**Exception Handling**

Exception handling is a mechanism in Java, that deals with runtime errors or exceptional situations that occur during the execution of a program. These exceptional situations can include things like division by zero, attempting to access a file that doesn't exist, or trying to access a resource that's unavailable.

**Java provides five keywords that are used to handle the exception.**

1. **`try`:**

This keyword is used to enclose the code that might throw an exception. Inside the `try` block, you place the code that you want to monitor for exceptions.

1. **`catch`:**

If an exception occurs within the `try` block, the corresponding `catch` block is executed. You can have multiple `catch` blocks to handle different types of exceptions. Each `catch` block specifies the type of exception it can handle.

1. **`finally`:**

The `finally` block is optional and is used to execute code that should always run, regardless of whether an exception occurred or not. It's commonly used for cleanup tasks like closing files or releasing resources.

1. **throw:**

The throw keyword is used to explicitly throw an exception within a method or block of code. It is followed by an instance of an exception class or subclass. This allows you to handle exceptional situations programmatically.

1. **throws:**

The throws keyword is used in the method signature to declare that a method may throw one or more types of exceptions. It doesn't actually throw the exception itself; it simply specifies the types of exceptions that the method may throw. This informs the caller of the method about the potential exceptions that need to be handled.

### Types of Java Exceptions

1. **Checked Exceptions:**

These are exceptions that the compiler forces you to handle. If a method may throw a checked exception, it must either handled using a `try-catch` block or declare that it throws the exception using the `throws` keyword. Examples include `IOException`, `SQLException`, etc.

1. **Unchecked Exceptions:**

Also known as runtime exceptions, these exceptions don't need to be explicitly handled by the programmer. They usually occur due to programming errors, such as trying to access an array element that doesn't exist (`ArrayIndexOutOfBoundsException`) or calling a method on a `null` object (`NullPointerException`). Unchecked exceptions extend `RuntimeException` or `Error` classes and are not enforced by the compiler.

1. **Errors:**

In Java, errors, subclasses of Throwable, signify severe runtime issues beyond program control like memory exhaustion (OutOfMemoryError), JVM problems (VirtualMachineError), or failed assertions (AssertionError). Unlike exceptions, errors aren't meant to be handled by application code, often requiring system-level intervention or application termination.