Exception Handling

Exception is an event that occurs during the execution of a program the disrupts the normal transaction flow.

An Exception can be anything which interrupts the normal flow of the program.

For **example**, following situations can cause an exception – **Opening a non-existing file, Network connection problem, Operands being manipulated are out of prescribed ranges, class file missing** which was supposed to be loaded and so on.

When an exception can occur?

Exception can occur at runtime (known as runtime exceptions) as well as at compile-time (known Compile-time exceptions).

Errors indicate serious problems and abnormal conditions that most applications should not try to handle. Error defines problems that are not expected to be caught under normal circumstances by our program.

For example memory error, hardware error, JVM error etc.

Exceptions are conditions within the code. A developer can handle such conditions and take necessary corrective actions.

Few examples –

- DivideByZero exception
- NullPointerException
- ArithmeticException
- ArrayIndexOutOfBoundsException

Advantages of Exception Handling

- Exception handling allows us to control the normal flow of the program by using exception handling in program.
- It throws an exception whenever a calling method encounters an error providing that the calling method takes care of that error.
- It also gives us the scope of organizing and differentiating between different error types using a separate block of codes. This is done with the help of try-catch blocks.

<u>Types of exceptions</u>

There are two types of exceptions

1)Checked exceptions
2)Unchecked exceptions

Checked exceptions

All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not.

If these exceptions are not handled/declared in the program, it will give compilation error.

Examples of Checked Exceptions :-

ClassNotFoundException
IllegalAccessException

NoSuchFieldException
EOFException etc.

Unchecked Exceptions

Runtime Exceptions are also known as Unchecked Exceptions as the compiler do not check whether the programmer has handled them or not but it’s the duty of the System to handle these exceptions and provide a safe exit.

Examples of Unchecked Exceptions:-

ArithmeticException
ArrayIndexOutOfBoundsException
NullPointerException
NegativeArraySizeException etc.

Exception handling in Java

In java, exception handling is done using five keywords,

1. **try**
2. **catch**
3. **throw**
4. **throws**
5. **finally**

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

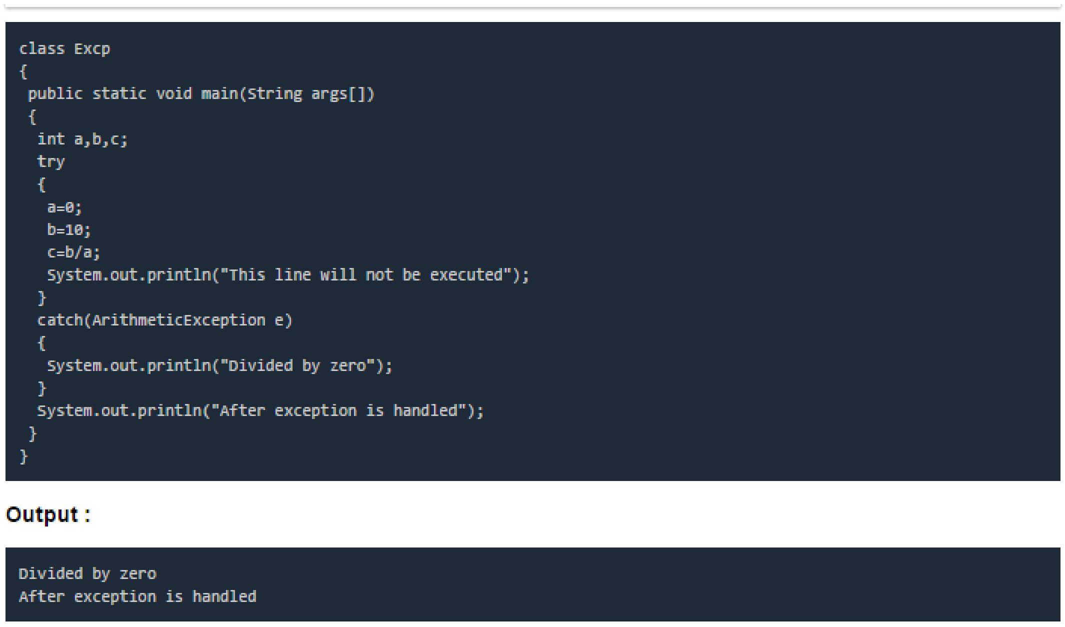
Try Block :

Try is used to guard a block of code in which exception may occur. This block of code is called guarded region.

Catch Block :

A **catch** statement involves declaring the type of exception you are trying to catch. If an exception occurs in guarded code, the catch block that follows the try is checked, if the type of exception that occured is listed in the catch block then the exception is handed over to the catch block which then handles it.

<u>Example using Try and catch</u>



```
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
```

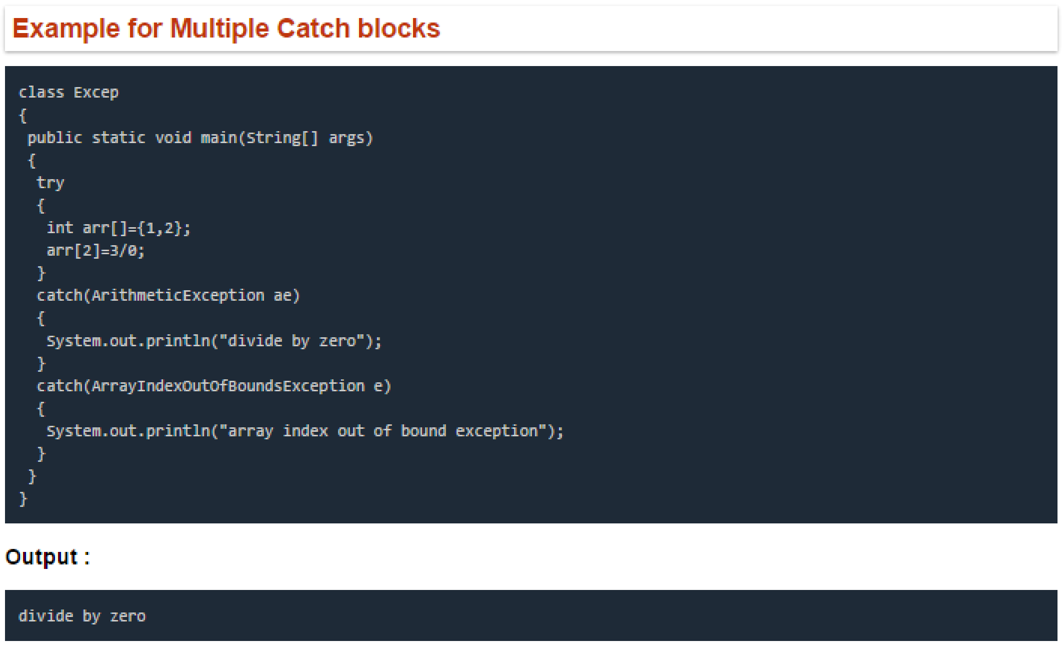
Multiple catch blocks:

A try block can be followed by multiple catch blocks. You can have any number of catch blocks after a single try block.

If an exception occurs in the guarded code the exception is passed to the first catch block in the list. If the exception type of exception, 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.

NOTE : Catch block should be ordered in such a way that specific exception catch block should be first and generic should be at last.(ie child exception catch block should be first then its parent exception catch block)



Example for Multiple Catch blocks

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

Output :

```
divide by zero
```

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()** function, to display error message in form of string.
3. While using multiple catch block, always make sure that exception subclasses comes 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.