**Throwable**

- implements Serializable

- The java.lang.Throwable class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement.

**Error**

- An Error "indicates serious problems that a reasonable application should not try to catch."

- Errors are also unchecked exception & the programmer is not required to do anything with these. Error is thrown by JVM in a scenario which is fatal, and most often, recovery from an Error is not possible & the program should be allowed to terminate. In fact it is a bad idea to use a try-catch clause for Errors.

**Exception**

- An Exception "indicates conditions that a reasonable application might want to catch."

- Exception designed to represent errors that expected and can be handled by a programmer without terminating program execution.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Checked Exceptions**  - try catch block is MANDATORY  - Checked exceptions are generally those from which a program can recover & it might be a good idea to recover from such exceptions programmatically. Examples include FileNotFoundException, ParseException, etc. A programmer is expected to check for these exceptions by using the try-catch block or throw it back to the caller   |  |  | | --- | --- | | Exception | Reason for Exception | | ClassNotFound Exception | This Exception occurs when Java run-time system fail to find the specified class mentioned in the program | | ParseException | Signals that an error has been reached unexpectedly while parsing. | | SQL Exception | An exception that provides information on a database access error or other errors. | | No Such Method Exception | This Exception occurs when the method you call does not exist in class | | FileNotFound Exception | This exception will be thrown when a file with the specified pathname does not exist. It will also be thrown if the file does exist but for some reason is inaccessible, for example when an attempt is made to open a read-only file for writing. | | **Unchecked Exceptions**  - try catch block is OPTIONAL  - These are those exceptions that might not happen if everything is in order, but they do occur. Examples include ArrayIndexOutOfBoundException, ClassCastException, etc. Many applications will use try-catch or throws clause for RuntimeExceptions & their subclasses but from the language perspective it is not required to do so. Do note that recovery from a RuntimeException is generally possible but the guys who designed the class/exception deemed it unnecessary for the end programmer to check for such exceptions.  - **Runtime Exception** is usually caused by bad programming or invalid users input (but this one should be checked or validated by the programmer)   |  |  | | --- | --- | | Exception | Reason for Exception | | Arithmetic Exception | This Exception occurs when you divide a number by zero. | | ClassCast Exception | Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.  Object x = new Integer(0);  System.out.println((String)x); | | Array Index Out Of Bounds Exception | If we request for an index that is either negative, or greater than or equal to the size of the array, this exception is thrown. | | Null Pointer Exception | In Java, a special null value can be assigned to an object reference. NullPointerException is thrown when an application attempts to use an object reference that has the null value | | Number Format Exception | It occurs when you try to do something like convert a String to a numeric value, like an int, float, double, long, etc. | |

**Compile Time and Runtime Errors**

**Compiler errors** are due to inaccuracies in code or syntax error, where the compiler throws an error to alert you to something which will not compile, and therefore cannot be run.

Example:

int num = "this is not an int";

A **run time error** will only occur when the code is actually running. These are the most difficult - and lead to program crashes and bugs in your code which can be hard to track down.

Example:

String helloWorld = "hello";

int willThrowRuntimeError = Convert.ToInt32(helloWorld);

**USER DEFINED EXCEPTION**

Because, the existing exceptions may not be suitable for your situation. By creating your own exception you can make it easier to understand why the exception is thrown and it will stand out better than if you were using an existing exception.

