Experiment No.2
Accepting Input Through Keyboard
Date of Performance:
Date of Submission:

Aim: To apply basic programing for accepting input through keyboard.

Objective: To use the facility of java to read data from the keyboard for any program

Theory:

Java brings various Streams with its I/O package that helps the user perform all the Java input-output operations. These streams support all types of objects, data types, characters, files, etc. to fully execute the I/O operations. Input in Java can be with certain methods mentioned below in the article.

Methods to Take Input in Java

There are two ways by which we can take Java input from the user or from a file

1. BufferedReader Class
2. Scanner Class

Using BufferedReader Class for String Input In Java

It is a simple class that is used to read a sequence of characters. It has a simple function that reads a character another read which reads, an array of characters, and a readLine() function which reads a line.

InputStreamReader() is a function that converts the input stream of bytes into a stream of characters so that it can be read as BufferedReader expects a stream of characters. BufferedReader can throw checked Exceptions.

Using Scanner Class for Taking Input in Java

It is an advanced version of BufferedReader which was added in later versions of Java. The scanner can read formatted input. It has different functions for different types of data types.

The scanner is much easier to read as we don't have to write throws as there is no exception thrown by it.

It was added in later versions of Java

It contains predefined functions to read an Integer, Character, and other data types as well.

Syntax of Scanner class

Scanner scn = new Scanner(System.in);

Code:

1} Scanner class

```
import java.util.Scanner;


public class UserProgram {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Name, Age, and Salary:");

        String name = scanner.nextLine();
        int age = scanner.nextInt();
        double salary = scanner.nextDouble;

        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Salary: " + salary);

        scanner.close();
    }
}
```

2} Buffer reader class

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

class FileReaderExample {
   public static void main(String[] args) {
      char[] array = new char[100];
      try {
         FileReader file = new FileReader("input.txt");
         BufferedReader input = new BufferedReader(file);
         input.read(array);
         System.out.println("Data in the File:");
         System.out.print(array);
         input.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

Conclusion:

1) Comment on how you have used BufferedReader and Scanner Class for accepting user input

The `BufferedReader` class is designed for efficiently reading text from character input streams, such as files. It provides a buffering mechanism, enabling more efficient handling of large text data by reading characters in chunks. In your code, you create a `BufferedReader` by wrapping a `FileReader`, and use the `read(char[] cbuf)` method to efficiently read a chunk of characters from a file named "input.txt." This class is ideal for reading text data and is often preferred when dealing with text files or streams.

On the other hand, the `Scanner` class, although not used in your provided code, is typically employed for parsing user input from the console. You create a `Scanner` instance associated with `System.in`, the standard input stream, to read and extract specific values from user input. The `Scanner` class offers various methods like `nextInt()`, `nextDouble()`, and `nextLine()` for reading different data types and tokens from user input, making it particularly useful for interactive console applications where you need to interpret input as various data types. In summary, `BufferedReader` is efficient for reading data from files or streams, while the `Scanner` class is often chosen for parsing user input interactively, particularly when dealing with diverse data types and tokenization of input. Both classes are valuable for handling input in Java applications, each serving distinct purposes.