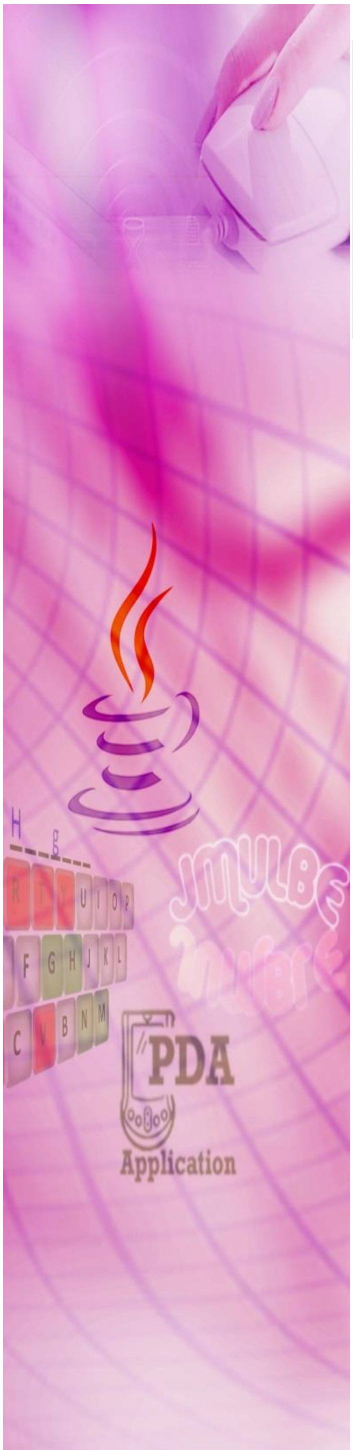
Session 11

Instructor Inputs



Session Overview

This session covers the Implementing Watch Service topic of the Introducing NIO section and the Performing the Read and Write Operations on a File section of Chapter 8. In addition, it contains the Identifying the Layers in the JDBC Architecture section, the Identifying the Types of JDBC Drivers section, and the following topics of the Using JDBC API section of the book, Programming in Java – SG:

- Loading a Driver
- Connecting to a Database

In this session, the students will explore the concept of watch service in Java. They will learn the concept of performing efficient read and write operations on a file by using NIO API. In addition, they will learn loading the drivers, which will enable an application to connect to a database.

Introducing NIO

Handling Tips

Start the discussion by introducing the concept of watch service to the students. Explain to the students that watch service can be used when there is a requirement to monitor a directory for events, such as the creation or deletion of a file/directory, as explained in the SG. Explain each of the steps and the code snippets required to implement watch service. Finally, explain the entire code that is required to implement watch service. In addition, while discussing this topic, you can discuss the additional information given under the Implementing Watch Service subtopic in the Additional Inputs topic.

Additional Inputs

Implementing Watch Service

In addition to the create, delete, and modify methods, the `StandardWatchEventKinds` class also defines the `OVERFLOW` constant. This constant is used for indicating that events might have been lost or discarded when watch service was implemented. In addition, you do not have to register for the `OVERFLOW` event to receive it.

Further, to get a count of the number of events that are detected by watch service, you can use the following code snippet:

```
key = watchService.take(); // get watch key
                for (WatchEvent event: key.pollEvents()) {
                    System.out.println(event.count());
                    System.out.println("The event that occurred on " +
event.context().toString() + " is " + kind);
                 }
```

Performing the Read and Write Operations on a File

Handling Tips

Explain the scenario to the students, and then tell that it is essential to ensure that an efficient approach is used for applications that perform a read or write operation frequently and deal with a significant amount of data. Else, the efficiency and throughput of the application may be reduced. Using an efficiency approach will further optimize the performance of the application. They can achieve the preceding tasks by making use of NIO. This reduces the code complexity.

Further, explain how to read and write to a file by using the `newbufferedreader()` and `newbufferedwriter()` methods, respectively. Ensure that the students have gained clarity on how to read and write from/to a file. After explaining the preceding methods, you can demonstrate the code given in the Additional Examples topic that reads the content of a file and writes the content in a new file.

Additional Examples

The following code demonstrates the read and write operations by using the `newbufferedreader()` and `newbufferedwriter()` methods, respectively:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
public class ReadWriteDemo {
public static void main(String a[]) {
 Path pathObject = Paths.get("D:/Hello.java");
 Path file = Paths.get("D:/CopyofHello.java");
 Charset charset = Charset.forName("US-ASCII");
 try (BufferedReader reader = Files.newBufferedReader(pathObject, charset)) {
 String line = null;
 while ((line = reader.readLine()) != null) {
 try (BufferedWriter writer = Files.newBufferedWriter(file, charset,
StandardOpenOption.CREATE, StandardOpenOption.APPEND))
 {
  writer.write(line.toUpperCase());
  writer.newLine();
  } catch (IOException e) {
  System.out.println(e);
  }
 }
 System.out.println("Copy created");
 } catch (IOException x) {
 System.out.println(x);
 }
 }
}
```

The preceding code reads a file and writes its content in another file in the uppercase format.

Activity 8.1: Performing File Operations Using NIO

Handling Tips

Discuss the problem statement with the students.

To perform the activity, 8.1, you need to create the **Details** and **Backup** folders in the **D:\** drive of your computer.

The solution file, **ComputerProducts.java**, for this activity is provided at the following location in the TIRM CD:

- **Datafiles For Faculty\Activities\Chapter 08\Activity 8.1\Solution**

Identifying the Layers in the JDBC Architecture

Handling Tips

Start the discussion by explaining to the students that the applications, such as Airline Management System or Railway Reservation System, need to store and retrieve a large amount of data. To handle the large amount of data in an efficient manner, the database can be used. Therefore, it is necessary to create a Java application that can interact with the database. Next, explain to the students that JDBC API provided with Java Software Development Kit (SDK) simplifies the process of creating a Java application that stores, updates, and retrieves the data in the database. Then, explain the JDBC architecture that enables a Java application to interact with the database. Further, use the figure provided in the SG to explain the two layers of the JDBC architecture. In addition, while explaining this topic, you can discuss the advantages of JDBC by using the information given in the Additional Inputs topic.

Additional Inputs

JDBC provides the following advantages:

- **Simplified enterprise development**: The combination of Java API and JDBC API makes application development easy and economical. JDBC API is easy to learn and deploy, and inexpensive to maintain.
- **Zero configurations for network computers**: With JDBC API, configuration is not required on the client side. With a driver written in the Java programming language, all the information required to make a connection is completely defined by the JDBC URL or by a data source object registered with the Java Naming and Directory Interface (JNDI) naming service.

Identifying the Types of JDBC Drivers

Handling Tips

Explain to the students that the JDBC drivers are software that converts the Java instructions into the database instructions and vice-versa. Next, list the various types of drivers supported by JDBC.

Further, explain the JDBC-ODBC Bridge driver with the help of the figure given in the SG. In addition, inform the students that the JDBC-ODBC Bridge driver is also called the Type 1 driver.

Thereafter, emphasize on the point that the type 1 driver enables a Java application to connect with any database that supports the ODBC driver.

Next, explain the Native-API driver with the help of the figure given in the SG. In addition, inform the students that the Native-API driver is also called the Type 2 driver. Thereafter, emphasize on the point that CLI needs to be loaded on the client computer to use the Type 2 driver.

Further, explain the Network Protocol driver with the help of the figure given in the SG. In addition, inform the students that the Network Protocol driver is also called the Type 3 driver. Thereafter, emphasize on the point that the CLI libraries are to be loaded on the server at the time of using the Type 3 driver.

Next, explain the Native-Protocol driver with the help of the figure given in the SG. In addition, inform the students that the Native-Protocol driver is also called the Type 4 driver. Further, emphasize on the point that the Type 4 driver is the most commonly-used driver because there is no need to install vendor-specific libraries.

Using JDBC API

Handling Tips

Explain to the students that the database drivers and JDBC API collectively enable them to manipulate data stored in a database. Thereafter, inform the students that JDBC API provides the various classes to handle the different activities required to manipulate data. Next, discuss the classes and interfaces of JDBC API. Further, list the various activities required to query a database using Java applications.

Next, explain to the students that to connect to a database, an appropriate driver must be loaded and registered with the driver manager. Thereafter, explain the various ways of loading and registering the driver. Further, explain the code used to load and register the Type 4 driver for SQL Server databases. In addition, while discussing this topic, you can discuss how to connect to the Oracle database by using the example given in the Additional Examples topic.

Thereafter, explain to the students that irrespective of the database, the `getConnection()` method of the `DriverManager` class is used to create a `Connection` object that enables them to connect to the database. Next, discuss the various overloaded forms of the `getConnection()` method.

Conclude the session by asking the students few questions related to the concepts the students have learnt in this session.

Additional Examples

You can use the following code snippet to connect to the Oracle database:

```
Connection con = Class.forName(“oracle.jdbc.driver.OracleDriver");
con=DriverManager.getConnection(
"jdbc:oracle:thin:@//machine_name:1521:database_name", "user_name",
"password");
```

In the preceding code snippet, `thin` is the Oracle JDBC driver, `machine_name` is the name of the machine running Oracle server, `1521` is the name of the port at which the Oracle server listens, `database_name` is the name of the database to which you need to connect, and `user_name` and `password` are the user name and password of the Oracle server, respectively.

FAQs

- *Are the watch events thread-safe?*

  Ans: Yes, the watch events are thread-safe.

- *What will happen if a watch key is not reset?*

  Ans: If the watch key is not reset, the watch key will not be able to receive further events.

- *What are the disadvantages of the Type 2 driver?*

  Ans: The Type 2 driver has the following disadvantages:

  - You need to install the vendor client library on the client machine to use the Type 2 driver.
  - The Type 2 driver cannot be used for Internet-based applications.
  - The Type 2 drivers cannot be used for all the databases, as many database providers do not provide the client side library required for the Type 2 drivers.

- *What is the Oracle thin driver?*

  Ans: The Oracle thin driver is the JDBC Type 4 driver, which uses Java to connect directly to the Oracle database. To use this driver, you do not require the Oracle client software installed on the client computer. However, the server must be configured with a TCP/IP listener. The thin driver is entirely written in Java and is platform-independent. Therefore, the thin driver can be downloaded into any browser as a part of a Java application.

- *Which changes are required in a Java application if the database server changes?*

  Ans: You need to perform the following changes in a Java application if the database server changes:

  - Update the code snippet used to load the driver, and map it to the driver used for the current database server.
  - Update the code snippet used to create the `Connection` object, and map it to the properties of the current database server.