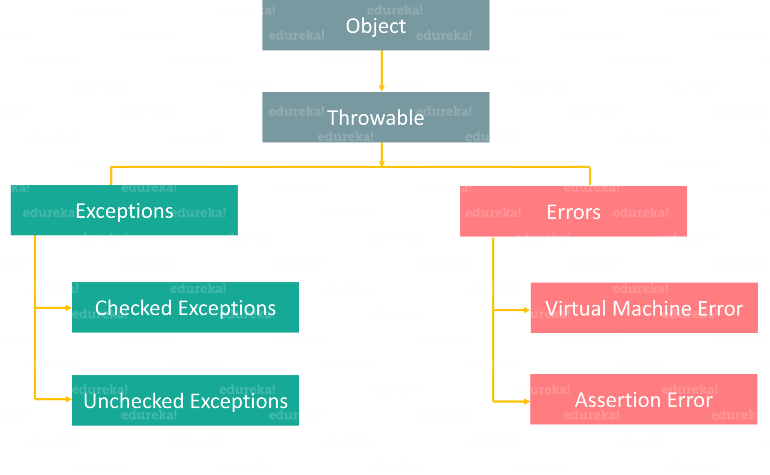
**EXCEPTION**

* An event that disrupts the normal flow of a program.
* Dividing by zero, class not found, etc. can be some reasons for an exception to occur.
* Possible to recover from an exception.
* Checked or unchecked.
* Can happen at compile-time or run-time.
* Caused by the application.

**ERROR**

* Impossible to recover from an error.
* Happens at run-time.
* Unchecked.
* Caused by the environment on which the program runs.

**EXCEPTION HIERARCHY**



* When an exception occurs in a method, the method creates an exception object, which contains the description of it and state of the program, and hands it off to the runtime system.
* The creation of object and its handover is called THROWING AN EXCEPTION.

**CHECKED & UNCHECKED**

* They happen at compile time and are meant to the handled by the programmer.
* They happen at run time and are ignored at the time of compilation.

**EXCEPTION HANDLING METHODS**

**Try**

* The part of code that may arise an exception is written inside a try block.

**Catch**

* If an exception occurs, the catch statement with the necessary handling code is executed.
* It always follows the try block.
* There may exists multiple catch blocks for a single try block.

**Finally**

* It is used to enclose important code which should be executed even if an exception has occurred.
* The code in finally always executes.
* Doesn’t matter if the exception has been handled or not.
* If used, the catch block isn’t necessary as it preserves the normal flow of the program.

**Throw**

* It is used to throw exceptions explicitly.
* Can be used to create a new exception.
* Syntax: **throw new exceptionName();**

**Throws**

* It specifies that a method may throw an exception.
* Syntax: **methodName() throws exceptionName{ methodDefinition }**
* Such methods should be used inside try blocks.

**FILE HANDLING IN JAVA**

* Java uses a concept of streams to execute input and output operations on a file.
* A stream is a sequence of data.
* Can be of two types: Byte stream and Character stream.
* Byte stream comprises of 8-bit data elements.
* Character stream comprises of 16-bit Unicode data elements.
* To import files, **java.io.File,** and to handle IO exceptions **java.io.IOException** packages need to be imported.
* **File fileName=new File (“PATH”);** //creates file object
* Methods: canRead, canWrite, createNewFile, Delete, Exists, getName, getAbsolutePath, Length, List, Mkdir.

**Write**

* To write data, **java.io.FileWriter** package is to be imported.
* Steps:
  + Create FileWriter object.
  + Call the write method or that object // obj.write(“CONTENTTOBEWRITTEN”);
* Read operation can be done via scanner class as in case of console.
* The files should be closed at the end of operation using **fileObject.close();**

**AWT vs SWING**

|  | **AWT** | **Swing** |
| --- | --- | --- |
| 1. | Java AWT is an API to develop GUI applications in Java | Swing is a part of Java Foundation Classes and is used to create various applications. |
| 2. | The components of Java AWT are heavy weighted. | The components of Java Swing are light weighted. |
| 3. | Java AWT has comparatively less functionality as compared to Swing. | Java Swing has more functionality as compared to AWT. |
| 4. | The execution time of AWT is more than Swing. | The execution time of Swing is less than AWT. |
| 5. | The components of Java AWT are platform dependent. | The components of Java Swing are platform independent. |
| 6. | MVC pattern is not supported by AWT. | MVC pattern is supported by Swing. |
| 7. | AWT provides comparatively less powerful components. | Swing provides more powerful components. |