**try-with-resources** is a Java statement that ensures that resources (like files, database connections, or network connections) are closed automatically after being used, without needing to explicitly close them in a finally block. This feature was introduced in **Java 7**.

The **resources** must implement the **AutoCloseable** interface, which ensures they are properly closed once the try block is exited, regardless of whether an exception is thrown or not.

**Example: Reading from a File Using try-with-resources**

Let's take an example where we read from a file using a BufferedReader. Using try-with-resources, we ensure that the file is closed automatically after the reading is done.

**Step-by-Step:**

1. Create a file sample.txt containing some text.
2. Use BufferedReader in a try-with-resources block to read the file content.
3. Ensure that the file resource is closed automatically when the try block finishes.

**Code Example:**

java

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class TryWithResourcesExample {

public static void main(String[] args) {

// Path to the file

String filePath = "sample.txt";

// Try-with-resources to ensure automatic resource management

try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {

String line;

while ((line = br.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

System.out.println("Error occurred while reading the file: " + e.getMessage());

}

}

}

**Explanation:**

* **try (BufferedReader br = new BufferedReader(new FileReader(filePath))**:
  + The resource BufferedReader is declared inside the parentheses. When the try block exits, whether normally or due to an exception, the BufferedReader is automatically closed.
* **br.readLine()**:
  + Reads the file line by line and prints the content to the console.
* **catch (IOException e)**:
  + If any IOException occurs (such as the file not being found), it is handled in the catch block.

**Sample Output (if sample.txt contains the following lines):**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Hello, this is a sample file.

We are using try-with-resources.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Hello, this is a sample file.

We are using try-with-resources.

**Benefits of try-with-resources:**

1. **Automatic Resource Management**: No need for manually closing resources in finally blocks.
2. **Simpler and Cleaner Code**: Removes the need to write close() methods explicitly.
3. **Handles Exceptions Properly**: Even if exceptions occur, resources are guaranteed to be closed.

**Example with Multiple Resources:**

You can manage multiple resources in a single try-with-resources block by separating the resource declarations with semicolons.

java

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

try (FileReader fr = new FileReader("sample.txt");

BufferedReader br = new BufferedReader(fr)) {

String line;

while ((line = br.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

System.out.println("Error occurred: " + e.getMessage());

}

This example demonstrates how **try-with-resources** ensures that both the FileReader and BufferedReader are automatically closed when the block finishes.