STUDENT OUTLINE

Lesson 3 - Data Types in Java

INTRODUCTION: As with most high level languages, Java provides standard data types to store information. Java is a richly *typed* language that gives the programmer a wide variety of data types to use. In this lesson you will declare variables, store values in them, and print out their values using the `System.out` object.

The key topics for this lesson are:

A. Identifiers in Java
B. Basic Data Types in Java
C. Declaring and Initializing Variables in Java
D. Printing Variables Using the `System.out` object
E. ASCII Code Values and Character Data
F. Assignment Statements and Math Operators

VOCABULARY:

IDENTIFIER	KEYWORDS
`int`	**`char`**
`boolean`	**`double`**
`float`	`String`
ESCAPE SEQUENCE	TYPE
TYPE CONVERSION	ASSIGNMENT STATEMENT
MODULUS	ASCII

DISCUSSION: A. <u>Identifiers in Java</u>

1. An identifier is a name that will be used to describe classes, methods, constants, variables, and other items.

2. The rules for writing identifiers in Java are:
   a. Identifiers must begin with a letter.
   b. Only letters, digits, or underscore may follow the initial letter.
   c. The blank space cannot be used.
   d. Identifiers cannot be reserved words. Reserved words or keywords are only for system use.

See Handout H.A.3.1, Reserved Words in Java.

3. Java is a case sensitive language. That is, Java will distinguish between upper and lower case letters in identifiers. Therefore:

   `grade` and `Grade` are different identifiers

4. A good identifier should help describe the nature or purpose of that function or variable. It is better to use

`grade` instead of `g`, `number` instead of `n`.

5. However, avoid excessively long or "cute" identifiers such as:

   `gradePointAverage` or `bigHugeUglyNumber`

   Remember that our goal is to write code that is easy to read and professional in nature.

6. Programmers will adopt different styles of using upper and lower case letters in writing identifiers. The reserved keywords in Java must be typed in lower case text, but identifiers can be typed using any combination of upper and lower case letters.

7. The following conventions will be used throughout the curriculum guide:

   a. A single word identifier will be written in lower case only. Examples: `grade`, `number`, `sum`.
   b. If an identifier is made up of several words, the first letter will be lower case. Subsequent words will begin with upper case. Some examples are: `stringType`, `passingScore`, `largestNum`.
   c. Identifiers used as constants will be fully capitalized. Examples: `PI`, `MAXSTRLEN`.

B. Basic Data Types in Java

1. Java provides eight primitive data types: **`byte`**, **`short`**, **`int`**, **`long`**, **`float`**, **`double`**, **`char`** and a **`boolean`**. The data types **`byte`**, **`short`**, **`int`**, and **`long`** are for integers, and the data types **`float`** and **`double`** are for real numbers.

2. Integer type - any positive or negative number without a decimal point.
   a. Examples: `7   -2   0   2025`.

3. Floating Point type - any signed or unsigned number with a decimal point.
   a. Examples: `7.5   -66.72   0.125   5`.
   b. A floating point value cannot contain a comma or $ symbol.
   c. A floating point value must have a decimal point.
   d. Invalid examples: `1,234.56   $66.95   125   7,895`
   e. Floating point values can be written using scientific notation:
   `1625. = 1.625e3      .000125 = 1.25e-4`

4. The following table summarizes the bytes allocated and the resulting size.

	Size	Minimum Value	Maximum Value
`byte`	1 byte	-128	127
`short`	2 bytes	-32768	32767
`int`	4 bytes	-2147483648	2147483647
`long`	8 bytes	-9223372036854775808	9223372036854775807
`float`	4 bytes	-3.40282347E+38	3.40282347E+38
`double`	8 bytes	-1.79769313486231570E+308	1.79769313486231570E+308

5. Character type - letters, digits 0..9, and punctuation symbols.
   a. Examples: `'A'`, `'a'`, `'8'`, `'*'`
   b. Note that a character type must be enclosed within single quotes.
   c. Java character types are stored using 2 bytes, usually according to the ASCII code. ASCII stands for American Standard Code for Information Interchange.

See Handout H.A.3.2, *ASCII Characters - A Partial List.*

   d. The character value `'A'` is actually stored as the integer value 65. Because a capital `'A'` and the integer 65 are physically stored in the same fashion, this will allow us to easily convert from character to integer types, and vice versa.
   e. Using the single quote as a delimiter leads to the question about how to assign the single quote (`'`) character to a variable. Java provides escape sequences for unusual keystrokes on the keyboard. Here is a partial list:

Character	Java Escape Sequence
Newline	`'\n'`
Horizontal tab	`'\t'`
Backslash	`'\\'`
Single quote	`'\''`
Double quote	`'\"'`
Null character	`'\0'`

6. Data types are provided by high level languages to minimize memory usage and processing time. Integers and characters require less memory and are easier to process. Floating-point values require more memory and time to process.

7. The final primitive data type is the type **`boolean`**. It is used to represent a single **`true`**/**`false`** value. A **`boolean`** value can have only one of two values:

   `true`   **`false`**

   In a Java program, the words **`true`** and **`false`** always mean these **`boolean`** values.

C. Declaring and Initializing Variables in Java

1. A variable must be declared before it can be initialized with a value. The general syntax of variable declarations is:

   ```
   data_type  variableName;
   ```

   for example

   ```
   int number;
   char ch;
   ```

2. Variables can be declared near the top of the method or in the midst of writing code. Variables can also be declared and initialized in one line. The following example program illustrates these aspects of variable declaration and initialization.

   Program 3-1

   ```
   public class DeclareVar
   {
     public static void main(String[] args)
     {
       // first is declared and initialized
       // second is just initialized
       int first = 5, second;
       double x;
       char ch;
       boolean done;

       second = 7;
       x = 2.5;
       ch = 'T';
       done = false;

       int sum = first + second;
     }
   }
   ```

   a. Multiple variables can be declared on one line.
   b. Initialization is accomplished with an equal (=) sign.
   c. Initialization can occur at declaration time or later in the program. The variable `sum` was declared and used in the same line.

3. Where the variables are declared is a matter of programming style. Your instructor will probably have some preferences regarding this matter.

D. Printing Variables Using the `System.out` Object

1. The `System.out` object is defined in each Java program. It has methods for displaying text strings and numbers in plain text format on the system display, which is sometimes referred to as the “console”. For example:

Program 3-2

```
public class PrintVar
{
  public static void main(String[] args)
  {
    int  number = 5;
    char  letter = 'E';
    double average = 3.95;
    boolean done = false;

    System.out.println("number = " + number);
    System.out.println("letter = " + letter);
    System.out.println("average = " + average);
    System.out.println("done = " + done);
    System.out.print("The ");
    System.out.println("End!");
  }
}
```

Run output:

```
number = 5
letter = E
average = 3.95
done = false
The End!
```

2. Method `System.out.println` displays (or prints) a line of text in the console window. When `System.out.println` completes its task, it automatically positions the output cursor (the location where the next character will be displayed) to the beginning of the next line in the console window (this is similar to pressing the *Enter* key when typing in a text editor—the cursor is repositioned at the beginning of the next line in your file).

3. The expression

   ```
   "number = " + number
   ```

   from the statement

   ```
   System.out.println("number = " + number);
   ```

   uses the + operator to "add" a string (the literal `"number = "`) and number (the `int` variable containing the number 5). Java has a version of the + operator for *String concatenation* that enables a string and a value of another data type to be concatenated (added). The result of this operation is a new (and normally longer) string. String concatenation is discussed in more detail in the next lesson.

4. The lines

   ```
   System.out.print("The ");
   System.out.println("End!");
   ```

   of Program 3-2 display one line in the console window. The first statement uses `System.out`'s method, `print`, to display a string. Unlike `println`, `print` does not position the output cursor at the beginning of the next line in the console window after displaying its argument. The next character displayed in the console window appears immediately after the last character displayed with `print`.

5. Note the distinction between sending a text constant, `"number = "`, versus a variable, `number`, to the `System.out` object. A **`boolean`** variable will be printed out as its representation of **`true`** or **`false`**.

E. ASCII Code Values and Character Data

1. As mentioned earlier in section B.5, a character value can easily be converted to its corresponding ASCII integer value.

2. A character value is stored using two byte of memory, which consists of 16 bits of binary (0 or 1) values.

3. The letter 'A' has the ASCII value of `65`, which is stored as the binary value `0000000001000001`. This is illustrated in the following program fragment:

   ```
   char letter = 'A';
   System.out.println("letter = " + letter);

   System.out.print("its ASCII value = ");
   System.out.print((int)letter);
   ```

   Run output:

   ```
   letter = A
   its ASCII value = 65
   ```

   The statement `(int)letter` is called a type conversion. The data type of the variable is converted to the outer type inside of the parentheses, if possible.

4. In Java, you can make a direct assignment of a character value to an integer variable, and vice versa. This is possible because both an integer and a character variable are ultimately stored in binary. However, it is better to be more explicit about such conversions by using type conversions. For example, the two lines of code below assign to `position` the ASCII value of `letter`.

```
char letter = 'C';  // ASCII value = 67
int position;

position = letter;  // This is legal, position now equals 67
```

vs.

```
position = (int)letter;  // This is easier to understand.
```

More detail about type conversions follows in section F. 9.

F. Assignment Statements and Math Operators

1. An assignment statement has the following basic syntax:

   `variable = expression;`

   a. The *`expression`* can be a literal constant value such as 2, 12.25, 't'.
   b. The *`expression`* can also be a numeric expression involving operands (values) and operators.
   c. The = operator returns the value of the expression. This means that the statement

      ```
      a = 5;
      ```

      assigns 5 to the variable and returns the value `5`. This allows for chaining of assignment operators.

      ```
      a = b = 5;
      ```

      The assignment operator (=) is right-associative. This means that the above statement is really solved in this order:

      ```
      a = (b = 5);// solved from right to left.
      ```

      Since (`b = 5`) returns the integer 5, the value 5 is also assigned to variable *`a`*.

2. Java provides 5 math operators as listed below:

`+`	Addition, as well as unary `+`
`-`	Subtraction, as well as unary `-`
`*`	Multiplication
`/`	Real and integer division
`%`	Modulus, remainder of integer or floating point division

3. The numerical result and data type of the answer depends on the type of operands used in a problem.

4. For all the operators, if both operands are integers, the result is an integer. Examples:

`2 + 3 = 5` (integer)	`9 - 3 = 6` (integer)
`4 * 8 = 32` (integer)	`11/2 = 5` (integer – **Note!**)

5. If either of the operands is a float type, the result is a float type. Examples:

   `2 + 3.000 = 5.000` (float)
   `25 / 6.75 = 3.7037` (float)
   `11.0 / 2.0 = 5.5` (float)

   a. When an integer and a float are used in a binary math expression, the integer is promoted to a float value, and then the math is executed.
   b. In the example `2 + 3.000 = 5.000`, the integer value 2 is promoted to a float (`2.000`) and then added to the `3.000`.

6. The modulus operator (`%`) returns the remainder of dividing the first operand by the second. For example:

```
10 % 3 = 1    2 % 4 = 2    16 % 2 = 0    27.475 % 7.22 = 5.815
```

7. Changing the sign of a value can be accomplished with the negation operator (`-`), often called the unary (`-`) operator. A unary operator works with only one value. Applying the negation operator to an integer returns an integer, while applying it to a float returns a float value. For example:

```
-(67) = -67      -(-2.345) = 2.345
```

8. To obtain the fractional answer to a question like `11/2 = 5.5`, a type conversion must be applied to one of the operands.

```
(double)11/2           results in 5.5
   11.000/2            then we do division
      5.5
```

   The type conversion operators are unary operators with the following syntax:

   `(`***`type`***`) operand`

9. There is more to cover regarding operators in Java. Topics such as math precedence and assignment operators will be covered in a later lesson.

SUMMARY/ REVIEW: This lesson has covered a great amount of detail regarding the Java language. At first you will have to memorize the syntax of data types, but with time and practice, fluency will come.

ASSIGNMENT: Lab Exercise, L.A.3.1, *MathFun*
Lab Exercise, L.A.3.2, *Easter*