Exception/Error Handling

What are exceptions?

What are exceptions?

- *Exceptions* are events that occur during the execution of programs that disrupt the normal flow of instructions

Why do we do exception handling?

- To separate error handling code from regular code
  - Benefit: Cleaner algorithms, less clutter
- To propagate errors up the call stack
  - Benefit: Nested methods do not have to explicitly catch-and-forward errors (less work, more reliable)
- Grouping of classes and differentiate error types.
  - Benefit: You can group errors by their generalize parent class, **or**

  Differentiate errors by their actual class

Errors vs Exceptions

- As per Java definition:
  - An Error "indicates serious problems that a reasonable application should not try to catch."

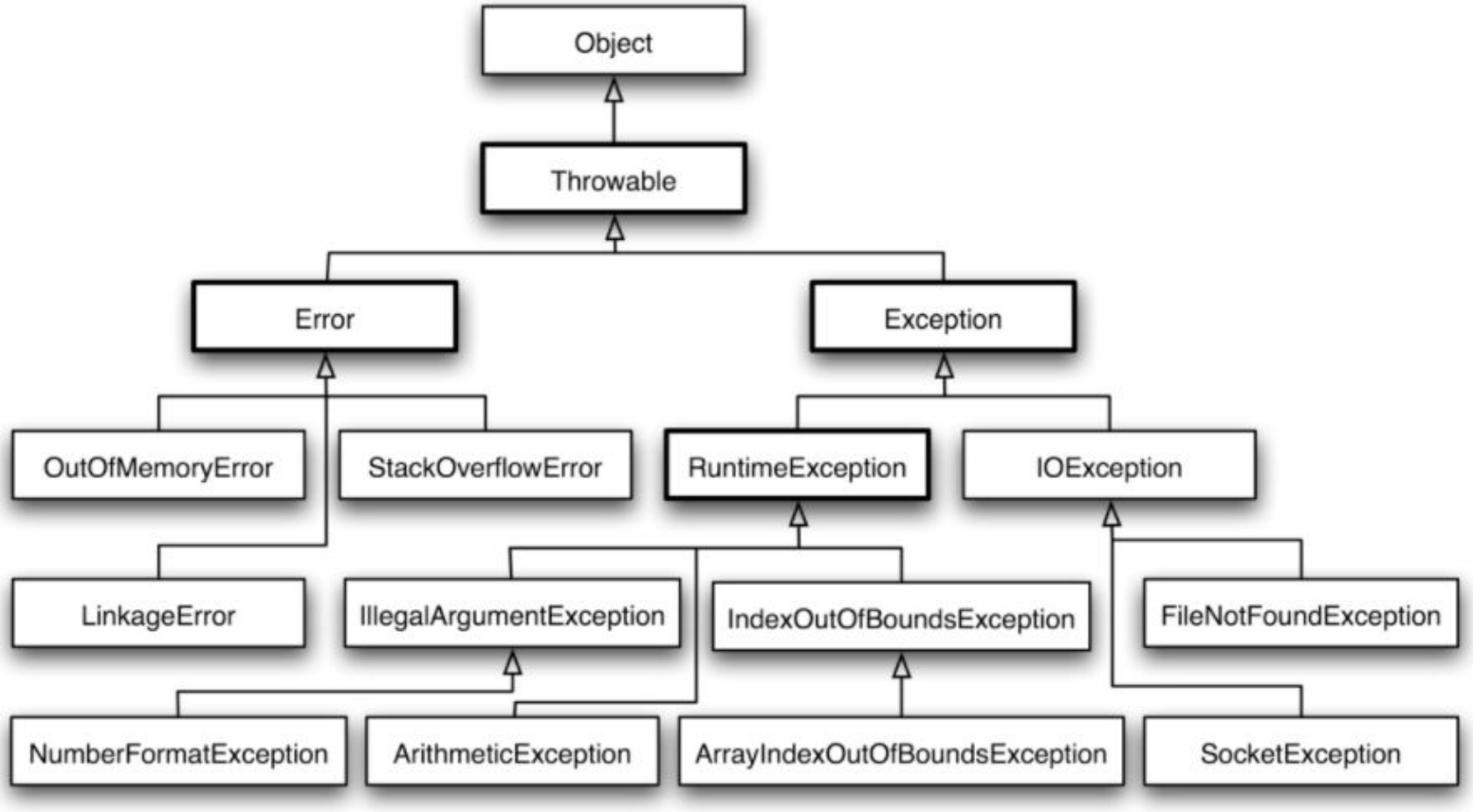
whereas

  - An Exception "indicates conditions that a reasonable application might want to catch."
- Errors are represented by java.lang.Error class which are mainly caused by the environment in which application is running

  Example: **OutOfMemoryError** occurs when JVM runs out of memory
  or **StackOverflowError**occurs when stack overflows
- Whereas exceptions are represented by java.lang.Exception class which are mainly caused by the application itself

Example: **NullPointerException** occurs when an application tries to access null object

Exception Hierarchy



Checked vs Un-checked Exceptions

- **Checked exceptions** are the exceptions that are **checked** at compile time. Enforces caller to catch with an *explicit try/catch block* or to be re-thrown
- **Unchecked exceptions** are the exceptions that are **not checked** at compiled time. These typically represent unrecoverable errors or unforeseen/unplannable exceptions
- In Java,
  - A checked exception is any subclass of Exception (or Exception itself), excluding class RuntimeException and its subclasses
  - Unchecked exceptions are RuntimeException and any of its subclasses. Class Error and its subclasses also are unchecked

Exception handling in Java

```
try {
    //logic which might throw IOException
    // ex: stream.read();

} catch(IOException exe) {
    //handle any IOException specific logic

}
```

Logic enclosed in try block

Catch block to deal with exceptions

NOTE: A try block should have at least one catch block or a finally block.

Handling Multiple Exceptions

```
try {
    //logic which might throw checked or unchecked exceptions

} catch(IOException exe) {
    //handle any IOException specific logic

} catch(<User defined exceptions>) {
    //handler for user specific exceptions

} catch(Exception e) {
    //Handler for all other exceptions thrown
    //by logic in try block
} catch(Throwable t) {

}
```

OPTIONAL, Specific handler for any IO Exceptions thrown in try block. User have the liberty to either rethrow the exception to caller or just ignore it and proceed further etc..

Since **Exception** being super class of all exceptions, this catch block will be the generic exception handler for exceptions having no specific catch block defined

NOTE: The order of catching the exception should be in the INVERTED order of exception hierarchy tree.
Like FileNotFoundException catch block should be defined before IOException catch block.

Displaying the exception information

```
try {

} catch (Exception exe) {
    System.out.println("exe" + exe);
}
```

toString() method overridden in Throwable class will be invoked which will print the call trace as well

Sample exception call trace

```
Exception in thread "main" java.lang.ArithmeticException: Throwing exception from exceptionSampleHandler4 static method
        at com.self.java.course.basic.exceptions.ExceptionsExample.exceptionSampleHandler4(ExceptionsExample.java:26)
        at com.self.java.course.basic.exceptions.ExceptionsExample.exceptionSampleHandler3(ExceptionsExample.java:21)
        at com.self.java.course.basic.exceptions.ExceptionsExample.exceptionSampleHandler2(ExceptionsExample.java:17)
        at com.self.java.course.basic.exceptions.ExceptionsExample.exceptionSampleHandler1(ExceptionsExample.java:13)
        at com.self.java.course.basic.exceptions.ExceptionsExample.main(ExceptionsExample.java:9)
```

Finally block

- The finally block *always* executes when the try block exits even if an unexpected exception occurs
- Finally block allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break.
- Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated

- **Cases when finally block will not be executed:**
  - If the JVM exits while the try or catch code is being executed
  - 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.

Throwing exception

- Rather than handling exceptions locally using try catch block, we can throw the exceptions back to caller for right actions using **throws** keyword

```
    try {
        exceptionSampleHandler3();
    } catch (IOException exe) {
        System.out.println("Handling exception in exceptionSampleHandler2() method");
    }
}

public static void exceptionSampleHandler3() throws IOException {
    exceptionSampleHandler4();
}

public static void exceptionSampleHandler4() throws IOException {

    throw new IOException(
            "Throwing exception from exceptionSampleHandler4 static method");
}
```

```
Handling exception in exceptionSampleHandler2() method
```

User Defined Exceptions

- In java, we can define our own exception classes as per our requirements. These exceptions are called **user defined exceptions in java OR Customized exceptions**.
- User defined exceptions must extend any one of the classes in the hierarchy of exceptions
- Helpful when we anticipate different error scenarios in logic and want to handle differently

```
public class UserDefinedException extends Exception {

    private int errorCode = 0;
    public UserDefinedException(int errorCode, String message) {
        super(message);
        this.errorCode = errorCode;
    }

    public UserDefinedException(int errorCode, String message, Throwable t) {
        super(message, t);
        this.errorCode = errorCode;
    }


}
```