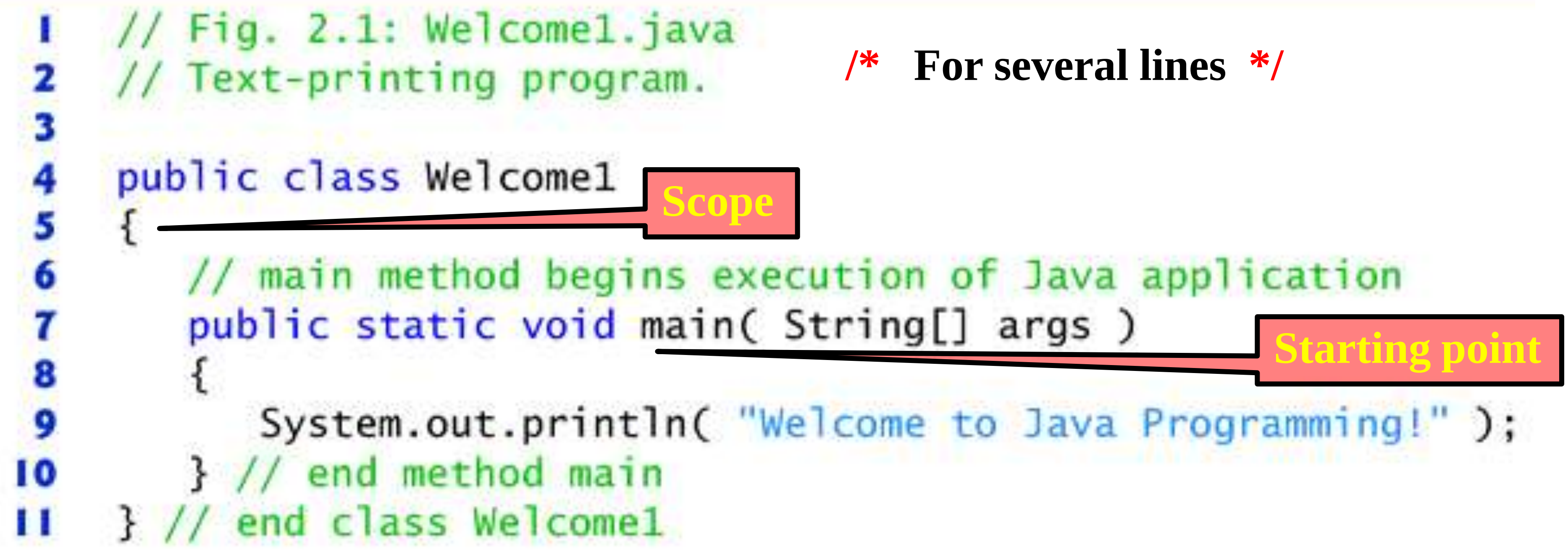
Introducing the Java (Cont.1)

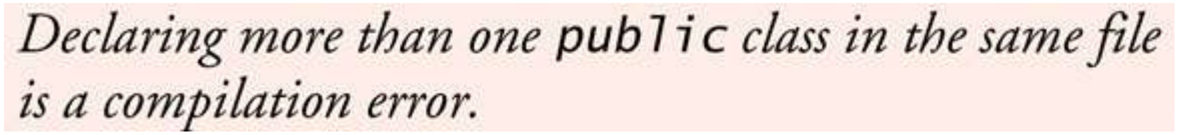


```java
// Fig. 2.1: Welcome1.java
// Text-printing program.

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
```

/* For several lines */

- **Every Java program consists of at least one class that you define.**
- **Java is case sensitive—uppercase.**
- `javac Welcome1.java`
- `javadoc Welcome1.java`
- **Class Welcome1 is public and should be declared in file named Welcome1.java**



Declaring more than one `public` *class in the same file is a compilation error.*

```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
```

Prints `Welcome to` and leaves cursor on same line

Prints `Java Programming!` starting where the cursor was positioned previously, then outputs a newline character

```
Welcome to Java Programming!
```

```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
```

Each \n moves the output cursor to the next line, where output continues

```
Welcome
to
Java
Programming!
```

Fig. 2.4 | Printing multiple lines of text with a single statement.

```java
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
```

Each \n moves the output cursor to the next line, where output continues

```
Welcome
to
Java
Programming!
```

Fig. 2.4 | Printing multiple lines of text with a single statement.

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.

```java
// Fig. 2.6: Welcome4.java
// Displaying multiple lines with method System.out.printf.

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
```

Each %s is a placeholder for a String that comes later in the argument list

Statements can be split over multiple lines.

```
Welcome to
Java Programming!
```

- `import` declaration
  - Helps the compiler locate a class that is used in this program.
  - Rich set of predefined classes that you can reuse rather than "reinventing the wheel."
  - 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).
  - You use `import` declarations to identify the predefined classes used in a Java program.

- Prompt
  - Output statement that directs the user to take a specific action.
- `System` is a class.
  - Part of package `java.lang`.
  - Class `System` is not imported with an `import` declaration at the beginning of the program.

- `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 information typed by the user.
- `Scanner` object translates these bytes into types that can be used in a program.

```java
// 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 a Scanner to obtain input from the 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

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

      sum = number1 + number2; // add numbers, then store total in sum
```

Imports class **Scanner** for use in this program

Creates **Scanner** for reading data from the user

Variables that are declared but not initialized

Reads an **int** value from the user

Reads another **int** value from the user

Sums the values of **number1** and **number2**

```java
      System.out.printf( "Sum is %d\n", sum ); // display sum
   } // end method main
} // end class Addition
```

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

- **Integer formatted output**

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

- **Format specifier `%d` is a placeholder for an `int` value**
- **The letter `d` stands for "decimal integer."**

Java operation	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`
Remainder	`%`	$r \bmod s$	`r % s`

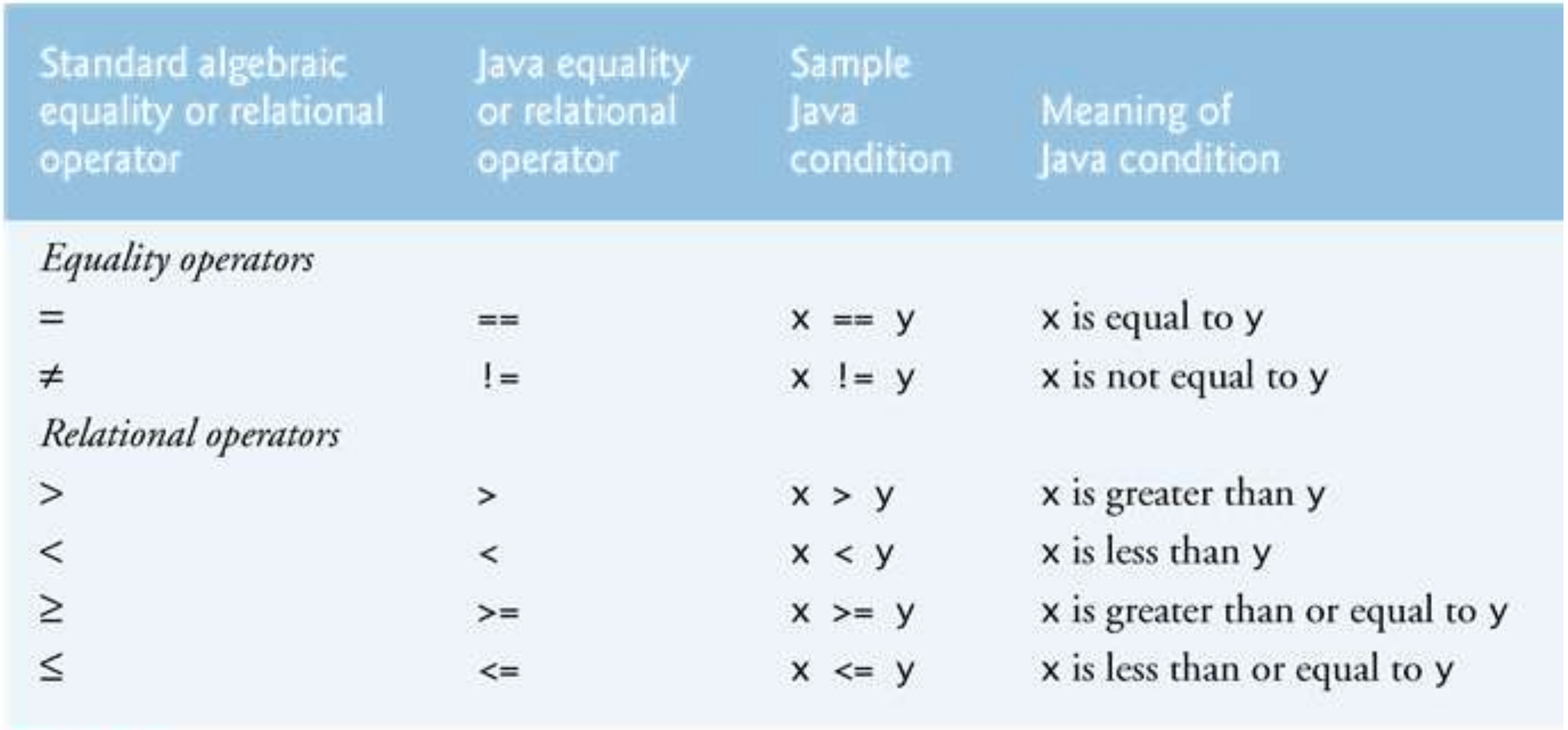
Fig. 2.11 | Arithmetic operators.

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.
`=`	Assignment	Evaluated last.

Fig. 2.12 | Precedence of arithmetic operators.



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.

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

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

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

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

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

      if ( number1 >= number2 )
         System.out.printf( "%d >= %d\n", number1, number2 );
   } // end method main
} // end class Comparison
```

Output statement executes only if the numbers are equal

Output statement executes only if the numbers are not equal

Output statement executes only if **number1** is less than **number2**

Output statement executes only if **number1** is greater than **number2**

Output statement executes only if **number1** is less than or equal to **number2**

Output statement executes only if **number1** is greater than or equal to **number2**

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 2 of 3.)