Lesson 06 Demo 01

Utilizing File Methods in Java

Objective: To utilize file methods in Java and efficiently handle file operations within a directory

Tools required: Eclipse IDE

Prerequisites: None

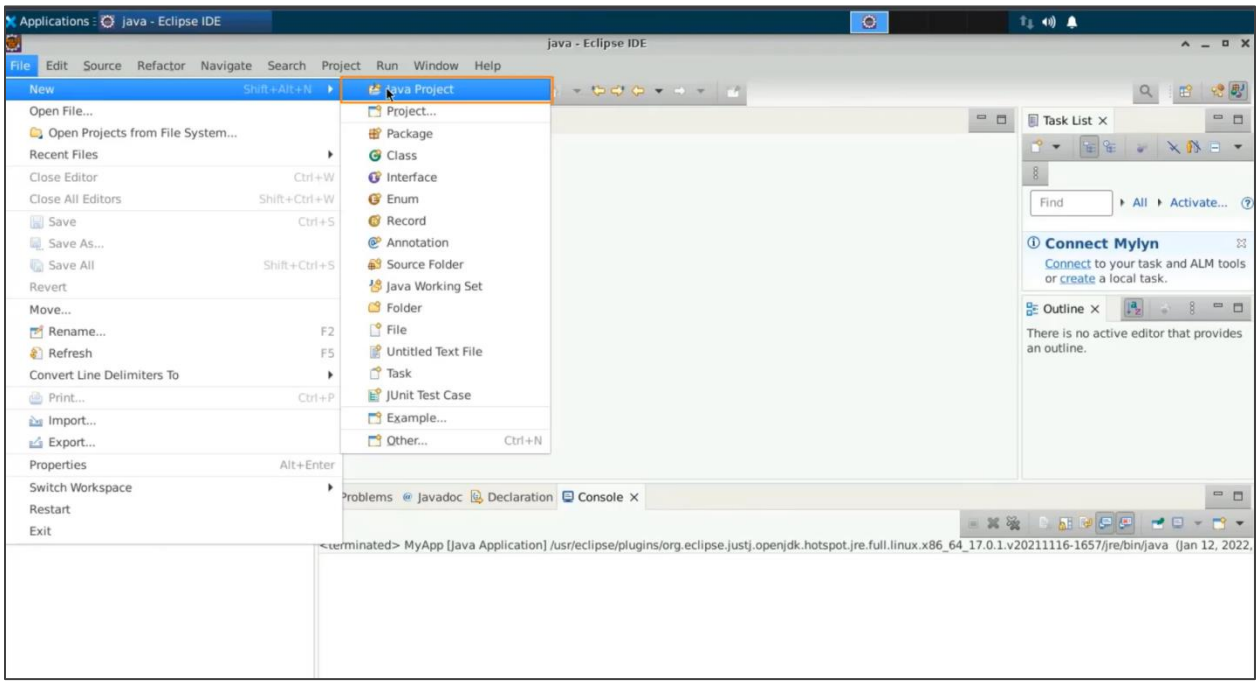
Steps to be followed:

1. Open the Eclipse IDE and create a new Java project
2. Create a temporary file in a directory
3. Use the try and catch block
4. Create another path for the file in the temporary directory path
5. Write the string content
6. Use this static method to directly write the string and execute the code

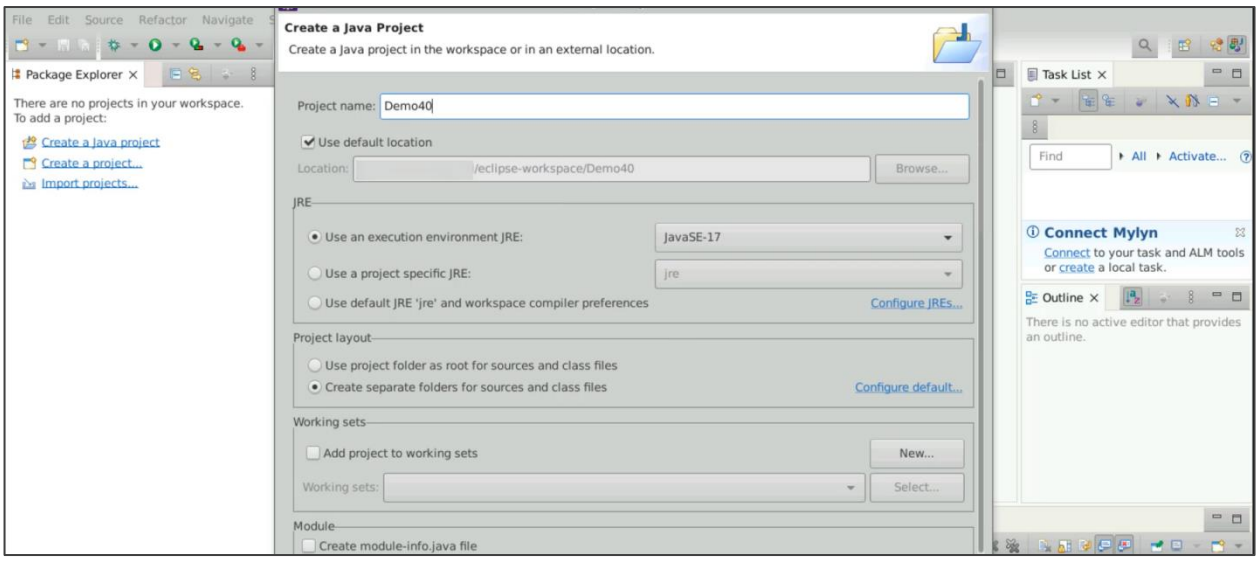
Step 1: Open the Eclipse IDE and create a new Java project

1.1 Open the **Eclipse IDE**

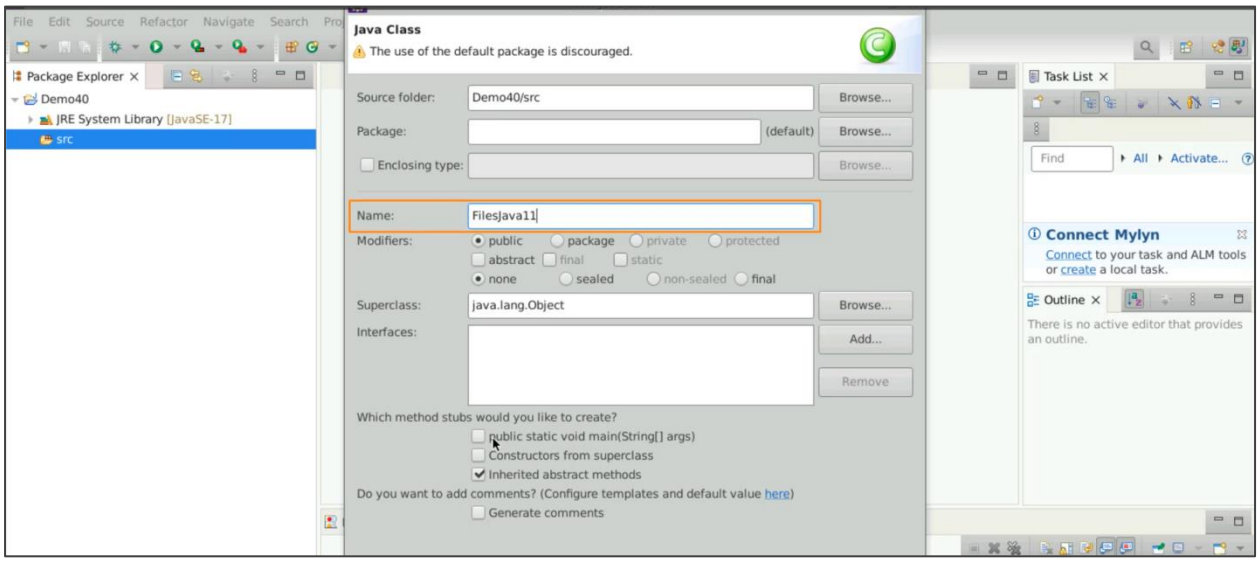
1.2 Select **File**, then **New,** and then **Java project**



1.3 Name the project **Demo40,** uncheck **Create a module-info.java file**, and press **Finish**

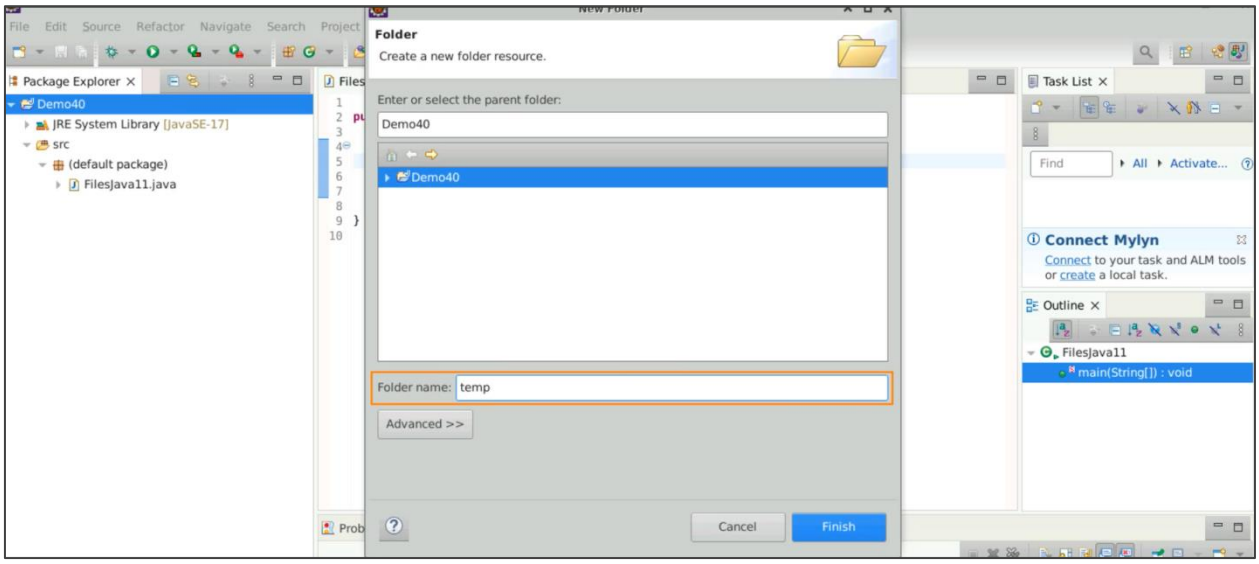


1.3 With **Demo40** selected, right-click on the **src**, and create a **new class**. Name this class **FilesJava11**, select the **main method,** and then select **finish**



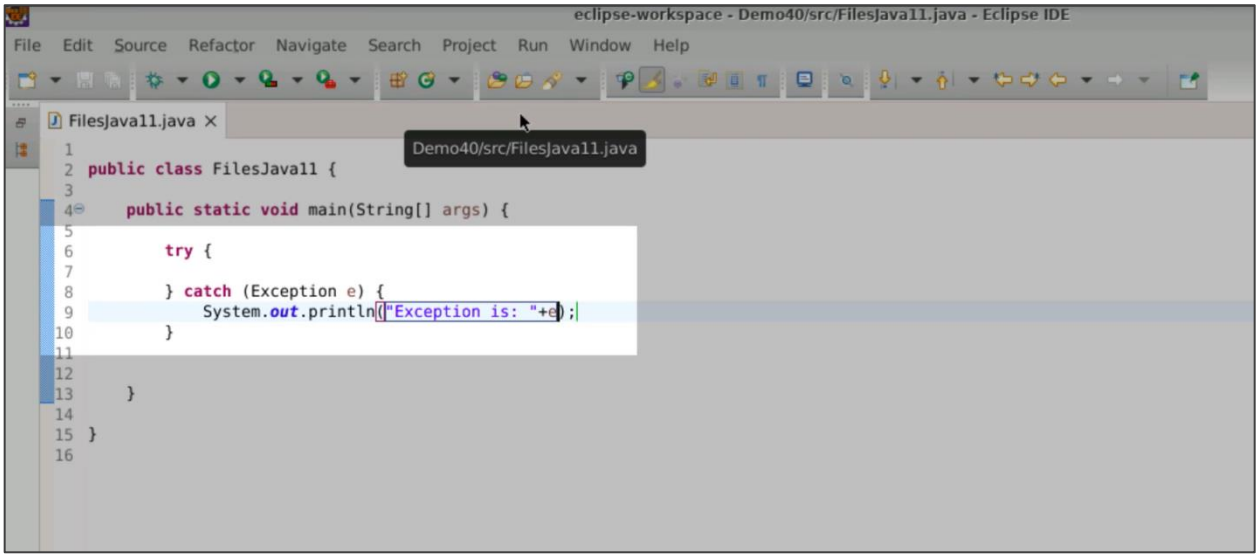
Step 2: Create a temporary file in a directory

2.1 Create a directory in the package. Select the project and create a new folder named **temp**, which stands for temporary and can contain some files

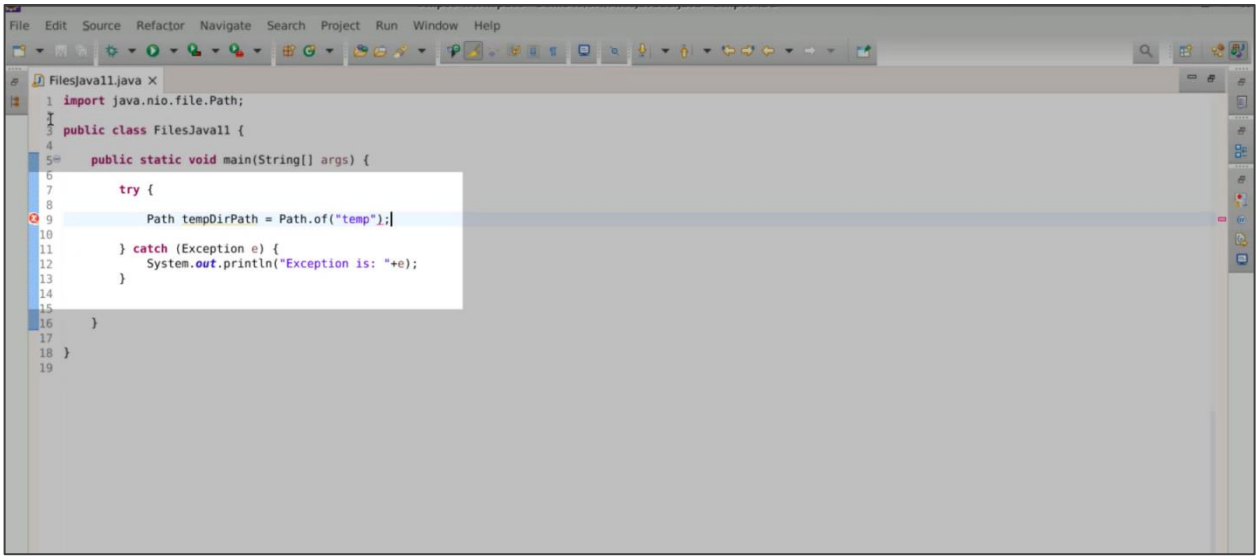


Step 3: Use the try and the catch block

3.1 To handle potential IO exceptions, it is advised to wrap your file IO operations within a try-catch block, as many APIs may throw IO exceptions, ensuring proper exception handling

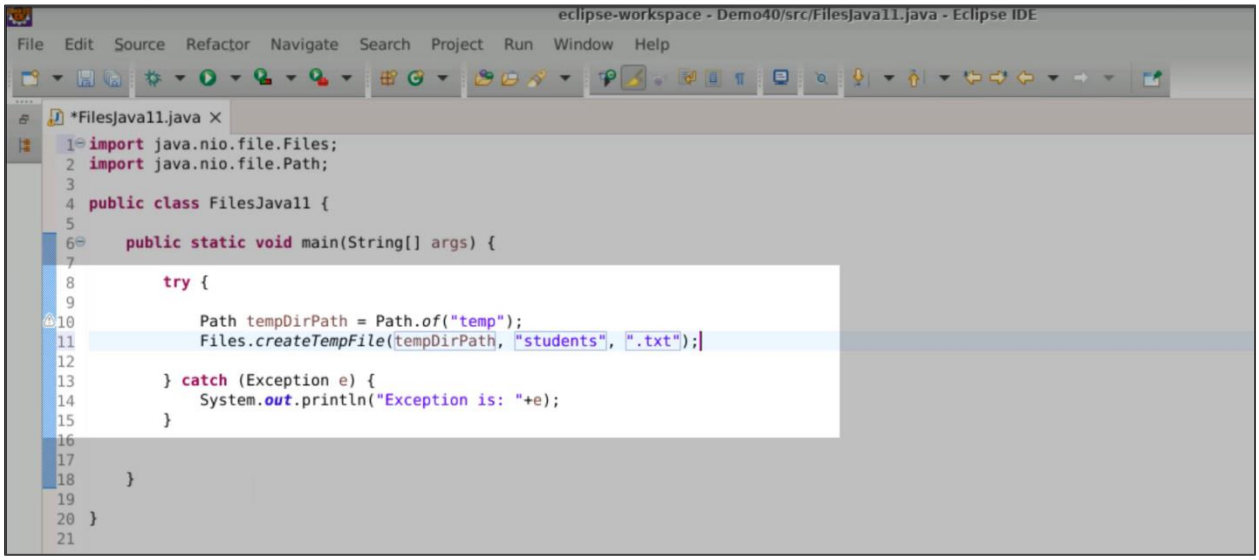


3.2 Utilize the **Java.io.file** API's **Path** method to create a temporary directory path using the static method **File.createTempDirectory** and provide the name of the directory as the input

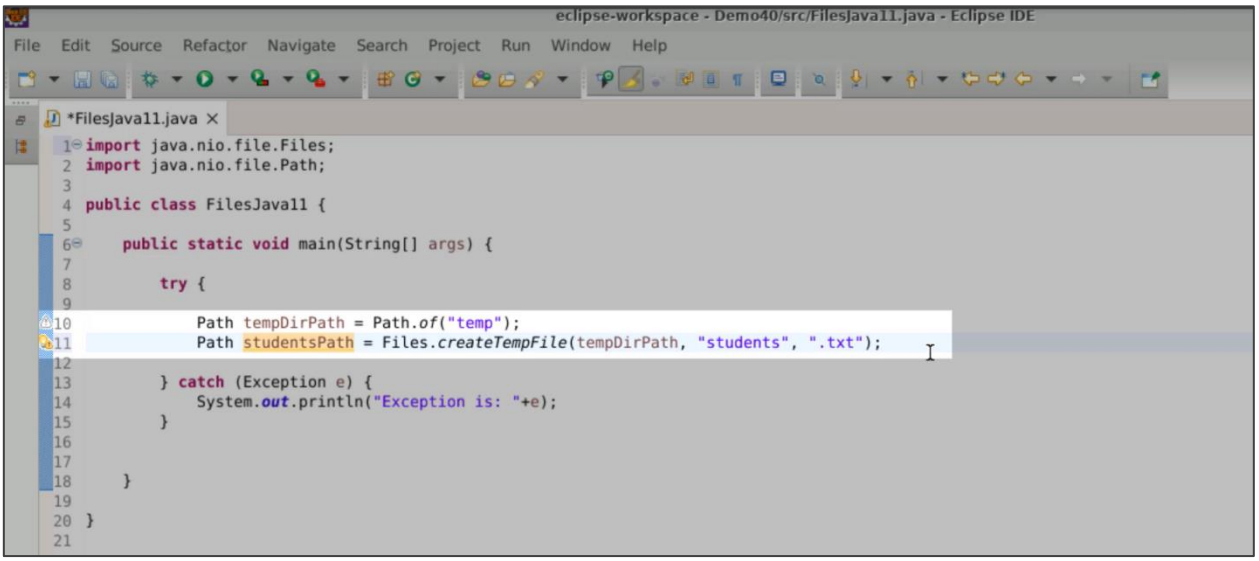


Step 4: Create one more path for the file in the temporary directory path

4.1 Create another path for a file within the temporary directory using the **Java.io.files createTempFile** method, providing the directory path as the temporary directory path and setting the filename as **students** with a **.txt** extension

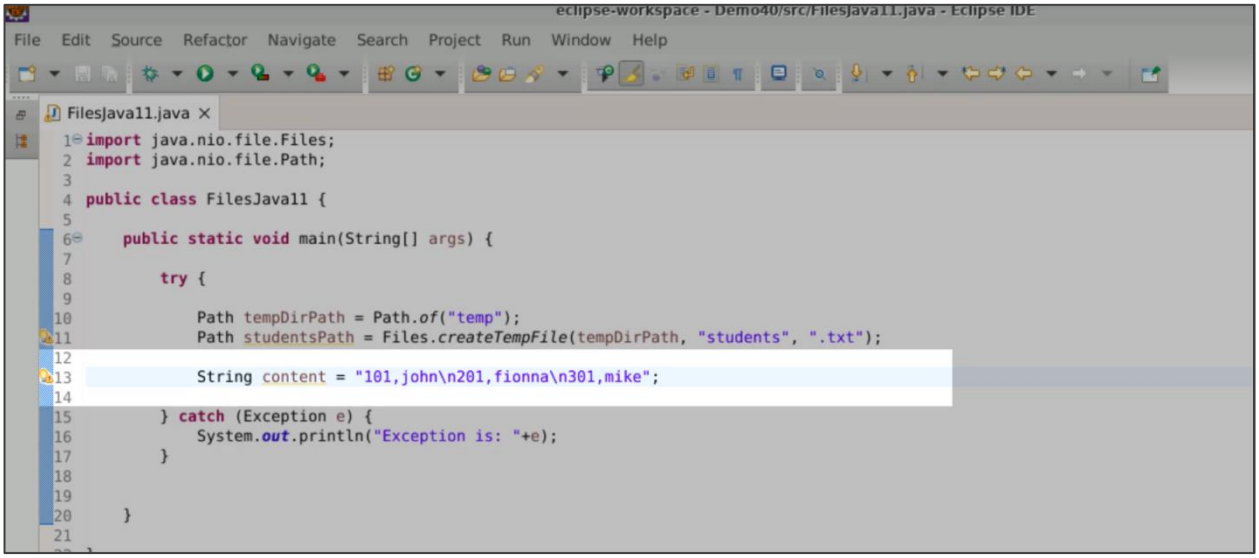


4.2 Store it in one of the other paths, which is called **studentsPath**. This is the name of your temporary file on the path



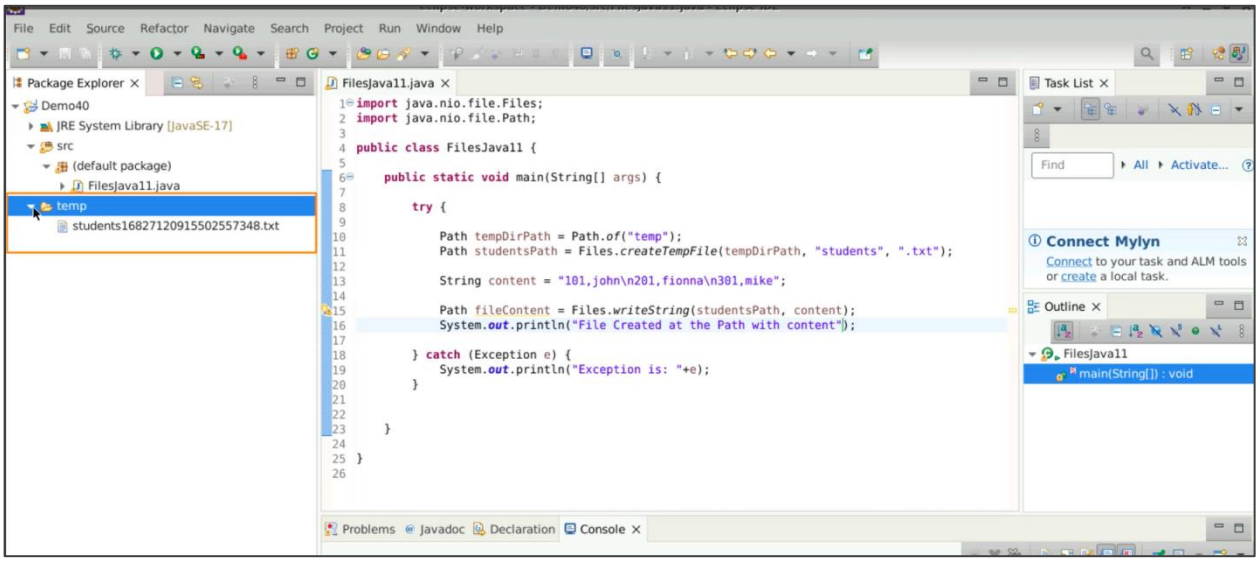
Step 5: Write the string content

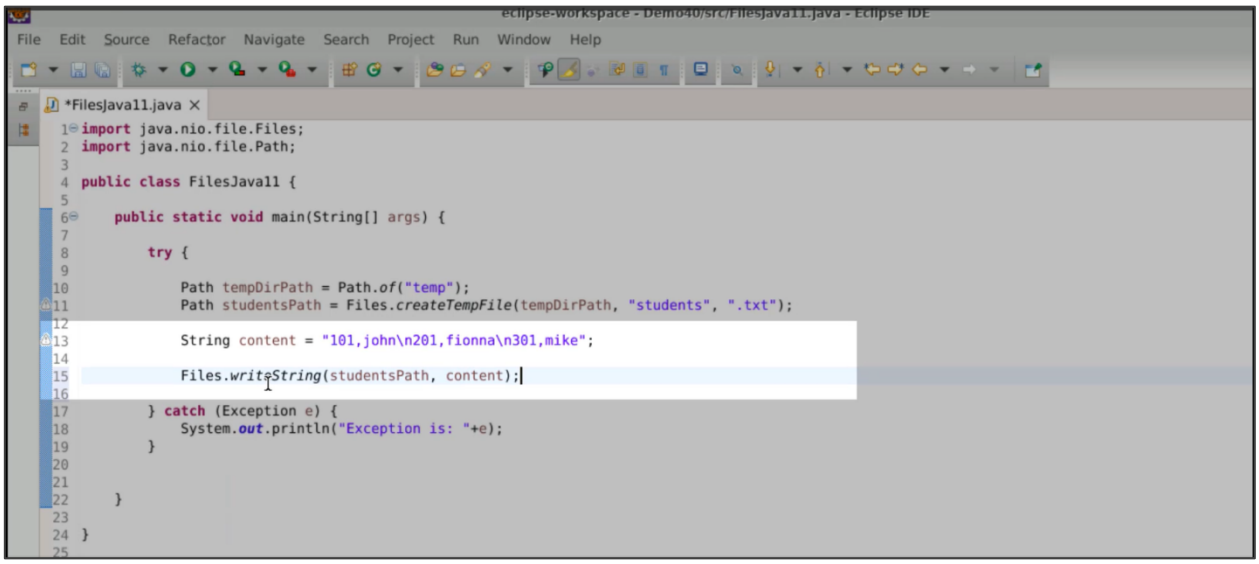
5.1 The content to be written into the file can be defined as a string with comma-separated values, including names like **John** and **Fionna** along with corresponding roll numbers like **101** and **201**, and additional records like **301, mike** with newlines



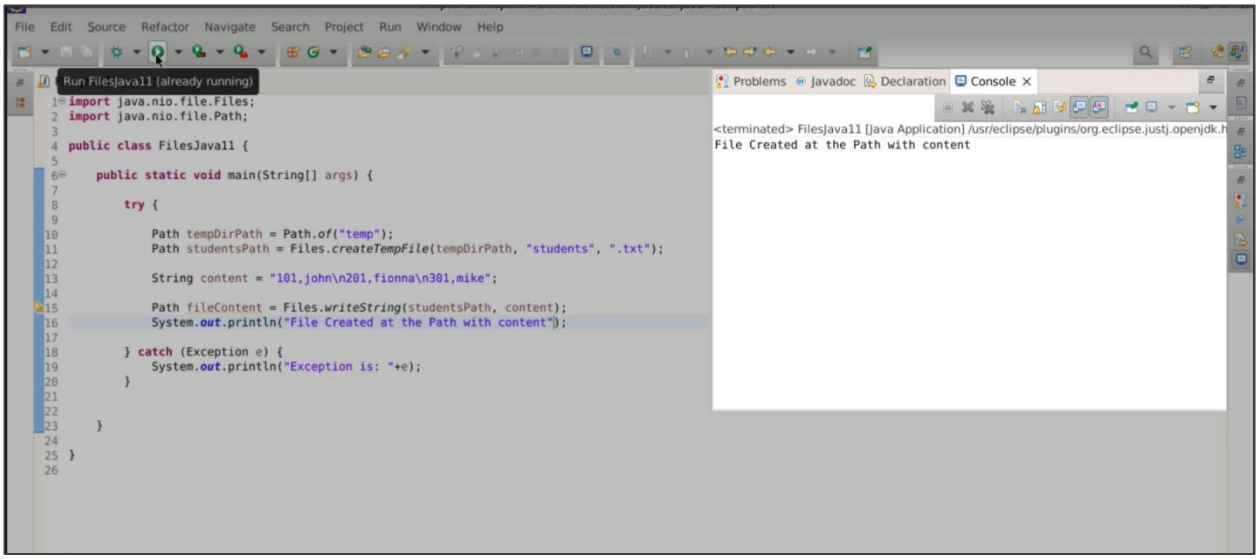
Step 6: Use this static method, to directly write the string and executing the code

6.1 To write the content directly onto the **students.txt** file, use the static method **writeString** on the **Files** class, providing the **studentPath** and the desired content as input

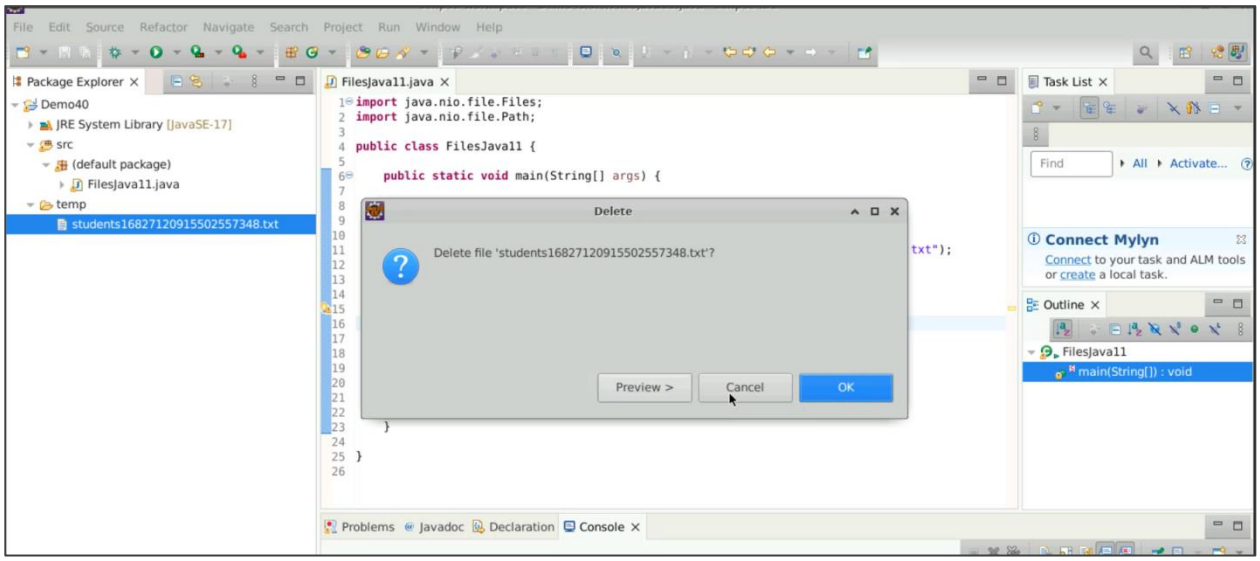




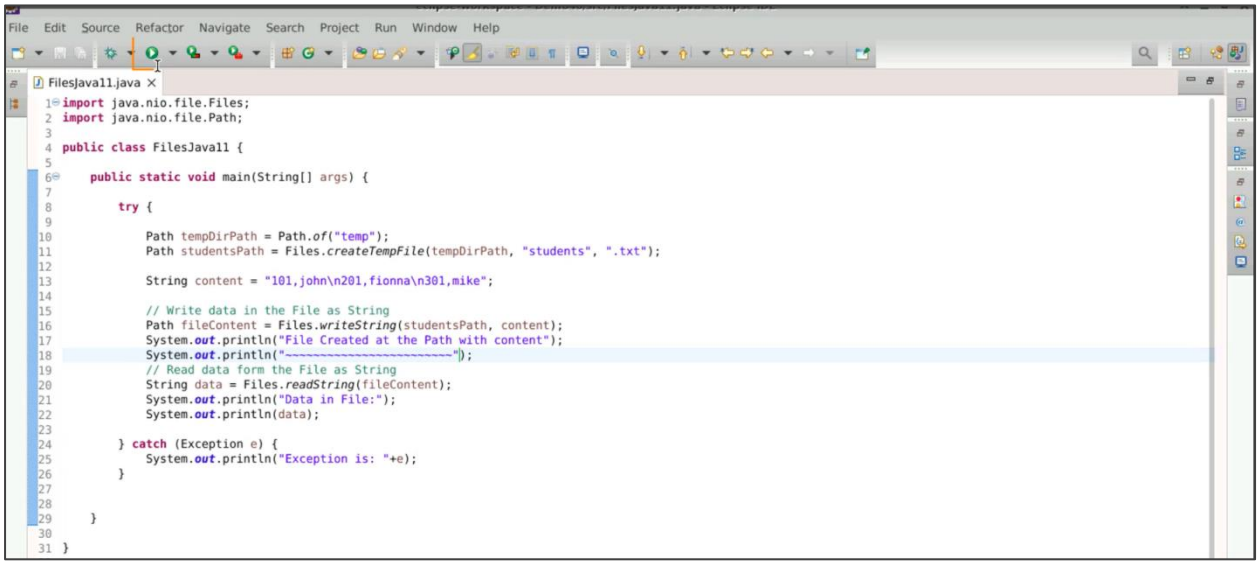
6.2 Let us run the code. When you run the program, it shows that **File Created at the Path with the content**



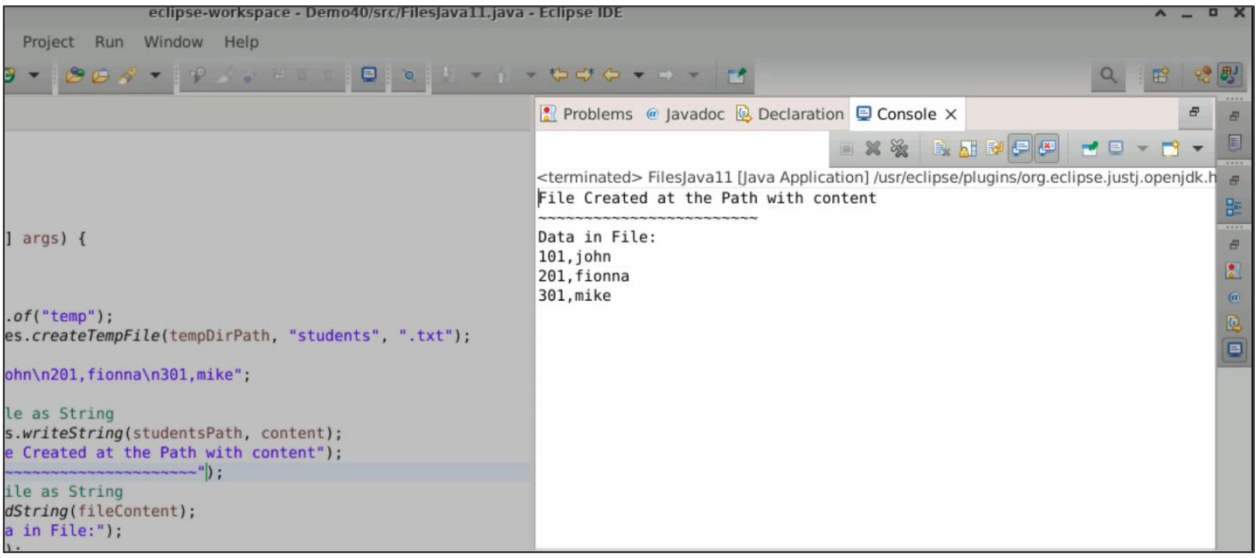
6.3 For simplicity, first remove this file. Now, there is no file. Write a string data, **files.readString** from this file content path



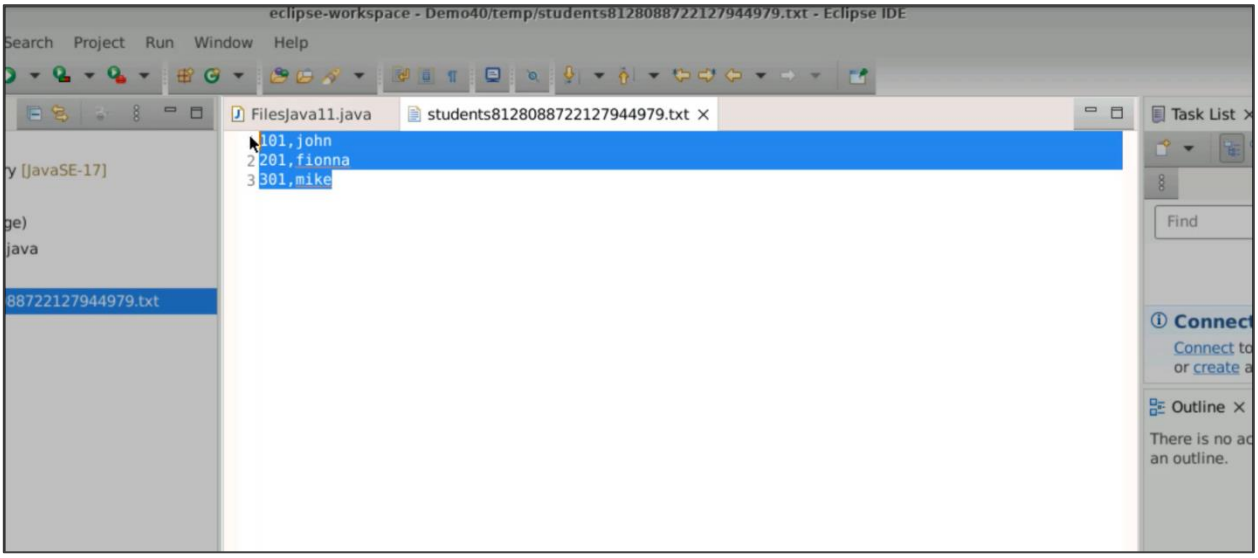
6.4 Execute the static method **readString** from Java 11 to read the contents of the file as a string, and print the retrieved data using **System.out.println**



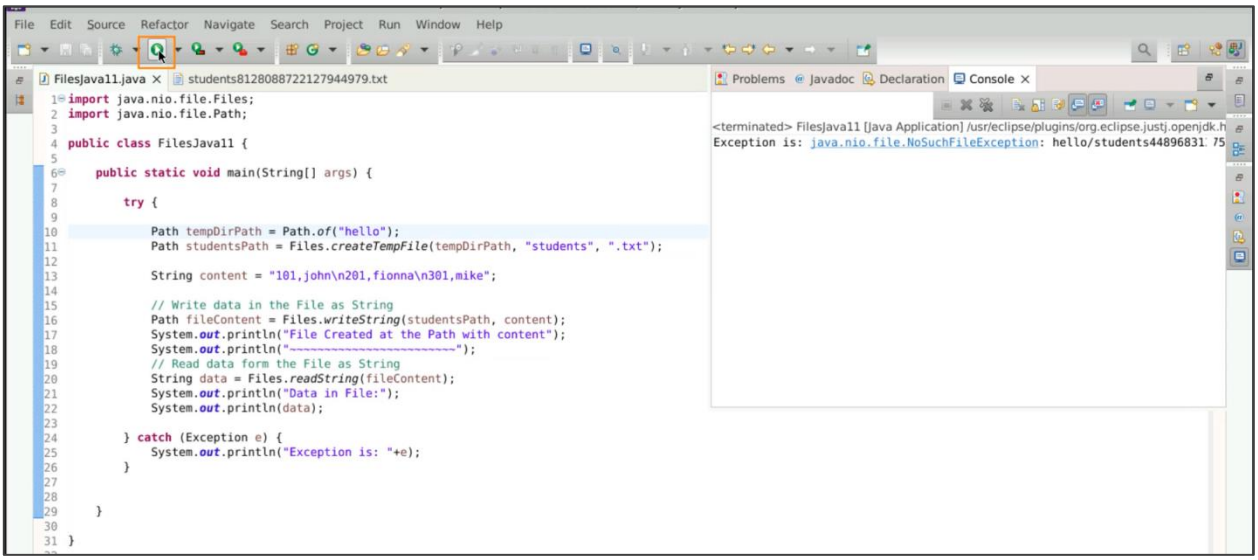
6.5 Let us run the code to see what happens when you save the file first and then try to read it. Add a small delimiter in between. As you can see, the data in the file is 101 John, 201 Fiona, and 301 Mike, which is the same data you wrote.



6.6 When you come back here and refresh on the temporary directory, you will see your file



6.7 If you provide a non-existent directory path like **hello**, it will throw a **NoSuchFileException**. It is important to create the directory explicitly before working with it; otherwise, the directory cannot be automatically created.



Remember this small prerequisite: The Java 11 feature **writeString** can write the content and **readString** can read the content. However, these methods are static and only work with the strings.

By following these steps, you have successfully utilized file methods in Java and efficiently handled file operations within a directory.