

Chapter 2 Elementary Programming



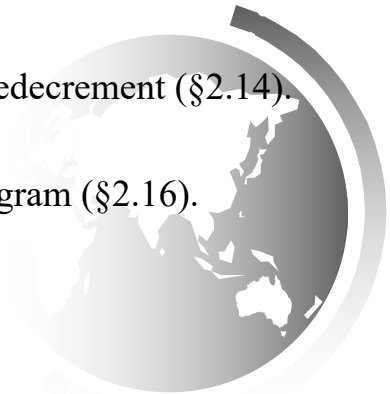
Motivations

In the preceding chapter, you learned how to create, compile, and run a Java program. Starting from this chapter, you will learn how to solve practical problems programmatically. Through these problems, you will learn Java primitive data types and related subjects, such as variables, constants, data types, operators, expressions, and input and output.



Objectives

- ✦ To write Java programs to perform simple computations (§2.2).
- ✦ To obtain input from the console using the **Scanner** class (§2.3).
- ✦ To use identifiers to name variables, constants, methods, and classes (§2.4).
- ✦ To use variables to store data (§§2.5–2.6).
- ✦ To program with assignment statements and assignment expressions (§2.6).
- ✦ To use constants to store permanent data (§2.7).
- ✦ To name classes, methods, variables, and constants by following their naming conventions (§2.8).
- ✦ To explore Java numeric primitive data types: **byte**, **short**, **int**, **long**, **float**, and **double** (§2.9.1).
- ✦ To read a **byte**, **short**, **int**, **long**, **float**, or **double** value from the keyboard (§2.9.2).
- ✦ To perform operations using operators **+**, **-**, *****, **/**, and **%** (§2.9.3).
- ✦ To perform exponent operations using **Math.pow(a, b)** (§2.9.4).
- ✦ To write integer literals, floating-point literals, and literals in scientific notation (§2.10).
- ✦ To write and evaluate numeric expressions (§2.11).
- ✦ To obtain the current system time using **System.currentTimeMillis()** (§2.12).
- ✦ To use augmented assignment operators (§2.13).
- ✦ To distinguish between postincrement and preincrement and between postdecrement and predecrement (§2.14).
- ✦ To cast the value of one type to another type (§2.15).
- ✦ To describe the software development process and apply it to develop the loan payment program (§2.16).
- ✦ To write a program that converts a large amount of money into smaller units (§2.17).
- ✦ To avoid common errors and pitfalls in elementary programming (§2.18).



Introducing Programming with an Example

Listing 2.1 Computing the Area of a Circle

This program computes the area of the circle.



ComputeArea

Animation

Run



Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```

radius

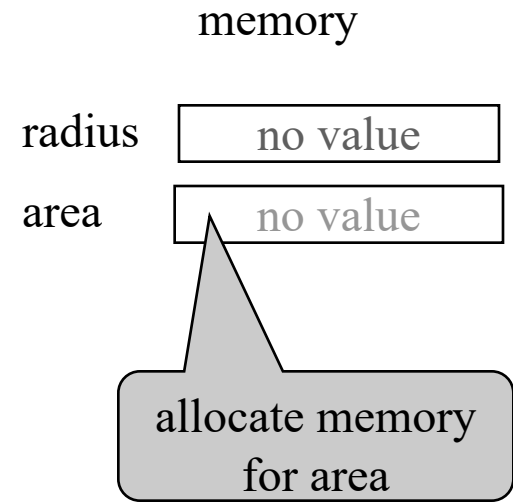
allocate memory
for radius

no value



Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```



Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```

radius

area

assign 20 to radius

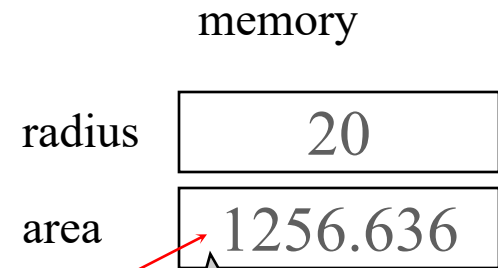
20

no value



Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```



compute area and assign it to variable area



Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```

memory

radius

20

area

1256.636

print a message to the console



```
CA Command Prompt  
c:\book>java ComputeArea  
The area for the circle of radius 20.0 is 1256.636
```

Identifiers

- ◆ An identifier is a sequence of characters that consist of letters, digits, underscores (`_`), and dollar signs (`$`).
- ◆ An identifier must start with a letter, an underscore (`_`), or a dollar sign (`$`). It cannot start with a digit.
- ◆ An identifier cannot be a reserved word. (See Appendix A, “Java Keywords,” for a list of reserved words).
- ◆ An identifier cannot be `true`, `false`, or `null`.
- ◆ An identifier can be of any length.



Variables

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is " +
    area + " for radius "+radius);
```

```
// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is " +
    area + " for radius "+radius);
```



Declaring Variables

```
int x;           // Declare x to be an
                 // integer variable;

double radius;  // Declare radius to
                 // be a double variable;

char a;         // Declare a to be a
                 // character variable;
```



Assignment Statements

```
x = 1;           // Assign 1 to x;  
radius = 1.0;   // Assign 1.0 to radius;  
a = 'A';        // Assign 'A' to a;
```



Declaring and Initializing in One Step

◆ `int x = 1;`

◆ `double d = 1.4;`



Named Constants

```
final datatype CONSTANTNAME = VALUE;
```

```
final double PI = 3.14159;
```

```
final int SIZE = 3;
```



Naming Conventions

- ◆ Choose meaningful and descriptive names.
- ◆ Variables and method names:
 - Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name. For example, the variables `radius` and `area`, and the method `computeArea`.



Naming Conventions, cont.

◆ Class names:

- Capitalize the first letter of each word in the name. For example, the class name `ComputeArea`.

◆ Constants:

- Capitalize all letters in constants, and use underscores to connect words. For example, the constant `PI` and `MAX_VALUE`



Numerical Data Types

| Name | Range | Storage Size |
|---------------------|--|-----------------|
| <code>byte</code> | -2^7 to $2^7 - 1$ (-128 to 127) | 8-bit signed |
| <code>short</code> | -2^{15} to $2^{15} - 1$ (-32768 to 32767) | 16-bit signed |
| <code>int</code> | -2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647) | 32-bit signed |
| <code>long</code> | -2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807) | 64-bit signed |
| <code>float</code> | Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38 | 32-bit IEEE 754 |
| <code>double</code> | Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308 | 64-bit IEEE 754 |



Numeric Operators

| Name | Meaning | Example | Result |
|------|----------------|------------|--------|
| + | Addition | 34 + 1 | 35 |
| - | Subtraction | 34.0 - 0.1 | 33.9 |
| * | Multiplication | 300 * 30 | 9000 |
| / | Division | 1.0 / 2.0 | 0.5 |
| % | Remainder | 20 % 3 | 2 |



Integer Division

- ◆ $5 / 2$ yields an integer 2.
- ◆ $5.0 / 2$ yields a double value 2.5
- ◆ $5 \% 2$ yields 1 (the remainder of the division)

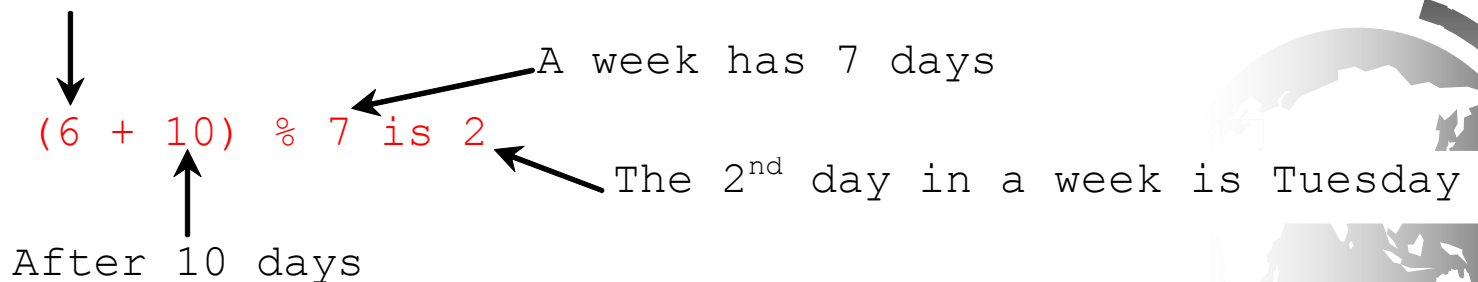
- ◆ `System.out.println(5.0/2);` //output 2.5
- ◆ `System.out.println(5/2.0);` //output 2.5
- ◆ `System.out.println(5/2);` //output 2



Remainder Operator

Remainder is very useful in programming. For example, an even number $\% 2$ is always 0 and an odd number $\% 2$ is always 1. So you can use this property to determine whether a number is even or odd. Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:

Saturday is the 6th day in a week



Exponent Operations

```
System.out.println(Math.pow(2, 3));
```

```
// Displays 8.0
```

```
System.out.println(Math.pow(4, 0.5));
```

```
// Displays 2.0
```

```
System.out.println(Math.pow(2.5, 2));
```

```
// Displays 6.25
```

```
System.out.println(Math.pow(2.5, -2));
```

```
// Displays 0.16
```



Number Literals

A *literal* is a constant value that appears directly in the program. For example, 34, 1,000,000, and 5.0 are literals in the following statements:

```
int i = 34;
```

```
long x = 1000000;
```

```
double d = 5.0;
```



Integer Literals

An integer literal can be assigned to an integer variable as long as it can fit into the variable. A compilation error would occur if the literal were too large for the variable to hold. For example, the statement `byte b = 1000` would cause a compilation error, because 1000 cannot be stored in a variable of the byte type.

An integer literal is assumed to be of the `int` type, whose value is between -2^{31} (-2147483648) to $2^{31}-1$ (2147483647). To denote an integer literal of the long type, append it with the letter `L` or `l`. `L` is preferred because `l` (lowercase `L`) can easily be confused with `1` (the digit one).

Floating-Point Literals

Floating-point literals are written with a decimal point. By default, a floating-point literal is treated as a double type value. For example, 5.0 is considered a double value, not a float value. You can make a number a float by appending the letter f or F, and make a number a double by appending the letter d or D. For example, you can use 100.2f or 100.2F for a float number, and 100.2d or 100.2D for a double number.

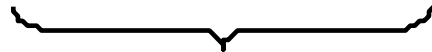


double vs. float

The double type values are more accurate than the float type values. For example,

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

displays `1.0 / 3.0 is 0.3333333333333333`



16 digits

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

displays `1.0F / 3.0F is 0.33333334`



7 digits



Scientific Notation

Floating-point literals can also be specified in scientific notation, for example, $1.23456e+2$, same as $1.23456e2$, is equivalent to 123.456 , and $1.23456e-2$ is equivalent to 0.0123456 . E (or e) represents an exponent and it can be either in lowercase or uppercase.



Arithmetic Expressions

$$\frac{3 + 4x}{5} - \frac{10(y - 5)(a + b + c)}{x} + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)$$

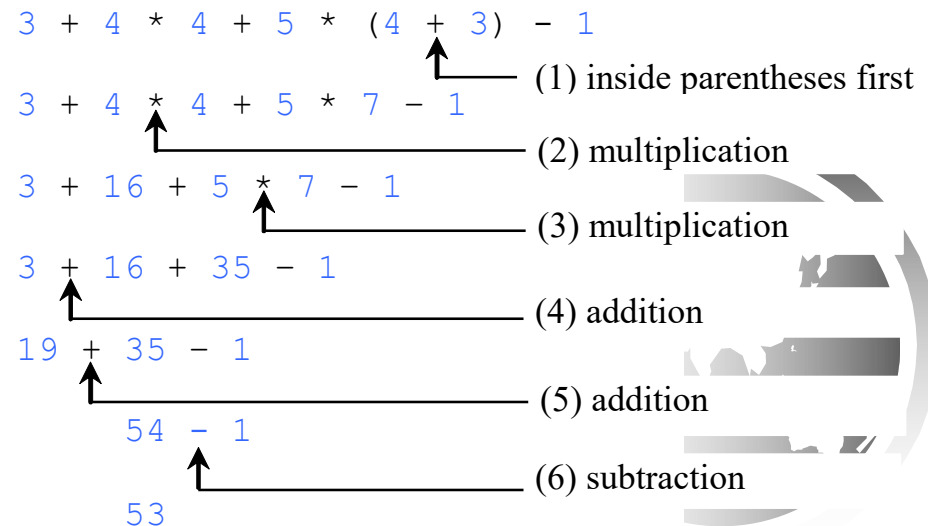
is translated to

$$(3+4*x)/5 - 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)$$



How to Evaluate an Expression

Though Java has its own way to evaluate an expression behind the scene, the result of a Java expression and its corresponding arithmetic expression are the same. Therefore, you can safely apply the arithmetic rule for evaluating a Java expression.



Numeric Type Conversion

Consider the following statements:

```
byte i = 100;
```

```
long k = i * 3 + 4;
```

```
double d = i * 3.1 + k / 2;
```



Conversion Rules

When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1. If one of the operands is double, the other is converted into double.
2. Otherwise, if one of the operands is float, the other is converted into float.
3. Otherwise, if one of the operands is long, the other is converted into long.
4. Otherwise, both operands are converted into int.



Type Casting

Implicit casting

```
double d = 3; (type widening)
```

Explicit casting

```
int i = (int)3.0; (type narrowing)
```

```
int i = (int)3.9; (Fraction part is truncated)
```

What is wrong? `int x = 5 / 2.0;`

range increases



byte, short, int, long, float, double



Augmented Assignment Operators

| <i>Operator</i> | <i>Name</i> | <i>Example</i> | <i>Equivalent</i> |
|-----------------|---------------------------|---------------------|------------------------|
| <code>+=</code> | Addition assignment | <code>i += 8</code> | <code>i = i + 8</code> |
| <code>-=</code> | Subtraction assignment | <code>i -= 8</code> | <code>i = i - 8</code> |
| <code>*=</code> | Multiplication assignment | <code>i *= 8</code> | <code>i = i * 8</code> |
| <code>/=</code> | Division assignment | <code>i /= 8</code> | <code>i = i / 8</code> |
| <code>%=</code> | Remainder assignment | <code>i %= 8</code> | <code>i = i % 8</code> |



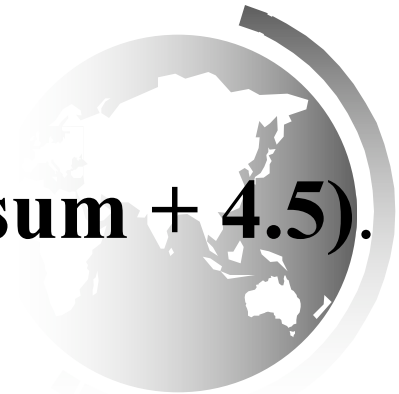
Casting in an Augmented Expression

In Java, an augmented expression of the form **x1 op= x2** is implemented as **x1 = (T)(x1 op x2)**, where **T** is the type for **x1**. Therefore, the following code is correct.

```
int sum = 0;
```

```
sum += 4.5; // sum becomes 4 after this statement
```

```
sum += 4.5 is equivalent to sum = (int)(sum + 4.5).
```



Reading Input from the Console

1. Create a Scanner object

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

2. Use the method `nextDouble()` to obtain to a double value. For example,

```
System.out.print("Enter a double value: ");  
Scanner input = new Scanner(System.in);  
double d = input.nextDouble();
```

Animation

 ComputeAreaWithConsoleInput

Run

 ComputeAverage

Run

Reading Numbers from the Keyboard

```
Scanner input = new Scanner(System.in) ;  
int value = input.nextInt() ;
```

| Method | Description |
|---------------------------|--|
| <code>nextByte()</code> | reads an integer of the <code>byte</code> type. |
| <code>nextShort()</code> | reads an integer of the <code>short</code> type. |
| <code>nextInt()</code> | reads an integer of the <code>int</code> type. |
| <code>nextLong()</code> | reads an integer of the <code>long</code> type. |
| <code>nextFloat()</code> | reads a number of the <code>float</code> type. |
| <code>nextDouble()</code> | reads a number of the <code>double</code> type. |

ComputeAreaWithConsoleInput

```
import java.util.Scanner; // Scanner is in the java.util package
```

```
public class ComputeAreaWithConsoleInput {  
    public static void main(String[] args) {  
        // Create a Scanner object  
        Scanner input = new Scanner(System.in);  
  
        // Prompt the user to enter a radius  
        System.out.print("Enter a number for radius: ");  
        double radius = input.nextDouble();  
  
        // Compute area  
        double area = radius * radius * 3.14159;  
  
        // Display result  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```



- ◆ Write a program that takes three numbers from the user, then computes and print their average.



Compute Average

```
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAverage {
    public static void main(String[] args) {
        // Create a Scanner object
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter three numbers
        System.out.print("Enter three numbers: ");
        double number1 = input.nextDouble();
        double number2 = input.nextDouble();
        double number3 = input.nextDouble();

        // Compute average
        double average = (number1 + number2 + number3) / 3;

        // Display result
        System.out.println("The average of " + number1 + " " + number2
            + " " + number3 + " is " + average);
    }
}
```



Problem: Converting Temperatures

Write a program that converts a Fahrenheit degree to Celsius using the formula:

$$celsius = \left(\frac{5}{9}\right)(fahrenheit - 32)$$

Note: you have to write

$$celsius = (5.0 / 9) * (fahrenheit - 32)$$



FahrenheitToCelsius

Run



Increment and Decrement Operators

| <i>Operator</i> | <i>Name</i> | <i>Description</i> | <i>Example (assume i = 1)</i> |
|-----------------|---------------|---|---|
| ++var | preincrement | Increment var by 1 , and use the new var value in the statement | <pre>int j = ++i; // j is 2, i is 2</pre> |
| var++ | postincrement | Increment var by 1 , but use the original var value in the statement | <pre>int j = i++; // j is 1, i is 2</pre> |
| --var | predecrement | Decrement var by 1 , and use the new var value in the statement | <pre>int j = --i; // j is 0, i is 0</pre> |
| var-- | postdecrement | Decrement var by 1 , and use the original var value in the statement | <pre>int j = i--; // j is 1, i is 0</pre> |



Increment and Decrement Operators, cont.

```
int i = 10;
```

```
int newNum = 10 * i++;
```

Same effect as

```
int newNum = 10 * i;  
i = i + 1;
```

```
int i = 10;
```

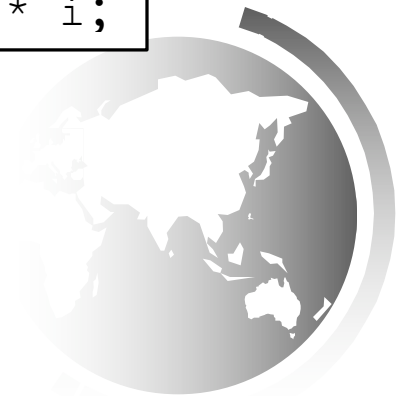
```
int newNum = 10 * (++i);
```

Same effect as

```
i = i + 1;  
int newNum = 10 * i;
```

i++ → Assign → increment

++i → increment → Assign



Increment and Decrement Operators, cont.

Using increment and decrement operators makes expressions short, but it also makes them complex and difficult to read. Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times such as this: `int k = ++i + i.`



Assignment Expressions and Assignment Statements

Prior to Java 2, all the expressions can be used as statements. Since Java 2, only the following types of expressions can be statements:

`variable op= expression; // Where op is +, -, *, /, or %`

`++variable;`

`variable++;`

`--variable;`

`variable--;`



What is the output of the following code:

```
int i = 5;
int j = 5;
System.out.println("i = "+ i);
System.out.println("j = "+ j);
System.out.println("-----");
int k = i++;
System.out.println("k = "+ k);
System.out.println("i = "+ i);
System.out.println("-----");
int x = ++j;
System.out.println("x = "+ x);
System.out.println("j = "+ j);
```



What is the output of the following code:

```
int i = 5;
```

```
int j = 5;
```

```
System.out.println(i++);
```

```
System.out.println(++j);
```

```
System.out.println(i);
```

```
System.out.println(j);
```



Problem: Keeping Two Digits After Decimal Points

Write a program that displays the sales tax with two digits after the decimal point.



SalesTax

Run

```
double purchaseAmount = 90.75;
double tax = purchaseAmount * 0.06;
// System.out.println("Sales tax is " + tax);
System.out.println("Sales tax is " + (int)(tax * 100) / 100.0 );
// Above expression is used instead of tax to show type casting
// first 100 will be convert to double to be multiplied by tax implicitly
// then result of that will be converted to int explicitly
// at last int dividend received by explicit type conversion will be
// converted to double implicitly
```



Common Errors and Pitfalls

- ◆ Common Error 1: Undeclared/Uninitialized Variables and Unused Variables
- ◆ Common Error 2: Integer Overflow
- ◆ Common Error 3: Round-off Errors
- ◆ Common Error 4: Unintended Integer Division
- ◆ Common Error 5: Redundant Input Objects



Common Error 1: Undeclared/Uninitialized Variables and Unused Variables

```
double interestRate = 0.05;
```

```
double interest = interestrate * 45;
```



Common Error 2: Integer Overflow

An **integer overflow error** occurs when an operation makes an **integer** value greater than its maximum. For example, if the maximum value you can have is 100000, and your current value is 99999, then adding **2** will make it '**overflow**'



Common Error 3: Round-off Errors

Rounding multiple times can cause **error** to accumulate. For example, if 9.945309 is **rounded** to two decimal places (9.95), then **rounded** again to one decimal place (10.0), the total **error** is 0.054691

```
System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
```

```
System.out.println(1.0 - 0.9);
```



Common Error 4: Unintended Integer Division

```
int number1 = 1;  
int number2 = 2;  
double average = (number1 + number2) / 2;  
System.out.println(average);
```

(a)

```
int number1 = 1;  
int number2 = 2;  
double average = (number1 + number2) / 2.0;  
System.out.println(average);
```

(b)



Common Pitfall 1: Redundant Input Objects

```
Scanner input = new Scanner(System.in);  
System.out.print("Enter an integer: ");  
int v1 = input.nextInt();
```

```
Scanner input1 = new Scanner(System.in);  
System.out.print("Enter a double value: ");  
double v2 = input1.nextDouble();
```

