**Exception Handling.**

**Introduction**

What is exception?

An unexpected or unwanted event, that disturbs the normal flow of the program is called Exception.

Important terminologies

1. It is highly recommended to handle the exception, because event the normal flow getting disturbed we have to intimate user to execute some other flow.
2. Main purpose of exception handling is graceful termination of the program.
3. Defining alternate ways to continue the rest of the program is called Exception handling.

Run time stack mechanism.

1. For any thread JVM will create run time stack.
2. Each and every method call performed by that thread will be stored in the corresponding stack.
3. Each entry in the stack is called stack frame or activation record.
4. After completing every method call, the corresponding entry from the stack will be removed.
5. After completing all method calls, the stack will be destroyed by JVM just before terminating the thread.

Default exception handling

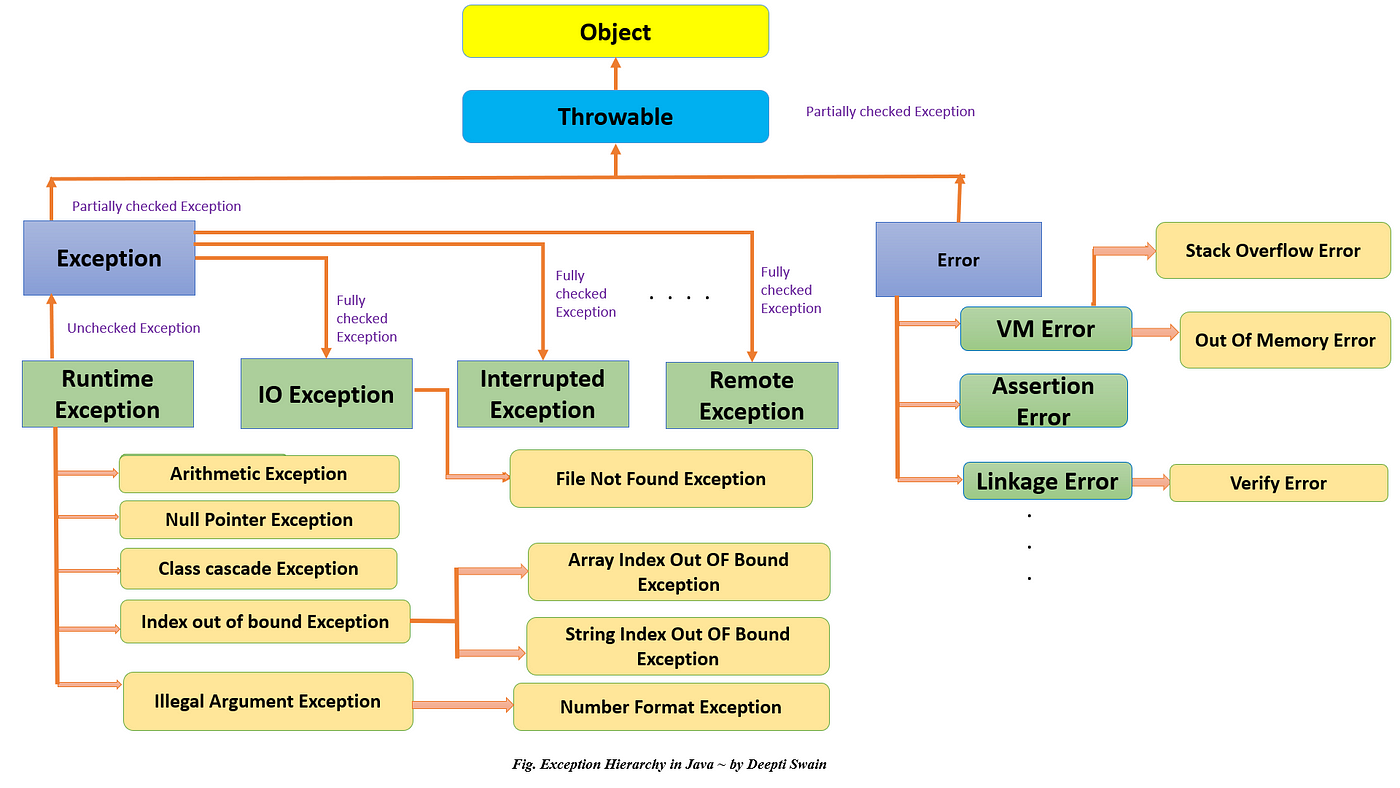
1. Inside method any error occurs, the method in which raised is responsible to create exception object by including
   1. Name of the exception
   2. Description of the exception
   3. Location of the exception(Stack trace)
2. After creating the object the method handover it to the JVM. It will check any exception handling code or not. If method doesn’t contain exception handling code, JVM will terminate the method abnormally and remove the corresponding entry from the stack.
3. Then identifies the caller method and check for the exception handling code. If not, then that caller method will terminate by JVM abnormally. This process continue until Main method. If the main method doesn’t have any handling code then the main method will terminated abnormally by JVM and the exception object handover to default exception handler which is part of JVM.
4. Default exception handler will print the exception in below format
   1. Exception in thread “main” name of the exception : description : Stack trace

**Note**

In a program, if at least one method terminates abnormally, then the program termination is abnormal termination.

**Exception Hierarchy**

1. Throwable is the root class of exception hierarchy.
2. It defines two child classes
   1. Exception
   2. Error
3. Most of the time exception are caused by our program. Error may cause due to lack of system resources. Errors are non – recoverable



Unchecked Exception

1. The exceptions which are not checked by the compiler is called unchecked exception.
2. Whether the exception is checked or unchecked exception it will occurs at run time only.
3. RuntimeException and its child classes, Error and its child classes are unchecked.

Checked exception

1. The exceptions which are checked by the compiler for smooth execution of the program are called checked exceptions
2. If there is chance of raising checked exception, then compulsory we should handle that checked exception either by try, catch or throws keyword, Otherwise we will get compile time error.
3. Except RuntimeException and Error all are checked exceptions.

Fully checked Vs Partially checked

1. A checked exception is said to be fully checked if and only if all its child classes are checked.
   1. IOException
   2. InterruptedException
2. A checked exception is said to be partially checked if and only if some of its child classes are unchecked
   1. Exception
   2. Error

Only possible of partially checked exceptions are Exception and Error.

Behaviour of exceptions

1. IOException – Fully checked
2. RuntimeException - Unchecked
3. InterruptedException – Fully checked
4. Error – Unchecked
5. Throwable – Partially checked.
6. ArithmeticException – Unchecked
7. NullPointerException – Unchecked
8. Exception – partially checked
9. FileNotFoundException – Fully checked

**Customized Exception Handling**

Try{ Risky code

}catch{

Handling code}

Remaining code

1. The code which may raise an exception is called risky code and we have to define It into try block.
2. We have to define handling code in catch block

Methods to print exception information

1. Exception.printStackTrace() – Will print the entire stack trace of the exception
2. Exception.getMessage() – will print (name of the exception: Description)
3. System.out.println(e.toString()) – Will print only message of the exception

Try with multiple catch blocks

1. The way of handling exception is varied from exception to exception.
2. It is highly recommended to write separate catch block to different exceptions
3. If try with multiple catch blocks present then the order of catch block is important. Otherwise we will get compiler error saying exception : xxx has already been caught.

Try{

Risky code

}catch(Exception e){

}catch(ArithmeticException ae){

}

Here the exception handled the Arithmetic exception also. So second catch block will be the unreachable code so compiler error will throw.

**Finaly, Finally, Finalize**

1. Final
   1. It is a modifier applicable for classes, methods, and variables.
   2. If a class declared as a final, we can’t extend it.
   3. If a method declared as final, we can’t override that in the child classes.
   4. If a variable declared as final, the we can’t perform reassignment of the variable.
2. Finally
   1. It is a block always associated with try or catch to maintain the cleanup code.
   2. Finally block always execute whether the program is normal or abnormal termination
3. Finalize
   1. It is method always invoked by garbage collector just before destroying an object to perform cleanup activities. Once the finalize method is completed then the garbage collector will destroy the object.

Note – Finally block is responsible to perform, cleanup related to try, catch blocks. i.e Whatever the resource we opened at try block will be closed inside finally block.

The finalize method is responsible to object. i.e Whatever resource associated with object will be deallocated before destroying the object.

Throw Keyword

1. It is used to handover our created exception object to the JVM by manually.
   1. Throw new Exception(“Exception occured”)
2. Thus the main objective of the Throw keyword. It is very helpful for user defined customized exception objects.
3. After throw statement, we are not allow to write any statement directly. Otherwise we will get compile time error.(Unreachable statement)
4. We can use throw for only Throwable types. The class which are extended the exception or its child classes will be the throwable types. So we can use them in throw.

Throws Keyword

1. In our program there maybe possibilities of raising checked exceptions, then compulsory we should handle that exception. Otherwise compile will throw error. We can handle this compile time error by using
   1. Try, catch
   2. Throws keyword(Delegating exception to caller)
2. Throws keyword required only for checked exceptions. Throws keyword required only to convince the compiler.
3. Usage of throws keyword doesn’t prevent the abnormal termination of the program.
4. It is applicable for throwable objects.

Customized exception

1. Sometimes, to meet programming requirements, we can write our own exceptions, such type of exceptions are called customized or user defined exceptions.
2. Throw keyword is best suitable for user defined or customized exceptions.

Try with resources

Try(BufferReader br = new BufferReader(“input.txt”)){

}catch(IOException e){

}

1. Resource will close automatically while end of the try block and we are not require to write the finally block.
2. We can declare multiple resources into try and those are separated by semicolon.
3. And the resource used in the try then it should be auto closable resources.
4. The classes implements the AutoClosable interface is called auto closable resources.
5. All resource reference variables within the try are implicitly final. We cannot perform reassignment within the try block.

Multi catch blocks

1. We can write single catch block that can handle multiple different kind of exceptions
   1. Catch(AE e1 | NPE e2 | IOE e3){}
2. In multi catch block there should not be any relation between exception types. Either parent to child or same exceptions. Otherwise compiler error will throw
   1. Catch(Exception e | NPE e1){} // Invalid

Exception propagation

When exception raised and if we are not handling that exception then exception object will be propagated to caller method, then the caller is responsible to handle it. This process is caller exception propagation.

ReThrowing Exception

We can use this approach to convert one exception to another.

Try{

SOUT(10/0)

}catch(AE e){

Throw new NullPointerException()

}