**The try-with-resources Statement**

The try-with-resources statement is a try statement that declares one or more resources. A *resource* is an object that must be closed after the program is finished with it. The try-with-resources statement ensures that each resource is closed at the end of the statement. Any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable, can be used as a resource.

The following example reads the first line from a file. It uses an instance of BufferedReader to read data from the file. BufferedReader is a resource that must be closed after the program is finished with it:

static String readFirstLineFromFile(String path) throws IOException {

**try (BufferedReader br =**

**new BufferedReader(new FileReader(path)))** {

return br.readLine();

}

}

In this example, the resource declared in the try-with-resources statement is a BufferedReader. The declaration statement appears within parentheses immediately after the try keyword. The class BufferedReader, in Java SE 7 and later, implements the interface java.lang.AutoCloseable. Because the BufferedReader instance is declared in a try-with-resource statement, it will be closed regardless of whether the try statement completes normally or abruptly (as a result of the method BufferedReader.readLine throwing an IOException).

Prior to Java SE 7, you can use a finally block to ensure that a resource is closed regardless of whether the try statement completes normally or abruptly. The following example uses a finally block instead of a try-with-resources statement:

static String readFirstLineFromFileWithFinallyBlock(String path)

throws IOException {

BufferedReader br = new BufferedReader(new FileReader(path));

try {

return br.readLine();

} finally {

if (br != null) br.close();

}

}

However, in this example, if the methods readLine and close both throw exceptions, then the method readFirstLineFromFileWithFinallyBlock throws the exception thrown from the finally block; the exception thrown from the try block is suppressed. In contrast, in the example readFirstLineFromFile, if exceptions are thrown from both the try block and the try-with-resources statement, then the method readFirstLineFromFile throws the exception thrown from the try block; the exception thrown from the try-with-resources block is suppressed. In Java SE 7 and later, you can retrieve suppressed exceptions; see the section [Suppressed Exceptions](https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html#suppressed-exceptions) for more information.

You may declare one or more resources in a try-with-resources statement. The following example retrieves the names of the files packaged in the zip file zipFileName and creates a text file that contains the names of these files:

public static void writeToFileZipFileContents(String zipFileName,

String outputFileName)

throws java.io.IOException {

java.nio.charset.Charset charset =

java.nio.charset.StandardCharsets.US\_ASCII;

java.nio.file.Path outputFilePath =

java.nio.file.Paths.get(outputFileName);

// Open zip file and create output file with

// try-with-resources statement

**try (**

**java.util.zip.ZipFile zf =**

**new java.util.zip.ZipFile(zipFileName);**

**java.io.BufferedWriter writer =**

**java.nio.file.Files.newBufferedWriter(outputFilePath, charset)**

**)** {

// Enumerate each entry

for (java.util.Enumeration entries =

zf.entries(); entries.hasMoreElements();) {

// Get the entry name and write it to the output file

String newLine = System.getProperty("line.separator");

String zipEntryName =

((java.util.zip.ZipEntry)entries.nextElement()).getName() +

newLine;

writer.write(zipEntryName, 0, zipEntryName.length());

}

}

}

In this example, the try-with-resources statement contains two declarations that are separated by a semicolon: ZipFile and BufferedWriter. When the block of code that directly follows it terminates, either normally or because of an exception, the close methods of the BufferedWriter and ZipFile objects are automatically called in this order. Note that the close methods of resources are called in the *opposite* order of their creation.

The following example uses a try-with-resources statement to automatically close a java.sql.Statement object:

public static void viewTable(Connection con) throws SQLException {

String query = "select COF\_NAME, SUP\_ID, PRICE, SALES, TOTAL from COFFEES";

**try (Statement stmt = con.createStatement())** {

ResultSet rs = stmt.executeQuery(query);

while (rs.next()) {

String coffeeName = rs.getString("COF\_NAME");

int supplierID = rs.getInt("SUP\_ID");

float price = rs.getFloat("PRICE");

int sales = rs.getInt("SALES");

int total = rs.getInt("TOTAL");

System.out.println(coffeeName + ", " + supplierID + ", " +

price + ", " + sales + ", " + total);

}

} catch (SQLException e) {

JDBCTutorialUtilities.printSQLException(e);

}

}

The resource java.sql.Statement used in this example is part of the JDBC 4.1 and later API.

**Note**: A try-with-resources statement can have catch and finally blocks just like an ordinary try statement. In a try-with-resources statement, any catch or finally block is run after the resources declared have been closed.

**Suppressed Exceptions**

An exception can be thrown from the block of code associated with the try-with-resources statement. In the example writeToFileZipFileContents, an exception can be thrown from the try block, and up to two exceptions can be thrown from the try-with-resources statement when it tries to close the ZipFile and BufferedWriter objects. If an exception is thrown from the try block and one or more exceptions are thrown from the try-with-resources statement, then those exceptions thrown from the try-with-resources statement are suppressed, and the exception thrown by the block is the one that is thrown by the writeToFileZipFileContents method. You can retrieve these suppressed exceptions by calling the Throwable.getSuppressed method from the exception thrown by the try block.

**Classes That Implement the AutoCloseable or Closeable Interface**

See the Javadoc of the [AutoCloseable](https://docs.oracle.com/javase/8/docs/api/java/lang/AutoCloseable.html" \t "_blank) and [Closeable](https://docs.oracle.com/javase/8/docs/api/java/io/Closeable.html" \t "_blank) interfaces for a list of classes that implement either of these interfaces. The Closeable interface extends the AutoCloseable interface. The close method of the Closeable interface throws exceptions of type IOException while the close method of the AutoCloseable interface throws exceptions of type Exception. Consequently, subclasses of the AutoCloseable interface can override this behavior of the close method to throw specialized exceptions, such as IOException, or no exception at all.