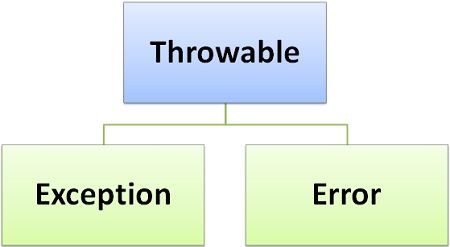
**Difference Between Error and Exception in Java :**



“Throwable” act as the root for Java’s error and exception hierarchy. “**Error**” is a critical condition that cannot be handled by the code of the program. “**Exception**” is the exceptional situation that can be handled by the code of the program.

The significant difference between error and exception is that an error is caused due to lack of system resources, and an exception is caused because of your code.

|  |  |  |
| --- | --- | --- |
| **BASIS FOR COMPARISON** | **ERROR** | **EXCEPTION** |
| **Basic** | An error is caused due to lack of system resources. | An exception is caused because of the code |
| **Recovery** | An error is irrecoverable. | An exception is recoverable. |
| **Keywords** | There is no means to handle an error by the program code. | Exceptions are handled using three keywords "try", "catch", and "throw". |
| **Consequences** | As the error is detected the program will terminated abnormally. | As an exception is detected, it is thrown and caught by the "throw" and "catch" keywords correspondingly. |
| **Types** | Errors are classified as unchecked type. | Exceptions are classified as checked or unchecked type. |
| **Package** | In Java, errors are defined "java.lang.Error" package. | In Java, an exceptions are defined in"java.lang.Exception". |
| **Example** | OutOfMemory, StackOverFlow. | Checked Exceptions : NoSuchMethod, ClassNotFound.  Unchecked Exceptions : NullPointer, IndexOutOfBounds. |

**Java – Try with Resources : I**ntroduced in Java 7 – allows us to declare resources to be used in a try block with the assurance that the resources will be closed when after the execution of that block.

The resources declared must implement the AutoCloseable interface.

A resource must be both declared and initialized inside the try,

try (PrintWriter writer = new PrintWriter(new File("test.txt"))) {

writer.println("Hello World");

}

**Replacing try–catch-finally With try-with-resources :**The simple way to use the new try-with-resources functionality is to replace the traditional  try-catch-finally block.

Let's compare the following code samples – first is a typical try-catch-finally block, then the new approach, using an equivalent try-with-resources block:

Scanner scanner = null;

try {

scanner = new Scanner(new File("test.txt"));

while (scanner.hasNext()) {

System.out.println(scanner.nextLine());

}

} catch (FileNotFoundException e) {

e.printStackTrace();

} finally {

if (scanner != null) {

scanner.close();

}

}

And here's the solution using try-with-resources:

try (Scanner scanner = new Scanner(new File("test.txt"))) {

while (scanner.hasNext()) {

System.out.println(scanner.nextLine());

}

} catch (FileNotFoundException fnfe) {

fnfe.printStackTrace();

}

**try-with-resources With Multiple Resources :** Multiple resources can be declared just fine in a try-with-resources block by separating them with a semicolon:

try (Scanner scanner = new Scanner(new File("testRead.txt"));

PrintWriter writer = new PrintWriter(new File("testWrite.txt"))) {

while (scanner.hasNext()) {

writer.print(scanner.nextLine());

}

}

**A Custom Resource With AutoCloseable :**To construct a custom resource that will be correctly handled by a try-with-resources block, the class should implement the Closeable or AutoCloseable interfaces, and override the close method:

public class MyResource implements AutoCloseable {

@Override

public void close() throws Exception {

System.out.println("Closed MyResource");

}

}

**Note : Resources that were defined/acquired first will be closed last.**

## **catch & finally :** A try-with-resources block can still have the catch and finally blocks – which will work in the same way as with a traditional try block.

**Java 9: Effectively Final Variables : Try-with-resources Java 9 Enhancement**

Before Java 9 a resource that is to be automatically closed must be created inside the parentheses of the try block of a try-with-resources construct. From Java 9, this is no longer necessary. If the variable referencing the resource is effectively final, you can simply enter a reference to the variable inside the try block parentheses. Here is an example of the Java 9 try-with-resources enhancement:

private static void printFile() throws IOException {

FileInputStream input = new FileInputStream("file.txt");

try(input) {

int data = input.read();

while(data != -1){

System.out.print((char) data);

data = input.read();

}

}

}

Before Java 9, we only could use fresh variables inside a try-with-resources block:

try (Scanner scanner = new Scanner(new File("testRead.txt"));

PrintWriter writer = new PrintWriter(new File("testWrite.txt"))) {

// omitted

}

As shown above, this was especially verbose when declaring multiple resources. As of Java 9 and as part of JEP 213, we can now use final or even effectively final variables inside a try-with-resources block:

final Scanner scanner = new Scanner(new File("testRead.txt"));

PrintWriter writer = new PrintWriter(new File("testWrite.txt"))

try (scanner;writer) {

// omitted

}

Put simply, a variable is effectively final if it doesn't change after the first assignment, even though it's not explicitly marked as final.

As shown above, the scanner variable is declared final explicitly, so we can use it with the try-with-resources block. Although the writer variable is not explicitly final, it doesn't change after the first assignment. Therefore, we're allowed to use the writer variable, too.

**how to use try-with-resources, how to replace try, catch, and finally with try-with-resources, building custom resources with AutoCloseable and order in which resources are closed.**

**Catching Multiple Exceptions in Java 7 :** In Java 7 it was made possible to catch multiple different exceptions in the same catch block. This is also known as multi catch.

Before Java 7 you would write something like this:

try {

// execute code that may throw 1 of the 3 exceptions below.

} catch(SQLException e) {

logger.log(e);

} catch(IOException e) {

logger.log(e);

} catch(Exception e) {

logger.severe(e);

}

As you can see, the two exceptions SQLException and IOException are handled in the same way, but you still have to write two individual catch blocks for them.

In Java 7 you can catch multiple exceptions using the multi catch syntax:

try {

// execute code that may throw 1 of the 3 exceptions below.

} catch(SQLException | IOException e) {

logger.log(e);

} catch(Exception e) {

logger.severe(e);

}

Notice how the two exception class names in the first catch block are separated by the pipe character |. The pipe character between exception class names is how you declare multiple exceptions to be caught by the same catch clause.

**Multithreading :**

**<https://techdifferences.com/difference-between-sleep-and-wait-method-in-java.html>**