**Default Exception Handling**

* Inside a method if any exception occurs the method in which it is raised is responsible to create exception object by including the following information.
  + Name of Exception
  + Description of Exception
  + Location at which exception occurs [Stack trace]
* After creation of exception object method handover that object to the JVM.
* JVM will check whether the method contains any exception handling code or not if the method does not contains the exception handling code then JVM terminates that method abnormally and the remove the corresponding entry from the stack.
* Then JVM identifies the caller method and checks whether caller method contains any exception handling code or not.
* If the caller method does not contain the handling code then JVM terminates that caller method also and remove the corresponding entry from the stack.
* This process will be continuing until main method and if the main method also does not contain the handling code then JVM terminate the main method also abnormally and remove the corresponding entry from the stack.
* Then JVM handovers responsibility of exception handling to Default Exception Handler and it is the part of JVM.
* Default exception handler prints exception information in the following format and terminates the program execution abnormally.
  + Exception in thread “main” [xxx] Name of exception: Description
  + Stack Trace

**Example 1**

**public** **class** Test {

**public** **static** **void** main(String[] args) {

*doStuff*();

}

**public** **static** **void** doStuff() {

*doMoreStudff*();

}

**public** **static** **void** doMoreStudff() {

System.***out***.println(10/0);

}

}

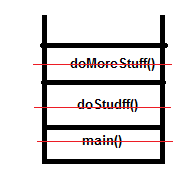
Output:

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

at Test.doMoreStudff(Test.java:11)

at Test.doStuff(Test.java:8)

at Test.main(Test.java:5)



**Example 2**

**public** **class** Test {

**public** **static** **void** main(String[] args) {

*doStuff*();

System.***out***.println(10/0);

}

**public** **static** **void** doStuff() {

*doMoreStudff*();

System.***out***.println("Hi");

}

**public** **static** **void** doMoreStudff() {

System.***out***.println("Hello");

}

}

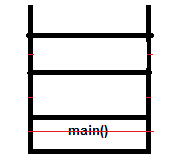
**Output**

Hello

Hi

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

at Test.main(Test.java:6)

****

Note

In a program if at least one method terminates abnormally then the program termination is abnormal termination and if all the method terminated normally then only program termination is normal termination.

**Exception hierarchy**

Throwable class access root for java exception hierarchy

Throwable class defined two child classes.

**Exception and Error**

**Exception**

Most of the time exceptions are caused by our program and these are recoverable.

For example if our programming requirement is to read data from the remote file locating at London at the run time if the remote file is not available then we will get run time exception saying FileNotFoundException.

If FileNotFoundException occurs we can provide the local file and continue the rest of the program normally.

**try** {

//Read data from the remote file locating at London

} **catch** (FileNotFoundException e) {

// Use local file and continue rest of the program normally

}

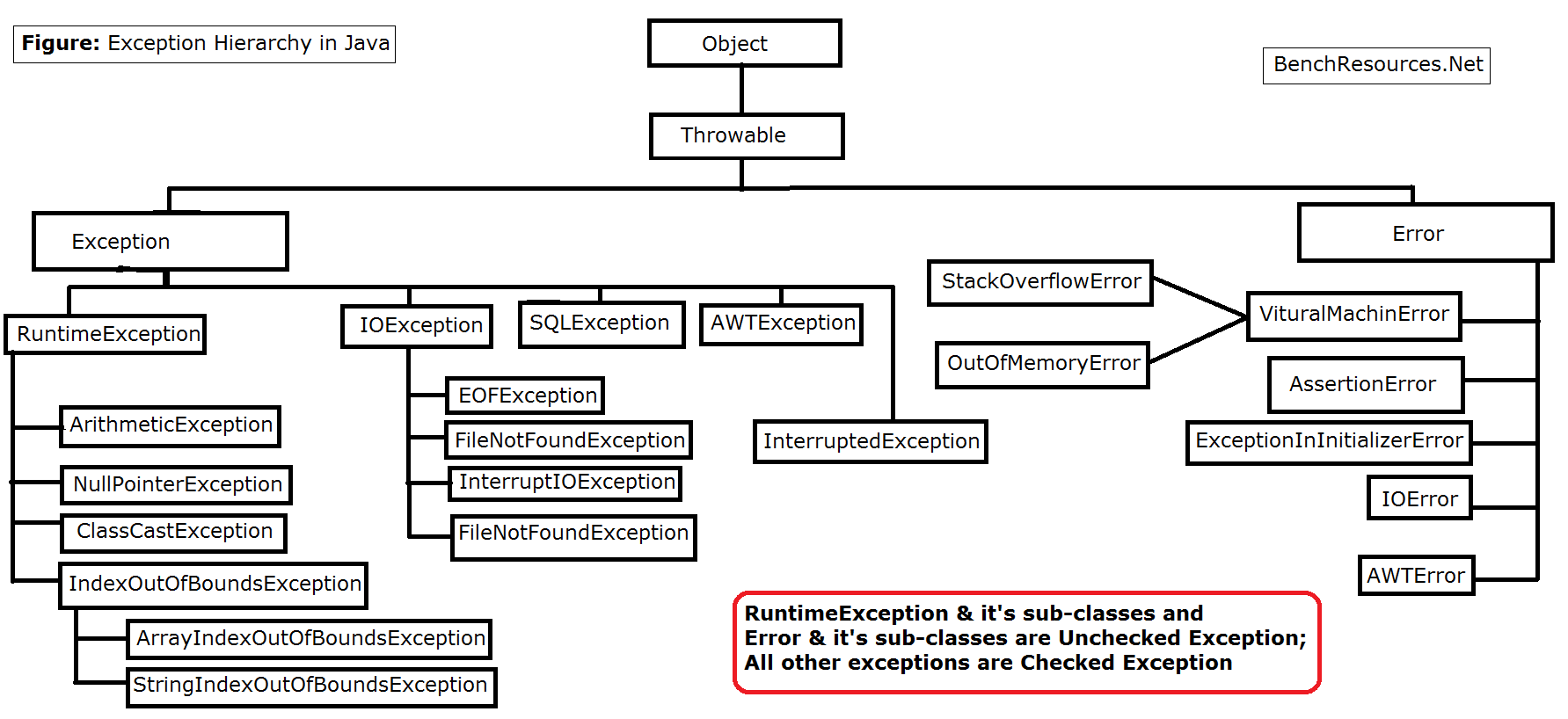
**Error**

Most of the times Errors are not caused by our program and these are due to lack of system resources.

Errors are non – recoverable.

For example if OutOfMemoryError occurs being a programmer we cannot do anything and the program will terminated abnormally.

System admin or server admin are responsible to increase to heap memory.



Final

Final is the modifier applicable for class method and variable.

If class is declared as final we cannot extends that class that is we cannot creates the child class for that class – that is inheritance is not possible for final classes.

If a method is final then we cannot override that method in the child class.

If a variable declared as final then we cannot perform re – assignment for that variable.

Finally:

Finally is the block always associated with try – catch to maintain the cleanup code

**try** {

//Risky code

} **catch** ( Exception e) {

//Handling code

}**finally** {

//Clean up code

}

The specialty of finally block is it will be executed always in respective of whether exception raised or not and whether exception has handled or not handled.

Finalize()

Finalize is the method always invoked by garbage collector just before destroying an object to perform the cleanup activities.

Once finalize method complete immediately garbage collector destroys that object.