Define a method vs. calling a method

```java
public class HelloPrinter {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

- Line 3 defines a method called `main`
- Line 5 calls a method called `println`, which is defined in the Java library
- You will learn later how to define your own method and to call that method.

## Chapter Goals

- To declare and initialize variables and constants
- To understand the properties and limitations of integers and floating-point numbers
- To appreciate the importance of comments and good code layout
- To write arithmetic expressions and assignment statements
- To create programs that read and process inputs, and display the results
- To learn how to use the Java `String` type

## Contents

- Variables
- Arithmetic
- Input and Output
- Problem Solving:
  - First Do It By Hand
- Strings

Numbers and character strings (such as the ones in this display board) are important data types in any Java program. In this chapter, you will learn how to work with numbers and text, and how to write simple programs that perform useful tasks with them.
2.1 Variables

- You may want to store values, e.g. \(3+4\), "Hello World"), in named storage locations for later use.
  - These locations are called variables.
- There are many different types (sizes) of storage to hold different kinds of values.
- 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

An Example: Soda Deal

- Soft drinks are sold in cans and bottles. A store offers a six-pack of 12-ounce cans for the same price as a two-liter bottle. Which should you buy? (12 fluid ounces equal approximately 0.355 liters.)

Variables: Types:

- Number of cans per pack (6) Whole number (integer)
- Ounces per can (12) Whole number (integer)
- Liters per bottle (0.355) Number with fraction (floating-point)

- A whole number (no fractional part) \(\text{int}\)
  - 6
- A number with a fraction part \(\text{double}\)
  - 12.0
- A word (a group of characters) \(\text{String}\)
  - "Hello World"
  - True or false \(\text{boolean}\)

Specify the type before the name in the declaration

\[
\begin{align*}
\text{int} & \text{ cansPerPack;} \\
\text{double} & \text{ canVolume;} \\
\text{String} & \text{ msg="Hello World";} \\
\text{boolean} & \text{ flag = true;} \\
\end{align*}
\]
Variables and contents

- You can (optionally) initialize the value of a variable when you declare it
  ```java
  int cansPerPack = 6;
  double canVolume = 12.0;
  ```
- Imagine a parking space in a parking garage
  - Identifier (name): J053
  - Contents (value): Bob’s Chevy

Why different types?

- Back to the garage analogy, parking spaces may be different sizes for different types of vehicles
  - Bicycle
  - Motorcycle
  - Full Size
  - Electric Vehicle

Number Literals in Java

- In Java, a number value is called number literal.
- 100,000 is not a number literal
- ½ is not a number literal

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>int</td>
<td>An integer has no fractional part.</td>
</tr>
<tr>
<td>-6</td>
<td>int</td>
<td>Integers can be negative.</td>
</tr>
<tr>
<td>0</td>
<td>int</td>
<td>Zero is an integer.</td>
</tr>
<tr>
<td>0.5</td>
<td>double</td>
<td>A number with a fractional part has type double.</td>
</tr>
<tr>
<td>1.8</td>
<td>double</td>
<td>An integer with a fractional part.0 has type double.</td>
</tr>
<tr>
<td>3.14</td>
<td>double</td>
<td>A number in exponential notation 3 x 10^2 or 100000. Numbers in exponential notation always have type double.</td>
</tr>
<tr>
<td>2.9e-2</td>
<td>double</td>
<td>Negative exponent: 2.96 x 10^-2 = 0.0296</td>
</tr>
<tr>
<td>339,000</td>
<td>int</td>
<td>Error: Do not use a comma as a decimal separation.</td>
</tr>
<tr>
<td>3.5</td>
<td>double</td>
<td>Error: Do not use fractions (use decimal notation): 3.5</td>
</tr>
</tbody>
</table>
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, not digits
     - 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

- Legal and illegal variable names

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>cansPerPack</td>
<td>Variable names consist of letters, numbers, and the underscore character.</td>
</tr>
<tr>
<td>x</td>
<td>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 39).</td>
</tr>
<tr>
<td>CanVolume</td>
<td>Variable names are case sensitive. This variable name is different from CansVolume, and it violates the convention that variable names should start with a lowercase letter.</td>
</tr>
<tr>
<td>CansVolume</td>
<td>Variable names cannot start with a number.</td>
</tr>
<tr>
<td>CanVolume</td>
<td>Error: Variable names cannot contain spaces.</td>
</tr>
<tr>
<td>doubleX</td>
<td>Error: You cannot use a reserved word as a variable name.</td>
</tr>
<tr>
<td>15/15</td>
<td>Error: You cannot use symbols such as / or .</td>
</tr>
</tbody>
</table>

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:
  - It copies the value on the right side into the variable on the left side
  - You will learn about the comparison operator in the next chapter

Updating a Variable

- Step by Step:
  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.
  ```java
  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?
    ```java
    double totalVolume = bottles * 2;
    ```
  - Also, if the constant is used in multiple places and needs to be changed, only the initialization changes.

Constant Declaration

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

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.
    total = bottles * BOTTLE_VOLUME;
```

- The name of a previously defined variable.

```
    This is an assignment.
    total = total + cans * CAN_VOLUME;
```

Java Comments

- There are three forms of comments:
  1: // single line (or rest of line to right)
  2: /* multi-line -- all comment until matching */
  3: /** multi-line Javadoc comments */

- Use comments at the beginning of each program, and to clarify details of the code.
Java Comment Example

```java
1 /**
2 * This program computes the volume (in liters) of a six-pack of soda cans and the total volume of a six-pack and a two-liter bottle.
3 */
4 public class Volume1
5 {
6     public static void main(String[] args)
7     {
9         int cansPerPack = 6;
10         final double canVolume = 12 * literPerOunce; // ??
11         double totalVolume = canVolume * cansPerPack * CAN_VOLUME;
12         System.out.println("A six-pack of 12-ounce cans contains ");
13         System.out.println(totalVolume);
14         System.out.println(" liters.");
15         final double BOTTLE_VOLUME = 2; // Two-liter bottle
17         double literPerOunce = 0.0296;
18     }
19     }
```

Lines 1 - 4 are Javadoc comments for the class Volume1

Lines 10 and 17 use single-line comment to clarify the unit of measurement

Common Error 2.1

- **Undeclared Variables**
  - You must declare a variable before you use it: (i.e. above in the code)
    ```java
double canVolume = 12 * literPerOunce; // ??
double literPerOunce = 0.0296;
```

- **Uninitialized Variables**
  - You must initialize (i.e. set) a variable’s contents before you use it
    ```java
    int bottles;
    int bottleVolume = bottles * 2; // ??
    ```

Value Ranges per Type

- **Integer Types**
  - **byte** (1 byte): A very small number (-128 to +127)
  - **short** (2 bytes): A small number (-32768 to +32767)
  - **int** (4 bytes): A large number (-2,147,483,648 to +2,147,483,647)
  - **long** (8 bytes): A huge number

- **Floating Point Types**
  - **float** (4 bytes): A huge number with decimal places
  - **double** (8 bytes): Much more precise, for heavy math

- **Other Types**
  - **boolean** (1 bit): true or false
  - **char** (2 bytes): 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**: 

### Common Error 2.2

- **Overflow** means that storage for a variable cannot hold the result:
  ```java
  int fiftyMillion = 5000000;
  System.out.println(100 * fiftyMillion);
  // Expected: 500000000
  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 limitation of binary values (no fractions):
      ```java
      double price = 4.35;
      double quantity = 100;
      double total = price * quantity;
      // Should be 100 * 4.35 = 435.0
      System.out.println(total); // Prints 434.9999999999999
      **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).)**

---

### 2.2 Arithmetic

- Java supports all of the same basic math as a calculator:
  - Addition `+`
  - Subtraction `-`
  - Multiplication `*`
  - Division `/`
- You write your expressions a bit differently though: Algebra Java
  ```math
  \frac{a + b}{2} (a + b) / 2
  ```
- Precedence is similar to Algebra:
  - PEMDAS
    - Parenthesis, Exponent, Multiply/Divide, Add/Subtract

---

### Converting 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:
  ```java
  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.**
  ```
Floating-Point to Integer Conversion

- The Java compiler does not allow direct assignment of a floating-point value to an integer variable
  ```java
  double balance = total + tax;
  int dollars = balance; // Error
  ```
- You can use the ‘cast’ operator: `(int)` to force the conversion:
  ```java
  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
  ```java
  long rounded = Math.round(balance);
  ```

Incrementing a Variable

- Step by Step:
  1. Do the right hand side of the assignment first:
     Find the value stored in counter, and add 1 to it
  2. Store the result in the variable named on the left side of the assignment operator (counter in this case)

Shorthand for Incrementing

- Incrementing (+1) and decrementing (-1) integer types is so common that there are shorthand version for each

<table>
<thead>
<tr>
<th>Long Way</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>counter = counter + 1;</code></td>
<td><code>counter++</code> ; <code>++counter;</code></td>
</tr>
<tr>
<td><code>counter = counter - 1;</code></td>
<td><code>counter--;</code> ; <code>--counter;</code></td>
</tr>
</tbody>
</table>
## Integer Division and Remainder

- When both parts of division are integers, the result is an integer.
  - All fractional information is lost (no rounding)
  - The value of result will be 1

- 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

## Integer Division and Remainder Examples

- If one of the value is a floating-point number, the result is a floating-point number:
  - `7.2 / 4` yields `1.8`
  - `7.2 % 4` yields `3.2`

## Powers and Roots

- In Java, the `Math` class contains methods `sqrt` and `pow` to compute power and square roots:
  - `(b x (1 + r / 100))^n` becomes: `b * Math.pow(1 + r / 100, n)`

## Mathematical Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.sqrt(x)</td>
<td><code>\sqrt{x}</code> (x ≥ 0)</td>
</tr>
<tr>
<td>Math.pow(x, y)</td>
<td><code>x^y</code> (x &gt; 0, or x = 0 and y ≥ 0, or x &lt; 0 and y is an integer)</td>
</tr>
<tr>
<td>Math.sin(x)</td>
<td>Sine of x (x in radians)</td>
</tr>
<tr>
<td>Math.cos(x)</td>
<td>Cosine of x</td>
</tr>
<tr>
<td>Math.tan(x)</td>
<td>Tangent of x</td>
</tr>
<tr>
<td>Math.toRadians(x)</td>
<td>Convert x degrees to radians (i.e., returns x · π/180)</td>
</tr>
<tr>
<td>Math.toDegrees(x)</td>
<td>Convert x radians to degrees (i.e., returns x · 180/π)</td>
</tr>
<tr>
<td>Math.exp(x)</td>
<td><code>e^x</code></td>
</tr>
<tr>
<td>Math.log(x)</td>
<td>Natural log (ln(x), x &gt; 0)</td>
</tr>
</tbody>
</table>
Common Error 2.4

- **Unintended Integer Division**
  - `int s1 = 5;`  
  - `int s2 = 3;`  
  - `int s3 = 2;`  
  - `double average = (s1 + s2 + s3) / 3; // ?`

**Why?**
- All of the calculation on the right happens first
  - Since all are `int`s, 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 `(0.0)` 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)) // (1, 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`

Increment and Decrement

- **Increment/decrement before variable:** the value of the variable is increased/decreased by 1 **before** it is used:
  - `int s = 5;`  
  - `System.out.println(++s); // ?`  
  - `System.out.println(s); // ?`
  - `System.out.println(s--); // ?`

- **Increment/decrement after variable:** the value of the variable is increased/decreased by 1 **after** it is used:
  - `int s = 5;`  
  - `System.out.println(s++); // ?`  
  - `System.out.println(s); // ?`
  - `System.out.println(s--); // ?`
Variables vs. Constants

- Their declarations and usages are similar but with difference:
  - `final double CAN_VOLUME;
  int cans;
  CAN_VOLUME = 0.355; // liters in a 12-ounce can
  cans = 1;
  System.out.println(CAN_VOLUME * cans); // ?
  cans = 2;
  System.out.println(CAN_VOLUME * cans); // ?`

2.3 Input and Output

Reading Input

- You might need to ask for input (aka prompt for input) and then save what was entered in a variable.
- This is a three step process in Java
  1) Import the Scanner class from its ‘package’
     ```java
     java.util import java.util.Scanner;
     ```
  2) Setup an object of the Scanner class
     ```java
     Scanner in = new Scanner(System.in);
     ```
  3) Use methods of the created Scanner object to get input
     ```java
     int bottles = in.nextInt();
     double price = in.nextDouble();
     ```

Syntax 2.3: Input Statement

- The `Scanner` class allows you to read keyboard input from the user
  - It is part of the Java API util package
    ```java
    import java.util.Scanner;
    Scanner in = new Scanner(System.in);
    System.out.println("Please enter the number of bottles: ");
    int bottles = in.nextInt();
    System.out.println(CAN_VOLUME * bottles);
    System.out.println("The program waits for your input,
    then places the input into the variable.
    
    Copyright © 2011 by John Wiley & Sons. All rights reserved.
    ```

Formatted Output

- Output String:
  ```java
  System.out.println("Hello World");
  ```
- We can control the output appearance of numeric variables using format specifier with the `printf` method:
  ```java
  System.out.printf("%.2f", price); // 1.22
  System.out.printf("%10.2f", price); // 1.22
  ```
- No new line at the end of the line.
Format Types

- Format specification and variable value type have to match. Otherwise, there is a run-time error.

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Decimal integer</td>
<td>123</td>
</tr>
<tr>
<td>f</td>
<td>Fixed floating-point</td>
<td>12.30</td>
</tr>
<tr>
<td>e</td>
<td>Exponential floating-point</td>
<td>1.23e+1</td>
</tr>
<tr>
<td>g</td>
<td>General floating-point</td>
<td>12.33</td>
</tr>
<tr>
<td>s</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(exponential notation is used for very large or very small values)</td>
<td></td>
</tr>
</tbody>
</table>

- You can also include text inside the quotes:

```
System.out.printf("Price per liter: %10.2f", price);
```

**Volume2.java**

```java
import java.util.Scanner;

public class Volume2 {
    public static void main(String[] args) {
        // Read price per pack
        Scanner in = new Scanner(System.in);
        System.out.print("Please enter the price for a six-pack: ");
        double packPrice = in.nextDouble();
        // Read can volume
        System.out.print("Please enter the volume for each can (in ounces): ");
        double canVolume = in.nextDouble();
        // Compute and print price per ounce
        double pricePerOunce = packPrice / canVolume;
        System.out.printf("Price per ounce: %8.2f, pricePerOunce = %8.2f", pricePerOunce);
        System.out.println();
    }
}
```

```
$ java Volume2
Please enter the price for a six-pack: 5.4
Please enter the volume for each can (in ounces): 0.355
Price per ounce: 2.54
$  
```

**Tip 2.2 Java API Documentation**

- Lists the classes and methods of the Java API
2.5 Strings and Characters

- A **string** is a sequence of **characters**
  - Letters: a, A;
  - Numbers: 3, 1;
  - Punctuation: _, -;
  - Space:

- Characters have their own type: **char**

- Use single quotes around a char
  ```java
  char initial = 'B';
  ```

- Use double quotes around a String
  ```java
  String initials = "BRL";
  ```

2.5 Strings

- The String Type:
  ```java
  Type Variable Literal
  String name = "Harry"
  ```

- Once you have a String variable, you can use various methods to process its contents. For example:
  ```java
  int n = name.length(); // n ?
  ```

- A String’s length is the number of characters inside:
  ```java
  int n = "".length(); // n ?
  ```

String Concatenation (+)

- You can concatenate one String onto the end of another String:
  ```java
  String fName = "Harry"
  String lName = "Morgan"
  String name = fName + lName; // HarryMorgan
  ```

- You wanted a space in between?
  ```java
  String name = fName + " " + lName; // Harry Morgan
  ```

- Concatenate Strings and numbers inside `println`:
  ```java
  System.out.println("3" + 4); // 34
  System.out.println("3" + "4"); // 34
  System.out.println(3 + 4); // 7
  ```

String Input

- You can read a String from the console with:
  ```java
  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
  - What is the value of name if the input given is “John Doe”?

- You can read an entire line from the console with:
  ```java
  System.out.print("Please enter your phone number: ");
  String phone = in.nextLine();
  ```
  - The `nextLine` method reads until the user hits ‘Enter’
  - What is the value of name if the input given is “864 6943”?
Print Special Character using \\

- How would you print a double quote?
  - Preface the " with a \ inside the double quoted String
  ```java
  System.out.print("He said "Hello\"");
  ```

- OK, then how do you print a backslash?
  - Preface the \ with another \!
  ```java
  System.out.println("C:\\Temp\\Secret.txt");
  ```

- Special characters inside Strings
  - Output a newline with a \n
  ```java
  System.out.print("*\n**\n***\n");
  ```

Get a char from a String

- Each char inside a String has an index number:
  - The first char is index zero (0)

- The charAt method returns a char at a given index inside a String:
  ```java
  String greeting = "Harry";
  char start = greeting.charAt(0);
  char last = greeting.charAt(4);
  ```

<table>
<thead>
<tr>
<th>char</th>
<th>index</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>r</td>
<td>2</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 9: String Operations (1)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>str = &quot;Hi&quot;;</td>
<td>str is set to &quot;Hi&quot;</td>
<td>When applied to strings, + denotes concatenation.</td>
</tr>
<tr>
<td>System.out.println(&quot;Please enter your name:&quot;);</td>
<td>Prints</td>
<td>Use concatenation to break up strings that don't fit into one line.</td>
</tr>
<tr>
<td>team = str + &quot;are&quot;;</td>
<td>team is set to &quot;Harry&quot;</td>
<td>Because &quot;erv&quot; is a string, + is converted to a string.</td>
</tr>
<tr>
<td>String first = new String();</td>
<td>first contains &quot;Harry&quot;</td>
<td>The next method places the next word into the string variable.</td>
</tr>
<tr>
<td>String last = new String();</td>
<td>last contains &quot;Morgan&quot;</td>
<td></td>
</tr>
</tbody>
</table>
| String greeting = "He\t
  said Hello\n"; | x is set to 5 | Each space counts as one character |
| String str = "Hello"; | x is set to 0 | |
| char ch = str.charAt(4); | 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)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{String \textit{str} = &quot;Sally&quot;;} \texttt{String \textit{str2} = \textit{str}.substring(3, 4);}</td>
<td>\texttt{str2} is set to &quot;al&quot;</td>
<td>Extracts the substring starting at position 1 and ending before position 6.</td>
</tr>
<tr>
<td>\texttt{String \textit{str} = &quot;Sally&quot;;} \texttt{String \textit{str2} = \textit{str}.substring(1);}</td>
<td>\texttt{str2} is set to &quot;al\textit{ly}&quot;</td>
<td>If you omit the end position, all characters from the position until the end of the string are included.</td>
</tr>
<tr>
<td>\texttt{String \textit{str} = &quot;Sally&quot;;} \texttt{String \textit{str2} = \textit{str}.substring(1, 2);}</td>
<td>\texttt{str2} is set to &quot;a&quot;</td>
<td>Extracts a String of length 1; contrast with \texttt{\textit{str}.charAt(1)}.</td>
</tr>
<tr>
<td>\texttt{String \textit{last} = \textit{str}.substring( \textit{str}.length() - 3);}</td>
<td>\texttt{last} is set to the string containing the last character in \textit{str}</td>
<td>The last character has position \textit{str}.length()-3.</td>
</tr>
</tbody>
</table>

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 \texttt{int} type for numbers that cannot have a fractional part.
- Use the \texttt{double} type for floating-point numbers.
- By convention, variable names should start with a lower case letter.
- You use a cast (\texttt{typeName}) to convert a value to a different type.

Summary: Operators

- The assignment operator = does not denote mathematical equality.
- You can assign and reassign a new value to a variable, replacing the previously stored value.
- You cannot change the value of a variable that is defined as \texttt{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.

Summary: Java API

- The Java library defines many mathematical functions, such as \texttt{Math.sqrt} and \texttt{Math.pow}.
- Java classes are grouped into packages. Use the \texttt{import} statement to use classes from packages.
- Use the \texttt{Scanner} class to read keyboard input in a console window.
- Use the \texttt{printf} method to specify how values should be formatted.
- The API (Application Programming Interface) documentation lists the classes and methods of the Java library.
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.
- String index numbers are counted starting with 0.
- Use the `charAt` method to extract a character of a String
- Use the `substring` method to extract a part of a String