Exception Handling

Errors are automatically of two difference types — Compile time error and runtime error. A runtime error is known as a exception.

```
class SimpleDivision {
        public static void main() {
                int a=2, b=0, c;
                system.out.println("The result of a/b is");
                c=a/b;
                system.out.println©;
                system.out.println("Thank You");
}       }
```

```
O/P     The result of a/b is
        java.lang.arithmatic exception / by 0
        c:\Java>_
```

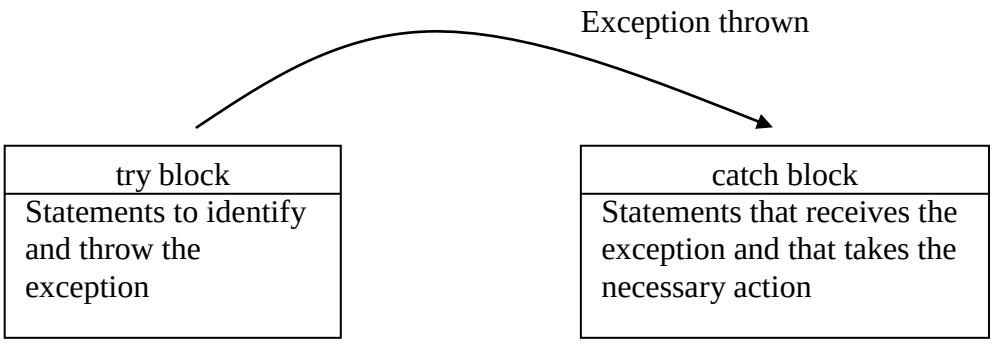
Exception and Exception handling :-

An exception is a runtime error. An exception is an abnormal condition that arises in a java program when an error is encountered. Java interpreter when encounter an exception (runtime error) it creates an object of that exception type and throws it to the java Runtime. Actually the interpreter informs that an error has taken place. If this object created by the java runtime is not handled or dealt with, the interpreter generates an error message and terminates the program prematurely. If we want to continue with the remaining part of the program, then the exception object is to be catched to deal with the exception and take the necessary steps. This mechanism is known as Exception Handling. There are four phase in the entire exception handling process —

1. Identify the error. (Hit the exception)
2. Inform about the error. (Throws the exception)
3. Accept the error information. (Catch the exception)
4. Take necessary actions. (Handles the exception)

Syntax of Exception Handling :-

```
try{
        //code that generates the exception
}
catch (Exception type exception object) {
        //code that handles the exception
}
```



Example :-

```
class SimpleDivision {
        public static void main() {
                int a=2, b=0, c=6;
                system.out.println("The result of a/b is");
                try {
                        c=a/b;
                        system.out.println(c);
                        system.out.println("Division done");
                }
                catch(Arithmetic Exception ae) {
                        system.out.println("Division not done due to zero   ");
                }
                system.out.println("Value of C is"+c);
                system.out.println("Thank you");
}       }
```

O/P

```
The result of a/b is
Division not done due to zero denomince
Value of C is 6
Thank you
```

Note 1: The statements suspectable of creating an exception are kept in the try block.

Note 2: If an exception is detected in any of the statement in the try block, the remaining of the statements in the try block are jump passed and execution passes on to the catch block.

Note 3: Catch block acts as a method which takes as parameter an object of that exception type which is thrown by the try block.

```
import java.util.*;
class RandomDivision {
        public static void main() {
                Random r = new Random();
                int a, b, c;
                system.out.println("The result of a/b is");
                try {
                        a=r.nextInt(10);
                        b=r.nextInt(10);
                        c=a/b;
                        system.out.println("a/b",+c);
                        system.out.println("This is not printed when b=0");
                }
                catch(Arithmetic Exception ae) {
                        system.out.println("This is not printed when b=0");
        }       }
        c=r.nextInt();
        system.out.println("Value of C is", +c);
}
```

O/P

When a=6, b=2

(1) a/b=3

This is not print when b=0

Value of C is 6

(2) When a=0, b=0

This is not print when b=0

Value of C is 9

Describing the exception :-

So far we are using our own exception message, we can also use Java's building exception diseriptim mechanism for e.g. in the above example we can write as

```
catch(ArithmaticException ae) {
    system.out.println(ae);
    }
```

When an exception really do take place we see the message java.lang. ArithmaticException:\by Zero.

If we want to known the origin of the exception we use the printStackTrace() method e.g. referring to the above ex. We write —

```
catch(ArithmaticException ae) {
    system.out.println(ae);
    ae printStackTrace();
    }
java.lang. ArithmaticException:\by Zero
at RandomDivision.main (RandomDivision.java: I)
```

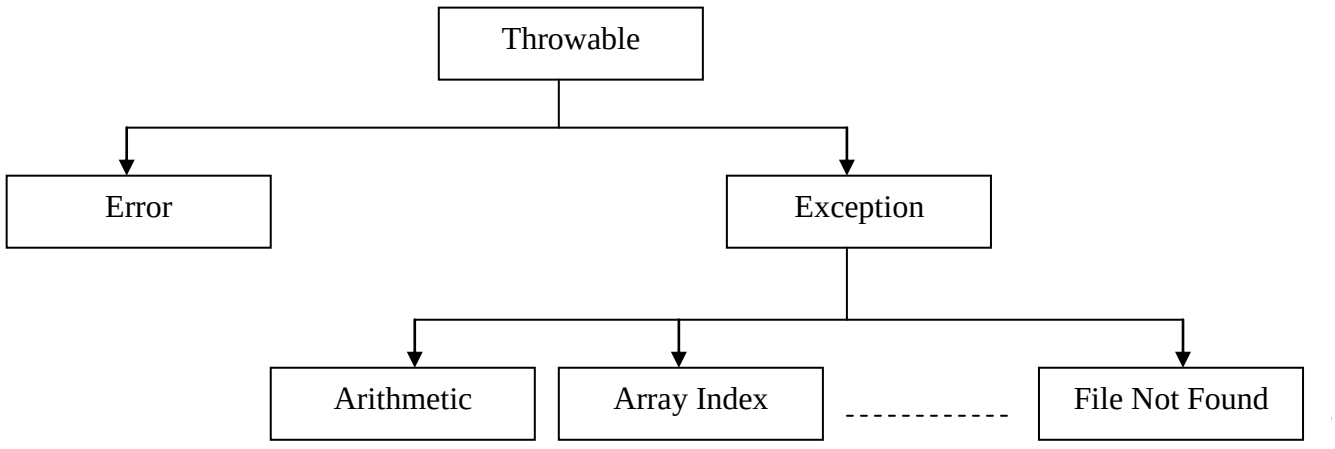
Built in Exception classes in Java :-

Some of the important built in Exception in Java are —

1. ArithmeticException(class).
2. ArrayIndexOutofBoundsException(class).
3. ArrayStoreException(class).
4. ClassCastException(class).
5. NullPointer Exception(class).
6. IOException Exception(class).
7. Interrupt Exception(class).
8. StringIndexOutofBound Exception(class).
9. NumberFormat(class).
10. FileNotFound Exception(class).

There are other exception class also.

Exception classes Hierarchy :-



Multiple catch Statement :-

A particular try block in a Java program may thrown more than one exception. Each exception in to be handled by separate catch statement. When ever an exception in encountered all the catch methods or catch blocks are examined and that particular catch block which accepts an exception which has been thrown by a try block in executed by passing the others. If no matching catch block is found responsibility goes to the default exception handled which gives a message and terminates the program permanently.

Example :-

```
import java.util.*;
class MultipleCatch {
        public static void main () {
                Random r = new  Random();
                int c[] = {9, 10};
                int a, b, d;
                try {
                        a = r.nextInt(10);
                        b = r.nextInt(10);
                        d=a/b;
                        system.out.println("d=", +d);
                        c[11] = 4;
                        system.out.println("End of try");
                }
                catch(ArithmaticException ae) {
                        system.out.println(ae);
                        ae printStackTrace();
                }
                catch(ArrayIndexOutofBoundException aiobe) {
                        system.out.println(aiobe);
                        aiobe printStackTrace();
                }
system.out.println("Thanks");
}       }
```

Output –

```
a=9     b=6     d=1
Java.lang.ArrayIndexOutofBoundException…
at Multiple catch main(Sample.java: 11)
Thanks
a=9     b=0     d=1
Java.lang.ArithmaticException …
at Multiple catch main(Sample.java: 9)
Thanks
```

throw clause :-

Some times we may not depend upon the java interpreter to throws an exception object, we may ourselves throw an exception (object) to the java runtime. This is accomplished by the throw clause —

```
import java.utill.*;
class ThrowTest {
    static void throwDemo(Random r) {
        try{
            if(r==null)
                throw new NullPointerException();
            int a = r.nextInt(10);
            system.out.println("a=", +a);
        }
        catch(NullPointerException npe) {
            system.out.println(npe);
            npe.printStackTrace();
        }
    }
    public static void main() {
        Random r = new Random();
        throwDemo(r);
        r=null;
        throwDemo(r);
}       }
```

Output :-
a=9
java.lang.null pointer Exception
at ThrowTest.mainthrowDemo(Sample.java: 6)

throws clause :-

Many a time, a method can generate an exception, does not handle it and poses on the responsibility of handling the exception to the calling method. Some methods (invoked methods) need to use the throws' clauses to inform the compiler that the possible exception is not handled by it self, but by the calling method.

Syntax :-

```
access return type method name(paramit… list)
    throws exception1, …. Exception {
        //bodey of the exception
    }
```

```
import java.io.*;
class ThrowsTest {
        static void getData() throws IOException {
        system.out.println("Enter your age");
        DataInputStream d = new DataInputStream(System.in);
        int age = Integer.parseInt(d.readLine());
        system.out.println("Your age=" +age);
        }
public static void main() {
        try {
                getData();
                }
        catch(IOException ie) { system.out.println(ie);}
}       }
```

Example :-

```
import java.io.*;
import java.util.*;
class MainthrowsException {
        public static void main(string v[]) throws IOException, ArithmaticException {
                Random r = new Random();
                int a = r.nextInt(10);
                DIS d= new DIS(system.in);
                system.out.println("Enter a number");
                int b= Integer.parseInt(d.readLine());
                int c=a/b;
                system.out.println("Thank You");
}       }
```

Note :- Although we can use throws clause in main(). In that case we have to scarifies the exception message and the statement.

User Defined Exception :-

If we want to throw an exception of our own depending on the condition we need to create an exception class of our own and throw an object of that exception class, when ever a condition is satisfied.

```
import java.util.*;
class MyException {
        public string toString() {
                return("Random number is b");
}       }
class MyExceptionTest {
        public static void main() {
                Random r = new Random();
                Try {
                        If(a==6)
                                throw new MyException();
                        system.out.println("a=" +a);
                }
                catch(MyException me) {
                        system.out.println(me);
                }
}       }
```

Output :-
a=6
Random number is 6