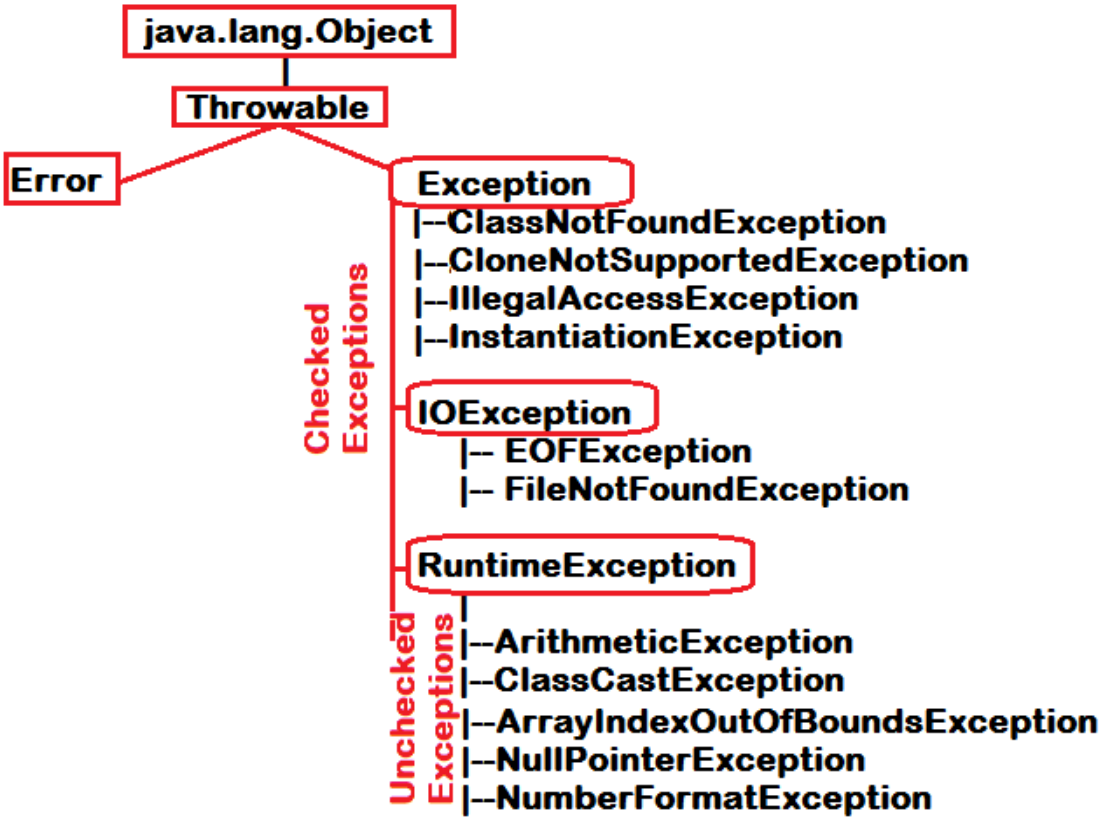
# Exception Handling:

- Mostly Error will occur at the time of compilation just because of regular syntax mistakes
- We can the make the program to compile success by making the necessary correction in the syntax mistakes
- In this case client is not getting any problem just because as it in production
- Exception is a typically an "error" it will occur at the time of execution at any point of time.
- If any exceptions are occur then normal flow of the program will get disturbed and program will be terminated abnormally or unsuccessfully
  - Some possible reasons for the Exceptions are
    - Invalid Number of Inputs
    - Invalid Input types
    - Any logical Error Occur
- Benefits Of Handling Exceptions are :
  - Making the program to terminate Successfully
  - Grouping and Supporting the Error Types
  - Separating Error Logic with Normal Logic
- Default Exception handler is the JVM
  - During Executing the Java Programs if at any where any thing goes wrong , then JVM will recognize the corresponding "ExceptionClassName" Based on the context.
  - It will print Exception Class Name , Description of the Exception
  - StackTrace of the Exception [Name of class | method name | program name and line number ] where the exception is got raised and finally making the program to terminate abnormally or unsuccessfully
- Keywords Related to Handle the Exception:
  - try | catch | throw | throws | finally

## List of possible Exception



## Functionality of Try and Catch Block:

```
try
{
  Risky Code
   or the lines Which are
    Possible for Exception
}

catch(ExceptionClassName  ref)
{
   Exception Handling
    Logic
}
```

- in the try block is always recommended to write the lines which are possible for only exceptions
- catch is the most immediate block to try.
- In the catch block is always recommend to write the logic related to handle the Exception | Solutions of the Exception
- During executing the statements of try block if at all any thing goes wrong , then JVM will recognize the corresponding Exception class name based on the context.
- It will create an object of the corresponding "Exception Class" and it will throw to the Exception handler
- The catch block the exception handler , it will receive the Exception class object . Handle the Exception and finally making the program to terminate successfully.

Note: The argument of catch block should be same as the Object which thrown by the catch block.

Example:

```
import static java.lang.System.*;
class Testing
{
        public static void main(String args[])
        {
        out.println("Hello 1");
        out.println("Hello 2 ");
        out.println(3/0);
        out.println("Hello 3");
        }
}

class ExceptionEx
{
        public static void main(String args[])
        {
                try{
                System.out.println(args[0]);
                System.out.println(args[1]);
                }

                catch(ArrayIndexOutOfBoundsException a)
                { System.out.println("Sorry Dear ");
                  System.out.println("Plz Enter 2 numbers "); }
        }
}
```

Example 2:

```
class ExceptionEx3{
        public static void main(String args[]){
                java.util.Scanner s=
                        new java.util.Scanner(System.in);
                System.out.println("Enter 2 numbers ");
                int x=s.nextInt();
                int y=s.nextInt();
                try{
                        int z=x/y;
                System.out.println("Result is : "+z);
                }
                catch(ArithmeticException a)
                {
        System.out.println("Sorry V R N D By Zero ..");
        System.out.println("Enter 2nd value ");
                 y=s.nextInt();
                 int z=x/y;
                 System.out.println("Result is : "+z);
                }
        }
}
```

Example :

```
class MCDemo{
        public static void main(String args[])
        {
                try{
                int x=Integer.parseInt(args[0]);
                System.out.println(args[0]); }

                /* In JDK1.6
                catch(ArrayIndexOutOfBoundsException a)
```

```
            {System.out.println("Plz Enter 1 value "); }

            catch(NumberFormatException ne)
            {System.out.println("Plz Enter int value "); }  */

           In JDK1.7
            catch(ArrayIndexOutOfBoundsException |
                  NumberFormatException n)
            { System.out.println("Plz Enter 1 int value "); }
      }
}
```

## Handling any Exception By a catch block:

It is possible by passing "Exception" as an argument to the catch block. As per the polymorphism super class reference can hold the object of the same class as well as it can hold the object of any of its subclass.

"Exception" is the super class for every Exception class, thus it can hold the object of any of it's subclasses.

- This process is also called generic exception handling mechanism

Excample:

```
class AnyExceptionDemo{
      public static void main(String args[])
      {
       try{
            int x=Integer.parseInt(args[0]);
            int y=Integer.parseInt(args[1]);
                int z=x/y;
              System.out.println("Result is : "+z);
      }
```

```
        catch(Exception e)
              { System.out.println
                    ("Sorry unable to continue...\n"+e);
             e.printStackTrace();
           }
        }
}
```

## "throw" keyword

- "throw " is keyword
- Exceptions are classified into 2 types
  - Predefined Exceptions [Predefined Exception classes]
  - User defined Exceptions [Use defined Exception Classes]
- All the Exceptions are the classes only
- **Pre-defined exceptions are the classes which are provided by JDK Software. All the predefined Exceptions are well known the JVM thus JVM will take responsibility in [raising the Exception] Based on the context**

- **User defined Exception are the classes which are defined by us based on our application requirements , User defined exceptions are unknown to the JVM thus JVM doesn't cares about invoking the user defined Exception, thus it is our responsibility to raise that exception based the application required | context**

- In order raise the user defined exception then we have use the keyword “throw”.
- If required we can raise either predefined Exception or user defined Exception
- Syn: throw new <ExceptionClsName>([list of args]);

```
//Bank.java
import java.util.Scanner;

class Bank{
        public static void main(String args[ ]){
                Scanner s=new Scanner(System.in);
                System.out.println("Enter Acc Number : ");
                int acno=s.nextInt();

                if(acno>0)
                  { System.out.println("Valid Acc Number "); }
                else
                 {
                    try{
                            throw new ArithmeticException();
                        }
                        catch(ArithmeticException ae)
                    {
                   System.out.println("Sorry Unable to continue...");
                                System.out.println("Reason is : "+ae);
                                System.out.println("Invalid Acc No : ");
                  }
```

```
                }
        }
}
```

```
/* Example on throw */
// throw new <ExceptionClassName>([list of args])

import java.util.Scanner;

class BankDemo{
        public static void main(String args[]){
                Scanner s=new Scanner(System.in);
                System.out.println("Enter acno ");
                int acno=s.nextInt();

                if(acno>0){
                        System.out.println("Valid Acno ");
                }
                else  {
                   try{
                        throw new NumberFormatException(); }

                        catch(NumberFormatException ne)
                        { System.out.println("From Catch ... ");
                          System.out.println(ne);
                          System.out.println("Invalid Acno... ");
                        }
                }
        }
}
```

## User defined Exception:

- The Exception classes which defined by us for our application requirements
- User defined Exception is nothing but defining user defined class
  - Steps for defining user defined Exception
    - Define A class that should be extended by either “Exception” or “RuntimeException”
    - If you want define the unchecked Exception then our Exception class should be the sub class of “RuntimeException”
    - If you want define the checked Exception then our Exception class should be the sub class of “ Exception”
- Step -2:
  - Define a parameterized constructor in order to display the description of user defined Exception
- Step-3:
  - We must call the parameterized constructor of the super class of “User defined Exception class ” using super( ) otherwise JVM Internally calls the default constructor of the super( ) implicitly it will print only

the "ExceptionClassName" but not "Exception Description"

- **Step-4:**
  - **Invoke the user defined Exception based on the context by using keyword "throw"**

**Example: first input is : sssit second input : kphb then I want display valid user .if not I want throw userdefined exception called "MyLoginException"**

**Example:**

**//Userdefined.java**

```
class MyLoginException extends RuntimeException
{
        MyLoginException(String s)
        { super(s); }
}
class Login{
    public static void main(String args[ ]) {
            String username=args[0]; //sai
            String password=args[1]; //uk
```

```
            if( username.equals("sssit")  && password.equals("kphb") )
              { System.out.println("Valid User "); }
            else
              {
                    try{
                     throw new MyLoginException("Invalid UN|PW ");
                    }
                    catch(MyLoginException me)
                     {
                           System.out.println("Sorry Unable to continue.....");
                           System.out.println("Reason :  \n "+me);
                     }
              }
      }
}
```

```
E:\Java_Online11\EXCEPTION>java Login sssit kphb
Valid User

E:\Java_Online11\EXCEPTION>java Login sai uk
Sorry Unable to continue.....
Reason :
 MyLoginException: Invalid UN|PW
```

## Throws:

- Technically Exception are classified into 2 types
  - Checked Exception
  - Un Checked Exceptions
- The Exception class which are the sub class of "RuntimeException" called "uncheckedException" where the

exception class which are other than "RuntimeException" are Checked Exception

- All The Checked Exception are meant for developer to handle by using try and catch block
- By the time of compiling the java program then java compiler will ensure is there any possibilities existed for checked Exceptions or not.
- By Chance if any checked exceptions are existed then java compiler will ensure that those exception are handled or not by the developer compiler using try and catch block, by chance if will fail to handle then that programs will not be compiled by the compiler it will raise an error
- All the checked exception must be handled by us . If your not interested to handle or if you want handle them later part of then at least we need to throw the exception to "JVM" using throws otherwise those programs will not compile
- Throws is not an exception handler rather it is an exception escaper
- Throws leads abnormal termination thus throws is not recommended

Note : It is always good programming practice to handle all checked or unchecked Exception for successful termination

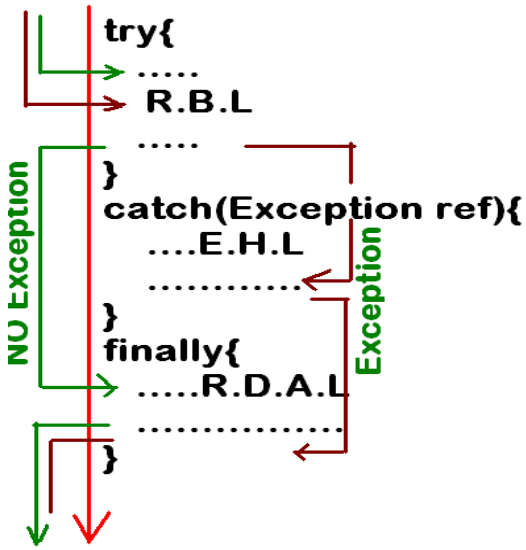
**Example On throws :**

```
import java.io.*;

class ReadData2
{
        public static void main(String args[])
                throws FileNotFoundException,EOFException
                                        or
                                IOException
                                        or
                                  Exception
        {
FileInputStream fis=new FileInputStream("Sample");
        }
}
```

## Finally:

- Finally is an optional block
- Finally block is mostly recommended whenever you want perform some operation compulsorily before terminating the program execution
- In the finally block it is recommended to write the logic related to resource de-allocations. Such closing the files, closing Databases , disconnection networks …
- Finally block will be executed all the cases when an Exception got raised or not .




**Example On finally:**

```
class Testing{
        public static void main(String args[]){
                try{
         System.out.println("Try Block ");
                        throw new ArithmeticException();
                }
                catch(NumberFormatException ne){
                        System.out.println("Catch Block ");
                        System.out.println("Statement 3");
                        System.out.println("Statement 4");
                }
                finally{
                        System.out.println("Finally");
                        System.out.println("Statement 5");
                        System.out.println("Statement 6");
                }
                System.out.println("OUTSIDE");
                System.out.println("Statement 7");
                System.out.println("N T M");
```

```
        }
}
```

IQs:
Note:
1.only try{ } //invalid
2.only catch( ){ }//invalid
3.only finally( ){ } //invalid

4.try{ }
catch( ){ } //valid

5.try{ }
finally{ } //valid

6.try{ }
catch( ){ }
finally{ } //valid

7.try{ }
finally{ }
catch( ){ } //invalid

8.try{ }
catch( ... ){ } //valid
catch(....){ }

9.try{ }
finally{ }
finally{ } //invalid

10.try{
try{ } catch( ... ){ } //valid
}

```
catch(....){ }
```

```
11.try{  }
catch(....)  //valid
{
      try{ }
  catch(....)
}
```

```
12.try{ }  //valid
catch(....){ }
finally
{
      try{ }
      catch(....){ }
}
```