I. Variables and Data Type

- <u>**variable**</u>: a named memory (i.e. RAM, which is volatile) location used to store/hold data, which can be changed during program execution

    in algebra: $3x + 5 = 20$, $x = 5$, x is the variable

- formally, a variable is defined as an object with the following properties:

    1. **name**: a tag to identify the variable in code

    2. **address**: a location in memory where the variable's value is stored

    3. **type**: determines the possible range of values the variable can store, as well as how much space the variable will occupy

    4. **value**: contents of the memory starting at *address* and as big as determined by the *type*

    5. **scope:** <u>where</u> in a program a variable's value is visible, i.e. where it can be referenced

    6. **lifetime:** <u>when</u> a program a variable's value is visible, i.e. where it can be referenced

- variable and constant **naming conventions**

    - name must begin with a letter, followed by 0 or more letters, numbers, and/or underscore characters
    - the name cannot be a keyword
    - use meaningful names, i.e. X is not a meaningful name
    - 1st word lower case, then upper case 1st letter of each following word, i.e. camelCase

    - as always, variable names are CASE SENSITIVE!

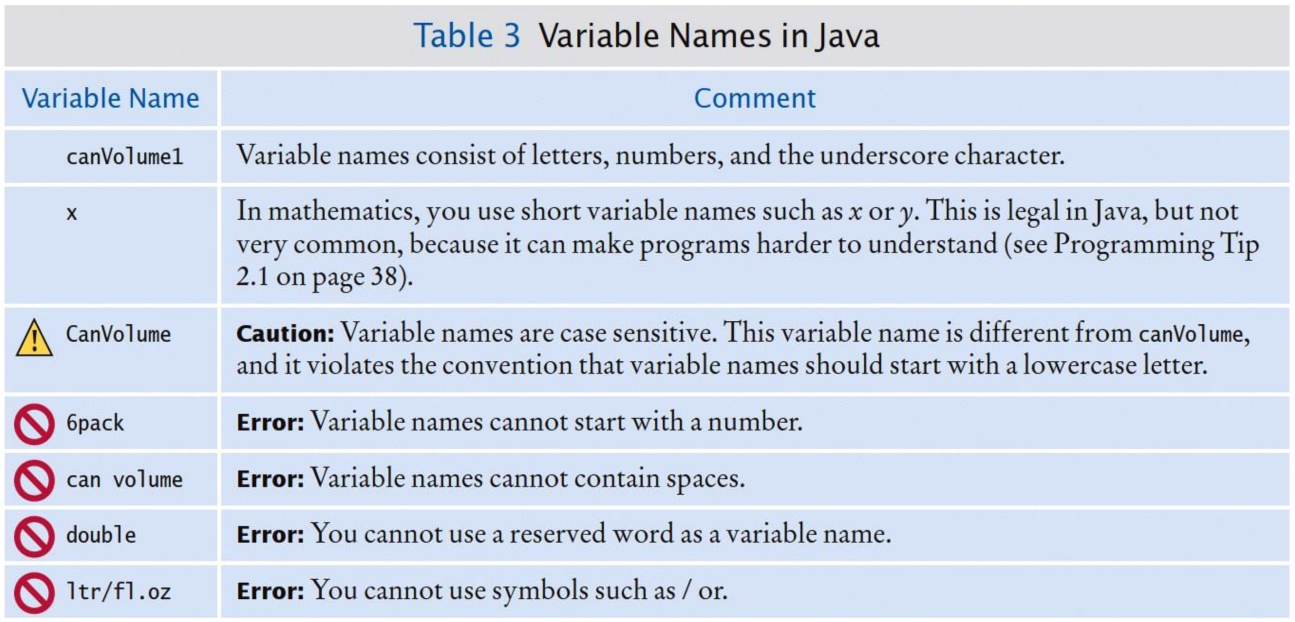


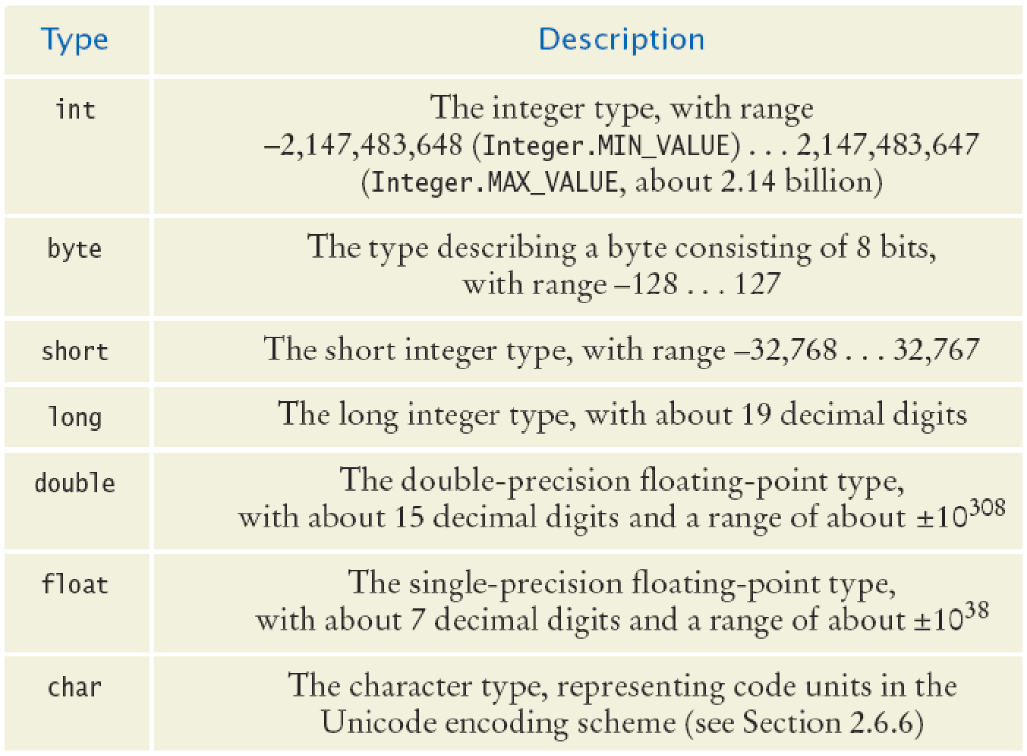
Table 3 Variable Names in Java

Variable Name	Comment
`canVolume1`	Variable names consist of letters, numbers, and the underscore character.
`x`	In mathematics, you use short variable names such as *x* or *y*. This is legal in Java, but not very common, because it can make programs harder to understand (see Programming Tip 2.1 on page 38).
⚠ `CanVolume`	**Caution:** Variable names are case sensitive. This variable name is different from `canVolume`, and it violates the convention that variable names should start with a lowercase letter.
🚫 `6pack`	**Error:** Variable names cannot start with a number.
🚫 `can volume`	**Error:** Variable names cannot contain spaces.
🚫 `double`	**Error:** You cannot use a reserved word as a variable name.
🚫 `ltr/fl.oz`	**Error:** You cannot use symbols such as / or.

Table 2.3
© John Wiley & Sons, Inc. All rights reserved.

Variable Declaration:

- all variables/constants in Java must be declared – this allows the programmer to give the variable a **TYPE** and a **NAME**



Type	Description
`int`	The integer type, with range –2,147,483,648 (`Integer.MIN_VALUE`) . . . 2,147,483,647 (`Integer.MAX_VALUE`, about 2.14 billion)
`byte`	The type describing a byte consisting of 8 bits, with range –128 . . . 127
`short`	The short integer type, with range –32,768 . . . 32,767
`long`	The long integer type, with about 19 decimal digits
`double`	The double-precision floating-point type, with about 15 decimal digits and a range of about $\pm 10^{308}$
`float`	The single-precision floating-point type, with about 7 decimal digits and a range of about $\pm 10^{38}$
`char`	The character type, representing code units in the Unicode encoding scheme (see Section 2.6.6)

- general form: [access] type var_name [= value] ; // indicates optional

- Examples:

```
int age;
int sum = 20;
double subTotal = 33.88;
String firstName ;          // String is a Class in Java, not a primitive type
```

- variable types should be selected based upon the values the variable are expected to contain

- variable type are essentially classes, and the declaring of the variable instantiates an object of that type

- Java **does not assign default values for local variables**, only for class and instance variables, which get initialized to 0, or something that closely resembles 0, e.g. strings initialized to null

- I strongly suggest declaring all variables together, then do assignments at the appropriate spot in the program (note this is different from the book)

Variable Assignment:

- assigning values to variables

- general syntax: var_name = expression ; (note: assignment works L ← R)

- expression can be

  - compatible literal value
  - another variable of ??? type
  - mathematical equation
  - call to a method that returns a compatible value
  - combination of one or more items in this list

- examples:

```
age = 22 ;          // what is name, what is type, what is data value
sum = sum + 1 ;
name = “Mark” ;
answer = sum ;
```

- bad examples:

```
int age = abc ;
int age = "22" ;
String Name = 22 ;
int sum = 22.5 ;
String firstName = Mark Thomas ;
```

Note: 22 = age is not a valid assignment (although it is in Algebra)

STRONGLY SUGGESTED techniques

1. declare all necessary variables needed for the entire block, grouping likely used variables together, in order

2. perform any necessary variable assignments, in order of need

Named Constants

- a named memory location used to store/hold data, which CANNOT be changed during program execution

- general syntax: [access] **final** type var_name = expression ;

- naming conventions: all Upper Case with _ between words, e.g. TAX_RATE

- examples:

```
final double TAX_RATE = 0.065 ;

final double  PI = 3.14159 ;
```

- may need to use type declaration characters following the number:

D / d : double **	F/f : float
S / s : short (integer)	L : long (integer)

** default

II. Variables and Arithmetic Operators

What is an arithmetic operator? Something that performs operations.

What does an operator perform operations on?   **Operands**.

	Name	Symbol	# Operands	
1.	addition	+	2	e.g. num = num + 5 ;
2.	subtraction	-	2	
3.	multiplication	*	2	
4.	division	/	2	
5.	negation	-	1	unary

Operator precedence – order of arithmetic evaluation

1. parenthesis
2. negation
3. multiplication/division
4. addition/subtraction

```
? = (2 + 3) * (3 + 3)          // answer =

? = 4 + 6 / 2                  // answer =

? = 1 + 2 * 3                  // answer =
```

To make precedence a bit more complex, if expressions contain operators of equal precedence, the order of operations is left to right.

```
? = 8 / 2 * 3                  // answer = 12

? = 8 + 40 / 8 * 2 + 4         // answer = 22

? = 7 – 2 – 4 + 1              // answer = 2
```

- Typical Types of Arithmetic Techniques

  1. counting

     - typically count a single unit/item
     - typically count by 1 (or –1)
     - e.g. count = count + 1 ;

  2. accumulation

     - act of accumulating a total value (i.e. a sum)
     - typically don't accumulate by a single unit/item
     - e.g. sum = sum + itemPrice ;

III. Division and Exponentiation

Division

- division in Java is based upon the **types** of the numbers in the operation

- if all numbers are type int, the result will be an int, e.g.

  - 8 / 5 yields 1, because 8 and 5 are int types
  - 8.0 / 5.0 yields 1.6, because both types are double
  - 8.0 / 5 yields 1.6, because one of the numbers is a double, so Java converts the smaller (int) to a larger (double), and the result is a double

- be very careful with this

Exponentiation

- exponentiation in Java uses the Math.pow method

- syntax: double Math.**pow**(double a, double b)

- example: twoSquared = Math.pow (2.0, 2.0);

IV. Getting Console Input (BJ 2.3)

- we have already used the System.out object, with println and print methods
- to get input, we use the System.in object, but a bit differently
- first, we need to define a scanner class as follows:

```
Scanner in = new Scanner(System.in);
```

- the new statement, called a constructor, construct or builds a new object, in this case, a Scanner object, passing it the System.in object to specify where to scan from
- the 2nd part of the above statement, declares a variable named **in** of type Scanner and assigns the newly constructed object to it
- next, to actually get the input, you use the **in** variable with the **.nextInt()** or **.nextDouble()** method as follows:

```
int number;

number = in.nextInt();
```

- note, when using a Scanner class, you'll most likely see an error as the class containing the Scanner class (java.util) has not been imported, to fix this, you can either

  1. select Source…Fix Imports…OK
  2. manually include the missing class statement: import java.util.Scanner;

V. Example Program

- example task: create a program to prompt the user to enter a number (as int), take the number and double it, then output results
- understand the problem?
- how many inputs? how many outputs?
- how many variables?
- see NumberDoubler program exercise in class/solution on syllabus

- steps (or algorithm)

  1. prompt user for input (use System.out.print)
  2. get the input from the console and assign to variable (use Scanner class)
  3. take the stored number and double it, assigning output to another variable
  4. display the output (the doubled number)

- **suggested practice**: convert NumberDoubler program to work on **double** types