<u>Exception Handling</u>

An Exception is an abnormal condition that arises in a code squence at run time . In other words, it is a run time error.In Languages that do not support exception handling, errors has to be checked and handled manually through the use of error codes.
By Handling those exception, we can safeguard other parts of our program.

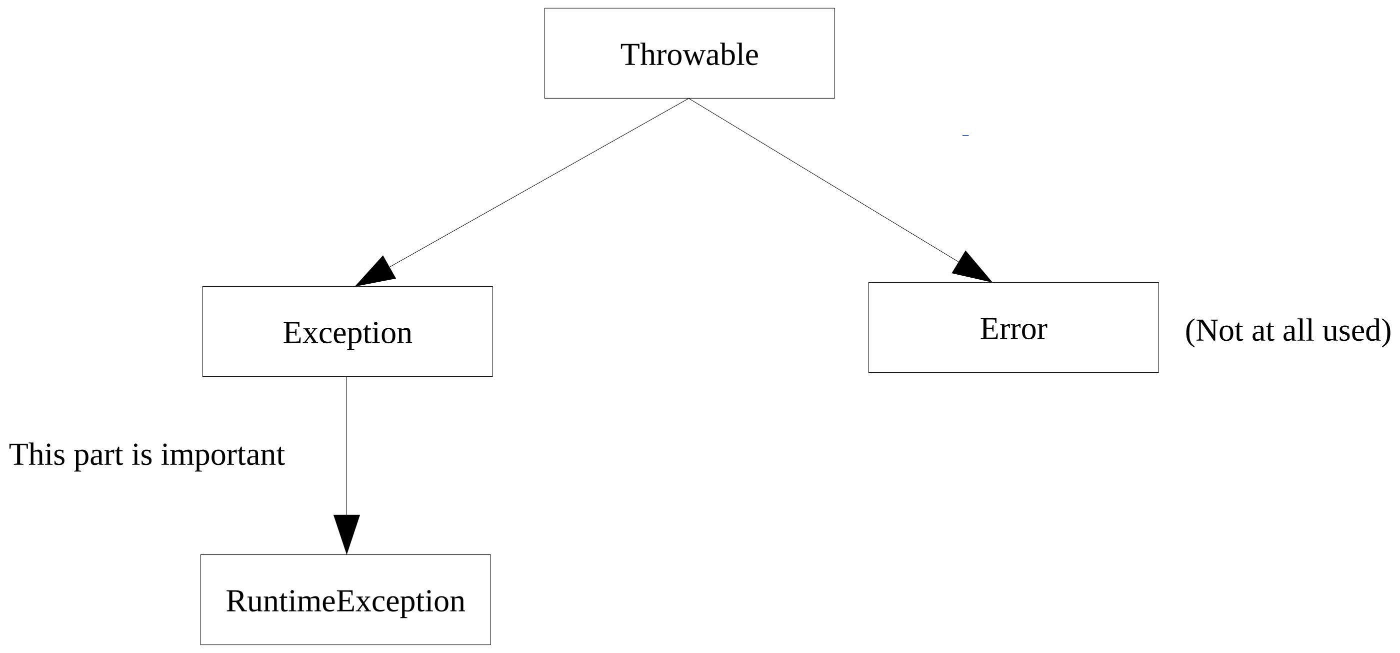
Fundamentals:

- A java exception is an object that describes an exceptional condtion htat has occured in a piece of code.
- When an exceptional occurs it is created and thrown in the method that caused that error. That method can handle the error or pass it on.
- But once an exception is thrown, it has to be handled at some part of your program.
- There are two types of exception – System generated exceptions and manual exceptions. System generated errors are automatically created and thrown.

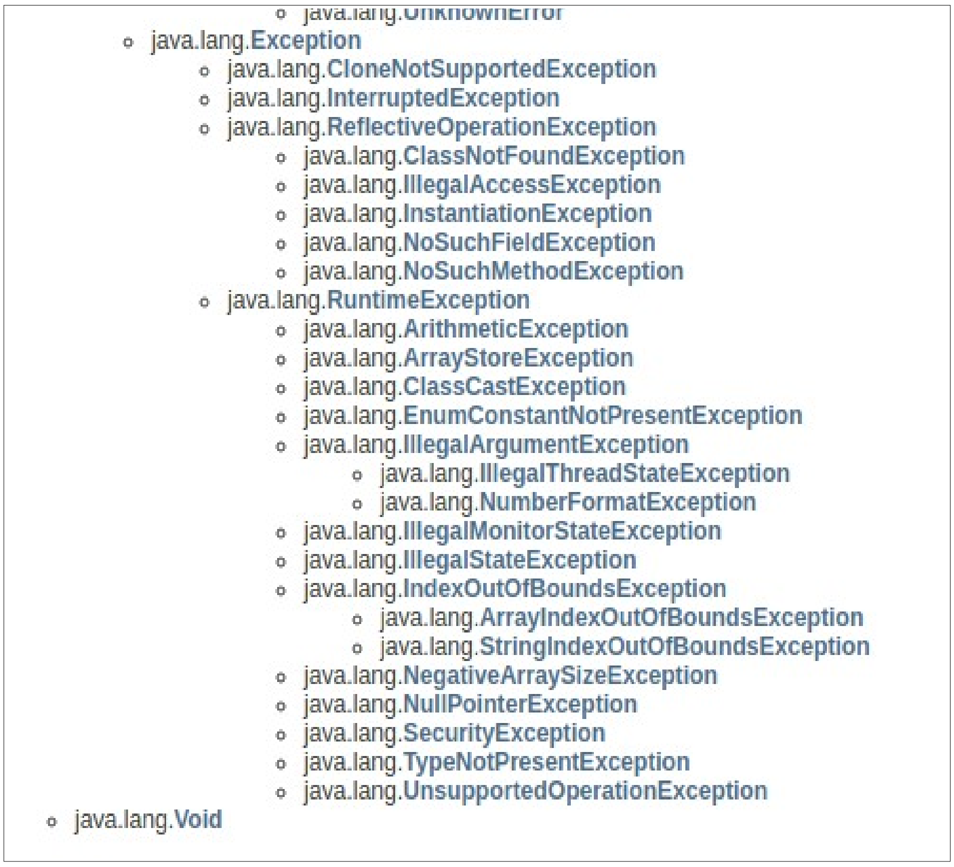
General form of exception-handling block

```
try{
    // code to be monitored
}
catch(Exceptiontype1 exob){
    // handler for Exceptiontype1
}
catch(Exceptiontype2 exob){
    // handler for Exceptiontype2
}
catch(Exceptiontype3 exob){
    // handler for Exceptiontype3
}
finally{
    // block of code to be executed after try block
ends
}
```

Exception Hierarchy



We are concerned with **Exception** class.It has one sub class called **RuntimeException** in which all exceptions are subclassed.



- java.lang.UnknownError
- java.lang.Exception
  - java.lang.CloneNotSupportedException
  - java.lang.InterruptedException
  - java.lang.ReflectiveOperationException
    - java.lang.ClassNotFoundException
    - java.lang.IllegalAccessException
    - java.lang.InstantiationException
    - java.lang.NoSuchFieldException
    - java.lang.NoSuchMethodException
  - java.lang.RuntimeException
    - java.lang.ArithmeticException
    - java.lang.ArrayStoreException
    - java.lang.ClassCastException
    - java.lang.EnumConstantNotPresentException
    - java.lang.IllegalArgumentException
      - java.lang.IllegalThreadStateException
      - java.lang.NumberFormatException
    - java.lang.IllegalMonitorStateException
    - java.lang.IllegalStateException
    - java.lang.IndexOutOfBoundsException
      - java.lang.ArrayIndexOutOfBoundsException
      - java.lang.StringIndexOutOfBoundsException
    - java.lang.NegativeArraySizeException
    - java.lang.NullPointerException
    - java.lang.SecurityException
    - java.lang.TypeNotPresentException
    - java.lang.UnsupportedOperationException
- java.lang.Void

Link for the above page:
https://docs.oracle.com/javase/7/docs/api/java/lang/package-tree.html

Uncaught Exception:

```
Class a{
public static void main(String[] ar){
        int a,b;
        b = 0;
        System.out.println(“Before Exception”);
        a = 40/b;
        System.out.println(“After Exception”);
    }
}
```

The same class with try-catch

```
Class a{
    public static void main(String[] ar){
        int a,b;
        b = 0;
        System.out.println(“Before Exception”);
        try{
            a = 40/b;
        }
        catch(Exception ex){
            System.out.println(“Exception Occured”);
        }
        System.out.println(“After Exception”);
    }
}
```

Another Example

```
Class a{
    public static void main(String[] ar){
        try{
            throwDemo();
        }
        catch(ArithmeticException ex){
            System.out.println(ex)
        }
    }

    void throwDemo(){
        int a,b;
        b = 0;
        a = 40/b;
    }
}
```

Multiple catch Blocks

When an exception is thrown, catch block is inspected one by one in order. Catch block which matches with a thrown exception type is executed.

1)IndexOutofBoundsException

```
class X{
    public static void main(String[] a){
        int a,b;
        b = 0;
        int[] x = new int[]{1,2,3};
            try{
                a=x[30];
            }
            catch(ArithmeticException ex)
            {
                System.out.println(ex);
            }
            catch(IndexOutOfBoundsException ex){
                System.out.println(ex);
            }
            System.out.println("Print me");
    }
    }
}
```

2)ArithmeticException

```
class X{
    public static void main(String[] a){
        int a,b;
        b = 0;
        int[] x = new int[]{1,2,3};
        try{
            a = 40/b;
        }
        catch(ArithmeticException ex)
        {
            System.out.println(ex);
        }
        catch(IndexOutOfBoundsException ex){
            System.out.println(ex);
        }
        System.out.println("Print me");
    }
}
```

Caution:

Have Exception hierarchy in mind while using multiple catch blocks. If you handled an exception of a class, Do not handle it’s subclass’s exceptions .

```
class X{
    public static void main(String[] a){
        int a,b;
        b = 0;
        int[] x = new int[]{1,2,3};
        try{
            a = 40/b;
        }
        catch(Exception ex)
        {
            System.out.println(ex);
        }
        catch(IndexOutOfBoundsException ex){
            System.out.println(ex);
        }
        System.out.println("Print me");
    }
}
```

The above will not compile. Becuase Exception is a superclass of IndexOutofBoundsException class.

Use of throw

so far we are handling exceptins that are thrown by java run time. But it is possible to throw an exception of our own. We can use **throw** keyword for this. It is oftern used when dealing with our own exceptions.

```
Class Throw{
    public static void main(String[] args){
        try{
            throwdemo();
        }
        catch(NullPointerException ex){
            System.out.println(ex);
        }
    }

    void throwdemo(){
        throw new NullpointerException(“Demo
Exception”);
    }
}
```

Use of throws

When a method is capable of throwing an exception, then we have to specify all the exceptions that the method can throw.
For example,

```
public void dosomething(int x, String[] s) throws InterruptedException,ClassNotFoundException{
        // method body
}
```

Note:
This is not necessary for type RuntimeException

Example:

```
class ThrowsExample{

    public static void main(String[] d){
        try{
            Throw();
        }
        catch(ClassNotFoundException ex){
            System.out.println(ex);
        }
    }

    public void Throw() throws
ClassNotFoundException{
        throw new ClassNotFoundException();
    }

}
```

Creating Own Exceptions

To create our own Exceptions, We simple extend the class Exception. If you want to provide clear error messages, then override **toString()** method.