EXPERIMENT – 02

Objective:

To create a well-structured Java program that processes and displays command line arguments.

Theory:Processing Command-Line Arguments in Java

In Java, **command-line arguments** provide a mechanism to pass information into a program at runtime. These are values entered in the terminal or command prompt when launching the program. Instead of hardcoding inputs within the program, command-line arguments allow the user to provide dynamic data every time the program is run.

The Java program receives these values through the main method using the String[] args parameter. The args array holds each individual argument passed during execution. Here's an example of how you might run a program with command-line arguments:

```
java MyProgram Hello World 2025
```

In this case, the program receives three command-line arguments: Hello, World, and 2025.

Understanding Command-Line Arguments:

Command-line arguments are essentially a set of strings that the program receives during execution. The program can then process these arguments based on its logic. Each argument passed can be accessed individually using the array index, like args[0], args[1], and so on.

The example below demonstrates how to use the args array to display the arguments received:

```
public class CommandLineExample {
    public static void main(String[] args) {
        if (args.length > 0) {
            for (int i = 0; i < args.length; i++) {
                System.out.println("Argument " + (i + 1) + ": " + args[i]);
            }
        } else {
            System.out.println("No arguments provided.");
        }
    }
}
```

Key Features:

- **Dynamic Input:** Allows users to input values when the program is launched.

- **Array of Strings:** The command-line arguments are stored in a string array (String[] args), and the length of this array tells you how many arguments were passed.
- **Flexible and User-Friendly:** Ideal for passing settings, paths, or data during execution, improving program flexibility.

Command-Line Argument Structure

Index	Argument	Description
0	Hello	First argument passed by user
1	World	Second argument passed by user
2	2025	Third argument passed by user

- **args[0]** will contain "Hello"
- **args[1]** will contain "World"
- **args[2]** will contain "2025"

This table shows the order and significance of each argument passed when running a Java program. It allows us to access arguments in the order they were entered.

Common Methods for Command-Line Argument Processing

Method	Purpose
args.length	Returns the number of arguments passed
Integer.parseInt(args[i])	Converts the string argument to an integer
Double.parseDouble(args[i])	Converts the string argument to a double
args[i]	Accesses the i-th argument in the array

Real-World Usage of Command-Line Arguments:

Command-line arguments are particularly useful in programs where users need to provide input without the need for interactive prompts during runtime. Common use cases include:

- **Configuration Settings:** Pass configuration files, user preferences, or mode settings.

- **File Handling:** Specify input/output file paths or filenames.
- **Automation:** Automate tasks such as batch processing or system scripts where input parameters are predefined at runtime.

Command-line arguments in Java provide a powerful way to interact with programs during execution. By capturing arguments from the terminal, Java applications become more dynamic, flexible, and user-friendly. With simple techniques such as checking args.length and parsing input data, developers can create programs that easily handle a variety of user inputs and configurations.

Enhanced Source Code:

File: CommandLineProcessor.java

```
public class CommandLineProcessor {
    public static void main(String[] arguments) {
        System.out.println("----- Java Command Line Argument Processor -----\n");

        if (arguments.length > 0) {
            System.out.println("Total Arguments Received: " + arguments.length);
            System.out.println("Listing Arguments:");

            for (int index = 0; index < arguments.length; index++) {
                System.out.println("Argument " + (index + 1) + ": " + arguments[index]);
            }
        } else {
            System.out.println("No arguments were provided. Please run the program with 
arguments.");
        }
        System.out.println("\nAnalyzing arguments for numeric values:");
        for (String arg : arguments) {
            try {
                Integer.parseInt(arg);
                System.out.println(arg + " is a valid integer.");
            } catch (NumberFormatException e) {
                System.out.println(arg + " is not an integer.");
            }
        }
    }
}
```

VS Code Execution Steps

1. **Open VS Code** and **install Java Extension Pack**.

2. **Create a folder/project**, then **create a .java file** (e.g., CommandLineProcessor.java).
3. **Paste your Java code** into the file and **save** it.
4. **Open Terminal** (Ctrl + ~), navigate to the file's directory.
5. **Compile the code**:

❖ javac CommandLineProcessor.java

6. **Run the program with arguments**:

❖ java CommandLineProcessor arg1 arg2 arg3

Sample Inputs and Outputs:

Input Arguments	Output Summary
5 10 Hello	Displays all arguments and checks for integers.
123	Displays one argument, detects it as integer.
abc def	Displays two arguments, none are integers.
	No arguments provided message shown.

```
PS C:\Users\WIN 10\Desktop\Java> java CommandLineProcessor 5 10 Hello

----- Java Command Line Argument Processor -----

Total Arguments Received: 3

Listing Arguments:

Argument 1: 5

Argument 2: 10

Argument 3: Hello

Analyzing arguments for numeric values:

5 is a valid integer.

10 is a valid integer.

Hello is not an integer.
```

```
PS C:\Users\WIN 10\Desktop\Java> java CommandLineProcessor 123

----- Java Command Line Argument Processor -----

Total Arguments Received: 1
```

```
Listing Arguments:

Argument 1: 123

Analyzing arguments for numeric values:

123 is a valid integer.


PS C:\Users\WIN 10\Desktop\Bootstrap> java CommandLineProcessor abc def

----- Java Command Line Argument Processor -----

Total Arguments Received: 2

Listing Arguments:

Argument 1: abc

Argument 2: def

Analyzing arguments for numeric values:

abc is not an integer.

def is not an integer.


PS C:\Users\WIN 10\Desktop\Java> java CommandLineProcessor

----- Java Command Line Argument Processor -----

No arguments were provided. Please run the program with arguments.


Analyzing arguments for numeric values:
```

Conclusion:

This experiment provided hands-on experience with writing Java programs and demonstrated the usage of command line arguments. It also enhanced our understanding of argument parsing, string arrays, loops, and exception handling in Java.