum);
 }
}
```

o/p:-

```
javac SampleException.java
java SampleException
Divided by zero   (or) java.lang.
                  ArithmeticException:
sum is 10.          / by zero
```

* Multiple catch blocks :-

A try block can be followed by multiple catch blocks. you can have any number of catch blocks after a single try block.

If an exception occurs in try block then the exception is passed to the first catch block in the list. If the exception type matches with the first catch block it gets caught, if not the exception is passed down to next catch block.

Rule:- At a time only one Exception is occured and at a time only one catch block is executed.

Rule: All catch blocks must be ordered from most specific to most general. i.e. catch for ArithmeticException must come before catch for Exception.

Example:-

MultipleException.java

```
class MultipleException
{
public static void main(String args[])
{
 try
 {
  int arr[] = {1,2};
  arr[4] = 3/0;
 }
 catch (ArithmeticException ae)
 {
  system.out.println(" Divide by Zero");
 }
 catch (ArrayIndexoutofBoundsException e)
 {
  system.out.println(" array index out of bound");
 } }
}
```

o/p:-
```
javac MultipleException.java
java MultipleException
Divide by zero.
```

Example for unreachable catch block

While using multiple catch statements, it is important to remember that all catchs blocks must be ordered from most specific to most general. i.e we have to write general exception class as last catch block.

UREnception.java

```
class UREnception
{
public static void main (String args[])
{
 try
 {
  int a[] = {1,2};
   a[3] = 5/0;
 }
 catch (Exception  e)  // This block handles all exceptions
 {
  System.out.println(" Generic Exception");
  }
 catch (ArithmeticException ae) //This block is unreachable
  {
   System.out.println(" Divided by zero");
  }
                    output:-  javac   UREnception.java
 }
                              compile Time  Error
}
```

The above program ariases compile time error, because you write general exception catch block as first catch block. So we need to write most specific exceptions first and then general exception catch block.

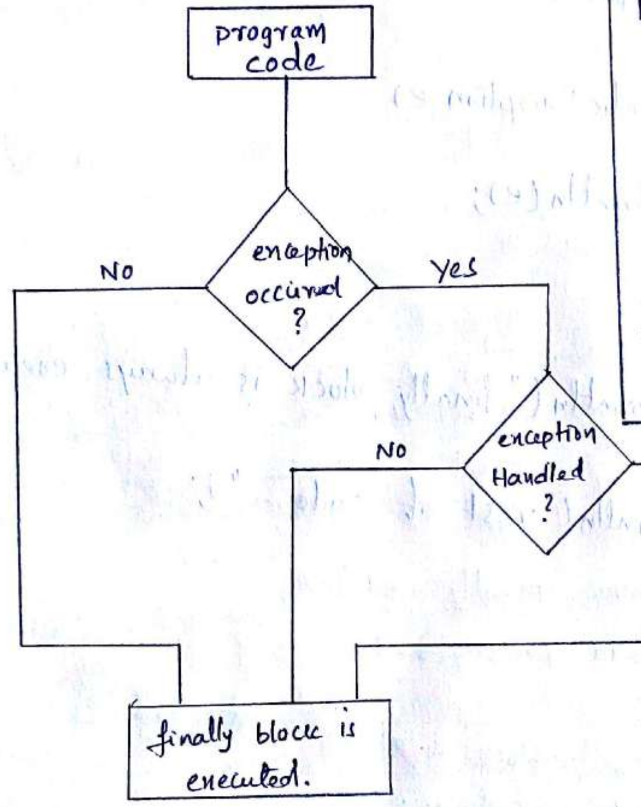
The general Exception class handles all exceptions if no other specific exception catch blocks.

* finally block :-

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

finally block is always executed whether exception is handled or not.

The finally block follows try or catch block.



Syntax:-

```
try
{
 // code
}
catch (exceptiontype e)
{
 //statements
}
finally
 {
  //statements
 }
```

Note:- If you don't handle exception, before terminating the program JVM executes finally block (if any).

→ Finally block in Java can be used to put "cleanup" code such as closing a file, closing connection etc.

→ Let's see the different cases where finally block can be used/executed.

The finally block can be executed when/where exception doesn't occur, exception occurs and not handled, exception occurs and handled.

Rule:- For each try block there can be zero or more catch blocks, but only one finally block.

Example :- Where exception doesn't occur

Finallycase1.java

```
class FinallyCase1
{
public static void main(String args[])
{
  try
  {
   int data = 25/5 ;
   System.out.println(data);
  }
  catch (ArithmeticException e)
  {
   System.out.println(e);
  }
  finally
  {
   System.out.println("finally block is always executed");
  }
  System.out.println("rest of code...");
 }         o/p: javac FinallyCase1.java
}               java FinallyCase1
                5
                finally block is always executed
                rest of code...
```

Example Where exception occurs and not handled

Finallycase2.java

```
class FinallyCase2
{
 public static void main(String args[])
 {
   try
   {
    int data = 25/0 ;
    System.out.println(data);
   }
```

Scanned by CamScanner

```
catch (NullPointerException e)
{
 system.out.println(e);
}
finally
{
 system.out.println("finally block is always executed");
}
system.out.println("rest of the code ... ");
}
}
```

O/P:-
```
javac FinallyCase2.java
java FinallyCase2
finally block is always executed
Exception in thread "Main" java.lang.ArithmeticE
xception : /by zero
```

Example: Where exception occurs and handled.

FinallyCase3.java

```
class FinallyCase3
{
public static void main(String args[])
{
 try
 {
  int data=25/0;
  System.out.println(data);
 }
 catch (ArithmeticException e)
 {
  system.out.println("Divided by zero");
 }
 finally
 {
  system.out.println("finally block is always executed");
 }
 system.out.println("rest of the code...");
}
}
```

O/P:
```
javac FinallyCase3.java
java FinallyCase3
Divided by zero
finally block is always execute
rest of code...
```

* throw Keyword :-

The java throw Keyword is used to explicitly throw an exception. We can throw either checked or unchecked exception in java by throw Keyword.

program execution stops on encountering throw statement.

Syntax:- throw throwble instance

(or)

throw exception

Example:

In this example, We have created the validate method that takes integer value as parameter. If the age is less than 18, We are throwing the AirthmeticException otherwise print a message "Welcome to vote".

ThrowExp.java

```
class ThrowExp
{
 void validate (int age)
 {
  if (age <18)
   throw new AirthmeticException ("Not valid");
  else
   System.out.println ("Welcome to vote");
 }
 public static void main(String args[])
 {
   ThrowExp t = new ThrowExp ();
   t. validate (13);
   System.out.println ("rest of code");
 }
}
```

o/p:
```
javac ThrowExp.java
java ThrowExp
java.lang. AirthmeticException: not valid
```

* **Throws Keyword :-**

The throws Keyword is used to declare an exception. It gives an infdmation to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

The throws Keyword mainly appilied on methods to provide infdmation to caller of the method about the exception.

Syntax:-

```
return_type method_name() throws exception_class_name
{
 //Method code
}
```

Example :-

Let's see the example of java throws keyword which describes that exceptions.

```
public class ThrowsDemo {
void validate (int age) throws ArithmeticException
{
  if (age <18)
    throw new ArithmeticException ("not valid");
  else
    System.out.println(" Welcome to vote ");
}
public static void main (String args[])
{ try
  {
   ThrowsDemo td = new ThrowsDemo();
    td.validate(12);
  }
  catch (ArithmeticException e)
  { system.out.println(e);
  }
  system.out.println("rest of code.");
} }
```

```
o/p: javac ThrowsDemo.java
     java ThrowsDemo
     java.lang.ArithmeticException
                        :not valid.
      rest of code..
```

Scanned by CamScanner