# Topic: 4.3.3 Exception handling

An exception is a special condition that changes the normal flow of the program execution. That is, when an event occurs that the compiler is unsure of how to deal with. Exceptions are the programming languages way of throwing up its hands and saying, "I can't deal with this situation, you need to."

There are a number of events that may seem awkward to the compiler of the programming language. Here are the most common examples:

- Your code expects a value from an input that is currently null.
- Dividing by zero.
- Accessing an array index that is out of bounds.
  (Underflow, or Overflow)
- A string is entered where a number is required as input.
  (or any similar data-type mismatch examples)
- Trying to access a file that isn't at the location specified.

In all these instances, we are trying something that the language deems impossible, and an exception is thrown.

Therefore, a good programmer should be more alert to the parts of the program that could trigger errors and should write error handling code to help him/her in managing the errors. Writing error handling code should be considered a good practice for programmers. However, there should not be too many error handling codes in the program as it creates problems for the programmer to maintain and troubleshoot the program later.

Error handling is an essential procedure in programming because it can help make the program error-free. An error-free program can run smoothly and efficiently, and the user does not have to face all sorts of problems such as program crash or system hang.
So in general, exception handling should be used whenever there is a possibility that an error may occur.

The Syntax for error handling in Visual Basic is **"On Error GoTo program_label"** where "program_label" is the section of the code that is designed by the programmer to handle the error committed by the user. Once an error is detected, the program will jump to the program_label section for error handling. It acts like a bookmark in VB.

# Topic: 4.3.3 Exception handling

**Example:**

The following code shows how to handle error 'Division by Zero'.

```
01 Private Sub CmdCalculate_Click()
02
03 Dim firstNum, secondNum As Double
04 firstNum = Txt_FirstNumber.Text
05 secondNum = Txt_SecondNumber.Text
06 On Error GoTo error_handler
07 Lbl_Answer.Caption = firstNum / secondNum
08 Exit Sub        'To prevent error handling even the inputs are valid
09
10 error_handler:
11 Lbl_Answer.Caption = "Error"
12 Lbl_ErrorMsg.Visible = True
13 Lbl_ErrorMsg.Caption = " You attempt to divide a number by zero!Try again!"
14
15 End Sub
```

The line 'On Error GoTo error_handler' is executed when there occurs an error in DIVISION.

When we input secondnum=0 then the 'error_handler' section of code is executed, simple!

It may be noticed that if there is no error, the error handling part of the code is NOT executed because of the EXIT SUB present on line 8.

## Java try-catch

### Java try block

The Java try block is used to enclose the code that might throw an exception, It must be used within the method. Java try block must be followed by catch block.

### Java catch block

The Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch blocks with a single try.

# Topic: 4.3.3 Exception handling

Take the following code for example.

```
public class Testtrycatch1{
  public static void main(String args[]){
     int data=50/0;//may throw exception
     System.out.println("rest of the code...");
}
}
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
```

Because of the exception in the code above, the rest of the code after the exception will not be compiled, the compiler will print an error message and terminate the faulty program.

We can easily resolve this issue by adding a try-catch block.

```
public class Testtrycatch2{
  public static void main(String args[]){
   try{
     int data=50/0;
   }catch(ArithmeticException e){System.out.println(e);}
   System.out.println("rest of the code...");
}
}
```

Output:

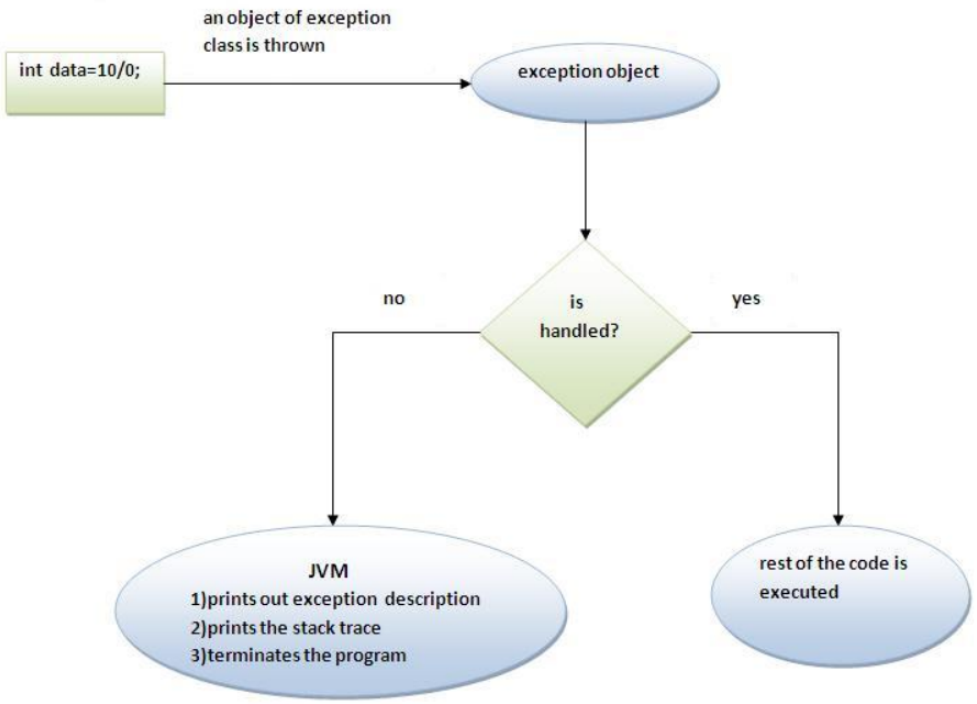
```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Now, as shown in the output above, the rest of the lines of code is executed

i.e. the statement is printed.

## Topic: 4.3.3 Exception handling

Internal working of java try-catch block



## Topic: 4.3.3 Exception handling

Similarly in the language Python, there are different kinds of errors that you may face such as ValueError, TypeError, NameError, IOError, EOFError, SyntaxError, etc…

**EOF Error:**

Raised when one of the built-in functions (input() or raw_input()) hits an end-of-file condition (EOF) without reading any data.

**Handling EOF Errors:**

```
import sys
try:
   name = raw_input("what is your name?")
except EOFError:
   print "\nYou did an EOF... "
   sys.exit()

If you do an ctrl+d, you will get an output like this:
>>what is your name?
>>You did an EOF...
```

**Keyboard Interrupt:**

Raised when the user hits the interrupt key (normally Control-C or Delete).

**Handling of Keyboard Interrupt:**

```
try:
   name = raw_input("Enter your name: ")
   print "You entered: " + name
except KeyboardInterrupt:
   print "You hit control-c"

If you press ctrl+c, you will get an output like this:
>>Enter your name: ^C
>>You hit control-c
```

## Topic: 4.3.3 Exception handling

**Value Error:**

Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value, and the situation is not described by a more precise exception.

**Handling of Value Error:**

```
while True:
try:
   x = int(raw_input("Please enter a number: "))
   break
except ValueError:
   print "Oops!  That was no valid number.  Try again..."
```