Getting Input from the Keyboard

CC1 – Computing Fundamentals

Table of Contents

Scanner

Scanner

is a class in the **java.util** package that allows for easy input parsing and retrieval of different data types from various input sources, such as the standard input, files, or strings.

Inputting Other Data Types

`nextLine()` Reads a single line value from the input.

Method	*Description*
`nextBoolean()`	Reads a boolean value from the user
`nextByte()`	Reads a byte value from the user
`nextDouble()`	Reads a double value from the user
`nextFloat()`	Reads a float value from the user
`nextInt()`	Reads a int value from the user
`next()`	Reads a String value from the user
`nextLong()`	Reads a long value from the user
`nextShort()`	Reads a short value from the user

Steps in Using Scanner

1. We import the necessary class: Scanner from the java.util package.
2. Inside the main method, we create a Scanner object named scanner that reads from the standard input (System.in).
3. We prompt the user to enter their name by using System.out.print() to display the message.
4. We read the name input from the user using the nextLine() method of Scanner and store it in a String variable named name.

Steps in Using Scanner

4. We prompt the user to enter their age.
5. We read the age input from the user using the nextInt() method of Scanner and store it in an int variable named age.
6. We display a message that includes the entered name and age.
7. We close the Scanner using the close() method to release system resources.

```
import java.util.Scanner;

public class ScannerExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter your name: ");
        String name = scanner.nextLine();

        System.out.print("Enter your age: ");
        int age = scanner.nextInt();

        System.out.println("Hello, " + name + "! You are " + age + " years old.");

        scanner.close();
    }
}
```

Sample Program Using Scanner

```
import java.util.*; //Import the Scanner class

public class ScannerInput {
  public static void main(String[] args) {

        //Create Scanner object
      Scanner s = new Scanner(System.in);
      System.out.println("Enter username: ");

      //Read user input
      String userName = s.nextLine();
      System.out.println("Username is: " + userName);
  }
}
```

BufferedReader

BufferedReader

- a class provided in the java.io package that is used for efficient reading of characters from an input stream, such as a file, network socket, or other Reader implementations.

IOException

- is a checked exception, which means it must be declared in the method signature or caught within a try-catch block.

Features of BufferedReader

- **Efficient character reading:** BufferedReader is designed to efficiently read characters from an input stream by minimizing the number of system calls.
- **Line-by-line reading:** BufferedReader provides a convenient method called readLine() that reads an entire line of text from the input stream.
- **Read operations:** In addition to readLine(), BufferedReader offers other methods for reading characters or chunks of characters from the input stream, such as read(), read(char[] cbuf, int off, int len), and more.

Features of BufferedReader

- **Integration with other input sources:** BufferedReader can be used with various input sources. It accepts any Reader implementation, including FileReader, InputStreamReader, or any other class that extends Reader.

- **Error handling:** BufferedReader throws IOException for input/output-related errors.

Methods of BufferedReader

- **read():** This method reads a single character from the input stream and returns its integer representation.

```
int charValue = reader.read();
```

- **read(char[] cbuf):** This method reads characters into an array cbuf from the input stream and returns the number of characters read.

```
char[] buffer = new char[1024];
int numCharsRead = reader.read(buffer);
```

Methods of BufferedReader

- **read(char[] cbuf, int off, int len):** This method reads characters into an array cbuf starting at the given offset off, and reads at most len characters.

```
char[] buffer = new char[1024];
int numCharsRead = reader.read(buffer, 0, 100);
```

- **readLine():** This method reads a line of text from the input stream and returns it as a string.

```
String line = reader.readLine();
```

Note:

- **FileInputStream** and **InputStreamReader** are both classes in Java that are used for reading data from input sources, but they serve different purposes.
- If you are working with binary data or need to handle low-level byte operations, **FileInputStream** is appropriate.
- If you are working with text-based data and want to deal with characters and character encodings, **InputStreamReader** is more suitable.

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

public class BufferedReaderExample {
    public static void main(String[] args) {
        BufferedReader reader = null;

        try {
            // Create a BufferedReader object to read from standard input (keyboard)
            reader = new BufferedReader(new InputStreamReader(System.in));

            // Read a line from the standard input
            String line = reader.readLine();

            // Process the line (or perform any desired operations)
            System.out.println("You entered: " + line);

        } catch (IOException e) {
            e.printStackTrace();
// Print the exception trace for debugging
        } finally {
            // Close the BufferedReader in the finally block to ensure it's always closed
            try {
                if (reader != null) {
                    reader.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
```

Steps in Using BufferedReader

1. We import the necessary classes: BufferedReader, IOException, and InputStreamReader.
2. Inside the main method, we declare a BufferedReader variable called reader and initialize it as null.
3. We use a try-catch-finally block to handle any potential exceptions.
4. Within the try block, we create a BufferedReader object, reader, by wrapping an InputStreamReader object that reads from the standard input (System.in).
5. We read a line from the standard input using the readLine() method of BufferedReader and store it in a String variable called line

Steps in Using BufferedReader

6. We can process the line variable or perform any desired operations with it. In this example, we simply print the line with a message.

7. In the catch block, any IOException that occurs during the input reading process is caught and the exception trace is printed for debugging purposes.

8. In the finally block, we ensure that the BufferedReader is closed by calling its close() method. The close() method can potentially throw an IOException, so we handle it by printing the exception trace.

Scanner vs. BufferedReader

Scanner

- Scanner is primarily used for parsing and retrieving different types of input tokens.

BufferedReader

- BufferedReader is mainly used for reading lines of text efficiently.

Scanner vs. BufferedReader

Scanner

- Scanner provides built-in methods for parsing input tokens, such as nextInt(), nextDouble(), and nextLine(), which automatically handle different data types.

BufferedReader

- BufferedReader does not have built-in tokenization capabilities and mainly focuses on reading lines as strings.

Scanner vs. BufferedReader

Scanner

- Scanner does not have built-in buffering, so it may not be as efficient when reading large amounts of data.

BufferedReader

- BufferedReader provides buffering capabilities, which means it reads data from the underlying source in larger chunks, reducing the number of I/O operations and improving performance.

Scanner vs. BufferedReader

Scanner

- Scanner can read input from various sources, including the standard input (System.in), files (File or InputStream), or strings (String)

BufferedReader

- BufferedReader is typically used for reading from files or other character-based input streams.

Scanner vs. BufferedReader

Scanner

- Scanner is suitable for token-based parsing and retrieving different data types

BufferedReader

- BufferedReader is more focused on efficient line-by-line reading of text.