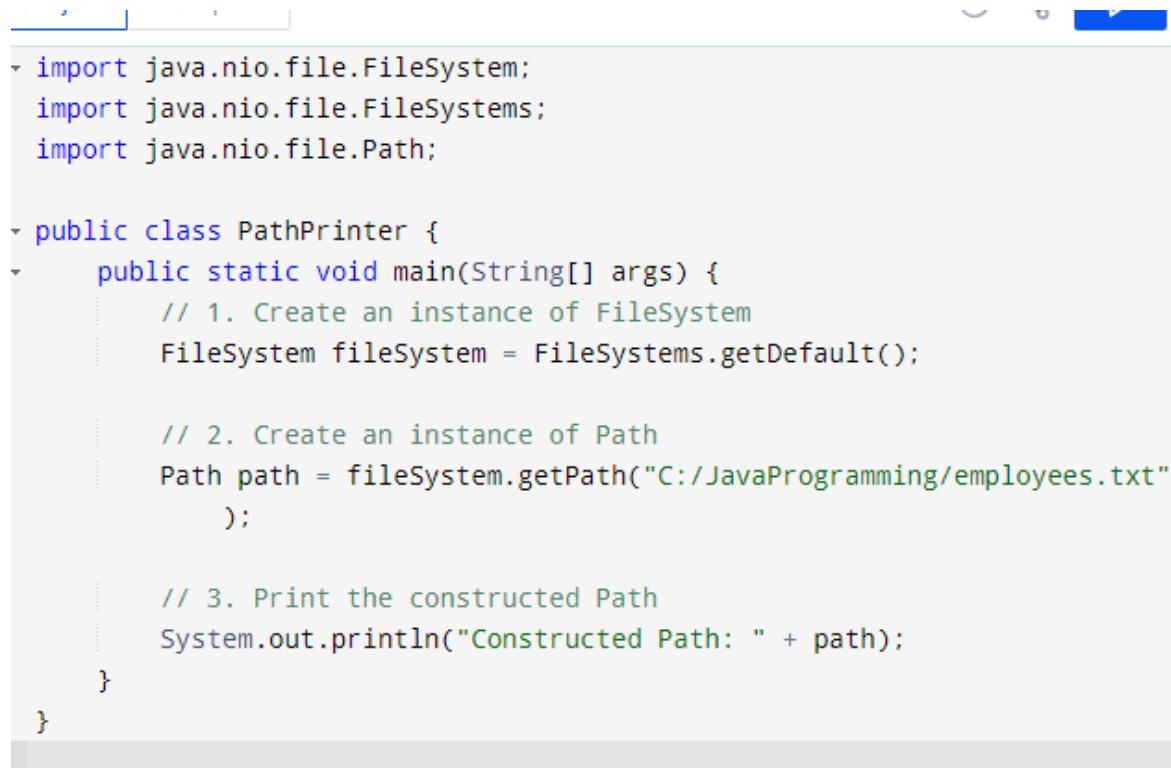


**1. Create a class with a static main that tests the ability to resolve and print a Path:** • Create an instance of a FileSystem class.

- Create an instance of the following Path interface.
- Print the constructed Path with System.out.println() method

A.



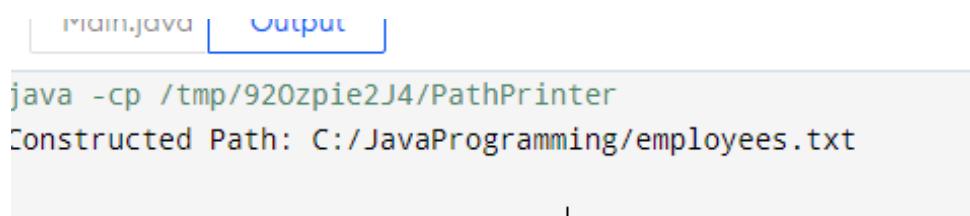
```
import java.nio.file.FileSystem;
import java.nio.file.FileSystems;
import java.nio.file.Path;

public class PathPrinter {
    public static void main(String[] args) {
        // 1. Create an instance of FileSystem
        FileSystem fileSystem = FileSystems.getDefault();

        // 2. Create an instance of Path
        Path path = fileSystem.getPath("C:/JavaProgramming/employees.txt");

        // 3. Print the constructed Path
        System.out.println("Constructed Path: " + path);
    }
}
```

**OUTPUT:**



```
main.java Output
java -cp /tmp/920zpie2J4/PathPrinter
Constructed Path: C:/JavaProgramming/employees.txt
```

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

#### A. Limitations of java.io Package:

##### Blocking I/O Operations:

**Issue:** Operations block the executing thread until completion.

**Impact:** Can lead to performance bottlenecks, especially in high-concurrency environments.

##### Limited Scalability:

**Issue:** Not optimized for handling multiple simultaneous connections or large data streams.

**Impact:** Less suitable for applications requiring high scalability, such as web servers.

#### **Lack of Non-Blocking I/O Support:**

**Issue:** Does not natively support asynchronous or non-blocking I/O operations.

**Impact:** Limits flexibility in designing responsive and efficient applications.

#### **Verbose Exception Handling:**

**Issue:** Heavy reliance on checked exceptions leads to verbose and cluttered code.

**Impact:** Makes the code harder to read and maintain.

#### **Limited File Attribute Access:**

**Issue:** Provides minimal access to file metadata and attributes.

**Impact:** Restricts the ability to perform advanced file operations, such as manipulating file permissions or retrieving detailed file information.

#### **Inflexible Buffering Mechanisms:**

**Issue:** Limited control over buffering strategies.

**Impact:** Can lead to inefficient data handling and increased memory usage.

#### **Synchronous I/O Operations:**

**Issue:** All I/O operations are synchronous.

**Impact:** Cannot initiate multiple I/O operations concurrently without managing threads explicitly.

#### **Limited Support for Character Encodings:**

**Issue:** Basic support for character encoding transformations.

**Impact:** Difficulties in handling diverse character encodings efficiently.

#### **Error-Prone Resource Management:**

**Issue:** Requires explicit closing of streams, often leading to resource leaks if not handled correctly.

**3. Create a class that does the following:** • Using a pre-Java 7 solution, create a class that tests streams in the static main. • The class should instantiate a new File class, a new FileReader class, and new BufferedReader class. • Read lines by using the readLine() method call. • The file path used should be: C:/JavaProgramming/employees.txt • The file should handle errors when the file is not found as well as reading the contents of the file when it is found.

A.

ain.java    Output

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
public class StreamTester {
    public static void main(String[] args) {
        String filePath = "C:/JavaProgramming/employees.txt";
        File file = new File(filePath);
        FileReader fileReader = null;
        BufferedReader bufferedReader = null;
        try {
            fileReader = new FileReader(file);
            bufferedReader = new BufferedReader(fileReader);
            String line;
            while ((line = bufferedReader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.err.println("An error occurred while processing the
                file: " + e.getMessage());
            e.printStackTrace();
        } finally {
            if (bufferedReader != null) {
                try {
                    bufferedReader.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Run

**OUTPUT:**

```
An error occurred while processing the file: C:/JavaProgramming/employees.txt
(No such file or directory)
java.io.FileNotFoundException: C:/JavaProgramming/employees.txt (No such file or
directory)
at java.base/java.io.FileInputStream.open0(Native Method)
at java.base/java.io.FileInputStream.open(FileInputStream.java:219)
at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)
at java.base/java.io.FileReader.<init>(FileReader.java:75)
at StreamTester.main(StreamTester.java:15)

==== Code Execution Successful ===
```