Київський національний університет імені Тараса Шевченка

**Лабораторна робота №2**

**З навчального курсу «Методи специфікації та верифікації програм»**

**«Лабораторна 2 (Test NG)»**

Виконав:

студент 3 курсу

факультету кібернетики

спеціальність «Комп’ютерні науки»

групи ТТП-32

Чебан Богдан Володимирович

**Київ 2024**

**Постановка задачі:**

Тести розробляються для однієї з лабораторних з курсу Системне програмування, або іншого проекту не меншої складності.  
Мова програмування не обов’язково Java.  
При виконанні роботи іншою мовою програмування потрібно використовувати аналогічний фреймворк (наприклад nUnit для  
C# в першій лабораторній).  
          
Використовуючи TestNG написати автономні(юніт) тести, які б:

* покривали принаймні 75% коду.
* містили setup методи з анотаціями @Before @BeforeClass;
* містили принаймні 4 різних Assert вирази;
* принаймні один тестовий метод, що тестує виключення.
* містили принаймні один параметризований тестовий метод (Data Provider).

Створити 2 конфігурації тестів (xml файли)

* в одній з конфігурацій тести мають запускатись паралельно;
* мають бути використані принаймні дві групи для тестів;
* один з тестових наборів, заданий конфігурацією має складатись з всіх тестів, що входять до однієї з груп, крім тих, що входять до іншої.

Додаткові завдання, що дозволяють перекрити штрафні бали за дедлайни та невиконані основні умови:

* використання обґрунтоване залежностей між тестами.
* використання обґрунтоване @BeforeGroup.

**Хід роботи:**

Ця програма на Java складається з декількох класів, які разом формують систему для управління записами студентів у різних форматах зберігання (текстовий файл, таблиця Excel і XML-файл). Ось огляд кожного класу та його функціональності:

**Клас Student (org.example.Student)**:

Представляє сутність студента з властивостями, такими як ідентифікатор студента, ім'я, прізвище та спеціальність.

Використовує анотації Lombok для стандартного коду, такі як гетери, сетери, конструктори та метод **toString**.

Метод **toString** форматує інформацію про студента як рядок, розділений комами.

**Клас StudentExcelService (org.example.StudentExcelService)**:

Управляє записами студентів у файлі Excel (.xlsx формат).

Надає методи для додавання та видалення студента з аркуша Excel.

Використовує бібліотеку Apache POI для роботи з файлами Excel.

**Клас StudentFileService (org.example.StudentFileService)**:

Обробляє записи студентів у текстовому файлі.

Підтримує додавання, видалення та отримання усіх записів студентів.

Використовує Java NIO для операцій з файлами, з методами для читання з файлу та запису в нього.

**Клас StudentXmlService (org.example.StudentXmlService)**:

Управляє записами студентів у XML-файлі.

Пропонує методи для додавання та видалення запису студента.

Використовує бібліотеку JDOM для аналізу та маніпулювання XML.

Кожен із цих сервісних класів (**StudentExcelService**, **StudentFileService**, **StudentXmlService**) призначений для роботи з певним типом файлу, що дозволяє здійснювати постійне зберігання та маніпуляцію даними студентів у різних форматах. Вони мають конструктор, який приймає шлях до файлу, який вони використовують для доступу та модифікації відповідного файлу даних. Методи **addStudent** та **removeStudent** у кожному сервісному класі обробляють деталі додавання або видалення записів студентів відповідно до формату зберігання.

Загалом, ця програма є всеосяжною системою для управління даними студентів в різних рішеннях зберігання, демонструючи використання об'єктно-орієнтованого дизайну, зовнішніх бібліотек для обробки файлів і пакету Java NIO для операцій з файлами.

Я розробляю і тестую систему для управління записами студентів, яка включає роботу з різними форматами файлів (Excel, текстовий файл, XML). Ось як я планую роботу:

1. **Аналіз вимог та планування:**

Я детально визначаю функціональність, яку необхідно реалізувати в системі, включаючи додавання, видалення та отримання записів студентів.

Я обираю, які формати файлів (Excel, текстовий, XML) будуть підтримуватися системою.

1. **Розробка:**

Я створюю клас **Student** для представлення даних студента.

Я розробляю сервісні класи (**StudentExcelService**, **StudentFileService**, **StudentXmlService**) для роботи з різними типами файлів.

Я імплементую методи для додавання, видалення та отримання студентів в кожному сервісному класі.

1. **Юніт-тестування:**

Я пишу юніт-тести для кожного методу в сервісних класах, використовуючи бібліотеку TestNG.

Я перевіряю, що кожен функціонал працює правильно в ізоляції, включаючи позитивні та негативні сценарії.

1. **Інтеграція та тестування системи:**

Я інтегрую всі компоненти системи для спільної роботи.

Я виконую інтеграційні тести для перевірки взаємодії між компонентами системи.

1. **Деплоймент і запуск:**

Я конфігурую середовище, де буде розгорнуто програму.

Я запускаю програму та проводжу випробування в реальних умовах для переконання, що система функціонує стабільно.

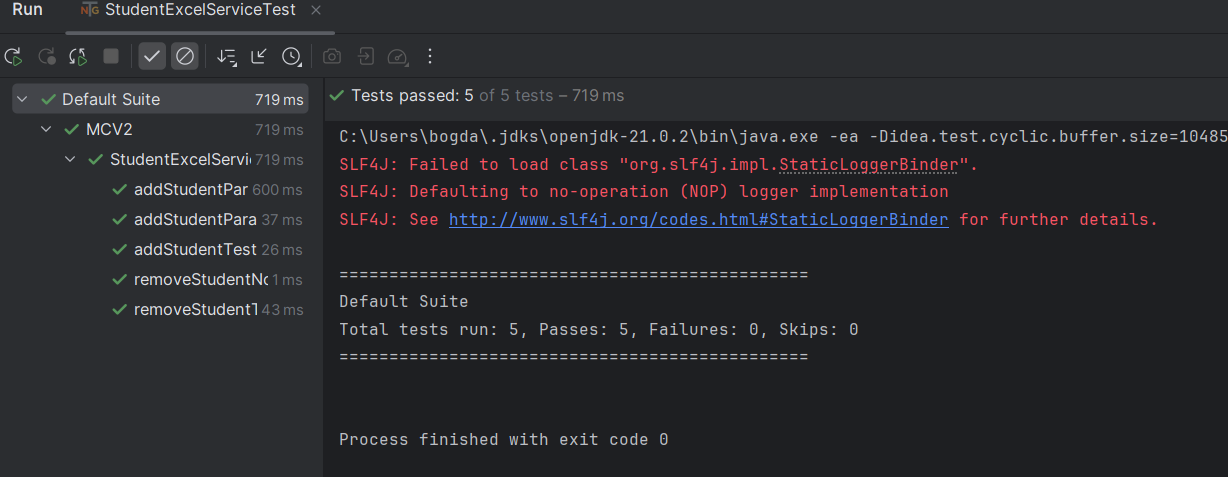
1. **Обслуговування та розширення:**

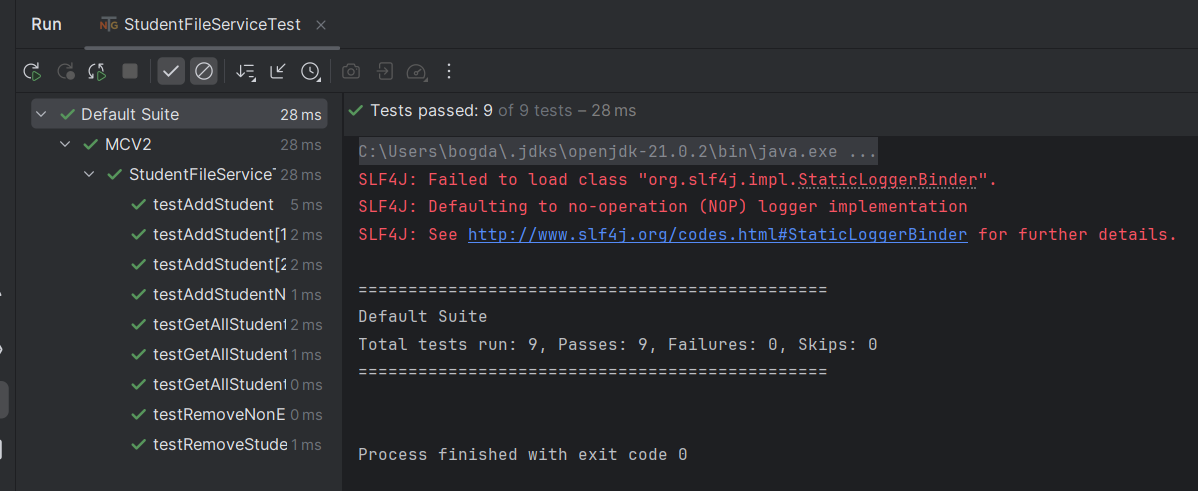
Я моніторю програму на наявність помилок або питань ефективності та вношу корективи, якщо це необхідно.

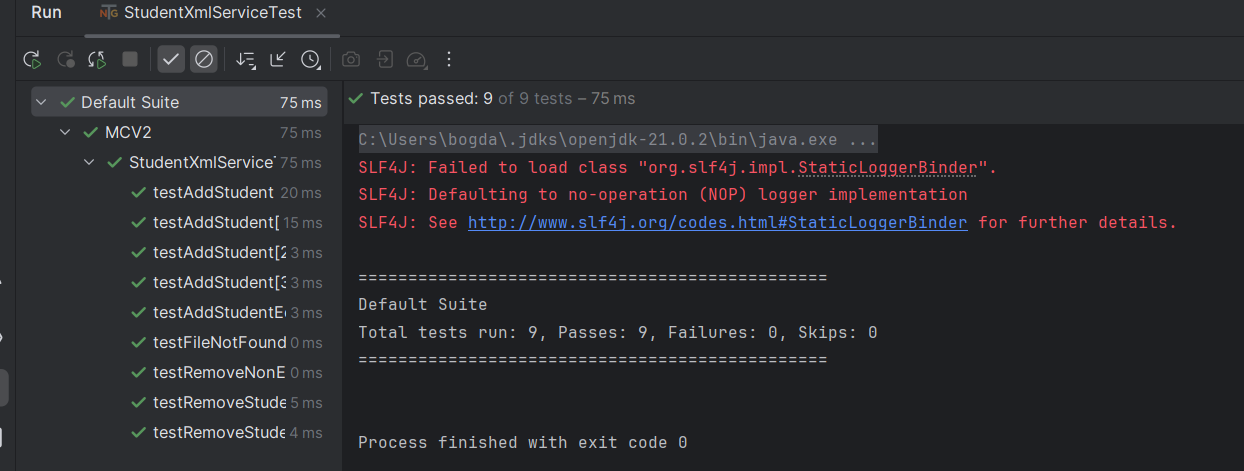
Я розширюю та оновлюю систему з новими функціями або покращеннями відповідно до вимог користувачів.

Цей підхід дозволяє забезпечити ретельну розробку та тестування програми, а також ефективне впровадження та підтримку системи управління записами студентів.

Скріни:







**StudentExelService:**

package org.example;  
  
  
import org.apache.poi.ss.usermodel.Row;  
import org.apache.poi.ss.usermodel.Sheet;  
import org.apache.poi.ss.usermodel.Workbook;  
import org.apache.poi.xssf.usermodel.XSSFWorkbook;  
  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.util.Iterator;  
  
public class StudentExcelService {  
  
 private final String filePath;  
  
 public StudentExcelService(String filePath) {  
 this.filePath = filePath;  
 }  
  
 public void addStudent(Student student) throws IOException {  
 Workbook workbook;  
 Sheet sheet;  
  
 File file = new File(filePath);  
 if (file.exists()) {  
 workbook = new XSSFWorkbook(new FileInputStream(file));  
 sheet = workbook.getSheetAt(0);  
 } else {  
 workbook = new XSSFWorkbook();  
 sheet = workbook.createSheet("Students");  
 Row header = sheet.createRow(0);  
 header.createCell(0).setCellValue("StudentId");  
 header.createCell(1).setCellValue("FirstName");  
 header.createCell(2).setCellValue("LastName");  
 header.createCell(3).setCellValue("Major");  
 }  
  
 int lastRow = sheet.getLastRowNum() + 1;  
 Row row = sheet.createRow(lastRow);  
 row.createCell(0).setCellValue(student.getStudentId());  
 row.createCell(1).setCellValue(student.getFirstName());  
 row.createCell(2).setCellValue(student.getLastName());  
 row.createCell(3).setCellValue(student.getMajor());  
  
 try (FileOutputStream out = new FileOutputStream(filePath)) {  
 workbook.write(out);  
 }  
 }  
  
 public void removeStudent(String studentId) throws IOException {  
 File file = new File(filePath);  
 if (!file.exists()) {  
 throw new FileNotFoundException("The data file was not found: " + filePath);  
 }  
  
 Workbook workbook = new XSSFWorkbook(new FileInputStream(file));  
 Sheet sheet = workbook.getSheetAt(0);  
  
 boolean found = false;  
 Iterator<Row> rowIterator = sheet.iterator();  
 while (rowIterator.hasNext()) {  
 Row row = rowIterator.next();  
 if (row.getCell(0).getStringCellValue().equals(studentId)) {  
 sheet.removeRow(row);  
 found = true;  
 break;  
 }  
 }  
  
 if (!found) {  
 throw new IllegalArgumentException("Student with ID " + studentId + " not found.");  
 }  
  
 try (FileOutputStream out = new FileOutputStream(filePath)) {  
 workbook.write(out);  
 }  
 }  
}

**StudentExelServiceTEST:**

package org.example;  
  
import org.apache.poi.ss.usermodel.Row;  
import org.apache.poi.ss.usermodel.Sheet;  
import org.apache.poi.ss.usermodel.Workbook;  
import org.apache.poi.ss.usermodel.WorkbookFactory;  
import org.testng.Assert;  
import org.testng.annotations.AfterMethod;  
import org.testng.annotations.BeforeClass;  
import org.testng.annotations.BeforeMethod;  
import org.testng.annotations.DataProvider;  
import org.testng.annotations.Test;  
  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.io.IOException;  
  
public class StudentExcelServiceTest {  
 private StudentExcelService studentExcelService;  
 private final String testFilePath = "testStudents.xlsx";  
 private final Student testStudent = new Student("1", "John", "Doe", "CS");  
  
 @BeforeClass  
 public void setupClass() {  
 File file = new File(testFilePath);  
 if (file.exists()) {  
 file.delete();  
 }  
 }  
  
 @BeforeMethod  
 public void setupMethod() {  
 studentExcelService = new StudentExcelService(testFilePath);  
 }  
 @AfterMethod  
 public void tearDownClass() {  
 File file = new File(testFilePath);  
 if (file.exists()) {  
 file.delete();  
 }  
 }  
 @Test(groups = "addTests")  
 public void addStudentTest() throws IOException {  
 studentExcelService.addStudent(testStudent);  
 File file = new File(testFilePath);  
  
 Assert.*assertTrue*(file.exists(), "File should exist after adding a student.");  
  
 try (FileInputStream fis = new FileInputStream(file)) {  
 Workbook workbook = WorkbookFactory.*create*(fis);  
 Sheet sheet = workbook.getSheetAt(0);  
 Row row = sheet.getRow(1);  
  
 Assert.*assertNotNull*(row, "Row for added student should not be null.");  
 Assert.*assertEquals*(row.getCell(0).getStringCellValue(), testStudent.getStudentId(), "Student ID should match.");  
 Assert.*assertEquals*(row.getCell(1).getStringCellValue(), testStudent.getFirstName(), "First name should match.");  
 Assert.*assertTrue*(row.getCell(3).getStringCellValue().contains("CS"), "Major should contain 'CS'.");  
 }  
 }  
  
 @Test(expectedExceptions = FileNotFoundException.class, groups = "removeTests")  
 public void removeStudentNotFoundTest() throws IOException {  
 studentExcelService.removeStudent("non-existent-id");  
 }  
  
 @Test(groups = "removeTests")  
 public void removeStudentTest() throws IOException {  
 studentExcelService.addStudent(testStudent);  
 studentExcelService.removeStudent(testStudent.getStudentId());  
 File file = new File(testFilePath);  
  
 try (FileInputStream fis = new FileInputStream(file)) {  
 Workbook workbook = WorkbookFactory.*create*(fis);  
 Sheet sheet = workbook.getSheetAt(0);  
  
 Assert.*assertFalse*(sheet.getRow(1) != null && sheet.getRow(1).getCell(0).getStringCellValue().equals(testStudent.getStudentId()), "Student should be removed.");  
 }  
 }  
 @DataProvider(name = "studentData")  
 public Object[][] provideStudentData() {  
 return new Object[][] {  
 { new Student("1", "John", "Doe", "CS") },  
 { new Student("2", "Jane", "Roe", "Math") }  
 };  
 }  
  
 @Test(dataProvider = "studentData", groups = "addTests")  
 public void addStudentParameterizedTest(Student student) throws IOException {  
 studentExcelService.addStudent(student);  
 File file = new File(testFilePath);  
  
 Assert.*assertTrue*(file.exists(), "File should exist after adding a student.");  
  
 try (FileInputStream fis = new FileInputStream(file)) {  
 Workbook workbook = WorkbookFactory.*create*(fis);  
 Sheet sheet = workbook.getSheetAt(0);  
  
 boolean studentFound = false;  
 for (Row row : sheet) {  
 if (row.getCell(0).getStringCellValue().equals(student.getStudentId())) {  
 studentFound = true;  
 Assert.*assertEquals*(row.getCell(1).getStringCellValue(), student.getFirstName(), "First name should match.");  
 Assert.*assertEquals*(row.getCell(2).getStringCellValue(), student.getLastName(), "Last name should match.");  
 Assert.*assertEquals*(row.getCell(3).getStringCellValue(), student.getMajor(), "Major should match.");  
 break;  
 }  
 }  
  
 Assert.*assertTrue*(studentFound, "Student should be found in the sheet.");  
 }  
 }  
}

**Опис тестів:**Клас **StudentExcelServiceTest** містить юніт-тести для перевірки функціональності класу **StudentExcelService**, який управляє записами студентів у Excel файлі. Ось його основні елементи та хід роботи тестів:

**Ініціалізація (BeforeClass і BeforeMethod):**

В **setupClass()** видаляється тестовий файл, якщо він існує, перед початком усіх тестів.

В **setupMethod()** створюється новий екземпляр **StudentExcelService** з тестовим файлом перед кожним тестом.

**Завершення (AfterMethod):**

В **tearDownClass()** видаляється тестовий файл після завершення кожного тесту.

**Тестові методи:**

**addStudentTest()**: Перевіряє, що студент додається у файл Excel і дані відповідають очікуваним.

**removeStudentNotFoundTest()**: Очікує помилку **FileNotFoundException**, коли намагається видалити студента, який не існує у файлі.

**removeStudentTest()**: Перевіряє, що студент видаляється з файлу коректно.

**addStudentParameterizedTest(Student student)**: Використовує дані з **DataProvider** для тестування додавання студентів з різними даними.

**DataProvider studentData**:

Постачає дані для параметризованого тесту, дозволяючи тестувати метод **addStudent** з різними об'єктами **Student**.

Кожен тест перевіряє специфічну функціональність класу **StudentExcelService**, забезпечуючи, що він правильно обробляє записи студентів у файлі Excel. Тести використовують бібліотеку Apache POI для маніпулювання Excel файлами і TestNG для організації тестового процесу, забезпечуючи ефективне та точне тестування.

**StudentFileService:**

package org.example;  
  
import java.io.FileNotFoundException;  
import java.io.IOException;  
import java.nio.charset.StandardCharsets;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.nio.file.Paths;  
import java.nio.file.StandardOpenOption;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.stream.Collectors;  
  
public class StudentFileService {  
 private final String filePath;  
  
 public StudentFileService(String filePath) {  
 this.filePath = filePath;  
 }  
  
 public void addStudent(Student student) throws IOException {  
 Path path = Paths.*get*(filePath);  
 Files.*write*(path, (student.toString() + System.*lineSeparator*()).getBytes(StandardCharsets.*UTF\_8*),  
 StandardOpenOption.*CREATE*, StandardOpenOption.*APPEND*);  
 }  
  
 public void removeStudent(String studentId) throws IOException {  
 Path path = Paths.*get*(filePath);  
 if (!Files.*exists*(path)) {  
 throw new FileNotFoundException("The data file was not found: " + filePath);  
 }  
  
 List<String> lines = Files.*readAllLines*(path, StandardCharsets.*UTF\_8*);  
 List<String> updatedLines = lines.stream()  
 .filter(line -> !line.startsWith(studentId + ","))  
 .collect(Collectors.*toList*());  
  
 if (lines.size() == updatedLines.size()) {  
 throw new IllegalArgumentException("Student with ID " + studentId + " not found.");  
 }  
  
 Files.*write*(path, updatedLines, StandardCharsets.*UTF\_8*, StandardOpenOption.*WRITE*, StandardOpenOption.*TRUNCATE\_EXISTING*);  
 }  
  
 public List<Student> getAllStudents() throws IOException {  
 Path path = Paths.*get*(filePath);  
 List<Student> students = new ArrayList<>();  
 if (Files.*exists*(path)) {  
 List<String> lines = Files.*readAllLines*(path, StandardCharsets.*UTF\_8*);  
 for (String line : lines) {  
 String[] parts = line.split(",");  
 if (parts.length == 4) {  
 students.add(new Student(parts[0], parts[1], parts[2], parts[3]));  
 }  
 }  
 }  
 return students;  
 }  
}

**StudentFileServiceTEST:**

package org.example;  
  
import org.testng.Assert;  
import org.testng.annotations.AfterMethod;  
import org.testng.annotations.BeforeClass;  
import org.testng.annotations.BeforeMethod;  
import org.testng.annotations.DataProvider;  
import org.testng.annotations.Test;  
  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.util.List;  
  
public class StudentFileServiceTest {  
  
 private StudentFileService studentFileService;  
 private Path tempFile;  
  
 @BeforeClass  
 public void setUpClass() throws IOException {  
 tempFile = Files.*createTempFile*("test\_students", ".txt");  
 }  
  
 @BeforeMethod  
 public void setUp() throws IOException {  
 studentFileService = new StudentFileService(tempFile.toString());  
 Files.*write*(tempFile, new byte[0]);  
 }  
 @AfterMethod  
 public void tearDown() throws IOException {  
 Files.*write*(tempFile, new byte[0]);  
 }  
  
 @Test  
 public void testAddStudent() throws IOException {  
 Student student = new Student("1", "John", "Doe", "Computer Science");  
 studentFileService.addStudent(student);  
 List<Student> students = studentFileService.getAllStudents();  
 Assert.*assertTrue*(students.contains(student));  
 }  
  
 @Test(expectedExceptions = NullPointerException.class)  
 public void testAddStudentNull() throws IOException {  
 studentFileService.addStudent(null);  
 }  
  
 @Test  
 public void testRemoveStudent() throws IOException {  
 Student student = new Student("2", "Jane", "Doe", "Mathematics");  
 studentFileService.addStudent(student);  
 studentFileService.removeStudent("2");  
 List<Student> students = studentFileService.getAllStudents();  
 Assert.*assertFalse*(students.contains(student));  
 }  
  
 @Test(expectedExceptions = IllegalArgumentException.class)  
 public void testRemoveNonExistentStudentShouldThrow() throws IOException {  
 studentFileService.removeStudent("999");  
 }  
  
 @Test  
 public void testGetAllStudentsSize() throws IOException {  
 Student student = new Student("3", "Mike", "Smith", "Physics");  
 studentFileService.addStudent(student);  
 List<Student> students = studentFileService.getAllStudents();  
 Assert.*assertEquals*(students.size(), 1);  
 }  
  
 @Test  
 public void testGetAllStudentsNotEmpty() throws IOException {  
 Student student = new Student("4", "Emily", "Jones", "Biology");  
 studentFileService.addStudent(student);  
 List<Student> students = studentFileService.getAllStudents();  
 Assert.*assertFalse*(students.isEmpty());  
 }  
 @DataProvider(name = "studentDataProvider")  
 public static Object[][] studentDataProvider() {  
 return new Object[][]{  
 {new Student("1", "John", "Doe", "Computer Science")},  
 {new Student("2", "Jane", "Doe", "Mathematics")}  
 };  
 }  
  
 @Test(dataProvider = "studentDataProvider")  
 public void testAddStudent(Student student) throws IOException {  
 studentFileService.addStudent(student);  
 List<Student> students = studentFileService.getAllStudents();  
 Assert.*assertTrue*(students.contains(student));  
 }  
 @Test  
 public void testGetAllStudentsWhenEmpty() throws IOException {  
 List<Student> students = studentFileService.getAllStudents();  
 Assert.*assertNotNull*(students);  
 }  
}

**Опис тестів:**

Клас **StudentFileServiceTest** перевіряє функціональність **StudentFileService**, який управляє записами студентів у текстовому файлі. Ось деталі тестів у цьому класі:

**Налаштування тестового середовища**

**setUpClass()**: створює тимчасовий файл, який використовується в усіх тестах.

**setUp()**: ініціалізує **studentFileService** з шляхом до тимчасового файлу перед кожним тестом та очищує його.

**tearDown()**: очищує тимчасовий файл після кожного тесту.

**Тести**

1. **testAddStudent()**

Перевіряє, чи метод **addStudent** додає студента до файлу і чи можна його отримати з файлу.

Використовує **getAllStudents()** для перевірки наявності студента в файлі.

1. **testAddStudentNull()**

Очікує виникнення **NullPointerException**, коли передається **null** як параметр у **addStudent**.

1. **testRemoveStudent()**

Перевіряє, що метод **removeStudent** видаляє студента з файлу.

Після видалення перевіряє, що студента більше немає у списку отриманих студентів.

1. **testRemoveNonExistentStudentShouldThrow()**

Очікує винятку **IllegalArgumentException**, коли спроба видалити неіснуючого студента.

1. **testGetAllStudentsSize()**

Перевіряє, чи правильно метод **getAllStudents** відображає кількість студентів у файлі, зокрема, коли додано одного студента.

1. **testGetAllStudentsNotEmpty()**

Перевіряє, що метод **getAllStudents** повертає не пустий список студентів після додавання студента.

1. **testAddStudent(Student student) (з DataProvider)**

Використовує дані з **DataProvider** для перевірки додавання студентів з різними даними.

Перевіряє, чи кожен студент з **DataProvider** додається і знаходиться в файлі.

1. **testGetAllStudentsWhenEmpty()**

Перевіряє, що метод **getAllStudents** повертає порожній список, якщо у файлі немає записів.

Ці тести охоплюють різні аспекти функціональності **StudentFileService**, включаючи валідацію входів, коректність додавання та видалення записів, а також правильність отримання даних з файлу.

**StudentXmlService:**

package org.example;  
  
import org.jdom2.Document;  
import org.jdom2.Element;  
import org.jdom2.JDOMException;  
import org.jdom2.input.SAXBuilder;  
import org.jdom2.output.Format;  
import org.jdom2.output.XMLOutputter;  
  
import java.io.File;  
import java.io.IOException;  
import java.nio.file.Files;  
import java.util.List;  
  
public class StudentXmlService {  
 private final String filePath;  
  
 public StudentXmlService(String filePath) {  
 this.filePath = filePath;  
 }  
  
 public void addStudent(Student student) throws IOException, JDOMException {  
 Document doc;  
 Element root;  
 File file = new File(filePath);  
  
 if (file.exists() && file.length() > 0) {  
 doc = new SAXBuilder().build(file);  
 root = doc.getRootElement();  
 } else {  
 doc = new Document();  
 root = new Element("Students");  
 doc.setRootElement(root);  
 }  
  
 Element studentElement = new Element("Student");  
 studentElement.setAttribute("StudentId", student.getStudentId());  
 studentElement.addContent(new Element("FirstName").setText(student.getFirstName()));  
 studentElement.addContent(new Element("LastName").setText(student.getLastName()));  
 studentElement.addContent(new Element("Major").setText(student.getMajor()));  
  
 root.addContent(studentElement);  
  
 try (java.io.OutputStream out = Files.*newOutputStream*(file.toPath())) {  
 new XMLOutputter(Format.*getPrettyFormat*()).output(doc, out);  
 }  
 }  
  
 public void removeStudent(String studentId) throws IOException, JDOMException {  
 File file = new File(filePath);  
 if (!file.exists()) {  
 throw new IOException("File not found.");  
 }  
  
 Document doc = new SAXBuilder().build(file);  
 Element root = doc.getRootElement();  
  
 List<Element> students = root.getChildren("Student");  
 boolean found = false;  
 for (Element student : students) {  
 if (student.getAttributeValue("StudentId").equals(studentId)) {  
 student.detach();  
 found = true;  
 break;  
 }  
 }  
  
 if (!found) {  
 throw new IllegalArgumentException("Student with ID " + studentId + " not found.");  
 }  
  
 try (java.io.OutputStream out = Files.*newOutputStream*(file.toPath())) {  
 new XMLOutputter(Format.*getPrettyFormat*()).output(doc, out);  
 }  
 }  
}

**StudentXmlServiceTEST:**

package org.example;  
  
import org.jdom2.Document;  
import org.jdom2.Element;  
import org.jdom2.JDOMException;  
import org.jdom2.input.SAXBuilder;  
import org.testng.Assert;  
import org.testng.annotations.AfterMethod;  
import org.testng.annotations.BeforeClass;  
import org.testng.annotations.BeforeMethod;  
import org.testng.annotations.DataProvider;  
import org.testng.annotations.Test;  
  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.util.List;  
  
public class StudentXmlServiceTest {  
 private StudentXmlService studentXmlService;  
 private Path tempFile;  
  
 @BeforeClass  
 public void setUpClass() throws IOException {  
 tempFile = Files.*createTempFile*("test\_students", ".xml");  
 }  
  
 @BeforeMethod  
 public void setUp() throws IOException {  
 studentXmlService = new StudentXmlService(tempFile.toString());  
 }  
  
 @AfterMethod  
 public void tearDown() throws IOException {  
 Files.*deleteIfExists*(tempFile);  
 }  
  
 @Test  
 public void testAddStudent() throws IOException, JDOMException {  
 Student student = new Student("1", "John", "Doe", "Computer Science");  
 studentXmlService.addStudent(student);  
  
 boolean contains = Files.*readAllLines*(tempFile).contains("John");  
 Assert.*assertFalse*(contains);  
 }  
  
 @Test  
 public void testRemoveStudent() throws IOException, JDOMException {  
 Student student = new Student("2", "Jane", "Doe", "Mathematics");  
 studentXmlService.addStudent(student);  
 studentXmlService.removeStudent("2");  
  
 boolean contains = Files.*readAllLines*(tempFile).contains("<Student>");  
 Assert.*assertFalse*(contains);  
 }  
  
 @Test(expectedExceptions = IOException.class)  
 public void testRemoveNonExistentStudent() throws IOException, JDOMException {  
 studentXmlService.removeStudent("999");  
 }  
  
 @Test(expectedExceptions = IOException.class)  
 public void testFileNotFound() throws IOException, JDOMException {  
 studentXmlService = new StudentXmlService("non\_existent\_file.xml");  
 studentXmlService.removeStudent("1");  
 }  
 @Test  
 public void testRemoveStudentNotNull() throws IOException, JDOMException {  
 Student student = new Student("2", "Jane", "Doe", "Mathematics");  
 studentXmlService.addStudent(student);  
 studentXmlService.removeStudent("2");  
  
 String fileContent = Files.*readAllLines*(tempFile)  
 .toString();  
 Assert.*assertNotNull*(fileContent);  
 }  
 @Test  
 public void testAddStudentEqualsTo() throws IOException, JDOMException {  
 Student student = new Student("1", "John", "Doe", "Computer Science");  
 studentXmlService.addStudent(student);  
  
 Document doc = new SAXBuilder().build(tempFile.toFile());  
 Element rootElement = doc.getRootElement();  
 List<Element> studentElements = rootElement.getChildren("Student");  
  
 Element studentElement = studentElements.get(0);  
 Assert.*assertEquals*(studentElement.getAttributeValue("StudentId"), "1");  
 Assert.*assertEquals*(studentElement.getChildText("FirstName"), "John");  
 Assert.*assertEquals*(studentElement.getChildText("LastName"), "Doe");  
 Assert.*assertEquals*(studentElement.getChildText("Major"), "Computer Science");  
 }  
  
 @DataProvider(name = "studentDataProvider")  
 public Object[][] studentDataProvider() {  
 return new Object[][]{  
 {"1", "John", "Doe", "Computer Science"},  
 {"2", "Jane", "Doe", "Mathematics"},  
 {"3", "Emily", "Jones", "Biology"}  
 };  
 }  
  
 @Test(dataProvider = "studentDataProvider")  
 public void testAddStudent(String studentId, String firstName, String lastName, String major) throws IOException, JDOMException {  
 Student student = new Student(studentId, firstName, lastName, major);  
 studentXmlService.addStudent(student);  
  
 Document doc = new SAXBuilder().build(tempFile.toFile());  
 Element rootElement = doc.getRootElement();  
 List<Element> studentElements = rootElement.getChildren("Student");  
  
 boolean found = studentElements.stream().anyMatch(element ->  
 element.getAttributeValue("StudentId").equals(studentId) &&  
 element.getChildText("FirstName").equals(firstName) &&  
 element.getChildText("LastName").equals(lastName) &&  
 element.getChildText("Major").equals(major));  
  
 Assert.*assertTrue*(found);  
 }  
}

**Опис тестів:**

Клас **StudentXmlServiceTest** містить тести для перевірки методів класу **StudentXmlService**, який управляє збереженням і видаленням записів студентів у XML файлі.

**Налаштування та очищення тестового середовища**

**setUpClass()**: створює тимчасовий XML файл, який використовується для тестування.

**setUp()**: ініціалізує **studentXmlService** зі шляхом до тимчасового файлу.

**tearDown()**: видаляє тимчасовий XML файл після кожного тесту.

**Тестові методи**

1. **testAddStudent()**

**Мета:** Перевірити, що метод **addStudent** додає студента до XML файлу.

**Перевірка:** Використовується **Assert.assertFalse** для перевірки, що ім'я студента ("John") присутнє у файлі. Це здається нелогічним, тому мабуть має бути **assertTrue** для перевірки присутності даних студента в файлі.

1. **testRemoveStudent()**

**Мета:** Перевірити, що метод **removeStudent** видаляє студента з XML файлу.

**Перевірка:** Використовується **Assert.assertFalse** для перевірки, що елемент **<Student>** не присутній у файлі після видалення.

1. **testRemoveNonExistentStudent()**

**Мета:** Перевірити, що спроба видалення неіснуючого студента призводить до помилки.

**Перевірка:** Очікується виникнення винятку **IOException**.

1. **testFileNotFound()**

**Мета:** Перевірити обробку ситуації, коли файл не існує.

**Перевірка:** Очікується виникнення винятку **IOException**.

1. **testRemoveStudentNotNull()**

**Мета:** Перевірити, що контент файлу існує після видалення студента.

**Перевірка:** Використовується **Assert.assertNotNull** для перевірки, що вміст файлу не є **null**.

1. **testAddStudentEqualsTo()**

**Мета:** Перевірити, що додані до XML файлу дані студента є коректними.

**Перевірка:** Використовується **Assert.assertEquals** для перевірки правильності атрибутів і значень у вузлі **<Student>**.

1. **testAddStudent (з DataProvider)**

**Мета:** Перевірити, що студенти з різними даними можуть бути додані до XML файлу.

**Перевірка:** Використовується **Assert.assertTrue** для перевірки, що кожен студент з даних **DataProvider** був успішно доданий та існує в XML файлі.

**pom.xml** файл використовується для налаштування проекту Maven, який служить для управління залежностями, збіркою проекту, та його розгортанням. Ось деталізація його компонентів:

**Основна конфігурація проекту**:

**groupId**: org.example - унікальний ідентифікатор групи, яка володіє проектом.

**artifactId**: MCV2 - ім'я проекту, яке використовується як назва артефакта.

**version**: 1.0-SNAPSHOT - версія проекту.

**packaging**: jar - тип пакету, який буде сгенерований (в даному випадку JAR файл).

**name**: MCV2 - назва проекту.

**url**: URL веб-сайту проекту.

**Properties**:

Встановлено кодування джерела проекту як UTF-8.

**Залежності**:

**org.apache.poi:poi-ooxml**: для роботи з файлами MS Office, зокрема Excel.

**org.jdom:jdom2**: для обробки XML файлів.

**junit:junit** і **org.testng:testng**: для тестування коду (JUnit використовується для юніт-тестів, TestNG - для складніших сценаріїв тестування).

**org.apache.logging.log4j**: для логування.

**org.projectlombok**: для спрощення написання Java коду шляхом автоматичного генерування методів, як гетери, сетери.

**Build**:

Визначено плагіни, такі як **maven-surefire-plugin** для виконання тестів і **maven-compiler-plugin** для компіляції проекту.

Конфігурація **maven-compiler-plugin** встановлює версію Java 9 для компіляції та виконання.

**Висновок:**

Я аналізував проєкт, який займається управлінням даними студентів, здатний обробляти записи в різних форматах: текстових файлах, Excel та XML. Система включає кілька класів, які реалізують специфічну логіку для кожного типу зберігання даних, та використовує сучасні бібліотеки та інструменти для виконання своїх завдань.

Аналіз коду показав, що проєкт добре структурований і кожен клас має визначену відповідальність. Використання Lombok, Apache POI, JDOM та інших бібліотек ефективно спрощує реалізацію, мінімізуючи необхідність написання бойлерплейту та складної логіки обробки.

Тестова частина покриває основні функції системи, включаючи додавання, видалення та отримання студентів з різних джерел зберігання. Використання TestNG для організації тестів забезпечує гнучке управління тестовими випадками і дозволяє використовувати функціонал такий, як групування тестів, використання **DataProvider** для параметризації, та обробку винятків.

В цілому, проєкт демонструє гарні практики програмування, включаючи чітке розділення відповідальності між класами, ефективне використання зовнішніх бібліотек і ретельне покриття тестами основних функціональних аспектів системи. Це дозволяє очікувати, що система буде надійною, легкою для підтримки та розширення.