

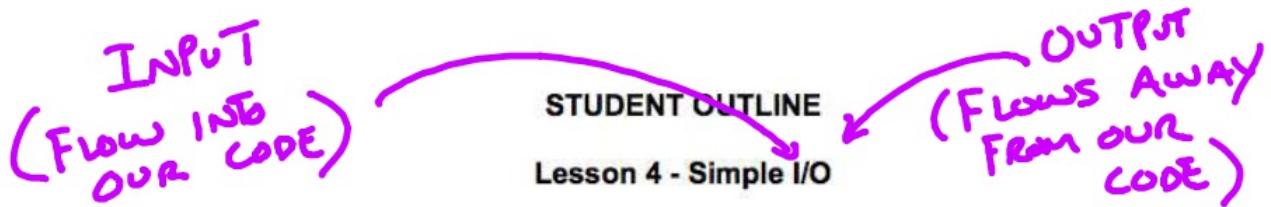
(int) 7.75 % 2 + 8.25 =

- a) 11.25 b) 11 c) 9.25 d) 9 e) undefined

T / F

Class DrawingTool requires a SketchPad object to be passed as a parameter to its constructor.

DrawingTool pen = new DrawingTool();  
↑  
DEFAULT CONSTRUCTOR!



**INTRODUCTION:** Input and output of program data are usually referred to as I/O. There are many different ways that a Java program can perform I/O (input and output). In this lesson, we present some very simple ways to handle simple text input typed in at the keyboard.

The key topics for this lesson are:

- A. Reading Input with the `ConsoleIO` Class
- B. Multiple Line Stream Output Expressions
- C. Formatting Output
- D. String Objects
- E. String Input

**VOCABULARY:** `ConsoleIO`      `Format`  
`String`      `CONCATENATE`

**DISCUSSION:** A. Reading Input with the `ConsoleIO` Class

1. Some of the programs from the preceding lessons have not been written in the most efficient manner. To change any of the data values in the programs, it would be necessary to change the variable initializations, recompile the program, and run it again. **It would be more practical if the program could ask for new values for each type of data and then compute the desired output.**
2. However, accepting user input in Java has some technical complexities. **Throughout this curriculum guide, we will use a special class, called `ConsoleIO`, 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.
4. To use the `ConsoleIO` class in a program, you first need to import from the `chn.util` package with the statement  
`import chn.util.*;`

I/O IS NOT  
ON THE AP  
EXAM

THANKS  
Dr. N!!

*PERUSE THIS!!*

*(Keyboard)*

5. To use a `ConsoleIO` method, you need to construct a `ConsoleIO` object.

```
ConsoleIO console = new ConsoleIO();
```

Next, call one of `ConsoleIO` methods

```
int n = console.readInt();
double d = console.readDouble();
boolean done = console.readBoolean();
String token = console.readToken();
String line = console.readLine();
```

6. Here are some example statements:

```
int num1;
double bigNum;
String line;
num1 = console.readInt();
bigNum = console.readDouble();
line = console.readLine();
```

When the statement `num1 = console.readInt()` is encountered, execution of the program is suspended until an appropriate value is entered on the keyboard.

7. Any whitespace (spaces, tabs, newline) will separate input values. When reading values, white space keystrokes are ignored. If it is desirable to input both whitespace and non-whitespace characters, the method `readLine()` is required.

8. The `readToken()` method reads and returns the next token from the current line. A token is a String of characters separated by the specified delimiters (whitespace). For example when the following code fragment is executed

```
String input = console.readToken();
System.out.print(input);
```

when given an input line of

twenty-three is my favorite prime number

would output

twenty-three

since this is the first string of characters read before a whitespace value (space) is encountered.

9. When requesting data from the user via the keyboard, it is good programming practice to provide a prompt. An un-introduced input statement leaves the user hanging without a clue of what the program wants and is quite rude. A good example:

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

*DEFAULT  
DELIMITER*

## B. Multiple Line Output Expressions

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

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

2. There will be occasions when the length of an output statement exceeds one line of code. This 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. This is not valid:

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

However, this will work:

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

KEEP + ON  
FIRST LINE

NOT ON AP  
TEST

✗

## C. Formatting Output

1. Formatting output in Java has some technical complexities. Throughout this curriculum guide, we will use a special class, called **Format**, to format numerical and textual values for a properly aligned output display.

2. The **Format** methods are available by including the import directive,

```
import apcslib.*;
```

at the top of the source code.

3. The basic idea of formatted output is to allocate the same amount of space for the output values and align the values within the allocated space. The space occupied by an output value is referred to as **the field** and the number of character allocated to a field is **its field width**.

4. To format an integer you would use the following expressions:

See Handout H.A.4.2,  
Format Class for a  
complete list of  
formatting features.

```
Format.left(int_expression, fieldWidth);
Format.center(int_expression, fieldWidth);
Format.right(int_expression, fieldWidth);
```

where *int\_expression* is an arithmetic expression whose value is an **int** and *fieldWidth* designates the field width. Example:

```
int i1 = 1234;
int i2 = 567;
int i3 = 891011;

System.out.println(Format.left(i1, 15) + "left");
System.out.println(Format.center(i2, 15) + "center");
System.out.println(Format.right(i3, 15) + "right");
```

Run output:

```
1234      left
      567    center
     891011right
```

5. The same type of format statements can be used for a **String** value.

```
Format.left(String_expression, fieldWidth);
Format.center(String_expression, fieldWidth);
Format.right(String_expression, fieldWidth);
```

where *String\_expression* is an expression that evaluates to a **String** and *fieldWidth* designates the field width. Example:

```
String s1 = "left";
String s2 = "center";
String s3 = "right";

System.out.println(Format.left(s1, 15) + "String");
System.out.println(Format.center(s2, 15) + "String");
System.out.println(Format.right(s3, 15) + "String");
```

Run output:

```
left      String
      center  String
      right   String
```

← 15 →  
WIDE

6. To format a real number (**float** or **double**) we need additional arguments to specify the decimal places:

```
Format.left(real_expression, fieldWidth, decimalPlaces);
Format.center(real_expression, fieldWidth, decimalPlaces);
Format.right(real_expression, fieldWidth, decimalPlaces);
```

EXTRA  
(RIGHT  
OF  
POINT)

where *real\_expression* is an arithmetic expression whose value is either a **float** or a **double** and *decimalPlaces* designates the number of digits shown to the right of the decimal point. The value for *fieldWidth* must be at least as large as the value of *decimalPlaces* plus two. Example:

```
double d1 = -123.4e-5;
double d2 = 678.9;
double d3 = 12345.6789;

System.out.println(Format.left(d1, 15, 6) + "left");
System.out.println(Format.center(d2, 15, 3) + "center");
System.out.println(Format.right(d3, 15, 2) + "right");
```

Run output:

```
-0.001234      left
678.900       center
12345.68right
```

#### D. String Objects

1. Next to numbers, *strings* are the most important data type that most programs use. A string is a sequence of characters such as "Hello". In Java, strings are enclosed in quotation marks, which are not themselves part of the string.

2. String objects can be constructed in two ways:

```
String name = "Bob Binary";
String anotherName = new String("Betty Binary");
```

SHORTCUT  
ONLY FOR  
Strings

Due to the usefulness and frequency of use of strings, a shorter version without the keyword **new**, was developed as a short-cut way of creating a **String** object. This creates a **String** object containing the characters between quote marks, just as before. A **String** created in this method is called a **String literal**. Only Strings have a short-cut like this. All other objects are constructed by using the **new** operator.

3. Assignment can be used to place a different *string* into the variable.

```
name = "Boris";
anotherName = new String("Bessy");
```

SAME  
NET EFFECT.  
BOTH CONSTRUCT  
A STRING OBJECT

4. The number of characters in a string is called the *length* of the string. For example, the length of "Hello World!" is 12. You can compute the length of a String with the `length` method.

```
int n = name.length();
```

Unlike numbers, strings are objects. Therefore, you can call methods on strings. In the above example, the length of the String object `name` is computed with the method call `name.length()`.

**CONSTRUCTS  
THE OBJECT, BUT  
IT HAS NO  
CHARACTERS**

5. A String of *length zero*, containing no characters, is called the *empty string* and is written as `"`. For example:

```
String empty = "";
```

6. Programmers often make one String object from two strings with the `+` operator, which *concatenates* (connects) two or more strings into one string. Concatenation and these string messages are illustrated below.

```
public class DemoStringMethods
{
    public static void main(String[] args)
    {
        String a = new String("Any old");
        String b = " String";
        String aString = a + b; // aString is "Any old String"

        // Show string a
        System.out.println("a: " + a);

        // Show string b
        System.out.println("b: " + b);

        // Show the result of concatenating a and b
        System.out.println("a + b: " + aString);

        // Show the number of characters in the string
        System.out.println("length: " + aString.length());
    }
}
```

Run output:

```
a: Any old
b: String
a + b: Any old String
length: 14
```

7. Notice that using strings in the context of input and output statements is identical to using other data types.
8. The `String` class will be covered in depth in a later lesson. For now you will use strings for simple input and output in programs.

E. String Input

By DEFAULT →

1. The `ConsoleIO` class has two methods for reading textual input from the keyboard.
2. The `readToken` message returns a reference to a `String` object that has from zero to many characters typed by the user at the keyboard. A *token* is a sequence of printable characters separated from the next word by *white space*. White space is defined as blank spaces, tabs, or newline characters in the input stream. White space separates `ints` and `doubles` on input. White space also separates words on input. When input from the keyboard, `readToken` stops adding text to the `String` object when the first white space is encountered on the input stream from the user.
3. A `readLine` message returns a reference to a `String` object that contains from zero to many characters entered by the user. With `readLine`, the `String` object may contain blank spaces and tabs. The newline marker is not included. It is discarded from the input stream.

↑  
STOP  
SIGN

4. Input from these string messages is illustrated below.

  
**EXPERIMENT**  
IN  
**BLUEJ**

```
import chn.util.*;  
  
public class DemoStringInput  
{  
    public static void main(String[] args)  
    {  
        ConsoleIO keyboard = new ConsoleIO();  
        String word1, word2, anotherLine;  
  
        // ask for input from the keyboard  
        System.out.print("Enter a line: ");  
  
        // grab the first "word"  
        word1 = keyboard.readToken();  
  
        // grab the second "word"  
        word2 = keyboard.readToken();  
  
        // ask for input from the keyboard  
        System.out.print("Enter another line: ");  
  
        // discard any remaining input from previous line  
        // and read the next line of input  
        anotherLine = keyboard.readLine();  
  
        // output the strings  
        System.out.println("word1 = " + word1);  
        System.out.println("word2 = " + word2);  
        System.out.println("anotherLine = " + anotherLine);  
    }  
}
```

Run output:

```
Enter a line: Hello World! This will be discarded.  
Enter another line: This line includes whitespace.  
word1 = Hello  
word2 = World!  
anotherLine = This line includes whitespace.
```

**SUMMARY/  
REVIEW:**

These two classes, **ConsoleIO** and **Format**, will be used in many programs. The labs in this lesson will provide an opportunity to practice using simple I/O and formatting.

**ASSIGNMENT:**

Lab Exercise, L.A. 4.1, *Change*  
Lab Exercise, L.A.4.2, *CarRental*