Entrada e saída de dados em Java

Quando se trata de ler dados em Java, três classes vêm à mente: `InputStream`, `BufferedReader`, e `Scanner`. Embora compartilhem algumas similaridades, cada uma tem seu propósito único e características.

`InputStream`: Um `InputStream` é uma classe fundamental que fornece uma forma de ler bytes a partir de qualquer fonte (por exemplo, arquivo, rede). É um fluxo de dados bruto que requer processamento adicional para converter os bytes em dados significativos. Pense nela como um "fluxo de entrada de dados bruto, sem processamento inicial".

```
// Java program to demonstrate BufferedReader
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Test {
   public static void main(String[] args)
      throws IOException
   {
      // Enter data using BufferReader
      BufferedReader reader = new BufferedReader(
         new InputStreamReader(System.in));

      // Reading data using readLine
      String name = reader.readLine();

      // Printing the read line
      System.out.println(name);
   }
}
```

`BufferedReader`: Um `BufferedReader` é uma classe mais alta que fornece uma forma de ler dados textuais (por exemplo, strings) a partir de um fluxo de saída subordinado (por exemplo, arquivo, console). É projetado para ler dados de carácter e proporciona suporte incorporado para transformação de formatos comuns. Pense nela como um "fluxo de entrada de texto".

```
// Java Program to Illustrate BufferedReader Class
// Via Its Methods

// Importing required classes
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

// Class
class GFG {

    // Main driver method
    public static void main(String[] args)
        throws IOException
    {

        // Creating object of FileReader and BufferedReader
        // class
        FileReader fr = new FileReader("file.txt");
        BufferedReader br = new BufferedReader(fr);

        char c[] = new char[20];

        // Illustrating markSupported() method
        if (br.markSupported()) {

            // Print statement
            System.out.println(
                "mark() method is supported");

            // Illustrating mark method
```

```
        br.mark(100);
}

// File Contents is as follows:
// This is first line
// this is second line

// Skipping 8 characters
br.skip(8);

// Illustrating ready() method
if (br.ready()) {

        // Illustrating readLine() method
        System.out.println(br.readLine());

        // Illustrating read(char c[],int off,int len)
        br.read(c);

        for (int i = 0; i < 20; i++) {
                System.out.print(c[i]);
        }

        System.out.println();

        // Illustrating reset() method
        br.reset();
        for (int i = 0; i < 8; i++) {

                // Illustrating read() method
                System.out.print((char)br.read());
        }
```

```
        }
    }
}
```

`Scanner`: Um `Scanner` é uma classe utilitária que simplifica o processo de leitura de dados de entrada a partir de várias fontes (por exemplo, terminal, arquivo, rede). É projetado para fornecer fácil acesso a tipos de entrada comumente usados, como inteiros, números floats, strings e datas. Pense nela como um "fluxo de entrada fácil".

Exemplo:

```
// Java program to read data of various types
// using Scanner class.
import java.util.Scanner;

// Driver Class
public class ScannerDemo1 {
    // main function
    public static void main(String[] args)
    {
        // Declare the object and initialize with
        // predefined standard input object
        Scanner sc = new Scanner(System.in);

        // String input
        String name = sc.nextLine();

        // Character input
        char gender = sc.next().charAt(0);

        // Numerical data input
        // byte, short and float can be read
        // using similar-named functions.
        int age = sc.nextInt();
```

```
            long mobileNo = sc.nextLong();
            double cgpa = sc.nextDouble();

            // Print the values to check if the input was
            // correctly obtained.
            System.out.println("Name: " + name);
            System.out.println("Gender: " + gender);
            System.out.println("Age: " + age);
            System.out.println("Mobile Number: " + mobileNo);
            System.out.println("CGPA: " + cgpa);
        }
}
```

Em resumo:

- `InputStream` é um fluxo de bytes simples, bruto
- `BufferedReader` é um fluxo de dados de texto com suporte para formatação incorporada
- `Scanner` é uma classe utilitária para leitura de dados de entrada fácil

Quando escolher entre essas classes, considere o tipo de dados que você está trabalhando e o seu caso específico. Por exemplo:
Use `InputStream` quando precisar de acesso bruto a bytes;
Use `BufferedReader` quando estiver trabalhando com dados textuais que requerem formatação;

Use `Scanner` quando quiser uma forma fácil e fácil de leitura de dados.

Saída de dados

Quando se trata de escrever dados em Java, três classes vêm à mente: `PrintStream`, `BufferedWriter`, e `PrintWriter`. Embora compartilhem algumas similaridades, cada uma tem seu propósito único e características.

`PrintStream`: Um `PrintStream` é uma classe fundamental que fornece uma forma de escrever bytes a partir de qualquer fluxo de saída (por exemplo, arquivo, console). É um fluxo de dados bruto que requer processamento adicional para formatar os dados para consumo humano. Pense nela como um "fluxo de saída bruto".

Import java.io.FileOutputStream;

- import java.io.PrintStream;
- public class PrintStreamTest{

```
public static void main(String args[])throws Exception{
   FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
  PrintStream pout=new PrintStream(fout);
  pout.println(2016);
  pout.println("Hello Java");
  pout.println("Welcome to Java");
  pout.close();
  fout.close();
   System.out.println("Success?");
  }
}
```

`BufferedWriter`: Um `BufferedWriter` é uma classe mais alta que fornece uma forma de escrever dados textuais (por exemplo, strings) a partir de um fluxo de saída subordinado (por exemplo, arquivo, console). É projetado para escrever dados de carácter e proporciona suporte incorporado para formatação comum. Pense nela como um "fluxo de saída de texto".

Import java.io.*;

```
public class BufferedWriterExample {
public static void main(String[] args) throws Exception{
    FileWriter writer = new FileWriter("D:\\testout.txt");
     BufferedWriter buffer = new BufferedWriter(writer);
     buffer.write("Welcome to JavaWorld.");
     buffer.close();
     System.out.println("Success");
     }
}
```

`PrintWriter`: Um `PrintWriter` é uma classe utilitária que simplifica o processo de escrita de dados formatados a partir de várias fontes (por exemplo, console, arquivo). É projetado para fornecer fácil acesso a formatações comumente usadas, como strings e datas. Pense nela como um "fluxo de saída fácil".

Import java.io.File;

```
import java.io.PrintWriter;
public class PrintWriterExample{
public static void main(String[] args) throws Exception {
//Data to write on Console using PrintWriter
    PrintWriter writer = new PrintWriter(System.out);
    writer.write("Welcome to Java World.");
 writer.flush();
```

```
    writer.close();
//Data to write in File using PrintWriter
        PrintWriter writer1 =null;
         writer1 = new PrintWriter(new File("D:\\testout.txt"));
         writer1.write("Like Java, Spring, Hibernate, Android, PHP etc.");
                writer1.flush();
         writer1.close();
    }
}
```

Em resumo:

- `PrintStream` é um fluxo de bytes simples, não processado inicialmente;
- `BufferedWriter` é um fluxo de dados textuais com suporte para formatação incorporada;
- `PrintWriter` é uma classe utilitária para escrita de dados formatados de forma fácil.

Quando escolher entre essas classes, considere o tipo de dados que você está trabalhando e o seu caso específico. Por exemplo:

Use `PrintStream` quando precisar de acesso bruto a bytes;

Use `BufferedWriter` quando estiver trabalhando com dados textuais que requerem formatação;

Use `PrintWriter` quando quiser uma forma fácil e fácil de escrita de dados.