



NYU

Courant Institute of Mathematical Sciences  
Department of Computer Science  
CS101 Introduction to Computer Science

Anasse Bari, Ph.D.

## Chapter#3: Expressions, Variables, Input output using Scanner, Introduction to Strings in Java



# Objectives

- ❖ Introducing primitive data types
- ❖ Defining and learning about constants, variables, declarations, named constants.
- ❖ Defining and using operators and operands
- ❖ Defining assignment statements
- ❖ Using boolean expressions
- ❖ Using the scanner class to obtain input from the console
- ❖ Introducing strings in Java
- ❖ Introducing two programming Tips
  - ❖ Limiting a double to two or n decimal numbers after the decimal point
  - ❖ Reading a string with spaces

# Understanding the Concept of “Variables”



Think of a Java program that can add two whole numbers (Integers) provided by the user ..  
Are the two whole numbers (Integers) always the same ? Is the result always the same?

# Our Second In Class Java Program:

AddTwoNumbers.java

(see attached file)

We will write a Java program that adds two *Integer* numbers provided by the user and display the results to the screen.

Goal: Write a Java Program that adds two *Integer* numbers provided by the user and display the results to the screen. **Where do we start?**

## Step#1: Writing the algorithm

Algorithm:

Start

1. Read the first number from the console
2. Read the second number from the console
3. Store both numbers
4. Computer the addition of first number and the second number
5. Store the result
6. Display the result

End

Goal: Write a Java Program that adds two *Integer* numbers provided by the user and display the results to the screen. **Where do we start?**

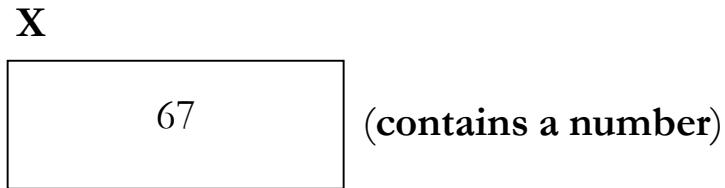
## Step#2: Writing the program

See attached  
**AddTwoNumbers.Java**  
program

**IMPORTANT**

# Constants and Variables

- The simplest terms that appear in a java expressions are constants and variables. The value of a constant does not change during the course of a program.
- **A variable is a placeholder for a value that can be updated as the program runs.**
- A variable in Java is most easily envisioned as a box capable of storing a value. *(a variable is a portion of memory to store a determined value)*



- Each variable has the following attributes:
  - **A name**, which enables you to differentiate one variable from another.
  - **A type** (*next slide*), which specifies what type of value the variable can contain.
  - **A value**, which represents the current contents of the variable.
- **The name and type of a variable are fixed. The value changes whenever you assign (put) a new value to the variable.**

# Primitive Data Types

- Although complex data values are represented using objects, Java defines a set of *primitive* types to represent simple data.
- Example of Primitive Data types:

**int**            This type is used to represent integers, which are whole numbers such as 17 or -53.

**double**        This type is used to represent numbers that include a decimal fraction, such as 3.14159265. In Java, such values are called floating-point numbers; the name double comes from the fact that the representation uses twice the minimum precision.

**boolean**       This type represents a logical value (true or false).

**char**            This type represents a single character.



# Primitive Data Types (cont'd)

## **int:**

- int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648. $(-2^{31})$  – [ the sign ^ means power]
- Maximum value is 2,147,483,647(inclusive). $(2^{31} - 1)$
- **Int is generally used as the default data type for integral values unless there is a concern about memory.**
- The default value is 0.
- Example: int a = 100000, int b = -200000

## **long:**

- Long data type is a 64-bit signed two's complement integer.
- Minimum value is -9,223,372,036,854,775,808. $(-2^{63})$
- Maximum value is 9,223,372,036,854,775,807 (inclusive).  $(2^{63} - 1)$
- **This type is used when a wider range than int is needed.**
- Default value is 0L.
- Example: long a = 100000L, long b = -200000L

# Primitive Data Types (cont'd)

## Byte:

- Byte data type is an 8-bit signed two's complement integer.
- Minimum value is -128 ( $-2^7$ )
- Maximum value is 127 (inclusive) ( $2^7 - 1$ )
- Default value is 0
- **Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.** Example: byte a = 100 , byte b = -50

## Short:

- Short data type is a 16-bit signed two's complement integer.
- Minimum value is -32,768 ( $-2^{15}$ )
- Maximum value is 32,767 (inclusive) ( $2^{15} - 1$ )
- **Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an int**
- Default value is 0.
- Example: short s = 10000, short r = -20000

# Primitive Data Types (cont'd)

## float:

- Float data type is a single-precision 32-bit IEEE 754 floating point.
- **Float is mainly used to save memory in large arrays of floating point numbers.**
- Default value is 0.0f.
- Float data type is never used for precise values such as currency. **(Use BigDecimal for precision)**

```
BigDecimal k = BigDecimal.valueOf(doublevalue);
```

- Example: float f1 = 234.5f

## double:

- double data type is a double-precision 64-bit IEEE 754 floating point.
- **This data type is generally used as the default data type for decimal values, generally the default choice.**
- Double data type should never be used for precise values such as currency. **((Use BigDecimal for precision)**

```
BigDecimal k = BigDecimal.valueOf(doublevalue);
```

- Default value is 0.0d.
- Example: double d1 = 123.4

# Primitive Data Types (cont'd)

## **boolean:**

- boolean data type represents one bit of information
- There are only two possible values: true and false.
- This data type is used for simple flags that track true/false conditions.
- Default value is false.
- Example: `boolean one = true`

## **char:**

- char data type is a single 16-bit Unicode character.
- Minimum value is `'\u0000'` (or 0).
- Maximum value is `'\uffff'` (or 65,535 inclusive).
- Char data type is used to store any character.
- Example: `char letterA = 'A'`

# Java Identifiers

- Names for variables (and other things) are called identifiers.
- Identifiers in Java conform to the following rules:
  - A variable name must begin with a letter or the underscore character.
  - The remaining characters must be letters, digits, or underscores.
  - The name must not be one of Java's reserved words:

**IMPORTANT**

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

- Identifiers should make their purpose obvious to the reader.
- Identifiers should adhere to standard conventions. Variable names, for example, should begin with a lowercase letter.

Another important type is *String*  
We will have a whole chapter on *Strings*.  
*For now you need to understand how to declare and use a String with the Scanner*

**IMPORTANT**

```
import java.util.Scanner; // a predefined Java library for keyboard input

public class Welcome {

    public static void main(String[] args) {

        System.out.print("Enter You User Name:"); // prompt for input

        Scanner input = new Scanner(System.in);
        // the next line declares a String variable named "nextName" and
        //makes the input scanner reads the string. (input.next())
        String nextName = input.next();

        System.out.println("Welcome to Java Programming, " + nextName + "!");

        input.close(); // closing the input Scanner

    } // end main
}
```

**IMPORTANT**

---

```
Enter You User Name: NYUuser
Welcome to Java Programming, NYUuser!
```

A Variable in Java has to be declared (Variable declaration)  
& initialized (Variable initialization)

**IMPORTANT**



# Variable Declarations

- In Java, you must declare a variable before you can use it. The declaration establishes the name and type of the variable and, in most cases, specifies the initial value as well.
- The most common form of a variable declaration is

```
type name = value; /* int z = 10; */
```

where *type* is the name of a **Java primitive type or class**, *name* is an identifier **that indicates the name of the variable**, and *value* is an **expression specifying the initial value**.

# Operators and Operands

- As in most languages, Java programs specify computation in the form of arithmetic expressions that closely resemble expressions in mathematics.
- The most common operators in Java are the ones that specify arithmetic computation:

+	Addition	*	Multiplication
-	Subtraction	/	Division
		%	Remainder
- Operators in Java usually appear between two subexpressions, which are called its **operands**. **Operators** that take two operands are called binary operators.
- The - operator can also appear as a **unary operator**, as in the expression -x, which denotes the negative of x.

# Division and Type Casts

**IMPORTANT**

- Whenever you apply a binary operator to numeric values in Java, the result will be of type `int` if both operands are of type `int`, but will be a `double` if either operand is a `double`.
- This rule has important consequences in the case of division. For example, the expression

`14 / 5`

seems as if it should have the value 2.8, but because both operands are of type `int`, Java computes an integer result by throwing away the fractional part. The result is therefore 2.

- If you want to obtain the mathematically correct result, you need to convert at least one operand to a `double`, as in

`(double) 14 / 5`

**IMPORTANT**

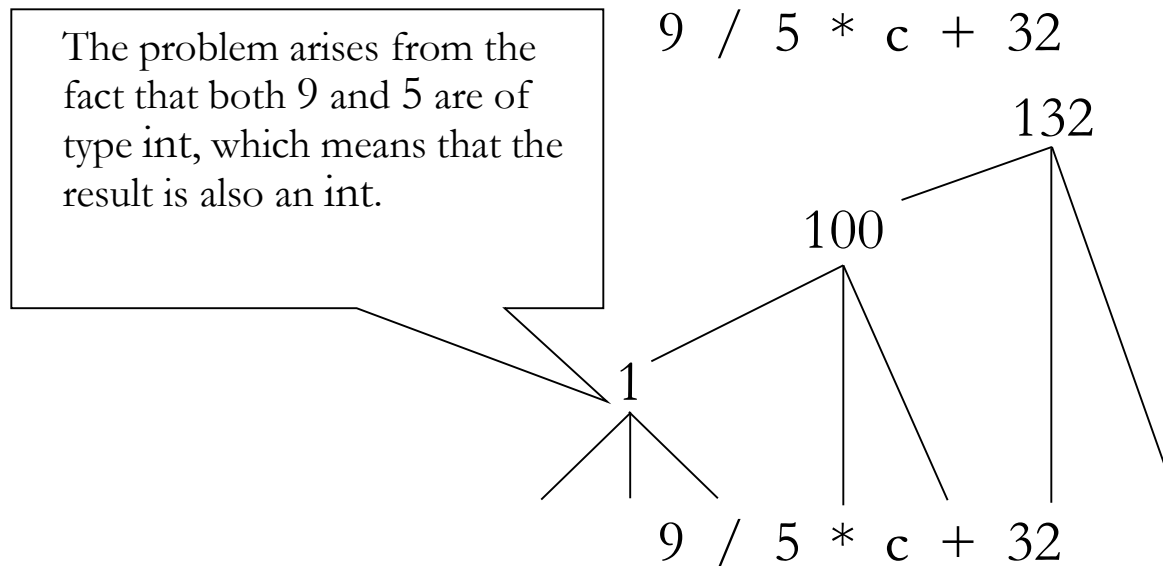
The conversion is accomplished by means of a type cast, which consists of a type name in parentheses.

# The Pitfalls of Integer Division

Consider the following Java statements, which are intended to convert 100° Celsius temperature to its Fahrenheit equivalent:

```
double c = 100;  
double f = 9 / 5 * c + 32;
```

The computation consists of evaluating the following expression:

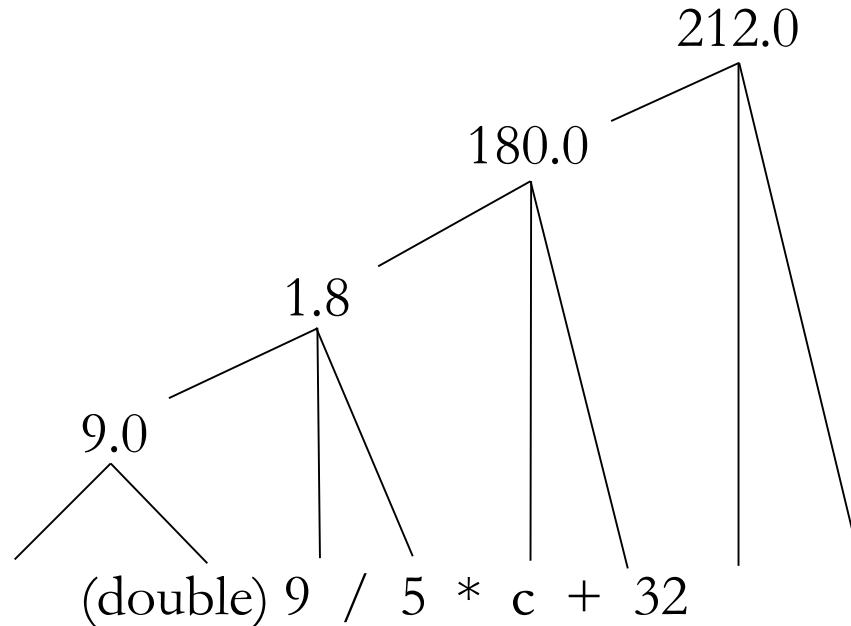


# The Pitfalls of Integer Division

You can fix this problem by converting the fraction to a double, either by inserting decimal points or by using a type cast:

```
double c = 100;  
double f = (double) 9 / 5 * c + 32;
```

The computation now looks like this:



# The Remainder Operator

- The only arithmetic operator that has no direct mathematical counterpart is %, which applies only to integer operands and computes the remainder when the first is divided by the second:

14 % 5    *returns*    4

14 % 7    *returns*    0

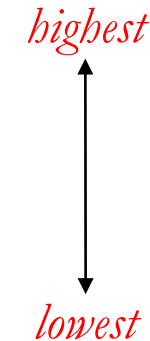
7 % 14    *returns*    7

- The result of the % operator make intuitive sense only if both operands are positive. The examples in this book do not depend on knowing how % works with negative numbers.
- The remainder operator turns out to be useful in a surprising number of programming applications and is well worth a bit of study.

# Precedence

- If an expression contains more than one operator, Java uses **precedence** rules to determine the order of evaluation. The arithmetic operators have the following relative precedence:

<i>unary -</i> ( <i>type cast</i> )
*    /    %
+    -

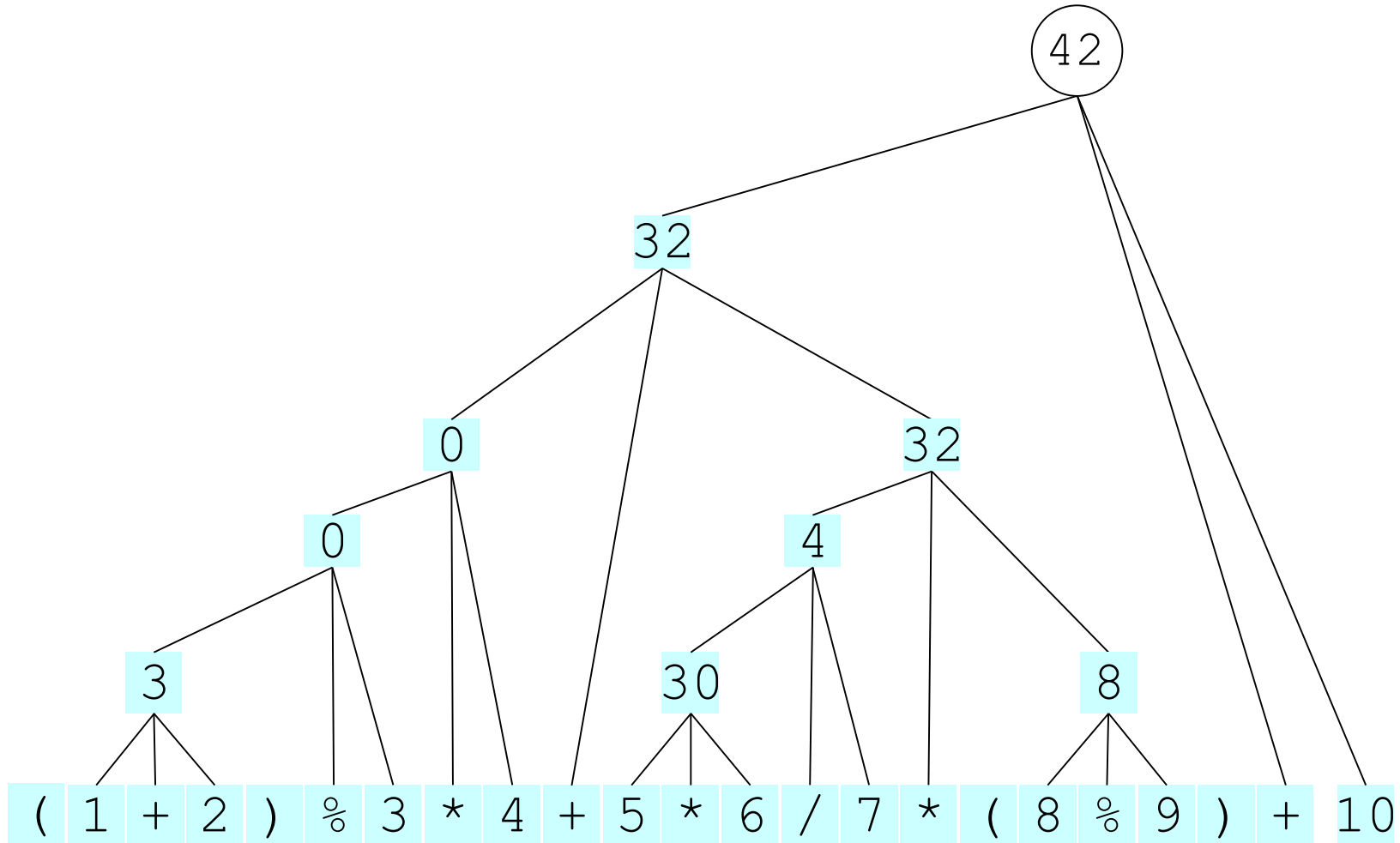


Thus, Java evaluates unary - operators and type casts first, then the operators \*, /, and %, and then the operators + and -.

- Precedence applies only when two operands compete for the same operator. If the operators are independent, Java evaluates expressions from left to right.
- Parentheses may be used to change the order of operations.

# Precedence Evaluation

What is the value of the expression at the bottom of the screen?





Simple rule

*Use parenthesis if you forgot the precedence rule*

# Assignment Statements

- You can change the value of a variable in your program by using an assignment statement, which has the general form:

```
variable = expression;
```

- The effect of an assignment statement is to compute the value of the expression on the right side of the equal sign and assign that value to the variable that appears on the left. Thus, the assignment statement

```
Z = X + Y; //recall from AddTwoNumbers.java
```

adds together the current values of the variables total and value and then stores that sum back in the variable total.

- When you assign a new value to a variable, the old value of that variable is lost.

# Shorthand Assignments

- Statements such as

`X = X + value;`

are so common that Java allows the following shorthand form:

`X += value;`

- The general form of a shorthand assignment is

***variable op= expression;***

where *op* is any of Java's binary operators. The effect of this statement is the same as

***variable = variable op (expression);***

For example, the following statement multiplies salary by 2.

`salary *= 2;`

# Increment and Decrement Operators

- Another important shorthand form that appears frequently in Java programs is the increment operator, which is most commonly written immediately after a variable, like this:

```
x++;
```

The effect of this statement is to add one to the value of `x`, which means that this statement is equivalent to

```
x += 1;
```

or in an even longer form

```
x = x + 1;
```

- The `--` operator (which is called the decrement operator) is similar but subtracts one instead of adding one.

# Boolean Expressions



George Boole (1791-1871)

In many ways, the most important primitive type in Java is boolean, even though it is by far the simplest. The only values in the boolean domain are true and false, but these are exactly the values you need if you want your program to make decisions.

The name boolean comes from the English mathematician George Boole who in 1854 wrote a book entitled *An Investigation into the Laws of Thought, on Which are Founded the Mathematical Theories of Logic and Probabilities*. That book introduced a system of logic that has come to be known as *Boolean algebra*, which is the foundation for the boolean data type.

# Boolean Operators

- The operators used with the boolean data type fall into two categories: relational operators and logical operators.
- There are six relational operators that compare values of other types and produce a boolean result:

<code>==</code>	Equals	<code>!=</code>	Not equals
<code>&lt;</code>	Less than	<code>&lt;=</code>	Less than or equal to
<code>&gt;</code>	Greater than	<code>&gt;=</code>	Greater than or equal to

For example, the expression `n <= 10` has the value `true` if `x` is less than or equal to 10 and the value `false` otherwise.

- There are also three logical operators:

<code>&amp;&amp;</code>	Logical AND	<code>p &amp;&amp; q</code>	means both <code>p</code> and <code>q</code>
<code>  </code>	Logical OR	<code>p    q</code>	means either <code>p</code> or <code>q</code> (or both)
<code>!</code>	Logical NOT	<code>!p</code>	means the opposite of <code>p</code>

# Notes on the Boolean Operators

- Remember that Java uses `=` to denote assignment. To test whether two values are equal, you must use the `==` operator.
- It is not legal in Java to use more than one relational operator in a single comparison as is often done in mathematics. To express the idea embodied in the mathematical expression

$$0 \leq x \leq 9$$

you need to make both comparisons explicit, as in

$$0 \leq x \ \&\& \ x \leq 9$$

- The `||` operator means *either or both*, which is not always clear in the English interpretation of *or*.
- Be careful when you combine the `!` operator with `&&` and `||` because the interpretation often differs from informal English.

# Short-Circuit Evaluation

- Java evaluates the `&&` and `||` operators using a strategy called short-circuit mode in which it evaluates the right operand only if it needs to do so.

- For example, if `n` is 0, the right hand operand of `&&` in

`n != 0 && x % n == 0`

is not evaluated at all because `n != 0` is false. Because the expression

`false && anything`

is always false, the rest of the expression no longer matters.

- One of the advantages of short-circuit evaluation is that you can use `&&` and `||` to prevent execution errors. If `n` were 0 in the earlier example, evaluating `x % n` would cause a “division by zero” error.



# Tips that you would need as you are programming

Limiting a number to two decimal points

```
1 import java.text.*;
2
3 public class Format {
4
5     public static void main(String[] args) {
6
7         double d = 1.566776;
8         DecimalFormat f = new DecimalFormat("#.##");
9         System.out.print(f.format(d));
10    }
11
12 }
13
```

```
13
15 }
17
```

# Tips that you would need

Reading Strings form the keyboard with or without spaces

Assuming sc is a scanner in your program..

....

**String S1 = sc.next();** //read until it finds a space

**String S2 = sc.nextLine();** //reads everything you type in the screen even if it is separated by a space.



NYU

Courant Institute of Mathematical Sciences  
Department of Computer Science  
CS101 Introduction to Computer Science

Anasse Bari, Ph.D.

End of Chapter#3: Expressions,  
Variables, Input output using Scanner,  
Introduction to Strings in Java

