2.07 Scanner Class methods

**Scanning for User Input**

int testScore = 96;  
double interestRate = 0.045;  
String lastName = "Gosling";

Assignment statements like these are used frequently in programs but mostly for data that does not change. For example, if you write a program to calculate the interest on a loan, you wouldn't want to change the source code every time the interest rate changes. Wouldn't it be easier to simply enter the interest rate from the keyboard? Then the program would always work for any interest rate.

Accepting user input from the keyboard in earlier versions of Java was not always easy; however, Java 5.0 simplified the process by introducing the **Scanner** class. The Java API for the **Scanner class**(https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html) describes many useful methods for accepting keyboard input, but for now we will only be concerned with those that allow entry of numbers and Strings.

Before you can use the methods of the Scanner class to accept user input, you need a little better understanding of the importance of an object.

String phrase = "Four score and seven years ago";

Because **phrase** is declared to be of type **String**, it is an object and has access to all of the methods of the **String** class. Consequently, any method of the **String** class can be called simply by using dot notation as illustrated in the following statements:

int stringLength = phrase.length();  
int positionOfLetterA = phrase.indexOf("a");  
String sub1 = phrase.substring(5,10);

In each of these statements, the **String** object **phrase** is separated from a method of the **String** class by a dot (i.e. a period). The concept of a method should be pretty clear to you by now; it is a segment of code that performs a specific task. For example,

* **phrase.length()** invokes the **String** class' **length()** method to determine the length of the String literal assigned to the **phrase** object.
* **phrase.indexOf("a")** calls the **String** class' **indexOf()** method to determine what position the letter "a" occupies in the String literal.
* **phrase.substring(5,10)** invokes the **substring()** method of the **String** class to extract the characters between index positions 5 (inclusive) and 10 (exclusive) of the String literal.

It is a gross oversimplification, but think of a class simply as a set of methods and think of an object simply as a copy of a class. Since an object is a copy of a class, it has its own set of methods and can call the methods to perform their tasks using dot notation.

While allowing input from the keyboard increases the user-friendliness of a program, there is also a downside. Users often make mistakes, either accidentally or intentionally, which can cause a program simply to malfunction or, at worst, crash.

The methods of the Scanner class can handle the acceptance of user input, but you have to prompt the user for the input you want; don’t make them guess!

* Open the [**02.07 Virtual Lecture Notes (Part 1)**](https://lti.flvsgl.com/flvs-cat-content/r7tmpltcc9i2gjfmjkqtf5ch4h/flvs-cat-session/apcomputersciencea_v20/module02/lesson07/pop/03_07b/03_07b_pop01.htm).
* Read the instructions and begin exploring the Scanner class.

|  |
| --- |
| 02.07 Virtual Lecture Notes (Part 1)Overview of the Scanner Class The **Scanner** class includes methods for accepting user input from the keyboard, as indicated in the following abbreviated Method Summary table.  String: next() Finds and returns the next complete token from this scanner. String: NextLine() Advances this scanner past the current line and returns the input that was skipped. int: nextInt() scans the next token of the input as an int. double: nextDouble: Scans the next token of the input as a double.  Notice that the Java API format of the **Scanner** class and the **String** class follows the same basic pattern and includes the following information.   1. The name of the method. 2. A description of the method. 3. A parameter list, which in this case is empty for all four methods. 4. The return type of the method (on the left).   These four methods, **nextInt(), nextDouble()**, **next()**, and **nextLine()** can accept simple numeric or alphanumeric input from the keyboard. Which one of the **Scanner** class methods would you use to accept a decimal value entered by a user?  Before these methods can be used, a **Scanner** object must be created. Once a **Scanner** object exists, using **Scanner** class methods will follow the same pattern as calling **String** class methods (e.g. **object.method()**).  In Java, declaring a String object uses a shortcut. However, declaring objects of most other classes requires an intermediate step. For example, a **Scanner** class object can be created as follows:  Scanner in = new Scanner(System.in);  This statement uses Java’s reserved word **new** to construct a **Scanner** object named **in**. The **Scanner** object does not have to be named **in**, but it is commonly used by programmers for input objects. Object names should start with a lowercase letter and conform to the “camelCase” font style if there are additional words in the identifier name.  With a Scanner object declared, the following statement could be written to accept a decimal number typed in from the keyboard:  double number = in.nextDouble();  Does this statement look familiar? Dot notation is used to separate the object on the left from the method on the right. When this statement is executed, the program invokes the **nextDouble()** method on the **in** object by pausing to wait for the user to enter a decimal number and then press the Enter key.  Once the input is typed and the Enter key is pressed, the decimal number is assigned to the identifier called **number**. Since **in** is an object of the **Scanner** class, it can invoke any of the methods of the **Scanner** class.  There are a lot of details we are ignoring for now about objects, but that doesn’t mean you can’t begin using the methods of the **Scanner** class to accept user input. |

In order to use the methods of the Scanner class, an object of type Scanner must be constructed. The syntax of the statement to declare an object is the same for most classes, so once you know the pattern you can use it over and over again. Are you ready to make your programs more interactive?

* Open the [**02.07 Virtual Lecture Notes (Part 2)**](https://lti.flvsgl.com/flvs-cat-content/r7tmpltcc9i2gjfmjkqtf5ch4h/flvs-cat-session/apcomputersciencea_v20/module02/lesson07/pop/03_07b/03_07b_pop02.htm).
* Create a new project called 02.07 Scanner Methods in the Mod02 Lessons folder.
* Download the [**KeyboardInput.java**](https://lti.flvsgl.com/flvs-cat-content/r7tmpltcc9i2gjfmjkqtf5ch4h/flvs-cat-session/apcomputersciencea_v20/module02/lesson07/docs/03_07b/keyboardinput.java) file to the newly-created project folder.

|  |
| --- |
| 02.07 Virtual Lecture Notes (Part 2)Importing Classes The **Scanner** class is a member of Java’s utility package, not the standard java.lang package, so it must be imported. Import statements should immediately precede the class declaration, as shown below:  import java.util.Scanner;    //imports Scanner methods public class KeyboardInput { //…body of program between the curly braces. }  Objects can be declared on one line or two; the choice depends on the context of the program. To declare an object in two lines, you would use the following general pattern.  ClassName objectName; objectName = new ClassName(parameter list);  The first statement declares that there will be an object of some specific type. Substitute the name of the Class for ClassName and the name you want to use for the object for the objectName identifier. The second statement actually constructs the new object using any parameters that may be in the parentheses. (You will learn much more about the syntax of these statements at a future date).  These two statements can also be combined into a single statement as follows:  ClassName objectName = new ClassName( parameter list);  This statement accomplishes the same purpose as the previous two but saves one-line of typing. You may use either a one line or a two-line statement to declare objects unless you are instructed otherwise. Some Methods of the Scanner Class**Part 1** int: nextInt() Scans the next token of the input as an int.  The **nextInt()** method allows the user to enter an integer (e.g. -32, 0, 438, 32768, etc.) from the keyboard, as illustrated in the following example:  Scanner in; in = new Scanner(System.in); System.out.print("Please enter an integer value: "); int intValue = in.nextInt();  This code segment declares **in** to be a **Scanner** object in the first statement, and then the new object is actually constructed in the second statement. After the **in** object is constructed, the keyboard input methods of the **Scanner** class can be used. The third line prompts the user to enter an integer from the keyboard, and the fourth line assigns the input to **intValue** which is an **int** primitive data type. Once again, notice the use of dot notation.  Try it! Compile and run the KeyboardInput.java program to observe how simple it is to use the nextInt() method. What happens if you accidentally enter a decimal value or a String when using the nextInt() method? What happens if you comment out the print statement with the prompt?  An important point to realize is that you do not have to declare a new **Scanner** object in order to use each method. **Part 2** double: nextDouble() Scans the next token of the input as a double.  The **nextDouble()** method allows the user to enter a decimal value (e.g. -43.1567, 0.000009231, 1342395.66, etc.) from the keyboard, as illustrated in the following example:  System.out.print("Please enter a decimal value: "); double decimalValue = in.nextDouble();  The first line prompts the user to enter a decimal value. The second statement assigns the number entered to the variable called **decimalValue**, which is of type **double**.  Try it! Uncomment the next portion of the **KeyboardInput.java** program. Compile and run it. Observe its performance. What happens if you accidentally enter an integer value or a String when using the **nextDouble()** method? **Part 3** String: next() Finds and returns the next complete token from this scanner.  The **next()** method allows the user to enter one token. For now, we'll define a token as anything that does not contain a space. The **next()** method is useful for single words or numbers, although a number stored as a **String** could no longer be used as an **int** or a **double** in an arithmetic expression.  The following segment of code illustrates the use of the **next()** method:  System.out.print("Please enter a single word: "); String word = in.next();  The first line prompts the user to enter a word. The second line assigns the user input to a reference variable called **word**.  Try it! Uncomment the next portion of the **KeyboardInput.java** program. Compile and run it. Observe the interactivity and the output. What happens if you accidentally enter an integer or a decimal value when using the **next()** method? What happens if you enter more than one word before pressing Enter? **Part 4 (Very important!)** The **next()** method only accepts one token at a time. A blank space is the default delimiter separating tokens. If you ask the user to enter several pieces of information before pressing the Enter key, you will need a separate **next()** statement for each item of input. For example, the following code segment allows the user to enter three words, then press the Enter key.  System.out.print("Enter three words separated by spaces: "); String word1 = in.next(); String word2 = in.next(); String word3 = in.next();  In this example, each word (i.e. token) is assigned to a separate **String** object (i.e. word1, word2, or word3) because the **next()** method is used three different times.  Try it! Uncomment the next portion of the **KeyboardInput.java** program. Compile and run it. Observe what happens when you type in three words separated by a space and then press the Enter key. What happens if too few or too many words are entered? Run the program again, and this time, instead of pressing the space key, press the Enter key after typing each word. Are the tokens accepted? **Part 5 (Very, very important!)** String: nextLine() Advances this scanner past the current line and returns the input that was skipped.  Since the **next()** method only accepts one token, how could you input a sentence that contains many tokens separated by spaces? For example, run the **KeyboardInput.java** program again. When prompted to enter a single word, type in the following sentence.  Please put plenty of anchovies on my pizza.  You typed in a whole sentence, but only one word (the first token) was printed. What happened to the rest of it? Java ignored everything after the first word! Luckily, we can use the **nextLine()** method to capture the rest of the input stream as follows:  System.out.println("Please enter a sentence: "); String firstWord = in.next(); String restOfLine = in.nextLine(); System.out.println("First word: " + firstWord); System.out.println("Rest of sentence: " + restOfLine); String sentence = firstWord + restOfLine; System.out.println("Whole sentence: " + sentence);  Notice the use of concatenation to join the two strings. What do you predict the output of this code segment will be? Uncomment the last portion of the **KeyboardInput.java** program. Compile and run it. Type in a sentence and then press the Enter key. What happens if you only type one word? What’s the longest sentence you can enter? **Part 6 (Extremely important!)** Users are often prompted to enter a string of text characters separated by spaces (e.g. a sentence, a first and last name). The previous section shows one way to accept multiple tokens within a string of characters. The following is a slightly shorter way to accomplish the same goal and is a little more sophisticated.  String text = in.next(); text += in.nextLine();  Carefully examine how these two lines work together.   * The first line declares a **String**object called **text**. It also invokes the **next()**method, which accepts only the first token typed in from the keyboard. The = sign causes the single token to be assigned to the text object. * The second statement invokes the **nextLine()** method, which accepts all of the remaining tokens typed in from the keyboard once the enter key is pressed. The += shortcut assignment operator concatenates the single token in **text** with all of the remaining tokens, and re-assigns the entire input typed in by the user to the **text** object.   You will find many uses for these two lines when entering strings of text, so be sure you understand how they work. |

The methods of the Scanner class will add user-friendly interactivity to your programs. Be sure you understand how to use all of the methods covered in the Virtual Lecture Notes.