# Exception Handling

## ❖ What Is an Exception?

Exception is an abnormal condition.

An exception (or exceptional event) is a problem that arises during the execution of a program. When an **Exception** occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

### Reasons for Exception

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

## ❖ What is exception handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

### Advantage of Exception Handling:

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.

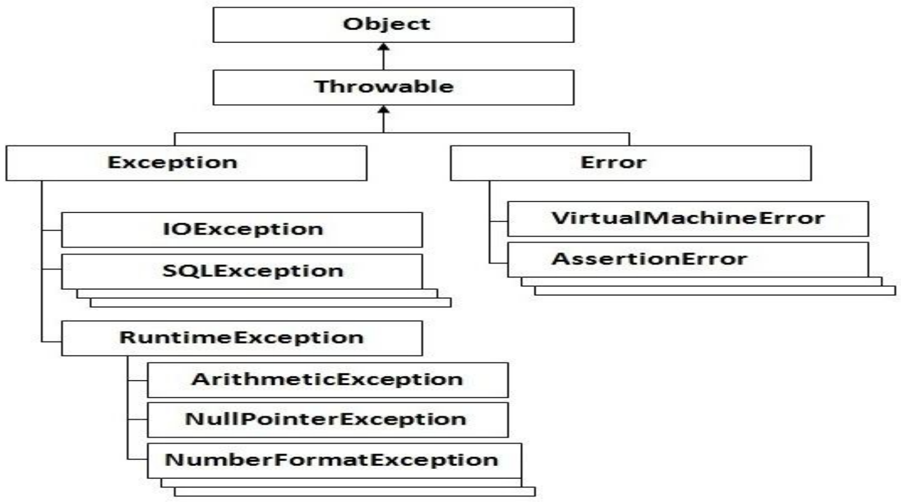
### Difference between error and exception :

**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

***Exception class Hierarchy :***



## ❖ Types of Exception:-

There are three types of exception:

1) Checked Exception
2) Unchecked Exception
3) Error

### 1) Checked Exception:

Exception that can be predicted by the programmer.

***Example :*** File that need to be opened is not found. These types of exceptions must be checked at compile time.

All exceptions are checked exceptions, except for those indicated by Error, RuntimeException, and their subclasses

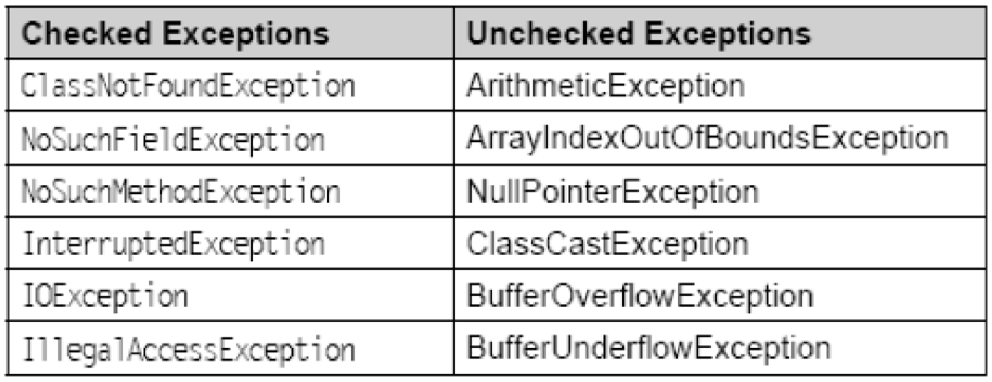
2) **Unchecked Exception :**

Unchecked exceptions are the class that extends RuntimeException. Unchecked exception are ignored at compile time.

***Example :*** ArithmeticException, NullPointerException, Array Index out of Bound exception. Unchecked exceptions are checked at runtime.

3) **Error:**

These are exceptional conditions that are external to the application, and that the application usually cannot predict or recover from them. Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.



| Checked Exceptions | Unchecked Exceptions |
|---|---|
| ClassNotFoundException | ArithmeticException |
| NoSuchFieldException | ArrayIndexOutOfBoundsException |
| NoSuchMethodException | NullPointerException |
| InterruptedException | ClassCastException |
| IOException | BufferOverflowException |
| IllegalAccessException | BufferUnderflowException |

## ❖ Java Exception Handling Mechanisms/Keywords :-

1) try
2) catch
3) finally
4) throw
5) throws

## 1) Java try 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:***

```
Try
{
//code that may throw exception
}
catch(Exception_class_Name ref)
{
 }
```

***Syntax of try-finally block***

```
Try
{
//code that may throw exception
}
Finally
{}
```

## 2) Java catch block :

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.

Problem without exception handling

```
public class Test
{
  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
```

As displayed in the above example, rest of the code is not executed (in such case, rest of the code... statement is not printed).

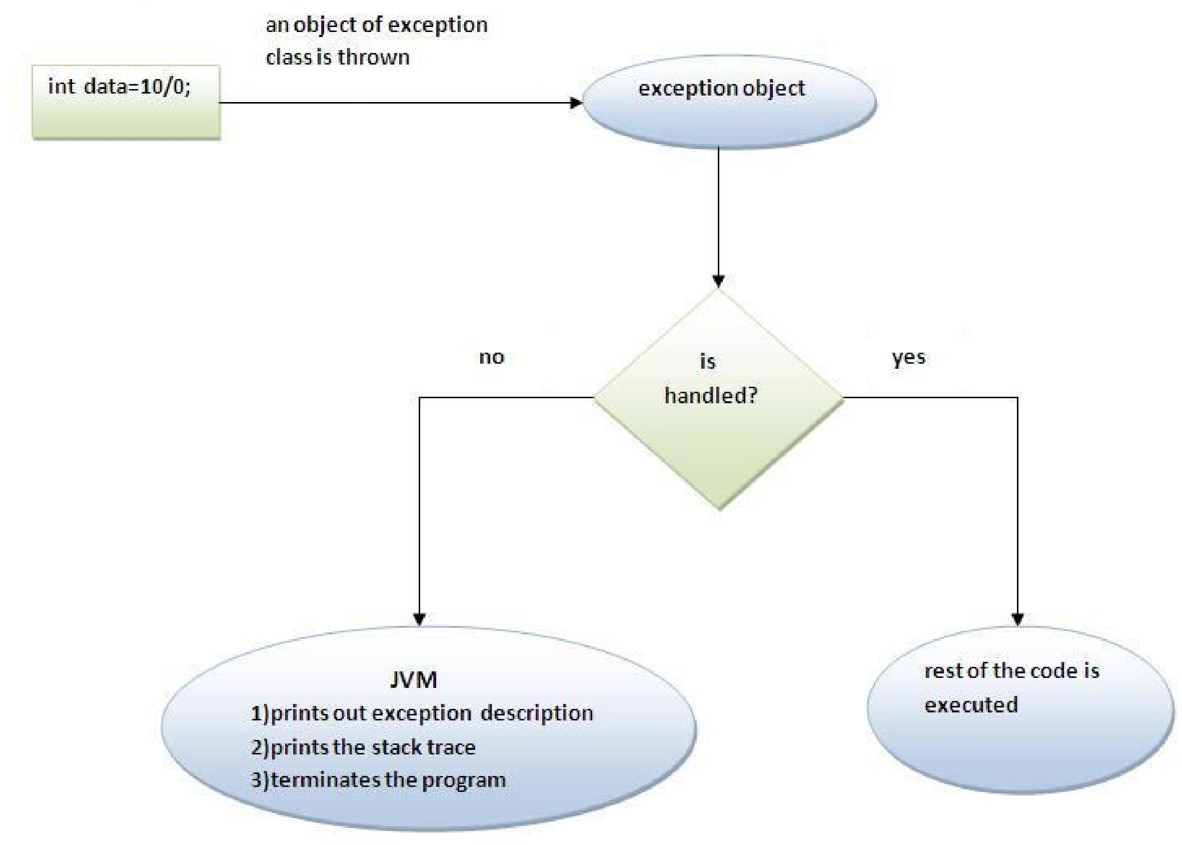
There can be 100 lines of code after exception. So all the code after exception will not be executed.

## Solution by exception handling

```
public class Test
{
  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...");
}
}
```

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

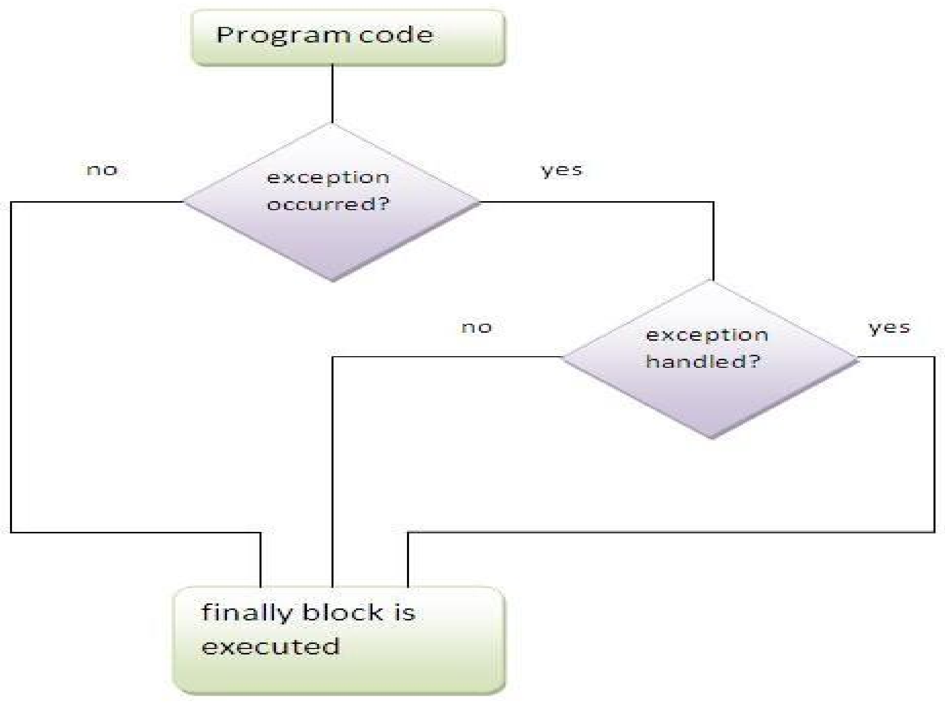
## Internal working of java try-catch block


int data=10/0;
an object of exception class is thrown
exception object
is handled?
no
yes
JVM
1)prints out exception description
2)prints the stack trace
3)terminates the program
rest of the code is executed

## 3) Java finally block :-

**Java finally block** is a block that is used *to execute important code* such as closing connection, stream etc. Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.



Why use java finally

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

***Note :- The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).***

## 4) Java throw keyword :-

The Java throw keyword is used to explicitly throw an exception. We can throw either checked or uncheked exception in java by throws keyword. The throw keyword is mainly used to throw custom (user defined) exception.

The syntax of java throw keyword :

```
throw exceptionobject;
```

**Example :**

```
public class TestThrow1
{
  static void validate(int age)
 {
    if(age<18)
     throw new ArithmeticException("not valid");
    else
     System.out.println("welcome to vote");
  }
  public static void main(String args[]){
     validate(13);
     System.out.println("rest of the code...");
 }
}
```

## 5) Java throws keyword :-

The **Java throws keyword** is used to declare an exception in method signature. It gives information to the programmer that there may occur an exception in a method so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Syntax of java throws :

```
Access specifier  return_type  method_name() throws exception_class_name
{
//method code
}
```

Which exception should be declared

checked exception only

There are two cases:

1. **Case1:** You caught the exception i.e. handle the exception using try/catch.
2. **Case2:** You declare the exception i.e. specifying throws with the method.

**Difference between throw and throws in Java :**

| | **Throw** | **throws** |
|---|---|---|
| 1) | Java throw keyword is used to explicitly throw an exception. | Java throws keyword is used to declare an exception in method signature. |
| 2) | Checked exception cannot be propagated using throw only. | Checked exception can be propagated with throws. |
| 3) | Throw is followed by an exception object. | Throws is followed by class. |
| 4) | Throw is used within the method. | Throws is used with the method signature. |
| 5) | You cannot throw multiple exceptions. | You can declare multiple exceptions e.g. public void method()throws IOException,SQLException. |