**Exception Handling in Java**  
> An Exception is an event, it occurs during execution of a program,   
when normal execution of the program is interrupted.  
  
> Exception handling is mechanism to handle exceptions

The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling.  
--------------------------  
Common Scenarios where exceptions may occur:  
**1) Scenario where ArithmeticException occurs**  
If we divide any number by Zero there ArithmeticException occurs  
  
Ex:  
  
int a = 10/0;  
----------------------------------------------  
**2) Scenario where NullPointerException occurs**  
if we have null value in any variable, performing any operation by the variable,  
  
ex:  
  
String s = null;  
System.out.println(s.length());//NullPointerException  
---------------------------------------------------  
**3) Scenario where NumberFormatException occurs**  
The wrong formatting of any value, may occur  NumberFormatException  
  
Ex:  
String s = "abc";  
        int y = Integer.parseInt(s);  
        System.out.println(y);//NumberFormatException  
--------------------------------------  
**4) Scenario where ArrayIndexOutOfBounds exception occurs**  
If we are inserting any value in the wrong index.  
  
Ex:  
int [] a = new int [5];  
        a[10] = 100;  
        System.out.println(a[10]);//ArrayIndexOutOfBounds  
--------------------------------------------------------------  
**Java Program Example:**------------  
int a =10;  
int b = 0;  
int result;  
result = a/b;  
System.out.println(result);  
System.out.println("Hello Java");  
System.out.println("Hello Selenium");  
----------------------------------  
**Use try block for Exception Handling**  
Syntax:  
--------  
try {  
Statements  
------  
-------  
------  
}  
catch (exception e) {  
Exception handling code  
}  
-----------------------------------  
Java Program With Exception handling  
int a =10;  
int b = 0;  
int result;  
  
try{  
result = a/b;  
System.out.println(result);  
}  
catch (ArithmeticException e){  
System.out.println("Divided by Zero Error");      
}  
System.out.println("Hello Java");  
System.out.println("Hello Selenium");  
}  
--------------------------------------  
Multiple try blocks for handling multiple exceptions  
Example:  
int a =10;  
int b = 0;  
int result;  
int [] array1 = new int [4];  
  
try{  
result = a/b;  
System.out.println(result);  
}  
catch (ArithmeticException e){  
System.out.println("Divided by Zero Error");      
}  
          
try{  
array1[10]= 100;  
System.out.println(array1[10]);  
}  
catch (ArrayIndexOutOfBoundsException e){  
System.out.println("Array Out of Bound Error");  
}  
System.out.println("Hello Java");  
System.out.println("Hello Selenium");  
}  
----------------------------------------------

**RunTimeException** is an unchecked **exception**. ... You **can throw** a**java**.lang.**RuntimeException** (or any derived **exception**), as it is a normal**exception**. The difference to other **exceptions** is that you do not need to mention it in the list of **thrown** but uncaught **exceptions** a method **can throw**, ie.Oct 20, 2013

**Finally**

* Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

