Java Programming 5-1: Basics of Input and Output Practice Solution

Vocabulary Definitions

1. **File Name**: The physical name of a file, or a symbolic link name.
2. **Leaf Node**: A type of node at the bottom of a top-down hierarchical (or inverted tree) that has no node below it.
3. **Symbolic Link**: A file name that maps to another file.
4. **Path**: A top-down single node hierarchy.
5. **Root Node (Linux)**: The top-most node of a file system hierarchy, also known as a volume name, and used on the Linux operating system.
6. **Root Node (Windows)**: The top-most node of a file system hierarchy, also known as a volume name, and used on the Windows operating system.
7. **Hierarchical File System**: A hierarchy of elements, starting from a top-most (or root node) and moving down to nodes without any subordinate nodes.
8. **Absolute Path**: This type of path starts with a logical mount, like `C:\` or `D:\` in Windows, or a `/` (forward slash) or combination of a forward slash and one or more node names, as long as it's qualified as a mount point.
9. **Relative Path**: A path that starts somewhere other than the root node and ends in a file name.
10. **Root**: The top-most node of an absolute or relative path.
11. **Symbolic Link**: A specialized file that points to another absolute or relative file name.

Try It/Solve It

1. Resolve and Print a Path

Here's a Java class that demonstrates resolving and printing a `Path` using the `java.nio.file.Path` interface:

```
import java.nio.file.Path;
import java.nio.file.Paths;

public class PathExample {
    public static void main(String[] args) {
        // Create an instance of a Path interface
        Path path = Paths.get("C:/JavaProgramming/employees.txt");

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

2. Limitations of the `java.io` Package

The main limitations of the `java.io` package include:

1. **Lack of Scalability**: It can be less efficient with large files or data streams, as it doesn't provide built-in support for buffering and non-blocking I/O.
2. **Limited Functionality for Modern I/O Needs**: It lacks some modern I/O capabilities such as non-blocking I/O and asynchronous file operations.
3. **Error Handling**: Error handling can be cumbersome and less flexible compared to newer I/O libraries.
4. **Complexity with Paths**: Working with file paths and directories can be more complex and less intuitive compared to the `java.nio.file` package.
5. **No Direct Support for File System Operations**: It doesn't offer methods for operations like file attributes and file system operations directly.

3. File Handling with `java.io`

Here’s a Java class that uses the `java.io` package to read from a file, handle errors, and print file contents:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class FileReadingExample {
    public static void main(String[] args) {
        // Define the file path
        String filePath = "C:/JavaProgramming/employees.txt";

        // Create File, FileReader, and BufferedReader objects
        File file = new File(filePath);
        FileReader fileReader = null;
        BufferedReader bufferedReader = null;

        try {
            // Initialize FileReader and BufferedReader
            fileReader = new FileReader(file);
            bufferedReader = new BufferedReader(fileReader);

            // Read lines from the file
            String line;
            while ((line = bufferedReader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.err.println("An error occurred while reading the file: " + e.getMessage());
        } finally {
            // Close resources
            try {
                if (bufferedReader != null) {
                    bufferedReader.close();
                }
                if (fileReader != null) {
                    fileReader.close();
                }
            } catch (IOException e) {
                System.err.println("An error occurred while closing the file: " + e.getMessage());
            }
        }
    }
}
```