// Exception Handling

2 terms

```
class E1
{
   public static void main ( )
     {
        Sopln (" KIIT ") ;
     }
}
```

(it compiles successfully)
but we can't display out

```
class E1
{
   public static void main ( )
     {
        Sopln (" KIIT ") ;
     }
   public static void main (String args[])
     {
      main( ) ;
     Sopln (" University") ;
     }
}
```

KIIT
UNIVERSITY

Types of Exception :-

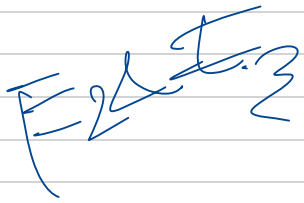
1. Check
2. Uncheck

Keywords :-

1. try
2. Catch
3. throw
4. throws
5. finally

class E1

{



* Object class is a superclass of all class.

throw - used to throw an exception explicitly.

```
class E4
 {
   public static void main (String ar[])
    {
      try
     {
       throw new ArrayIndexOutOfBoundException ("KIIT");
     }
      catch (ArrayIndexOutOfBoundsException e)
     {
       Sopln (e);
      }
    }
```

/* Output

java.lang.ArrayoutofBoundsEception : KIIT

In all built in exception there are 2 constructors :-

(i) Default

(ii) Parameterised Const. (String parameter)

// public String getMessage ()

→ The getMessage() method of the Throwable class is used to return a detailed message of the Throwable object which can also be null.

(Throwable class is a superclass of Error class)

```
class E5
 {
   public static void main (String ar[])
    {
      try
       {
         throw new ArrayIndexOutOfBoundException ("CSE");
       }
       catch (Exception e)
        {
          Sopln (" e.getMessage());
        }
    }
 }
```

/* Output

CSE

// public void printStackTrace ()

printStackTrace () method of java.lang.Throwable class used to print this throwable object along with other details like class name & line no where the exception occured means its backtrace

```
Class E6
 {
    public static void main ( String ar[])
     {
       try
        {
          throw new ArrayIndexOutOfBoundException ( " KIIT ");
        }
        catch ( Exception e)
         {
           e. printStackTrace ( );
         }
      }
   }
```

// try- finally

```
class E7
 {
    public static void main ( String ax[])
     {
       try
        {
         int a = 10/0 ;
         Sopln (" Value ") ;
        }
        finally
         {
          Sopln (" KIIT UNI ") ;
         }
      }
   }
```

/* Output

KIIT UNI

Exception in thread " main ". ..

. - . . . . . 10/0 . . . .

# User Defined Exception or Custom Exception

WAP to calc the factorial of a no if the no is positive or throw it the no is negative.

- All user defined exception classes are child class of exception class.

// * class may be empty or it may contain constructors.
* define that exception class first

```
class NegativeFactException
{

}
class E8
{
  void fact (int p) throws NegativeFactException
  {
   int f = 1 ;
   int ( p>=0)
    {
      for (int i= 1 ; i<=n ; i++)
       {
         f = f * i ;
       }
       Sopln (" Factorial of "+ p +" = " + f ) ;
       }
       else
       {
         throw new NegativeFactException ( ) ;
       }
     }
     public static void main (String args[])
     {
       E8 ob = new E8( ) ;
       try
       {
         ob.fact (-7) ;
       }
        catch ( NegativeFactException e )
        {
          Sopln (" Successfully run");
        }
     }
}
```

* throws keyword is associated with a func^ or method. (before { } )

* throws will be always associated but before { } .

Why ?

If there is a possibility of throwing an exception from the body of a method then we use throws keyword along with that exception name.

* It is mandatory for user def^ exception & check exception (all user defined exception are check exception

If there is a possibility of throwing multiple exception from a body of method, then we use throws keyword along with exception name then all exception key will be a separated key.

If there is a posibility of method throwing a exception then it is the responsibility of caller to handle it.

// throws keyword is not mandatory for unchecked exception

```
class E13
 {
   void show ( int p) throws ArithematicException
    {
      int f = 10/p ;
      Sopln (" Value of f = " + f);
    }
   public static void main ( String ar[])
    {
      E13 ob = new E13();
      try
       {
         ob.show (0);
       }
        catch ( ArithematicException e)
        {
         Sopln (e);
        }
    }
 }
```

is not mandatory but we can use

// Output :-
java.lang.ArithematicException : / by Zero

* User def^n exceptions are checked & Arithematic Exceptions are unchecked.

```
class E14      // Multiple catch
 {
   public static void main ( String ar[])
   {
    try
   {
   int a = Integer.parseInt (ar[0]) ;
   Sopln ("Value = " + a) ;
   int b = 10/a ;
   Sopln (" Value of b= " +b) ;
   }
    catch (ArithematicException e)
    {
      Sopln (e);
    }
    catch ( NumberFormatException e)
    {
     Sopln (e);
    }
   }
 }
```

* we have written 2 catch block

Case 1
if we don't pass any value from cmd line ; it will show out-of-bounds error bcoz ar[0] is dec.

Case 2
if we pass 4 in cmdline input
Output :-
Value = 4
Value of b = 2

Case 3
if we pass four in cmdline input
Output
NumberFormatException

Case 4
if we pass 0 in cmdline input
Output (10/0 is exception point)
Value = 0
java.lang.ArithematicException /by Zero

WAP to calc the factorial of a no if the no is positive or throw it the no is negative.

- All user defined exception classes are child class of exception class.

// * class may be empty or it may contain constructors.
* define that exception class first

```
class NegativeFactException
 {

 }
  class E8
   {
     void fact (int p) throws NegativeFactException
    }
     int f = 1 ;
     int ( p>=0)
      {
        for (int i= 1 ; i<=n ; i++)
          {
            f = f * i ;
          }
          Sopln (" Factorial of " + p + " = " + f ) ;
          }
           else
           {
             throw new NegativeFactException ( ) ;
           }
       }
        public static void main (String args[])
        {
          E8 ob = new E8( ) ;
          try
          {
            ob. fact (-7) ;
          }
           catch ( NegativeFactException e )
           {
             Sopln (" Successfully run");
           }
       }
```

* throws keyword is associated with a func or method. (before { } )

* throws will be always associated but before { }.

Why ?

If there is a possibility of throwing an exception from the body of a method then we use throws keyword along with that exception name.

* It is mandatory for user def^ exception & check exception (all user defined exception are check exception

If there is a possibility of throwing multiple exception from a body of method, then we use throw keyword along with exception name then all exception key will be a separated key.

If there is a posibility of method throwing a exception then it is the responsibility of caller to handle it.

Exception class is the super class of all exception ; but it is used at last catch block.