Experiment No.2
Accepting Input Through Keyboard
Date of Performance: 01/08/2024
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 Using Scanner Class:

```
import java.io.*;
import java.util.*;
public class UserInput
{
   public static void main(String[] args)
   {
      Scanner S = new Scanner(System.in);
      System.out.println("Enter Your Name:");
      String name = S.nextLine();
      System.out.println("Enter Your Age:");
      int age = S.nextInt();
      System.out.println("Your Name:"+ name);
      System.out.println("Your Age:"+ age);
   }
}
```

OutPut:

```
Enter Your Name:
JAVA
Enter Your Age:
4 50
Your Name:JAVA
Your Age:50
```

ScreenShot:


Enter Your Name:
JAVA
Enter Your Age:
50
Your Name:JAVA
Your Age:50
...Program finished with exit code 0
Press ENTER to exit console.

Code Using BufferClass:

```
import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;


public class BufferedReader

{

   public static void main(String[] args) throws IOException

   {

      BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

      System.out.println("Enter your name:");


         String name = reader.readLine();

         System.out.println("Hello, " + name);

   }

}
```

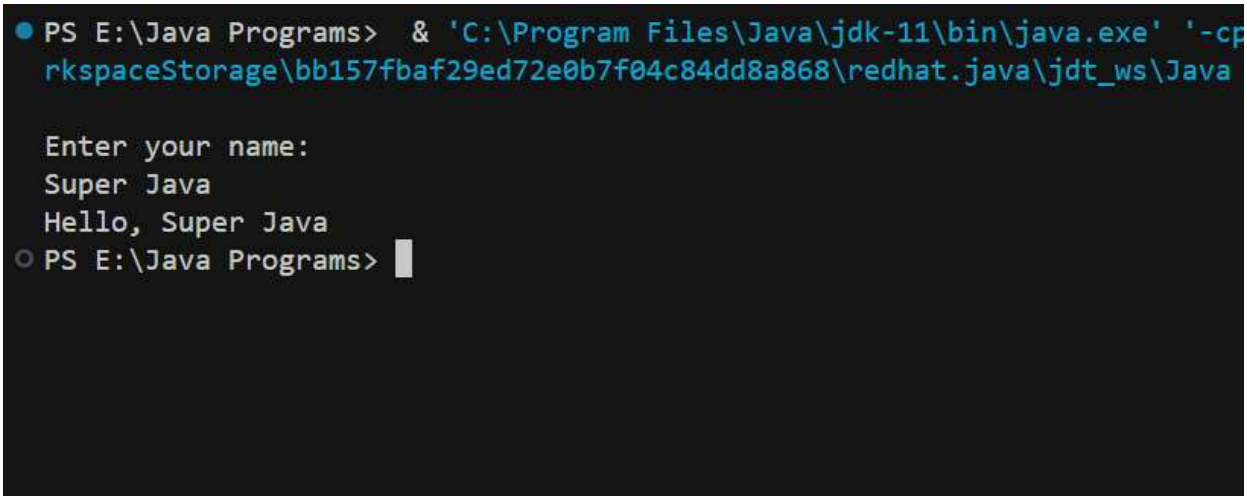
Code:

```
Enter your name:

Super Java

Hello, Super Java
```

ScreenShot:



Conclusion:

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

Ans:

In Java, both the BufferedReader and Scanner classes are commonly used for accepting user input, but they serve slightly different purposes and have their own advantages and disadvantages.

1. Using BufferedReader:

The BufferedReader class is used to read text from an input stream efficiently. It buffers the characters for efficient reading, which makes it faster for large inputs.

Advantages:

- Efficient for reading large amounts of data.
- Synchronous, making it suitable for multi-threaded environments.

Disadvantages:

- Requires more code to handle input parsing.
- Not as user-friendly for simple input tasks.

2. Using Scanner

The Scanner class is more user-friendly and provides methods to parse primitive types and strings using regular expressions.

Advantages:

- Easier to use for simple input tasks.
- Provides convenient methods for parsing different types of input.

Disadvantages:

- Less efficient for large inputs due to smaller buffer size.
- Not synchronized, making it less suitable for multi-threaded environments.