Java Scanner Class

Java **Scanner class** allows the user to take input from the console. It belongs to **java.util** package. It is used to read the input of primitive types like int, double, long, short, float, and byte. It is the easiest way to read input in Java program.

Syntax

```
1. Scanner sc=new Scanner(System.in);
```

The above statement creates a constructor of the Scanner class having **System.in** as an argument. It means it is going to read from the standard input stream of the program. The **java.util** package should be import while using Scanner class.

It also converts the Bytes (from the input stream) into characters using the platform's default charset.

Methods of Java Scanner Class

Java Scanner class provides the following methods to read different primitives types:

Method	Description
int nextInt()	It is used to scan the next token of the input as an integer.
float nextFloat()	It is used to scan the next token of the input as a float.
double nextDouble()	It is used to scan the next token of the input as a double.
byte nextByte()	It is used to scan the next token of the input as a byte.
String nextLine()	Advances this scanner past the current line.
boolean nextBoolean()	It is used to scan the next token of the input into a boolean value.
long nextLong()	It is used to scan the next token of the input as a long.
short nextShort()	It is used to scan the next token of the input as a Short.
BigInteger nextBigInteger()	It is used to scan the next token of the input as a BigInteger.
BigDecimal nextBigDecimal()	It is used to scan the next token of the input as a BigDecimal.

Example 1:(taking srting input from user using scanner class)

```
import java.util.*;

class InputScanner1
{
public static void main(String[] args)
{
Scanner sc= new Scanner(System.in); //System.in is a standard input stream

System.out.println("Enter a string: ");

String str= sc.nextLine();              //reads string

System.out.println("You have entered: "+str);

}
}
```

```
Enter a string:
hello khyati
You have entered: hello khyati
```

Example 2:(taking integer input from user using scanner class)

```
import java.util.*;

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

Scanner sc= new Scanner(System.in);    //System.in is a standard input stream

System.out.print("Enter first number- ");
int a= sc.nextInt();

System.out.print("Enter second number- ");
int b= sc.nextInt();

System.out.print("Enter third number- ");
int c= sc.nextInt();
```

```
int d=a+b+c;

System.out.println("Total= " +d);
}
}
```

```
Enter first number- 3
Enter second number- 4
Enter third number- 5
Total= 12
```

Why is Scanner skipping nextLine() after use of other next functions?

The nextLine() method of java.util.Scanner class advances this scanner past the current line and returns the input that was skipped. This function prints the rest of the current line, leaving out the line separator at the end. The next is set to after the line separator. Since this method continues to search through the input looking for a line separator, it may search all of the input searching for the line to skip if no line separators are present.

Example:

// Java program to show the issue with nextLine() method of Scanner Class

```
import java.util.Scanner;

public class ScannerProb
{
   public static void main(String[] args)
   {
      // Declare the object and initialize with predefined standard input object

     Scanner sc = new Scanner(System.in);

      // Taking input

      String name = sc.nextLine();
      char gender = sc.next().charAt(0);
      int age = sc.nextInt();
      String fatherName = sc.nextLine();
      String motherName = sc.nextLine();
```

```
        // 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("Father's Name: " + fatherName);
        System.out.println("Mother's Name: " + motherName);
    }
}
```

Expected Output:

Name: abc

Gender: m

Age: 1

Father's Name: xyz

Mother's Name: pqr

Actual Output:

Name: abc

Gender: m

Age: 1

Father's Name:

Mother's Name: xyz

As you can see, the nextLine() method skips the input to be read and takes the name of Mother in the place of Father. Hence the expected output does not match with the actual output. This error is very common and causes a lot of problems.

Why does this issue occur?

This issue occurs because, when nextInt() method of Scanner class is used to read the age of the person, it returns the value 1 to the variable age, as expected. But the cursor, after reading 1, remains just after it.

abc

m

1_ // Cursor is here
xyz
pqr

So when the Father's name is read using nextLine() method of Scanner class, this method starts reading from the cursor's current position. In this case, it will start reading just after 1. So the next line after 1 is just a new line, which is represented by '\n' character. Hence the Father's name is just '\n'.

How to solve this issue?

This issue can be solved by either of the following two ways:

1. reading the complete line for the integer and converting it to an integer, or

Syntax:

// Read the complete line as String

// and convert it to integer

int var = Integer.parseInt(sc.nextLine());

Although this method is not applicable for input String after Byte character(Byte.parseByte(sc.nextLine()). Second method is applicable in that case.

2. by consuming the leftover new line using the nextLine() method.

Syntax:

```
// Read the integer
int var = sc.nextInt();


// Read the leftover new line
sc.nextLine();
```

Problem Solution

```
// Taking input

    String name = sc.nextLine();
    char gender = sc.next().charAt(0);

// Consuming the leftover new line using the nextLine() method

    sc.nextLine();

// reading the complete line for the integer and converting it to an integer

   int age = Integer.parseInt(sc.nextLine());

    String fatherName = sc.nextLine();
    String motherName = sc.nextLine();
```

Scanner Vs. Bufferreader

1. Scanner is a much more powerful utility than BufferedReader. It can parse the user input and read int, short, byte, float, long and double apart from String. On the other hand BufferedReader can only read string in java.

   2. BuffredReader has significantly large buffer (8KB) than Scanner (1KB), which means if you are reading long String from file, you should use BufferedReader but for short input and input other than String, you can use Scanner class.

   3. BufferedReader is older than Scanner. It's present in Java from JDK 1.1

onward but Scanner is only introduced in JDK 1.5 release.

4. Scanner uses regular expression to read and parse text input. It can accept custom delimiter and parse text into primitive data type e.g. int, long, short, float or double using nextInt(), nextLong(), nextShort(), nextFloat(), and nextDouble() methods, while BufferedReader can only read and store String using readLine() method.

5. Another major difference between BufferedReader and Scanner class is that BufferedReader is **synchronized** while Scanner is not. This means, you cannot share Scanner between multiple threads but you can share the BufferedReader object.