STUDENT LESSON

A3 – Primitive Data Types

STUDENT LESSON

A3 – Primitive Data Types

INTRODUCTION:

When designing a class, one of the programmer's jobs is to define the attributes of the class. Java allows two different types of attributes: objects and primitive data types. Using objects as attributes will come naturally as you become more used to OOP. Java provides several primitive data types to store basic information and uses objects to fill in gaps that are not provided with the primitive data types. Java is a richly typed language, which means that it gives the programmer a wide variety of data types to use. As the name suggests, primitive data types are very basic in nature. In this lesson you will declare variables, store values in them, learn operations to manipulate and use those values, and print out the values using the `System.out` object.

The key topics for this lesson are:

A. Identifiers in Java
B. Primitive Data Types in Java
C. Declaring and Initializing Variables in Java
D. Printing Variables Using the `System.out` Object
E. Assignment Statements
F. Math Operators
G. Precedence of Math Operators
H. Assignment Operators
I. Increment and Decrement Operators

VOCABULARY:

ASCII	ASSIGNMENT OPERATOR
`boolean`	**`char`**
DECREMENT OPERATOR	**`double`**
ESCAPE SEQUENCE	**`float`**
INCREMENT OPERATOR	IDENTIFIER
`int`	MODULUS OPERATOR
PRECEDENCE	PRIMITIVE DATA TYPE
RESERVED WORDS	STRING LITERAL
TYPE CONVERSION	

DISCUSSION:

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

1. An identifier is a name that will be used to describe classes, methods, constants, variables; anything a programmer is required to define.

2. The rules for naming identifiers in Java are:

    - Identifiers must begin with a letter.
    - Only letters, digits, or an underscore may follow the initial letter.
    - The blank space cannot be used.

See Handout A3.1, ***Reserved Words in Java***

- Identifiers cannot be reserved words. Reserved words or keywords are already defined in Java. These include words such as *new*, *class*, *int*, etc.

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

   Be careful both when naming identifiers and when typing them into the code. Be consistent and don't use both upper and lower case names for the same identifier.

4. A good identifier should help describe the nature or purpose of whatever it is naming. For a variable name, it is better to use

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

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

   ```
   gradePointAverageForStudentsAToZ
   or
   bigHugeUglyNumberThatIsPrettyPointlessButINeedItAnyway
   ```

   Remember that the goal is to write code that is professional in nature; other programmers need to understand your code.

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 this curriculum guide:

   - A single word identifier will be written in lower case only. Examples: `grade`, `number`, `sum`.
   - Class names will begin with upper case. Examples: String, DrawingTool, SketchPad, Benzene.
   - If an identifier is made up of several words, all words beyond the first will begin with upper case. Examples: `stringType`, `passingScore`, `largestNum`, `DrawHouse`, `SketchPad`.
   - Identifiers used as constants will be fully capitalized. Examples: `PI`, `MAXSTRLEN`.

B. <u>Primitive Data Types in Java</u>

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

2. The College Board Advanced Placement (AP) Examinations only require you to know about the **int**, **double**, and **boolean** data types. This curriculum will, from time to time, also use the **char** type when appropriate.

3. An integer is any positive or negative number without a decimal point.

   Examples: `7   -2   0   2025`

4. A double is any signed or unsigned number with a decimal point. Doubles cannot contain a comma or any symbols. A double value can be written using scientific notation.

   - Valid numbers: `7.5   -66.72   0.125   5`
   - Invalid numbers: `$37,582.00   #5.0   10.72%`
   - Scientific notation: `1625. = 1.625e3   .00125 = 1.25e-4`

   (Note: When applying 5 to a double variable, Java will automatically add the decimal point for you.)

5. 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

6. Character type consists of letters, digits 0 through 9, and punctuation symbols. A character must be enclosed within single quotes.

   Examples: `'A'`, `'a'`, `'8'`, `'*'`

See Handout A3.2, *ASCII Characters - A Partial List*

Java characters are stored using 2 bytes according to the ASCII code. ASCII stands for American Standard Code for Information Interchange.

Using ASCII, 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.

```
char letter = 'A';
int number = 75;
System.out.println("letter = " + letter);

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

System.out.print("ASCII value 75 = ");
System.out.println((char)number);
```

Run output:

```
letter = A
its ASCII value = 65

ASCII value 75 = K
```

The statement `(int)letter` is called *casting*. The data type of the variable is converted to the type in the parentheses temporarily. If you try to convert something to an incompatible type you will get an error.

7. 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
//However, another programmer might think this is an
//accident.
```

vs.

```
position = (int)letter;
//Here it is immediately clear what you mean to do.
//This is called self-documenting because the code shows
//what the meaning is without the need for comments.
```

8. Programmers use the single quote to represent char data and double quotes to denote String types. To use either of those characters in a String literal, such as in a `System.out.println` statement, you must use an escape sequence. 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'`

```
System.out.println(“This is a\ntest and only\’ a test.”);
```

Run output:

```
This is a
```

```
test and only’ a test.
```

9. 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.

10. The last 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 reserved words **`true`** and **`false`** always refer to 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 in a class outside of any methods or inside of a method. Variables can also be declared and initialized in one line. The following example code illustrates these aspects of variable declaration and initialization.

    ```
    // first is declared and initialized
    // second is just declared
    int first = 5, second;
    double x;
    char ch;
    boolean done;

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

    int sum = first + second;
    ```

    Code Sample 3-1

    Note that multiple variables can be declared on one line. Initialization is done using the assignment operator (=). Initialization can occur at declaration time or later in the program. The variable `sum` was declared and used in the same line.

3. Where variables are declared is a matter of programming style and need since this determines how and where they can be used.

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

1. The `System.out` object is automatically created in every 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:

```
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!
```

Code Sample 3-2

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).

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 defines 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 later on.

4. The lines

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

of Code Sample 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 String literal, `"number = "`, versus a variable, `number`, to the `System.out` object. A **`boolean`** variable will be printed as **`true`** or **`false`**.

6. The result (or output) of formulas using Strings may also be printed. Note how the placement of the quotes affects the output.

```
System.out.println( 2 + 2);
//Output: 4
System.out.println(“2 + 2”);
//Output: 2 + 2
System.out.println(“2” + “2”);
//Output: 22
```

E. Assignment Statements

1. An assignment statement has the following basic syntax:

`variable = expression``;`

The assignment operator (=) assigns the value of the expression on the right to the variable.

```
    a = 5;
```

This is not the same as saying, “a equals five,” but is more akin to, “a receives the value five.”

The *`expression`* can be a literal constant value such as 2, 12.25, 't' or it can also be a numeric expression involving operands (variables or constants) and operators.

a = 5 + 2;
b = 6 * a;

The assignment operator returns the value of the expression. Returning values in this way allows for chaining of assignment operators. Chaining is when you have more than one assignment in a statement as shown below.

```
    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. Variables contain either primitive data or object references. Notice that there is a difference between the two statements:

```
primitiveValue = 18234;
```

and

```
myPencil = new DrawingTool();
```

A variable will *never* actually contain an object, only a reference to an object. In the first statement, `primitiveValue` is a primitive type, so the assignment statement puts the data directly into it. In the second statement, `myPencil` is an object reference variable (the only other possibility) so a reference to the object is put into that variable. The object reference tells the program where to find an object.

Variable Type	**Information It Contains**	**When On the Left of "="**
primitive	Contains actual data	Previous data is replaced with new data
object	Contains a reference, i.e. information on how to find the object referred to by the variable	Old reference is replaced with a new reference

3. The two types of variables are distinguished by how they are declared. Unless it was declared to be of a primitive type, it is an object reference variable. A variable will not change its declared type.

4. Be aware. Because these assignments work differently with primitive data types and with objects, you may experience unexpected behavior. Consider the following:

```
DrawingTool pencil = new DrawingTool(300,300);
pencil.setColor(Color.red);
DrawingTool pen = pencil;
pen.setColor(Color.blue);
pencil.forward(100);
```

In this example, pencil will draw a blue line instead of red. This happens because we have only created one object here; we only have one new statement. When we say DrawingTool pen = pencil, all we are doing is assigning the variable name of ‘pen’ to the same object that pencil is already assigned to. This means that both pen and pencil refer to the same

DrawingTool object, and what you tell pen to do is also happening to the pencil, because they are the same object. This may be confusing at first, but once you begin to utilize OOP, it will begin to make more sense.

F. Math Operators

1. Java provides 5 math operators as listed below:

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

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

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

```
2 + 3 -> 5                    9 - 3 -> 6
4 * 8 -> 32                   11/2 -> 5
```

Notice that 11/2 is 5, and not 5.5. This is because ints work *only* with whole numbers. The remaining half is lost in integer division.

4. If either of the operands is a double type, the result is a double type. Examples:

```
2 + 3.000 -> 5.000
25 / 6.75 -> 3.7037
11.0 / 2.0 -> 5.5
```

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

5. 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
```

6. 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 double returns a double value. For example:

```
-(67) -> -67               -(-2.345) -> 2.345
```

7. To obtain the answer of 5.5 to a question like `11/2`, we must cast one of the operands.

`(double)11/2` results in `5.5`

The casting operators are unary operators with the following syntax:

```
(type) operand
```

The same effect can also result from simply

11.0/2

G. Precedence of Math Operators

1. Precedence rules govern the order in which an expression is solved. For example:

   `2 + 3 * 6 -> 20` the `*` operator has priority over `+`.

2. Associativity refers to the order in which operators are applied if they have the same precedence level. The two possibilities are from left-to-right or right-to-left.

3. A unary operator is used on only one number. An example of a unary operator is the negative sign in the expression –*a*, meaning the negative of *a*.

4. The following table summarizes precedence and associativity of math operators:

Level of Precedence	**Operator**	**Associativity**
Highest	unary `-`	right to left
	`* / %`	left to right
Lowest	`+ -`	left to right

5. An example follows:

```
9  +  16  /  3  *  7  %  8  -  5     (solve / first)
    9  +  5  *  7  %  8  -  5        (solve * second)
       9  +  35  %  8  -  5          (solve % next)
          9  +  3  -  5              (solve left-to-right)
                7
```

6. Parentheses take priority over all the math operators.

   `(5+6)/(9-7) -> 11/2 -> 5`
   (integer division, which drops remainders, is used here)

H. Assignment Operators

1. The statement `number = number + 5;` is an example of an accumulation statement. The old value of `number` is incremented by `5` and the new value is stored in `number`.

2. The above statement can be replaced as follows:

   ```
   number += 5;
   ```

3. Java provides the following assignment operators:

   ```
   +=     -=     *=     /=     %=
   ```

   These statements are preferable to saying number = number + 5 because they are more convenient and easier to read. You can immediately tell at a glance exactly what is being done.

4. The following examples are equivalent statements:

   ```
   rate *= 1.05;            rate = rate * 1.05;
   sum += 25;               sum = sum + 25;
   number %= 5;             number = number % 5;
   ```

5. The precedence of the assignment operators is the lowest of all operators.

I. <u>Increment and Decrement Operators</u>

1. Incrementing or decrementing by one is a common task in programs. This task can be accomplished by the statements:

   ```
   n = n + 1;     or     n += 1;
   ```

2. Java also provides a unary operator called an increment operator, ++.

3. The statement `n = n + 1` can be rewritten as `++n`. The following statements are equivalent:

   ```
   n = n + 1;                ++n;
   sum = sum + 1;            ++sum;
   ```

4. Java also provides for a decrement operator, `--`, which decrements a value by one. The following are equivalent statements:

   ```
   n = n - 1;                --n;
   sum = sum - 1;            --sum;
   ```

5. The increment and decrement operators can be written as either a prefix or postfix unary operator. If the ++ is placed before the variable it is called a pre-increment operator (`++number`), but it can follow after the variable (`number++`), which is called a post-increment operator. The following three statements have the same effect:

   ```
   ++number;  number++;  number = number + 1;
   ```

6. Before looking at the difference between prefix and postfix unary operators, it is important to remember Java operators solve problems and often return values. Just as the assignment operator (=) returns a value, the `++` and `--` operators return values. Consider the following code fragments:

```
int  a=1, b;
b = ++a;
```

After execution of the above code
`a = 2` and `b = 2`

```
int  a=1, b;
b = a++;
```

After execution of the above code
`a = 2` and `b = 1`

7. The statement `b = ++a` uses the pre-increment operator. It increments the value of `a` and returns the <u>new</u> value of `a`.

8. The statement `b = a++` uses the post-increment operator. It returns the value of `a` and then increments a by 1.

9. The precedence and associativity of the unary increment and decrement operators is the same as the unary `-` operator.

SUMMARY/ REVIEW:

This lesson has covered a great amount of detail regarding the Java language. At first, it is necessary to memorize the syntax of data types and their operations, but with time and practice, fluency will come. As classes are designed and code is written to solve problems, a primitive data type will often be chosen to store basic information.

HANDOUTS AND ASSIGNMENTS:

Handout A3.1, *Reserved Words in Java*
Handout A3.2, *ASCII Characters*
Lab Assignment, A3.1, *Easter*
Lab Assignment, A3.2, *Coins*
Worksheet A3.1, *DataTypes*
Worksheet A3.2, *Precedence and Assignment Operators*
Worksheet A3.3, *Math Operators*
Worksheet A3.4, *Vocab A1-A3 Review*