Module 1: Introduction to Computers, Programs, and Java

Objectives

- To review Program Design and Problem-Solving Techniques
- To describe the relationship between Java and the World Wide Web (§1.5).
- To understand the meaning of Java language specification, API, JDK, and IDE (§1.6).
- To write a simple Java program (§1.7).
- To display output on the console (§1.7).
- To explain the basic syntax of a Java program (§1.7).
- To create, compile, and run Java programs (§1.8).
- To use sound Java programming style and document programs properly (§1.9).
- To explain the differences between syntax errors, runtime errors, and logic errors (§1.10).
- To develop Java programs using NetBeans (§1.11).

What is Computer Science?

- Computer Science can be summarized with two simple words: **problem solving**.
- Computer Science is the study of problems, problem-solving, and the solutions that come out of this problem-solving process.
- Given a problem, the goal is to develop an ***algorithm*** to solve the problem.
- An algorithm is a step-by-step list of instructions to solve the problem.

What is Programming?

- Once you have developed the algorithm on paper, you must now "prove it" and show that it works.
- Programming is the process of encoding your algorithm into a programming language, so that it can then be executed by a computer.
- But what is the first step?
- You need a solution.
- You need an algorithm!

So who is good at Programming?

- Are you good at problem solving?
- Are you good at strategy?
- These are the core fundamentals of programming.

Program Design & Problem-Solving Techniques

How Do We Write a Program?

- A Computer is not intelligent.
  - It cannot analyze a problem and come up with a solution.
  - A human (the *programmer*) must analyze the problem, develop the instructions for solving the problem, and then have the computer carry out the instructions.
- To write a program for a computer to follow, we must go through a two-phase process: ***problem solving*** and ***implementation.***

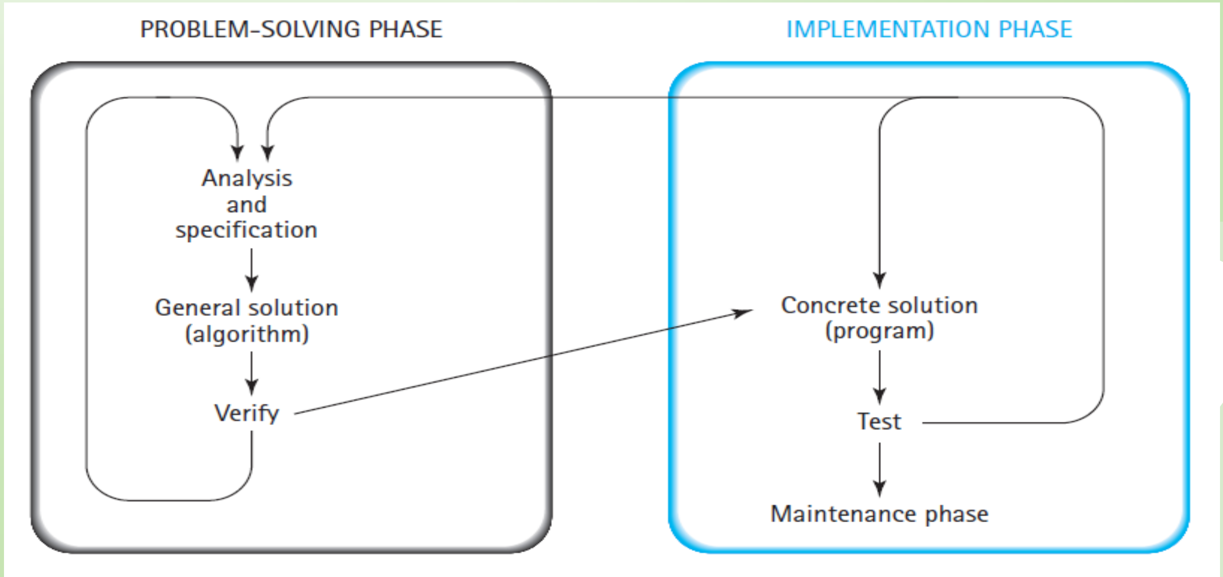

PROBLEM-SOLVING PHASE
IMPLEMENTATION PHASE
Analysis and specification
General solution (algorithm)
Verify
Concrete solution (program)
Test
Maintenance phase

Figure *Programming process*

Problem-Solving Phase

- *Analysis and Specification*- Understand (define) the problem and what the solution must do.
- *General Solution (Algorithm)*- Specify the required data types and the logical sequences of steps that solve the problem.
- *Verify*- Follow the steps exactly to see if the solution really does solve the problem.

Implementation Phase

- *Concrete Solution (Program)*- Translate the algorithm (the general solution) into a programming language.

- *Test*- Have the computer follow the instructions.
  - Then manually check the results.
  - If you find errors, analyze the program and the algorithm to determine the source of the errors, and then make corrections.

- Once a program is tested, it enters into next phase (maintenance).

- Maintenance requires Modification of the program to meet changing requirements or to correct any errors that show up while using it.

Steps in program development

Steps in Program Development

1. Define the problem into three separate components:
   - inputs
   - processing steps to produce required outputs.
   - outputs

Steps in Program Development

2. Outline the solution.
   - Decompose the problem to smaller steps.
   - Establish a solution outline.

3. Develop the outline into an algorithm.
   - The solution outline is now expanded into an algorithm.

Steps in Program Development

4. Test the algorithm for correctness.
   - Very important in the development of a program, but often forgotten
   - Major logic errors can be detected and corrected at an early stage.

5. Code the algorithm into a specific programming language.

Steps in Program Development

6. Run the program on the computer.
   - This step uses a program compiler and programmer-designed test data to machine-test the code for
     - syntax errors
     - logic errors

7. Document and maintain the program.

Algorithms & Flowcharts

- What is pseudocode?
  - Structured English (formalized and abbreviated to look like high-level computer language)
- What is an algorithm?
  - Lists the steps involved in accomplishing a task (like a recipe)
  - An algorithm must:
    - Be lucid (clear), precise and unambiguous
    - Give the correct solution in all cases
    - Eventually end

Pseudocode & Algorithm

Example 1: Write an algorithm to determine a student’s final grade and indicate whether it is passing or failing. The final grade is calculated as the average of four marks.

Pseudocode & Algorithm

Pseudocode:

```
➢ Input a set of 4 marks
➢ Calculate their average by summing and dividing
  by 4
➢ if average is below 50
        Print "FAIL"
  else
        Print "PASS"
```

Pseudocode & Algorithm

Detailed Algorithm

```
Step 1:   Input M1,M2,M3,M4
Step 2:   GRADE ← (M1+M2+M3+M4)/4
Step 3:   if (GRADE < 50) then
              Print "FAIL"
          else
              Print "PASS"
          endif
```

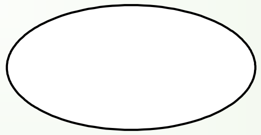
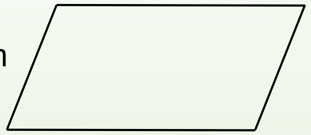
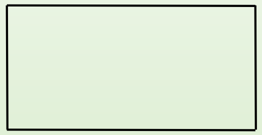
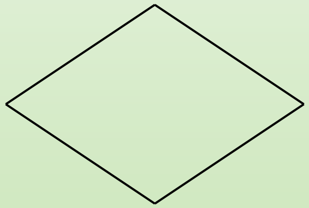
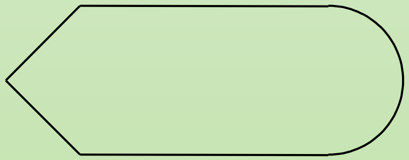
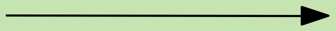
Flowchart

- A graphical representation of the sequence of operations in an information system or program.

- Program flowcharts show the sequence of instructions in a single program or subroutine.
  - shows logic of an algorithm
  - emphasizes individual steps and their interconnections
  - e.g. control flow from one action to the next

Note: Different symbols are used to draw each type of flowchart.

Flowchart Symbols

Name	**Symbol**	**Use in Flowchart**
Oval		Denotes the beginning or end of the program
Parallelogram		Denotes an input operation
Rectangle		Denotes a process to be carried out e.g. addition, subtraction, division etc.
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes. (e.g. IF/THEN/ELSE)
Hybrid		Denotes an output operation
Flow line		Denotes the direction of logic flow in the program

Example 1

- Write a Pseudocode, an algorithm and draw a flowchart to convert the length in feet to centimeter.

Pseudocode:

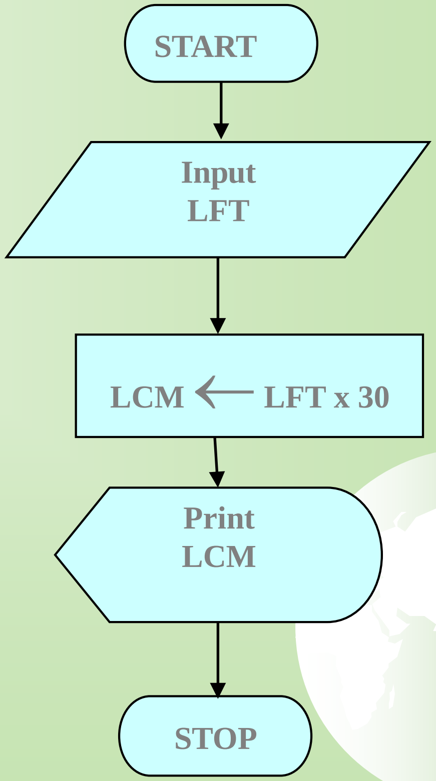
- *Input the length in feet (LFT)*
- *Calculate the length in cm (LCM) by multiplying LFT with 30*
- *Print length in cm (LCM)*

Example 1

Algorithm

- Step 1: Input LFT
- Step 2: LCM ← LFT x 30
- Step 3: Print LCM

Flowchart



Example 2

Write a **Pseudocode, an algorithm and draw a flowchart that will read the two sides of a rectangle and calculate its area.**

Pseudocode

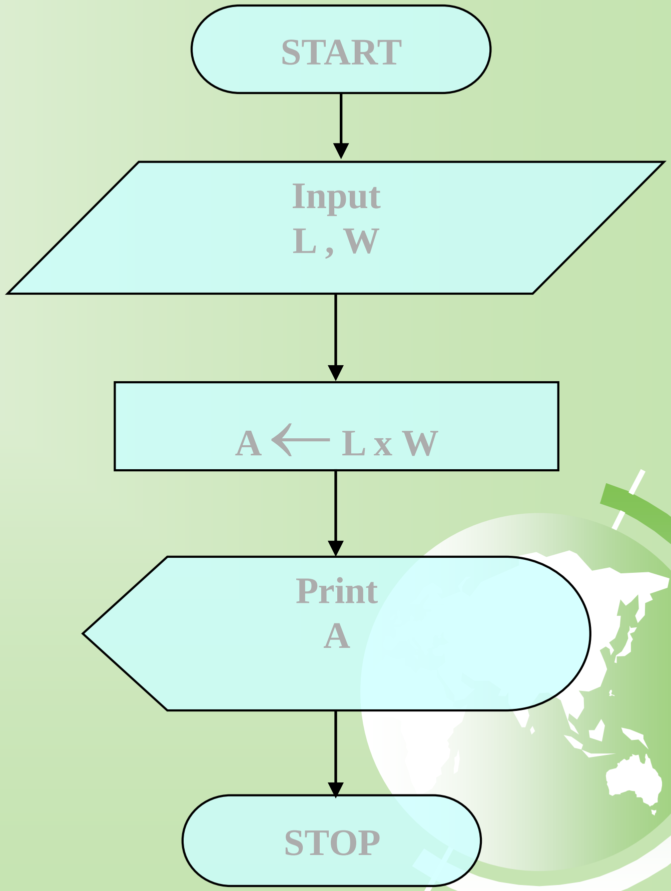
- *Input the Length (L) and width (W) of a rectangle*
- *Calculate the area (A) by multiplying L with W*
- *Print A*

Example 2

Algorithm

- Step 1: Input L, W
- Step 2: A ← L x W
- Step 3: Print A

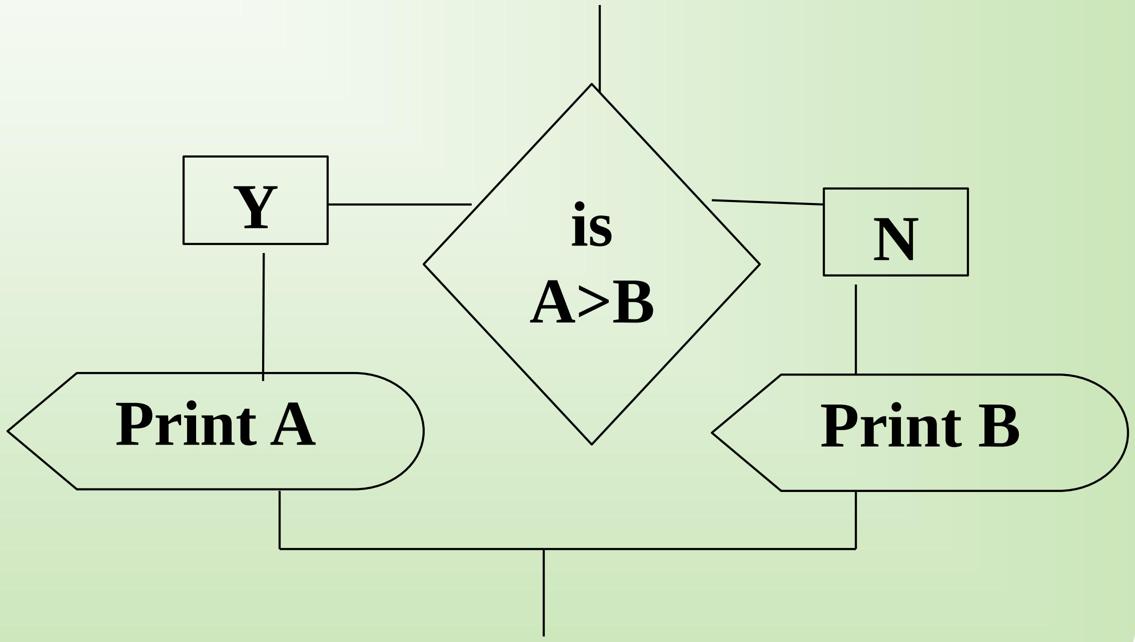
Flowchart



Decision Structures

- The expression A>B is a logical expression
- *it describes a **condition** we want to test*
- ***if A>B is true (if A is greater than B)** we take the action on left*
- print the value of A
- ***if A>B is false (if A is not greater than B)** we take the action on right*
- print the value of B

Decision Structures



IF–THEN–ELSE STRUCTURE

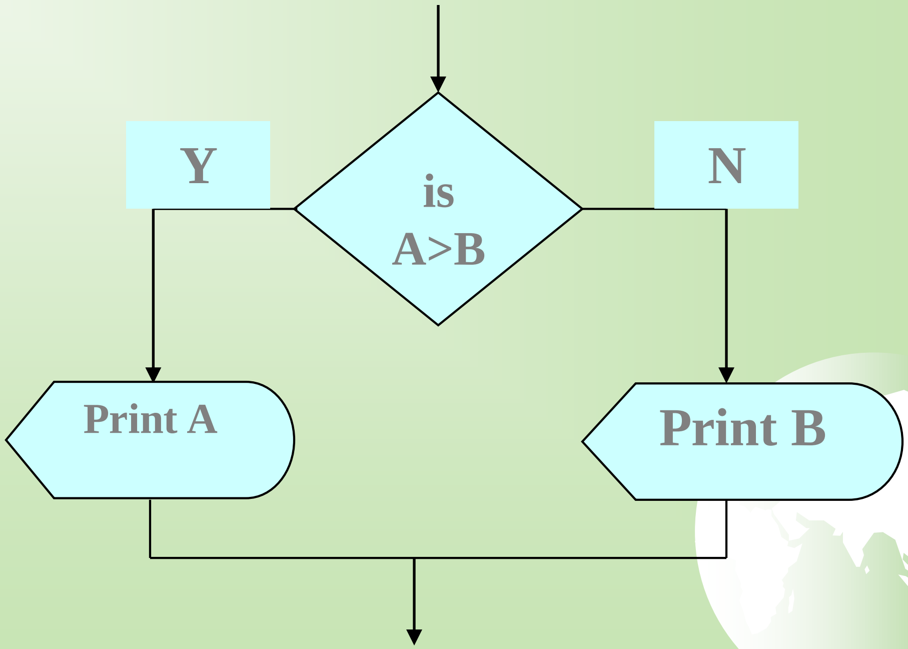
- The structure is as follows

```
If condition  then
        true alternative
else
        false alternative
End if
```

IF–THEN–ELSE STRUCTURE

- The algorithm for the flowchart is as follows:

```
If A>B then
    print A
else
    print B
endif
```



Relational Operators

Relational Operators	
Operator	**Description**
>	Greater than
<	Less than
=	Equal to
≥ Or >=	Greater than or equal to
≤ Or <=	Less than or equal to
≠ Or !=	Not equal to

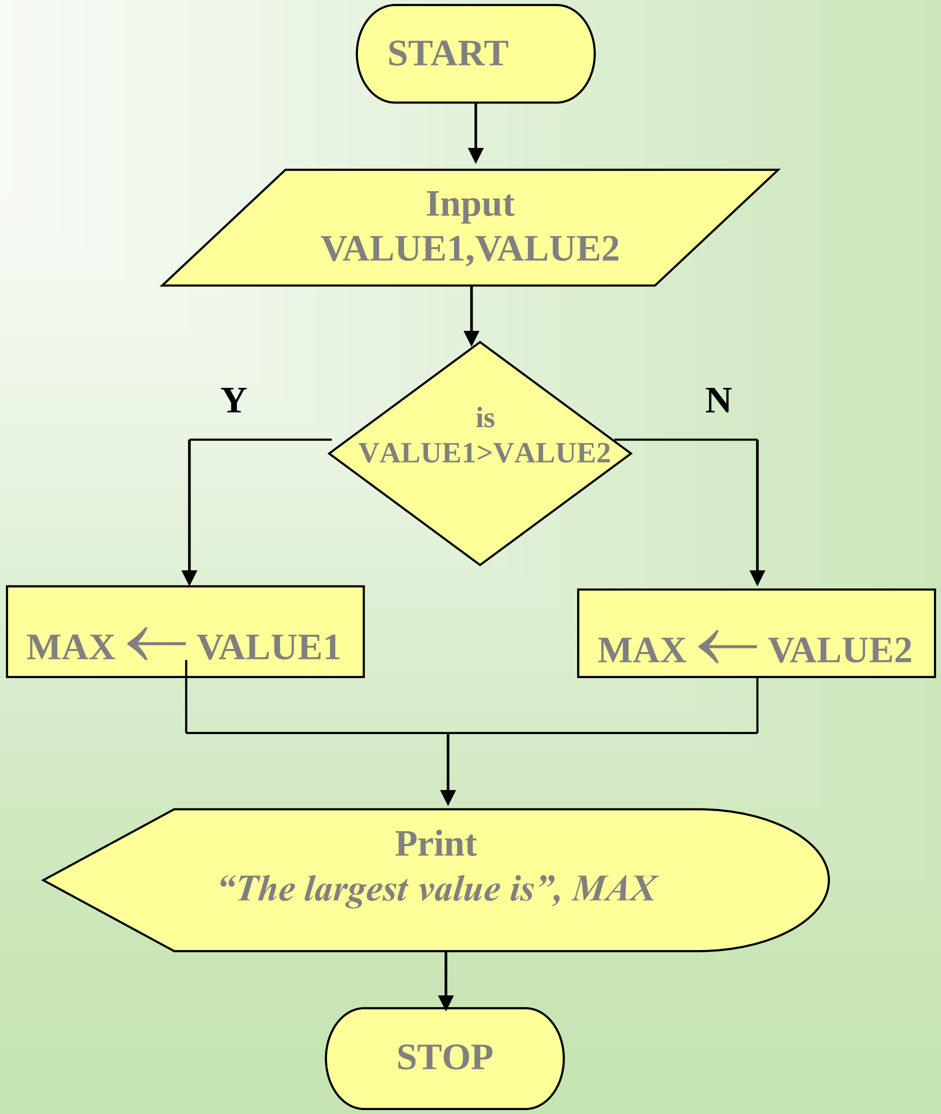
Example 4

- Write an algorithm that reads two values, determines the largest value and prints the largest value with an identifying message.

ALGORITHM

```
Step 1:     Input VALUE1, VALUE2
Step 2:     if (VALUE1 > VALUE2) then
                MAX ← VALUE1
            else
                MAX ← VALUE2
            endif
Step 3:     Print "The largest value is", MAX
```

Example 4


START
Input
VALUE1,VALUE2
is
VALUE1>VALUE2
Y
N
MAX ← VALUE1
MAX ← VALUE2
Print
"The largest value is", MAX
STOP

Programs

Computer *programs*, known as *software*, are instructions to the computer.

You tell a computer what to do through programs. Without programs, a computer is an empty machine. Computers do not understand human languages, so you need to use computer languages to communicate with them.

Programs are written using programming languages.

Programming Languages

Machine Language   Assembly Language   High-Level Language

Machine language is a set of primitive instructions built into every computer. The instructions are in the form of binary code, so you have to enter binary codes for various instructions. Program with native machine language is a tedious process. Moreover the programs are highly difficult to read and modify. For example, to add two numbers, you might write an instruction in binary like this:

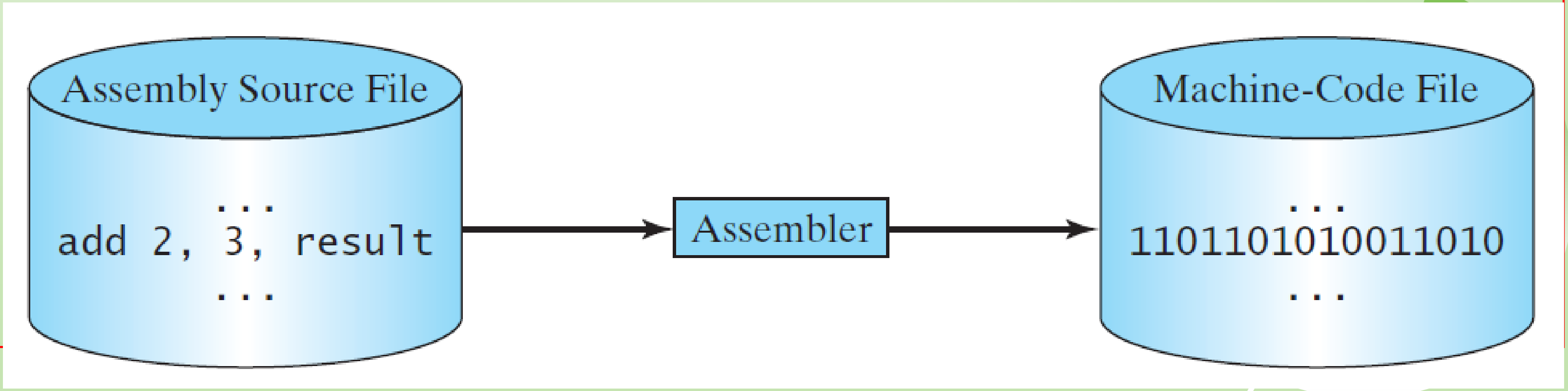
```
1101101010011010
```

Programming Languages

Machine Language   Assembly Language   High-Level Language

Assembly languages were developed to make programming easy. Since the computer cannot understand assembly language, however, a program called assembler is used to convert assembly language programs into machine code. For example, to add two numbers, you might write an instruction in assembly code like this:

ADDF3 R1, R2, R3



Programming Languages

Machine Language   Assembly Language   High-Level Language

The high-level languages are English-like and easy to learn and program. For example, the following is a high-level language statement that computes the area of a circle with radius 5:

```
area = 5 * 5 * 3.1415;
```

Popular High-Level Languages

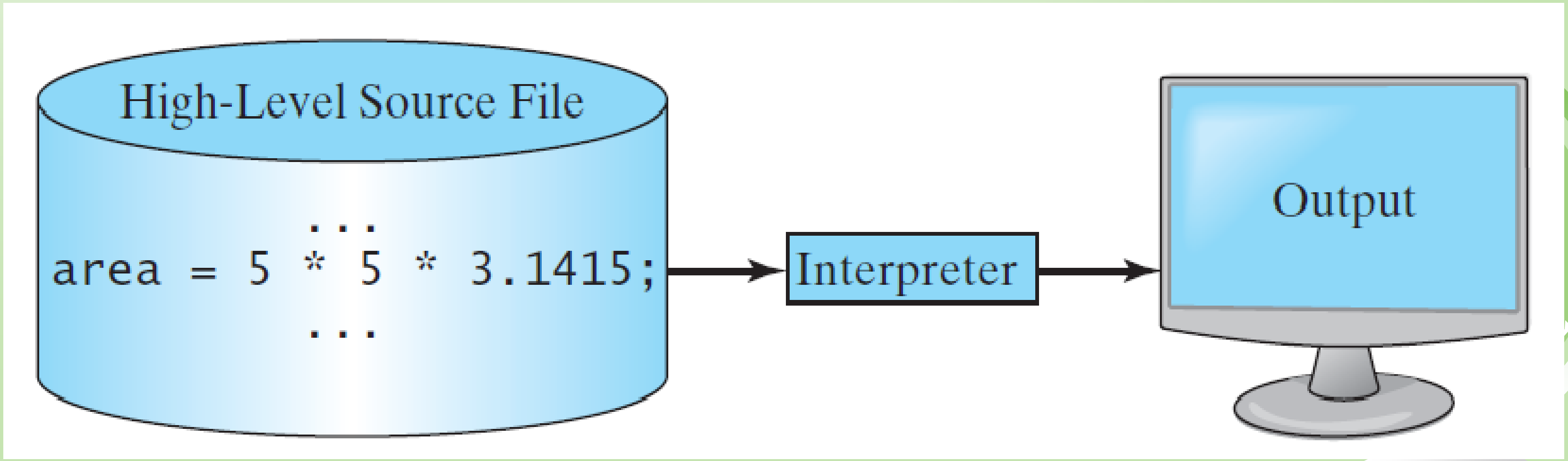
Language	Description
Ada	Named for Ada Lovelace, who worked on mechanical general-purpose computers. The Ada language was developed for the Department of Defense and is used mainly in defense projects.
BASIC	Beginner's All-purpose Symbolic Instruction Code. It was designed to be learned and used easily by beginners.
C	Developed at Bell Laboratories. C combines the power of an assembly language with the ease of use and portability of a high-level language.
C++	C++ is an object-oriented language, based on C.
C#	Pronounced "C Sharp." It is a hybrid of Java and C++ and was developed by Microsoft.
COBOL	COmmon Business Oriented Language. Used for business applications.
FORTRAN	FORmula TRANslation. Popular for scientific and mathematical applications.
Java	Developed by Sun Microsystems, now part of Oracle. It is widely used for developing platform-independent Internet applications.
Pascal	Named for Blaise Pascal, who pioneered calculating machines in the seventeenth century. It is a simple, structured, general-purpose language primarily for teaching programming.
Python	A simple general-purpose scripting language good for writing short programs.
Visual Basic	Visual Basic was developed by Microsoft and it enables the programmers to rapidly develop graphical user interfaces.

Interpreting/Compiling Source Code

- A program written in a high-level language is called a *source program* or *source code*.
- Because a computer cannot understand a source program, a source program must be translated into machine code for execution.
- The translation can be done using another programming tool called an *interpreter* or a *compiler*.

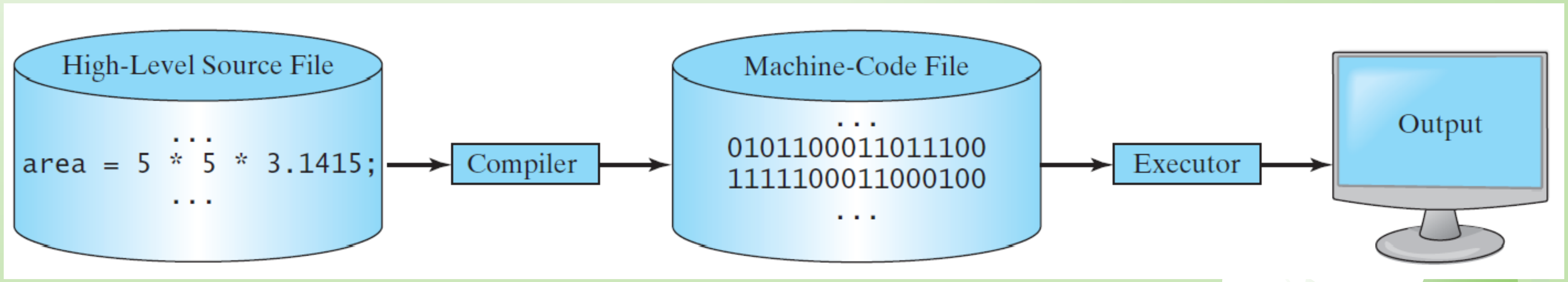
Interpreting Source Code

An interpreter reads one statement from the source code, translates it to the machine code or virtual machine code, and then executes it right away, as shown in the following figure. Note that a statement from the source code may be translated into several machine instructions.



Compiling Source Code

A compiler translates the entire source code into a machine-code file, and the machine-code file is then executed, as shown in the following figure.



Why Java?

The answer is that Java enables users to develop and deploy applications on the Internet for servers, desktop computers, and small hand-held devices. The future of computing is being profoundly influenced by the Internet, and Java promises to remain a big part of that future. Java is the Internet programming language.

- ☞Java is a general purpose programming language.
- ☞Java is the Internet programming language.

Java, Web, and Beyond

- Java can be used to develop standalone applications.
- Java can be used to develop applications running from a browser.
- Java can also be used to develop applications for hand-held devices.
- Java can be used to develop applications for Web servers.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

www.cs.armstrong.edu/liang/JavaCharacteristics.pdf

Companion Website

Characteristics of Java

- ☞ Java Is Simple
- ☞ Java Is Object-Oriented
- ☞ Java Is Distributed
- ☞ Java Is Interpreted
- ☞ Java Is Robust
- ☞ Java Is Secure
- ☞ Java Is Architecture-Neutral
- ☞ Java Is Portable
- ☞ Java's Performance
- ☞ Java Is Multithreaded
- ☞ Java Is Dynamic

Java is partially modeled on C++, but greatly simplified and improved. Some people refer to Java as "C++--" because it is like C++ but with more functionality and fewer negative aspects.

Companion Website

Characteristics of Java

- ☞ Java Is Simple
- ☞ Java Is Object-Oriented
- ☞ Java Is Distributed
- ☞ Java Is Interpreted
- ☞ Java Is Robust
- ☞ Java Is Secure
- ☞ Java Is Architecture-Neutral
- ☞ Java Is Portable
- ☞ Java's Performance
- ☞ Java Is Multithreaded
- ☞ Java Is Dynamic

Java is inherently object-oriented. Although many object-oriented languages began strictly as procedural languages, Java was designed from the start to be object-oriented. Object-oriented programming (OOP) is a popular programming approach that is replacing traditional procedural programming techniques.

One of the central issues in software development is how to reuse code. Object-oriented programming provides great flexibility, modularity, clarity, and reusability through encapsulation, inheritance, and polymorphism.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Distributed computing involves several computers working together on a network. Java is designed to make distributed computing easy. Since networking capability is inherently integrated into Java, writing network programs is like sending and receiving data to and from a file.

Companion Website

Characteristics of Java

- ☞ Java Is Simple
- ☞ Java Is Object-Oriented
- ☞ Java Is Distributed
- ☞ Java Is Interpreted
- ☞ Java Is Robust
- ☞ Java Is Secure
- ☞ Java Is Architecture-Neutral
- ☞ Java Is Portable
- ☞ Java's Performance
- ☞ Java Is Multithreaded
- ☞ Java Is Dynamic

You need an interpreter to run Java programs. The programs are compiled into the Java Virtual Machine code called bytecode. The bytecode is machine-independent and can run on any machine that has a Java interpreter, which is part of the Java Virtual Machine (JVM).

Companion Website

Characteristics of Java

- ☞ Java Is Simple
- ☞ Java Is Object-Oriented
- ☞ Java Is Distributed
- ☞ Java Is Interpreted
- ☞ Java Is Robust
- ☞ Java Is Secure
- ☞ Java Is Architecture-Neutral
- ☞ Java Is Portable
- ☞ Java's Performance
- ☞ Java Is Multithreaded
- ☞ Java Is Dynamic

Java compilers can detect many problems that would first show up at execution time in other languages.

Java has eliminated certain types of error-prone programming constructs found in other languages.

Java has a runtime exception-handling feature to provide programming support for robustness.

Companion Website

Characteristics of Java

- ☞ Java Is Simple
- ☞ Java Is Object-Oriented
- ☞ Java Is Distributed
- ☞ Java Is Interpreted
- ☞ Java Is Robust
- ☞ Java Is Secure
- ☞ Java Is Architecture-Neutral
- ☞ Java Is Portable
- ☞ Java's Performance
- ☞ Java Is Multithreaded
- ☞ Java Is Dynamic

Java implements several security mechanisms to protect your system against harm caused by stray programs.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Write once, run anywhere

With a Java Virtual Machine (JVM), you can write one program that will run on any platform.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Because Java is architecture neutral, Java programs are portable. They can be run on any platform without being recompiled.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java’s performance Because Java is architecture neutral, Java programs are portable. They can be run on any platform without being recompiled.

Companion Website

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Distributed
- Java Is Interpreted
- Java Is Robust
- Java Is Secure
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Multithread programming is smoothly integrated in Java, whereas in other languages you have to call procedures specific to the operating system to enable multithreading.

Companion Website

Characteristics of Java

- ☞ Java Is Simple
- ☞ Java Is Object-Oriented
- ☞ Java Is Distributed
- ☞ Java Is Interpreted
- ☞ Java Is Robust
- ☞ Java Is Secure
- ☞ Java Is Architecture-Neutral
- ☞ Java Is Portable
- ☞ Java's Performance
- ☞ Java Is Multithreaded
- ☞ Java Is Dynamic

Java was designed to adapt to an evolving environment. New code can be loaded on the fly without recompilation. There is no need for developers to create, and for users to install, major new software versions. New features can be incorporated transparently as needed.

JDK Versions

- JDK 1.02 (1995)
- JDK 1.1 (1996)
- JDK 1.2 (1998)
- JDK 1.3 (2000)
- JDK 1.4 (2002)
- JDK 1.5 (2004) a. k. a. JDK 5 or Java 5
- JDK 1.6 (2006) a. k. a. JDK 6 or Java 6
- JDK 1.7 (2011) a. k. a. JDK 7 or Java 7
- JDK 1.8 (2014) a. k. a. JDK 8 or Java 8

JDK Editions

- ☞ Java Standard Edition (J2SE)
  - – J2SE can be used to develop client-side standalone applications or applets.
- ☞ Java Enterprise Edition (J2EE)
  - – J2EE can be used to develop server-side applications such as Java servlets, Java ServerPages, and Java ServerFaces.
- ☞ Java Micro Edition (J2ME).
  - – J2ME can be used to develop applications for mobile devices such as cell phones.

This book uses J2SE to introduce Java programming.

Popular Java IDEs

- ☞ NetBeans
- ☞ Eclipse

Developing Java Programs Using NetBeans

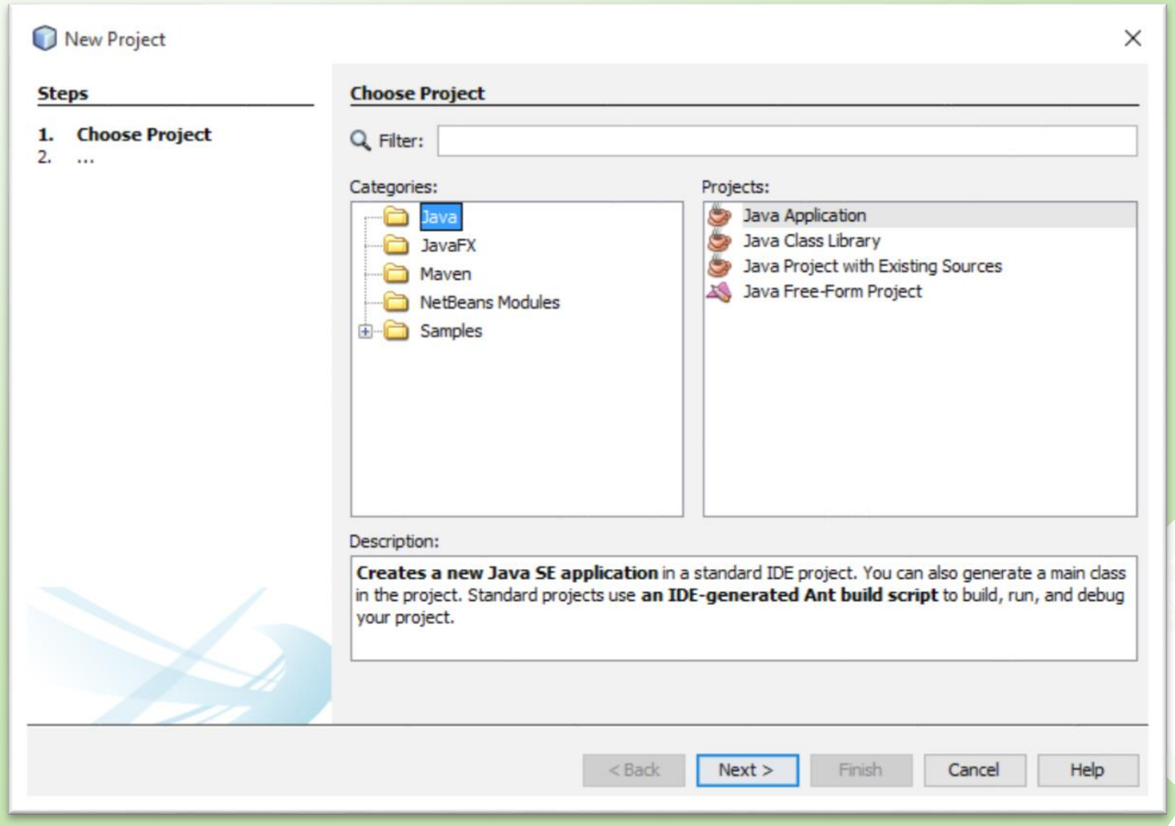
- NetBeans is a free IDE for developing Java Programs
- What is an IDE?
  - Stands for “Integrated Development Environment”
  - An IDE is a software application that provides computer programmers a suitable environment for software development

Developing Java Programs Using NetBeans

- Creating a Java Project:
  - Before you can create Java programs, you must create a Java Project in NetBeans.
  - Think of a project as a folder to your Java program and all supporting files.
  - You should create a new project for each Java program you write.

Creating a Java Project in NetBeans

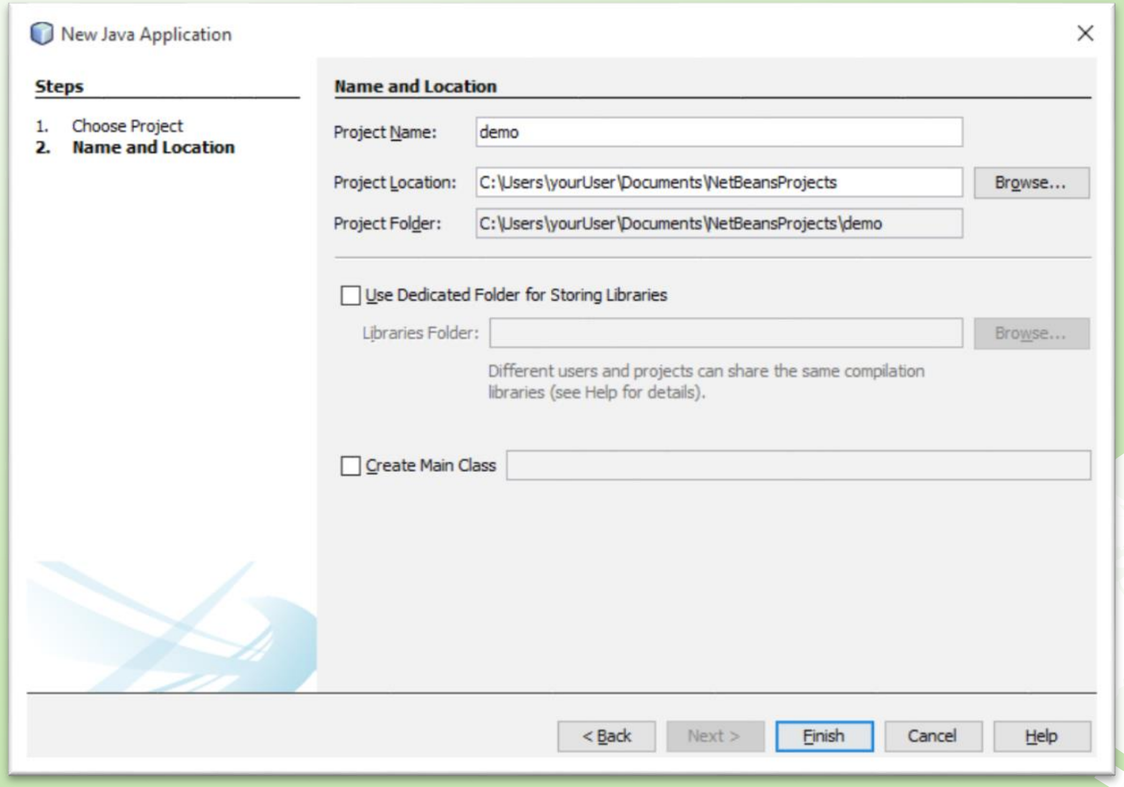
1) Choose *File*, *New Project* to display the New Project dialog box, as shown below:



Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

Creating a Java Project in NetBeans

2) Select Java in the Categories section and Java Application in the Projects section and then click *Next*, which will display the New Java Application dialog box, as shown below:



New Java Application

Steps
1. Choose Project
2. **Name and Location**

Name and Location

Project Name: demo
Project Location: C:\Users\yourUser\Documents\NetBeansProjects
Project Folder: C:\Users\yourUser\Documents\NetBeansProjects\demo

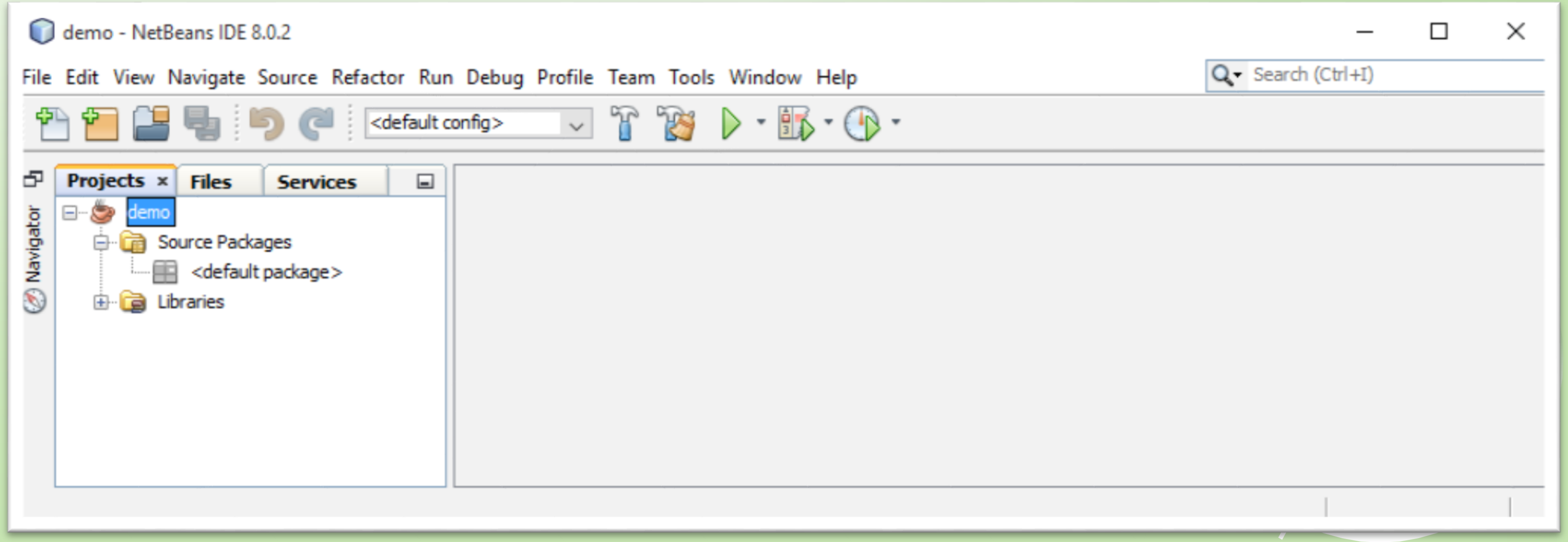
[ ] Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).

[ ] Create Main Class

Creating a Java Project in NetBeans

3) We must now name the project. Type **demo** in the Project Name field. Keep the default settings for Project Location and Folder. And uncheck the two check-boxes.

4) Click *Finish* to create the Project, as shown below:

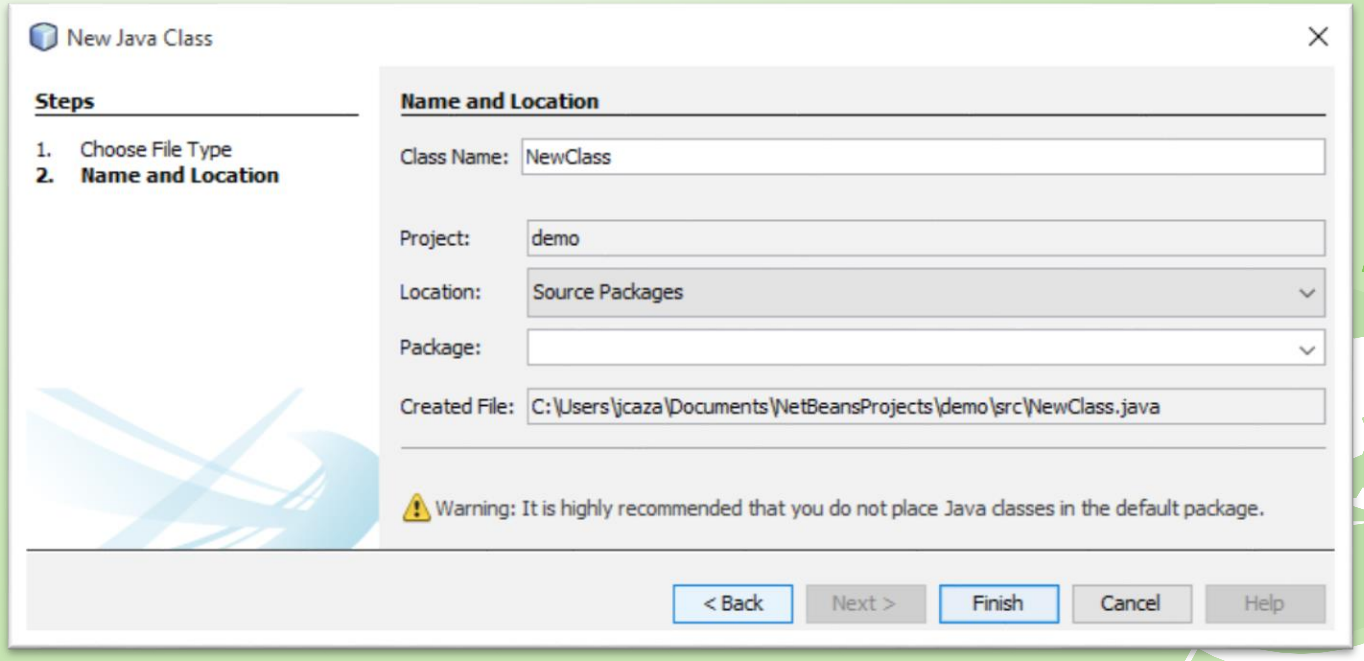


Creating a Java Project in NetBeans

- We just created a project.
- Now that the project is created, we can create Java programs inside the project.
- This process is called “Creating a Java Class”.

Creating a Java Class in NetBeans

1) From the <u>left side</u> of NetBeans, you can see your new project, **demo**, under the Projects tab. Right-click on the actual project name, "demo". A menu appears. Choose *New*, *Java Class*, which displays the New Java Class dialog box, as shown below:



Class Name: NewClass

Project: demo

Location: Source Packages

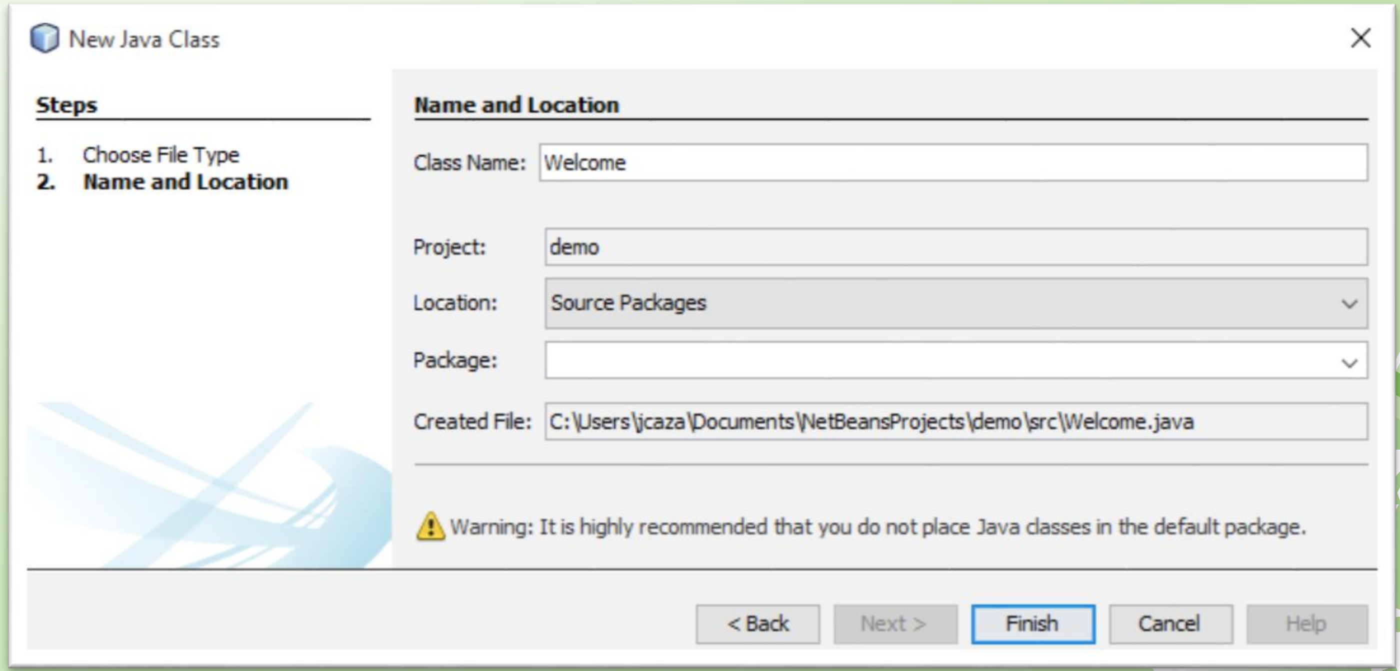
Package:

Created File: C:\Users\jcaza\Documents\NetBeansProjects\demo\src\NewClass.java

Warning: It is highly recommended that you do not place Java classes in the default package.

Creating a Java Class in NetBeans

2) You must now give your new Java program a name. Type **Welcome** in the Class Name field. You can leave all other fields as their default values.



Class Name: Welcome

Project: demo

Location: Source Packages

Package:

Created File: C:\Users\jcaza\Documents\NetBeansProjects\demo\src\Welcome.java

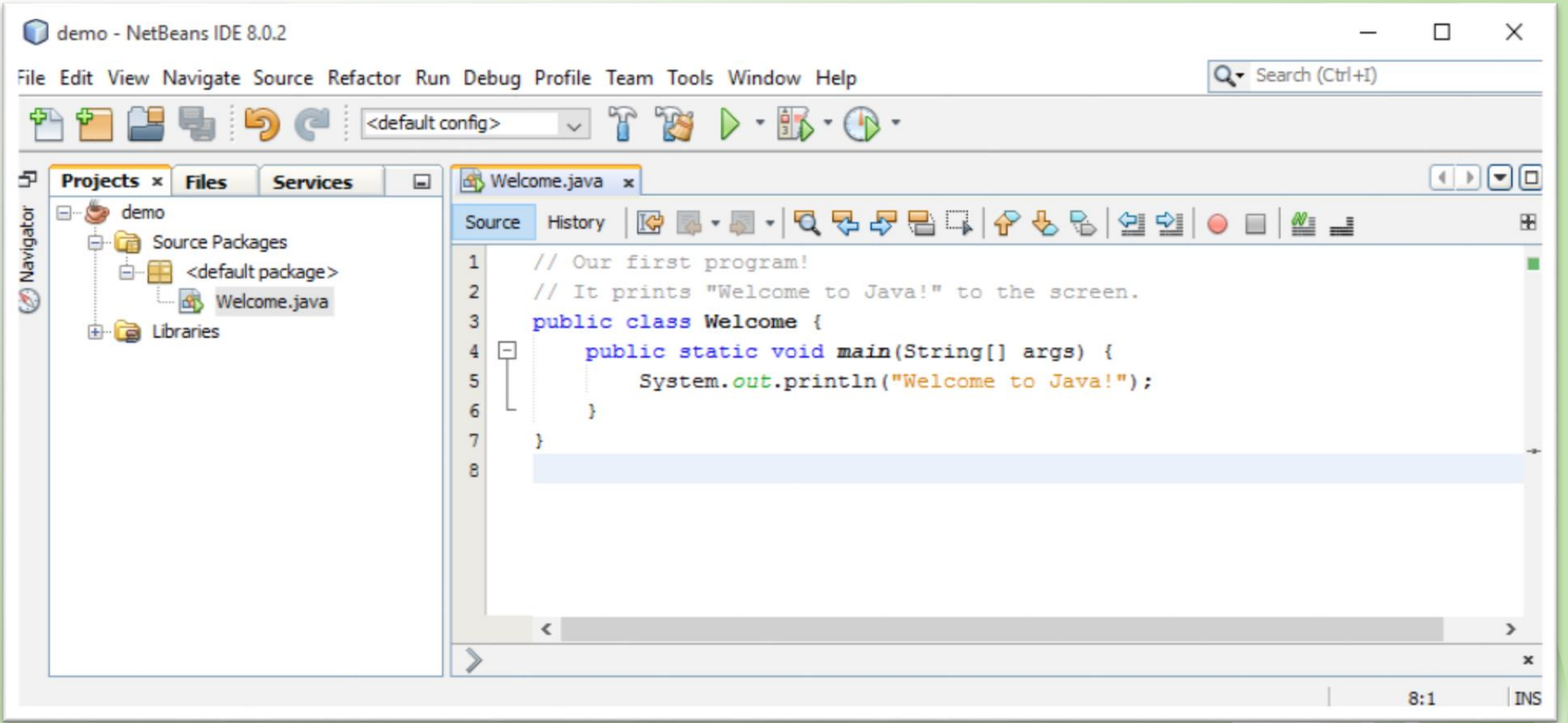
Warning: It is highly recommended that you do not place Java classes in the default package.

Creating a Java Class in NetBeans

3) Click *Finish* to create the Welcome class. Because the package field was left empty, the source code for your new program, `Welcome.java`, will be placed under the <default package> node.

4) Now that we have our first Java program, we can modify the code inside the Welcome class to make it match the figure on the next page.

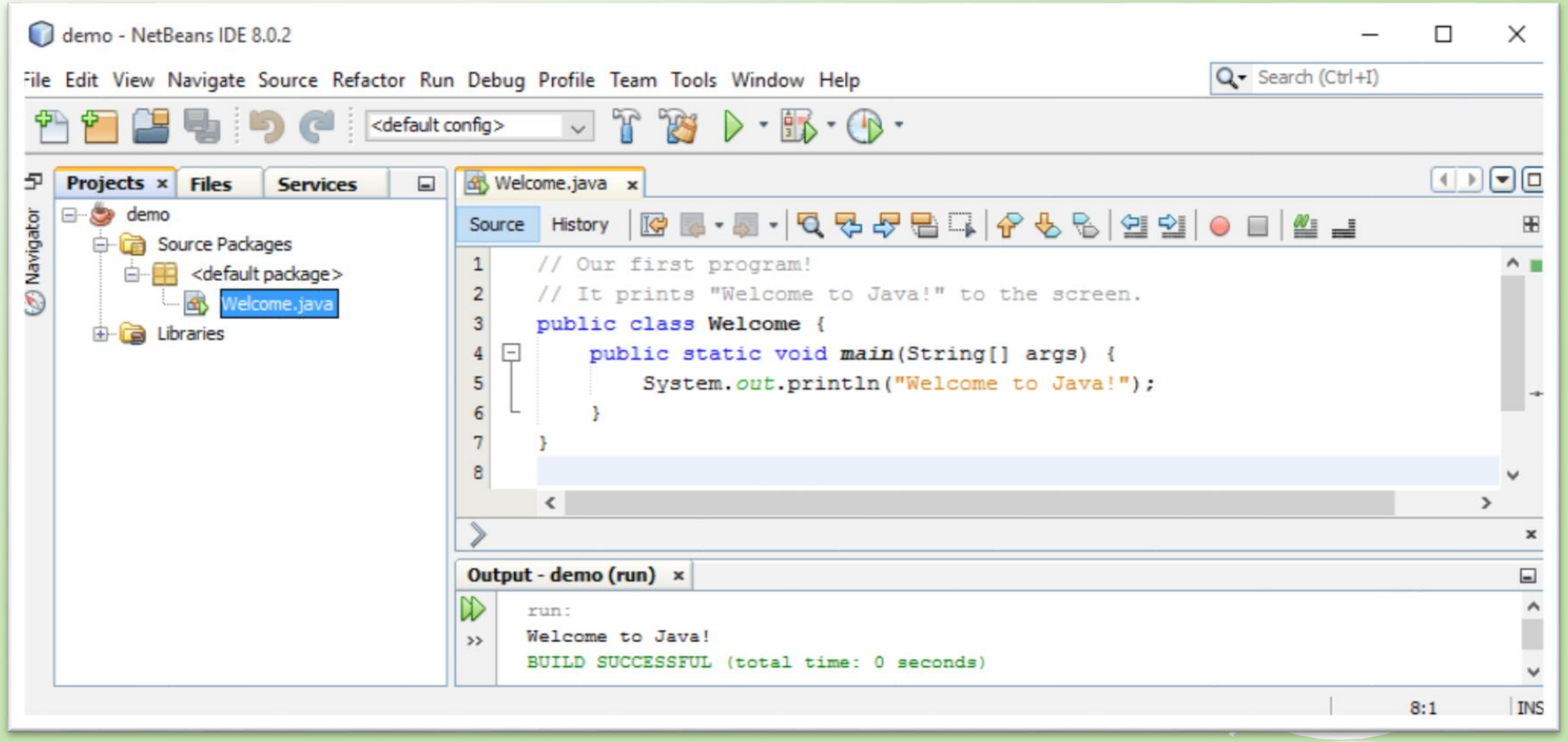
Creating a Java Class in NetBeans



```
// Our first program!
// It prints "Welcome to Java!" to the screen.
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

Run your Java Program in NetBeans

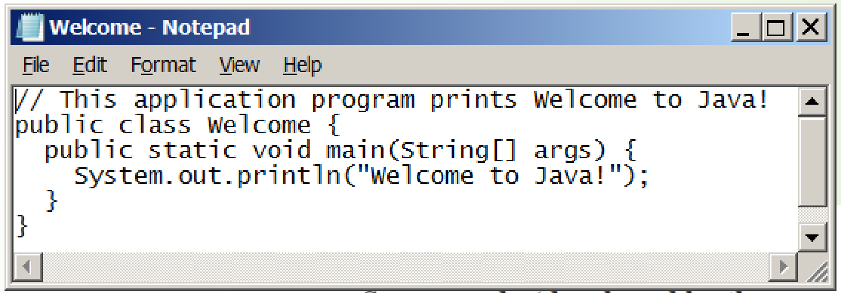
- The final step is easy: just run the program and see the result.
- To run `Welcome.java`, right-click Welcome.java to display a menu, and then choose **Run File**, or simply press **Shift + F6**.



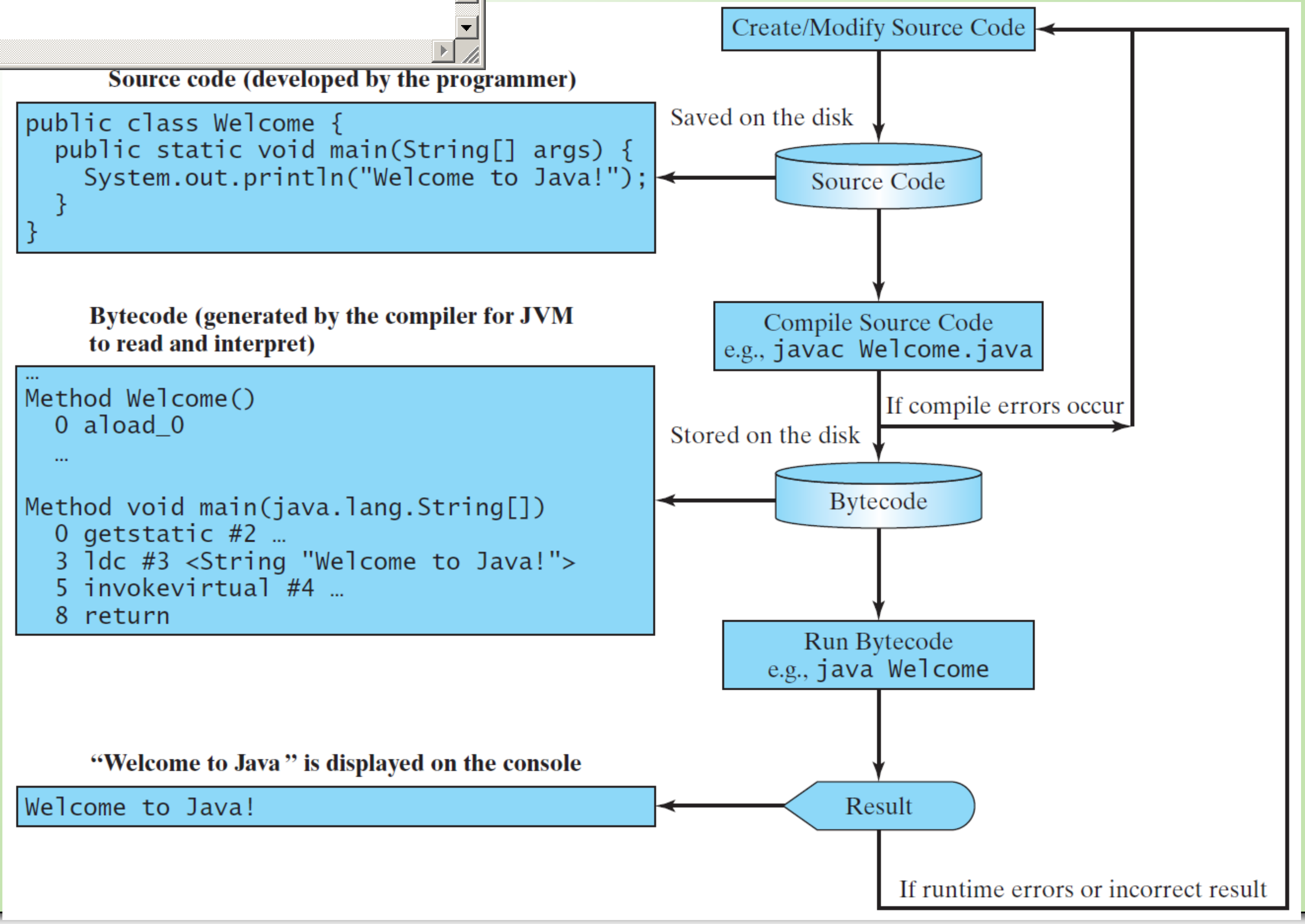
```
// Our first program!
// It prints "Welcome to Java!" to the screen.
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

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

Creating, Compiling, and Running Programs



```
// This application program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```


Source code (developed by the programmer)
Create/Modify Source Code
Saved on the disk
Source Code
Bytecode (generated by the compiler for JVM to read and interpret)
Compile Source Code e.g., javac Welcome.java
If compile errors occur
Stored on the disk
Bytecode
Run Bytecode e.g., java Welcome
"Welcome to Java" is displayed on the console
Result
If runtime errors or incorrect result

```
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

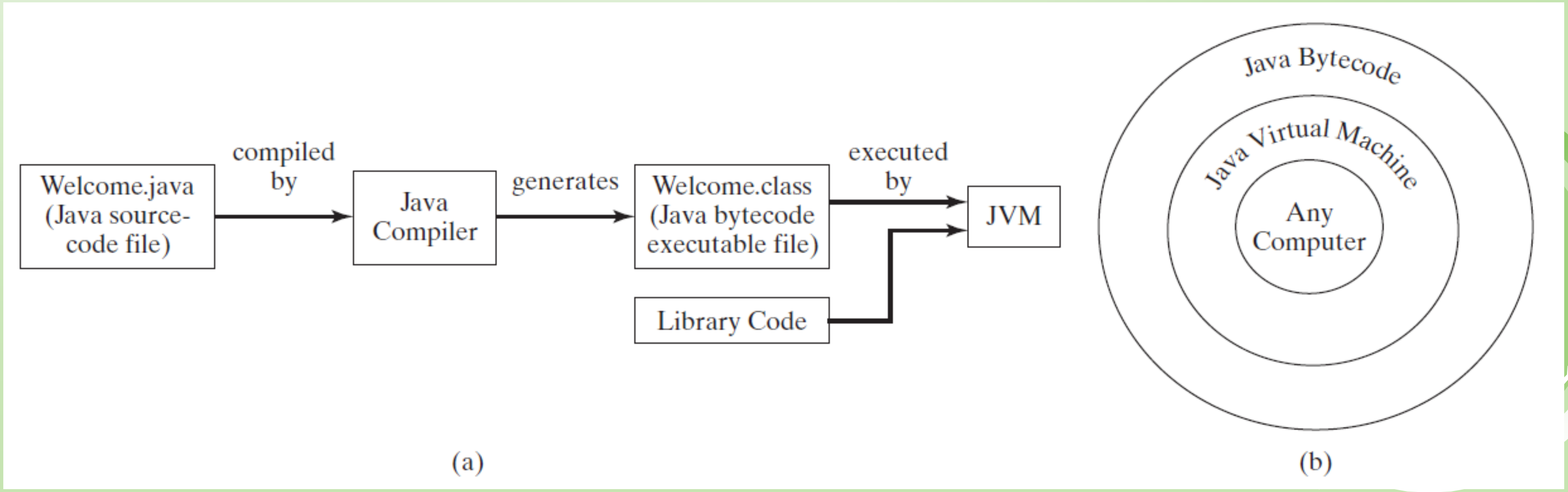
```
…
Method Welcome()
  0 aload_0
  …

Method void main(java.lang.String[])
  0 getstatic #2 …
  3 ldc #3 <String "Welcome to Java!">
  5 invokevirtual #4 …
  8 return
```

```
Welcome to Java!
```

Compiling Java Source Code

You can port a source program to any machine with appropriate compilers. The source program must be recompiled, however, because the object program can only run on a specific machine. Nowadays computers are networked to work together. Java was designed to run object programs on any platform. With Java, you write the program once, and compile the source program into a special type of object code, known as *bytecode*. The bytecode can then run on any computer with a Java Virtual Machine, as shown below. Java Virtual Machine is a software that interprets Java bytecode.



animation

Trace a Program Execution

Enter main method

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

animation

Trace a Program Execution

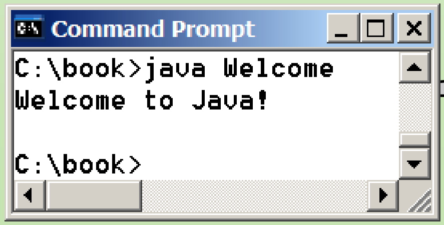
Execute statement

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

animation

Trace a Program Execution

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

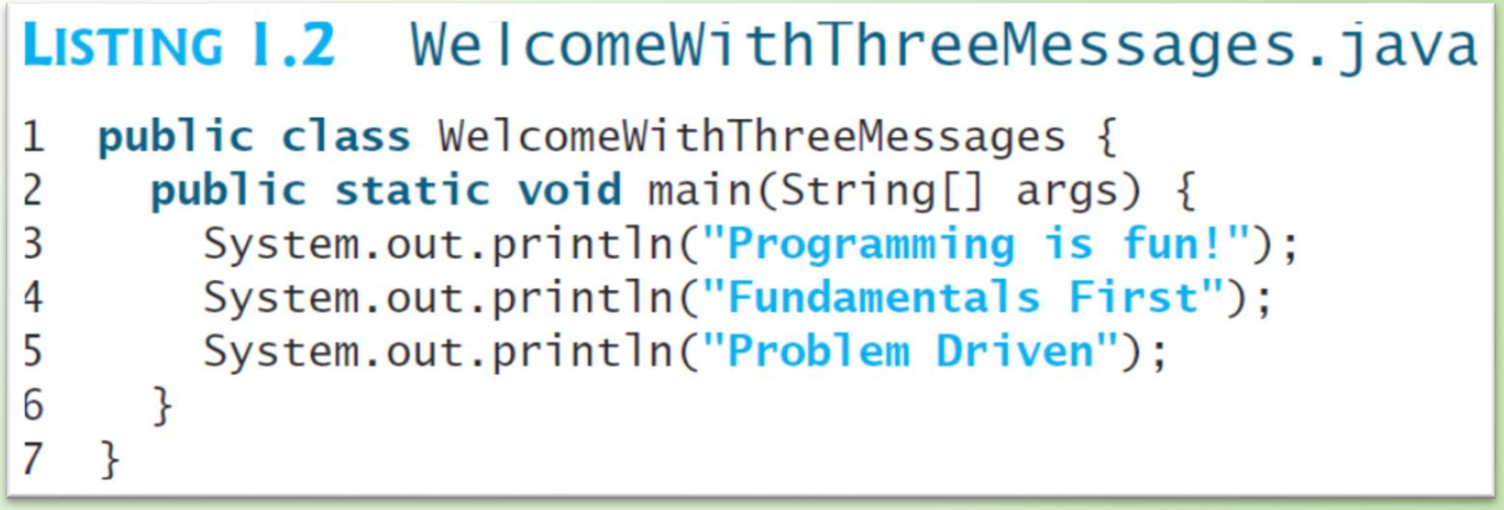


```
C:\book>java Welcome
Welcome to Java!

C:\book>
```

print a message to the console

Two Small Programs


LISTING 1.2 WelcomeWithThreeMessages.java

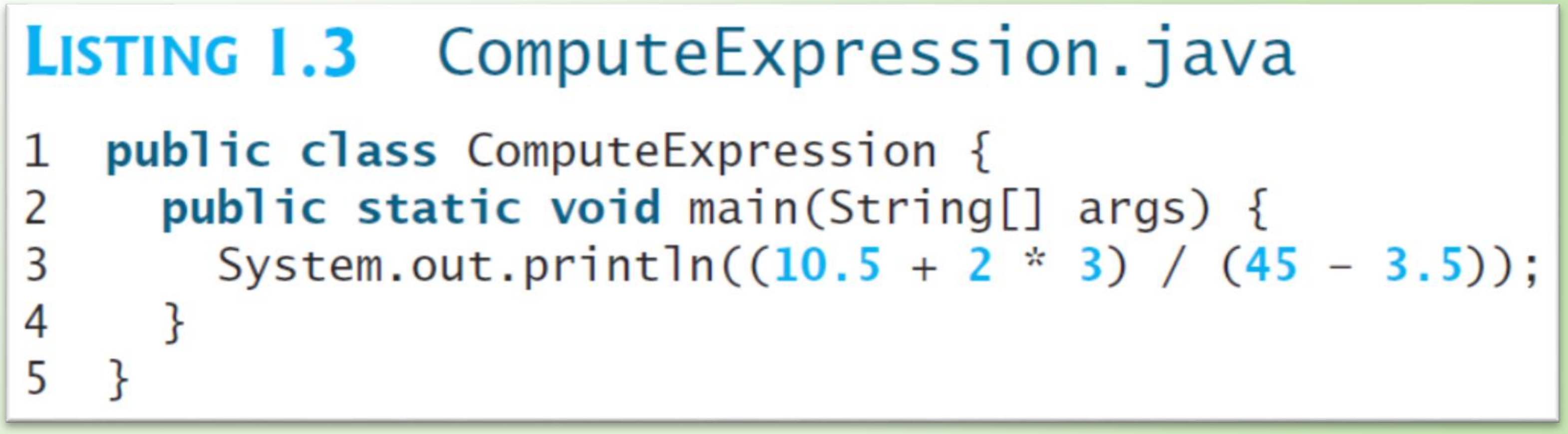
```java
public class WelcomeWithThreeMessages {
  public static void main(String[] args) {
    System.out.println("Programming is fun!");
    System.out.println("Fundamentals First");
    System.out.println("Problem Driven");
  }
}
```

<ins>Output:</ins>

```
Programming is fun!
Fundamentals First
Problem Driven
```

$$\frac{10.5 + 2 \times 3}{45 - 3.5}$$

Two Small Programs


LISTING 1.3 ComputeExpression.java

```java
public class ComputeExpression {
  public static void main(String[] args) {
    System.out.println((10.5 + 2 * 3) / (45 - 3.5));
  }
}
```

Output:

```
0.39759036144578314
```

Anatomy of a Java Program

- Class name
- Main method
- Statements
- Statement terminator
- Reserved words
- Comments
- Blocks

Class Name

Every Java program must have at least one class. Each class has a name. By convention, class names start with an uppercase letter. In this example, the class name is Welcome.

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Main Method

Line 2 defines the main method. In order to run a class, the class must contain a method named main. The program is executed from the main method.

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Statement

A statement represents an action or a sequence of actions. The statement System.out.println("Welcome to Java!") in the program in Listing 1.1 is a statement to display the greeting "Welcome to Java!“.

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Statement Terminator

Every statement in Java ends with a semicolon (;).

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

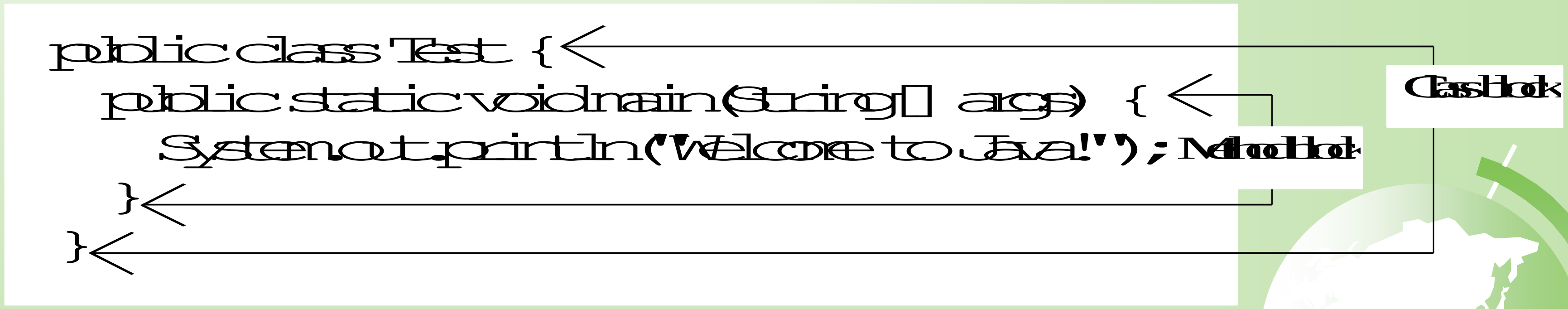
Reserved words

Reserved words or keywords are words that have a specific meaning to the compiler and cannot be used for other purposes in the program. For example, when the compiler sees the word class, it understands that the word after class is the name for the class.

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Blocks

A pair of braces in a program forms a block that groups components of a program.



```
public class Test {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Special Symbols

Character	Name	Description
{}	Opening and closing braces	Denotes a block to enclose statements.
()	Opening and closing parentheses	Used with methods.
[]	Opening and closing brackets	Denotes an array.
//	Double slashes	Precedes a comment line.
" "	Opening and closing quotation marks	Enclosing a string (i.e., sequence of characters).
;	Semicolon	Marks the end of a statement.

{ … }

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

(...)

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

;

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

// ...

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

" ... "

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Programming Style and Documentation

- Appropriate Comments
- Naming Conventions
- Proper Indentation and Spacing Lines
- Block Styles

Appropriate Comments

Include a summary at the beginning of the program to explain what the program does, its key features, its supporting data structures, and any unique techniques it uses.

Include your name, class section, instructor, date, and a brief description at the beginning of the program.

Comments

Three types of comments in Java.

Line comment: A line comment is preceded by two slashes (//) in a line.

Paragraph comment: A paragraph comment is enclosed between /* and */ in one or multiple lines.

javadoc comment: javadoc comments begin with <u>/**</u> and end with <u>*/</u>. They are used for documenting classes, data, and methods. They can be extracted into an HTML file using JDK's <u>javadoc</u> command.

Naming Conventions

- ☞ Choose meaningful and descriptive names.
- ☞ Class names:
  - – Capitalize the first letter of each word in the name. For example, the class name `ComputeExpression`.

Proper Indentation and Spacing

- Indentation
  - Indent using a tab
  - Or with three to five spaces.
  - Be CONSISTENT!

- Spacing
  - Use blank line to separate segments of the code.

Block Styles

Use end-of-line style for braces.

Next-line style

```
public class Test
{
  public static void main(String[] args)
  {
    System.out.println("Block Styles");
  }
}
```

End-of-line style

```
public class Test {
  public static void main(String[] args) {
    System.out.println("Block Styles");
  }
}
```

Programming Errors

- Syntax Errors
  - Errors that are detected by the compiler
    - Examples: mistyping a keyword, using an opening brace but not the closing, etc.
    - Usually easy to detect, because the compiler tells you where the error is and what caused them.

Programming Errors

☞ Syntax Error Example:

LISTING 1.4 ShowSyntaxErrors.java

```
1  public class ShowSyntaxErrors {
2    public static main(String[] args) {
3      System.out.println("Welcome to Java);
4    }
5  }
```

- The keyword **void** is missing before **main** in line 2.
- The string **Welcome to Java** should be closed with a closing quotation mark in line 3.

Programming Errors

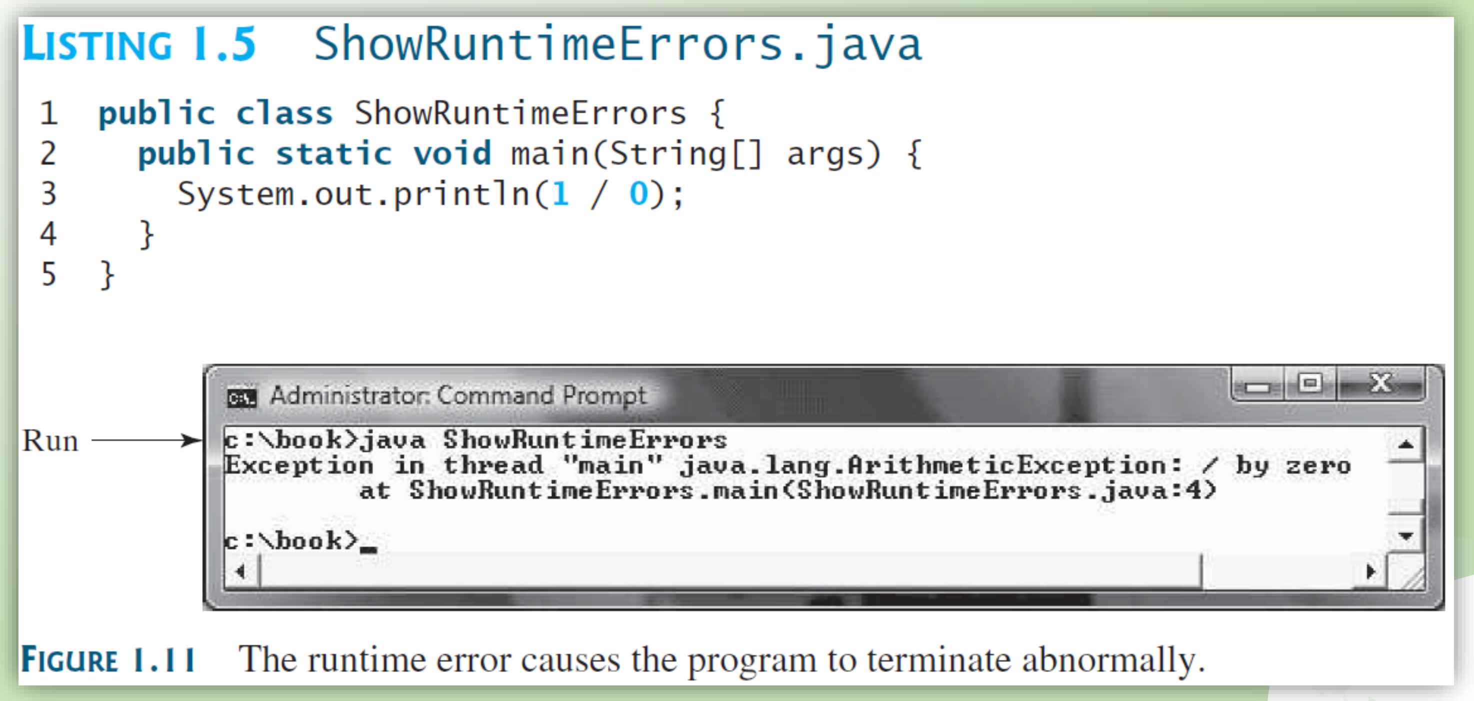
- Syntax Error Example:
  - Since a single error will often display many lines of compile errors, it is a good practice to fix errors from the top line and work downward.
  - Fixing errors that occur earlier in the program may also fix additional errors that occur later.

Programming Errors

- Runtime Errors
  - Causes the program to terminate in an abnormal way
    - Known as “crashing” or “my program crashed”
  - Often caused by input mistakes, where the user enters a value the program cannot handle
  - Another example: divide by zero

Programming Errors

- Runtime Error Example



LISTING 1.5 ShowRuntimeErrors.java

```java
public class ShowRuntimeErrors {
  public static void main(String[] args) {
    System.out.println(1 / 0);
  }
}
```

Run

```
c:\book>java ShowRuntimeErrors
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ShowRuntimeErrors.main(ShowRuntimeErrors.java:4)

c:\book>_
```

FIGURE 1.11 The runtime error causes the program to terminate abnormally.

Programming Errors

- Logic Errors
  - Produces incorrect result
  - Program does not run the way we intended
  - Usually the result of logical mistakes
  - In fact, the program “works”, but the output is wrong due to our logical mistake.
  - These errors are harder to detect.

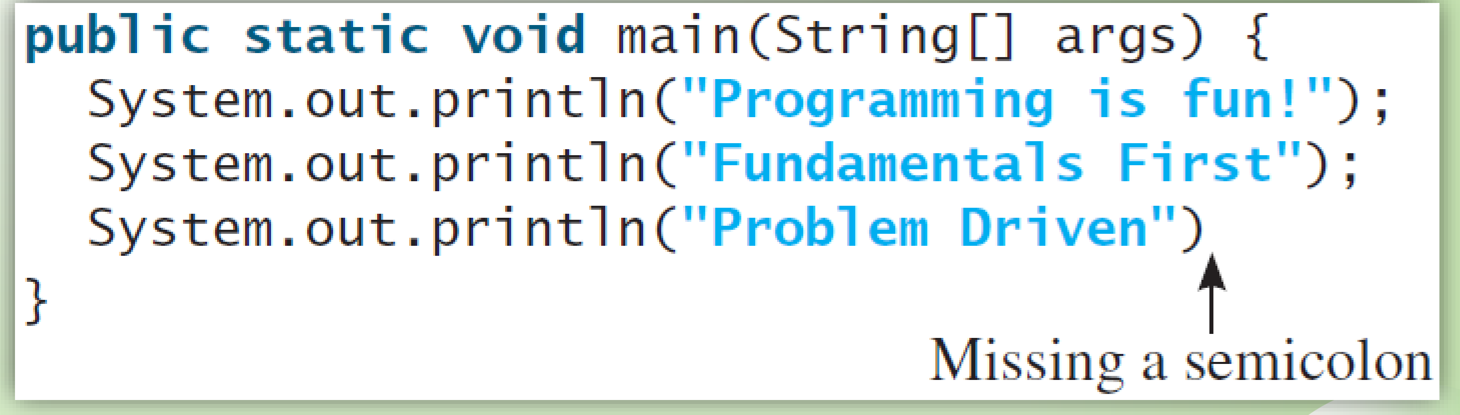
Programming Errors

- ☞ Common Errors
  - Missing a closing brace, missing a semicolon, missing quotation marks for strings, and misspelling names are common errors for new programmers.

Programming Errors

☞ Common Errors

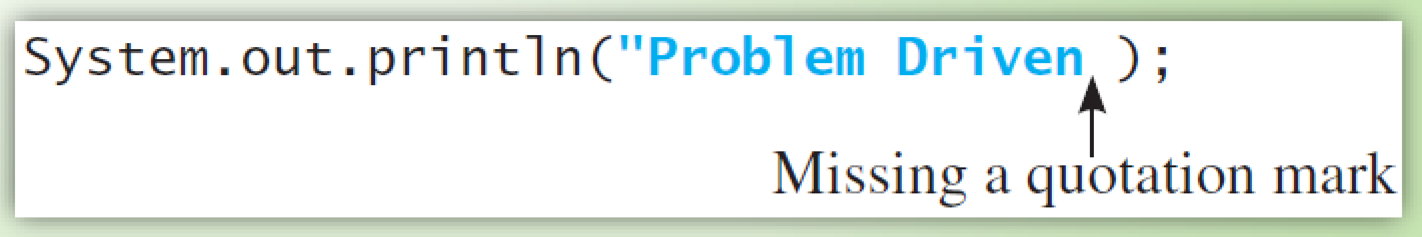
– Missing a semicolon:



```
public static void main(String[] args) {
  System.out.println("Programming is fun!");
  System.out.println("Fundamentals First");
  System.out.println("Problem Driven")
}
```

Programming Errors

- Common Errors
  - Missing quotation marks:

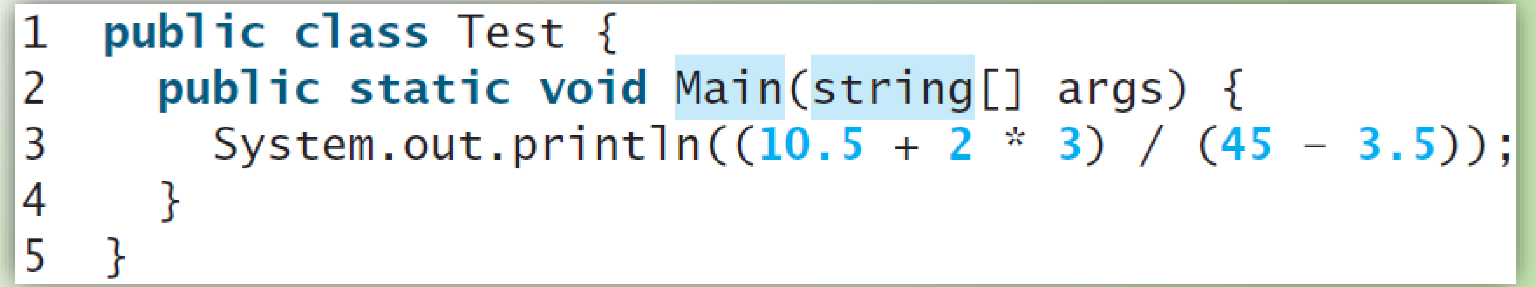


```
System.out.println("Problem Driven );
```

  - Thankfully, your IDE (such as NetBeans) will insert the closing quotation marks automatically

Programming Errors

- Common Errors
  - Misspelling Names:



```
public class Test {
  public static void Main(string[] args) {
    System.out.println((10.5 + 2 * 3) / (45 - 3.5));
  }
}
```

  - `main` is misspelled as `Main`
  - `String` is misspelled as `string`