CRUD Operations in Excel Using Apache POI with Java and Maven

Github link: [hacker123shiva/crud-operation-excel-java: CRUD Operations in Excel Using Apache POI with Java and Maven (github.com)]()
LinkedId: https://www.linkedin.com/in/shivasrivastava1/
Java Dev Community: https://www.linkedin.com/groups/14530255/

In this blog, we'll walk you through how to perform **CRUD** (Create, Read, Update, Delete) operations on an Excel sheet using **Apache POI**. We'll use **Maven** to manage dependencies and demonstrate how to implement a solution in a Java project.

Table of Contents:

1. Maven Project Setup

To get started, you'll need to create a Maven project. If you're using an IDE like Eclipse or IntelliJ IDEA, you can create a new Maven project directly.

`pom.xml` - Add the following dependencies to your Maven `pom.xml` file to include Apache POI for Excel manipulation.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.telusko</groupId>
```

```xml
  <artifactId>crud-excel-maven-project</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>shiva</name>
  <dependencies>
    <!-- Apache POI for Excel manipulation -->
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi</artifactId>
        <version>5.2.3</version>
    </dependency>
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml</artifactId>
        <version>5.2.3</version>
    </dependency>

 <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.18</version>
    <scope>provided</scope>
 </dependency>

    <!-- Optional: Logging for better traceability -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.36</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>1.7.36</version>
    </dependency>


 </dependencies>
 </project>
```

This will allow you to work with Excel files and use Lombok for generating getter/setter methods automatically.

2. Project Structure

```
crud-excel-maven-project
├── src
│   └── main
│       ├── java
│       │   └── com.telusko
│       │       ├── Main.java
│       │       ├── entity
│       │       │   └── Student.java
│       │       ├── service
│       │       │   └── ExcelService.java
│       │       └── utility
│       │           └── ExcelHelper.java
│       │
├── pom.xml
```

3. Core Classes and Methods

Main Class (`Main.java`)

The **Main** class serves as the entry point for running the CRUD operations. It creates a list of students, calls the service methods for file creation, reading, updating, and deleting data from the Excel file

```java
package com.telusko;

import com.telusko.entity.Student;
import com.telusko.service.ExcelService;

import java.util.Arrays;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        ExcelService service = new ExcelService();

        // Create a list of students
        List<Student> students = Arrays.asList(
            new Student(1, "Shiva", 20, "shiva@gmail.com"),
            new Student(2, "Puchu", 21, "puchu@gmail.com"),
            new Student(3, "Arjun", 21, "arjun@gmail.com")
        );

        try {
            // Create and write to Excel file
            service.createExcelFile(students);

            // Read data from Excel file
            List<Student> readStudents = service.readExcelFile();
            readStudents.forEach(student ->
System.out.println(student.getName()));

            // Update student name in Excel
            service.updateStudentName(2, "UpdatedName");

            // Delete a student by ID
            service.deleteStudentById(3);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Explanation:

- **createExcelFile(students):** Creates an Excel file with student details.
- **readExcelFile():** Reads and prints all student records from the Excel sheet.
- **updateStudentName(2, "UpdatedName"):** Updates the name of the student with ID 2.
- **deleteStudentById(3):** Deletes the student with ID 3 from the Excel file.

Student Class (`Student.java`)

The **Student** entity represents the data model. It uses Lombok annotations for automatic getter/setter generation, constructors, and `toString()` method.

```java
package com.telusko.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Student {
    private int id;
    private String name;
    private int age;
    private String email;
}
```

Explanation:

- **@Data:** Automatically generates getters, setters, and other useful methods like `toString()`.
- **@NoArgsConstructor, @AllArgsConstructor:** Generate constructors with no arguments and all arguments, respectively.

ExcelService Class (`ExcelService.java`)

The **ExcelService** class handles the core business logic for the CRUD operations. It utilizes the `ExcelHelper` class for reading, writing, updating, and deleting rows in the Excel file.

```java
package com.telusko.service;

import com.telusko.entity.Student;
import com.telusko.utility.ExcelHelper;

import java.io.IOException;
import java.util.List;

public class ExcelService {

    private static final String FILE_PATH = "students.xlsx";

    // Create new Excel file with student data
    public void createExcelFile(List<Student> students) throws IOException
{
        ExcelHelper.writeExcel(FILE_PATH, students);
    }

    // Read data from Excel file
    public List<Student> readExcelFile() throws IOException {
        return ExcelHelper.readExcel(FILE_PATH);
    }

    // Update student name in Excel
    public void updateStudentName(int id, String newName) throws
IOException {
        ExcelHelper.updateExcel(FILE_PATH, id, newName);
    }

    // Delete student from Excel by ID
    public void deleteStudentById(int id) throws IOException {
        ExcelHelper.deleteExcelRow(FILE_PATH, id);
    }
}
```

Explanation:

- **createExcelFile:** Writes the student list to the Excel sheet.
- **readExcelFile:** Reads student data from the Excel sheet and returns a list.
- **updateStudentName:** Updates the student name by searching for the given ID.
- **deleteStudentById:** Deletes a row in the Excel sheet matching the given student ID.

ExcelHelper Class (`ExcelHelper.java`)

This utility class performs the actual read/write/update/delete operations using **Apache POI**.

```
package com.telusko.utility;

import org.apache.poi.ss.usermodel.*;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import com.telusko.entity.Student;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class ExcelHelper {

    private static final String[] HEADERS = {"ID", "Name", "Age", "Email"};
    private static final String SHEET_NAME = "Students";

    // Write Excel file with student data
    public static void writeExcel(String filePath, List<Student> students)
throws IOException {
        Workbook workbook = new XSSFWorkbook();
        Sheet sheet = workbook.createSheet(SHEET_NAME);
        // Create Header row
        Row headerRow = sheet.createRow(0);
        for (int i = 0; i < HEADERS.length; i++) {
            Cell cell = headerRow.createCell(i);
            cell.setCellValue(HEADERS[i]);
        }

        // Write student data to rows
        int rowIdx = 1;
        for (Student student : students) {
            Row row = sheet.createRow(rowIdx++);
            row.createCell(0).setCellValue(student.getId());
            row.createCell(1).setCellValue(student.getName());
            row.createCell(2).setCellValue(student.getAge());
            row.createCell(3).setCellValue(student.getEmail());
        }

        try (FileOutputStream fileOut = new FileOutputStream(filePath)) {
            workbook.write(fileOut);
        }
        workbook.close();
    }

    // Read data from Excel file
    public static List<Student> readExcel(String filePath) throws
IOException {
        List<Student> students = new ArrayList<>();
        try (FileInputStream fileIn = new FileInputStream(filePath)) {
            Workbook workbook = new XSSFWorkbook(fileIn);
            Sheet sheet = workbook.getSheet(SHEET_NAME);

            Iterator<Row> rows = sheet.iterator();
            rows.next();  // Skip header row

            while (rows.hasNext()) {
                Row row = rows.next();
                Student student = new Student();

                student.setId((int) row.getCell(0).getNumericCellValue());
                student.setName(row.getCell(1).getStringCellValue());
                student.setAge((int) row.getCell(2).getNumericCellValue());
                student.setEmail(row.getCell(3).getStringCellValue());
                students.add(student);
            }
        }
        return students;
    }

    // Update student name in Excel file by ID
    public static void updateExcel(String filePath, int id, String newName)
throws IOException {
        try (FileInputStream fileIn = new FileInputStream(filePath)) {
            Workbook workbook = new XSSFWorkbook(fileIn);
            Sheet sheet = workbook.getSheet(SHEET_NAME);

            for (Row row : sheet) {
                if (row.getRowNum() == 0) continue;  // Skip header row
                if ((int) row.getCell(0).getNumericCellValue() == id) {
                    row.getCell(1).setCellValue(newName);
                    break;
                }
            }

            try (FileOutputStream fileOut = new FileOutputStream(filePath))
{
                workbook.write(fileOut);
            }
            workbook.close();
        }
    }

    // Delete student row from Excel file by ID
    public static void deleteExcelRow(String filePath, int id) throws
IOException {
        try (FileInputStream fileIn = new FileInputStream(filePath)) {
            Workbook workbook = new XSSFWorkbook(fileIn);
            Sheet sheet = workbook.getSheet(SHEET_NAME);

            for (Row row : sheet) {
                if ((int) row.getCell(0).getNumericCellValue() == id) {
                    int rowIndex = row.getRowNum();
                    sheet.removeRow(row);
                    break;
                }
            }

            try (FileOutputStream fileOut = new FileOutputStream(filePath))
{
                workbook.write(fileOut);
            }
            workbook.close();
        }
    }
}
```

Explanation:

1. Create Operation

- **`XSSFWorkbook()`**: Creates a new Excel workbook.
  - `Workbook workbook = new XSSFWorkbook();`
- **`createSheet()`**: Creates a new sheet within the workbook.
  - `Sheet sheet = workbook.createSheet("SheetName");`
- **`createRow(int rowIndex)`**: Creates a new row at the specified index in the sheet.
  - `Row row = sheet.createRow(0);`
- **`createCell(int cellIndex)`**: Creates a new cell in the specified row.
  - `Cell cell = row.createCell(0);`
- **`setCellValue()`**: Sets a value for a cell.
  - `cell.setCellValue("Data");`
- **`write(OutputStream)`**: Writes the workbook content to an output stream (like a file).
  - `workbook.write(new FileOutputStream("file.xlsx"));`

2. Read Operation

- **`FileInputStream()`**: Opens an Excel file for reading.
  - `FileInputStream fis = new FileInputStream("file.xlsx");`
- **`XSSFWorkbook(InputStream)`**: Loads an existing workbook from an input stream.
  - `Workbook workbook = new XSSFWorkbook(fis);`
- **`getSheet(String sheetName)`**: Retrieves a specific sheet from the workbook.
  - `Sheet sheet = workbook.getSheet("SheetName");`
- **`iterator()`**: Iterates over rows or cells in a sheet.
  - `Iterator<Row> rows = sheet.iterator();`
  - `Iterator<Cell> cells = row.iterator();`

- **`getCell(int cellIndex)`**: Retrieves a specific cell from a row.
    - `Cell cell = row.getCell(0);`
- **`getNumericCellValue()`**: Gets numeric data from a cell.
    - `int id = (int) cell.getNumericCellValue();`
- **`getStringCellValue()`**: Gets string data from a cell.
    - `String name = cell.getStringCellValue();`

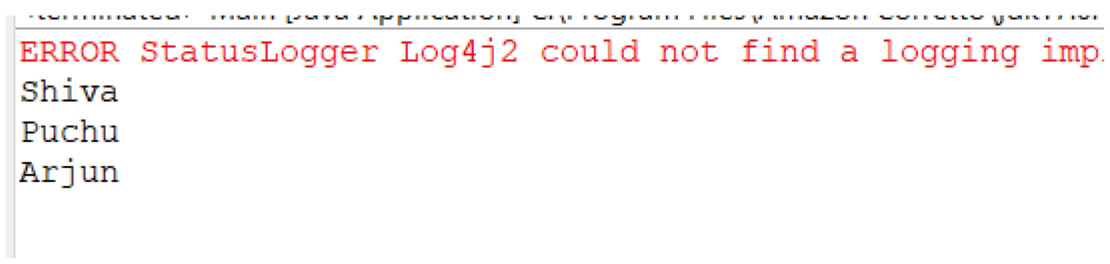
3. Update Operation

- **`getCell(int cellIndex)`**: Fetches a specific cell for updating.
    - `Cell cell = row.getCell(1);`
- **`setCellValue()`**: Sets a new value for the cell (overwrite existing).
    - `cell.setCellValue("UpdatedName");`
- **`write(OutputStream)`**: Rewrites the workbook to save the updates.
    - `workbook.write(new FileOutputStream("file.xlsx"));`

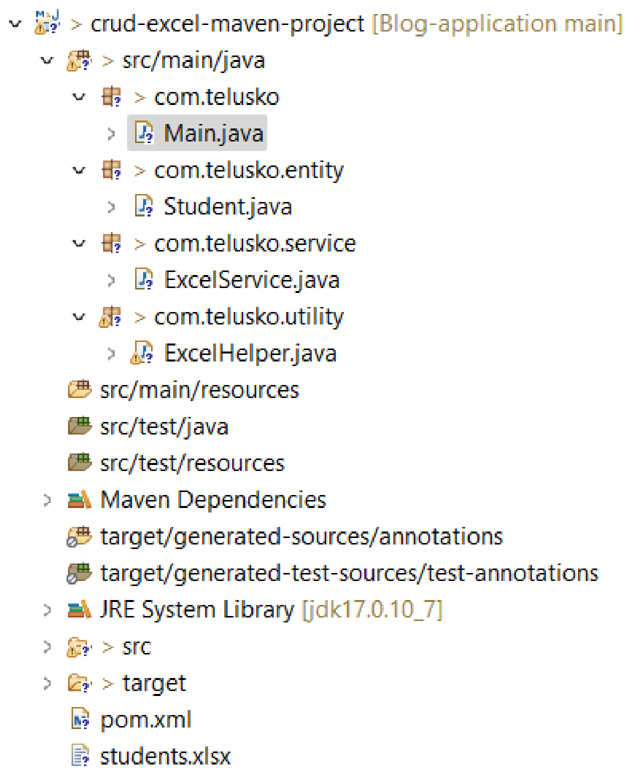
4. Delete Operation

- **`removeRow(Row row)`**: Deletes a specific row from the sheet.
    - `sheet.removeRow(row);`
- **`write(OutputStream)`**: Rewrites the workbook to save changes after deletion.
    - `workbook.write(new FileOutputStream("file.xlsx"));`

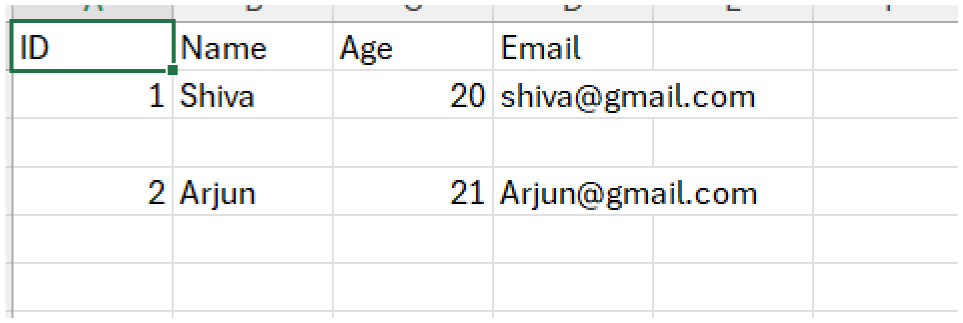
Output:



```
ERROR StatusLogger Log4j2 could not find a logging imp
Shiva
Puchu
Arjun
```



```
crud-excel-maven-project [Blog-application main]
  src/main/java
    com.telusko
      Main.java
    com.telusko.entity
      Student.java
    com.telusko.service
      ExcelService.java
    com.telusko.utility
      ExcelHelper.java
  src/main/resources
  src/test/java
  src/test/resources
  Maven Dependencies
  target/generated-sources/annotations
  target/generated-test-sources/test-annotations
  JRE System Library [jdk17.0.10_7]
  src
  target
  pom.xml
  students.xlsx
```



ID	Name	Age	Email
1	Shiva	20	shiva@gmail.com
2	Arjun	21	Arjun@gmail.com

Conclusion

In this blog, we demonstrated how to use **Apache POI** to perform **CRUD** operations in an Excel file using Java. By leveraging `ExcelHelper` for reading, writing, updating, and deleting rows, and integrating it with a **Student** entity and **ExcelService**, we built a solution that can manipulate Excel files programmatically.