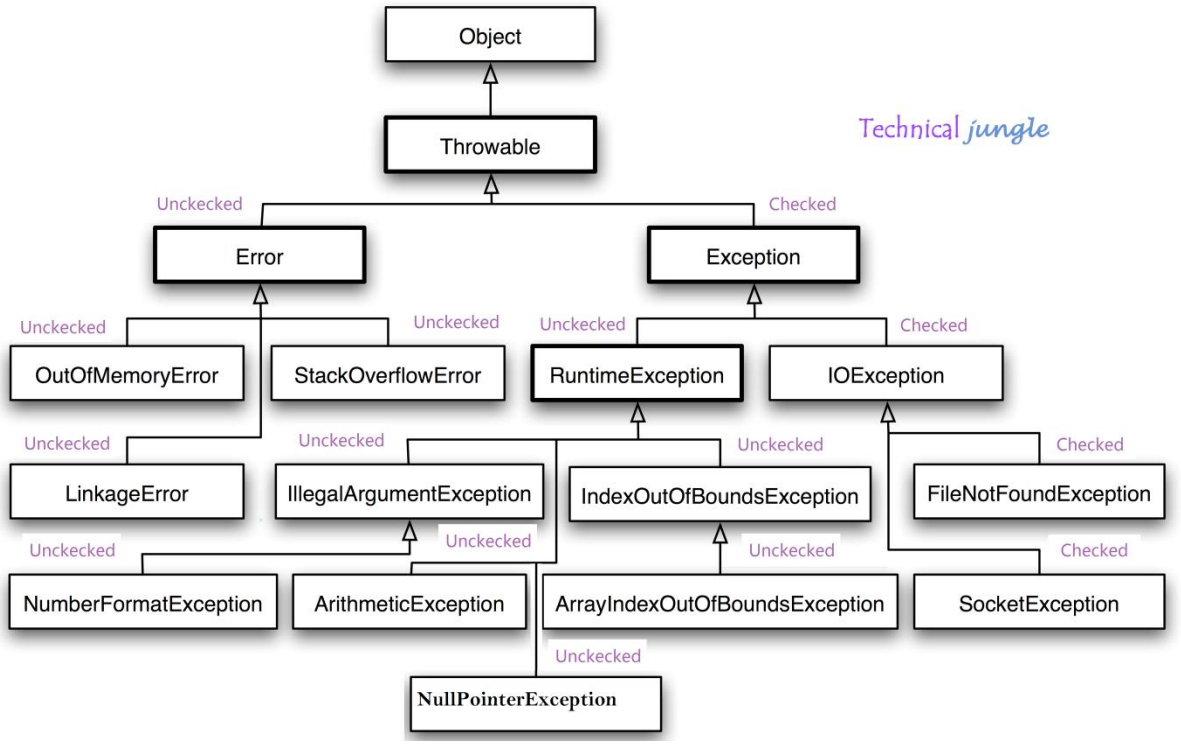
Exception Hierarchy

The possible exceptions in a Java program are organized in a hierarchy of exception classes. The **Throwable class**, which is an immediate **subclass of Object**, is at the root of the exception hierarchy. **Throwable has two immediate subclasses: Exception and Error.**



All of the subclasses of Exception represent exceptional conditions that a normal Java program may want to handle. Many of the standard exceptions are also subclasses of Runtime Exception. **Runtime exceptions represent runtime conditions that can generally occur in any Java method, so a method is not required to declare that it throws any of the runtime exceptions**. However, if a method can throw any of the other standard exceptions, it must declare them in its throws clause.

A Java program should try to handle all of the standard exception classes, since they represent routine abnormal conditions that should be anticipated and caught to prevent program termination.

All **exception classes are subtypes of the java.lang.Exception** class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class.

Errors are abnormal conditions that happen in case of severe failures, these are not handled by the Java programs. Errors are generated to indicate errors generated by the runtime environment. Example: JVM is out of memory. Normally, programs cannot recover from errors.

StackOverflowError:

When a function call is invoked by a Java application, a **stack frame** is allocated on the call stack. The stack frame contains the parameters of the invoked method, its local parameters, and the return address of the method. The return address denotes the execution point from which, the program execution shall continue after the invoked method returns. If there is no space for a new stack frame then, the StackOverflowError is thrown by the Java Virtual Machine (JVM).

The most common case that can possibly exhaust a Java application's stack is *recursion*. In recursion, a method invokes itself during its execution. Recursion is considered as a powerful general-purpose programming technique, but must be used with caution, in order for the StackOverflowError to be avoided.

OutOfMemoryError:

Usually, this error is thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.

OutOfMemoryError usually means that you're doing something wrong, either holding onto objects too long, or trying to process too much data at a time. Sometimes, it indicates a problem that's out of your control, such as a third-party library that caches strings, or an application server that doesn't clean up after deploys. And sometimes, it has nothing to do with objects on the heap.

Similarity between Exception and Error

- Error and Exception both occur during runtime.
- In both cases Abrupt termination happens.

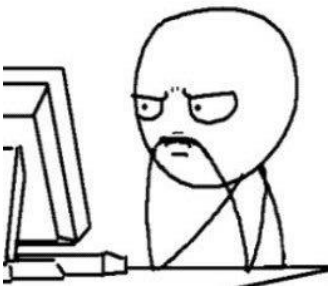
Now let's see Difference

Exception	Error
1) Exceptions can be recovered	1) Errors cannot be recovered.
2) Exceptions can be classified in to two types: a) Checked Exception b) Unchecked Exception	2) There is no such classification for errors. Errors are always unchecked.
3) In case of checked Exceptions compiler will have knowledge of checked exceptions and force to keep try catch blocks.	3) In case of Errors compiler won't have knowledge of errors.

Different ways to write try-catch-finally

Before looking at the valid combinations let us see the invalid combinations..

- **If you are trying to write only try, only catch, or only finally then it is not valid.** Let's think logically. Without knowing if certain statements will give exception, how will catch block know what to catch. And if anything is in try block which might throw exception then in the absence of catch block who will catch it??



Valid combinations of try-catch-finally

1st possibility:

```
try
{
    //some statements
}
catch ()
{
    //some statements
}
```

2nd possibility:

```
try
{
    //some statements
}
catch ()
{
    //some statements
}
catch ()
{
    //some statements
}
```

3rd possibility:

```
try
{
    //some statements
}
catch ()
{
    //some statements
}
finally
{
    //some statements
}
```

4th possibility:

```
try
{
    try
    {
        //some statements
    }
    catch ()
    {
        //some statements
    }
    finally
    {
        //some statements
    }
}
catch ()
{
    //some statements
}
finally
{
    //some statements
}
```

5th posiibility:

```
try
{
    //some statements
}
catch ()
{
    try
    {
        //some statements
    }
    catch ()
    {
        //some statements
    }
    finally
    {
        //some statements
    }
}
finally
{
    //some statements
}
```

6th possibility:

```
try
{
    //some statements
}
catch ()
{
    //some statements
}
finally
{
    try
    {
        //some statements
    }
    catch ()
    {
        //some statements
    }
    finally
    {
        //some statements
    }
}
```

7th possibility:

```
try
{
    //some statements
}
finally
{
    //some statements
}
```

Rules for method overriding, considering the Exceptions

- **If the parent class method throws an exception then the child class overridden method can either throw the same exception or not throw any exception at all.**
- **If the parent class method throws an exception, then the child class overridden method can throw a different exception provided is-a relationship exists between the exceptions.**
- **If the parent class method throws an exception then the child class overridden method can throw a different exception even though is-a relationship does not exist provided both the exceptions are RuntimeException.**