Exception Handling :-

- An **exception** is unexecpected / unwanted / abnormal situation that is occured at runtime is called exception.

  Dictionary Meaning :- Exception is abnormal condition.

- In Java, an exception is an event that disrupts the normal flow of the program.

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

Error vs Exception

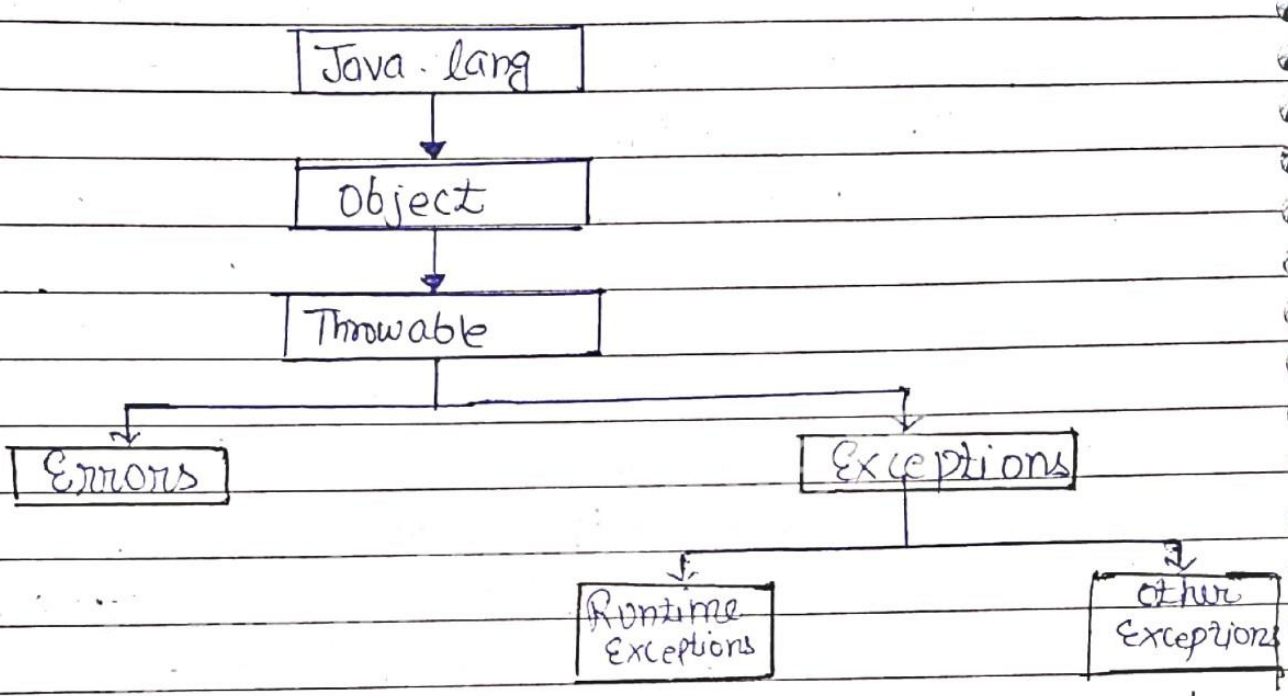
Error :- An Error indicates serious problem that a reasonable application should not try to catch.

Exception :- It indicates conditions that a reasonable application might try to catch.

- Different approaches to handle exceptions :-
  (i) try ... catch block
  (ii) finally block
  (iii) throw and throws keyword

__/__/____

* Java Exception Handling model :-



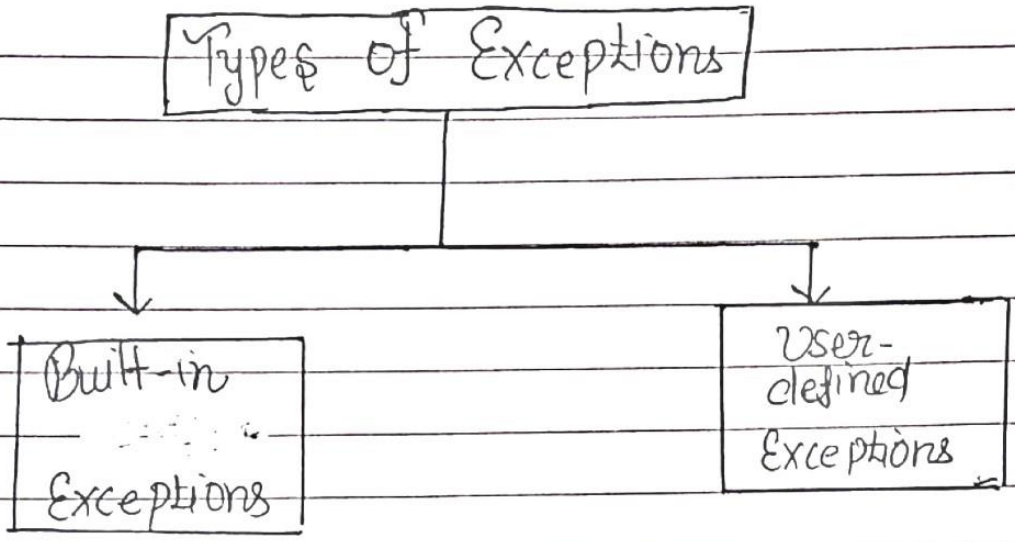
→ In Java, runtime errors are thrown as exceptions.

→ An exception is an object that represents an error or a condition that prevents execution from proceeding normally

→ If the exception is not handled, the Program will terminate abnormally.

→ Exceptions are thrown from a method. The caller of the method can catch and handle the exception.

__/__/____



* Built-in Exceptions :- Are the exceptions which are available in Java libraries.

List of important built-in Exceptions :-

- Arithmetic Exception :- When an exceptional condition has occured in an arithmetic operation.
- ArrayIndexOutOfBoundException :- It is thrown to indicate that an array has been accessed with an illegal index.
- ClassNotFoundException :- When we try to access a class whose definition is not found.

__/__/____

- FileNotFoundException:- When a file is not accessible or does not open.
- IOException:- It is thrown when an input-output operation failed.
- NullPointerException:- When referring to the members of a null object.
- StringIndexOutOfBoundsException:- It is thrown by String class methods to indicate than an index is either negative or greater than the size of string.

Example:-

```
class ArithmeticException
{
  public static void main (String args[])
  {
    try
    {
      int a= 20, b= 0;
      int c= a/b;
      System.out.println("Result= "+c);
    }
    catch (ArithmeticException e)
    {
      System.out.println("Can't divide by 0");
    }
  }
}
```

__/__/____

* User-defined Exception:-

- Java Provides us facility to create our own exceptions which are basically derived classes of Exception.
- Creating our own Exception is known as custom Exception or User defined Exception.
- Basically, Java custom exceptions are used to customize the exception according to user need.

Example:-

```
class InvalidAge xception extends Exception
{
  InvalidAgeException(String msg)
  {
    System.out.println(msg);
  }
}

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

__/__/____

* User-defined Exception:-

- Java Provides us facility to create our own exceptions which are basically derived classes of Exception.
- Creating our own Exception is known as custom Exception or User defined Exception.
- Basically, Java custom exceptions are used to customize the exception according to user need.

Example:-

```
class InvalidAge xception extends Exception
{
  InvalidAgeException(String msg)
  {
    System.out.println(msg);
  }
}

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

__/__/____

```
        {
          vote (12);
        }
        catch (Exception e)
        {
          System.out.println(e);
        }
    }

public static void vote () throws
                                InvalidAgeException.
    {
      if (age < 18)
        {
          throw new InvalidAgeException ("Not
                                eligible for voting");
        }
        else
        {
      System.out.println ("Eligible for voting");
        }
    }
```

* Try and Catch block in JAVA :-

- Java **try ... catch** block in Java is used to handle exceptions and

- The **try** statement allows you to define a block of code to be tested for errors while it is being executed.

- The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.

Syntax :-

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

Catch block is used to handle the Exception by declaring the type of exception with parameter.

__/__/____

Program without try/catch block :-

```
public class Main
{
public static void main (String args[])
{
  int[] myNumbers = {1,2,3};
  System.out.println(myNumbers[10]); //error
}
}
```

Output :- Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException : 10

Program using try...catch to catch an error and execute some code.

```
public class Main
{
public static void main (String args[])
{
  try {
    int[] myNumber = {1,2,3};
    System.out.println(myNumbers[10]);
  } catch (Exception e) {
    System.out.println("Something went wrong");
  }
}
}
```

Output :- Something went Wrong.

* Multiple Try and Catch clauses:-

- A try block can be followed by one or more catch blocks.
- Each catch block must contain a different exception handler.
- So, whenever we needed to handle more than one specific exception, then we had to have different catch block of Java.

Program of multiple try-catch:-

```
class Handler
{
public static void main (String args[])
{
  try
  {
    int a=10, b=0, c;
    c = a/b;
    System.out.println(c);
  }
catch (ArithmeticException e)
  {
    System.out.println("cannot divide by zero");
  }
```

__/__/____

```
try
{
  int a[] = {10, 20, 30, 40};
  System.out.println(a[5]);
}
catch (ArrayIndexOutOfBoundsException b)
{
  System.out.println("beyond the array limit");
}
}
```

Output :-

Cannot divide by zero
beyond the array limit

__/__/____

* Nested try block :-

- In Java, we can use a try block within a try block. It is called nested try block.

Syntax :-

```
//main try block
try
{
 statement 1;
 statement 2;
      // try catch block within another try block
 try
   {
     statement 3;
     statement 4;
      // try catch block within nested try block
   try
     {
      statement 5;
      statement 6;
   }
   catch(Exception e2)
     {
      ---
     }
```

```
    }
    catch (Exception e1)
    {
      ---
    }
}
  // catch block of outer try block
catch (Exception e3)
{
  //---
}
---
```

Program of Nested try block :-

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

  try
  {

  try
  {
      int a[] = {10,20,30};
      System.out.println(a[2]);
  }
```

```
catch (ArrayIndexOutOfBoundsException a)
{
    System.out.println(a);
}
System.out.println(10/0);
}

catch (ArithmeticException e)
{
    System.out.println(e);
}
    System.out.println("jpnotes");
```

Output:-

```
30
java.lang.ArithmeticException: / by zero
jpnotes.
```

__/__/____

* finally block:-

- Java finally block is always executed whether an exception is handled or not.
- The important statements to be printed can be placed in finally block.

Syntax:-

```
try
{
  --
}
catch
{
  // handling exception.
}
finally
{
  // Statements to be executed
}
```

Program to show the concept of finally block:-

```
Class Test
{
```

```
public static void main (String args [])
{
 try
 {
  int a = 25/5;
  System.out.println(a);
 }

 catch (NullPointerException e)
 {
  System.out.println (e);
 }

 finally
 {
 System.out.println (" finally block is always
                      executed");
 }

 System.Out.Println (" jpnotes ");
}
}
```

Output:-

5

finally block is always executed

jpnotes

__/__/____

* Java throw and throws keyword :-

- The Java `throw` keyword is used to explicity throw a single exception.
- When we `throw` an exception, the flow of the program moves from the `try` block to the `catch` block.

Example

```
class Main
  {
  public static void dividebyzero()
    {
      throw new ArithmeticException("Trying to divide by 0");
    }

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