

## 2.5 Another Application: Adding Integers

Our next application reads (or inputs) two **integers** (whole numbers, such as **–22**, **7**, **0** and **1024**) typed by a user at the keyboard, computes their sum and displays it. This program must keep track of the numbers supplied by the user for the calculation later in the program. 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. In the sample output, we use bold text to identify the user’s input (i.e., **45** and **72**). As per our convention in prior programs, lines 1–2 state the figure number, filename and purpose of the program.

---

```
1  // Fig. 2.7: Addition.java
2  // Addition program that inputs two numbers then
3  import java.util.Scanner; // program uses class
4
5  public class Addition {
6      // main method begins execution of Java appli
7      public static void main(String[] args) {
8          // create a Scanner to obtain input from t
9          Scanner input = new Scanner(System.in);
10
11         System.out.print("Enter first integer: ");
12         int number1 = input.nextInt(); // read fir
13
14         System.out.print("Enter second integer: ")
15         int number2 = input.nextInt(); // read sec
```

```
16
17     int sum = number1 + number2; // add number
18
19     System.out.printf("Sum is %d%n", sum); //
20 } // end method main
21 } // 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.

## 2.5.1 import Declarations

A great strength of Java is its rich set of predefined classes that you can *reuse* rather than “reinventing the wheel.” These 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)**. Line 3

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

is an **import declaration** that helps the compiler locate a class that's used in this program. It indicates that the program uses the predefined **Scanner** class (discussed shortly) from the package named **java.util**. The compiler then ensures that you use the class correctly.



## Common Programming Error 2.5

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



## Common Programming Error 2.6

*Forgetting to include an **import** declaration for a class that must be imported results in a compilation error containing a message such as “cannot find symbol.” When this occurs, check that you provided the proper **import** declarations and that the names in them are correct, including proper capitalization.*

## 2.5.2 Declaring and Creating a Scanner to Obtain User Input from the Keyboard

A **variable** is a location in the computer’s memory where a value can be stored for use later in a program. All Java variables *must* be declared with a **name** and a **type** *before* they can be used. A variable’s *name* enables the program to access the variable’s *value* in memory. A variable name can be any valid identifier—again, a series of characters consisting of letters, digits, underscores (\_) and dollar signs (\$) that does *not* begin with a digit and does *not* contain spaces. A variable’s *type* specifies what kind of information is stored at that location in memory. Like other statements, declaration statements end with a semicolon (;).

Line 9 of `main`

---

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



is a **variable declaration statement** that specifies the *name* (`input`) and *type* (`Scanner`) of a variable that’s used in this program. A `Scanner` (package `java.util`) enables a program to read data (e.g., numbers and strings) for use in a program. The 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 `=` in line 9 indicates that `Scanner` variable `input` should be **initialized** (i.e., prepared for use in the program) in its declaration with the result of the expression to the right of the equals sign—`new Scanner(System.in)`. This expression uses the `new` keyword to create a `Scanner` object that reads characters typed by the user at the keyboard. The **standard input object**, `System.in`, enables applications to read *bytes* of data typed by the user. The `Scanner` translates these bytes `into` types (like `ints`) that can be used in a program.



## Good Programming Practice 2.8

*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 associated documentation or creating and viewing an excessive number of comments).*



## Good Programming Practice 2.9

*By convention, variable-name identifiers use the camel-case naming convention with a lowercase first letter—for example, `firstNumber`.*

## 2.5.3 Prompting the User for Input

Line 11

---

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

uses `System.out.print` to display the message "Enter first integer: ". This message is called a **prompt** because it directs the user to take a specific action. We use method `print` here rather than `println` so that the user's input appears on the same line as the prompt. Recall from [Section 2.2](#) that identifiers starting with capital letters typically represent class names. Class `System` is part of package `java.lang`.



## Software Engineering Observation 2.1

*By default, package `java.lang` is imported in every Java program; thus, classes in `java.lang` are the only ones in*

the Java API that do not require an `import` declaration.

## 2.5.4 Declaring a Variable to Store an Integer and Obtaining an Integer from the Keyboard

The variable declaration statement in line 12

---

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



declares that variable `number1` holds data of type `int`—that is, *integer* values, which are whole numbers such as `72`, `-1127` and `0`. The 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; however, for readability, you can place underscores in numbers. So `60_000_000` represents the `int` value `60,000,000`.

Some other types of data are `float` and `double`, for holding real numbers, and `char`, for holding character data. Real numbers contain decimal points, such as in `3.4`, `0.0` and `-11.19`. Variables of type `char` represent individual characters, 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 tab character, `\t`). The types `int`, `float`, `double`

and `char` are called **primitive types**. Primitive-type names are keywords and must appear in all lowercase letters.

Appendix D summarizes the characteristics of the eight primitive types (`boolean`, `byte`, `char`, `short`, `int`, `long`, `float` and `double`).

The `=` in line 12 indicates that `int` variable `number1` should be initialized in its declaration with the result of `input.nextInt()`. This uses the `Scanner` object `input`'s `nextInt` method to obtain an `integer` from the user at the keyboard. At this point the program *waits* for the user to type the number and press the *Enter* key to submit the number to the program.

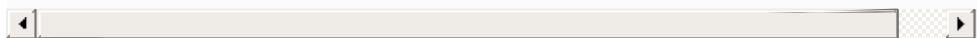
Our program assumes that the user enters a valid integer value. If not, a logic error will occur and the program will terminate. Chapter 11, Exception Handling: A Deeper Look, discusses how to make your programs more robust by enabling them to handle such errors. This is also known as making your program *fault tolerant*.

## 2.5.5 Obtaining a Second Integer

Line 14

---

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



*prompts* the user to enter the second integer. Line 15

---

```
int number2 = input.nextInt(); // read second number
```



declares the `int` variable `number2` and initializes it with a second `integer` read from the user at the keyboard.

## 2.5.6 Using Variables in a Calculation

Line 17

---

```
int sum = number1 + number2; // add numbers then stor
```



declares the `int` variable `sum` and initializes it with the result of `number1 + number2`. When the program encounters the addition operation, it performs the calculation using the values stored in the variables `number1` and `number2`.

In the preceding statement, the addition operator is a **binary operator**, because it has *two operands*—`number1` and `number2`. Portions of statements that contain calculations are called **expressions**. In fact, an expression is any portion of a statement that has a *value*. The value of the expression `number1 + number2` is the *sum* of the numbers. Similarly, the value of the expression `input.nextInt()` (lines 12

and 15) is the integer typed by the user.



## Good Programming Practice 2.10

*Place spaces on either side of a binary operator for readability.*

### 2.5.7 Displaying the Calculation Result

After the calculation has been performed, line 19

---

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

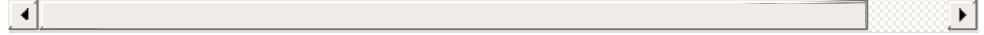
uses method `System.out.printf` to display the `sum`. The format specifier `%d` is a *placeholder* for an `int` value (in this case the value of `sum`)—the letter `d` stands for “decimal integer.” The remaining characters in the format string are all fixed text. So, method `printf` displays “Sum is ”, followed by the value of `sum` (in the position of the `%d` format specifier) and a newline.

Calculations also can be performed *inside* `printf` statements. We could have combined the statements at lines 17

and 19 into the statement

---

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



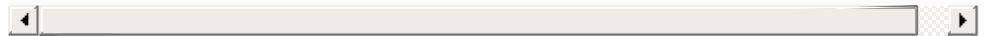
The parentheses around the expression `number1 + number2` are optional—they’re included to emphasize that the value of the *entire* expression is output in the position of the `%d` format specifier. Such parentheses are said to be **redundant**.

## 2.5.8 Java API Documentation

For each new Java API class we use, we indicate the package in which it’s located. This information helps you locate descriptions of each package and class in the Java API documentation. A web-based version of this documentation can be found at

---

<http://docs.oracle.com/javase/8/docs/api/index.html>



You can download it from the Additional Resources section at

---

<http://www.oracle.com/technetwork/java/javase/downloa>



Appendix F shows how to use this documentation.

## 2.5.9 Declaring and Initializing Variables in Separate Statements

Each variable must have a value *before* you can use the variable in a calculation (or other expression). The variable declaration statement in line 12 both declared `number1` and initialized it with a value entered by the user.

Sometimes you declare a variable in one statement, then initialize in another. For example, line 12 could have been written in two statements as

---

```
int number1; // declare the int variable number1
number1 = input.nextInt(); // assign the user's input
```



The first statement declares `number1`, but does *not* initialize it. The second statement uses the **assignment operator**, `=`, to *assign* (that is, give) `number1` the value entered by the user. You can read this statement as “`number1` gets the value of `input.nextInt()`.” Everything to the *right* of the assignment operator, `=`, is always evaluated *before* the assignment is performed.