Reading User Input Using the Scanner

by Sophia

WHAT'S COVERED

In this lesson, you will learn about how to use a Java `Scanner` to read different types of input in Java. Specifically, this lesson covers:

1. Reading String Data
2. Reading Numeric Input

1. Reading String Data

In this tutorial, you will learn simple ways that Java can handle keyboard input. In order for a useful interactive program to work, the program must receive information from a user. Additionally, the program must be able to send information to the user. As discussed previously, the output from a program can be displayed by using the `System.out.print()` and `System.out.println()` statements.

The actions of receiving input from the keyboard and sending output to the display window creates standard user interfaces for programs. In Java, any source or destination for input or output (I/O) is considered a stream of bytes or characters. To perform keyboard input, users extract characters from System.in, the input stream that is connected to the keyboard.

Getting keyboard input from System.in involves reading the input and then interpreting it as the correct data type.

THINK ABOUT IT

Think about an email, a text, or document for which you used a keyboard to provide user input today. User input from the keyboard consists of a sequence of characters or digits which represent a word, phrase, integer, or real number.

An entire sequence of characters is normally typed by the user. This input represents data to be stored in a single variable until the user hits the return or enter key. This action signals the end of a piece of requested data. Java has a special mechanism that uses an input stream. This input stream has a method that collects characters until it reads the character or characters that correspond to hitting the return or enter key.

After reading the input sequence, Java converts the input to the appropriate data type. The sequence 12345, as a `String`, "12345," could be a ZIP code, a set of digits that is not really a number, or the integer 12,345. Java has a `Scanner` class which reads the stream of keyboard input and has built-in methods that allow it to interpret the input as the correct data type.

The `Scanner` class is in a section of the Java library found in the `java.util` package. This allows programs that use a `Scanner` to import the corresponding section of the library using an import statement:

EXAMPLE

```
import java.util.Scanner;
```

The Scanner class provides the following methods for reading common types of data:

Method	Description
next()	Reads data until a space or new line as a String
nextLine()	Reads data until the end of the line as a String
nextInt()	Reads the input as an integer
nextLong()	Reads the input as a long integer
nextFloat()	Reads the input as a float
nextDouble()	Read the input as a double

The first two methods, `next()` and `nextLine()`, require a more detailed discussion. As the chart indicates, they both read the input as a string. The difference between the two is described by how much of the input they read and interpret as a String. The `next()` method processes the input until whitespace is encountered. Whitespace includes a plain space, a tab character (`\t`), or a new line character (`\n`). The `nextLine()` method reads and processes all input (including spaces and tabs) until a new line is found. This means that if the input includes multiple words, use `nextLine()`, since `next()` will stop reading when it finds a space.

Here are two sample programs that show the difference in functionality between `next()` and `nextLine()`:

```
import java.util.Scanner;

public class ScannerExample {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter 2 or more words separated by spaces: ");
    String inputRead = input.next();
    System.out.println("Input read by next(): " + inputRead);
  }
}
```

The result should look like this:

```
Enter 2 or more words separated by spaces: hello world
Input read by next(): hello
```

The following is an example of how to use `nextLine()` in a program:

```
import java.util.Scanner;

public class ScannerExample {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter 2 or more words separated by spaces: ");
    String inputRead = input.nextLine();
    System.out.println("Input read by nextLine(): " + inputRead);
  }
}
```

The result should look like this:

```
Enter 2 or more words separated by spaces: hello world
Input read by nextLine(): hello world
```

Looking at the list of `Scanner` methods, it is important to note that there is no `nextChar()` method. If a program needs to read a single character, the input needs to be read as a String (using `next()` or `nextLine()`).

Then, the first (and presumably only) character needs to be extracted using the `charAt()` method as demonstrated below:

```
import java.util.Scanner;

class CharReader {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter a character: ");
    // There shouldn't be a space, so use next()
    String stringInput = input.next();
    // 1st & only char is at index/position 0
    char singleChar = stringInput.charAt(0);
    System.out.println("You entered " + singleChar + ".");
  }
}
```

The output screen of CharReader.java looks like this:

```
Enter a character: A
You entered A.
```

2. Reading Numeric Input

Scanner methods for reading numeric types read the input as a String, and then convert it to the appropriate numeric data type.

Here is an example that uses nextInt() to read and parse the input as an int:

```
import java.util.Scanner;

class NumericScanner {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter a whole number: ");
    int wholeNumber = input.nextInt();
    System.out.println("You entered " + wholeNumber + ".");
  }
}
```

Running this code (in a file named NumericScanner.java) should produce results like this:

```
Enter a whole number: 42
You entered 42.
```

If the input cannot be successfully converted to a valid integer, the program will produce an error (an exception) and end:

EXAMPLE

```
Enter a whole number: 4z
Exception in thread "main" java.util.InputMismatchException
    at java. Base/java.util.Scanner.throwFor(Scanner.java:939)
    at java. Base/java.util.Scanner.next(Scanner.java:1594)
    at java. Base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java. Base/java.util.Scanner.nextInt(Scanner.java:2212)
    at NumericScanner.main(NumericScanner.java:7)
```

The same thing happens when `nextInt()` tries to read a numeric value with a decimal:

EXAMPLE

```
Enter a whole number: 4.02
Exception in thread "main" java.util.InputMismatchException
    at java. Base/java.util.Scanner.throwFor(Scanner.java:939)
    at java. Base/java.util.Scanner.next(Scanner.java:1594)
    at java. Base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java. Base/java.util.Scanner.nextInt(Scanner.java:2212)
    at NumericScanner.main(NumericScanner.java:7)
```

It is important to make sure that the input is within the range of the data type chosen, for the `Scanner`'s next__() method and the variable to hold the input. Reading a value that is out of range will result in an error:

EXAMPLE

```
Enter a whole number: 2147483648
Exception in thread "main" java.util.InputMismatchException: For input string: "2147483648"
    at java. Base/java.util.Scanner.nextInt(Scanner.java:2264)
    at java. Base/java.util.Scanner.nextInt(Scanner.java:2212)
    at NumericScanner.main(NumericScanner.java:7)
```

Since the input 2147483648 is greater than the maximum value for the int data type (2147483647), the program should use the data type long for the variable. The `Scanner` method `nextLong()` can be used if it needs to work with such values.

When using the methods for reading floating-point types (`nextFloat()` and `nextDouble()`), an entry that cannot be parsed as a float or double value will result in an error. You will learn about dealing with these types of errors in a later tutorial.

SUMMARY

In this lesson, you considered common methods that the Scanner class provides for reading valid input as **string data**. These included char, int, and double values. You also explored how to read user **numeric input** using the Java Scanner class.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/