In Java, errors and exceptions are both types of throwable objects, but they represent different types of problems that can occur during the execution of a program.

**Errors** are usually caused by serious problems that are outside the control of the program, such as running out of memory or a system crash. Errors are represented by the Error class and its subclasses. Some common examples of errors in Java include:

* **OutOfMemoryError:** Thrown when the Java Virtual Machine (JVM) runs out of memory.
* **StackOverflowError:** Thrown when the call stack overflows due to too many method invocations.
* **NoClassDefFoundError:** Thrown when a required class cannot be found.

Since errors are generally caused by problems that cannot be recovered from, it’s usually not appropriate for a program to catch errors. Instead, the best course of action is usually to log the error and exit the program.

**Exceptions**, on the other hand, are used to handle errors that can be recovered from within the program. Exceptions are represented by the **Exception** class and its subclasses.

* **NullPointerException**: Thrown when a null reference is accessed.
* **IllegalArgumentException:** Thrown when an illegal argument is passed to a method.
* **IOException:** Thrown when an I/O operation fails.

Since exceptions can be caught and handled within a program, it’s common to include code to catch and handle exceptions in Java programs. By handling exceptions, you can provide more informative error messages to users and prevent the program from crashing.

**Exception**

* Exception is an event that occurs during the execution of a program that disrupts the normal flow of the program's instructions.
* Exceptions can be recovered by using try-catch block.
* Exceptions are classified into two types: checked exceptions and unchecked exceptions.
* Checked exceptions are the exceptions that are checked at compile time.
* Unchecked exceptions are the exceptions that are not checked at compile time.

**Error**

* Error is a serious problem that occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
* Error cannot be recovered by using try-catch block.
* Errors are classified into two types: fatal errors and non-fatal errors.
* Fatal errors are the errors that terminate the program.
* Non-fatal errors are the errors that do not terminate the program.

**Examples of Exception**

ArithmeticException, ArrayIndexOutOfBoundsException, ClassNotFoundException, FileNotFoundException, IOException, and NullPointerException.

**Examples of Error**

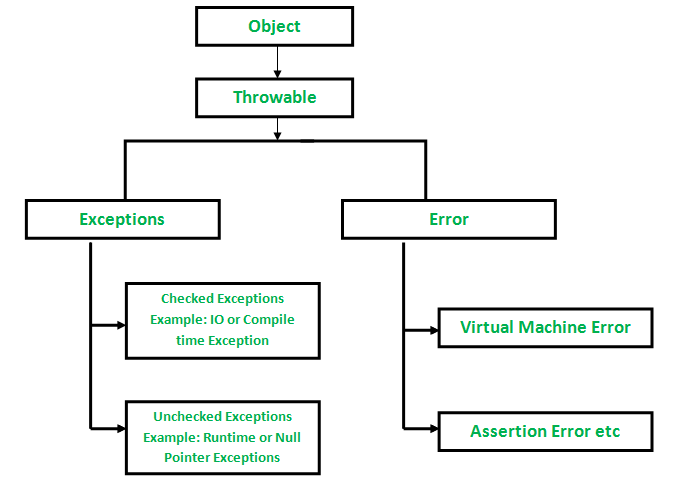
OutOfMemoryError, StackOverflowError, NoClassDefFoundError, VirtualMachineError, and LinkageError.

**Difference between Exception and Error**

The main difference between exception and error is that exception can be recovered by using try-catch block, but error cannot be recovered by using try-catch block.

Another difference between exception and error is that exception is classified into two types: checked exceptions and unchecked exceptions, but error is classified into two types: fatal errors and non-fatal errors.

Finally, exception is an event that occurs during the execution of a program, that disrupts the normal flow of the program's instructions, but error is a serious problem that occurs during the execution of a program, that disrupts the normal flow of the program's instructions.



## **Checked Exceptions in Java**

These are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using the throws keyword. Ex: **ClassNotFoundException, IOException, SQLException and InterruptedException**

## **UnChecked Exceptions in Java**

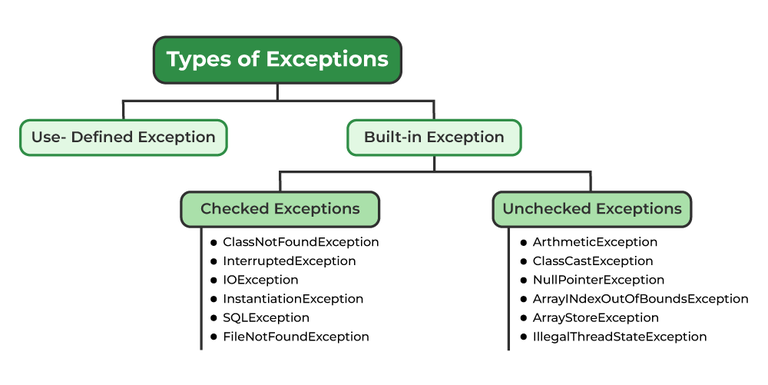
Unchecked exceptions are runtime exceptions that are not required to be caught or declared in a throws clause. These exceptions are usually caused by programming errors, such as attempting to access an index out of bounds in an array or attempting to divide by zero.

Unchecked exceptions include all subclasses of the RuntimeException class, as well as the Error class and its subclasses.

***1. ArrayIndexOutOfBoundsException:*** *This exception is thrown when you attempt to access an array index that is out of bounds.*

***2. NullPointerException:*** *This exception is thrown when you attempt to access a null object reference.*

***3. ArithmeticException:*** *This exception is thrown when you attempt to divide by zero or perform an invalid arithmetic operation.*



Exception handling:

Java exception handling is managed via five keywords: **try**, **catch**, **throw**, **throws**, and **finally**.

* For each try block, there can be zero or more catch blocks, but **only one** final block.
* The finally block is optional. It always gets executed whether an exception occurred in the try block or not. If an exception occurs, then it will be executed after **try and catch blocks.** And if an exception does not occur, then it will be executed after the **try** block. The finally block in Java is used to put important codes such as clean-up code e.g., closing the file or closing the connection.

## **Java throw**

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exceptions. The throw keyword is mainly used to throw custom exceptions.

### **Syntax in Java throw**

**throw *Instance***

Example:

**throw new ArithmeticException("/ by zero");**

But this exception i.e., *Instance* must be of type **Throwable** or a subclass of **Throwable**.

## **Java throws**

throws is a keyword in Java that is used in the signature of a method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.

### **Syntax of Java throws**

**type method\_name(parameters) throws exception\_list**

exception\_list is a comma separated list of all the

exceptions which a method might throw.

**Exception Handling:**

In a program, if there is a chance of raising an exception then the compiler always warns us about it and compulsorily we should handle that checked exception, Otherwise, we will get compile time error saying **unreported exception XXX must be caught or declared to be thrown**. To prevent this compile time error we can handle the exception in two ways:

1. By using try catch
2. By using the **throws** keyword(It is used to escape from handling the exception)

We can use the throws keyword to delegate the responsibility of exception handling to the caller (It may be a method or JVM) then the caller method is responsible to handle that exception.

**Control flow in try-catch OR try-catch-finally**

**1. Exception occurs in try block and handled in catch block:** If a statement in try block raises an exception, then the rest of the try block doesn’t execute and control passes to the **corresponding** catch block. After executing the catch block, the control will be transferred to finally block(if present) and then the rest program will be executed

**2. Exceptions occurring in try-block are not handled in catch block:** In this case, the default handling mechanism is followed. If a finally block is present, it will be executed followed by the default handling mechanism.

**3. Exception doesn’t occur in try-block:** In this case catch block never runs as they are only meant to be run when an exception occurs. finally block(if present) will be executed followed by rest of the program.

**Custom Exception or User Defined Exception:**

Java provides us the facility to create our own exceptions which are basically derived classes of Exception. Creating our own Exception is known as a custom exception or user-defined exception.

**Why do we create custom exceptions?**

Business logic exceptions: These are the exceptions related to business logic and workflow. It is useful for the application users or the developers to understand the exact problem.

In order to create a custom exception, we need to extend the Exception class that belongs to **java.lang package.**