public class Exception extends throwable

All exception and errors types are sub classes of class **Throwable**,

**Exception:**

It is an event that will occur during execution of a program which can interrupt the flow of the program.

**The Three Kinds of Exceptions:**

1.Checked exception

- also called compile time exception. We know in advance(that is during compile time) that this exception may occur and interrupt the execution of the application.

e.g. IOException, SQLException

2.Error

- these are the exceptional conditions that are external to the application. Because of hardware or system malfunction, the application will throw error.

e.g. OutOfMemoryError, VirtualMachineError, AssertionError

3.runtime exception -

- these are the exceptional conditions that are internal to the application. Because of programming bugs, such as logic errors or improper use of an API, the program execution will throw runtime exception.

e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException

both error and runtime exception are called **unchecked** exception.

For exception handling we are using three blocks:

**try** - the code that might throw an exception should be enclosed within try block

**catch** - it is exception handler. one try block can have more than one exception handler. The argument type declares the type of exception that the handler can handle and must be the name of a class that inherits from the Throwable class.

In Java SE 7 and later, a single catch block can handle more than one type of exception.

catch (IOException|SQLException ex) {

logger.log(ex);

throw ex;

}

If a catch block handles more than one exception type, then the catch parameter is implicitly final. In this example, the catch parameter ex is final and therefore you cannot assign any values to it within the catch block.

**finally**- this is mainly used for code cleanup(releases resources that are no longer needed). The finally block always executes when the try block exits. finally block is excecuted even if an unexpected exception occurs.

**the place where the finally block will not be excuted is:**

1.if JVM exits while the try or catch code is being executed, then the finally block may not execute.

2.if the thread executing the try or catch code is interrupted or killed, the finally block may not execute even though the application as a whole continues.

**The try-with-resources Statement:**

It is useful when we r using resources in our try block. The try-with-resources statement is a try statement that declares one or more resources. It ensures that each resource is closed at the end of the statement

Any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable, can be used as a resource.

static String readFirstLineFromFile(String path) throws IOException {

try (BufferedReader br =

new BufferedReader(new FileReader(path))) {

return br.readLine();

}

}

The class BufferedReader, in Java SE 7 and later, implements the interface java.lang.AutoCloseable. Because the BufferedReader instance is declared in a try-with-resource statement, it will be closed regardless of whether the try statement completes normally or abruptly

Note: A try-with-resources statement can have catch and finally blocks just like an ordinary try statement. In a try-with-resources statement, any catch or finally block is run after the resources declared have been closed.

Custom Exception

public class InvalidCountryException extends Exception{

InvalidCountryException(String message)

{

  super(message);

}

}