Java try with Resource Statement

Try with resource is a new feature of Java that was introduced in Java 7 and further improved in Java 9. This feature add another way to exception handling with resources management. It is also referred as **automatic resource management**. It close resources automatically by using AutoCloseable interface.

Resource can be any like: file, connection etc and we don't need to explicitly close these, JVM will do this automatically.

Suppose, we run a JDBC program to connect to the database then we have to create a connection and close it at the end of task as well. But in case of try-with-resource we don't need to close the connection, JVM will do this automatically by using AutoCloseable interface.

Try with Resource Syntax

```
try(resource-specification(there can be more than one resource))
{
    //use the resource
}
catch()
{
    // handler code
}
```

This **try statement** contains a **parenthesis** in which one or more resources is declared. Any object that implements `java.lang.AutoCloseable` or `java.io.Closeable`, can be passed as a parameter to **try statement**. A resource is an object that is used in program and must be closed after the program is finished. The **try-with-resources statement** ensures that each resource is closed at the end of the statement of the try block. We do not have to explicitly close the resources.

Example without using try with Resource Statement

Lets see the scenario where we are not using try-with-resource block whereas we are using normal try block that's why we need to close the file reference explicitly.

```
import java.io.*;
class Demo
{
    public static void main(String[] args)
    {
        try {
            String str;
            //opening file in read mode using BufferedReader stream
            BufferedReader br = new BufferedReader(new FileReader("d:\\myfile.txt"));
            while((str=br.readLine())!=null)
            {
                System.out.println(str);
            }
            br.close();     //closing BufferedReader stream
        }
        catch(IOException ie)
        {
            System.out.println("I/O Exception "+ie);
        }
```

Output:

```
I/O Exception java.io.FileNotFoundException: d:\myfile.txt (No such file or directory)
```

Example try with Resource Statement

Here, we are using try-with-resource to open the file and see we did not use close method to close the file connection.

```
import java.io.*;
class Demo
{
    public static void main(String[] args)
    {
        try(BufferedReader br = new BufferedReader(new FileReader("d:\\myfile.txt")))
        {
            String str;
            while((str = br.readLine()) != null)
            {
                System.out.println(str);
            }
        }
        catch(IOException ie)
        {
            System.out.println("I/O Exception "+ie);
        }
    }
}
```

NOTE: In the above example, we do not need to explicitly call `close()` method to close **BufferedReader stream.**

Try with resource – Java 9

In Java 7, try-with-resource was introduced and in which resource was created inside the try block. It was the limitation with Java 7 that a connection object created outside can not be refer inside the try-with-resource.

In Java 9 this limitation was removed so that now we can create object outside the try-with-resource and then refer inside it without getting any error.

Example

```
import java.io.*;
class Demo
{
    public static void main(String[] args) throws FileNotFoundException
    {
        BufferedReader br = new BufferedReader(new FileReader("d:\\myfile.txt"));
        try(br)  // resource is declared outside the try
        {
            String str;
            while((str = br.readLine()) != null)
            {
                System.out.println(str);
            }
        }
        catch(IOException ie)
        {
            System.out.println("I/O Exception "+ie);
        }
    }
```

Points to Remember

1. A resource is an object in a program that must be closed after the program has finished.
2. Any object that implements `java.lang.AutoCloseable` or `java.io.Closeable` can be passed as a parameter to try statement.
3. All the resources declared in the try-with-resources statement will be closed automatically when the try block exits. There is no need to close it explicitly.
4. We can write more than one resources in the try statement.
5. In a try-with-resources statement, any catch or finally block is run after the resources declared have been closed.

← Prev
Next →