Fundamentals of Java Programs

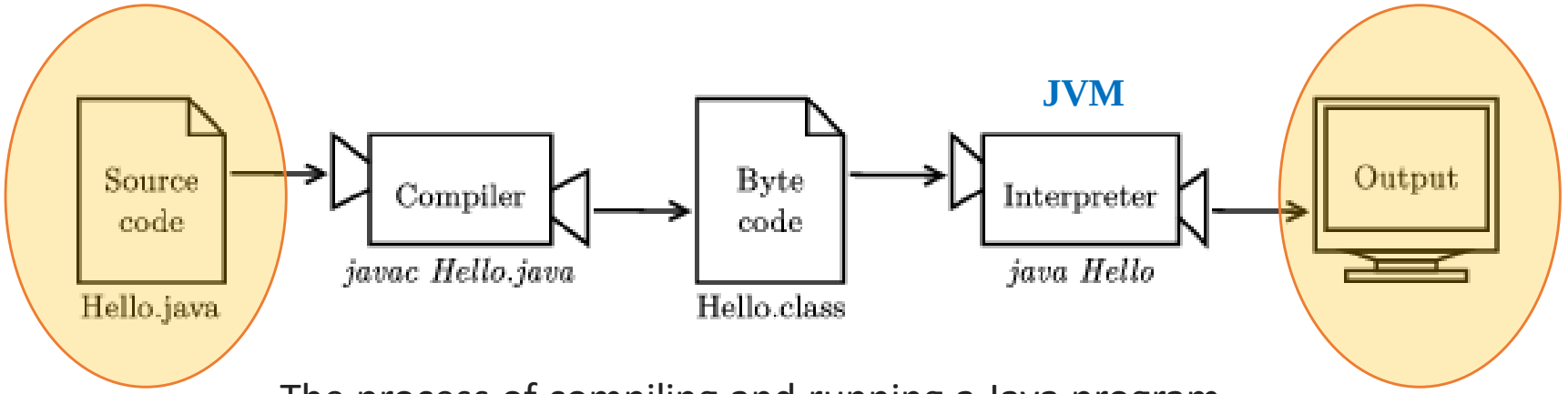
(Part 1)

Reference

- Readings
  - Chapter 2: Introduction to Java Applications; Input/Output and Operators
  - Chapter 3: Control Statements: Part I; Assignment, ++ and -- Operators

Introduction

- Java application programming
  - Java application: a computer program that executes when you use the java command to launch the Java Virtual Machine (JVM).



The process of compiling and running a Java program.

https://trinket.io/java/78526d761b

First Program

```
1  // Fig. 2.1: Welcome1.java
2  // Text-printing program.
3
4  public class Welcome1
5  {
6     // main method begins execution of Java application
7     public static void main(String[] args)
8     {
9        System.out.println("Welcome to Java Programming!");
10    } // end method main
11 } // end class Welcome1
```

```
Welcome to Java Programming!
```

Comments

Class Name

main Method

Statement

Notice that the output does not include the quotation marks.

Fig. 2.1 | Text-printing program.

First Program

- A **statement** is a line of code that performs a basic operation.
- In the previous example, the line is a **print statement** that displays a message on the screen.

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

- Instructs the computer to perform an action
  - Display the characters contained between the double quotation marks.
  - Together, the quotation marks and the characters between them are a string also known as a character string or a string literal.
  - White-space characters in strings are *not* ignored by the compiler.
  - Strings *cannot* span multiple lines of code.

First Program

- A **method** is a named sequence of statements. This program defines one method named main.

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

- Starting point of every Java application.
- Parentheses after the identifier `main` indicate that it's a program building block called a method.
- Java class declarations normally contain one or more methods.
- `main` must be defined as shown; otherwise, the JVM will not execute the application.
- Methods perform tasks and can return information when they complete their tasks.
- Keyword `void` indicates that this method will not return any information.

First Program

Class declaration

```
public class Welcome1
```

- Every Java program consists of at least one class that you define.
- `class` keyword introduces a class declaration and is immediately followed by the class name.
- Keywords (Appendix C) are reserved for use by Java and are always spelled with all lowercase letters.
- The name of the class must match the name of the file it is in, so this class must be in a file named Welcome1.java
- By convention, begin with a capital letter and capitalize the first letter of each word they include (e.g., SampleClassName).
- A class name is an identifier—a series of characters consisting of letters, digits, underscores (_) and dollar signs ($) that does not begin with a digit and does not contain spaces.

First Program

Class Body

- A left brace, `{`, begins the body of every class declaration.
- A corresponding right brace, `}`, must end each class declaration.

https://trinket.io/java/485ba2116c

First Program

Commenting Your Programs

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

- `//` indicates that the line is a comment.
- Used to document programs and improve their readability.
- Compiler ignores comments.
- A comment that begins with `//` is an end-of-line comment—it terminates at the end of the line on which it appears.

- Traditional comment, can be spread over several lines as in

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

- This type of comment begins with /* and ends with */.
- All text between the delimiters is ignored by the compiler.

- Blank lines, space characters and tabs are known as white space (or whitespace).
  - They are ignored by the compiler. But make programs easier to read.

Common Programming Error 2.1

Forgetting one of the delimiters of a traditional or Java-doc comment is a syntax error. A ***syntax error*** *occurs when the compiler encounters code that violates Java's language rules (i.e., its syntax). These rules are similar to a natural language's grammar rules specifying sentence structure. Syntax errors are also called* ***compiler errors***, ***compile-time errors*** *or* ***compilation errors***, *because the compiler detects them when compiling the program. When a syntax error is encountered, the compiler issues an error message. You must eliminate all compilation errors before your program will compile properly.*

Common Programming Error 2.2

A compilation error occurs if a `public` *class's filename is not exactly same name as the class (in terms of both spelling and capitalization) followed by the* `.java` *extension.*

Common Programming Error 2.3

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

Error-Prevention Tip 2.3

When the compiler reports a syntax error, it may not be on the line that the error message indicates. First, check the line for which the error was reported. If you don't find an error on that line, check several preceding lines.

https://trinket.io/java/8d74ae0d59

First Program

- **Displaying strings**
- Phrases that appear in quotation marks are called **strings**, because they contain a sequence of "characters" strung together. Characters can be letters, numbers, punctuation marks, symbols, spaces, tabs, etc.

- **System.out.println** appends a special character, called a **newline**, that moves to the beginning of the next line. If you don't want a newline at the end, you can use print instead of println:

System Class

- System.out object
  - Standard output object.
  - Allows a Java application to display information in the command window from which it executes.
- System.out.println method
  - Displays (or prints) a line of text in the command window.
  - The string in the parentheses the argument to the method.
  - Positions the output cursor at the beginning of the next line in the command window.
- System.out's method print displays a string.
- Unlike println, print does not position the output cursor at the beginning of the next line in the command window.
  - The next character the program displays will appear immediately after the last character that print displays.

https://trinket.io/java/6b7b5158a4

First Program

```
1  // Fig. 2.3: Welcome2.java
2  // Printing a line of text with multiple statements.
3
4  public class Welcome2
5  {
6     // main method begins execution of Java application
7     public static void main(String[] args)
8     {
9        System.out.print("Welcome to ");
10       System.out.println("Java Programming!");
11    } // end method main
12 } // end class Welcome2
```

```
Welcome to Java Programming!
```

Fig. 2.3 | Printing a line of text with multiple statements.

https://trinket.io/java/56a156837e

First Program

- **Escape Sequence**
- it is possible to display multiple lines of output in just one line of code. You just must tell Java where to put the line breaks.
- The backslash (\) is called an escape character.
  - Indicates a “special character”
- Backslash is combined with the next character to form an escape sequence—\n represents the newline character.

```
1  // Fig. 2.4: Welcome3.java
2  // Printing multiple lines of text with a single statement.
3
4  public class Welcome3
5  {
6     // main method begins execution of Java application
7     public static void main(String[] args)
8     {
9        System.out.println("Welcome\nto\nJava\nProgramming!");
10    } // end method main
11 } // end class Welcome3
```

```
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.

Declaring variables

- Programs remember numbers and other data in the computer's memory and access that data through program elements called variables.
- Every variable has a name, a type, a size (in bytes) and a value.
- **A variable** is a named location that stores a value. Values may be numbers, text, images, sounds, and other types of data. To store a value, you first must declare a variable.

```
String message;
```

- This statement is a declaration, because it declares that the variable named **message** has the type **String**.
- Each variable has a type that determines what kind of values it can store. For example, the **int** type can store integers, and the **char** type can store characters.
- To declare an integer variable named x, you simply type: `int x;`

Declaring variables

- Variable name is an identifier—a series of characters consisting of letters, digits, underscores (_) and dollar signs ($) that does not begin with a digit and does not contain spaces.
- Java is case sensitive—uppercase and lowercase letters are distinct—so `a1` and `A1` are different (but both valid) identifiers.

- For declaring multiple variables with the same type on one line: hour and minute are both integers. Note that each declaration statement ends with a semicolon.

```
int hour, minute;
```

- You can use any name you want for a variable. But there are about 50 reserved words, called keywords, that you are not allowed to use as variable names. These words include **public, class, static, void, and int**, which are used by the compiler to analyze the structure of the program.

Built-in Data Type

- **int**: hold integer values [whole numbers such as 72, –1127 and 0].
- **float** and **double**: hold real number [contain decimal points, such as 3.4, 0.0 and –11.19].
- **char** : hold character data between single quotation mark [such as an uppercase letter (e.g., A), a digit (e.g., 7), a special character (e.g., *or %) or an escape sequence (e.g., the newline character, \n)].
- **string** : hold sequence of character between double quotation mark [such as “Hello World”, “Hi”, “3.14”, “true”, “2020”].
- **boolean** : hold true or false.

Assignment

- Now that we have declared variables, we want to use them to store values. We do that with an assignment statement.

```
message = "Hello!"; // give message the value "Hello!"
hour = 11;          // assign the value 11 to hour
minute = 59;        // set minute to 59
```

- When you **declare a variable**, you create a named storage location.
- When you **make an assignment** to a variable, you update its value.
- When a new value is placed into a variable, the new value replaces the previous value (if any)

Assignment

- Generally, a variable must have the same type as the value you assign to it. For example, you cannot store a string in minute or an integer in message. We will see some examples that seem to break this rule, but we'll get to that later.

```
message = "123"; // legal
message = 123;   // not legal
```

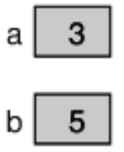
- Variables must be initialized (assigned for the first time) before they can be used. You can declare a variable and then assign a value later, as in the previous example. You can also declare and initialize on the same line:

```
String message = "Hello!";
int hour = 11;
int minute = 59;
```

Assignment

- Because Java uses the = symbol for assignment, it is tempting to interpret the statement a = b as a statement of equality. It is not!
  - In Java a = 7; is a legal assignment statement, but 7 = a; is not.
  - The left side of an assignment statement has to be a variable name (storage location).
- In Java, an assignment statement can make two variables equal, but they don't have to stay that way.

```
int a = 5; int b = a;       // a and b are now equal
a = 3;                      // a and b are no longer equal
```



Printing Variables

- You can display the value of a variable using print or println. The following statements declare a variable named firstLine, assign it the value "Hello, again!", and display that value.

```
String firstLine = "Hello, again!";
System.out.println(firstLine);
```

```
Output: Hello, again!
```

- To display the *name* of a variable, you have to put it in quotes.

```
System.out.print("The value of firstLine is ");
System.out.println(firstLine);
```

```
Output: The value of firstLine is Hello, again!
```

Printing Variables

- **System.out.printf** method displays formatted data
- Multiple method arguments are placed in **a comma-separated list.**
- Method printf's first argument is a **format string**
- May consist of **fixed text** and **format specifiers**.
  - Fixed text is output as it would be by print or println.
  - Each format specifier is a placeholder for a value and specifies the type of data to output.
  - Format specifiers begin with a percent sign (%) and are followed by a character that represents the data type. **Ex:** %s is a placeholder for a string.

%d	Decimal integer	12345
%08d	Padded with zeros, at least 8 digit wide	00012345
%f	Floating-point	6.789000
%.2f	Rounded to 2 decimal places	6.79

```
1  // Fig. 2.6: Welcome4.java
2  // Displaying multiple lines with method System.out.printf.
3
4  public class Welcome4
5  {
6     // main method begins execution of Java application
7     public static void main(String[] args)
8     {
9        System.out.printf("%s%n%s%n",
10          "Welcome to", "Java Programming!");
11    } // end method main
12 } // end class Welcome4
```

```
Welcome to
Java Programming!
```

Fig. 2.6 | Displaying multiple lines with method `System.out.printf`.

Arithmetic Operations

- **Operators** are symbols that represent simple computations. For example, the addition operator is **+**, subtraction is **-**, multiplication is ***, and division is **/**.
- An **expression**, which represents a single value to be computed. When the program runs, each variable is replaced by its current value, and then the operators are applied. The values operators work with are called **operands**.

```
int hour = 11;
int minute = 59;
System.out.print("Number of minutes since midnight: ");
System.out.println(hour * 60 + minute);
```

```
Output: Number of minutes since midnight: 719
```

Built-in Data Type Operations

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.

type	*set of values*	*common operators*	*sample literal values*
`int`	integers	`+ - * / %`	`99 12 2147483647`
`double`	floating-point numbers	`+ - * /`	`3.14 2.5 6.022e23`
`boolean`	boolean values	`&& || !`	`true false`
`char`	characters		`'A' '1' '%' '\n'`
`String`	sequences of characters	`+`	`"AB" "Hello" "2.5"`

Ex: https://trinket.io/java/618528ea37 (omit import statement)

The Scanner Class

- Scanner is a class that provides methods for inputting words, numbers, and other data. Scanner is provided by java.util, which is a package that contains classes so useful they are called "utility classes". Before you can use Scanner, you have to import it like t

```
import java.util.Scanner;
```

- This import statement tells the compiler that when you say Scanner, you mean the one defined in java.util. Import statements can't be inside a class definition. By convention, they are usually at the beginning of the file.

- Variable declaration from Scanner class:

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

`Scanner` methods:

int	nextInt()
short	nextShort()
double	nextDouble()
float	nextFloat()
char	next().chatAt()
string	nextLine()

Example of Adding Integers

```
1  // Fig. 2.7: Addition.java
2  // Addition program that inputs two numbers then displays their sum.
3  import java.util.Scanner; // program uses class Scanner
4
5  public class Addition
6  {
7     // main method begins execution of Java application
8     public static void main(String[] args)
9     {
10       // create a Scanner to obtain input from the command window
11       Scanner input = new Scanner(System.in);
12
13       int number1; // first number to add
14       int number2; // second number to add
15       int sum; // sum of number1 and number2
16
17       System.out.print("Enter first integer: "); // prompt
18       number1 = input.nextInt(); // read first number from user
19
20       System.out.print("Enter second integer: "); // prompt
21       number2 = input.nextInt(); // read second number from user
22
23       sum = number1 + number2; // add numbers, then store total in sum
24
25       System.out.printf("Sum is %d%n", sum); // display sum
26    } // end method main
27 } // end class Addition
```

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

Fig. 2.7 | Addition program that inputs two numbers then displays their sum. (Part

Error Types

- **Compile-time** errors occur when you violate the syntax rules of the Java language.
  - https://trinket.io/java/35ddb7b272 (Remove ; from line 5)

- **Run-time error**, so-called because it does not appear until after the program has started running.
  - Ex: divide by zero

- **logic error**. If your program has a logic error, it will compile and run without generating error messages, but it will not do the right thing.
  - https://trinket.io/java/35ddb7b272

Integer Division

- The arithmetic operators are binary operators because they each operate on two operands.
- Integer division yields an integer quotient.
  - Any fractional part in integer division is simply *truncated* (i.e., *discarded*)—no *rounding* occurs.
- The remainder operator, %, yields the remainder after division.

```
int minute = 59;
System.out.print("Fraction of the hour that has passed: ");
System.out.println(minute / 60);
```

```
Output: Fraction of the hour that has passed: 0
```

Floating-point numbers

- A more general solution is to use **floating-point numbers**, which can represent fractions as well as integers.
- In Java, the default floating-point type is called **double**, which is short for double-precision.
- You can create double variables and assign values to them using the same syntax we used for the other types:

```
double minute = 59.0;
System.out.print("Fraction of the hour that has passed: ");
System.out.println(minute / 60.0);
```

Output: Fraction of the hour that has passed: 0.9833333333333333

Conversion

- The following is illegal because the variable on the left is an int and the value on the right is a double:

```
int x = 1.1; // compiler error
```

- In many cases Java automatically converts from one type to another if promotion is possible:

```
double y = 1; // legal, but bad style
```

- The expression on the right divides two integers, so Java does integer division, which yields the int value 0. Converted to double, the value assigned to y is 0.0.

```
double y = 1 / 3; // common mistake (logic error)
```

Conversion

- The simplest way to convert a floating-point value to an integer is to use a type cast, so called because it “casts” a value from one type to another. The syntax for type casting is to put the name of the type in parentheses and use it as an operator.
- The casting used to convert from double to int to avoid compilation error “incompatible types: possible lossy conversion from double to int”
- The value of x and y in the following code did not change they still contain a double value in the memory.

```
 public static void main(String[] args)
{ double x=3.5;
  double y=2.3;
  int z=(int)x+(int)y;
  System.out.print("Sum="+z);}
```

Output:

Sum = 5

Conversion

```
public static void main(String [] args)
{
  int total =1000;
  int counter=15;
  double average;

  average=total/counter;
  System.out.println("Average without cast="+ average);

  average=(double)total/counter;
  System.out.println("Average with cast left operand="+ average);

  average=total/(double)counter;
  System.out.println("Average with cast right operand="+ average);

  average=(double)total/(double)counter;
  System.out.println("Average with cast both operand="+ average);
}
```

Output

```
Average without cast=66.0
Average with cast left operand=66.66666666666667
Average with cast right operand=66.66666666666667
Average with cast both operand=66.66666666666667
```

Conversion

- The previous code use casting from int to double to avoid ***losing data*** “when performing arithmetic operation such as division.
- As we can see from the first operation when we did not use casting the result lost its fraction part.
- The value of total and counter in the following code did not change they still contain an integer value in the memory.

Type	Valid promotions
`double`	None
`float`	`double`
`long`	`float` or `double`
`int`	`long`, `float` or `double`
`char`	`int`, `long`, `float` or `double`
`short`	`int`, `long`, `float` or `double` (but not `char`)
`byte`	`short`, `int`, `long`, `float` or `double` (but not `char`)
`boolean`	None (`boolean` values are not considered to be numbers in Java)

Fig. 5.4 | Promotions allowed for primitive types.

Common Programming Error 5.8

Casting a primitive-type value to another primitive type may change the value if the new type is not a valid promotion. For example, casting a floating-point value to an integer value may introduce truncation errors (loss of the fractional part) into the result.

Rules of Operator Precedence

- Parentheses are used to group terms in expressions in the same manner as in algebraic expressions.
- If an expression contains nested parentheses, the expression in the innermost set of parentheses is evaluated first.
- As in algebra, it's acceptable to place unnecessary parentheses in an expression to make the expression clearer

Operators						Associativity	Type
++	--					right to left	unary postfix
++	--	+	-	(*type*)		right to left	unary prefix
*	/	%				left to right	multiplicative
+	-					left to right	additive
<	<=	>	>=			left to right	relational
==	!=					left to right	equality
?:						right to left	conditional
=	+=	-=	*=	/=	%=	right to left	assignment

Fig. 3.14 | Precedence and associativity of the operators discussed so far.

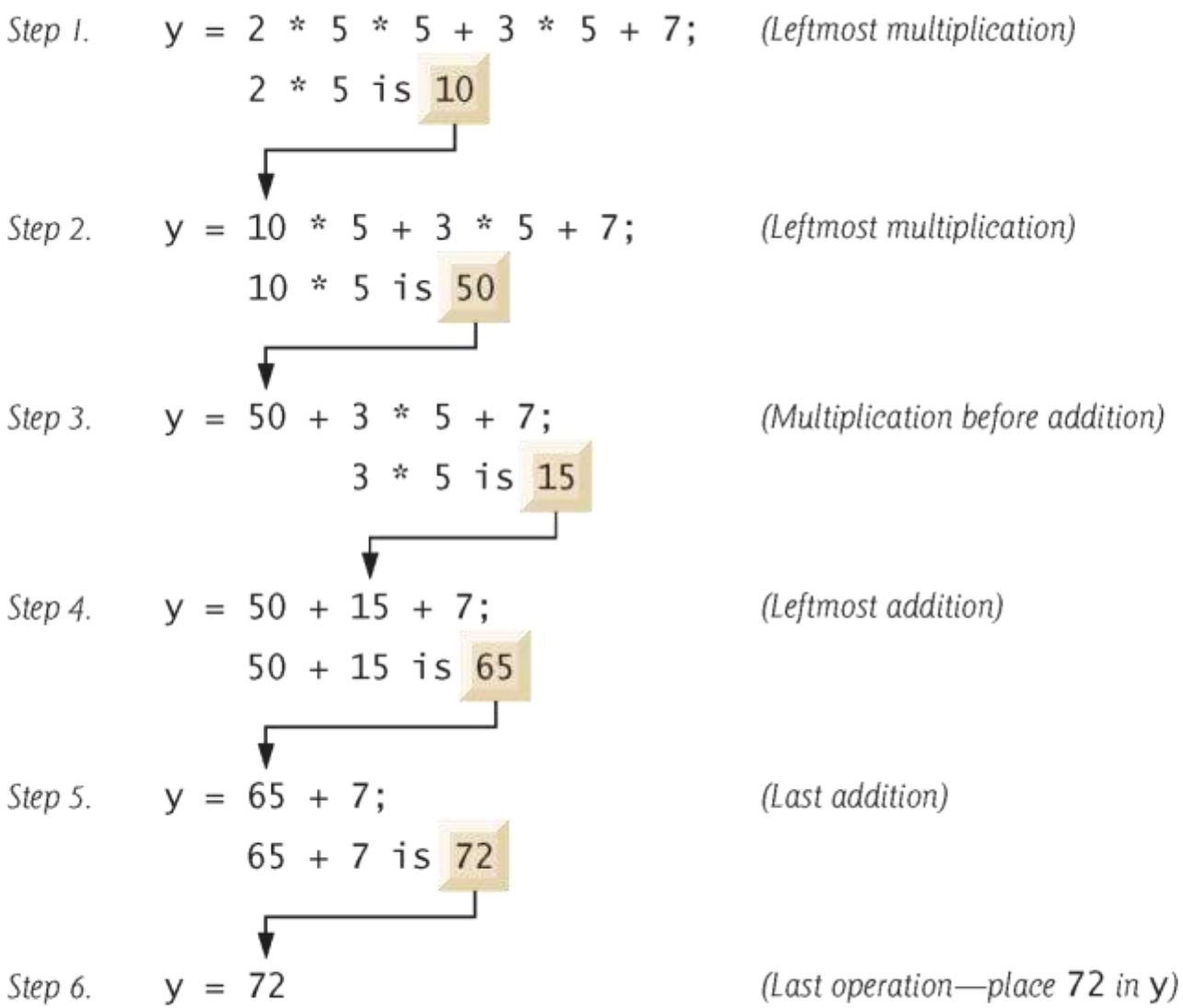


Fig. 2.13 | Order in which a second-degree polynomial is evaluated.

Increment and Decrement Operators

- Unary increment operator, ++, adds one to its operand
- Unary decrement operator, --, subtracts one from its operand
- An increment or decrement operator that's prefixed to (placed before) a variable is referred to as the prefix increment or prefix decrement operator, respectively.
- An increment or decrement operator that's postfixed to (placed after) a variable is referred to as the postfix increment or postfix decrement operator, respectively.

```
1  // Fig. 3.13: Increment.java
2  // Prefix increment and postfix increment operators.
3
4  public class Increment
5  {
6     public static void main(String[] args)
7     {
8        // demonstrate postfix increment operator
9        int c = 5;
10       System.out.printf("c before postincrement: %d%n", c); // prints 5
11       System.out.printf("    postincrementing c: %d%n", c++); // prints 5
12       System.out.printf(" c after postincrement: %d%n", c); // prints 6
13
14       System.out.println(); // skip a line
15
16       // demonstrate prefix increment operator
17       c = 5;
18       System.out.printf(" c before preincrement: %d%n", c); // prints 5
19       System.out.printf("     preincrementing c: %d%n", ++c); // prints 6
20       System.out.printf("  c after preincrement: %d%n", c); // prints 6
21    }
22 } // end class Increment
```

Fig. 3.13 | Prefix increment and postfix increment operators. (Part 1 of 2.)

```
c before postincrement: 5
    postincrementing c: 5
 c after postincrement: 6

 c before preincrement: 5
     preincrementing c: 6
  c after preincrement: 6
```

Fig. 3.13 | Prefix increment and postfix increment operators. (Part 2 of 2.)

Operation on Strings

- The + operator works with strings, but it might not do what you expect. For strings, the + operator performs concatenation, which means joining end-to-end.
- When more than one operator appears in an expression, they are evaluated according to order of operations.

```
System.out.println(1 + 2 + "Hello"); // the output is 3Hello
System.out.println("Hello" + 1 + 2); // the output is Hello12
```

Compound Assignment Operators

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to `c`
`-=`	`d -= 4`	`d = d - 4`	1 to `d`
`*=`	`e *= 5`	`e = e * 5`	20 to `e`
`/=`	`f /= 3`	`f = f / 3`	2 to `f`
`%=`	`g %= 9`	`g = g % 9`	3 to `g`

Fig. 3.11 | Arithmetic compound assignment operators.

Math Methods

- The Java library includes a Math class that provides common mathematical operations. Math is in the java.lang package, so you don't have to import it. You can use, or invoke, Math methods like this:

```
double root = Math.sqrt(17.0);
double angle = 1.5;
double height = Math.sin(angle);
```

Software Engineering Observation 5.4

Class `Math` *is part of the* `java.lang` *package, which is implicitly imported by the compiler, so it's not necessary to import class* `Math` *to use its methods.*

Method	Description	Example

`abs(`*x*`)`	absolute value of *x*	`abs(23.7)` is `23.7`
		`abs(0.0)` is `0.0`
		`abs(-23.7)` is `23.7`

`ceil(`*x*`)`	rounds *x* to the smallest integer not less than *x*	`ceil(9.2)` is `10.0` `ceil(-9.8)` is `-9.0`
`cos(`*x*`)`	trigonometric cosine of *x* (*x* in radians)	`cos(0.0)` is `1.0`
`exp(`*x*`)`	exponential method e^x	`exp(1.0)` is `2.71828` `exp(2.0)` is `7.38906`
`floor(`*x*`)`	rounds *x* to the largest integer not greater than *x*	`floor(9.2)` is `9.0` `floor(-9.8)` is `-10.0`
`log(`*x*`)`	natural logarithm of *x* (base *e*)	`log(Math.E)` is `1.0` `log(Math.E * Math.E)` is `2.0`
`max(`*x*`, `*y*`)`	larger value of *x* and *y*	`max(2.3, 12.7)` is `12.7` `max(-2.3, -12.7)` is `-2.3`
`min(`*x*`, `*y*`)`	smaller value of *x* and *y*	`min(2.3, 12.7)` is `2.3` `min(-2.3, -12.7)` is `-12.7`
`pow(`*x*`, `*y*`)`	*x* raised to the power *y* (i.e., x^y)	`pow(2.0, 7.0)` is `128.0` `pow(9.0, 0.5)` is `3.0`
`sin(`*x*`)`	trigonometric sine of *x* (*x* in radians)	`sin(0.0)` is `0.0`
`sqrt(`*x*`)`	square root of *x*	`sqrt(900.0)` is `30.0`
`tan(`*x*`)`	trigonometric tangent of *x* (*x* in radians)	`tan(0.0)` is `0.0`

Fig. 5.2 | `Math` class methods. (Part 2 of 2.)