11.9-try with resources

Try-with-Resources and Working with finally Block in Java

Working with try and finally Block

In Java, when working with resources (such as file streams, database connections, Networks etc.), it is essential to close the resource after completing its task. **Failing** to close a resource **may result in memory leaks** or prevent other parts of the application from accessing that resource.

The usual approach to ensure that resources are closed is by using a try block with a finally block. The finally block **guarantees** that the statements within it will be executed, whether an error occurs or not. This is especially useful for closing resources like file streams, which must be closed regardless of the outcome of the try block.

Key Points about the finally Block:

- It always executes, even if an exception is thrown.
- It ensures that critical operations, such as closing resources, are performed.
- It works with or without the catch block.

Example 1: Using try, catch, and finally

```java
public class Hello1 {
    public static void main(String[] args) {
        int i = 2;
        int j = 0;
        try {
            j = 18 / i;
        } catch(Exception e) {
            System.out.println("Something went wrong");
        } finally {
            System.out.println("Bye!");
        }
    }
}
```

Output:

```
Bye!
```

Explanation:

In the above example, even though no exception occurs, the finally block is executed.

Example 2: Handling Exceptions with finally

```
public class Hello1 {
    public static void main(String[] args) {
        int i = 0;
        int j = 0;
        try {
            j = 18 / i;
        } catch(Exception e) {
            System.out.println("Something went wrong");
        } finally {
            System.out.println("Bye!");
        }
    }
}
```

Output:

```
Something went wrong
Bye!
```

Explanation:

Here, when the exception occurs, the flow jumps to the catch block, but the finally block still executes. This ensures that necessary actions like closing resources are performed even in the case of an exception.

Example 3: Using finally to Close Resources

```java
public class Hello {
    public static void main(String[] args) throws IOException {
        int num = 0;
        BufferedReader br = null;
        try {
            System.out.println("Enter your number: ");
            InputStreamReader in = new InputStreamReader(System.in);
            br = new BufferedReader(in);
            num = Integer.parseInt(br.readLine());
            System.out.println("You entered: " + num);
        } finally {
                br.close();  // Closing resource
            }
    }
}
```

Explanation:

In this example, the **BufferedReader** resource is closed in the finally block to ensure it is always closed, regardless of whether an exception occurs.

Try-with-Resources Feature in Java

Java provides an alternative approach called **Try-with-Resources** to manage resources efficiently without needing a finally block to close them explicitly. This approach ensures that the resources are automatically closed after the try block completes.

Key Points about Try-with-Resources:

- The try-with-resources statement ensures that each resource is closed automatically at the end of the statement.
- Any object that implements **java.lang.AutoCloseable** (like BufferedReader, FileInputStream, etc.) can be used in this structure.
- It simplifies **resource management** and reduces the risk of resource leaks.

Syntax of Try-with-Resources:

```
try (declare resources here) {
    // use resources
} catch(Exception e) {
    // handle exceptions
}
```

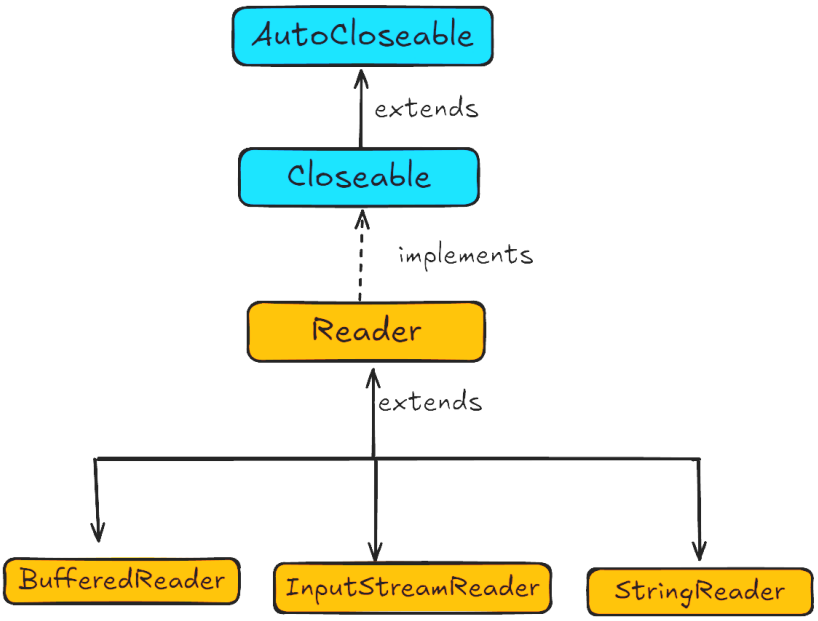
Example: Try-with-Resources

```
public class Hello {
    public static void main(String[] args) throws IOException {
        int num = 0;

        // Using try-with-resources
        try (BufferedReader br = new BufferedReader(new InputStreamReader(System.in))) {
            System.out.println("Enter your number: ");
            num = Integer.parseInt(br.readLine());
            System.out.println("You entered: " + num);
        }
    }
}
```

Explanation:

In the above example, the BufferedReader is declared in the try block, and it automatically closes after the execution of the block. Since BufferedReader implements interface **Closeable**, which extends the another interface **AutoCloseable**, Java ensures it is closed without requiring a finally block.



Advantages of Try-with-Resources:

- **Cleaner code**: No need for explicit resource closing in finally.
- **Automatic resource management**: Resources are automatically closed once the try block completes.
- **Fewer bugs**: Reduces the chances of forgetting to close a resource and causing memory leaks

Conclusion

Using the try with resources statement is the recommended practice for managing resources in Java, as it automatically handles the closing of resources. However, using a finally block provides a clear structure for resource management and can be beneficial in scenarios where specific actions need to be performed regardless of exceptions.

TELUSKO