GERMANNA
ACADEMIC CENTER
FOR EXCELLENCE

Java: Input and Output Handout

This handout demonstrates the basic use of two common programming tasks in Java: obtaining **inputs** from the user and **outputting** information to the console (also known as printing). It covers the basics of working with the Scanner class and the print methods. For information about working with files as input, please see the Academic Center for Excellence's (ACE) Java Programming: File Handling handout. Readers should already be familiar with *data types* and *variables*. For more information on these prerequisite topics, please see the ACE handout titled "Java: Datatypes and Variables."

Input

In programming, 'input' refers to gathering information from outside the program itself for use within it. In this handout, input refers to obtaining information via typing it on the keyboard. The standard approach to obtaining input in Java is to use the Scanner class, which is a commonly used class that streamlines the process of recording characters typed on the keyboard.

Importing the Scanner class

To use the Scanner class, it needs to be imported into the program. This can be done with the use of a simple import statement:

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

The Scanner class can also be used by importing the util library with a wildcard. The util library includes the Scanner class, so by importing everything with the wildcard, it is also imported.

```
import java.util.*;
```

Use of the wildcard import in this fashion, while functional, is discouraged in more complex programs.

Creating a Scanner Object

Once the Scanner class has been imported, a Scanner object must be instantiated (creating a functional example of a provided class). This is the same process as declaring any other variable except that the Scanner requires an input stream. An input stream is the source that the Scanner object will refer to for input. For most assignments at Germanna, the input stream will be *System.in*.

```
Scanner scnr = new Scanner(System.in);
```

Scanner Methods

With the Scanner object instantiated using the System.in input stream, the various methods in the Scanner class can now be utilized to collect different types of data from the keyboard. Which specific method to use will depend on the demands of the specific program being coded. Below is a brief reference for the most commonly used methods of the Scanner class.

Method Name	Description
next()	Scans over whitespace until a string is found, then returns that string
nextInt()	Performs the same as next(), but reads the input as an integer
nextDouble()	Performs the same as next(), but reads the input as a double
nextBoolean()	Performs the same as next(), but interprets the input as a boolean value
nextLine()	Reads a line of input as a string, discards the new line character

For more information, see Oracle's Java Documentation for the Scanner Class.

To use a particular method, there must be an instantiated Scanner object to call it through. To use the value collected by the method, that value must be assigned to a variable.

```
int userid = scnr.nextInt();
```

In the above example, the nextInt() method is called on the scnr object with the output of the method being assigned to the userid variable.

Input - Common Errors

Below are examples and explanations of several common errors students may encounter when working with Scanner objects.

- **Failing to import the Scanner class**

  Failing to import the Scanner class and then attempting to create a Scanner object, will result in *cannot find symbol* errors. The compiler will throw a *cannot find symbol* error for each time the Scanner class is referenced in the code.

  To address this problem, make sure to import the Scanner class. It is good practice to place import statements at the top of the file.

- **Using next() to collect numerical values**

  The next() method returns a String. This is not a problem so long as the program does not need to perform any mathematical calculations using the value.

  To address this problem, be mindful of what the program needs to do with a value that has been gathered. It may be a better option to use nextInt() or nextDouble() instead to meet the needs of the program.

- **Using next()/nextInt() and nextLine() in the same scope**

  Using the next() or nextInt() with nextLine() methods in the same program scope requires special care. The next(), or nextInt(), method will leave a newline character in the system buffer, which will prompt a subsequent nextLine() to return an empty string rather than waiting for the user to type something. It is best practice to call nextLine() immediately after next() or nextInt() to consume the leftover newline before using nextLine() to collect further input.

Output

When a program runs, the user does not see the code. That is only for the programmer's eyes. Thus, it is up to the programmer to determine what material to show to the user and how to output that material. In programming, 'output' refers to the process of presenting information to the user through some form such as printing it on a paper or displaying it on the screen. For the purpose of this handout, output refers to displaying information on the terminal screen. The most common way of accomplishing this is using methods found in System.out.

The two most used methods are `println()` and `print()`. The only difference between them, that is relevant to this handout, is that `println()` adds a new line character after whatever it displays, while `print()` does not. Use of these methods will commonly be referred to as "printing" or "printing to the screen".

Outputting Variables to the Console

When you use either of the print methods, you will need to supply the materials to be printed to the screen. There are several ways this can be done.

By using a string literal:

```
System.out.println("This is a string literal");
```

By referencing a variable:

```
int calculation = 12 * 12;
System.out.print(calculation);
```

Or through the use of concatenation, the process of creating a new string by combining two other strings.

```
String day_name = "Monday";
System.out.println("Today is " + day_name +".");
```

Execution of the above example will produce the following output:

```
Today is Monday.
```

It is important to recognize that the `print()` and `println()` methods will only accept a single value as an argument. Concatenation takes multiple values and combines them into one. You cannot supply multiple separate values separated by commas to these methods.

Output - Common Errors

Below are examples and explanations of several common errors students may encounter when working with the print methods.

- **Failing to properly terminate a string**
  Strings are a container and they must be properly closed in order for the system to work with them. Strings open with a quote and must be closed with a quote. Failing to do so will result in an error declaring "unclosed string literal".

- **Failing to properly call the desired method**
  When you call the print() or println() methods you need to be exact. Accidentally attempting to call system.out.println() or System.Out.println() will result in an error. The proper way to call the methods are:

```
System.out.println()
System.out.print()
```

Notice that '`System`' is capitalized, while '`out`', `print()` and `println()` are not. Attempting to use `system.out.println()` will result in an error because there is no "system" module.

The Academic Center for Excellence (ACE) offers free on-campus or online tutoring appointments. For further assistance with programming concepts, please call ACE at (540) 891- 3017 or email us at ACE@germanna.edu.