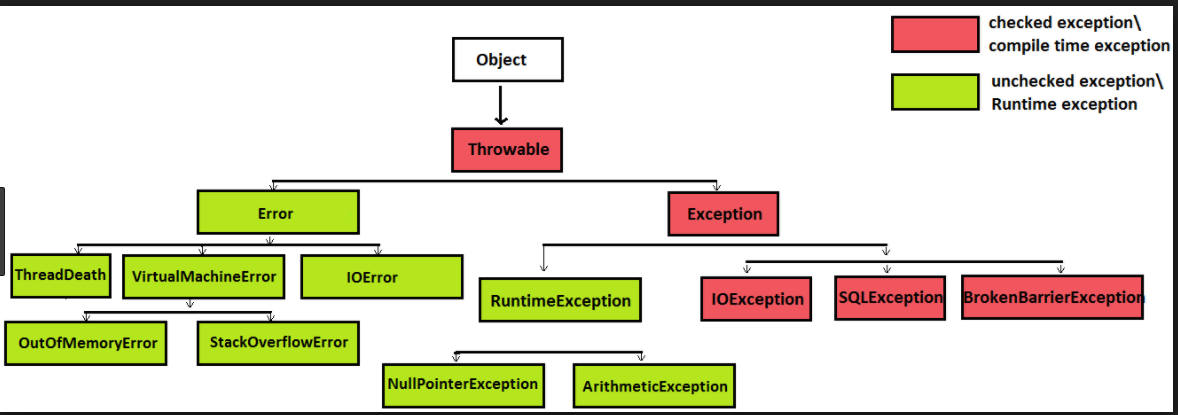
1. **Exception**: It’s an abnormal condition happens in a code sequence at run time.
   1. Once an exception occur an object represented the exception is thrown into method that caused it. The method can catch and process the exception or pass it upstream in call stack.
   2. If exception is not handled in user program then it would be ultimately handled by java default exception handler which prints reason of exception and a stack trace. At this point program terminates.
2. **Keywords**: try, catch, throw, throws, finally
   1. Any code that must need to run after try block must be in finally.
3. **Exception Hierarchy**:
   1. Exception represents exception conditions that user should handle in their program.
   2. Error are not expected to be caught by user as it represents catastrophic failure e.g., StackOverflowError



1. **Try-catch-finally**:
   1. If a code block throws multiple exception we can use multiple catch block after try block.
      1. Try statements can be nested. Each time a try statement is entered the context of that exception is pushed on to a stack. This stack unwinds on exception if exception can’t be handled in inner try block.
      2. A try block requires at least one catch/finally.
      3. Exception subclass must come before superclass in sequence because in java unreachable code is creates compilation error.
      4. ‘finally’ code block runs after try/catch.
      5. We can club multiple exception that are in same level in inheritance hierarchy in a multi-catch statement
      6. Each parameter in multi-catch statement is effectively final.
2. **Throw**: JRE can throw any object type of Throwable or its subclasses.
   1. **void** m4() **throws** ClassNotFoundException
   2. The exception class in throws must be of same/super type of the checked exception thrown.
   3. **Precise rethrow**: you can list specific exceptions in the throws clause of your method, even if they are not explicitly handled by a catch block.
      1. The compiler will look at a method and figure out what the most specific exception is that might get thrown, as opposed to simply looking at the Exception which is the most general. The Java 7 compiler will then allow you to list the specialized exceptions that might get thrown from the method.
      2. Avoid is reassigning the exception in the catch block otherwise we need to mention precisely in the throws the generic exception we are catching.
3. **Throws**: If a method is raises a checked exception which it doesn’t handle then it must say so in its method signature.
4. **Try-With-Resources**: In Java SE 7 or 8 versions, we should follow these rules to use Try-With-Resources statement for Automatic Resource Management(ARM)
   1. Any Resource (which Pre-defined Java API class or User Defined class) must implement java.lang.AutoCloseable.
   2. Resource object must refer either final or effectively final variable
   3. If Resource is already declared outside the Try-With-Resources Statement, we should re-refer with local variable (As shown in the above Example-1 code)
   4. That newly created local variable is valid to use within Try-With-Resources Statement.
   5. Java SE 9, if we have a resource which is already declared outside the Try-With-Resource Statement as final or effectively final, then we do NOT need to declare a local variable.
5. **Custom exception**: To create custom exception we need to extend Exception class.
6. **Chained Exception**: Through chained exception we can associate one exception to other. A ArithmaticException can occur due to an IOException. It would be useful for user to let them know the original cause.
7. **Exception**:
   1. **Constructor**: Exception([String msg, Throwable cause]),
8. **Throwable**:
   1. **Methods**: getCause()[returns underlying exception or null], initCause(throwable) [initCause can’t be set more than once and if already set through constructor it won’t work. ]