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>_

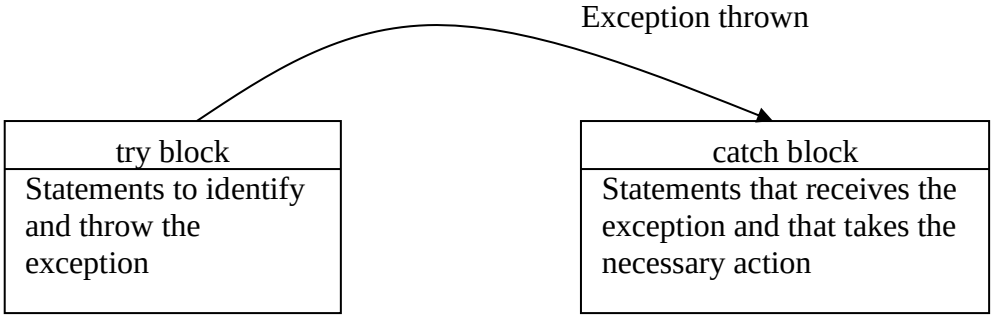
Exception and Exception handling :-

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)

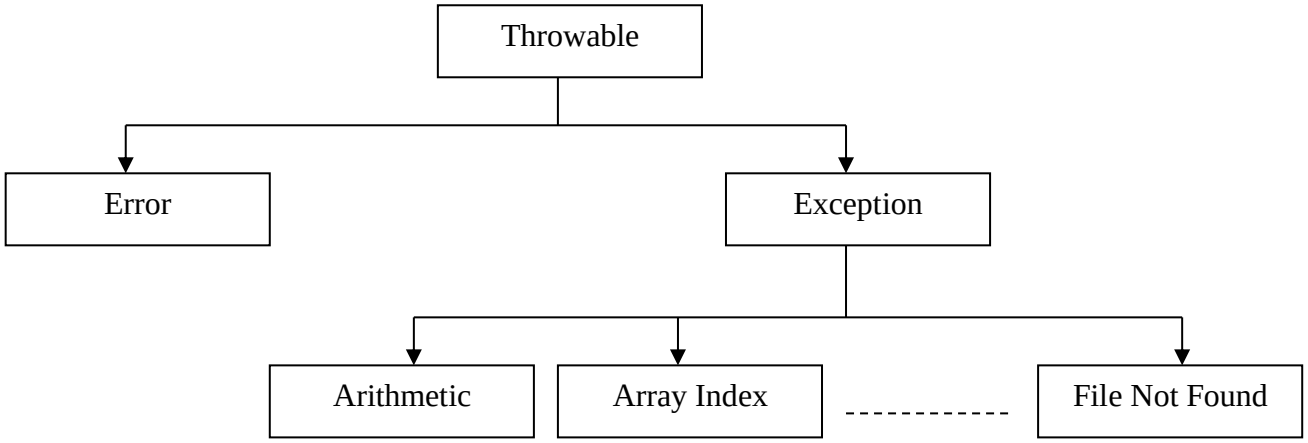
Built in Exception classes in Java :-

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) {
```

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
```

```
                system.out.println(aiobe);
                aiobe printStackTrace();
        }
system.out.println("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

Wrapper Class :-

Several built-in methods in java take as arguments objects of class type. But sometimes we need to pass through these methods variables of built in type. Thus in these cases we need to convert the variables of basic type to objects of class type. This is accomplished by wrapper classes can be used to —

1. Convert data of basic types to objects.
2. Convert objects to basic type.
3. Convert strings to basic type.

There are several wrapper classes like Integer, Float, Long, Double, Sont, Byte, Character and Boolean.

1. Let us consider the basic type variables int I, float f, double d, long l. To convert this basic variables to objects of the corresponding we make use of the wrapper class.
   ```
   Integer intobj = new Integer(i);
           s.push(intobj);
   Float flotobj = new Float(f);
   Double doubleobj = new Double(d);
   ```
2. Converting object to basic type:—
   ```
   int x = intobj.intvalue();
   float f = floatobj. floatvalue();
   double d = doubleobj.doublevalue();
   lang l = langobj.langvalue();
   ```
3. Converting string to Basic type:—
   ```
   String s = "1"; String f = "33.99"
   int x = Integer.valueof(s).intvalue;
   ```

Note :-
For integer and long class we have two short cart methods called parseInt() and parseLong() which can be used to convert string to integer and long variables.

```
String s;
int x = Integer.parseInt(b);
long l = Long.parseLong(s);
```

```
        int obj
float f = Float.vlueof(f).floatvalue;
String d="1.aaaaa001";
double d= Double.valueof(d).doublevalue;
lang l = lang.valueof(l).langvalue;
```

Vector Class :-

Arrays in Java are some what static in the sense that they are not extensible once they are declared. To create dynamic arrays we can use the vector class of the java.util package. A vector object is actually an array which can hold objects of different type and form which objects can be removed at any point of time. A vector object is created as below:-

(i) Vector v = new Vector();
This creates a vector of initial capacity of 10 and increases itself by double the amount at each stage.

(ii) Vector v = new Vector(size);
This creates a vector of initial capacity size and increases itself by double the amount at each stage.
e.g. - Vector v = new Vector(b, 2);

(iii) Vector v = new Vector(size, incr);
This creates a vector of initial capacity size and increases itself by 'incr' amount at each step.
e.g.- Vector v = new Vector(b, 2);

Advantage of Vector over an Array :-

1. A vector is extensible at any stage unlike an Array.
2. A vector can hold object of different types which an array can't.
3. Any object from a vector can be deleted at any point of time.

Disadvantage :-

A vector can store object only and not variables of basic data type.

Vector methods:-

The vector class has several methods for manipulating its object.

Vector v = new Vector();

1. v.capacity(); // returns the capacity of vector.
2. v.size(); // returns the number of elements present in the vector.
3. v.firstElement(); // returns the fast object of the vector.
4. v.lastElement(); // returns the last object of vector.
5. v.addElement(t); // adds an object at the end of the vector.
6. v.elementAt(n); // returns the elements at the n th portion.
7. v.removeElementAt(n); // deletes the element at the n th position.
8. v.contains(obj); // returns true if the vector realy contains object.
9. v.removeElement(item); // removes the element called item.

Example :-

```
import java.util.*;
class vectorDemo {
        public static void main() {
                int i=2; float f=22.45; bolean b=true; Sring s="India";
                Vector v = new Vector(3, 2);
                system.out.println("Initial capacity" +v.capacity());
                system.out.println("Initial size" +v.size());
                v.addElement(new Integer(i));
                v.addElement(new Float(i));
                v.addElement(new Boolean(b));
                v.addElement(s);
                system.out.println("New Capacity is" +v.capacity());
                system.out.println("New Size is" +v.size);
                if(v.contains(s))
                        system.out.println("Vectors contains India");
                system.out.println("Vector is");
```

Output :-

```
Initial Capacity = 3
Initial Size = 0
New Capacity = 5
New Size = 4
Vector Contains India
The Vector is
2
22.45
true
India
```

```
                Iterator itr = v.itrator();
                while(itr.hasNext())
                        system.out.println(itr.next());
}       }
```

Stack Class :-

Java has a stack class to implements the last in first out stack data structure. The stack class is actually a subclass of vector class and so uses all the methods of the vector class. In addition to this the stack class has some extra methods.

Stack st = new Stack;

st.push(Item); // inserts the element item into the stack.
st.pop(); // pops a element form the stack.
st.empty(); // returns true if the stack is empty.
st.peek();returns the topmost elements of the stack without removing it.

Example :-

```
import java.util.*;
class stackTest {
        public static void main() {
                int i;
                Stack st = new Stack();
                for(i=1; i<=10; i++) {
                        system.out.println("Stack is" +st);
                        st.push(new Integer(i));
                }
                system.out.println("Stack is" +s+);
                for(i=1; i<=10; i++) {
                        system.out.println("Popped item is" +st.pop());
                        if(st.empty()) {
                                system.out.println("Stack is empty");
                                break;
                        }
                system.out.println("Stack is" +s+);
                }
}       }
```

Output :-

```
Stack is []
Stack is [1]
Stack is [1 2]
Stack is [1 2 3]
……..
……..
Stack is [1 2 3 4 5 6 7 8 9]
Stack is [1 2 3 …….. 10]
Popped item is 10
Stack is [1 2 3 ……. 9]
Popped item is 9
Stack is [1 2 3 ….. 8]
……..
……..
Stack is []
Stack is empty
```

Garbage Collection :-

Introduction :-

Garbage Collection is the technique by which Java detects the objects that are no longer in use (an object is no longer in use if there is no more reference to it) and returns the space occupied by this objects to the free memory pool called help. The application developers never need to interface in this entire process. Java runtime invokes the garbage collection (in fact the ge() method) after certain interval of time automatically. There is no need to invoke it (ge()) explicitly as in C or C++.

Why is garbage collection done?

Objects when created are allotted memory so when these objects go out of use their space should be freed. Otherwise the huge family if object would invade (acquire) the entire memory and create memory crisis and since memory of a system is not infinite we need garbage collection form time to time.

The finalise() method :-

Sometimes an object needs to perform certain important task before being destroyed e.g. an object may need to close a file or discontinue a network connection before destination such important tasks are to be included in a method called finalise(). The finalized method is involved whenever the garbage collection tries to destroy the object. The general form of the finalise method is—

```
protected void finalise() {
        // body of the method
}
```

Note :-

1. The finallise method is invoked by the garbage collection just before destroying the object. So we don't know when exactly the method is called.
2. The finalise() method is declared in the object class and hence it can be overridden by any class.
3. The finalise() method is invoked only once by a particular object.
4. When a finalise() method throws an Exception, the Exception is ignored but the destruction of the object is halted.

   If a finally block is associated with a try, the finally block will executed upon the conclusion of try.

Invoking the Garbage Collection Explicitly :-

Although the garbage collection the invoked by the runtime after certain intervals of time, we may some times need to invoke it explicitly and de-allocate the unused object before the next appointed round of the garbage collector. The garbage collector method is invoked in the following way.

```
Runtime r = Runtime.getRuntime();
r.ge();
```

Funda 3 :-

There are two methods freeMemory() and totalMemory(), which can we used to view the memory space which is still free and the total memory space of the heap area.

Syntax:-

```
final long freeMemory();
final long totalMemory();
```

Both of these method returns the memory space in byte. They are invoked as —

```
long l = r. freeMemory();
long l = r. totalMemory();
```

Where r is an object of Runtime class.