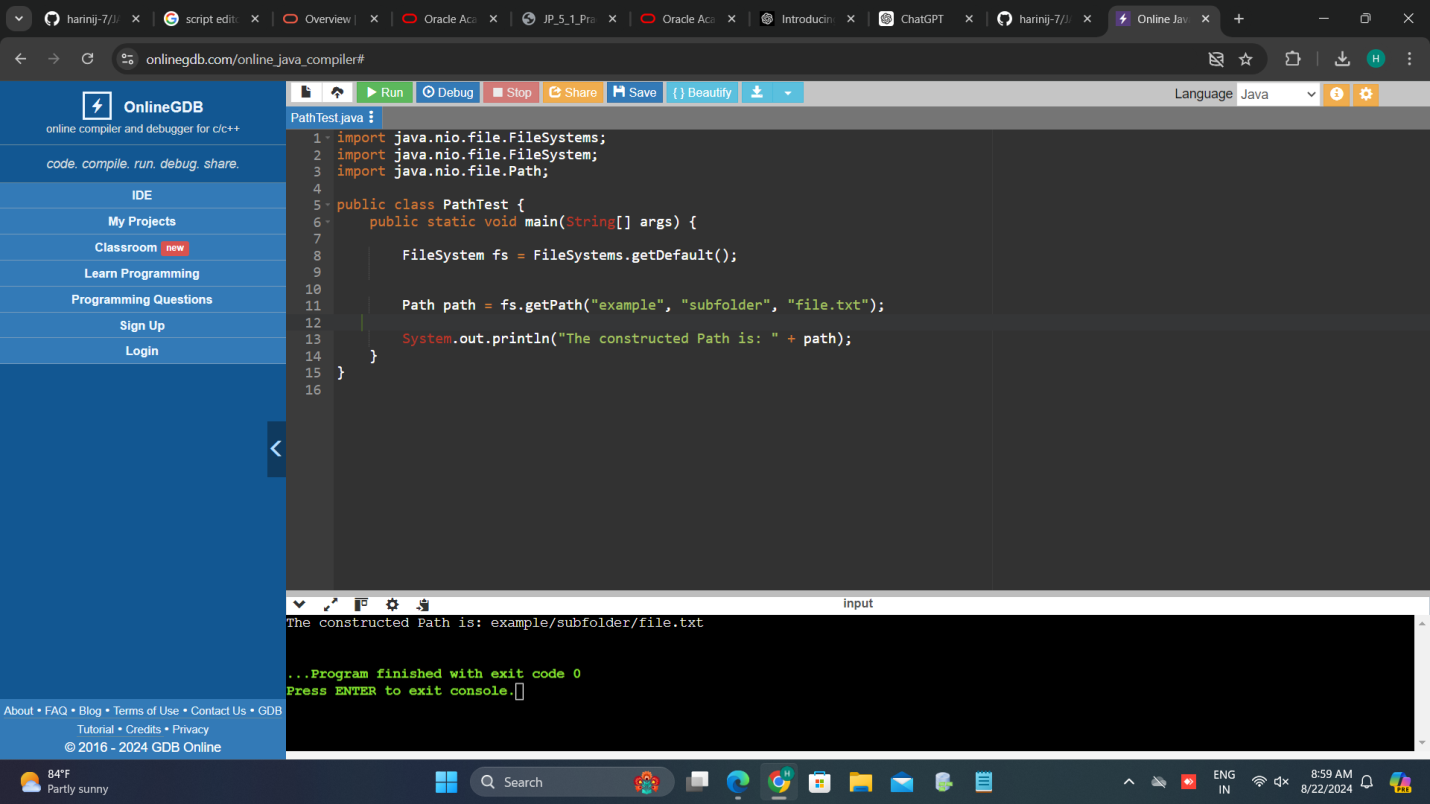
# JAVA PRACTICE PROGRAM

# MODULE-3(5.1)

1. To create a class with a static **main** method that tests the ability to resolve and print a **Path**, follow these steps:

1. Import the necessary packages.
2. Create an instance of a **FileSystem** class.
3. Create a **Path** instance using the **FileSystem**.
4. Print the **Path** using **System.out.println()**.



**2. Identify the main limitations of the Java.io Package**.

The `java.io` package in Java, while foundational for input and output operations, has several limitations:

**1. Lack of Flexibility with Character Encoding:**

- The `java.io` package primarily uses the platform's default character encoding, which can lead to issues when dealing with files or streams that use different encodings. For more control over character encoding, the `java.nio.charset` package provides more flexibility with `Charset` and `CharsetDecoder`.

**2. Performance Limitations:**

- The `java.io` package operates with streams and readers/writers which can sometimes lead to less efficient I/O operations compared to the `java.nio` package. The `java.nio` package (New I/O) offers better performance through its buffer-based I/O and support for non-blocking I/O.

**3. Limited Support for Asynchronous I/O:**

- The `java.io` package does not support asynchronous or non-blocking I/O operations natively. For non-blocking I/O, you need to use the `java.nio` package, which provides `Selectors` and `Channels` for such operations.

***4. No Direct Support for Memory-Mapped Files:***

- The `java.io` package lacks support for memory-mapped files, which allows you to map a file directly into memory and perform I/O operations directly on it. This feature is available in the `java.nio` package via `MappedByteBuffer`.

**5. Less Modern API:**

- The `java.io` package's API is considered less modern compared to `java.nio`. The `java.nio` package introduced buffers, channels, and selectors, offering more advanced and efficient ways to handle I/O.

**6. Limited File System Operations:**

- Basic file system operations such as creating directories or handling file attributes are somewhat limited compared to the `java.nio.file` package, which provides a more comprehensive and flexible file I/O API.

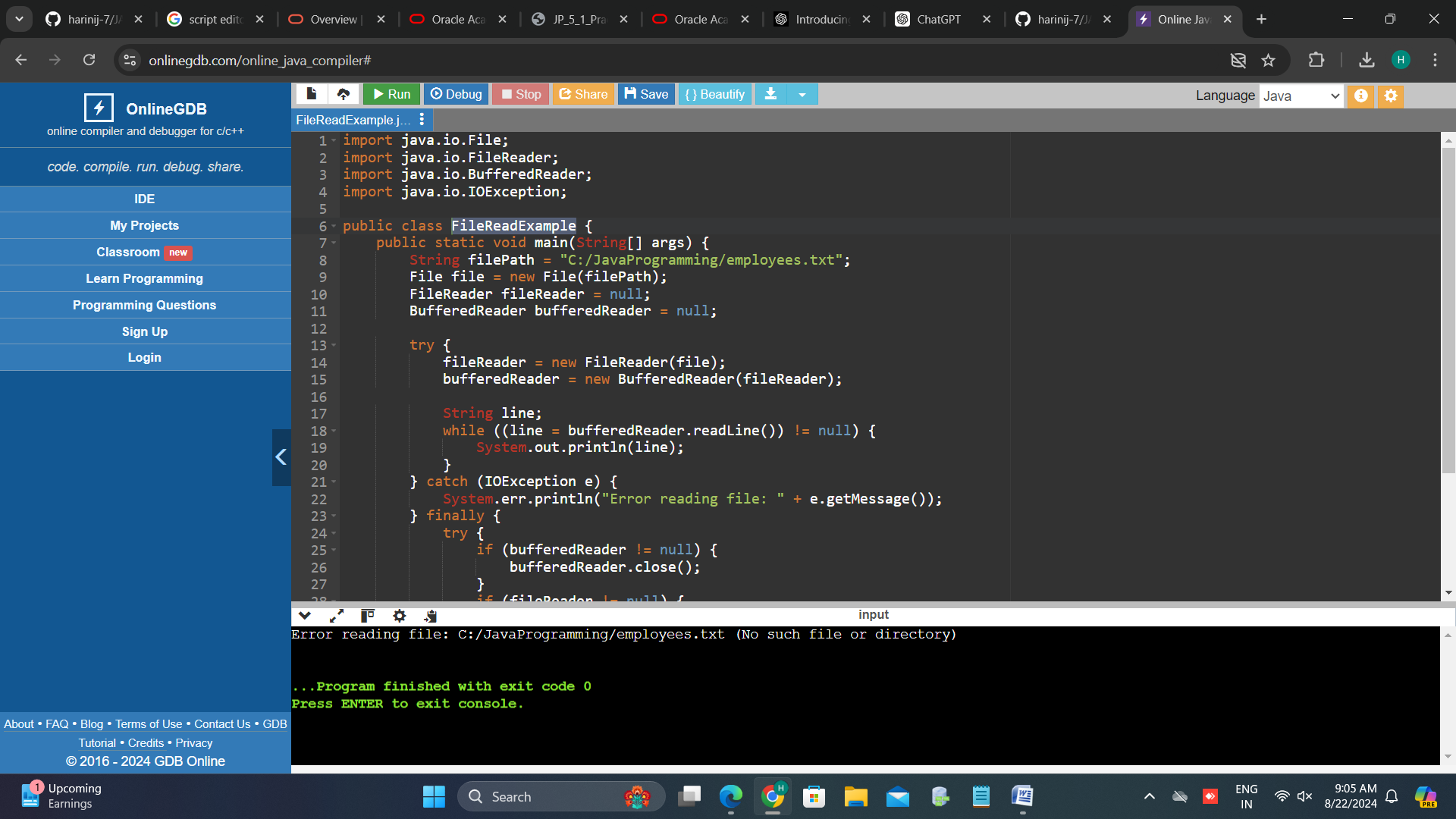
**7. Error Handling:**

- Error handling in `java.io` can be less granular, as many methods throw `IOException` for a wide range of errors. The `java.nio` package allows for more specific exceptions, helping developers handle different types of I/O errors more effectively.

8. Object Serialization Issues:

- The `java.io` package's serialization mechanism is powerful but can be complex and error-prone. It also has performance and security implications. The `java.io` package relies on `Serializable` and `Externalizable`, which can be less intuitive and more error-prone compared to modern serialization frameworks.

**3.**



# Explanation:

**1. File Path:**

- The `filePath` variable specifies the path to the file `employees.txt`.

**2. File Class:**

- `File file = new File(filePath);` creates a `File` object representing the file.

**3. FileReader and BufferedReader:**

- `FileReader fileReader = new FileReader(file);` creates a `FileReader` object for reading the file.

- `BufferedReader bufferedReader = new BufferedReader(fileReader);` creates a `BufferedReader` object to read text efficiently.

**4. Reading Lines:**

- `bufferedReader.readLine()` reads lines from the file. It returns `null` when the end of the file is reached.

**5. Error Handling:**

- The `try-catch` block catches `IOException` if the file is not found or other I/O errors occur.

- The `finally` block ensures that the `BufferedReader` and `FileReader` are closed to release resources, even if an exception occurs.

**6. Resource Management:**

- The `finally` block includes additional error handling to address potential issues when closing the readers.