STUDENT OUTLINE

Lesson 5 – Math Functions, Constants

INTRODUCTION: The Java language provides some unusual operators called assignment operators. These shortcuts allow you to write shorter source code and they allow the compiler to create more efficient runtime code. Because of the abundance of operators in this language, the operator precedence table in Java is a long one. We begin our study of operator precedence with the familiar math operators.

In keeping with the concept of recycling code, we will examine some of the methods available in the `Math` class. A fundamental set of math methods is pre-defined in Java and ready for use.

The key topics for this lesson are:

A. Standard Math Functions
B. Precedence of Math Operators
C. Assignment Operators
D. Increment Operators
E. Named Constants

VOCABULARY:

PRECEDENCE
ASSIGNMENT OPERATORS
INCREMENT OPERATOR
DECREMENT OPERATOR
NAMED CONSTANTS

DISCUSSION:

A. Standard Math Functions

1. The `Math` class in the `java.lang` package contains class methods for commonly used mathematical function. Java loads the `java.lang` package automatically, so no special actions are required to access these.

THESE ARE KNOWN AS static METHODS & CONSTANTS. SO, NO INSTANTIATION IS REQUIRED!

2. For example, to express the mathematical formula

$$\frac{1}{2}\sin\left(x-\frac{\pi}{y^3}\right)$$

Using the `Math` class constant and methods would give

```
(1.0/2.0) * Math.sin(x - Math.PI / Math.pow(y, 3))
```

BASE EXP.

Notice how the class methods and class constants are referred to in the expression. The syntax is

```
class_name.method_name ( parameters );
```

Example: `Math.pow(y,3)` produces y^3

or

```
class_name.class_constant;
```

Example: `Math.PI` produces the value of the constant π

3. A complete list of Java math methods like those shown above is provided at the Java site of Sun Microsystems, Inc.:

   https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

B. 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 + 5 * 7 % 8 - 5      (solve * first)
  9 + 35 % 8 - 5       (solve % next)
    9 + 3 - 5          (solve left-to-right)
        7
```

WE CAN OVER RIDE PEMDAS

6. Parentheses take priority over all the math operators.

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

C. Assignment Operators A.K.A. ACCUMULATORS

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`. THE OLD VALUE IS LOST

SAME!

2. The above statement can be replaced as follows:

```
number += 5;
```

3. Java provides for the following assignment operators:

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

DEPENDS SEE NEXT PAGE

UNARY (U) PEMDASA (U) BINARY

4. The following examples are equivalent statements:

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

INCLUDES =

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

D. Increment Operators

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

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

++

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

PRE-INCREMENT

DO THIS BEFORE OTHER OPERATIONS

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

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

INCREASE BY 1

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;
```

DECREASE BY 1

POSITION MAKES A HUGE DIFFERENCE

POST-INCREMENT

- FINISH WHAT YOU'RE DOING, AND WHEN YOU'RE DONE RAISE THE VALUE BY 1

a 2

b 2

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

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

6. Before we look 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:

a 2

b 1

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

After execution of the code `b = ++a;`

`a = 2` and `b = 2`

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

After execution of the code `b = a++;`

`a = 2` and `b = 1`

7. The statement `b = ++a` uses the preincrement operator. It increments the value of `a` and returns the new value of `a`.

8. The statement `b = a++` uses the postincrement operator. It increments the value of `a` and returns the old value of `a`.

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

E. Named Constants

1. Java allows for declaration of named constants in a program. The general syntax is:

```
final data_type CONSTANT_NAME = expression;
```

   a. `final` is a Java reserved word
   b. *`data_type`* is any previously defined type
   c. *`CONSTANT_NAME`* is a valid Java identifier
   d. *`expression`* is any valid expression of the appropriate type

2. Examples:

```
final double PI = 3.14159;
final double GAS_CONSTANT = 8.206E-2;
final int DAY_IN_WEEK = 7;
final char BIG_J = 'J';
```

3. Programmers declare constant identifiers using all uppercase letters. This distinguishes such identifiers from variables. This curriculum guide will use uppercase identifiers for constant values.

4. The major benefit of using constant values occurs when such constants need to be modified. Suppose you have a program that uses a tax rate value in 20 different lines of code. If you had used the literal value of `0.0525`, you would need to change the value in 20 different spots. However, if you had declared a constant identifier and used the identifier in those 20 spots, then one change at the top of the program would update the value throughout the program.

```
final double TAXRATE = 0.0825;
```

   All references to `TAXRATE` are now changed.

SUMMARY/ REVIEW: Your lab work will incorporate the material from the previous lessons. The worksheet will provide practice on operator precedence and assignment operators.

ASSIGNMENT: Lab Exercise, L.A.5.2, *Quadratic*