STUDENT LESSON

A7 – Simple I/O

INTRODUCTION: The input and output of a program's data is usually referred to as I/O. There are many different ways that a Java program can perform I/O. In this lesson, we present some very simple ways to handle text input typed in at the keyboard as well as how to format text to the screen. The Advanced Placement subset does not require that you know how to use any specific input and output classes, only that you know how to use I/O in some manner. This curriculum will use the Scanner class and the `printf()` method provided with Java 1.5.

The key topics for this lesson are:

A. Reading Input with the `Scanner` Class
B. Multiple Line Stream Output Expressions
C. Formatting Output

VOCABULARY:

CONVERSION	FLAGS
PRECISION	printf
Scanner	System.in
System.out	WIDTH

DISCUSSION:

A. Reading Input with the `Scanner` Class

1. Some of the programs from the preceding lessons have been written without any flexibility. To change any of the data values in the programs, it is necessary to change the variable initializations, recompile the program, and run it again. Sometimes prompting the user for a value and then processing the data is more efficient and convenient.

2. However, accepting user input in Java can be complex. Throughout this curriculum, we will use the `Scanner` class to make processing input easier and less tedious.

3. Just as the System class provides `System.out` for output, there is an object for input, `System.in`. Unfortunately, Java's `System.in` object does not directly support convenient methods for reading numbers and strings. We need to have a class sitting between the `System.in` object and ourselves to filter what comes through. This is what the `Scanner` class does.

4. Scanner is part of the java.util package, so we need to start off by adding the Scanner class to our import section.

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

5. Next, we create our Scanner object and pass in the System.in object.

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

   This tells the Scanner to look at the System.in object for all of its input. In Lesson A13, *Exceptions and File I/O*, we will learn how to change the object we pass to the Scanner so that we can read the data stored in text files.

6. Here are some example statements:

```
int num1;
double bigNum;
boolean isTrue;

num1 = in.nextInt( );
bigNum = in.nextDouble( );
isTrue = in.nextBoolean( );
```

   When the statement `num1 = in.nextInt()` is encountered, the program pauses until an appropriate value is entered on the keyboard.

7. Any whitespace (spaces, tabs, newline) will separate input values. When reading values, whitespace keystrokes are ignored.

8. When requesting data from the user via the keyboard, it is good programming practice to provide a prompt. An unintroduced input statement leaves the user hanging without a clue of what the program wants. For example:

```
System.out.print("Enter an integer -->  ");
number = in.nextInt();
```

B. Multiple Line Stream Output Expressions

1. We have already used examples of multiple output statements such as:

```
System.out.println("The value of sum = " + sum);
```

2. When the length of an output statement exceeds one line of code, it can be broken up several different ways:

```
System.out.println("The sum of " + num1 + " and " + num2 +
                   " = " + (num1 + num2));
```

   or

```
System.out.print("The sum of " + num1 + " and " + num2);
System.out.println( " = " + (num1 + num2));
```

3. You cannot break up a `String` constant and wrap it around a line.

```
System.out.print("A long string constant must be broken
up into two separate quotes.  This will NOT work.");

System.out.print("A long string constant must be broken up"
               + " into two separate quotes. This will work.");
```

C. Formatting Output

1. To format, we will learn a new printing method, `printf()`. It works similarly to the `print()` and `println()` methods that you have already been using to output text.

2. The `printf()` method takes two arguments. The first one is the formatting String, a special sequence of characters that tells `printf()` how to display the second argument. The syntax for the formatting String is:

   %[flags][width][.precision]conversion

   The '%' sign tells the `printf` method that formatting is coming. All of your formatted String constants will start with %. It does not have to be the very first thing in your String constant, just the first part of any formatted text.

3. The last part of the formatting String, conversion, is one of the most important parts. It is what determines how the `printf()` method reacts to a message you send it. The most important conversion tags for you to know are 's', 'd', and 'f'. 'd' is used for integers (base-10 notation), 'f' is for numbers with decimal places (doubles), and 's' is for String literals. The conversion tag always comes at the end of the formatting String.

Conversion Tag	Usage Type	Example
s	String literals	`printf("%s", "Sam")`
d	ints	`printf("%d", 5182)`
f	doubles	`printf("%f", 2.123456)`

4. Precision is very easy and straightforward. When using a formatting String with the 's' conversion tag, this will tell `printf()` the maximum number of characters to print out. When used with the 'f' conversion tag, you can specify how many decimal places to print out, rounded to the closest number. If you don't specify how many decimal places to display with 'f' then it will default to six places.

```
System.out.printf("%.2s", "Hello") -> He
System.out.printf("%.10s", "Hello") -> Hello
System.out.printf("%.5f", Math.PI) -> 3.14159
System.out.printf("%.4f", Math.PI) -> 3.1416
System.out.printf("%f", Math.PI) -> 3.141593
```

5. Width tells `printf()` the minimum number of characters to print out. This allows for creating right-aligned lists or menus. `printf()` does not

distinguish between normal characters and special characters (escape sequences), so `"Prices:"` and `"Prices:\n"` are two different sizes. If you want to print your data out left-aligned, you can simply add a '-' character to the left of the width value.

Note: In the example below, the first line has a width of 11 instead of 10 to adjust for the `\n` error. Because numeric entries cannot use these special characters, it is better to use `println()` to separate lines when utilizing `printf()`. The example below shows one instance of using the `\n` character as well as doing two columns of formatted output.

```
System.out.printf("%-10s", "Name:");
System.out.printf("%11s", "Price:\n");
System.out.printf("%-10s", "Soda");
System.out.printf("%10.2f", 10.25);
System.out.println();
System.out.printf("%-10s", "Candy");
System.out.printf("%10.2f", 1.50);
```

Run Output:

```
Name:       Price:
Soda         10.25
Candy         1.50
```

6. Flags are special characters that give special properties to the values passed in. Adding a '+' sign in the formatting String will give numbers a positive or negative sign when printed. Putting in a '(' will cause negative numbers to be enclosed with parentheses. The most useful of the flags is ',' because it will add commas into large numbers (in the correct spot for the region, i.e. Japan puts numbers into groups of four unlike the US, which puts numbers in groups of three). To get a dollar sign directly before your printed value, place the '$' character directly before the '%' sign.

```
System.out.printf("%,d", 12345678) -> 12,345,678
System.out.printf("$%,d", 12345678) -> $12,345,678
System.out.printf("%,(d", 12345678) -> 12,345,678
System.out.printf("%,(d", -12345678) -> (12,345,678)
```

7. You may put multiple arguments in one call to `printf()` for organizational purposes. Simply put multiple formatting Strings in the first passed argument to `printf()` and then add your additional arguments, separated by commas.

```
double mySum = 123.456
System.out.printf("%10s %10.2f", "Total:", mySum);
```

8. This curriculum uses just a few of the many things that `printf()` is capable of. Once you get more familiar with using the formatting options shown in this guide, you can look at the API for the Formatter class for more information on formatting Strings. The `printf()` method is an easy way to manage the Formatter class in a way that is very similar to the common `print()` and `println()` functions.

SUMMARY/ REVIEW: These two classes, `Scanner` and `Formatter`, will be used in many programs. They provide you with the flexibility needed to start creating very robust programs. You can now interact with a user instead of simply displaying results on the screen. Go back through some of the programs you have already made and see which ones could benefit from accepting user input or formatting data.

ASSIGNMENT: Lab Assignment A7.1, *GroceryList*
Worksheet A7.1, *Scanner Review*
Worksheet A7.2, *printf Review*