Programming Club

Primitive Values in Java

Introduction:

In order for a Java program to be successful, values must be created, calculated, used, and presented. When a value is created, it is stored in memory. But there are two things that must be identified: the variable name and the type. The variable name allows you to reference the value throughout the program. The type is what kind of value it is. The type only has to be shown when the value is created. The simplest kinds of types are called primitive types.

int num = 4;

Value

Variable name

Type

Types of Numerical Primitives:

So, the line of code above is assigning an int value to the variable num. But what does int do? It is a primitive type that represents integers. Some other primitive types that also represent integers are byte, short, and long. Except, byte and short have a more limited range of integers they can represent, byte being the most limited, being only able to represent integers from -128 to 127. long is used when we have very large numbers; its maximum integer value is in the quintillions, as opposed to int, whose max is about 2 billion. But int is most frequently used when we want to represent integers. But, in this case, we could replace int with any of the other types because 4 is within the range of integers for all of them.

But what about decimals? There are two types for representing decimals (or floating point values): double and float. They are not too different; double just has a broader variety of digits that can be shown in the decimal. Therefore, it is used more often. When we assign a decimal value to a variable, the value must have a decimal point to be valid. Here is an example of a decimal value being assigned:

double decimal = 7.681;

Changing Types:

Suppose you have an integer value, but you want to change it to a decimal value so it can then be expressed in decimal form. If you put the type in parentheses next to the variable and set it equal to your new variable, you can do this.

Ex: double alternative = (double) num;

Now, alternative is a decimal version of num. When printed, it will show 4.0. But, not all type conversions work. You might get an error sometimes.

Changing the Value:

If we wanted to change the value of num, we could simply set it equal to a different integer value. But, what if we want to apply a numerical expression on num? This will be useful when the value of a variable is unknown because it is user inputted. Here are the basic mathematical expressions: addition (+), subtraction (-), multiplication (\*), division (/), and remainder (%). The remainder operator does a division but then returns the remainder. For example: 27 % 4 = 3 because 27 / 4 will yield a remainder 3. When we change the value of a variable, we can either create a new variable or keep the same variable, as long it remains the same type.

Ex: int newNum = num / 4;

num = num / 4;

Both num are newNum are going to be equal to 1, except for that now num’s original value (4) is lost in memory. But keeping the same variable can sometimes make things simpler. Some short cuts (highlighted in red) for changing the value but keeping the same variable go as follows:

num = num + 4; OR num += 4;

num = num – 4; OR num -= 4;

num = num \* 4; OR num \*= 4;

num = num / 4; OR num /= 4;

num = num % 4; OR num %= 4;

num = num + 1; OR num++;

num = num – 1; OR num--;

The last two are special short cuts where you can just do ++ when incrementing a value by 1 and -- when decrementing a value by 1.

Important notes to remember:

* There is an order of operations; it is just like in algebra (PEMDAS). You can definitely use parentheses when writing code to tell the program to run the code within the parentheses first.
* When working with ints, the program will use integer math meaning that when we divide two integers and have the answer be of type int, the answer will be the greatest integer less than or equal to the actual answer: For example: If we say: int quotient = 7 / 5, quotient will be equal to 1 and not 1.4. To get the exact answer, we need to do two things: first make the quotient variable of type double; then change the expression on the other side of the equals sign to type double as well. If we change the line to this: double quotient = (double)(7 / 5), then quotient will be equal to 1.4.

Other types of Primitives

There are some non-numerical primitives: char and boolean. A variable of type char will represent a single character that can be anything from a letter to a number to a symbol (/, #, !). When we create a new variable of type char, we put the character within British quotation marks.

Ex: char character = ‘N’;

Now we have a char value stored in memory, the value being ‘N’. Variables of type boolean can represent either true or false. This is useful for conditionals (which you will learn more about another day) that evaluate conditions and make decisions based off of whether the condition is true or false. The condition is a value of type boolean. This enables the program to make decisions based on given information.

Printing Out Values

After all the changes we make, we want to see what the value actually is. To do this, we instruct the program to print out the value. To do this we can write either system.out.print() or system.out.println() with the variable name inside the parentheses. The only difference is that system.out.print() prints the information and stays on the same line but system.out.println() prints and then returns to a new line.

Does this all make sense? Look at the following program, keeping in mind what you’ve learned, and try to understand it.

