Java Basics Unit

Lesson 3: Programs, Statements, and Variables

The Learning House

427 S 4th Street #300

Louisville KY 40202

Lesson 3: Programs, Statements, and Variables

Overview

Have you ever tried to learn another spoken language? Have you thought about what is involved in this process? Natural languages are composed of several things: language constructs (nouns, verbs, adjectives, etc.), word order, and tense, to name a few. These features are more or less common to all natural languages, so, to learn a new language, you must understand how these features work in that language.

Learning a programming language is similar — you must understand the language constructs and how to say something that makes sense in the language. When learning your first programming language, you have a couple of additional challenges. Not only do you have to learn the programming language, you also have to learn the meta-language of programming. In other words, you must learn how to talk about programs and programming languages. You will also have to learn the language of the industry and how we talk about software, the process of software development, and computer technology in general. This is a lot to do at once and can be quite difficult at first; however, it will get easier as you gain experience.

In this lesson, we begin our journey with the smallest pieces of the Java language: programs, statements, and variables.

Approach

We will take a bottom-up approach in this course. We will start with the smallest pieces of the language and build our vocabulary — at first it will be a lot like “See Spot run,” but we will quickly move on from there. As our vocabulary grows, we will look at more complicated statements and will add larger and larger constructs. Learning to write software is like learning how to write fiction or learning how to play an instrument. In fiction writing, you start with the basic vocabulary and build it (after all, you can’t write the next great American novel if you only know 6 words). Then, you move on to more complicated sentences, paragraphs, chapters, and finally the novel itself. Similarly, when learning to play an instrument, you start with the individual notes, then scales, then more complicated rhythms, melodies, chords, harmonies, and finally entire songs.

Learning to write software is a process: you must start at the beginning, build a strong foundation, and — most importantly — practice. A lot.

Concepts

To begin, we will take a look at some concepts we’ll need in order to write good software: computers, data and information, programs and programming, objects, specifications, and, finally, syntax and semantics. We must keep these concepts in mind because computer programs are not merely a sequence of characters on the screen, typed in the right order that magically make things happen — we write programs that instruct the computer to process data and to solve real problems.

Computers

So... what exactly is a computer, anyway? We see and use them every day, but what are they? One definition is:

An electronic device that processes data according to a set of instructions contained in a program.

This is a pretty general definition that covers all the bases for our purposes. One thing to keep in mind is it takes both the hardware (computer) and the software (program) to accomplish anything useful. Most people do not program computers — most people simply use them. We will be doing both in this class.

Data vs. Information

Data and information are two words that are closely related and are often used interchangeably, but they have different meanings. Data is the raw material from which information is built. An example of data is a number like **120**. You can recognize it as a number but it doesn't have any meaning… Does it refer to your current speed? The temperature? How much money you have in your pocket (dollars or pennies)? Something else? Information is interpreted data, and it is the interpretation that gives it meaning. The computer processes data, but we give that data meaning and interpret the output as information.

Programs and Programming

What is a program? What is programming? A program is a set of instructions that, when run on a computer, solves a particular problem. For example, maybe the program will add any two given numbers together and produce the sum. Programming is the act of writing those instructions for the computer to solve the particular problem. One thing to keep in mind:

The computer cannot and will not help you solve any problem; it simply does what you tell it to do. This means that **you must be able to solve a problem on paper before you can tell the computer how to solve it!**

For the most part, programming is not about math — it is about solving problems, organization, and structure.

Models and Metaphors

We use models and metaphors to help us program solutions to problems. If our programs are to solve real world problems, they must distill the problem down to its essence, model the important pieces, and then calculate the result. One great example of a computer metaphor is the concept of a "desktop" on your computer. It is not really a desktop, but the metaphor is useful in helping us to organize files (folders and files are also metaphors) and icons that we use.

Objects

One of the most important metaphors that we use in Java is the Object. This metaphor is not used in Java alone; it is used in all object-oriented languages. Objects represent things and the world is full of things. Things have properties (weight, volume, color, etc.) and they have actions that they can perform (walk, turn left, fly, etc.). Objects model these properties and actions. We are not going to get into objects too deeply until Unit 02 but keep the Object concept in mind as we move through Unit 01.

Specifications

Specifications are important to the process of writing software — specifications tell us what problem we are to solve. Without a specification, we have no way of knowing what kind of program to write. Specifications don't have to be elaborate but they do have to state the problem to be solved and any constraints around the problem.

Syntax vs. Semantics

One thing that can be very frustrating to beginning programmers is how literal the compiler is. If you don't type in perfectly formed Java statements, the compiler will not understand what you mean. Beginning programmers spend a lot of time worrying about syntax — after all, if the compiler won't compile your code, you can't run your program. You will learn proper Java syntax in this course (as discussed above, we'll build your vocabulary); however, the main emphasis of this course will be on semantics: the meaning and purpose of the various language constructs and (most importantly) how and when to use them to solve problems.

Language Building Blocks

Now that we are familiar with some of the concepts involved in programming, we are ready to move on to the Java language itself. In the first two lessons of this unit, we wrote the very simple Hello, World! program. We got familiar with the very basics of a Java program and now it is time to take a deeper look. We'll start with some beginning vocabulary pieces and then look at the rules for how those pieces can be used. These pieces are all related, so we will cover all of them in this section.

Comments

Comments are used to document your code. Putting comments in your code while you are developing it is a **very** important habit to get into. These comments may be used by other developers that come along later to maintain or modify your code, but often these comments will be useful to you as well. It is quite common to be asked to go back and enhance or fix code that you wrote 6 months or a year ago. It is likely that you would have worked on several projects since that original code was written, which makes it hard to remember why you did what you did — this is where code comments can save the day. One of my favorite quotes of software development is by Hal Abelson:

Programs must be written for people to read, and only incidentally for machines to execute.

Java supports three types of comments:

- single line,
- multi-line, and
- doc.

Single line comments

Single line comments begin with `//`. The compiler will ignore everything between the `//` and the end of the line.

```
System.out.println("Hello, World!"); // print to console
```

Multi Line comments

Multi Line comments begin with `/*` , can span multiple lines, and end with `*/` . The compiler ignores everything between `/*` and `*/` inclusive.

```
/*
    public static void main(String[] args)
    This is the entry point for the program.
    Prints "Hello, World!" to the console
*/
public static void main(String[] args) {
    System.out.println("Hello, World!");
}
```

Doc comments

Doc comments begin with `/**` , can span multiple lines, and end with `*/` . The compiler ignores everything between `/**` and `*/` inclusive. Doc comments are used by the javadoc utility to generate HTML documentation for your program. We'll explore javadoc in a later lesson.

```
/**
    Documentation for a method
*/
```

Identifiers

An identifier is a sequence of characters used to name a variable, method, class, or package. A Java identifier must follow these rules:

1. It cannot span multiple lines
2. It must only contain numbers, letters, underscores (_), dashes (-), and dollar signs ($)
3. It must start with a letter, underscore, dash, or dollar sign
4. It cannot contain any spaces

Java reserves the following keywords, which means you cannot use them for identifiers:

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
extends	false	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	null	package	private	protected
public	return	short	static	strictfp
super	switch	synchronized	this	throw
throws	transient	true	try	void
volatile	while			

Java also reserves the use of **true, false,** and **null**.

Data Types

A data type describes the internal representation of a piece of data and the operations that can be applied to that data. Java has two kinds of data types: **primitive** (which are part of the language definition) and **reference** (or **developer defined**). We will cover primitive data types in this lesson. Java has the following primitive data types:

Type	Contains	Default	Size	Range
boolean	true or false	false	1 bit	NA
char	Unicode character	\u0000	16 bits	\u0000 to \uFFFF
byte	Signed integer	0	8 bits	-128 to 127
short	Signed integer	0	16 bits	-32768 to 32767
int	Signed integer	0	32 bits	-2147483648 to 2147483647
long	Signed integer	0	64 bits	-9223372036854775808 to 9223372036854775807
float	IEEE 754 floating point	0.0	32 bits	±1.4E-45 to ±3.4028235E+38
double	IEEE 754 floating point	0.0	64 bits	±4.9E-324 to ±1.7976931348623157E+308

Type Conversion

On occasion, you will need to convert one data type to another. In general, the compiler will automatically convert a 'narrower' data type (i.e. one containing fewer bits) to a 'wider' data type (i.e. one containing a larger number of bits) but will not do the reverse. In the reverse case, the developer is generally required to explicitly tell the compiler how to convert the data type with a 'cast' operator, which will be covered later.

To illustrate this, imagine that you have a small carry-on piece of luggage for your trip. If you wanted to, you could easily fit the contents of the small carry-on into a larger suitcase. You wouldn't even have to think about it, everything would just fit. However, if you started with the large suitcase and wanted to switch to the smaller

carry-on, you would have to make decisions as to what items (shirt? shorts? camera?) that you would have to leave out. You would have to make some explicit decisions about what stays home and what goes with you.

Literals

A literal is a sequence of characters that represents a data item in source code. Java recognizes six data literals:

Boolean: the reserved words 'true' and 'false' are Boolean literals

1. **Character**: a character literal is represented by a single character surrounded by single quotes ('B'), a unicode escape sequence (u\000), or a regular escape sequence ('\n')
2. **Floating-point number**: examples of floating-point literals are: 3.14, 2.56E+31, 4.56D. F or f can be used to specify a float data type; D or d is used to specify a double data type.
3. **Whole number**: integer literals can be written as decimal (104) or hexadecimal (0x19F). They are of int data type unless followed by an upper or lower case L, in which case they are of long data type.
4. **Null**: the reserved word 'null' is a null literal. We will look at this in more detail in the next unit.
5. **String**: a sequence of characters surrounded by double quotes. "Hello, World!" is a string literal.

Variables

A variable is a named piece of memory where you can stash the value of something. In order to reserve your chunk of memory, you must **declare** your variable according to the following pattern:

data_type variable_name;

For example:

```
public static void main(String[] args) {

    // declare an int called counter
    int counter;

    // declare a boolean called isDone
    boolean isDone;
}
```

Once you have declared a variable, you can assign a value to it:

```
public static void main(String[] args) {

    // declare an int called counter
    int counter;
    // now assign the number 7 to counter
    counter = 7;

    // declare a boolean called isDone
    boolean isDone;
    // now assign false to isDone
```

```
        isDone = false;
    }
```

You can also declare a variable and assign a value at the same time, if you wish:

```
    public static void main(String[] args) {

        // declare an int called counter and assign the value 7
        int counter = 7;

        // declare a boolean called isDone and assign false
        boolean isDone = false;
    }
```

When assigning values to variables, you must be sure to assign the correct type. Failing to do so will result in a compilation error. For example, you cannot assign the number 34 to a boolean variable.

Expressions

An expression is a statement that can be evaluated to produce a result. The result of the expression can be used in your program (for example, we could assign the result of the expression to a variable). Expressions can be simple or complex but all expressions are made up of **operators** and **operands**. You are familiar with operators and operands from everyday math — operators are the things that do the work (such as +, -, *, /) and the operands on the things on which the work is done (i.e. the numbers).

Operators come in three flavors: unary, binary, and ternary. Unary operators require only one operand (the negation operator is an example). Unary operators can be either **prefix** (coming before the operand) or **postfix** (coming after the operand). Binary operators require two operands (basics math operations fall into this category). Binary operators are **infix** (the operator is between the operands). Ternary operators require three operands (there is only one ternary operator in Java: the conditional operator, which is an infix operator).

Here are some code examples for mathematical expressions and operators:

```
18      public static void main(String[] args) {
19          // Declare variables to use in the examples
20          int result;
21          int operand1;
22          int operand2;
23          int operand3;
24
25          //
26          // Assignment
27          //
28
29          // Initialize result with the value of 0 by using the assignment (=)
30          // operator. The assignment operator takes the value on the right
31          // and assigns it to the variable on the left
32          result = 0; // now result has the value of 0
33
34          // Initialize the operands
35          operand1 = 5;
36          operand2 = 7;
37
38          // Assignment works with variable values as well as literal values.
39          // We'll set operand3 to the same value as operand2
40          operand3 = operand2; // now both have the value 7
41
```

Figure 1: Assignment

```
42          //
43          // Addition
44          //
45
46          // Addition is a binary infix operator.  It works with literals:
47          result = 42 + 53;  // result is now 95
48
49          // It also works with variables:
50          result = operand1 + operand2;  // result now equals 12
51
52          // It works with a combination of literals and variables:
53          result = 1 + operand1;  // result now equals 6
54
55          // You can chain addition operators together:
56          result = 1 + operand1 + operand2 + operand3; // result now equals 20
57
58          // Finally, the += operator is used to add a value to a variable.
59          // result += operand1 is equivilant to result = result + operand1.
60          // NOTE: the initial value of result is used to calculate the new value
61          //       of result:
62          result = 2;  // set result to 2
63          result += 4;  // result is now equal to 6 (2 + 4)
64          result += operand1; // result is now equal to 11 (6 + 5)
65
```

Figure 2: Addition

```
66          //
67          // Subtraction
68          //
69
70          // Subtraction is a binary infix operator.  It works with literals:
71          result = 9 - 5;  // result is now 4
72
73          // It also works with variables:
74          result = operand1 - operand2;  // result now equals -2
75
76          // It works with a combination of literals and variables:
77          result = 15 - operand1;  // result now equals 10
78
79          // You can chain subtraction operators together:
80          result = 19 - operand1 - operand2 - operand3; // result now equals 0
81
82          // Finally, the -= operator is used to add a value to a variable.
83          // result -= operand1 is equivilent to result = result - operand1.
84          // NOTE: the initial value of result is used to calculate the new value
85          //       of result:
86          result = 2;  // set result to 2
87          result -= 4;  // result is now equal to -2 (2 - 4)
88          result -= operand1; // result is now equal to -7 (-2 - 5)
89
```

Figure 3: Subtraction

```
90          //
91          // Multiplication
92          //
93
94          // Multiplication is a binary infix operator.  It works with literals:
95          result = 2 * 3;  // result is now 6
96
97          // It also works with variables:
98          result = operand1 * operand2;  // result now equals 35
99
100         // It works with a combination of literals and variables:
101         result = 2 * operand1;  // result now equals 10
102
103         // You can chain multiplication operators together:
104         result = 2 * operand1 * operand2 * operand3; // result now equals 490
105
106         // Finally, the *= operator is used to add a value to a variable.
107         // result *= operand1 is equivilant to result = result * operand1.
108         // NOTE: the initial value of result is used to calculate the new value
109         //       of result:
110         result = 2;  // set result to 2
111         result *= 4;  // result is now equal to 8 (2 * 4)
112         result *= operand1; // result is now equal to 40 (8 * 5)
113
```

Figure 4: Multiplication

```
114            //
115            // Division and Modulus
116            //
117
118            // Division is a binary infix operator.  It works with literals:
119            result = 6 / 3;  // result is now 2
120
121            // It also works with variables:
122            result = operand1 / operand2;  // result now equals 0
123
124            // What?!?!?!?  When dividing integers, integer division is used -
125            // we only get the whole number part of the quotient.  In this case,
126            // 7 go into 5 0 times with a remainder of 5
127
128            // We use the modulus operator (%) to get the remainder:
129            result = operand1 % operand2;  // result now equals 5
130
131            // It works with a combination of literals and variables:
132            result = 20 / operand1;  // result now equals 4
133
134            // You can chain division operators together:
135            result = 245 / operand1 / operand2 / operand3; // result now equals 1
136
137            // Finally, the /= operator is used to add a value to a variable.
138            // result /= operand1 is equivilant to result = result / operand1.
139            // NOTE: the initial value of result is used to calculate the new value
140            //       of result:
141            result = 40;  // set result to 40
142            result /= 4;  // result is now equal to 10 (40 / 4)
143            result /= operand1; // result is now equal to 2 (10 / 5)
144        }
145
146    }
147
```

Figure 5: Division and Modulus

Note that division by 0 will result in an error because division by zero is mathematically undefined.

Console Input and Output

Before we move on to building our next program, we need to take a look at how we can get input from and send output to the Console (we saw the output part in Hello, World!). This code may not make a lot of sense to you right now — that's okay. Console input and output (I/O) is not too complicated and this code will make sense as we go through the course. For now, take it on faith.

The ability to get input from the user allows us to write more interesting and useful programs. The programs we've seen so far just do one thing and they'll always just do one thing unless we go in and actually change to

source code. If we can get input from the user, we can write programs that adapt and react based on the data that the user gives us; this allows us to do much more than we could before.

We will use a combination of System.out.println and an object called a Scanner to perform console input and output. To illustrate this, we'll create a small program called Adder.

Adder: Version 1

The first version of Adder will just add two variables and print the results to the screen. The values of the variables will be hard coded, so this version is pretty boring:

Notes:

1. On line 29, the + operator is acting as the **String concatenation operator** instead of the addition operator. In Java we use the + operator to concatenate String values. The output of this program is: Sum is: 5

```
18      public static void main(String[] args) {
19          // declare sum and initialize to 0
20          int sum = 0;
21          // declare and initialize our operands
22          int operand1 = 3;
23          int operand2 = 2;
24
25          // assign the sum of operand1 and operand2 to sum
26          sum = operand1 + operand2;
27
28          // print the sum to the console
29          System.out.println("Sum is: " + sum);
30      }
```

Figure 6: Adder Version 1

Adder: Version 2

Now we'll modify Version 1 so that we ask the user for the value for operand1 and operand2. We'll use the techniques shown in Version 2 throughout the first part of the course:

Notes:

1. Line 28: We declare and initialize a Scanner. Notice that the data type is **Scanner** and the variable name is **sc**. Also note the special initialization for sc:
   new Scanner(System.in)
   This is how we initialize non-primitive data types in Java. We'll explore this in detail in Unit 02.
2. Lines 32 and 33: The Console only understands text (string data type) — it has no concept of numbers. This means that we must read the user input into variables of type string and then convert them to numbers later.
3. Lines 36 and 42: We use System.out.println(...) to prompt the user for input. Without these lines, the user would have no idea what to type in.

4. Lines 39 and 45: sc.nextLine() waits for the user to type something into the Console and hit the Enter key. When the Enter key is hit, nextLine() reads everything that the user typed on that line and stores it in the variable on the left hand side of the = operator.
5. Lines 49 and 50: Integer.parseInt(...) converts the value given to it (in our case stringOperand1 and stringOperand2) from text to an actual number. Be careful here — if it can't convert the input to a number, it will throw an error. We'll see how to handle these errors (called exceptions) later on.

```
20      public static void main(String[] args) {
21          // declare sum and initialize to 0
22          int sum = 0;
23          int operand1 = 0;
24          int operand2 = 0;
25
26          // declare and initialize a Scanner object - the Scanner reads input
27          // from the console
28          Scanner sc = new Scanner(System.in);
29          // declare and initialize String (text) variables to hold the values
30          // that the user types in - the Console only know about text, it knows
31          // nothing about numbers
32          String stringOperand1 = "";
33          String stringOperand2 = "";
34
35          // ask the user to input the first operand:
36          System.out.println("Please enter the first number to be added:");
37          // now wait until the user types something in - put the value in
38          // stringOperand1
39          stringOperand1 = sc.nextLine();
40
41          // ask the user to input the second operand:
42          System.out.println("Please enter the second number to be added:");
43          // now wait until the user types something in - put the value in
44          // stringOperand2
45          stringOperand2 = sc.nextLine();
46
47          // in order to add the values input by the user we must convert the
48          // String values into int values.  We use the parseInt method for this:
49          operand1 = Integer.parseInt(stringOperand1);
50          operand2 = Integer.parseInt(stringOperand2);
51
52          // assign the sum of operand1 and operand2 to sum
53          sum = operand1 + operand2;
54
55          // print the sum to the console
56          System.out.println("Sum is: " + sum);
57      }
```

Figure 7: Adder Version 2

WindowMaster

Now, let's put everything we've learned together and build our first useful program. The purpose of the program is to calculate the total cost for a home replacement window. Here are the requirements:

1. It must prompt the user for the height of the window (in feet)
2. It must prompt the user for the width of the window (in feet)
3. It must calculate and display the area of the window
4. It must calculate and display the perimeter of the window
5. Based on the area and perimeter, it must calculate the total cost of the window
    a. The glass for the windows cost $3.50 per square foot
    b. The trim for the windows cost $2.25 per linear foot

One of the first things we must do is decide the number and type of the variables that we'll need:

- String variable for height (read from Console)
- String variable for width (read from Console)
- float variable for height (converted from String — use float because we don't want to be limited to whole feet)
- float variable for width (converted from String — use float because we don't want to be limited to whole feet)
- float variable for area of window (calculated from height and width)
- float variable for perimeter of window (calculated from height and width)
- float variable for cost (calculated from area, perimeter, and costs)

That looks like a pretty good start so let's look at the code:

Notes:

1. Lines 20 - 35: Variable declaration and initialization — similar to Adder
2. Lines 36 - 40: Get user input — similar to Adder
3. Lines 43 - 44: Convert input into number — similar to Adder except we are converting to floats instead of ints so we use Float.parseFloat(...)
4. Lines 47 - 53: Do our calculations using statements and expressions — similar to Adder
5. Lines 55 - 59: Print results to the Console — similar to Adder

```
19      public static void main(String[] args) {
20          // declare variables for height and width
21          float height;
22          float width;
23          // declare String variables to hold the user's height and width
24          // input
25          String stringHeight;
26          String stringWidth;
27
28          // declare other variables
29          float areaOfWindow;
30          float cost;
31          float perimeterOfWindow;
32
33          // Declare and initialize our Scanner
34          Scanner sc = new Scanner(System.in);
35
36          // Get input from user
37          System.out.println("Please enter window height:");
38          stringHeight = sc.nextLine();
39          System.out.println("Please enter window width:");
40          stringWidth = sc.nextLine();
41
42          // Convert String values of height and width to floats
43          height = Float.parseFloat(stringHeight);
44          width = Float.parseFloat(stringWidth);
45
46          // calculate area of window
47          areaOfWindow = height * width;
48
49          // calculate the perimiter of the window
50          perimeterOfWindow = 2 * (height + width);
51
52          // calculate total cost - use hard coded for material cost
53          cost = ((3.50f * areaOfWindow) + (2.25f * perimeterOfWindow));
54
55          System.out.println("Window height = " + stringHeight);
56          System.out.println("Window width = " + stringWidth);
57          System.out.println("Window area = " + areaOfWindow);
            System.out.println("Window perimeter = " + perimeterOfWindow);
59          System.out.println("Total Cost = " + cost);
60      }
```

Figure 8: WindowMaster

Wrap-up

Wow — we covered a lot in this lesson. Here's what you learned:

1. You learned about some concepts surrounding programs and programming, such as:
    a. Computers
    b. Data vs. information
    c. Programs and programming
    d. Models and metaphors
    e. Objects
    f. Specifications
    g. Syntax and semantics
2. You learned about the basic building blocks of the Java language:
    a. Comments
    b. Identifiers
    c. Data types
    d. Literals
    e. Variables
    f. Expressions
3. You learned how to interact with a user via the Console.
    a. You wrote two programs that interact with the user and perform calculations based on the user input.

Next up, we'll see how to make decisions and repeat ourselves in code. We'll also take a look a Boolean expressions and learn how to map out our program logic with flowcharts.