1a

Introduction to Java Applications

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

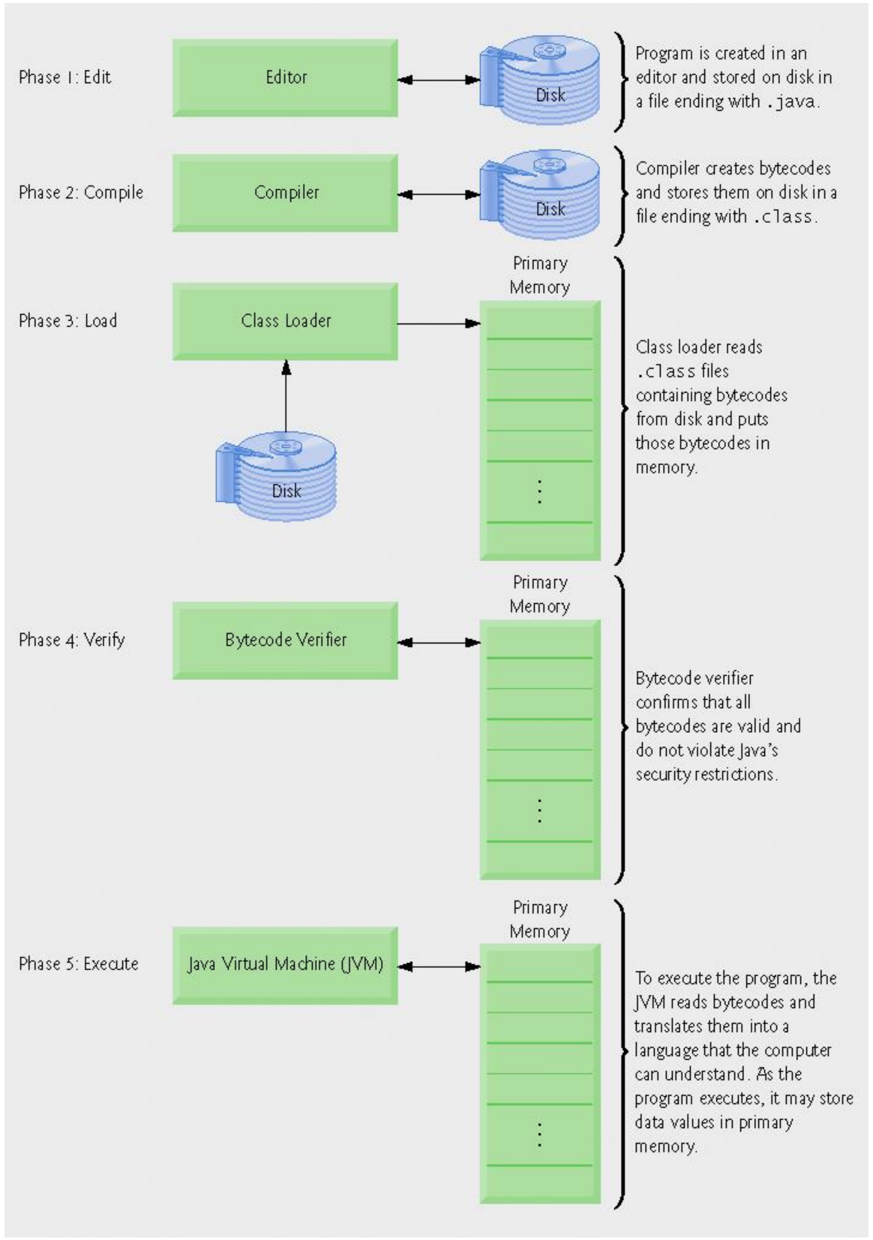

Phase 1: Edit
Editor
Disk
Program is created in an editor and stored on disk in a file ending with .java.
Phase 2: Compile
Compiler
Disk
Compiler creates bytecodes and stores them on disk in a file ending with .class.
Phase 3: Load
Class Loader
Primary Memory
Disk
Class loader reads .class files containing 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.

1.14 Notes about Java and Java How to Program, Seventh Edition

- Stresses clarity
- Portability
  - An elusive goal due to differences between compilers, JVMs and computers
  - Always test programs on all systems on which the programs should run

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.

Portability Tip 1.2

Although it is easier to write portable programs in Java than in other programming languages, differences between compilers, JVMs and computers can make portability difficult to achieve. Simply writing programs in Java does not guarantee portability.

Error-Prevention Tip 1.1

Always test your Java programs on all systems on which you intend to run them, to ensure that they will work correctly for their intended audiences.

Good Programming Practice 1.2

Read the documentation for the version of Java you are using. Refer to it frequently to be sure you are aware of the rich collection of Java features and are using them correctly.

Good Programming Practice 1.3

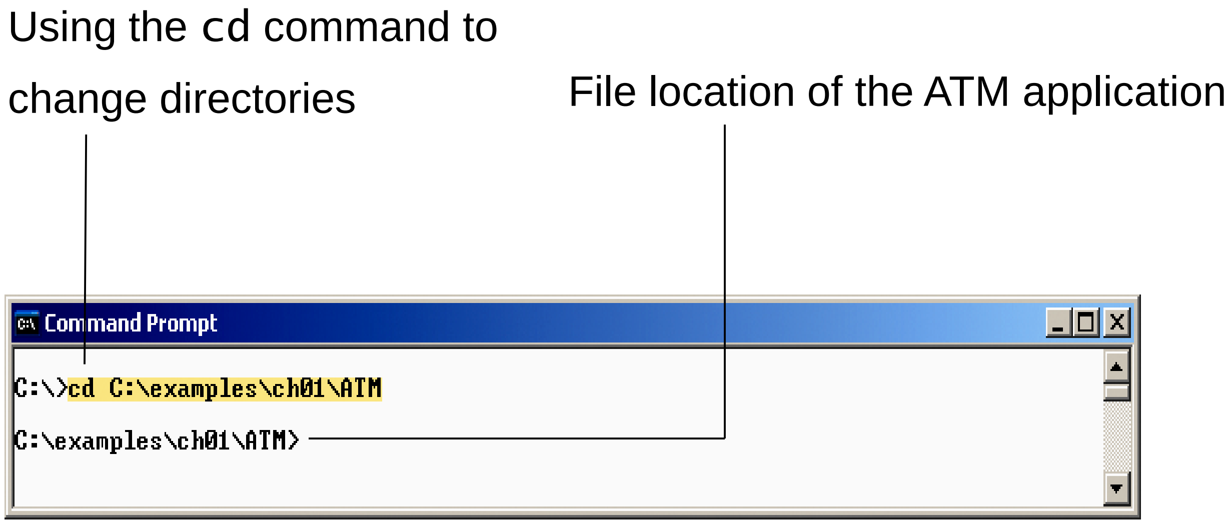
Your computer and compiler are good teachers. If, after carefully reading your Java documentation manual, you are not sure how a feature of Java works, experiment and see what happens. Study each error or warning message you get when you compile your programs (called *compile-time errors* or *compilation errors*), and correct the programs to eliminate these messages.

Software Engineering Observation 1.4

Some programmers like to read the source code for the Java API classes to determine how the classes work and to learn additional programming techniques.

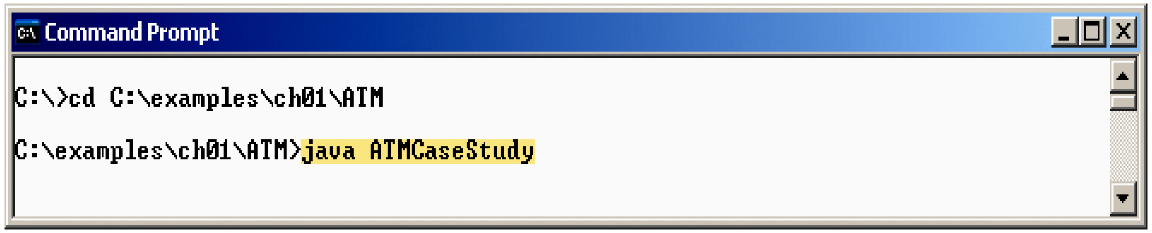
1.15 Test-Driving a Java Application

- **Test-driving the ATM application**
  - **Check system setup**
  - **Locate the ATM application (Fig. 1.2)**
  - **Run the ATM application (Fig. 1.3)**
  - **Enter an account number (Fig. 1.4)**
  - **Enter a PIN (Fig. 1.5)**
  - **View the account balance (Fig. 1.6)**
  - **Withdraw money from the account (Fig. 1.7)**
  - **Confirm that the account information has been updated (Fig. 1.8)**
  - **End the transaction (Fig. 1.9)**
  - **Exit the ATM application**
- **Additional applications (Fig. 1.10)**


Using the cd command to change directories
File location of the ATM application
Command Prompt

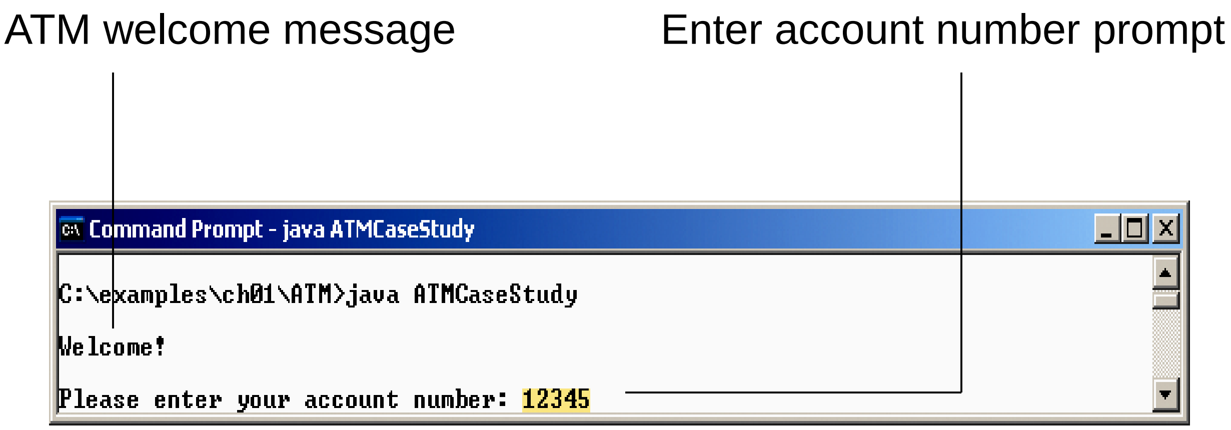
```
C:\>cd C:\examples\ch01\ATM

C:\examples\ch01\ATM>
```

Fig. 1.2 | **Opening a Windows XP *Command Prompt* and changing directories.**



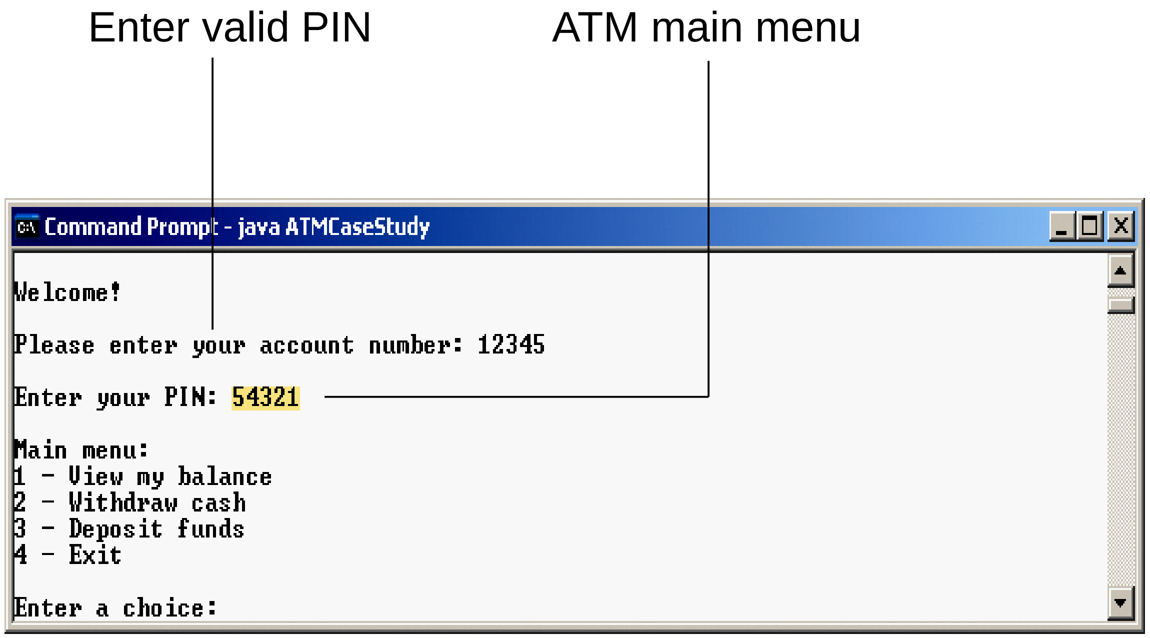
```
C:\>cd C:\examples\ch01\ATM

C:\examples\ch01\ATM>java ATMCaseStudy
```

Fig. 1.3 | Using the `java` command to execute the ATM application.


ATM welcome message
Enter account number prompt
Command Prompt - java ATMCaseStudy

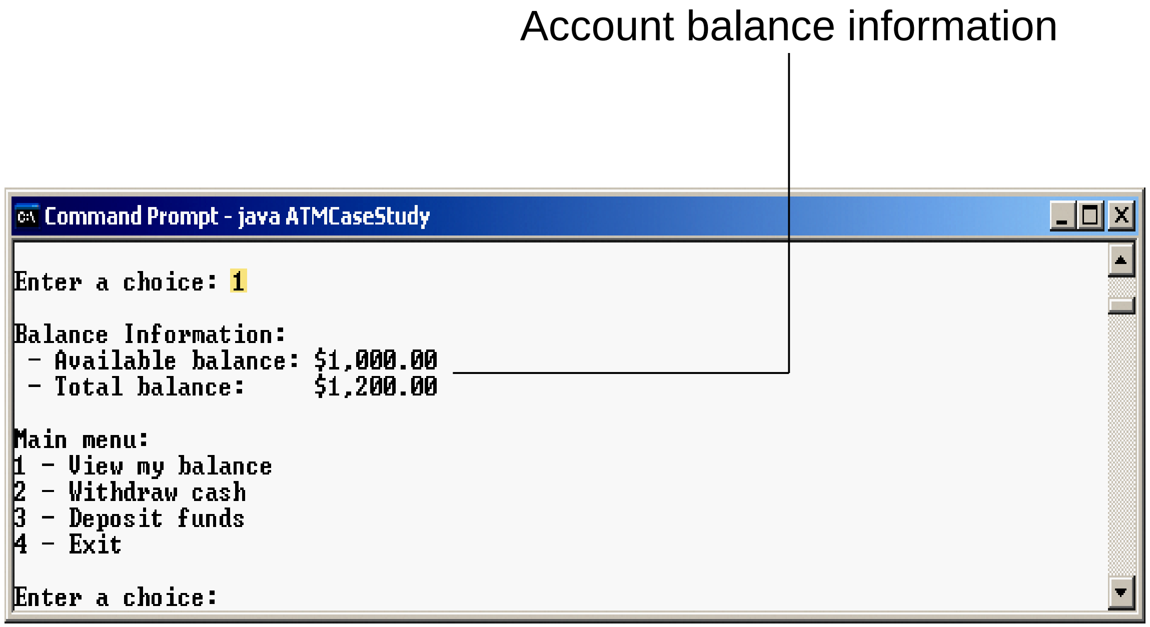
```
C:\examples\ch01\ATM>java ATMCaseStudy
Welcome!
Please enter your account number: 12345
```

Fig. 1.4 | **Prompting the user for an account number.**



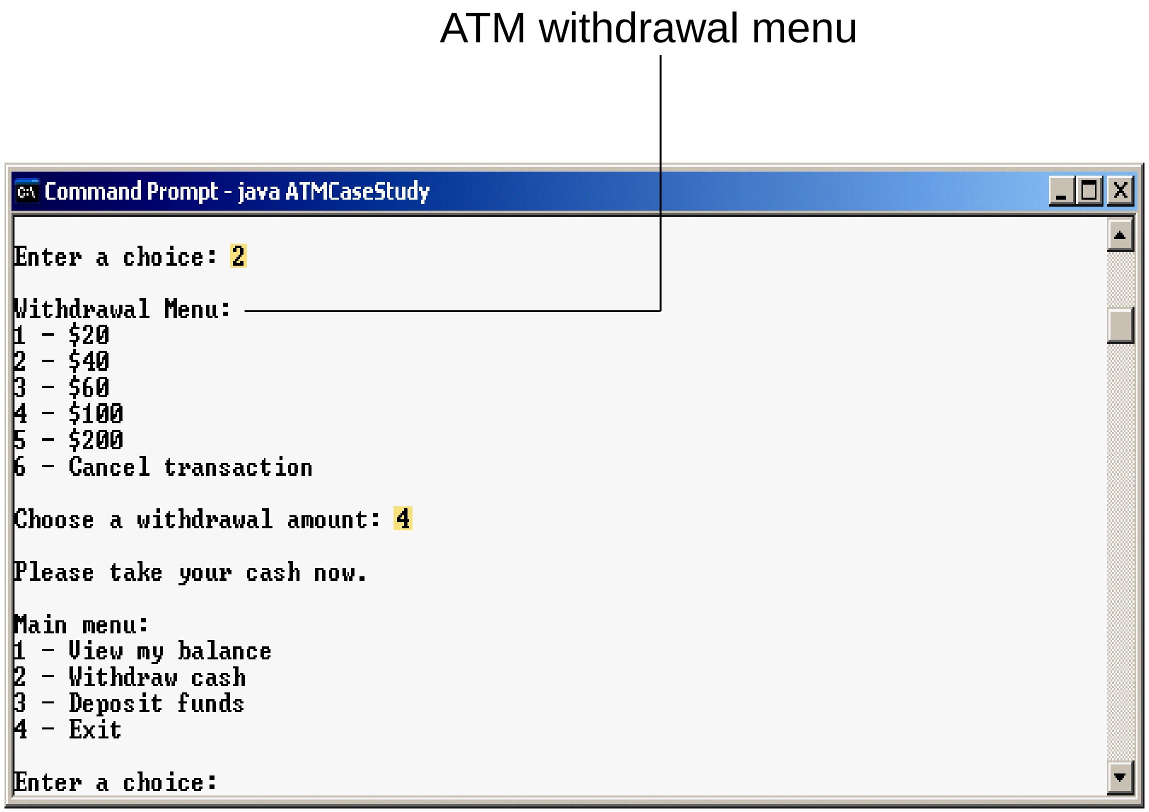
```
Welcome!

Please enter your account number: 12345

Enter your PIN: 54321

Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit

Enter a choice:
```

Fig. 1.5 | Entering a valid PIN number and displaying the ATM application's main menu.


Account balance information
Command Prompt - java ATMCaseStudy

```
Enter a choice: 1

Balance Information:
 - Available balance: $1,000.00
 - Total balance:     $1,200.00

Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit

Enter a choice:
```

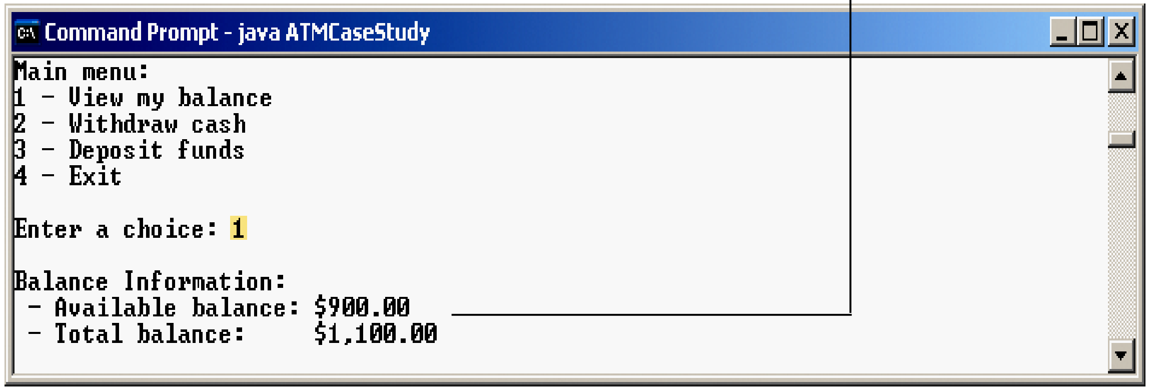
Fig. 1.6 | ATM application displaying user account balance information.



```
Enter a choice: 2

Withdrawal Menu:
1 - $20
2 - $40
3 - $60
4 - $100
5 - $200
6 - Cancel transaction

Choose a withdrawal amount: 4

Please take your cash now.

Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit

Enter a choice:
```

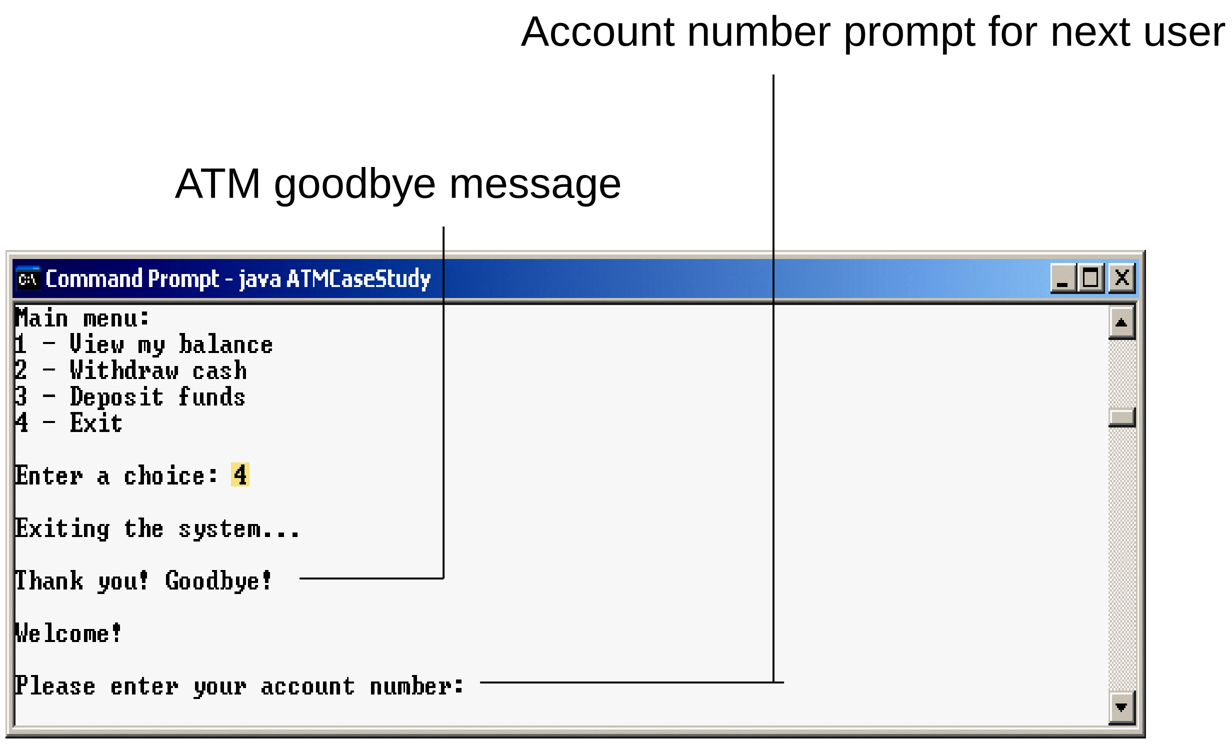
Fig. 1.7 | **Withdrawing money from the account and returning to the main menu.**

Confirming updated account balance information after withdrawal transaction



```
Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit

Enter a choice: 1

Balance Information:
 - Available balance: $900.00
 - Total balance:     $1,100.00
```

Fig. 1.8 | Checking new balance.


Account number prompt for next user
ATM goodbye message
Command Prompt - java ATMCaseStudy

```
Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit

Enter a choice: 4

Exiting the system...

Thank you! Goodbye!

Welcome!

Please enter your account number:
```

Fig. 1.9 | **Ending an ATM transaction session.**

Application Name	Chapter Location	Commands to Run
Tic-Tac-Toe	Chapters 8 and 24	`cd C:\examples\ch01\Tic-Tac-Toe` `java TicTacToeTest`
Guessing Game	Chapter 11	`cd C:\examples\ch01\GuessGame` `java GuessGame`
Logo Animator	Chapter 21	`cd C:\examples\ch01\LogoAnimator` `java LogoAnimator`
Bouncing Ball	Chapter 23	`cd C:\examples\ch01\BouncingBall` `java BouncingBall`

Fig. 1.10 | Examples of additional Java applications found in *Java How to Program, 6/e*.

1.16 Software Engineering Case Study: Introduction to Object Technology and the UML (Required)

- **Object orientation**
- **Unified Modeling Language (UML)**
  - **Graphical language that uses common notation**
  - **Allows developers to represent object-oriented designs**

1.16 Software Engineering Case Study (Cont.)

- Objects
  - Reusable software components that model real-world items
  - Look all around you
    - People, animals, plants, cars, etc.
  - Attributes
    - Size, shape, color, weight, etc.
  - Behaviors
    - Babies cry, crawl, sleep, etc.

1.16 Software Engineering Case Study (Cont.)

- **Object-oriented design (OOD)**
  - **Models software in terms similar to those used to describe real-world objects**
  - **Class relationships**
  - **Inheritance relationships**
  - **Models communication among objects**
  - **Encapsulates attributes and operations (behaviors)**
    - **Information hiding**
    - **Communication through well-defined interfaces**
- **Object-oriented language**
  - **Programming in object-oriented languages is called *object-oriented programming* (*OOP*)**
  - **Java**

1.16 Software Engineering Case Study (Cont.)

- Classes are to objects as blueprints are to houses
- Associations
  - Relationships between classes
- Packaging software in classes facilitates reuse

1.16 Software Engineering Case Study (Cont.)

- **Object-Oriented Analysis and Design (OOA/D)**
  - **Essential for large programs**
  - **Analyze program requirements, then develop a design**
  - **UML**
    - **Unified Modeling Language**
    - **Standard for designing object-oriented systems**

1.16 Software Engineering Case Study (Cont.)

- **History of the UML**
  - **Need developed for process with which to approach OOA/D**
  - **Brainchild of Booch, Rumbaugh and Jacobson**
  - **Object Management Group (OMG) supervised**
  - **Version 2 is current version**

1.16 Software Engineering Case Study (Cont.)

- **UML**
  - **Graphical representation scheme**
  - **Enables developers to model object-oriented systems**
  - **Flexible and extensible**

1.18 Software Technologies

- **Refactoring**
  - **Reworking code to make cleaner/easier to maintain**
  - **Widely used in agile development methodologies**
  - **Many tools available**
- **Design patterns**
  - **Proven architectures for constructing flexible/maintainable object-oriented software**

1.18 Software Technologies (cont.)

- **Game programming**
  - **Game business is larger than first run movie business**
  - **College courses and majors now devoted to sophisticated game-programming techniques**

1.18 Software Technologies (cont.)

- Open source software
  - Individuals and companies contribute to developing, maintaining and evolving software in exchange for the right to use that software for their own purposes, typically at no charge
  - Code typically scrutinized by much larger audiences, so bugs get removed faster
  - Java now open source
  - Some open source organizations
    - Eclipse Foundation, Mozilla Foundation, Apache Software Foundation, SourceForge

1.18 Software Technologies (cont.)

- Linux
  - Open source operating system
  - One of the greatest successes of the open source movement
- MySQL
  - an open source database management system
- PHP
  - most popular open source server-side Internet “scripting” language for developing Internet-based applications
- LAMP—Linux, Apache, MySQL and PHP (or Perl or Python)
  - An acronym for the set of open source technologies that many developers used to build web applications

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

```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 clazss Welcome1
```

```
Welcome to Java Programming!
```

`Welcome1.java`

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.)

```
9          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.)

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

- Ends method declaration

```
13  } // 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**
  - **Open a 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 2.3

When attempting to compile a program, if you receive a message such as "`bad command or filename,`" "`javac: command not found`" or "`'javac' is not recognized as an internal or external command, operable program or batch file,`" then your Java software installation was not completed properly. If you are using the Java Development Kit, this indicates that the system's *PATH* environment variable was not set properly. Please review the installation instructions in the Before You Begin section of this book carefully. On some systems, after correcting the PATH, you may need to reboot your computer or open a new command window for these settings to take effect.

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.

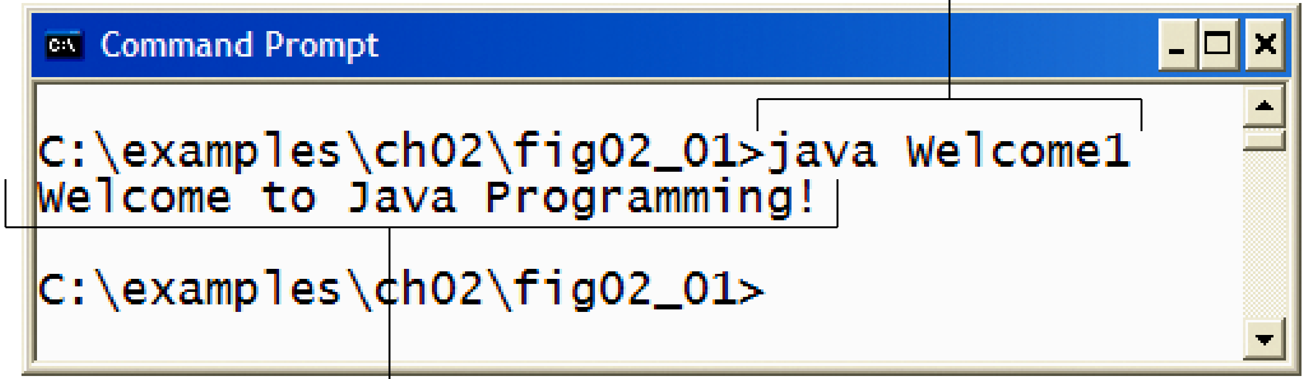
Error-Prevention Tip 2.5

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`

You type this command to execute the application



```
C:\examples\ch02\fig02_01>java Welcome1
Welcome to Java Programming!

C:\examples\ch02\fig02_01>
```

The program outputs

Welcome to Java Programming!

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

Error-Prevention Tip 2.6

When attempting to run a Java program, if you receive a message such as “`Exception in thread "main" java.lang.NoClassDefFoundError: Welcome1`,” your *CLASSPATH* environment variable has not been set properly. Please review the installation instructions in the Before You Begin section of this book carefully. On some systems, you may need to reboot your computer or open a new command window after configuring the *CLASSPATH*.

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

Outline

```
// 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`.

`Addition.java`

(1 of 2)

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

```
      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
  - Used by compiler to identify and locate classes used in Java programs
  - Tells compiler to load class `Scanner` from `java.util` package

```
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**
    - **Declare with name and type before use**
  - **`Input` is of type `Scanner`**
    - **Enables a program to read data for use**
  - **Variable name: any valid identifier**
- **Declarations end with semicolons `;`**
- **Initialize variable in its declaration**
  - **Equal sign**
  - **Standard input object**
    - **`System.in`**

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.)

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

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

```
18          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.)

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

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

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

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

```
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.)

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

- Use `System.out.printf` to display results
- Format specifier `%d`
  - Placeholder for an `int` value

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

- Calculations can also be performed inside `printf`
- 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**

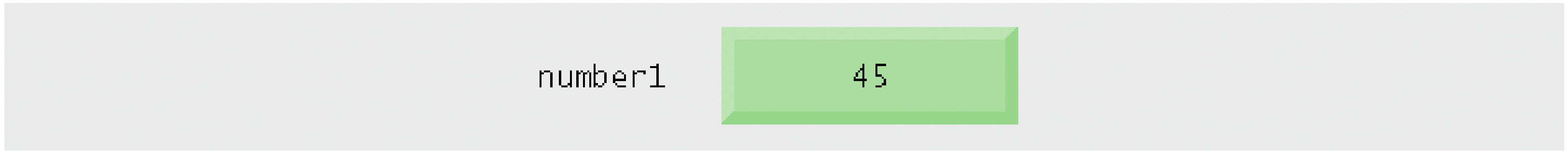


Fig. 2.8 | Memory location showing the name and value of variable `number1`.

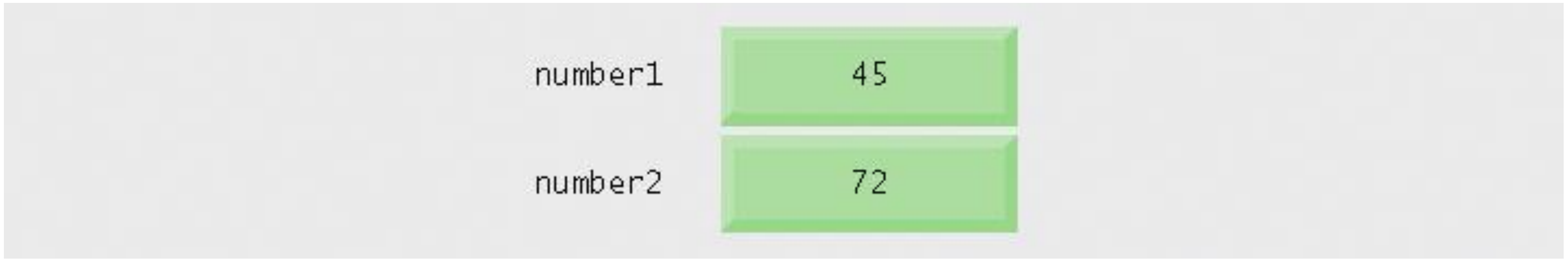


Fig. 2.9 | Memory locations after storing values for `number1` and `number2`.

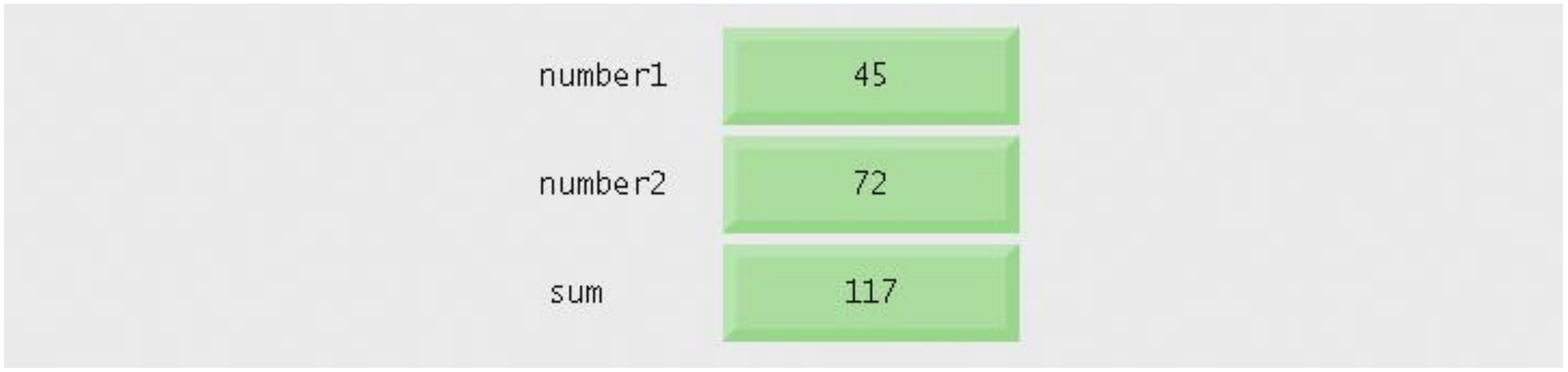


Fig. 2.10 | Memory locations after calculating and storing the sum of `number1` and `number2`.

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`

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.**

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.

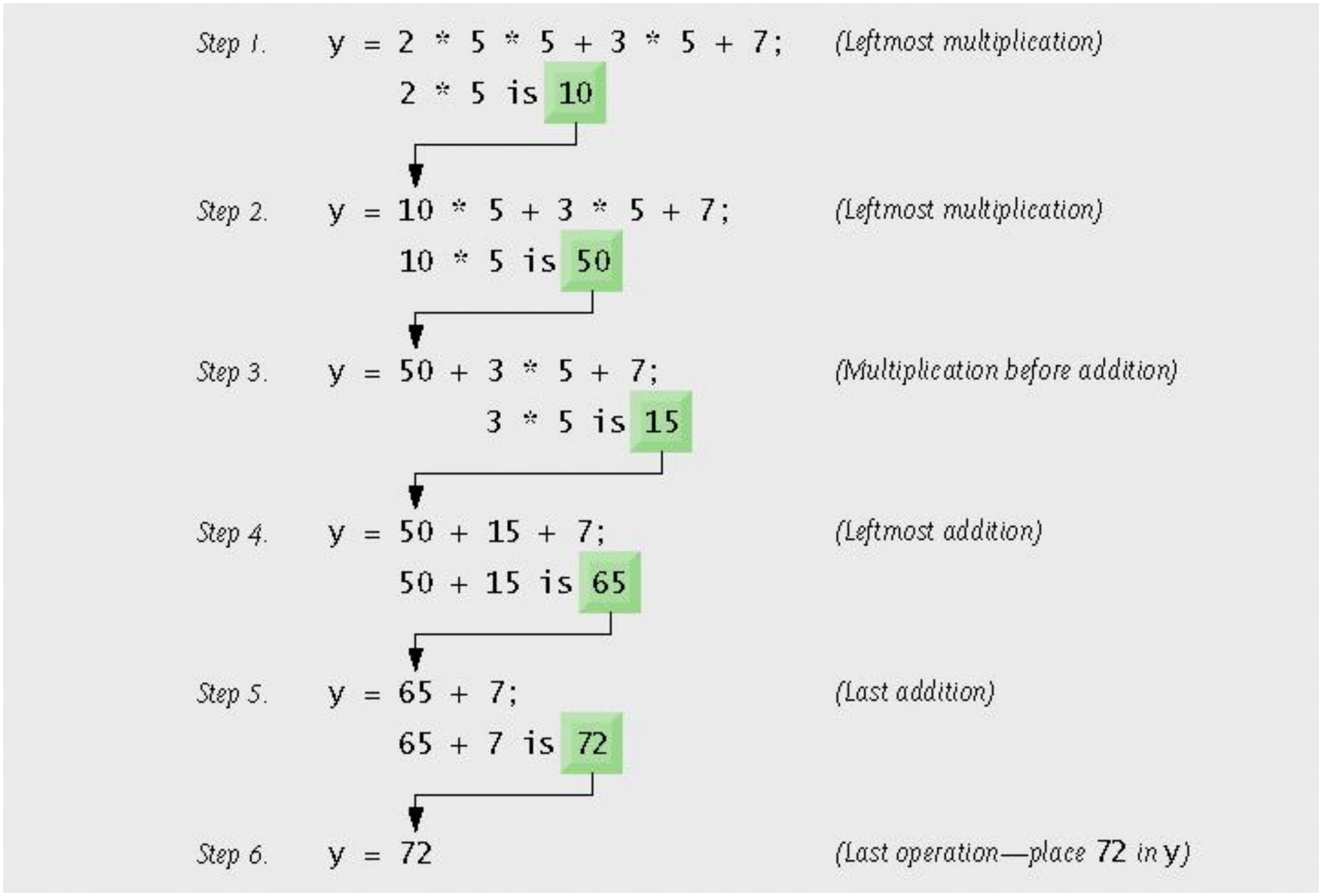


Fig. 2.13 | **Order in which a second-degree polynomial is evaluated.**

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

```java

      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

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 chart (see the complete chart in Appendix A) 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.