Throwable class Method to print Exception :
----
* Throwable class has provided the following methods to print the Exception object OR to print the error message of the exception.

1) public String toString() :
----
* It is a non static overridden method of Throwable class. It is used to convert the Exception object into String format.

* It provides the Exception message in the following format :
  Fully Qualified Name : error message

2) public String getMessage() :
----
* It is a predefined non static method of Throwable class. It is only used to print the **error message.**

3) public void printStackTrace() :
----
* It is a predefined non static Method of Throwable class. It is used to print the complete exception details like exception FQN, exception error message, the class name and method name where exception encounter as well as the line number.

```
package com.ravi.basic;

public class PrintStackTrace
{
    public static void main(String[] args)
    {
        System.out.println("Main Method started");

        try
        {
            String str = null;
            System.out.println(str.toUpperCase());
        }
        catch(Exception e)
        {
            System.out.println(e.toString());
            System.out.println("..................");

            System.out.println(e.getMessage());
            System.out.println("..................");

            e.printStackTrace();
        }

        System.out.println("Main Method ended");
    }

}
```

Working with Specific Exception :

While working with exception, in the corresponding catch block we can take Exception (super class) which can handle any type of Exception.

On the other hand we can also take specific type of exception (ArithmetiException, InputMismatchException and so on) which will handle only one type i.e specific type of exception.

```
//Program
package com.ravi.basic;

import java.util.InputMismatchException;
import java.util.Scanner;

public class SpecificException
{
    public static void main(String[] args)
    {
     System.out.println("Main started");

        Scanner sc = new Scanner(System.in);

        try
        {
            System.out.print("Enter your Roll :");
            int roll = sc.nextInt();
            System.out.println("Your Roll is :"+roll);

        }
        catch(InputMismatchException e)
        {
            e.printStackTrace();
        }
        sc.close();
        System.out.println("Main ended");
    }
}
```

```
public class ExceptionDemo
{
    public static void main(String[] args)
    {
        System.out.println("Main method started");
        try
        {
            throw new OutOfMemoryError();
        }
        catch (Error e)
        {
            System.out.println(e);
        }
        System.out.println("Main method ended");
    }
}
```

Working with infinity and Not a number :

10/0;   Infinity
10/0.0; Infinity [POSITIVE_INFINITY]

0/0;   Undefined
0/0.0; Undefined [NaN]

* While working with integral literal in both the cases i.e Infinity (10/0) and undefined (0/0) we will get java.lang.ArithmeticException and It is an abnormal termination because Java software people **has not provided any final and static variable to deal with Infinity and undefined.**

* On the other hand while working with floating point literal in both the cases i.e Infinity (10/0.0) and undefined (0/0.0) java software has provided the final and static variable support to deal with Infinity and undefined and It is a normal termination. The final and static variables are as follows :

  1) 10/0.0;   => POSITIVE_INFINITY
  2) -1/0.0;   => NEGATIVE_INFINITY
  3) 0/0.0;   => NaN (Not a number)

java.lang.Float and java.lang.Double classes are provided the support for these final and static variable, the same OR same type of variables are not available in Integeral Literal classes.

```
package com.ravi.basic;

public class InfinityFloatingPoint
{
    public static void main(String[] args)
    {
       System.out.println("Main Started");
       System.out.println(10/0.0);  //POSITIVE_INFINITY (Infinity)
       System.out.println(-10/0.0); //NEGATIVE_INFINITY (-Infinity)
       System.out.println(0/0.0);   //NaN
       System.out.println(10/0);
       System.out.println("Main Ended");
    }

}
```

Working with multiple try catch :

According to our application requirement we can provide multiple try-catch in my application to work with multiple execptions.
Example :

```
class MultipleTryCatch
{
   public void accept()
   {
       try
       {
       }
       catch(X e)
       {
       }

       try
       {
       }
       catch(X e)
       {
       }
   }
}
```

//Program :

```
package com.ravi.basic;

public class MultipleTryCatch
{
    public static void main(String[] args)
    {
      System.out.println("Main method Started!!!!");

      try
      {
         Object obj[] = new String[3];
         obj[0] = "NIT";
         obj[1] = "Java";
         obj[2] = 12;
      }
      catch(ArrayStoreException e)
      {
         System.err.println("Illegal Data insertion in the Array");
      }

      try
      {
         int arr[] = new int[-10];
      }
      catch(NegativeArraySizeException e)
      {
         System.err.println("Array size must be a positive Integer");
      }

      try
      {
         String str = "Java";
         int no = Integer.parseInt(str);
         System.out.println(no);
      }
      catch(NumberFormatException e)
      {
         System.err.println("Number is not in a proper format");
      }

      System.out.println("Main method Ended!!!!");
    }
}
```

Note : In the above programming approach, the drawback is, Our end clinet will get all the exception error message at a time so, It is not a good practise.

* In order to avoid the above said problem, Java has introduced industry standard concept i.e single try with multiple catch blocks

Single try with multiple catch blocks :

```
try
{
  int z = 10/0;
  System.out.println(arr[10]);
}
catch(ArrayIndexOutOfBoundsException e)
{
  System.out.println("Array Problem");
}
catch(ArithmeticException e)
{
   System.out.println("Divide by zero Problem");
}
catch(Exception e)
{
   System.out.println("General Problem");
}
```

```
try
{
   int z = 10/2;
   System.out.println(arr[10]);
}
catch(Exception e)
{
  Syserr("Please don't divide the no by zero");
}
```

* According to industry standard we should write try with multiple catch blocks so we can provide proper information for each and every exception to the end user.

* In this approach, even we have multiple exceptions in the try block but clinet will get exactly one exception message.

* While working with multiple catch block always the super class catch block must be last catch block otherwise we will get an error i.e Unreachable code.

* From JDK 1.7V to reduce the length of the code in catch block, they have provided **|** symbol as shown below.

```
try
{
  int z = 10/0;
  System.out.println(arr[10]);
}
catch(ArithmeticException |  ArrayIndexOutOfBoundsException e)
{

}
```

//Program :

```
package com.ravi.basic;
public class MultyCatch
{
    public static void main(String[] args)
    {
        System.out.println("Main Started...");
        try
        {
            int c = 10/0;
            System.out.println("c value is :"+c);

            int []x = {12,78,56};
            System.out.println(x[4]);
        }

        catch(ArrayIndexOutOfBoundsException e1)
        {
            System.err.println("Array is out of limit...");
        }
        catch(ArithmeticException e1)
        {
            System.err.println("Divide By zero problem...");
        }
        catch(Exception e1)
        {
            System.out.println("General Problem");
        }

        System.out.println("Main Ended...");
    }
}
```

```
package com.ravi.basic;

public class MultyCatch1
{
    public static void main(String[] args)
    {
        System.out.println("Main method started!!!");
        try
        {
           String str1 = "null";
           System.out.println(str1.toUpperCase()); //NULL

           String str2 = "Ravi";
           int x = Integer.parseInt(str2);
           System.out.println("Number is :"+x);
        }
        catch(NumberFormatException  | NullPointerException  e)
        {
          if(e instanceof NumberFormatException)
          {
             System.err.println("Number is not in a proper format");
          }
          else if(e instanceof NullPointerException)
          {
             System.err.println("ref variable is pointing to null");
          }
        }
        System.out.println("Main method ended!!");
    }

}
```