LECTURE 2

TYPES AND METHODS

Lecture Goals

- To understand types and typing
- To declare and initialize variables and constants
- To write arithmetic expressions and assignment statements
- To learn how to use the Java `String` type
- To be able to implement methods
- To become familiar with parameter passing
- To be able to determine the scope of a variable

2.1 Variables

- Most computer programs hold temporary values in named storage locations
  - Programmers name them for easy access
- There are many different types (sizes) of storage to hold different things
- You ‘declare’ a variable by telling the compiler:
  - What **type** (size) of variable you need
  - What name you will use to refer to it

Syntax 2.1: Variable Declaration

- When declaring a variable, you often specify an initial value
- This is also where you tell the compiler the size (type) it will hold

See page 35 for rules and examples of valid names.

Types introduced in this chapter are the number types `int` and `double` (page 34) and the `String` type (page 60).

```
int cansPerPack = 6;
```

A variable declaration ends with a semicolon.

Use a descriptive variable name. See page 38.

Supplying an initial value is optional, but it is usually a good idea. See page 37.

Python vs. Java

Dynamically Typed

- Variables are used without explicit type
- Variables can change type
- Type determined by how the variable is used (operated upon)
- Type errors cause runtime error

Statically Typed

- Variables are declared with explicit type
- Variable type is fixed once declared
- Type checked at compile time
- Type errors cause compiler error
- Limited type conversion

Example Declarations

Table 1 Variable Declarations in Java

Variable Name	Comment
`int cans = 6;`	Declares an integer variable and initializes it with 6.
`int total = cans + bottles;`	The initial value need not be a fixed value. (Of course, `cans` and `bottles` must have been previously declared.)
`bottles = 1;`	**Error:** The type is missing. This statement is not a declaration but an assignment of a new value to an existing variable—see Section 2.1.4.
`int volume = "2";`	**Error:** You cannot initialize a number with a string.
`int cansPerPack;`	Declares an integer variable without initializing it. This can be a cause for errors—see Common Error 2.1 on page 37.
`int dollars, cents;`	Declares two integer variables in a single statement. In this book, we will declare each variable in a separate statement.

Why different types?

- There are three different types of variables that we will use in this chapter:
  1) A whole number (no fractional part) `int`
  2) A number with a fraction part `double`
  3) A word (a group of characters) `String`

- Specify the type before the name in the declaration

```
int cansPerPack = 6;
double canVolume = 12.0;
```

Number Literals in Java

Number	Type	Comment
6	`int`	An integer has no fractional part.
-6	`int`	Integers can be negative.
0	`int`	Zero is an integer.
0.5	`double`	A number with a fractional part has type `double`.
1.0	`double`	An integer with a fractional part .0 has type `double`.
1E6	`double`	A number in exponential notation: 1×10^6 or 1000000. Numbers in exponential notation always have type `double`.
2.96E-2	`double`	Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$
100,000		**Error:** Do not use a comma as a decimal separator.
3 1/2		**Error:** Do not use fractions; use decimal notation: 3.5

Floating-Point Numbers

- Java stores numbers with fractional parts as ‘floating point’ numbers.
- They are stored in four parts
  - Sign
  - Mantissa
  - Radix
  - Exponent

Parts of a floating point number -5:

Sign	Mantissa	Radix ^{exponent}
-1	5	10^{0}

- A ‘`double`’ is a double-precision floating point number: It takes twice the storage (52 bit mantissa) as the smaller ‘`float`’ (23 bit mantissa)

See JavaWorld article for more detail

Naming Variables

- Name should describe the purpose
  - ‘`canVolume`’ is better than ‘`cv`’
- Use These Simple Rules

1) Variable names must start with a letter or the underscore (_) character
   - Continue with letters (upper or lower case), digits or the underscore
2) You cannot use other symbols (? or %...) and spaces are not permitted
3) Separate words with ‘camelHump’ notation
   - Use upper case letters to signify word boundaries
4) Don’t use reserved ‘Java’ words (see Appendix C)

Variable Names in Java

Variable Name	Comment
canVolume1	
x	
CanVolume	
6pack	
can volume	
double	
ltr/fl.oz	

The Assignment Statement

- Use the ‘assignment statement’ (with an '=') to place a new value into a variable

```
int cansPerPack = 6;    // declare & initialize
cansPerPack = 8;        // assignment
```

- Beware: The **=** sign is **NOT** used for comparison:
  - use **==** or **equals()** instead!

Assignment Syntax

- The value on the right of the '=' sign is copied to the variable on the left

This is an initialization of a new variable, NOT an assignment.

```
double total = 0;
    .
    .
total = bottles * BOTTLE_VOLUME;

    .
    .
    .
total = total + cans * CAN_VOLUME;
```

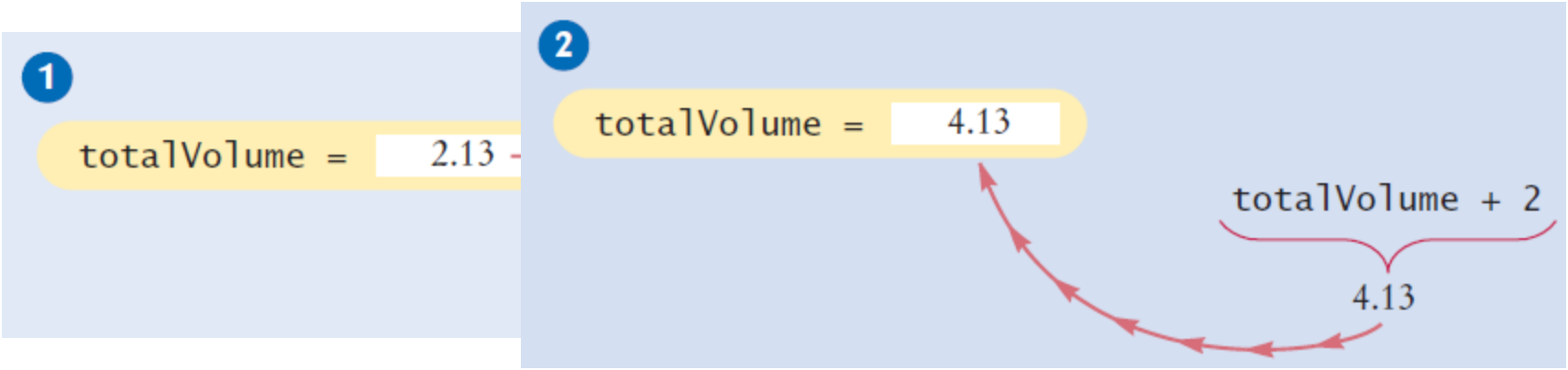
This is an assignment.

The name of a previously defined variable

The expression that replaces the previous value

The same name can occur on both sides. See Figure 1.

Updating a Variable



- Step by Step:

```
totalVolume = totalVolume + 2;
```

1. Calculate the right hand side of the assignment
   Find the value of `totalVolume`, and add 2 to it
2. Store the result in the variable named on the left side of the assignment operator (`totalVolume` in this case)

Constants

- When a variable is defined with the reserved word `final`, its value can never be changed

```
final double BOTTLE_VOLUME = 2;
```

- It is good style to use named constants to explain numerical values to be used in calculations
  - Which is clearer?

```
double totalVolume = bottles * 2;
double totalVolume = bottles * BOTTLE_VOLUME;
```

- No magic numbers (why 2?)
- Change constant only in one place

Constant Declaration

The `final` reserved word indicates that this value cannot be modified.

```
final double CAN_VOLUME = 0.355; // Liters in a 12-ounce can
```

Use uppercase letters for constants.

This comment explains how the value for the constant was determined.

- It is customary (not required) to use all UPPER_CASE letters for constants

Java Comments

```
// single line (or rest of line to right)

/*
        multi-line – all comment until matching
 */
```

Use comments to add explanations for humans who read your code. The compiler ignores comments.

Java Comment Example

```
1   /**
2      This program computes the volume (in liters) of a six-pack of soda
3      cans and the total volume of a six-pack and a two-liter bottle.
4   */
5   public class Volume1
6   {
7      public static void main(String[] args)
8      {
9        int cansPerPack = 6;
10       final double CAN_VOLUME = 0.355; // Liters in a 12-ounce can
11       double totalVolume = cansPerPack * CAN_VOLUME;
12
13       System.out.print("A six-pack of 12-ounce cans contains ");
14       System.out.print(totalVolume);
15       System.out.println(" liters.");
16
17       final double BOTTLE_VOLUME = 2; // Two-liter bottle
```

Common Error 2.1

- Undeclared Variables
  - You must declare a variable before you use it: (i.e. above in the code)

```
double canVolume = 12 * literPerOunce; // ??
double literPerOunce = 0.0296;
```

- Uninitialized Variables
  - You must initialize (i.e. set) a variable's contents before you use it

```
int bottles;
int bottleVolume = bottles * 2;   // ??
```

Common Error 2.2

- Overflow means that storage for a variable cannot hold the result

```
int fiftyMillion = 50000000;
System.out.println(100 * fiftyMillion);
    // Expected: 5000000000
```

Will print out 705032704

- Why?
  - The result (5 billion) overflowed `int` capacity
  - Maximum value for an `int` is **+2,147,483,647**
- Use a `long` instead of an `int` (or a `double`)

Common Error 2.3

- Roundoff Errors
  - Floating point values are not exact
    - This is a limitations of binary values (no fractions):

```
double price = 4.35;
double quantity = 100;
double total = price * quantity;
 // Should be 100 * 4.35 = 435.00
System.out.println(total); // Prints 434.99999999999999
```

- You can deal with roundoff errors by rounding to the nearest integer (see Section 2.2.5) or by displaying a fixed number of digits after the decimal separator (see Section 2.3.2).

All of the Java Numeric Types

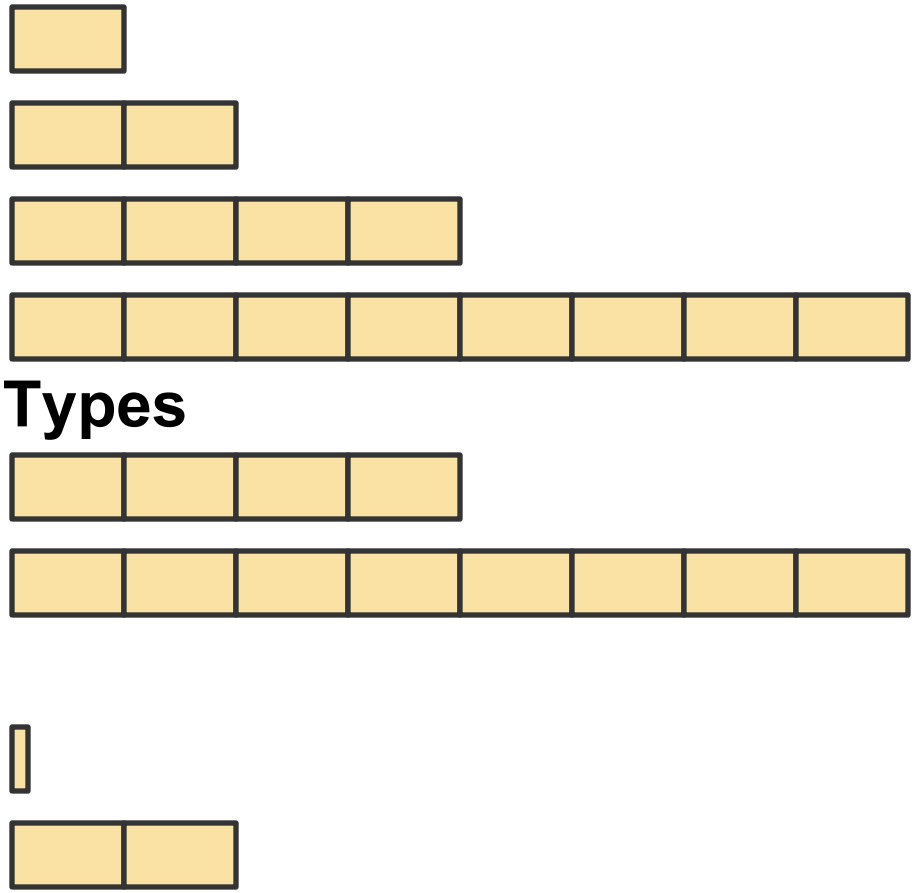
Type	Description	
`int`	The integer type, with range –2,147,483,648 (`Integer.MIN_VALUE`) . . . 2,147,483,647 (`Integer.MAX_VALUE`, about 2.14 billion)	Whole Numbers (no fractions)
`byte`	The type describing a byte consisting of 8 bits, with range –128 . . . 127	
`short`	The short integer type, with 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 $\pm 10^{308}$	Floating point Numbers
`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)	Characters (no math)

Value Ranges per Type

- **Integer Types**
  - `byte:` **A very small number (-128 to +127)**
  - `short:` **A small number (-32768 to +32767)**
  - `int:` **A large number (-2,147,483,648 to +2,147,483,647)**
  - `long:` **A huge number**
- **Floating Point Types**
  - `float:` **A huge number with decimal places**
  - `double:` **Much more precise, for heavy math**
- **Other Types**
  - `boolean: true` **or** `false`
  - `char:` **One symbol in single quotes 'a'**

Storage per Type (in bytes)

- **Integer Types**
  - `byte:`
  - `short:`
  - `int:`
  - `long:`
- **Floating Point Types**
  - `float:`
  - `double:`
- **Other Types**
  - `boolean:`
  - `char:`



Mixing Numeric Types

- It is safe to convert a value from an integer type to a floating-point type
  - No ‘precision’ is lost
- But going the other way can be dangerous
  - All fractional information is lost
  - The fractional part is discarded (not rounded)
- If you mix types integer and floating-point types in an expression, no precision is lost:

```
double area, pi = 3.14;
int radius = 3;
area = radius * radius * pi;
```

Mixing integers and floating-point values in an arithmetic expression yields a floating-point value.

Operators

- Assignment: =
- Comparison: == < >. <= >=
- Inc/decrement: ++ --
- Arithmetic: + - * / %

```
int pennies = 1729;
int dollars = pennies / 100;  // 17
int cents = pennies % 100;    // 29
```

Integer Division and Remainder

- When both parts of division are integers, the result is an integer.
  - All fractional information is lost (no rounding)

```
int result = 7 / 4;
```

  - The value of result will be 1

Integer division loses all fractional parts of the result and does not round

- If you are interested in the remainder of dividing two integers, use the % operator (called modulus):

```
int remainder = 7 % 4;
```

  - The value of remainder will be 3
  - Sometimes called modulo divide

Floating-Point to Integer Conversion

- The Java compiler does not allow direct assignment of a floating-point value to an integer variable

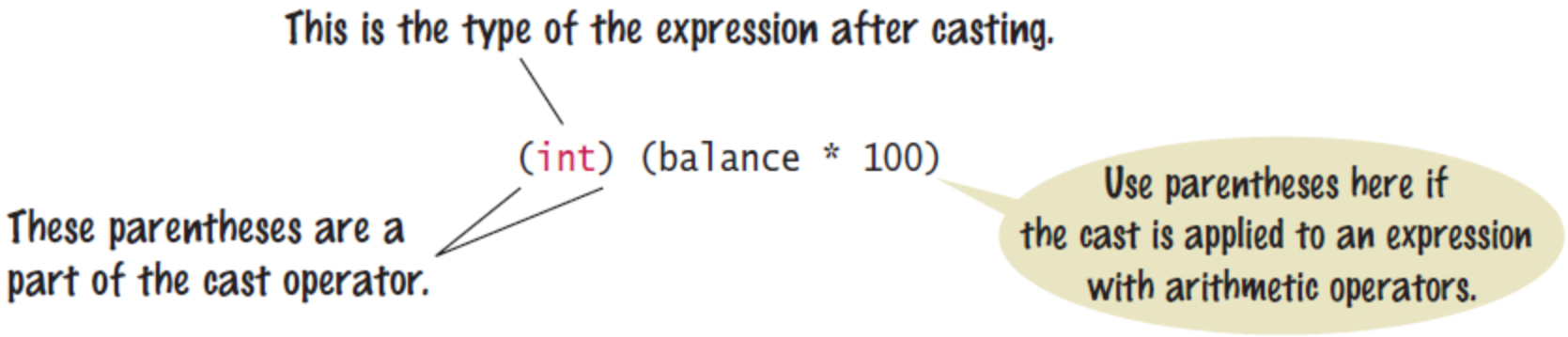
```
double balance = total + tax;
int dollars = balance; // Error
```

- You can use the 'cast' operator: `(int)` to force the conversion:

```
double balance = total + tax;
int dollars = (int) balance; // no Error
```

- You lose the fractional part of the floating-point value (no rounding occurs)

Cast Syntax



- Casting is a very powerful tool and should be used carefully
- To round a floating-point number to the nearest whole number, use the `Math.round` method
- This method returns a long integer, because large floating-point numbers cannot be stored in an `int`

```
long rounded = Math.round(balance);
```

Mathematical Methods

Method	Returns
`Math.sqrt(x)`	Square root of x (≥ 0)
`Math.pow(x, y)`	x^y ($x > 0$, or $x = 0$ and $y > 0$, or $x < 0$ and y is an integer)
`Math.sin(x)`	Sine of x (x in radians)
`Math.cos(x)`	Cosine of x
`Math.tan(x)`	Tangent of x
`Math.toRadians(x)`	Convert x degrees to radians (i.e., returns $x \cdot \pi/180$)
`Math.toDegrees(x)`	Convert x radians to degrees (i.e., returns $x \cdot 180/\pi$)
`Math.exp(x)`	e^x
`Math.log(x)`	Natural log ($\ln(x)$, $x > 0$)

Arithmetic Expressions

Mathematical Expression	Java Expression	Comments
$\frac{x + y}{2}$	`(x + y) / 2`	The parentheses are required; `x + y / 2` computes $x + \frac{y}{2}$.
$\frac{xy}{2}$	`x * y / 2`	Parentheses are not required; operators with the same precedence are evaluated left to right.
$\left(1 + \frac{r}{100}\right)^n$	`Math.pow(1 + r / 100, n)`	Use `Math.pow(x, n)` to compute x^n.
$\sqrt{a^2 + b^2}$	`Math.sqrt(a * a + b * b)`	`a * a` is simpler than `Math.pow(a, 2)`.
$\frac{i + j + k}{3}$	`(i + j + k) / 3.0`	If i, j, and k are integers, using a denominator of 3.0 forces floating-point division.
π	`Math.PI`	`Math.PI` is a constant declared in the `Math` class.

Common Error 2.4

- Unintended Integer Division

```
System.out.print("Please enter your last three test
  scores: ");
int s1 = in.nextInt();
int s2 = in.nextInt()
int s3 = in.nextInt();
double average = (s1 + s2 + s3) / 3; // Error
```

- Why?
  - All of the calculation on the right happens first
    - Since all are `ints`, the compiler uses integer division
  - Then the result (an `int`) is assigned to the `double`
    - There is no fractional part of the `int` result, so zero (.0) is assigned to the fractional part of the `double`

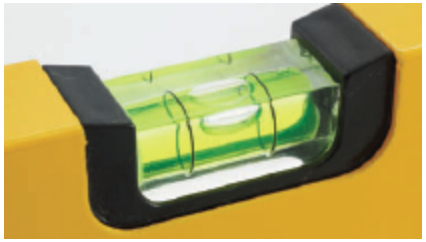
Common Error 2.5

- Unbalanced Parenthesis
  - Which is correct?

```
(-(b * b - 4 * a * c) / (2 * a)    // 3 (, 2 )
-(b * b - (4 * a * c))) / 2 * a)   // 2 (, 2 )
```

- The count of (and) must match
- Unfortunately, it is hard for humans to keep track
  - Here's a handy trick
  - Count (as +1, and) as -1: Goal: 0

```
-(b * b - (4 * a * c) ) ) / 2 * a)
 1        2          1 0 -1      -2
```



Summary: Variables

- A variable is a storage location with a name.
- When declaring a variable, you usually specify an initial value.
- When declaring a variable, you also specify the type of its values.
- Use the `int` type for numbers that cannot have a fractional part.
- Use the `double` type for floating-point numbers.
- By convention, variable names should start with a lower case letter.
- An assignment statement stores a new value in a variable, replacing the previously stored value

Summary: Operators

- The assignment operator `=` does not denote mathematical equality.
- You cannot change the value of a variable that is defined as `final`.
- The `++` operator adds 1 to a variable; the `--` operator subtracts 1.
- If both arguments of `/` are integers, the remainder is discarded.
- The `%` operator computes the remainder of an integer division.
- The Java library declares many mathematical functions, such as `Math.sqrt` and `Math.pow`.
- You use a cast (*typeName*) to convert a value to a different type.
- Java classes are grouped into packages. Use the `import` statement to use classes from packages.

BREAK

2.5 Strings

- The `String` Type:
  - Type Variable Literal
  - `String name = “Harry”`
- Once you have a `String` variable, you can use methods such as:

```
int n = name.length();  // n will be assigned 5
```

- A `String`’s length is the number of characters inside:
  - An empty `String` (length 0) is shown as “”
  - The maximum length is quite large (an `int`)

String Concatenation (+)

- You can ‘add’ one String onto the end of another

```
String fName = “Harry”
String lName = “Morgan“
String name = fname + lname;  // HarryMorgan
```

- You wanted a space in between?

```
String name = fname + “ “ + lname;  // Harry Morgan
```

- To concatenate a numeric variable to a String:

```
String a = “Agent“;
int n = 7;
String bond = a + n;    // Agent7
```

- Concatenate Strings and numerics inside println:

```
System.out.println("The total is " + total);
```

String Input

- You can read a `String` from the console with:

```
System.out.print("Please enter your name: ");
String name = in.next();
```

  - The `next` method reads one word at a time
  - It looks for 'white space' delimiters
- You can read an entire line from the console with:

```
System.out.print("Please enter your address: ");
String address = in.nextLine();
```

  - The `nextLine` method reads until the user hits 'Enter'
- Converting a `String` variable to a number

```
System.out.print("Please enter your age: ");
String input = in.nextLine();
int age = Integer.parseInt(input);  // only digits!
```

String Escape Sequences

- How would you print a double quote?
  - Preface the " with a \ inside the double quoted String

```
System.out.print("He said \"Hello\"");
```

- OK, then how do you print a backslash?
  - Preface the \ with another \!

```
System.out.print("“C:\\Temp\\Secret.txt“);
```

- Special characters inside Strings
  - Output a newline with a ‘\n’

```
System.out.print("*\n**\n***\n");
```

```
*
**
***
```

Strings and Characters

- Strings are sequences of characters
  - Unicode characters to be exact
  - Characters have their own type: `char`
  - Characters have numeric values
    - See the ASCII code chart in Appendix B
    - For example, the letter 'H' has a value of 72 if it were a number

- Use single quotes around a `char`

```
char initial = 'B';
```

- Use double quotes around a `String`

```
String initials = "BRL";
```

Copying a `char` from a `String`

- Each `char` inside a `String` has an index number:

0	1	2	3	4	5	6	7	8	9
c	h	a	r	s		h	e	r	e

- The first `char` is index zero (`0`)
- The `charAt` method returns a `char` at a given index inside a `String`:

```
String greeting = "Harry";
char start = greeting.charAt(0);
char last = greeting.charAt(4);
```

0	1	2	3	4
H	a	r	r	y

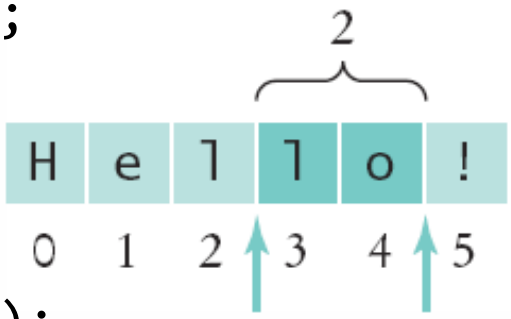
Copying portion of a `String`

- A substring is a portion of a `String`
- The `substring` method returns a portion of a `String` at a given index for a number of `chars`, starting at an index:

0	1	2	3	4	5
H	e	l	l	o	!

0	1
H	e

```
String greeting = "Hello!";
String sub = greeting.substring(0, 2);
```




```
String sub2 = greeting.substring(3, 5);
```

Table 9: String Operations (1)

Table 9 String Operations

Statement	Result	Comment
`string str = "Ja";` `str = str + "va";`	`str` is set to `"Java"`	When applied to strings, `+` denotes concatenation.
`System.out.println("Please"` `  + " enter your name: ");`	Prints `Please enter your name:`	Use concatenation to break up strings that don't fit into one line.
`team = 49 + "ers"`	`team` is set to `"49ers"`	Because `"ers"` is a string, 49 is converted to a string.

`String first = in.next();`
`String last = in.next();`
`(User input: Harry Morgan)`
	`first` contains `"Harry"`
	`last` contains `"Morgan"`
		The `next` method places the next word into the string variable.

`String greeting = "H & S";` `int n = greeting.length();`	`n` is set to 5	Each space counts as one character.
`String str = "Sally";` `char ch = str.charAt(1);`	`ch` is set to `'a'`	This is a `char` value, not a `String`. Note that the initial position is `0`.

Table 9: String Operations (2)

Statement	Result	Comment
`String str = "Sally";` `String str2 = str.substring(1, 4);`	`str2` is set to `"all"`	Extracts the substring starting at position 1 and ending before position 4.
`String str = "Sally";` `String str2 = str.substring(1);`	`str2` is set to `"ally"`	If you omit the end position, all characters from the position until the end of the string are included.
`String str = "Sally";` `String str2 = str.substring(1, 2);`	`str2` is set to `"a"`	Extracts a `String` of length 1; contrast with `str.charAt(1)`.
`String last = str.substring(` `  str.length() - 1);`	`last` is set to the string containing the last character in `str`	The last character has position `str.length() - 1`.

5.1 Java Methods

A method packages a computation consisting of multiple steps into a form that can be easily understood and reused.

- You *declare* a method by defining a named block of code

```
public static void main(String[] args)
{
  double result = Math.pow(2, 3);
  . . .
}
```

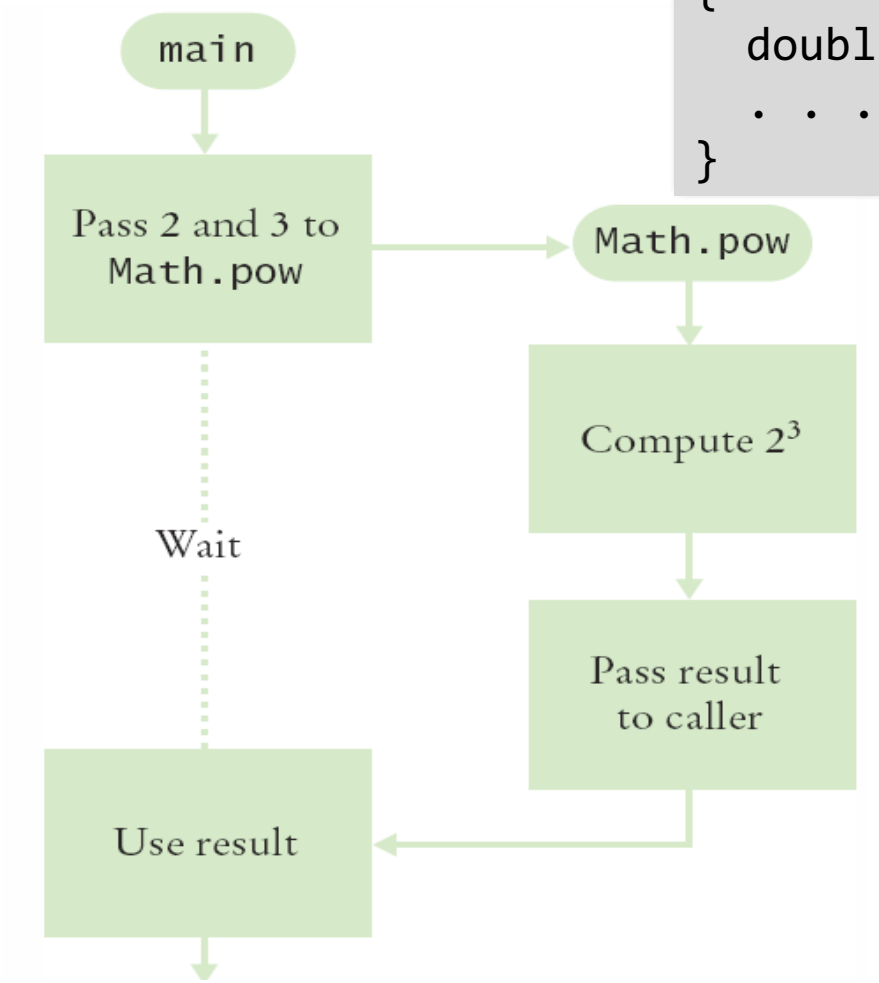
- You *call* a method in order to execute its instructions

What is a method?

- Some methods you have already used are:
  - `Math.pow()`
  - `String.length()`
  - `Character.isDigit()`
  - `Scanner.nextInt()`
  - `main()`
- They have:
  - May have a capitalized name and a dot (.) before them
  - A method name
    - Follow the same rules as variable names, camelHump style
  - `( )` - a set of parenthesis at the end
    - A place to provide the method input information

Flowchart of Calling a Method

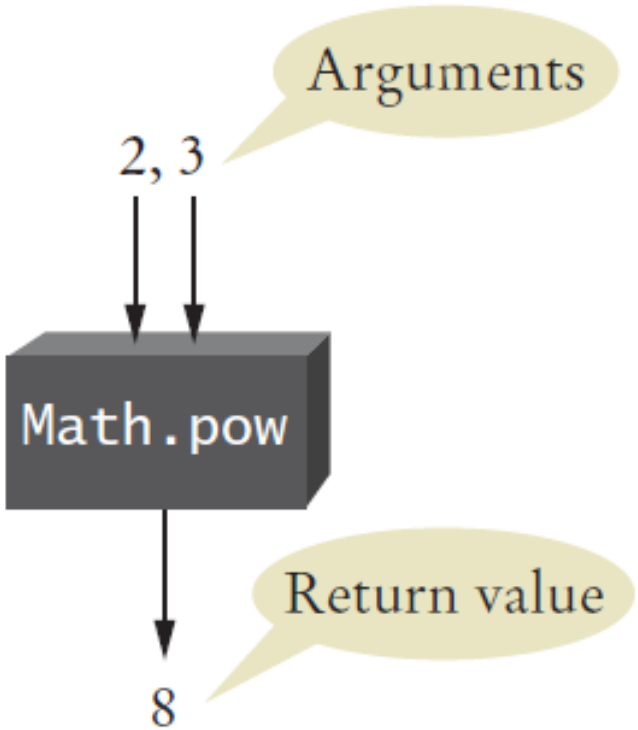
```
public static void main(String[] args)
{
  double result = Math.pow(2, 3);
  . . .
}
```



- One method 'calls' another
  - main calls `Math.pow()`
  - Passes two arguments
    - 2 and 3
  - `Math.pow` starts
    - Uses variables (2, 3)
    - Does its job
    - Returns the answer
  - main uses result

Arguments and Return Values

```
public static void main(String[] args)
{
  double result = Math.pow(2,3);
  . . .
}
```



- `main` 'passes' two arguments (2 and 3) to `Math.pow`
- `Math.pow` calculates and returns a value of 8 to `main`
- `main` stores the return value to variable 'result'

Black Box Analogy

- A thermostat is a 'black box'
  - Set a desired temperature
  - Turns on heater/AC as required
  - You don't have to know how it really works!
    - How does it know the current temp?
    - What signals/commands does it send to the heater or A/C?
- Use methods like 'black boxes'
  - Pass the method what it needs to do its job
  - Receive the answer

5.2 Implementing Methods

- A method to calculate the volume of a cube
  - What does it need to do its job?
  - What does it answer with?
- When writing this method:
  - Pick a name for the method (`cubeVolume`).
  - Declare a variable for each incoming argument (`double sideLength`) (called parameter variables)
  - Specify the type of the return value (`double`)
  - Add modifiers such as `public static`
    - (see Chapter 8)

When declaring a method, you provide a name for the method, a variable for each argument, and a type for the result

```
public static double cubeVolume(double sideLength)
```

Declaring cubeVolume method

- Then write the body of the method
  - The body is surrounded by curly braces { } (instead of with indentation as done in Python)
  - The body contains the variable declarations and statements that are executed when the method is called
  - It will also return the calculated answer

```
public static double cubeVolume(double sideLength)
{
  double volume = sideLength * sideLength * sideLength;
  return volume;
}
```

Syntax 5.1: Method Declaration

Type of return value

Type of parameter variable

Name of method

Name of parameter variable

```
public static double cubeVolume(double sideLength)
{
   double volume = sideLength * sideLength * sideLength;
   return volume;
}
```

Method body, executed when method is called.

`return` statement exits method and returns result.

Calling cubeVolume method

- The values returned from `cubeVolume` are stored in local variables inside `main`
- The results are then printed out

```
public static void main(String[] args)
{
   double result1 = cubeVolume(2);
   double result2 = cubeVolume(10);
   System.out.println("A cube of side length 2 has volume
         "  + result1);
   System.out.println("A cube of side length 10 has volume
         " + result2);
}
```

Cubes.java

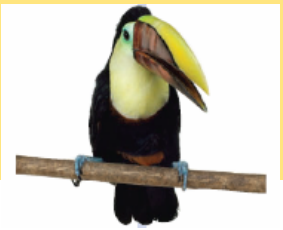
```java
/**
   This program computes the volumes of two cubes.
*/
public class Cubes
{
   public static void main(String[] args)
   {
      double result1 = cubeVolume(2);
      double result2 = cubeVolume(10);
      System.out.println("A cube with side length 2 has volume " + result1);
      System.out.println("A cube with side length 10 has volume " + result2);
   }

   /**
      Computes the volume of a cube.
      @param sideLength the side length of the cube
      @return the volume
   */
   public static double cubeVolume(double sideLength)
   {
      double volume = sideLength * sideLength * sideLength;
      return volume;
   }
}
```

Program Run

```
A cube with side length 2 has volume 8
A cube with side length 10 has volume 1000
```

Method Comments

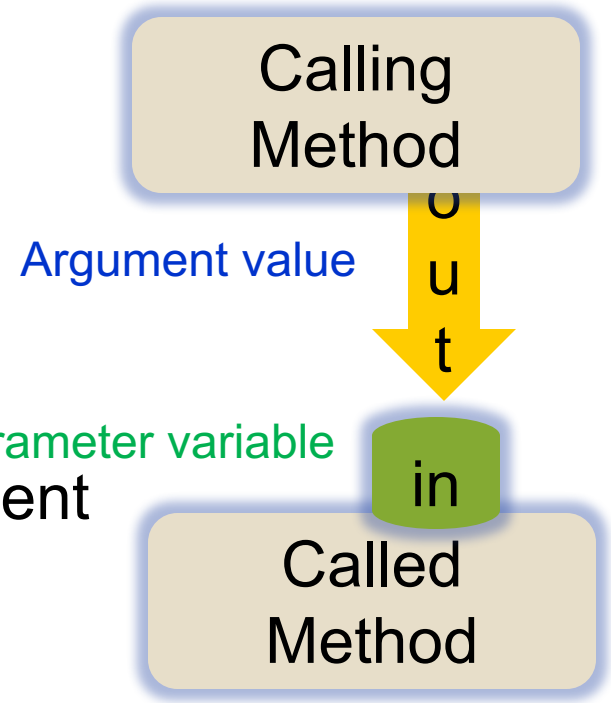


- Write a Javadoc comment above each method
- Start with `/**`
  - Note the purpose of the method
  - `@param` Describe each parameter variable
  - `@return` Describe the return value
- End with `*/`

```
/**
  Computes the volume of a cube.
  @param sideLength the side length of the cube
  @return the volume
*/
public static double cubeVolume(double sideLength)
```

5.3 Parameter Passing

- Parameter variables receive the argument values supplied in the method call
  - They both must be the same type
- The argument value may be:
  - The contents of a variable
  - A 'literal' value (2)
  - aka. 'actual parameter' or argument
- The parameter variable is:
  - Declared in the called method
  - Initialized with the value of the argument value
  - Used as a variable inside the called method
  - aka. 'formal parameter'



Parameter Passing Steps

```
public static void main(String[] args)
{
  double result1 = cubeVolume(2);
  . . .
}
```

result1 = 8

```
public static double cubeVolume(double sideLength)
{
  double volume = sideLength * sideLength * sideLength;
  return volume;
}
```

sideLength = 2

volume = 8

Common Error 5.1

- Trying to Modify Arguments
  - A copy of the argument values is passed
  - Called method (`addTax`) can modify local copy (`price`)
    - But not original in calling method
      - `total`

```
public static void main(String[] args)
{
  double total = 10;
  addTax(total,       7.5);
}
```

total

10.0

copy

```
public static int addTax(double price, double rate)
{
  double tax = price * rate / 100;
  price = price + tax; // Has no effect outside the method
  return tax;
}
```

price

10.75

5.4 Return Values

- Methods can (optionally) return one value
  - Declare a return type in the method declaration
  - Nothing to return? void return type, just call return;
  - Add a return statement that returns a value
    - A return statement does two things:
      1) Immediately terminates the method
      2) Passes the return value back to the calling method

return type

```
public static double cubeVolume (double sideLength)
{
  double volume = sideLength * sideLength * sideLength;
  return volume;
}
```

return statement

- The return value may be a value, a variable or a calculation
  - Type must match return type

Programming Tips

- Keep methods short
  - If more than one screen, break into ‘sub’ methods
- Trace your methods
  - One line for each step
  - Columns for key variables

intName(number = 416)

part	name
~~416~~	~~""~~
~~16~~	~~"four hundred"~~
0	"four hundred sixteen"

- Use Stubs as you write larger programs
  - Unfinished methods that return a ‘dummy’ value

```
public static String digitName(int digit)
{
  return "mumble";
}
```

Summary: Variables

- A variable is a storage location with a name.
- When declaring a variable, you usually specify an initial value.
- When declaring a variable, you also specify the type of its values.
- Use the `int` type for numbers that cannot have a fractional part.
- Use the `double` type for floating-point numbers.
- By convention, variable names should start with a lower case letter.
- An assignment statement stores a new value in a variable, replacing the previously stored value

Summary: Operators

- The assignment operator `=` does not denote mathematical equality.
- You cannot change the value of a variable that is defined as `final`.
- The `++` operator adds 1 to a variable; the `--` operator subtracts 1.
- If both arguments of `/` are integers, the remainder is discarded.
- The `%` operator computes the remainder of an integer division.
- The Java library declares many mathematical functions, such as `Math.sqrt` and `Math.pow`.
- You use a cast (*typeName*) to convert a value to a different type.
- Java classes are grouped into packages. Use the `import` statement to use classes from packages.

Summary: Methods

- A method is a named sequence of instructions.
- Arguments are supplied when a method is called. The return value is the result that the method computes.
- When declaring a method, you provide a name for the method, a variable for each argument, and a type for the result.
- Method comments explain the purpose of the method, the meaning of the parameters and return value, as well as any special requirements.
- Parameter variables hold the arguments supplied in the method call.

Summary: Method Returns

- The `return` statement terminates a method call and yields the method result.
  - Turn computations that can be reused into methods.
  - Use a return type of `void` to indicate that a method does not return a value.

Summary: `Strings`

- `Strings` are sequences of characters.
- The length method yields the number of characters in a `String`.
- Use the + operator to concatenate `Strings`; that is, to put them together to yield a longer `String`.
- Use the `next` (one word) or `nextLine` (entire line) methods of the `Scanner` class to read a `String`.
- Whenever one of the arguments of the + operator is a `String`, the other argument is converted to a `String`.
- If a `String` contains the digits of a number, you use the `Integer.parseInt` or `Double.parseDouble` method to obtain the number value.
- `String` index numbers are counted starting with 0.
- Use the `substring` method to extract a part of a `String`