Exception handling in java-

Exception is the abnormal condition that occur during execution of program to stop the entire flow of application called as "Exception."

It is highly recommended to handle exceptions. The main objective of exception handling is normal (graceful) termination of the program.

Exception handling doesn't mean repairing an exception. We have to define alternative way to continue rest of the program normally. This way of defining alternative is nothing but exception handling.

Why?

```
package com.velocity;

public class ExceptionDemo {

	public static void main(String[] args) {

		System.out.println("First line");
		System.out.println("Second line");
		System.out.println("Third line");

	}
}
```

Output-
First line
Second line
Third line

Now we add one exception line code as below

```
package com.velocity;

public class ExceptionDemo {

	public static void main(String[] args) {
		System.out.println("First line");
		System.out.println("Second line");
```

```
            System.out.println("Third line");
            int a = 10 / 0;
            System.out.println("Fourth line");
            System.out.println("Fifth line");
      }
}
```

Output-

```
First line
Second line
Third line
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at com.velocity.ExceptionDemo.main(ExceptionDemo.java:10)
```

In this example, two statement are not executed, if you want to execute that two statements then how to do it in java?
Then you should go for exception handling.

```
package com.velocity;

public class ExceptionDemo {

      public static void main(String[] args) {

            System.out.println("First line");
            System.out.println("Second line");
            System.out.println("Third line");

            try {
                  int a = 10 / 0;
            }
            catch(Exception e) {
                  System.out.println(e);
            }

            System.out.println("Fourth line");
            System.out.println("Fifth line");
      }
}
```

Output-
First line
Second line
Third line
java.lang.ArithmeticException: / by zero
Fourth line
Fifth line

Here two statements are executed i.e fourth line and fifth line, we have achieved this by using try and catch block.

Default Exception Handler :

1. If any exception occurs inside any method then that method is responsible to create Exception object with the following information.
    a. Name of the exception.
    b. Description of the exception.
    c. Location of the exception.(StackTrace)
2. After creating that Exception object, this method will handovers that object to the JVM.
3. JVM checks whether the method contains any exception handling code or not. If method won't contain any handling code then JVM terminates that method abnormally and removes corresponding entry form the stack.
4. JVM identifies the caller method and checks whether the caller method contain any handling code or not. If the caller method also does not contain handling code then JVM terminates that caller method also abnormally and removes corresponding entry from the stack.
5. Now this process will be continued until main() method and if the main() method also doesn't contain any exception handling code then JVM terminates main() method also and removes corresponding entry from the stack.
6. Then JVM handovers the responsibility of exception handling to the default exception handler.
7. Default exception handler just print exception information to the console in the following format and terminates the program abnormally.
Exception in thread "xxx(main)" Name of exception: description
Location of exception (stack trace)
8. Default Exception Handler uses printStackTrace() method by default.

Syntax :

```
try {
        //risky code
}
catch (x e){
        //Alternate code
}
finally{
        //clean up code
}
```

Note:

1) In our program the code which may raise exception is called risky code, we have to place risky code inside try block and the corresponding handling code inside catch block.
2) In try block, the line at which exception is raised, there after, rest of the code in try block is not executed. Hence always write only risky code in try block. Size of try block should be as minimum as possible.
3) If any exception occurs outside of try block then it will lead to abnormal termination of program. Exceptions can even occur in catch and finally block also.
4) X e can be same type or parent of exception being raised.

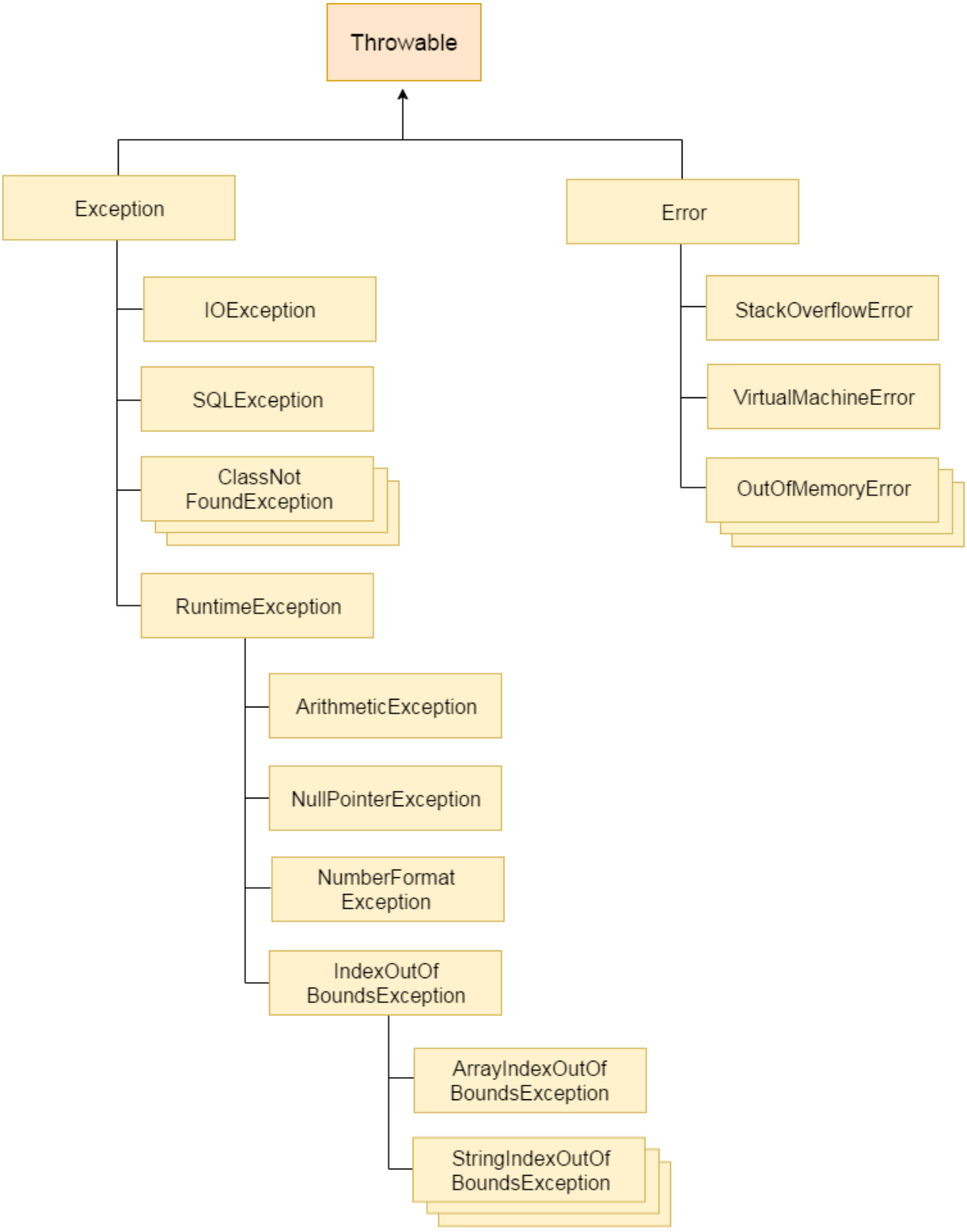
Exception hierarchy-

Throwable-

- In the above given Hierarchy Throwable is a class which is at the top of the exception hierarchy, from which all exception classes are derived.
- It is the super class of all Exceptions in Java.
- Both Exception and Errors are java classes which are derived from the Throwable class.

Exception –

- Most of the cases exceptions are caused by our program and these are recoverable.
- This is the subclass of throwable class.
- For ex. ClassNotFound Exception, FileNotFound Exception, etc.


Throwable
Exception
Error
IOException
SQLException
ClassNot FoundException
RuntimeException
StackOverflowError
VirtualMachineError
OutOfMemoryError
ArithmeticException
NullPointerException
NumberFormat Exception
IndexOutOf BoundsException
ArrayIndexOutOf BoundsException
StringIndexOutOf BoundsException

Types of Exceptions: Checked and Unchecked Exceptions

1) Checked Exception
The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions
e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception
The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

Error—

- Error is subclass of throwable class.
- Errors are mostly the abnormal conditions.
- Error does not occur because of the programmer's mistakes, but when system is not working properly or a resource is not allocated properly.
- Memory out of bound exception, stack overflow etc., are examples of Error.
- Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Note :
Whether it is checked or unchecked exception, all exceptions always occur at run time only during program execution.
In checked exception, compiler just check if there is a possibility of exception being raised or not.
Types of checked exceptions:

1) Fully checked :
   A checked exception is said to be fully checked if and only if all its child classes are also checked.
   For ex. Interrupted Exception, IOException
2) Partially checked :
   A checked exception is said to be partially checked if and only if some of its child classes are unchecked**.**
   For ex**.** Throwable and Exception

Possible way to write try catch block

1.
```
try {
 //not allowed,try without catch/finally
}
```
2.
```
try {
//allowed
}
catch(Exception e){
}
```
3.
```
try {
//allowed
}
finally{
}
```
4.
```
try{
 //allowed
}
catch (Exception e){
}
finally{
}
```

5.
```
try {
 //allowed
}
catch(ArithmaticException e1){
 }
Catch(Exception e){
 }
```

6.
```
try {
//not allowed
}
catch(Exception e){
 }
catch(ArithmaticException e1){
}
```

Not allowed – Reason: The bigger exception cannot be in the
// first catch because it will accommodate all exceptions and there
// will be no chance to reach the second catch of NullpointerException

7.
```
try{
}
catch(Exception e) {
}
catch(NullPointerException npe) {
}
```

8.
```
try {
// not allowed. In between try-catch-finally we can't take any other
statement
}
catch(ArithmaticException e){
}
System.out.println();
catch(Exception e1){
}
```
Note: Curely braces are complusary in try-catch-finally.

finally-

- The finally block is used when an important part of the code needs to be executed. It is always executed whether or not the exceptions are handled.

- It is not recommended to take clean up code inside try block because there is no guarantee for the execution of every statement inside a try.
- It is not recommended to place clean up code inside catch block because if there is no exception then catch block won't be executed.
- We require some place to maintain clean up code which should be executed always irrespective of whether exception raised or not raised and whether handled or not handled. Such type of best place is nothing but finally block.
- Hence the main objective of finally block is to maintain cleanup code.
- Finally block will always get executed until we shut down JVM. To shut down JVM in java we call System.exit (). If you write this in try block then this is the only case when finally block will not be executed.
- Normally, finally block contains the code to release resources like DB connections, IO streams etc.

Flow in try-catch-finally :

```
try{
        statement1;
        statement2;
        statement3;
}
catch(X e) {
        statement4;
}
finally{
        Statement5;
}

Statement6;
```

Case 1:
If there is no exception in any statement,
Output : statement1->statement2->statement3->statement5->statement6 -> normal termination of program
Case 2:
If there exception in statement 2, and corresponding catch block handles/matches the exception then

Output : statement1->statement4->statement5->statement6->normal termination of program

Case 3:
If there exception in statement 2, and corresponding catch block doesn't handles/matches the exception then
Output : statement1->statement5->abnormal termination
Case 4:
If there is exception in statement 5 :
Output : statement1->statement2->statement3->abnormal termination of program
Case 5:
If exception in statement2 and catch block handles/matches the exception but again exception in statement4 then
Output : statement1->statement5->abnormal termination

Various ways to print the exception message:
Throwable class has following methods to print the exception message.

Method	**Description**
getMessage()	This method returns only description of the exception. **Description :**
toString()	This method prints exception information in the following format. **Name of exception : Description**
printStackTrace()	This method prints exception information in the following format. **Name of the exception: description of exception** **Stack trace**