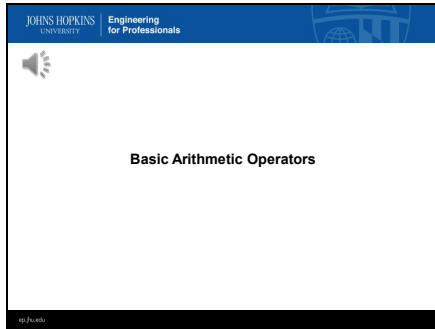


1



In this lecture you will learn about Java's basic arithmetic operators.

2

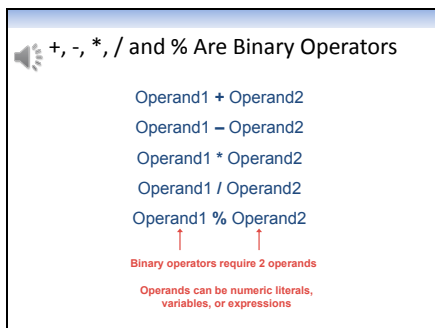
Operator Type	Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	%

Java has a total of nine (9) arithmetic operators. You will learn about all of them in this course, but in this lecture you will learn how to use 5 basic arithmetic operators to perform calculations involving addition, subtraction, multiplication, division and remainders.

The addition operator is a plus sign, the subtraction operator is a hyphen, the multiplication operator is an asterisk, the division operator is a slash, and the remainder operator is % sign.

These operators can be used with the primitive numeric types...byte, short, int, long, float and double to perform basic arithmetic calculations.

3



The basic arithmetic operators are BINARY operators. A binary operator is an operator that requires 2 operands.

Operands are things that the operation, for example, addition, are applied to. In your Java programs, operands can be numeric literals, variables, or expressions.

4

Basic Arithmetic Computations

```

public class ArithmeticOperatorDemo1
{
    public static void main( String [] args )
    {
        int x = 2;
        int y = 5;
        float z = 10.5F;
        float q;

        x = x + y; // Result is 7
        y = 3 * x - 1; // Result is 20
        q = z / 2F; // Result is 5.25
        ...
    }
}

```

Here's a simple program that demonstrates some basic arithmetic operators.

This program also illustrates some recommended style guidelines.

Notice that whenever an '=' sign is used it is surrounded by blank spaces. This is not absolutely necessary, and Java doesn't care if the blank spaces are there or not, but it is a recommended practice because it makes programs easier to read. Similarly, I've used blank spaces around the arithmetic operators for the same reason.

I encourage you to follow these practices when you write your own programs.

5

Java Statements and Expressions

```

public class ArithmeticOperatorDemo1
{
    public static void main( String [] args )
    {
        int x; // This is a declaration statement
        int y = 10; // This is a declaration statement
        x = 3; // This is an assignment statement

        x = x + y; // x + y is an expression
        y = 3 * x - 1; // 3 * x - 1 is an expression too
        ...
    }
}

```

1. Take the current value of x. (5)
2. Add the value of y to it. (15)
3. Store 15 in the memory associated with x.

This is a good time to say a few things about Java programming statements and expressions.

A **STATEMENT** is a programming instruction that causes something to happen in a program. A programming statement in Java is terminated with a semi-colon (;). In this program there are 5 programming statements.

I didn't mention this before, but Java statements must be terminated with semi-colons, and if you go back and look at all of the programs we've written so far you will see that all the statements are terminated properly. If you forget to terminate a statement the compiler will not compile your program. In this program there are 5 programming statements.


An **EXPRESSION** is a statement that results in a single value being produced...and the key phrase here is **SINGLE VALUE**. Expressions are needed to perform calculations, and can be a series of variables, operators and method calls.

As a programmer, it is important to understand how Java evaluates expressions and performs computations.

Consider the statement $x = x + y$. The equal sign is actually an operator called the assignment operator. This statement is an assignment statement that tells Java to take the result of the expression on the right-hand side of the equal sign and store it in the memory location associated with the variable on the left-hand side. In other words, it is telling Java to replace the current value of x with whatever value the right-hand side expression computes to.

In performing this instruction, Java will evaluate the expression on the right-hand side of the statement first. It will take the current value stored in the variable x ...which is 5. Then it will add to it the value stored in the variable y ...which is currently 10...to get the sum 15. Then, it will store 15 in the memory location associated with x .

6

 **Remainder is Lost in Integer Division**

```
public class IntegerDivisionDemo
{
    public static void main( String [] args )
    {
        int x = 10;
        int y = 3;

        int z = x / y;


        System.out.println( "\n" + "z = " + z );
    }
}
```

When two integers are divided in Java, any remainder portion is lost.

For example, this program divides an integer variable x , that has a value of 10, by an integer variable y , that has a value of 3. It's easy for us to do this calculation in our heads...3 goes into 10 3 times with a remainder of 1.

But, when this program runs, it will produce a result of 3...without any remainder...because integer types must be whole numbers.

7

 Try Compiling & Running This Program

```

public class IntegerDivisionDemo
{
    public static void main( String [] args )
    {
        int x = 10;
        int y = 3;

        int z = x / y;

        System.out.println( "\n" + "z = " + z );
    }
}


```

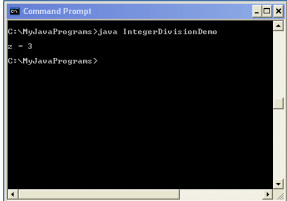
I'd like you to pause the lecture at this point.

Then, I'd like you to compile and run the IntegerDivisionDemo program and verify that it's output is 3.

When you've finished, continue the lecture.


8

 IntegerDivisionDemo Program Output



If your program did not produce this answer, then please pause the lecture at this point, double check your program to make sure your source code is exactly the same as the IntegerDivisionDemo source code in this lecture, and continue the lecture when you get the correct answer.

9

 Remainder is Retained in Float Division

```

public class FloatDivisionDemo
{
    public static void main( String [] args )
    {
        float x = 10F;
        float y = 3F;

        float z = x / y;

        System.out.println( "\n" + "z = " + z );
    }
}

```

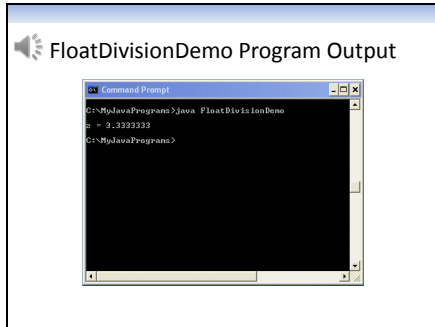
We just learned that when two integers are divided in Java, any remainder portion is lost.

If we need a program to perform more accurate division operations, then we need to use floating point types when division is performed.

This program is the same as the last one, but I've declared the variables to be of type float rather than type int.

Let's see what happens now.

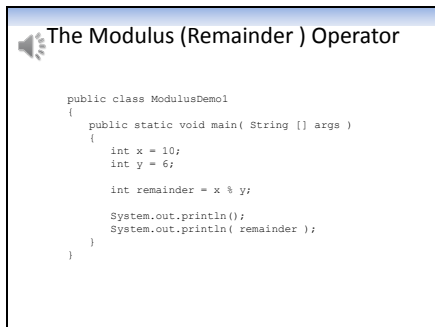
10



Notice that the remainder of the division is retained...to 7 decimal places.

That's a lot more accurate than using integers.

11



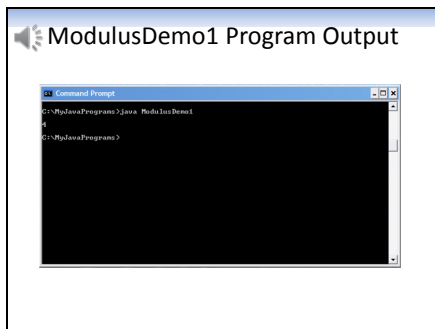
Sometimes it is useful to know the remainder of a division.

The Java modulus operator, which is a % sign, gives the remainder associated with a division.

In this program, x is 10 and y is 6. 6 goes into 10 once with a remainder of 4, so the variable named remainder should be 4.

Let's check the program's output.

12



Here's the output of the ModulusDemo1 program.

And sure enough, the remainder is 4.

I've demonstrated the modulus operator with integer types, but it works with any numeric type.