## Java try block and Java catch block

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 either catch or finally block.

#### **Syntax of java try-catch**

 try{

 //code that may throw exception

 }catch(Exception\_class\_Name ref){}

Which we can also refer to visual basic when someone is using try and end try, it explains that all codes which we be used within must be examined before to be compiled and tell the program If there is an error or not…

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

You can use multiple catch block with a single try.

To guard against and handle a run-time error, simply enclose the code that you want to monitor inside a **try** block. Immediately following the **try** block, include a **catch** clause that specifies the exception type that you wish to catch. To illustrate how easily this can be done, the following program includes a **try** block and a **catch** clause which processes the **ArithmeticException** generated by the division-by-zero error:

**Synthetic code:**

class Exc2 {   
public static void main(String args[]) {   
int d, a;   
try { // monitor a block of code.   
d = 0;   
a = 42 / d;   
System.out.println("This will not be printed.");   
} catch (ArithmeticException e) { // catch divide-by-zero   
error   
System.out.println("Division by zero.");   
}   
System.out.println("After catch statement.");   
}   
}

This program generates the following output:

Division/0  
After catch statement.

Notice that the call to **println( )** inside the **try** block is never executed. Once an exception is thrown, program control transfers out of the **try** block into the **catch** block. Put differently, **catch** is not "called," so execution never "returns" to the **try** block from a **catch**. Thus, the line "This will not be printed." is not displayed. Once the **catch** statement has executed, program control continues with the next line in the program following the entire **try**/**catch** mechanism.

A **try** and its **catch** statement form a unit. The scope of the **catch** clause is restricted to those statements specified by the immediately preceding **try** statement. A **catch** statement cannot catch an exception thrown by another **try** statement (except in the case of nested **try** statements, described shortly). The statements that are protected by **try** must be surrounded by curly braces. (That is, they must be within a block.) You cannot use **try** on a single statement.

The goal of most well-constructed **catch** clauses should be to resolve the exceptional condition and then continue on as if the error had never happened. For example, in the next program each iteration of the **for** loop obtains two random integers. Those two integers are divided by each other, and the result is used to divide the value 12345. The final result is put into **a**. If either division operation causes a divide-by-zero error, it is caught, the value of **a** is set to zero, and the program continues.

**Codes wirting down**

// Handle an exception and move on.   
import java.util.Random;   
class HandleError {   
public static void main(String args[]) {   
int a=0, b=0, c=0;   
Random r = new Random();   
for(int i=0; i<32000; i++) {   
try {   
b = r.nextInt();   
c = r.nextInt();   
a = 12345 / (b/c);   
} catch (ArithmeticException e) {   
System.out.println("Division by zero.");   
a = 0; // set a to zero and continue   
}   
System.out.println("a: " + a);   
}   
}   
}