try and catch in Java

Try and catch both are Java keywords and used for exception handling. The try block is used to enclose the suspected code. Suspected code is a code that may raise an exception during program execution.

Try block

For example, if a code raises arithmetic exception due to divide by zero then we can wrap that code into the try block.

```
try {
  int a = 10;
  int b = 0
  int c = a/b; // exception
}
```

Catch block

The catch block also known as handler is used to handle the exception. It handles the exception thrown by the code enclosed into the try block. Try block must provide a catch handler or a finally block.

The catch block must be used after the try block only. We can also use multiple catch block with a single try block.

```
try{
  int a = 10;
  int b = 0
  int c = a/b; // exception
}catch(ArithmeticException e){
  System.out.println(e);
}
```

Try Catch Syntax

To declare try catch block, a general syntax is given below.

```
try {
  // suspected code
}
catch (ExceptionClass ec) {}
```

Exception handling is done by transferring the execution of a program to an appropriate exception handler (catch block) when exception occurs.

Example: Exception Handling

Now lets understand the try and catch by a simple example in which we are dividing a number by zero. The code is enclosed into try block and a catch handler is provided to handle the exception.

```
class Excp
{
  public static void main(String args[])
  {
    int a,b,c;
    try
    {
      a = 0;
      b = 10;
      c = b/a;
      System.out.println("This line will not be executed");
    }
    catch(ArithmeticException e)
    {
```

```
      System.out.println("Divided by zero");
    }
    System.out.println("After exception is handled");
  }
}
```

Output:

Divided by zero

After exception is handled

Explanation:

An exception will be thrown by this program as we are trying to divide a number by zero inside try block. The program control is transferred outside try block. Thus, the line "This line will not be executed" is never parsed by the compiler. The exception thrown is handled in catch block. Once the exception is handled, the program control is continued with the next line in the program i.e., after catch block. Thus, the line "After exception is handled" is printed.

Multiple catch blocks

A try block can be followed by multiple catch blocks. It means we can have any number of catch blocks after a single try block. If an exception occurs in the guarded code (try block) the exception is passed to the first catch block in the list. If the exception type matches with the first catch block it gets caught, if not the exception is passed down to the next catch block. This continue until the exception is caught or falls through all catches.

Multiple Catch Syntax:

To declare the multiple catch handler, we can use the following syntax.

```
try
 {
  // suspected code
 }
 catch(Exception1 e)
 {
   // handler code
 }
catch(Exception2 e)
{
 // handler code
}
```


Now let's see an example to implement the multiple catch blocks that are used to catch possible exception.

The multiple catch blocks are useful when we are not sure about the type of exception during program execution.

Multiple Catch blocks

In this example, we are trying to fetch integer value of an Integer object. But due to bad input, it throws number format exception.

```
class Demo{
  public static void main(String[] args) {
    try
    {
      Integer in = new Integer("abc");
      in.intValue();
    }
    catch (ArithmeticException e)
    {
      System.out.println("Arithmetic " + e);
    }
    catch (NumberFormatException e)
    {
      System.out.println("Number Format Exception " + e);
    }
  }
}
```

Output

Number Format Exception java.lang.NumberFormatException: For input string: "abc"

Explanation

In the above example, we used multiple catch blocks and based on the type of exception second catch block is executed.

Multiple Exception

Let's understand the use of multiple catch handler by one more example, here we are using three catch handlers to catch the exception.

```
public class CatchDemo2{
  public static void main(String[] args)
 {
   try
   {
     int a[]=new int[10];
     System.out.println(a[20]);
   }
   catch(ArithmeticException e)
   {
     System.out.println("Arithmetic Exception --> "+e);
   }
   catch(ArrayIndexOutOfBoundsException e)
   {
     System.out.println("ArrayIndexOutOfBounds Exception --> "+e);
   }
   catch(Exception e)
   {
     System.out.println(e);
   }
  }
}
```

At a time, only one exception is processed and only one respective catch block is executed.

Example for Unreachable Catch block

While using multiple catch statements, it is important to remember that sub classes of class Exception inside catch must come before any of their super classes otherwise it will lead to compile time error. This is because in Java, if any code is unreachable, then it gives compile time error.

```
class Excep
{
  public static void main(String[] args)
  {
    try
    {
      int arr[]={1,2};
      arr[2]=3/0;
    }
    catch(Exception e)    //This block handles all Exception
    {
      System.out.println("Generic exception");
    }
    catch(ArrayIndexOutOfBoundsException e)    //This block is unreachable
    {
      System.out.println("array index out of bound exception");
    }
  }
}
```

Nested try statement

try statement can be nested inside another block of try. Nested try block is used when a part of a block may cause one error while entire block may cause another error. In case if inner try block does not have a catch handler for a particular exception, then the outer try catch block is checked for match.

```
class Excep{
  public static void main(String[] args) {
    try {
      int arr[]={5,0,1,2};
      try {
        int x = arr[3]/arr[1];
      }
      catch(ArithmeticException ae)
      {
        System.out.println("divide by zero");
      }
      arr[4]=3;
    }
    catch(ArrayIndexOutOfBoundsException e)
    {
      System.out.println("array index out of bound exception");
    }
  }
}
```

Output

divide by zero

array index out of bound exception.

Important points to Remember

1) If you do not explicitly use the try catch blocks in your program, java will provide a default exception handler, which will print the exception details on the terminal, whenever exception occurs.

2) Super class Throwable overrides toString() method, to display error message in form of string.

3) While using multiple catch blocks, always make sure that sub-classes of Exception class come before any of their super classes. Else you will get compile time error.

4) In nested try catch, the inner try block uses its own catch block as well as catch block of the outer try, if required.

5) Only the object of Throwable class or its subclasses can be thrown.