1a

Introduction to software development with Java

OBJECTIVES

In this lecture you will learn:

- To write simple Java applications.
- To use input and output statements.
- Java’s primitive types.
- Basic memory concepts.
- To use arithmetic operators.
- The precedence of arithmetic operators.
- To write decision-making statements.
- To use relational and equality operators.

Outline

1.13 Typical Java Development Environment

Java programs go through five phases

- **Edit**
  - **Programmer writes program using an editor; stores program on disk with the `.java` file name extension**
- **Compile**
  - **Use javac (the Java compiler) to create bytecodes from source code program; bytecodes stored in `.class` files**
- **Load**
  - **Class loader reads bytecodes from `.class` files into memory**
- **Verify**
  - **Bytecode verifier examines bytecodes to ensure that they are valid and do not violate security restrictions**
- **Execute**
  - **Java Virtual Machine (JVM) uses a combination of interpretation and just-in-time compilation to translate bytecodes into machine language**

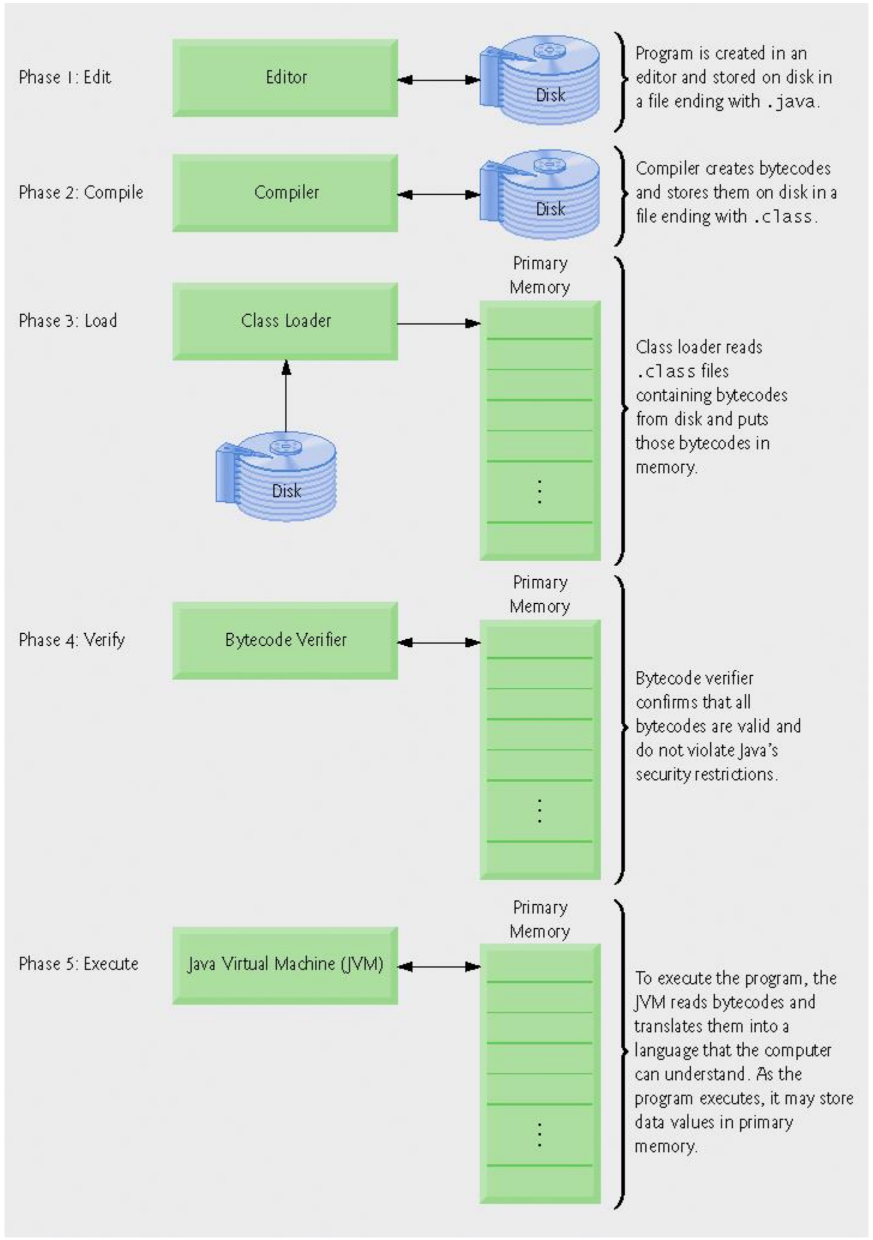

Phase 1: Edit
Editor
Disk
Program is created in an editor and stored on disk in a file ending with .java.
Phase 2: Compile
Compiler
Disk
Compiler creates bytecodes and stores them on disk in a file ending with .class.
Phase 3: Load
Class Loader
Disk
Primary Memory
Class loader reads .class files containing bytecodes from disk and puts those bytecodes in memory.
Phase 4: Verify
Bytecode Verifier
Primary Memory
Bytecode verifier confirms that all bytecodes are valid and do not violate Java's security restrictions.
Phase 5: Execute
Java Virtual Machine (JVM)
Primary Memory
To execute the program, the JVM reads bytecodes and translates them into a language that the computer can understand. As the program executes, it may store data values in primary memory.

Fig. 1.1 | **Typical Java development environment.**

Common Programming Error 1.1

Errors like division by zero occur as a program runs, so they are called *runtime errors* or *execution-time errors*. *Fatal runtime errors* cause programs to terminate immediately without having successfully performed their jobs. *Nonfatal runtime errors* allow programs to run to completion, often producing incorrect results.

Good Programming Practice 1.1

Write your Java programs in a simple and straightforward manner. This is sometimes referred to as KIS (“keep it simple”). Do not “stretch” the language by trying bizarre usages.

2.1 Introduction

Java application programming

- **Display messages**
- **Obtain information from the user**
- **Arithmetic calculations**
- **Decision-making fundamentals**

2.2 First Program in Java: Printing a Line of Text

- **Application**
  - **Executes when you use the `java` command to launch the Java Virtual Machine (JVM)**
- **Sample program**
  - **Displays a line of text**
  - **Illustrates several important Java language features**

Outline

```
// Fig. 2.1: Welcome1.java
// Text-printing program.

public class Welcome1
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.println( "Welcome to Java Programming!" );

   } // end method main

} // end class Welcome1
```

`Welcome1.java`

```
Welcome to Java Programming!
```

A *string literal* is represented by putting **double quotes** around the text

Examples:

```
"This is a string literal."
"123 Main Street"
"X"
```

Every character string is an object in Java, defined by the `String` class

Every string literal represents a `String` object

2.2 First Program in Java: Printing a Line of Text (Cont.)

```
1  // Fig. 2.1: Welcome1.java
```

- Comments start with: `//`
  - Comments ignored during program execution
  - Document and describe code
  - Provides code readability
- Traditional comments: `/* ... */`

```
/* This is a traditional
   comment. It can be
   split over many lines */
```

```
2  // Text-printing program.
```

- Another line of comments
- Note: line numbers not part of program, added for reference

Common Programming Error 2.1

Forgetting one of the delimiters of a traditional or Javadoc comment is a syntax error. The *syntax* of a programming language specifies the rules for creating a proper program in that language. A *syntax error* occurs when the compiler encounters code that violates Java's language rules (i.e., its syntax). In this case, the compiler does not produce a `.class` file. Instead, the compiler issues an error message to help the programmer identify and fix the incorrect code. Syntax errors are also called *compiler errors*, *compile-time errors* or *compilation errors*, because the compiler detects them during the compilation phase. You will be unable to execute your program until you correct all the syntax errors in it.

Good Programming Practice 2.1

Every program should begin with a comment that explains the purpose of the program, the author and the date and time the program was last modified. (We are not showing the author, date and time in this book's programs because this information would be redundant.)

2.2 First Program in Java: Printing a Line of Text (Cont.)

```
3
```

- Blank line
  - Makes program more readable
  - Blank lines, spaces, and tabs are white-space characters
    - Ignored by compiler

```
4   public class Welcome1
```

- Begins class declaration for class `Welcome1`
  - Every Java program has at least one user-defined class
  - Keyword: words reserved for use by Java
    - `class` keyword followed by class name
  - Naming classes: capitalize every word
    - `SampleClassName`

Good Programming Practice 2.2

Use blank lines and space characters to enhance program readability.

2.2 First Program in Java: Printing a Line of Text (Cont.)

```
4   public class Welcome1
```

- Java identifier
    - Series of characters consisting of letters, digits, underscores (_) and dollar signs ($)
    - Does not begin with a digit, has no spaces
    - Examples: `Welcome1`, `$value`, `_value`, `button7`
        - `7button` is invalid
    - Java is case sensitive (capitalization matters)
        - `a1` and `A1` are different
- In chapters 2 to 7, start each class with `public class`
    - Details of this covered later

Good Programming Practice 2.3

By convention, always begin a class name’s identifier with a capital letter and start each subsequent word in the identifier with a capital letter. Java programmers know that such identifier normally represent Java classes, so naming your classes in this manner makes your programs more readable.

Common Programming Error 2.2

Java is case sensitive. Not using the proper uppercase and lowercase letters for an identifier normally causes a compilation error.

2.2 First Program in Java: Printing a Line of Text (Cont.)

```
4   public class Welcome1
```

- Saving files
  - File name must be class name with `.java` extension
  - `Welcome1.java`

```
5   {
```

- Left brace {
  - Begins body of every class
  - Right brace ends declarations (line 13)

Common Programming Error 2.3

It is an error for a `public` class to have a file name that is not identical to the class name (plus the `.java` extension) in terms of both spelling and capitalization.

Common Programming Error 2.4

It is an error not to end a file name with the `.java` extension for a file containing a class declaration. If that extension is missing, the Java compiler will not be able to compile the class declaration.

Good Programming Practice 2.4

Whenever you type an opening left brace, {, in your program, immediately type the closing right brace, }, then reposition the cursor between the braces and indent to begin typing the body. This practice helps prevent errors due to missing braces.

Good Programming Practice 2.5

Indent the entire body of each class declaration one “level” of indentation between the left brace, {, and the right brace, }, that delimit the body of the class. This format emphasizes the class declaration's structure and makes it easier to read.

Good Programming Practice 2.6

Set a convention for the indent size you prefer, and then uniformly apply that convention. The *Tab* key may be used to create indents, but tab stops vary among text editors. We recommend using three spaces to form a level of indent.

Common Programming Error 2.5

It is a syntax error if braces do not occur in matching pairs.

2.2 First Program in Java: Printing a Line of Text (Cont.)

```
7      public static void main( String args[] )
```

- Part of every Java application
  - Applications begin executing at `main`
    - Parentheses indicate `main` is a method (Ch. 3 and 6)
    - Java applications contain one or more methods
  - Exactly one method must be called `main`
- Methods can perform tasks and return information
  - `void` means `main` returns no information
  - For now, mimic `main`'s first line

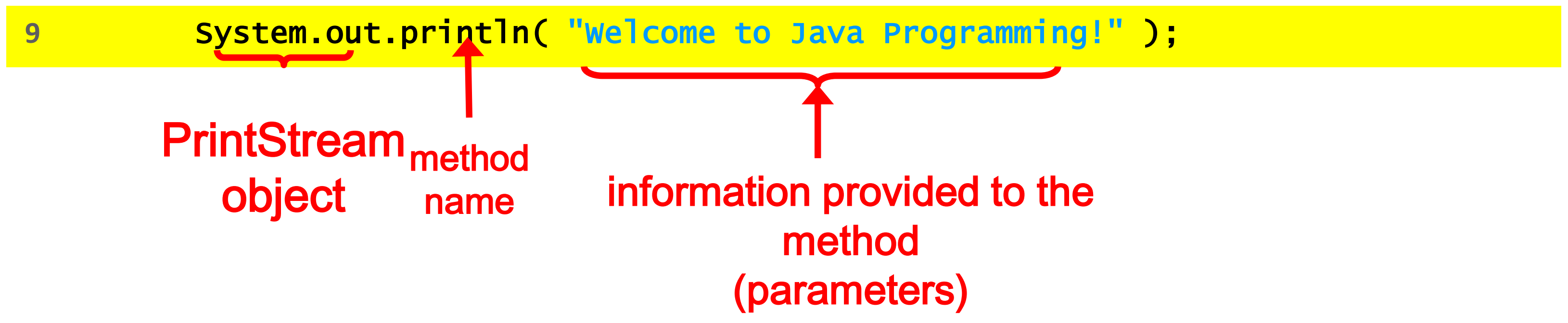
```
8      {
```

- Left brace begins body of method declaration
  - Ended by right brace `}` (line 11)

Good Programming Practice 2.7

Indent the entire body of each method declaration one “level” of indentation between the left brace, {, and the right brace, }, that define the body of the method. This format makes the structure of the method stand out and makes the method declaration easier to read.

2.2 First Program in Java: Printing a Line of Text (Cont.)



```
        System.out.println( "Welcome to Java Programming!" );
```

- Instructs computer to perform an action
  - Prints string of characters
    - String – series of characters inside double quotes
  - White-spaces in strings are not ignored by compiler
- `System.out`
  - Standard output object
  - Print to command window (i.e., MS-DOS prompt)
- Method `System.out.println`
  - Displays line of text
- This line known as a statement
  - Statements must end with semicolon ;

Common Programming Error 2.6

Omitting the semicolon at the end of a statement is a syntax error.

Error-Prevention Tip 2.1

When learning how to program, sometimes it is helpful to “break” a working program so you can familiarize yourself with the compiler's syntax-error messages. These messages do not always state the exact problem in the code. When you encounter such syntax-error messages in the future, you will have an idea of what caused the error. Try removing a semicolon or brace from the program of Fig. 2.1, then recompile the program to see the error messages generated by the omission.

Error-Prevention Tip 2.2

When the compiler reports a syntax error, the error may not be on the line number indicated by the error message. First, check the line for which the error was reported. If that line does not contain syntax errors, check several preceding lines.

2.2 First Program in Java: Printing a Line of Text (Cont.)

```
   } // end method main
```

- Ends method declaration

```
} // end class Welcome1
```

- Ends class declaration
- Can add comments to keep track of ending braces

Good Programming Practice 2.8

Following the closing right brace (}) of a method body or class declaration with an end-of-line comment indicating the method or class declaration to which the brace belongs improves program readability.

2.2 First Program in Java: Printing a Line of Text (Cont.)

Compiling a program

- NetBeans uses online compiling
- For compiling at the command prompt window, go to directory where program is stored
  - Type `javac Welcome1.java`
  - If no syntax errors, `Welcome1.class` created
    - Has bytecodes that represent application
    - Bytecodes passed to JVM

Error-Prevention Tip 2.4

The Java compiler generates syntax-error messages when the syntax of a program is incorrect. Each error message contains the file name and line number where the error occurred. For example, `Welcome1.java:6` indicates that an error occurred in the file `Welcome1.java` at line 6. The remainder of the error message provides information about the syntax error.

Error-Prevention Tip 2.5

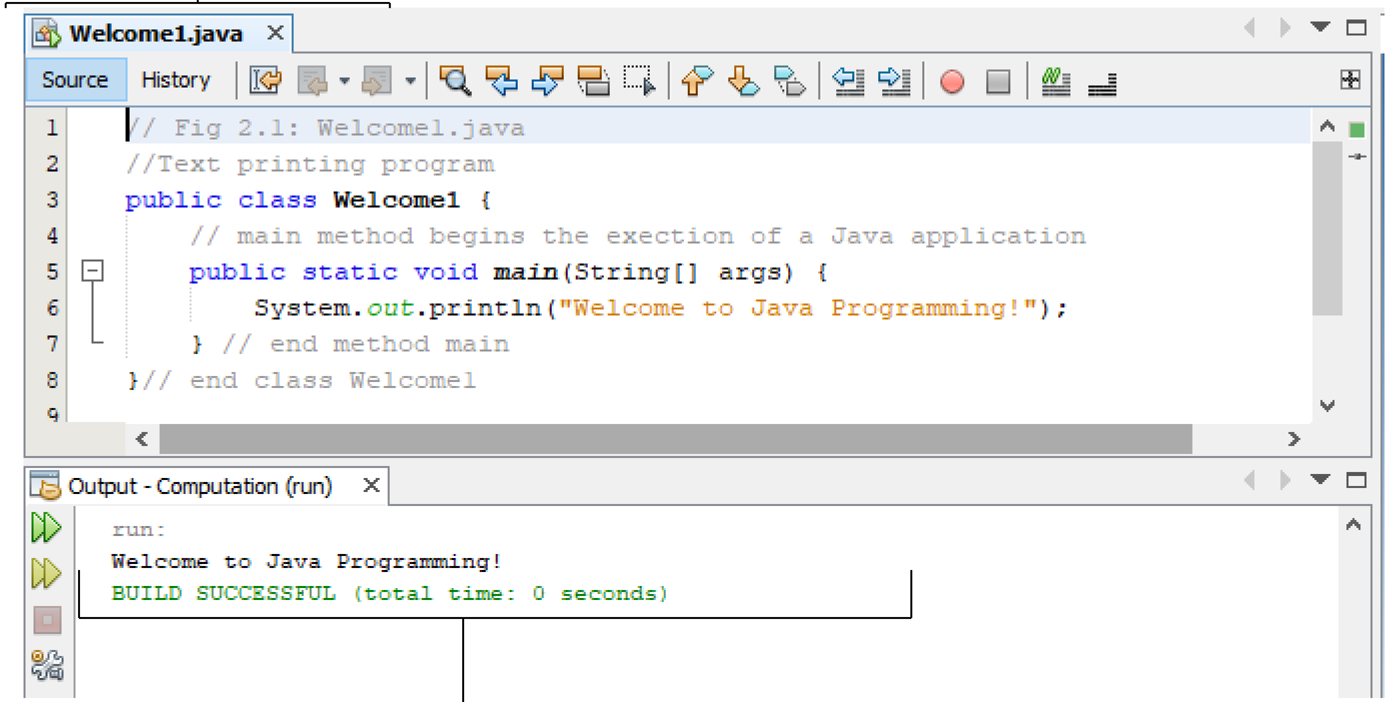
The compiler error message "`Public class` ClassName `must be defined in a file called` ClassName`.java`" indicates that the file name does not exactly match the name of the public class in the file or that you typed the class name incorrectly when compiling the class.

2.2 First Program in Java: Printing a Line of Text (Cont.)

Executing a program

- Type `java Welcome1`
  - Launches JVM
  - JVM loads `.class` file for class `Welcome1`
  - `.class` extension omitted from command
  - JVM calls method `main`

Right click in the Projects tab on the Java class with a main method



```
// Fig 2.1: Welcome1.java
//Text printing program
public class Welcome1 {
    // main method begins the exection of a Java application
    public static void main(String[] args) {
        System.out.println("Welcome to Java Programming!");
    } // end method main
}// end class Welcome1
```

```
run:
Welcome to Java Programming!
BUILD SUCCESSFUL (total time: 0 seconds)
```

The program outputs

Welcome to Java Programming!

Fig. 2.2 | **Executing `Welcome1` in a Microsoft Windows XP** Command Prompt **window.**

2.3 String Concatenation

The *string concatenation operator* (+) is used to append one string to the end of another

```
"Peanut butter " + "and jelly"
```

It can also be used to append a number to a string

A string literal cannot be broken across two lines in a program

2.3 String Concatenation

```
public class Facts
{
   //-----------------------------------------------------------------
   //  Prints various facts.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      // Strings can be concatenated into one long string
      System.out.println("We present the following facts for your "
                          + "extracurricular edification:");
      System.out.println();
      // A string can contain numeric digits
      System.out.println("Letters in the Hawaiian alphabet: 12");
      // A numeric value can be concatenated to a string
      System.out.println("Dialing code for Antarctica: " + 672);

      System.out.println("Year in which Leonardo da Vinci invented " +
                         "the parachute: " + 1515);

      System.out.println("Speed of ketchup: " + 40 + " km per year");
   }
}
```

Output

```
We present the following facts for your extracurricular edification:

Letters in the Hawaiian alphabet: 12
Dialing code for Antarctica: 672
Year in which Leonardo da Vinci invented the parachute: 1515
Speed of ketchup: 40 km per year
```

2.3 String Concatenation

The + operator is also used for arithmetic addition

The function that it performs depends on the type of the information on which it operates

If both operands are strings, or if one is a string and one is a number, it performs string concatenation

If both operands are numeric, it adds them

The + operator is evaluated left to right, but parentheses can be used to force the order

2.3 String Concatenation

```
public class Addition
{
   //-----------------------------------------------------------------
   //  Concatenates and adds two numbers and prints the results.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      System.out.println("24 and 45 concatenated: " + 24 + 45);

      System.out.println("24 and 45 added: " + (24 + 45));
   }
}
```

Output

```
24 and 45 concatenated: 2445
24 and 45 added: 69
```

2.3 Modifying Our First Java Program

Modify example in Fig. 2.1 to print same contents using different code

2.3 Modifying Our First Java Program (Cont.)

Modifying programs

- `Welcome2.java` (Fig. 2.3) produces same output as `Welcome1.java` (Fig. 2.1)
- Using different code

```
9          System.out.print( "Welcome to " );
10         System.out.println( "Java Programming!" );
```

- Line 9 displays “Welcome to ” with cursor remaining on printed line
- Line 10 displays “Java Programming! ” on same line with cursor on next line

```java
// Fig. 2.3: Welcome2.java
// Printing a line of text with multiple statements.

public class Welcome2
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.print( "Welcome to " );
      System.out.println( "Java Programming!" );

   } // end method main

} // end class Welcome2
```

`System.out.print` keeps the cursor on the same line, so `System.out.println` continues on the same line.

```
Welcome to Java Programming!
```

Outline

1. Comments

2. Blank line

3. Begin class `Welcome2`

3.1 Method `main`

4. Method `System.out.print`

4.1 Method `System.out.println`

5. end `main`, `Welcome2`

Program Output

2.3 Modifying Our First Java Program (Cont.)

Escape characters

- **Backslash (\ )**
- **Indicates special characters to be output**

Newline characters (`\n`)

- **Interpreted as "special characters" by methods `System.out.print` and `System.out.println`**
- **Indicates cursor should be at the beginning of the next line**
- **`Welcome3.java` (Fig. 2.4)**

```
9         System.out.println( "Welcome\nto\nJava\nProgramming!" );
```

- **Line breaks at `\n`**

Outline

```java
// Fig. 2.4: Welcome3.java
// Printing multiple lines of text with a single statement.

public class Welcome3
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.println( "Welcome\nto\nJava\nProgramming!" );

   } // end method main

} // end class Welcome3
```

```
Welcome
to
Java
Programming!
```

A new line begins after each `\n` escape sequence is output.

Welcome3.java

1. main

2. System.out.println (uses \n for new line)

Program Output

Escape sequence	Description
`\n`	Newline. Position the screen cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor at the beginning of the current line—do not advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double-quote character. For example, `System.out.println( "\"in quotes\"" );` displays `"in quotes"`

Fig. 2.5 | **Some common escape sequences.**

2.4 Displaying Text with `printf`

`System.out.printf`

- Displays formatted data

```
      System.out.printf( "%s%n%s%n",
         "Welcome to", "Java Programming!" );
```

- Format string
  - Fixed text
  - Format specifier – placeholder for a value
- Format specifier `%s` – placeholder for a string
- Format specifier `%n` – new line, portable across platforms(!)

Outline

```
// Fig. 2.6: Welcome4.java
// Printing multiple lines in a dialog box.

public class Welcome4
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.printf( "%s%n%s%n",
         "Welcome to", "Java Programming!" );

   } // end method main

} // end class Welcome4
```

`System.out.printf` displays formatted data.

```
Welcome to
Java Programming!
```

`Welcome4.java`

`main`

`printf`

`Program output`

Good Programming Practice 2.9

Place a space after each comma (,) in an argument list to make programs more readable.

Common Programming Error 2.7

Splitting a statement in the middle of an identifier or a string is a syntax error.

2.5 Another Java Application: Adding Integers

Upcoming program

- **Use `Scanner` to read two integers from user**
- **Use `printf` to display sum of the two values**
- **Use packages**

Integers

- **Whole numbers, like –22, 7, 0 and 1024)**

Programs remember numbers and other data in the computer's memory and access that data through program elements called variables.

The program of Fig. 2.7 demonstrates these concepts.

```
// Fig. 2.7: Addition.java
// Addition program that displays the sum of two numbers.
import java.util.Scanner; // program uses class Scanner

public class Addition
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      int number1; // first number to add
      int number2; // second number to add
      int sum; // sum of number1 and number2

      System.out.print( "Enter first integer: " ); // prompt
      number1 = input.nextInt(); // read first number from user

```

import declaration imports class Scanner from package java.util.

Declare and initialize variable input, which is a Scanner.

Declare variables number1, number2 and sum.

Read an integer from the user and assign it to number1.

Outline

Addition.java

(1 of 2)

aration

Scanner

nt

Outline

```
      System.out.print( "Enter second integer: " ); // prompt
      number2 = input.nextInt(); // read second number from user

      sum = number1 + number2; // add numbers

      System.out.printf( "Sum is %d%n", sum ); // d

   } // end method main

} // end class Addition
```

Read an integer from the user and assign it to `number2`.

Calculate the sum of the variables `number1` and `number2`, assign result to `sum`.

Display the sum using formatted output.

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Two integers entered by the user.

`Addition.java`

(2 of 2)

4. Addition

5. `printf`

2.5 Another Java Application: Adding Integers (Cont.)

```
3   import java.util.Scanner;  // program uses class Scanner
```

– `import` declarations

Classes are grouped into *packages—named groups of related classes*—and are collectively referred to as the Java class library, or the Java Application Programming Interface (Java API, JavaFX API).

You use `import` declarations to identify the predefined classes used in a Java program.

Tells compiler to load class `Scanner` from `java.util` package

Always use Fix Imports to import classes in NetBeans

```
5   public class Addition
6   {
```

- **Begins `public` class `Addition`**
  - **Recall that file name must be `Addition.java`**
- **Lines 8-9: begin `main`**

Common Programming Error 2.8

All `import` declarations must appear before the first class declaration in the file. Placing an `import` declaration inside a class declaration’s body or after a class declaration is a syntax error.

Error-Prevention Tip 2.7

Forgetting to include an `import` declaration for a class used in your program typically results in a compilation error containing a message such as "`cannot resolve symbol.`" When this occurs, check that you provided the proper `import` declarations and that the names in the `import` declarations are spelled correctly, including proper use of uppercase and lowercase letters.

2.5 Another Java Application: Adding Integers (Cont.)

```
// create Scanner to obtain input from command window
Scanner input = new Scanner( System.in );
```

- Variable Declaration Statement
- Variables
  - Location in memory that stores a value
    - *Must* be declared with a name and a type before they can be used.
    - A variable's *name* enables the program to access the value of the variable in memory. Can be any valid identifier
    - A variable's type specifies what kind of information is stored at that location in memory
  - `input` is of type `Scanner`
- Declarations end with semicolons `;`
- Initialize variable in its declaration
  - Assignment operator uses the equal sign
  - Standard input object
    - `System.in`

2.5 Another Java Application: Adding Integers (Cont.)

```
        // create Scanner to obtain input from command window
        Scanner input = new Scanner( System.in );
```

`Scanner`

- Enables a program to read data for use in a program.
- Data can come from many sources, such as the user at the keyboard or a file on disk.
- Before using a `Scanner`, you must create it and specify the source of the data.

The equals sign (=) in a declaration indicates that the variable should be initialized (i.e., prepared for use in the program) with the result of the expression to the right of the equals sign.

The `new` keyword creates an object.

Standard input object, `System.in`, enables applications to read bytes of data typed by the user.

`Scanner` object translates these bytes into types that can be used in a program.

2.5 Another Java Application: Adding Integers (Cont.)

Because strings are so common, we don't have to use the `new` operator to create a `String` object

```
title = "Java Software Solutions";
```

This is special syntax that works only for strings

Each string literal (enclosed in double quotes) represents a `String` object

2.5 Another Java Application: Adding Integers (Cont.)

- Once a `String` object has been created, neither its value nor its length can be changed
- Therefore we say that an object of the `String` class is ***immutable***
- However, several methods of the `String` class return new `String` objects that are modified versions of the original

2.5 Another Java Application: Adding Integers (Cont.)

```
        int number1; // first number to add
         int number2; // second number to add
       int sum; // sum of number 1 and number 2
```

- Declare variable `number1`, `number2` and `sum` of type `int`
  - `int` holds integer values (whole numbers): i.e., `0`, `-4`, `97`
  - Types `float` and `double` can hold decimal numbers
  - Type `char` can hold a single character: i.e., x, $, \n, 7
  - `int`, `float`, `double` and `char` are primitive types
- Can add comments to describe purpose of variables

```
int number1, // first number to add
    number2, // second number to add
    sum; // sum of number1 and number2
```

- Can declare multiple variables of the same type in one declaration
- Use comma-separated list

Good Programming Practice 2.10

Declare each variable on a separate line. This format allows a descriptive comment to be easily inserted next to each declaration.

Good Programming Practice 2.11

Choosing meaningful variable names helps a program to be *self-documenting* (i.e., one can understand the program simply by reading it rather than by reading manuals or viewing an excessive number of comments).

Good Programming Practice 2.12

By convention, variable-name identifiers begin with a lowercase letter, and every word in the name after the first word begins with a capital letter. For example, variable-name identifier `firstNumber` has a capital `N` in its second word, `Number`.

2.5 Another Java Application: Adding Integers (Cont.)

```
        System.out.print( "Enter first integer: " ); // prompt
```

- Message called a **prompt** - directs user to perform an action
- Package `java.lang`

```
        number1 = input.nextInt(); // read first number from user
```

- Result of call to `nextInt` given to `number1` using assignment operator =
  - Assignment statement
  - = binary operator - takes two operands
    - Expression on right evaluated and assigned to variable on left
  - Read as: `number1` gets the value of `input.nextInt()`

Software Engineering Observation 2.1

By default, package `java.lang` is imported in every Java program; thus, `java.lang` is the only package in the Java API that does not require an `import` declaration.

Good Programming Practice 2.13

Place spaces on either side of a binary operator to make it stand out and make the program more readable.

2.5 Another Java Application: Adding Integers (Cont.)

```
20          System.out.print( "Enter second integer: " ); // prompt
```

- Similar to previous statement
  - Prompts the user to input the second integer

```
21          number2 = input.nextInt(); // read second number from user
```

- Similar to previous statement
  - Assign variable number2 to second integer input

```
23          sum = number1 + number2; // add numbers
```

- Assignment statement
  - Calculates sum of `number1` and `number2` (right hand side)
  - Uses assignment operator = to assign result to variable `sum`
  - Read as: `sum` gets the value of `number1 + number2`
  - `number1` and `number2` are operands

2.5 Another Java Application: Adding Integers (Cont.)

Variable declaration statement

```
int number1 = input.nextInt(); // read first
number from user
```

declares that variables `number1` holds data of type `int`

- Range of values for an `int` is –2,147,483,648 to +2,147,483,647.
- The `int` values you use in a program may not contain commas.

For readability, you can place underscores in numbers

- 60_000_000 represents the `int` value 60,000,000

2.5 Another Java Application: Adding Integers (Cont.)

- **`Scanner` method `nextInt`**
  - **Obtains an integer from the user at the keyboard.**
  - **Program *waits* for the user to type the number and press the *Enter* key to submit the number to the program.**
- **The result of the call to method `nextInt` is placed in variable `number1`**
  - **The = indicates that `int` variable `number1` should be initialized in its declaration with the result of `input.nextInt()`**

2.5 Another Java Application: Adding Integers (Cont.)

```
        System.out.printf( "Sum is %d%n: " , sum ); // display sum
```

- Use `System.out.printf` to display results
- Format specifier `%d` is a *placeholder* for an `int` value
  - The letter `d` stands for “decimal integer.”
- Calculations can also be performed inside `printf`

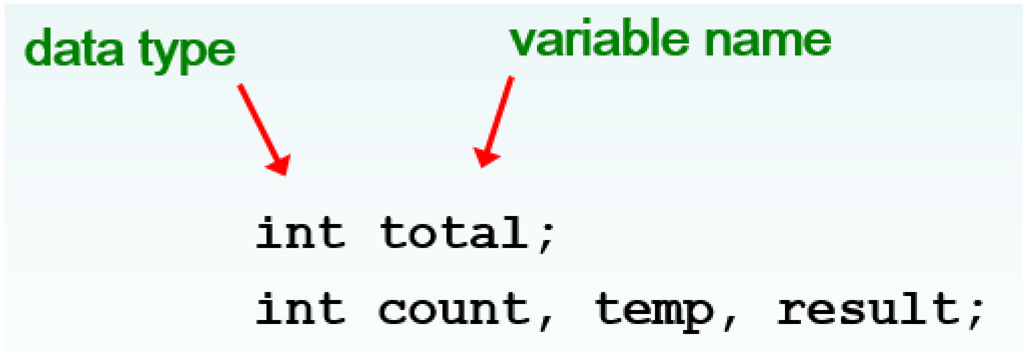
```
        System.out.printf( "Sum is %d%n: " , ( number1 + number2 ) );
```

- Parentheses around the expression `number1` + `number2` are not required

2.6 Memory Concepts

Variables

- **Every variable has a name, a type, a size and a value**
  - **Name corresponds to location in memory**
- **When new value is placed into a variable, replaces (and destroys) previous value**
- **Reading variables from memory does not change them**



```
int total;
int count, temp, result;
```

A variable can be given an initial value in the declaration

```
int number1 = 45;
int total    = 32, max = 149;
```

When a variable is referenced in a program, its current value is used. The value that was in **`total`** is overwritten

```
total = 55;
```

An *assignment statement* changes the value of a variable. You can only assign a value to a variable that is consistent with the variable's declared type

The assignment operator is the = sign

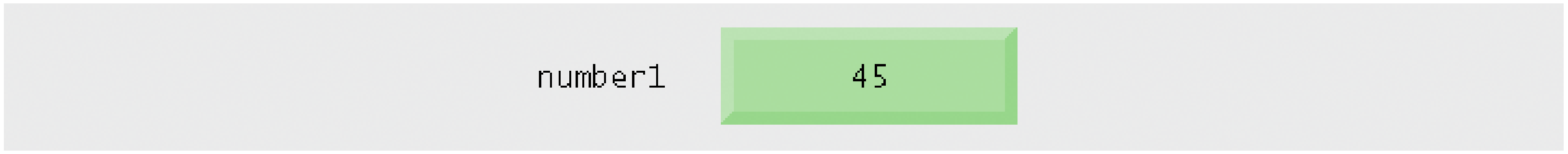


Fig. 2.8 | **Memory location showing the name and value of variable `number1`.**

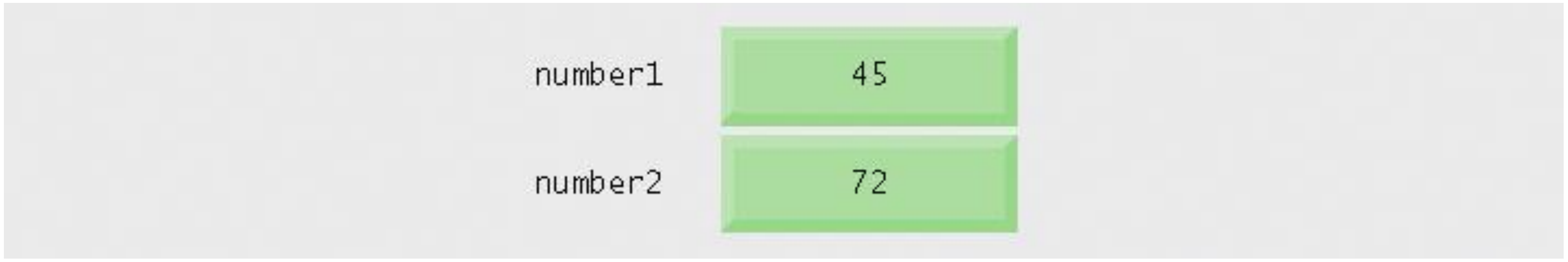


Fig. 2.9 | Memory locations after storing values for `number1` and `number2`.

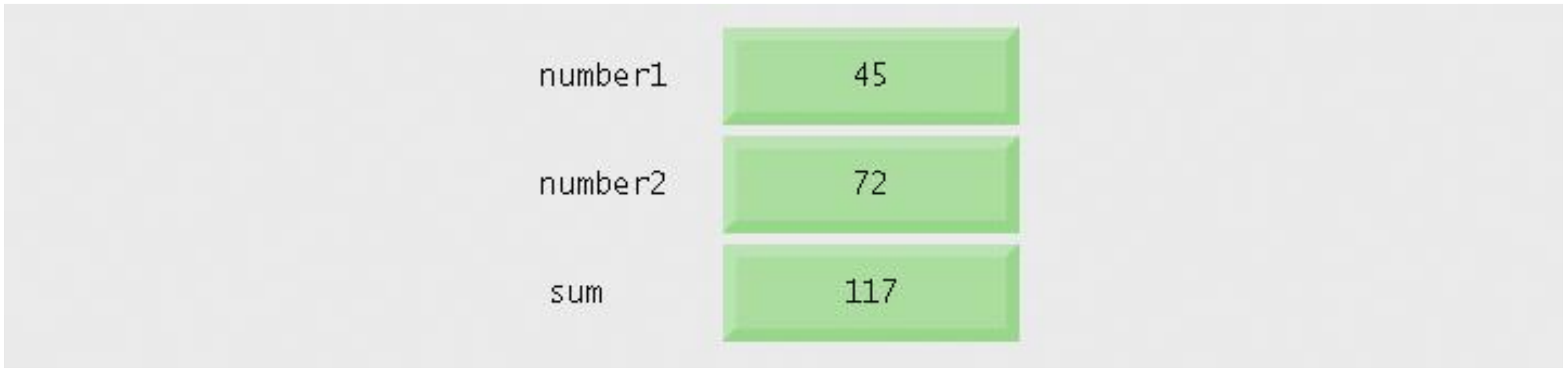


Fig. 2.10 | Memory locations after calculating and storing the sum of `number1` and `number2`.

Constants

A *constant* is an identifier that is similar to a variable except that it holds the same value during its entire existence

As the name implies, it is constant, not variable

The compiler will issue an error if you try to change the value of a constant

In Java, we use the `final` modifier to declare a constant

```
final int MIN_HEIGHT = 69;
```

Constants

Constants are useful for three important reasons

First, they give meaning to otherwise unclear literal values

- **Example: `MAX_LOAD` means more than the literal 250**

Second, they facilitate program maintenance

- **If a constant is used in multiple places, its value need only be set in one place**

Third, they formally establish that a value should not change, avoiding inadvertent errors by other programmers

Primitive Data

There are eight primitive data types in Java

Four of them represent integers:

– `byte`, `short`, `int`, `long`

Two of them represent floating point numbers:

– `float`, `double`

One of them represents characters:

– `char`

And one of them represents boolean values:

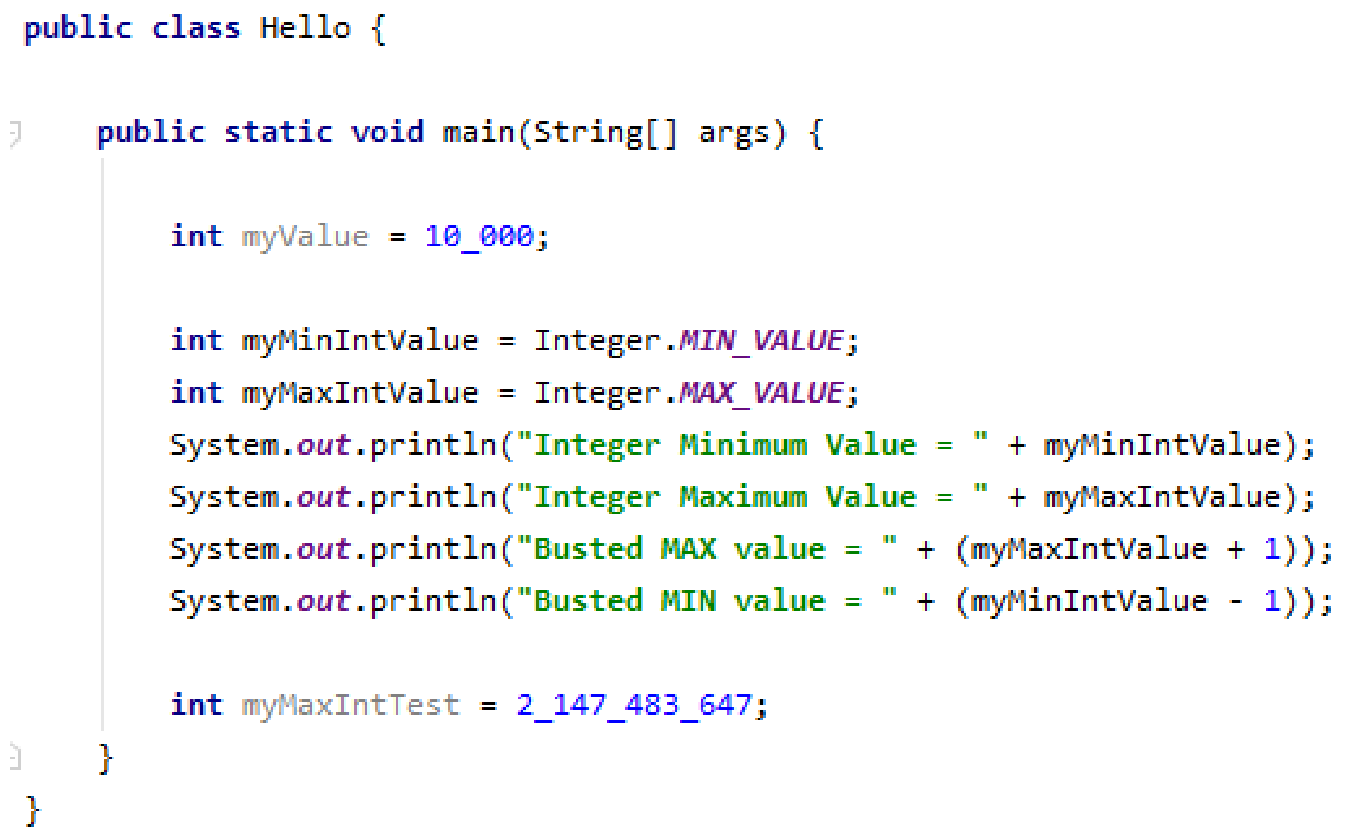
– `boolean`

Numeric Primitive Data

The difference between the numeric primitive types is their size and the values they can store:

Type	Storage	Min Value	Max Value
`byte`	8 bits	-128	127
`short`	16 bits	-32,768	32,767
`int`	32 bits	-2,147,483,648	2,147,483,647
`long`	64 bits	< -9 x 10^{18}	> 9 x 10^{18}
`float`	32 bits	+/- 3.4 x 10^{38} with 7 significant digits	
`double`	64 bits	+/- 1.7 x 10^{308} with 15 significant digits	

Numeric Primitive Data



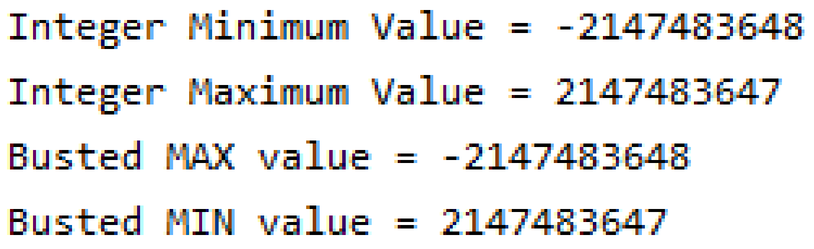
```
public class Hello {

    public static void main(String[] args) {

        int myValue = 10_000;

        int myMinIntValue = Integer.MIN_VALUE;
        int myMaxIntValue = Integer.MAX_VALUE;
        System.out.println("Integer Minimum Value = " + myMinIntValue);
        System.out.println("Integer Maximum Value = " + myMaxIntValue);
        System.out.println("Busted MAX value = " + (myMaxIntValue + 1));
        System.out.println("Busted MIN value = " + (myMinIntValue - 1));

        int myMaxIntTest = 2_147_483_647;
    }
}
```



```
Integer Minimum Value = -2147483648
Integer Maximum Value = 2147483647
Busted MAX value = -2147483648
Busted MIN value = 2147483647
```

Characters

A `char` variable stores a single character

Character literals are delimited by single quotes:

```
'a'    'X'     '7'     '$'     ','     '\n'
```

Example declarations:

```
char topGrade = 'A';
char terminator = ';', separator = ' ';
```

Note the difference between a primitive character variable, which holds only one character, and a `String` object, which can hold multiple characters

Boolean

A `boolean` value represents a true or false condition

The reserved words `true` and `false` are the only valid values for a boolean type

```
boolean done = false;
```

A `boolean` variable can also be used to represent any two states, such as a light bulb being on or off

2.7 Arithmetic

Arithmetic calculations used in most programs

- **Usage**
  - **`*` for multiplication**
  - **`/` for division**
  - **`%` for remainder**
  - **`+`, `-`**
- **Integer division truncates remainder**

  `7 / 5` evaluates to `1`
- **Remainder operator `%` returns the remainder**

  `7 % 5` evaluates to `2`

2.7a Division and Remainder

If both operands to the division operator (`/`) are integers, the result is an integer (the fractional part is discarded)

`14 / 3`	equals	4
`8 / 12`	equals	0

The remainder operator (%) returns the remainder after dividing the first operand by the second

`14 % 3`	equals	2
`8 % 12`	equals	8

Java operation	Arithmetic operator	Algebraic expression	Java expression
Addition	`+`	$f + 7$	`f + 7`
Subtraction	`-`	$p - c$	`p - c`
Multiplication	`*`	bm	`b * m`
Division	`/`	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`

Fig. 2.11 | **Arithmetic operators.**

2.7 Arithmetic (Cont.)

Operator precedence

- **Some arithmetic operators act before others (i.e., multiplication before addition)**
  - **Use parenthesis when needed**
- **Example: Find the average of three variables `a`, `b` and `c`**
  - **Do not use:** `a + b + c / 3`
  - **Use:** `( a + b + c ) / 3`

Operator(s)	Operation(s)	Order of evaluation (precedence)

`*`
`/`
`%`
	Multiplication
	Division
	Remainder
		Evaluated first. If there are several operators of this type, they are evaluated from left to right.

`+` `-`	Addition Subtraction	Evaluated next. If there are several operators of this type, they are evaluated from left to right.

Fig. 2.12 | Precedence of arithmetic operators.

2.7 Arithmetic (Cont.)

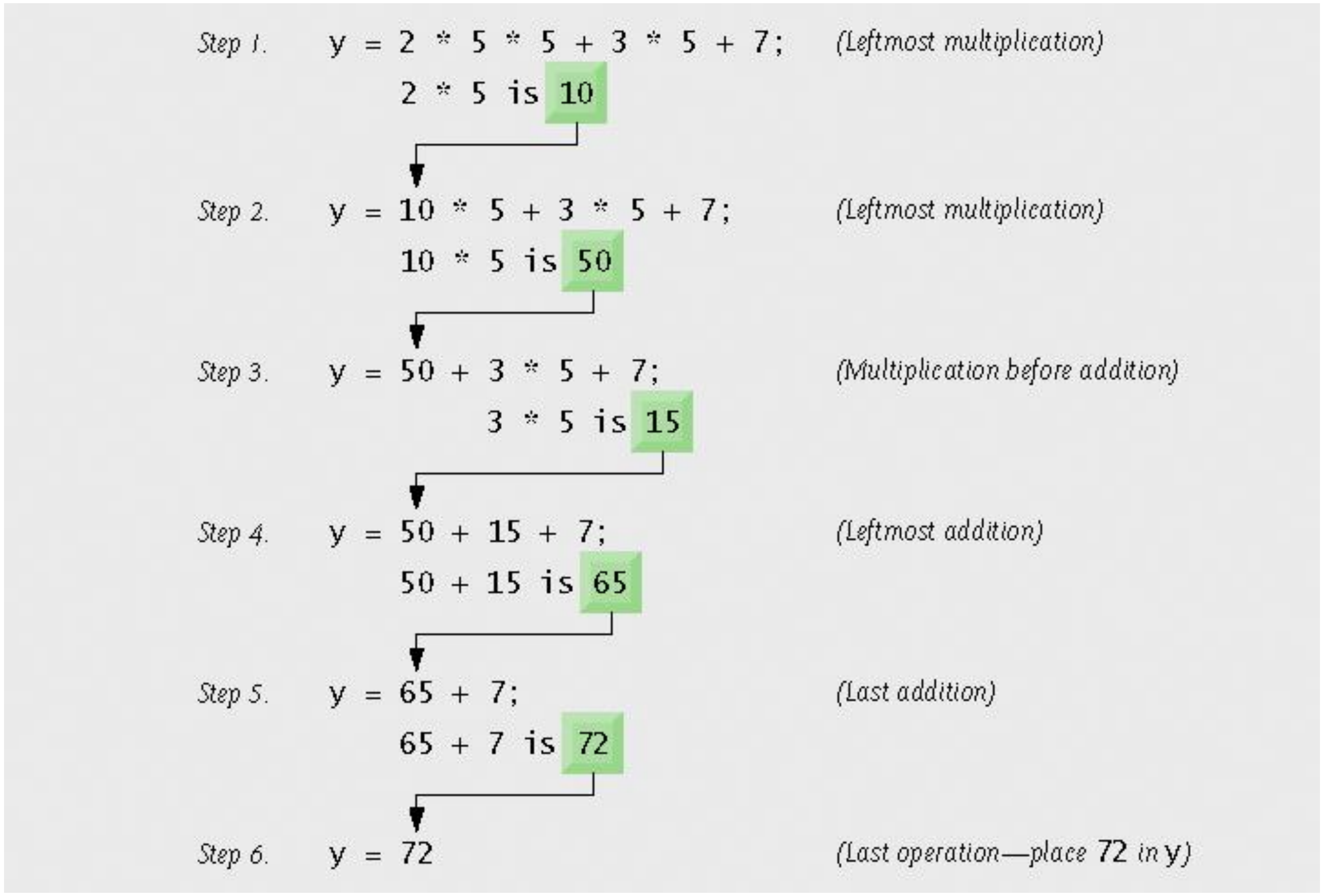
Operator %

- **Returns the remainder of division of two numbers**
- **Examples:**

```
 3 % 10 = 3
-3 % 10 = -3
 3 % -10 = 3
-13 % -10 = -3
 2.50 %  5 = 2.50
-5.50 %  5 = -0.50
-5.50 % -5 = -0.50
-5.50 % -0.5 = 0
```

Good Programming Practice 2.14

Using parentheses for complex arithmetic expressions, even when the parentheses are not necessary, can make the arithmetic expressions easier to read.



```
Step 1.   y = 2 * 5 * 5 + 3 * 5 + 7;      (Leftmost multiplication)
              2 * 5 is 10

Step 2.   y = 10 * 5 + 3 * 5 + 7;         (Leftmost multiplication)
              10 * 5 is 50

Step 3.   y = 50 + 3 * 5 + 7;             (Multiplication before addition)
                   3 * 5 is 15

Step 4.   y = 50 + 15 + 7;                (Leftmost addition)
              50 + 15 is 65

Step 5.   y = 65 + 7;                     (Last addition)
              65 + 7 is 72

Step 6.   y = 72                          (Last operation—place 72 in y)
```

Fig. 2.13 | **Order in which a second-degree polynomial is evaluated.**

2.7 Assignment Operators

There are many assignment operators in Java, including the following:

Operator	Example	Equivalent To
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

2.7 Assignment Operators

The right hand side of an assignment operator can be a complex expression

The entire right-hand expression is evaluated first, then the result is combined with the original variable

Therefore

```
            result /= (total-MIN) % num;
```

is equivalent to

```
            result = result / ((total-MIN) %
num);
```

2.8 Decision Making: Equality and Relational Operators

- Condition
  - Expression can be either `true` or `false`
- `if` statement
  - Simple version in this section, more detail later
  - If a condition is `true`, then the body of the `if` statement executed
  - Control always resumes after the `if` statement
  - Conditions in `if` statements can be formed using equality or relational operators (next slide)

Standard algebraic equality or relational operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
Equality operators			
=	`==`	`x == y`	`x` is equal to `y`
≠	`!=`	`x != y`	`x` is not equal to `y`
Relational operators			
>	`>`	`x > y`	`x` is greater than `y`
<	`<`	`x < y`	`x` is less than `y`
≥	`>=`	`x >= y`	`x` is greater than or equal to `y`
≤	`<=`	`x <= y`	`x` is less than or equal to `y`

Fig. 2.14 | **Equality and relational operators.**

```
// Fig. 2.15: Comparison.java
// Compare integers using if statements, relational operators
// and equality operators.
import java.util.Scanner; // program uses class Scanner

public class Comparison
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      int number1; // first number to compare
      int number2; // second number to compare

      System.out.print( "Enter first integer: " ); // prompt
      number1 = input.nextInt(); // read first number from user

      System.out.print( "Enter second integer: " ); // prompt
      number2 = input.nextInt(); // read second n

      if ( number1 == number2 )
         System.out.printf( "%d == %d%n", number1,

      if ( number1 != number2 )
         System.out.printf( "%d != %d%n", number1, number2 );

      if ( number1 < number2 )
         System.out.printf( "%d < %d%n", number1,
```

Test for equality, display result using `printf`.

Compares two numbers using relational operator <.

Outline

Comparison.java

(1 of 2)

1. Class Comparison

1.1 main

1.2 Declarations

1.3 Input data (nextInt)

1.4 Compare two inputs using if statements

Outline

```

      if ( number1 > number2 )
         System.out.printf( "%d > %d%n", number1, number2 );

      if ( number1 <= number2 )
         System.out.printf( "%d <= %d%n", number1,

      if ( number1 >= number2 )
         System.out.printf( "%d >= %d%n", number1, number2 );

   } // end method main

} // end class Comparison
```

Compares two numbers using relational operators >, <= and >=.

Comparison.java

(2 of 2)

```
Enter first integer: 777
Enter second integer: 777
777 == 777
777 <= 777
777 >= 777
```

```
Enter first integer: 1000
Enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000
```

```
Enter first integer: 2000
Enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
```

Program output

2.8 Decision Making: Equality and Relational Operators (Cont.)

- Line 6: begins class `Comparison` declaration
- Line 12: declares Scanner variable input and assigns it a Scanner that inputs data from the standard input
- Lines 14-15: declare `int` variables
- Lines 17-18: prompt the user to enter the first integer and input the value
- Lines 20-21: prompt the user to enter the second integer and input the value

2.8 Decision Making: Equality and Relational Operators (Cont.)

```
23        if ( number1 == number2 )
24           System.out.printf( "%d == %d%n", number1, number2 );
```

- `if` statement to test for equality using (==)
  - If variables equal (condition true)
    - Line 24 executes
  - If variables not equal, statement skipped
  - No semicolon at the end of line 23
  - Empty statement
    - No task is performed
- Lines 26-27, 29-30, 32-33, 35-36 and 38-39
  - Compare `number1` and `number2` with the operators `!=`, <, >, <= and >=, respectively

2.8a Input Tokens

Once created, the `Scanner` object can be used to invoke various input methods, such as:

```
answer = scan.nextLine();
```

Unless specified otherwise, *white space* is used to separate the elements (called *tokens*) of the input

White space includes space characters, tabs, new line characters

The `next` method of the `Scanner` class reads the next input token and returns it as a string

Methods such as `nextInt` and `nextDouble` read the next input token of the corresponding data types. In case the token is not of the expected data type the program aborts execution.

Common Programming Error 2.9

Forgetting the left and/or right parentheses for the condition in an `if` statement is a syntax error—the parentheses are required.

Common Programming Error 2.10

Confusing the equality operator, ==, with the assignment operator, =, can cause a logic error or a syntax error. The equality operator should be read as “is equal to,” and the assignment operator should be read as “gets” or “gets the value of.” To avoid confusion, some people read the equality operator as “double equals” or “equals equals.”

Common Programming Error 2.11

It is a syntax error if the operators ==, !=, >= and <= contain spaces between their symbols, as in = =, ! =, > = and < =, respectively.

Common Programming Error 2.12

Reversing the operators !=, >= and <=, as in =!, => and =<, is a syntax error.

Good Programming Practice 2.15

Indent an `if` statement’s body to make it stand out and to enhance program readability.

Good Programming Practice 2.16

Place only one statement per line in a program. This format enhances program readability.

Common Programming Error 2.13

Placing a semicolon immediately after the right parenthesis of the condition in an `if` statement is normally a logic error.

Good Programming Practice 2.17

A lengthy statement can be spread over several lines. If a single statement must be split across lines, choose breaking points that make sense, such as after a comma in a comma-separated list, or after an operator in a lengthy expression. If a statement is split across two or more lines, indent all subsequent lines until the end of the statement.

Good Programming Practice 2.18

Refer to the operator precedence when writing expressions containing many operators. Confirm that the operations in the expression are performed in the order you expect. If you are uncertain about the order of evaluation in a complex expression, use parentheses to force the order, exactly as you would do in algebraic expressions. Observe that some operators, such as assignment, =, associate from right to left rather than from left to right.

Operators	Associativity	Type
`*` `/` `%`	left to right	multiplicative
`+` `-`	left to right	additive
`<` `<=` `>` `>=`	left to right	relational
`==` `!=`	left to right	equality
`=`	right to left	assignment

Fig. 2.16 | **Precedence and associativity of operations discussed.**

Data Conversion

Sometimes it is convenient to convert data from one type to another

For example, in a particular situation we may want to treat an integer as a floating point value

These conversions do not change the type of a variable or the value that's stored in it – they only convert a value as part of a computation

Data Conversion

Widening conversions **are safest because they tend to go from a small data type to a larger one (such as a `short` to an `int`)**

Narrowing conversions **can lose information because they tend to go from a large data type to a smaller one (such as an `int` to a `short`)**

In Java, data conversions can occur in three ways:

- **assignment conversion**
- **promotion**
- **casting**

Data Conversion

Widening Conversions

From	To
`byte`	`short`, `int`, `long`, `float`, or `double`
`short`	`int`, `long`, `float`, or `double`
`char`	`int`, `long`, `float`, or `double`
`int`	`long`, `float`, or `double`
`long`	`float` or `double`
`float`	`double`

Narrowing Conversions

From	To
`byte`	`char`
`short`	`byte` or `char`
`char`	`byte` or `short`
`int`	`byte`, `short`, or `char`
`long`	`byte`, `short`, `char`, or `int`
`float`	`byte`, `short`, `char`, `int`, or `long`
`double`	`byte`, `short`, `char`, `int`, `long`, or `float`

Assignment Conversion

Assignment conversion **occurs when a value the right-side variable type in an assignment can be implicitly converted to the type of the left-side variable**

Example:

```
int dollars = 20;
double money = dollars;
```

Only widening conversions can happen via assignment

Note that the value or type of `dollars` did not change. The following assignment, however, causes a compilation error because it results in loss of precision.

```
int dollars = 20.0;
```

Warning

The following conversions are legal, but they may lose information:

From **`int`** to **`float`**

From **`long`** to float or **`double`**

For example, consider the assignment

```
float f = 123456789;
```

Because a float only has about **`seven`** significant digits, **`f`** is actually:

```
1.23456792E8
```

Casting

Casting is the most powerful, and dangerous, technique for conversion

Both widening and narrowing conversions can be accomplished by explicitly casting a value

To cast, the type is put in parentheses in front of the value being converted

```
int total = 50;
float result = (float) total / 6;
```

Without the cast, the fractional part of the answer would be lost

Promotion

Promotion **happens automatically when operators in expressions convert their operands**

Example:

```
int count = 12;
double sum = 490.27;
double result = sum / count;
```

The value of `count` is converted to a floating point value to perform the division calculation

Problems to solve

Problem 1

What is the output of the following program? Explain the output.

```
public class TestString {

    public static void main(String[] args) {
        String s1 = "smart";
        String s2 = "sm" + "art";
        String s3 = " boy";
        if (s1 == s2)//equal strings
            System.out.println("s1 equals s2");
        else
            System.out.println("s1 is not equal to s2");
        if ((s2 + s3) == "smart boy") //different strings
            System.out.println("s2 + s3 equals \"smart boy\"");
        else
            System.out.println("s2 + s3 is not equal to \"smart boy\"");
    }
}
```

Problems to solve

Problem 2

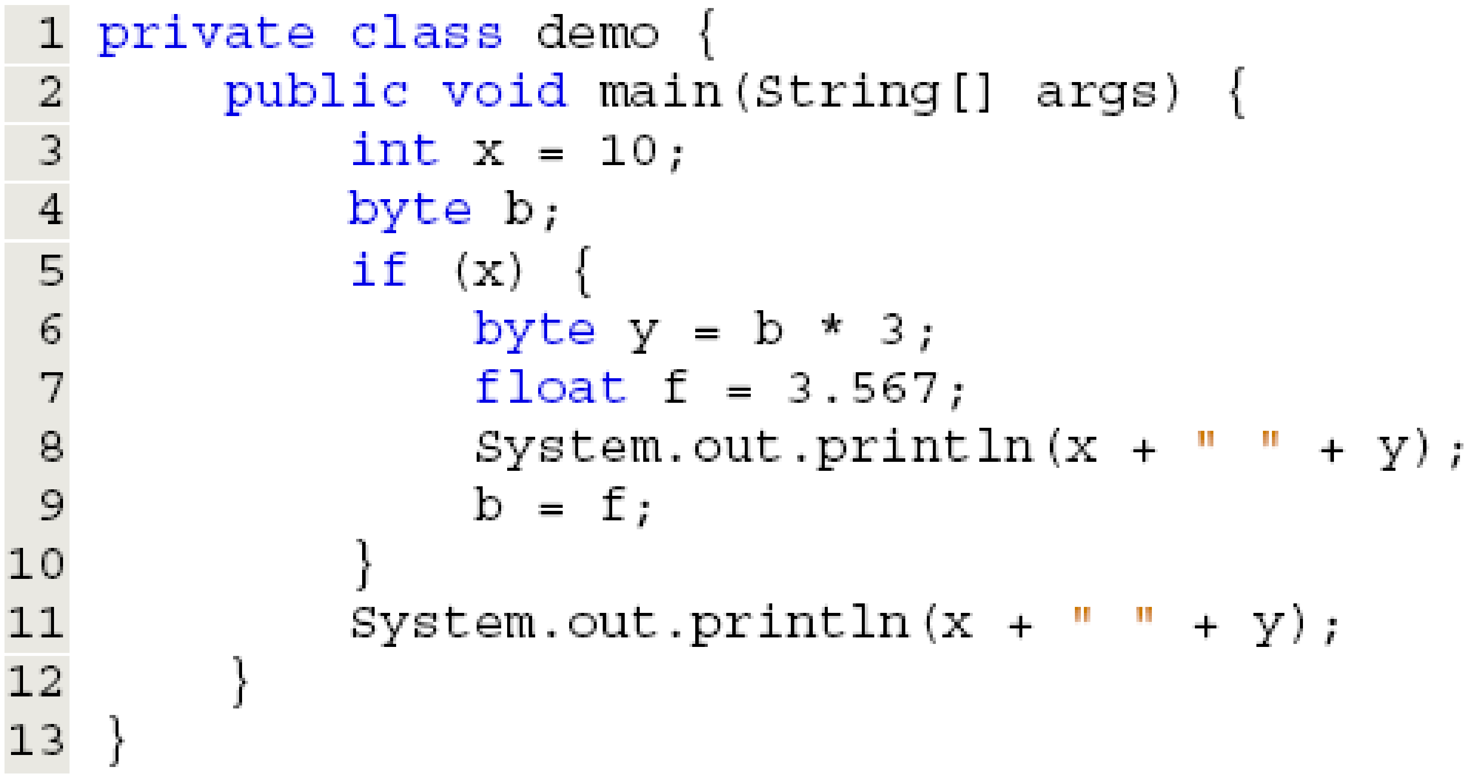
What are the results of the following expressions?

```
    12 / 2
12.0 / 2.0
    10 / 4
  10 / 4.0
  -4 % -10
 4.0 / 10
   -12 % 3
   -10 % 3
   3 % -10
5.50 % 2.50
```

Problems to solve

Problem 3

Identify errors in the following program and state the reason



```
private class demo {
    public void main(String[] args) {
        int x = 10;
        byte b;
        if (x) {
            byte y = b * 3;
            float f = 3.567;
            System.out.println(x + " " + y);
            b = f;
        }
        System.out.println(x + " " + y);
    }
}
```

Problems to solve

Problem 4

In this lecture, you learned about integers and the type int. Java can also represent uppercase letters, lowercase letters and a considerable variety of special symbols. Every character has a corresponding integer representation. The set of characters a computer uses together with the corresponding integer representations for those characters is called that computer's character set. You can indicate a character value in a program simply by enclosing that character in single quotes, as in 'A'.

Write an application that displays the integer equivalents of some uppercase letters, lowercase letters, digits and special symbols. Display the integer equivalents of the following: A B C a b c 0 1 2 $ * + / and the blank character.

Problems to solve

Problem 5

Write an application that inputs one number consisting of five digits from the user, separates the number into its individual digits and prints the digits separated from one another by three spaces each. For example, if the user types in the number 42339, the program should print

4 2 3 3 9

Assume that the user enters the correct number of digits. What happens when you enter a number with more than five digits? What happens when you enter a number with fewer than five digits? [Hint: It’s possible to do this exercise with the techniques you learned in this lecture. You’ll need to use both division and remainder operations to “pick off ” each digit.]