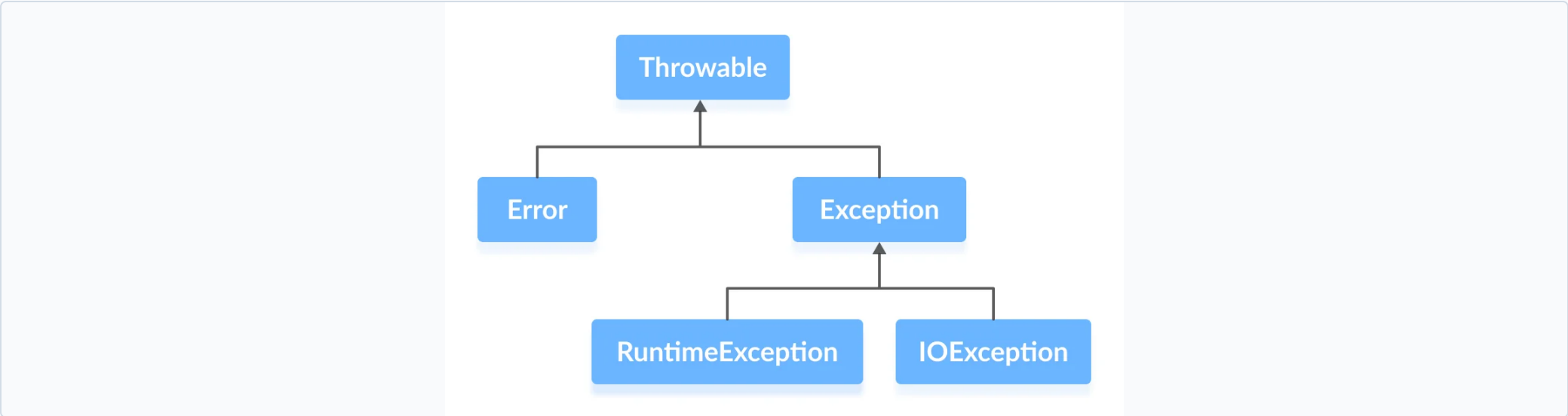# Java Exceptions

An exception is an unexpected event that occurs during program execution. It affects the flow of the program instructions which can cause the program to terminate abnormally.

An exception can occur for many reasons. Some of them are:

- Invalid user input

- Device failure

- Loss of network connection

- Physical limitations (out of disk memory)

- Code errors

- Opening an unavailable file

Java Exception hierarchy

Here is a simplified diagram of the exception hierarchy in Java.

As you can see from the image above, the `Throwable` class is the root class in the hierarchy.

Note that the hierarchy splits into two branches: Error and Exception.

Errors

Errors represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc.

Errors are usually beyond the control of the programmer and we should not try to handle errors.

Exceptions

Exceptions can be caught and handled by the program.

When an exception occurs within a method, it creates an object. This object is called the exception object.

It contains information about the exception such as the name and description of the exception and state of the program when the exception occurred.

We will learn how to handle these exceptions in the next tutorial. In this tutorial, we will now focus on different types of exceptions in Java.

Java Exception Types

The exception hierarchy also has two branches: `RuntimeException` and `IOException`.

1. RuntimeException

A **runtime exception** happens due to a programming error. They are also known as **unchecked exceptions**.

These exceptions are not checked at compile-time but run-time. Some of the common runtime exceptions are:

- Improper use of an API - `IllegalArgumentException`
- Null pointer access (missing the initialization of a variable) - `NullPointerException`
- Out-of-bounds array access - `ArrayIndexOutOfBoundsException`
- Dividing a number by 0 - `ArithmeticException`

You can think about it in this way. "If it is a runtime exception, it is your fault".

The `NullPointerException` would not have occurred if you had checked whether the variable was initialized or not before using it.

An `ArrayIndexOutOfBoundsException` would not have occurred if you tested the array index against the array bounds.

2. IOException

An `IOException` is also known as a **checked exception**. They are checked by the compiler at the compile-time and the programmer is prompted to handle these exceptions.

Some of the examples of checked exceptions are:

- Trying to open a file that doesn't exist results in `FileNotFoundException`
- Trying to read past the end of a file