3.07 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.

Compile and run the KeyboradInputV1.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?

An important point to realize is that you do not have to declare a new **Scanner** object in order to use each method. What happens if you copy and paste the object declaration statements (i.e. the first two lines) into the code a second time?

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.

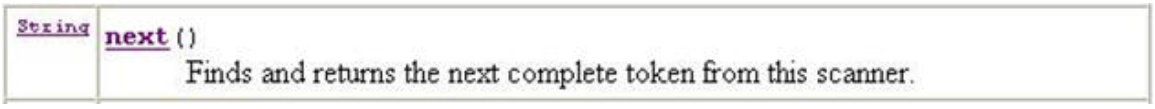
```
Scanner in = new Scanner(System.in);
System.out.print("Please enter a decimal value:  ");
double decimalValue = in.nextDouble();
```

In this segment of code a **Scanner** object called **in** is declared and constructed in a single statement. It is generally your choice whether to declare and construct objects in one or two statements. The **Scanner** object does not have to be named **in**, but it is a commonly used by programmers for input objects. Objects should start with a lowercase letter and conform to the "camel" font style if there are additional words in the identifier name. For clarity, sometimes the class and the object are given the same name except for capitalization of the first letter (e.g. if the class is named Scanner, the object could be named scanner); however, you could just as easily call the Scanner object something like keyboardInput.

Once the **Scanner** object is constructed the keyboard input methods can be used. The second line prompts the user to enter a decimal value and the third line assigns the number entered to the variable called **decimalValue** which is of type **double**.

Compile and run the KeyboardInputV2.java program and observe its performance. What happens if you accidentally enter an integer value or a String when using the **`nextDouble()`** method?

Part 3



The **`next()`** method allows the user to enter one token which for now is 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. A blank space is the default delimiter separating tokens, but other delimiters can also be defined (e.g. a dash, a tab, comma, etc.).

The following segment of code illustrates the use of the **next()** method..

```
Scanner in = new Scanner(System.in);
System.out.print("Please enter a single word:  ");
String word = in.next();
```

Here a **`Scanner`** object called **`in`** is declared and constructed in the first statement. After the **`Scanner`** object is constructed the keyboard input methods can be called. The second line prompts the user to enter a word and the third line assigns the user input to a reference variable called **`word`**.

Compile and run the KeyboardInputV3.java program and observe the interactivity and the output. What happens if you accidentally enter an integer or a decimal value when using the **`next()`** method?

Part 4 (Very Important!)

The **`next()`** method only accepts one token at a time. 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 3 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.

Compile and run the KeyboardInputV4.java program, 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 press the Enter key after typing each word. Are the tokens accepted?

Part 5 (Very, very important!)

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 KeyboardInputV3.java program again and type in the following sentence instead of just one word.

> 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, so we will use the **`nextLine()`** method to capture the rest of the input stream as follows.

```
Scanner in = new Scanner(System.in);
System.out.print("Please enter a sentence:  ");
String firstToken = in.next();
String restOfSentence = in.nextLine();
String sentence = firstToken + restOfSentence;
System.out.println();
System.out.println(firstToken);
System.out.println(restOfSentence);
System.out.println(sentence);
```

Notice the use of concatenation in the fifth line. What do you predict the output of this code segment will be? Compile and run the KeyboardInputV5.java program, 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.