

# Chapter 1 Introduction to Computers, Programs, and Java



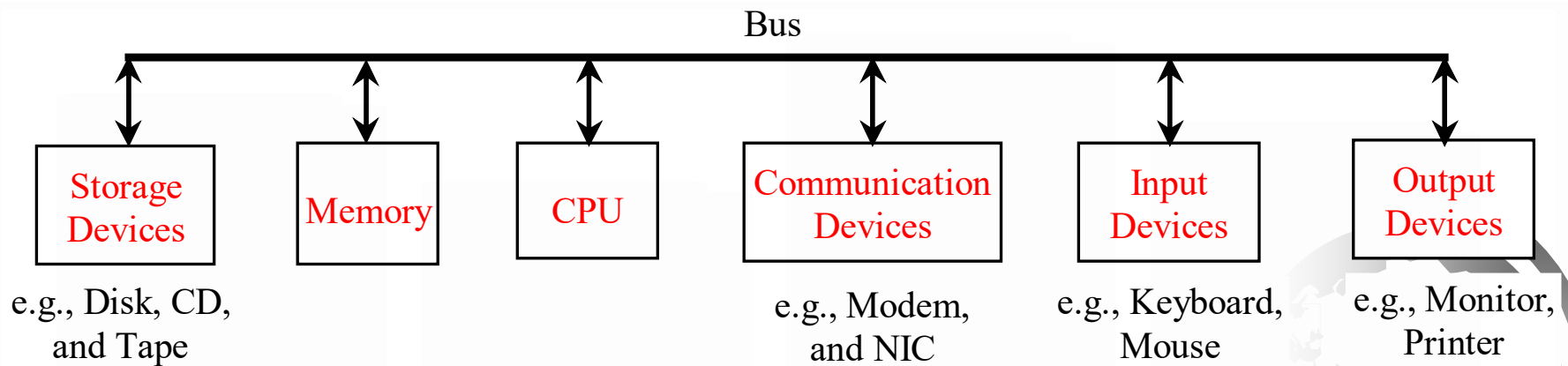
# Objectives

- To understand computer basics, programs, and operating systems (§§1.2–1.4).
- 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).
- To develop Java programs using Eclipse (§1.12).



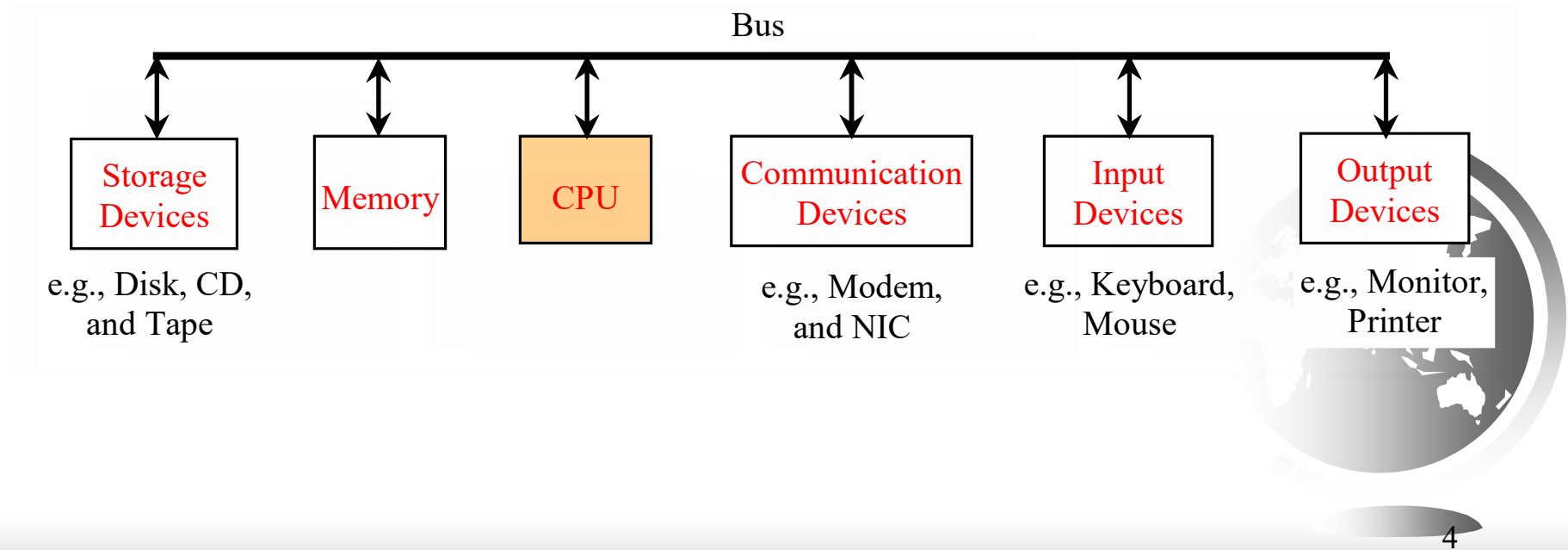
# What is a Computer?

A computer consists of a CPU, memory, hard disk, floppy disk, monitor, printer, and communication devices.



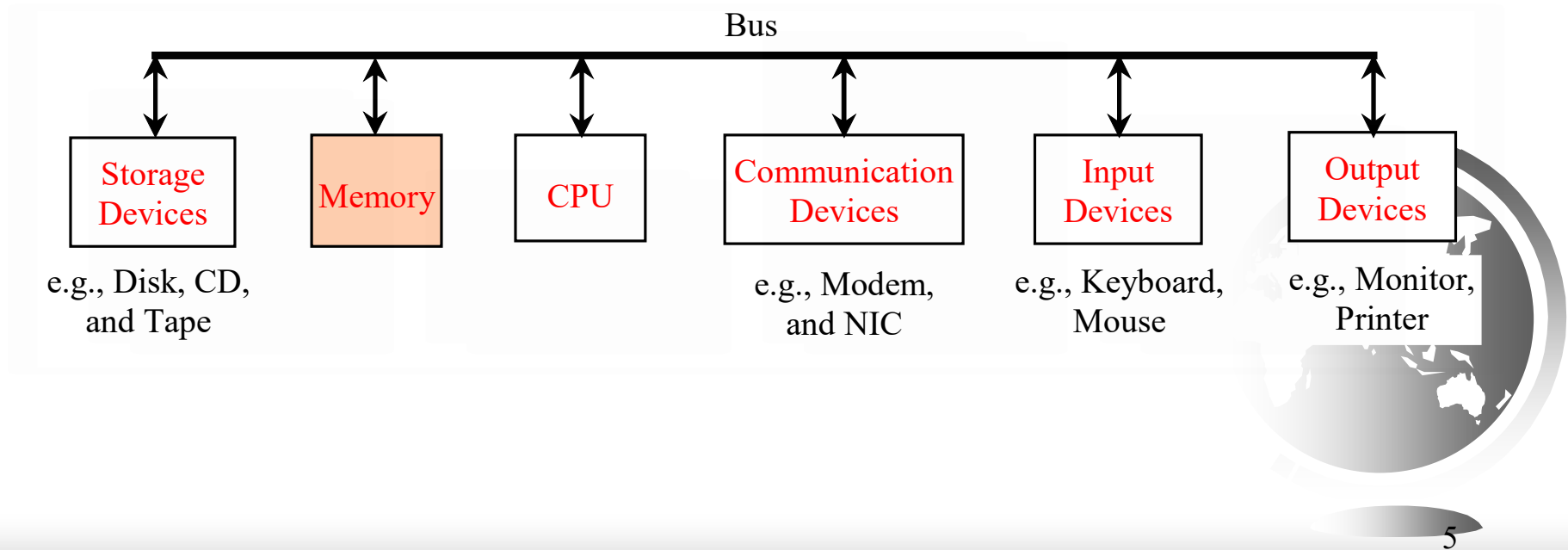
# CPU

*The central processing unit (CPU) is the brain of a computer. It retrieves instructions from memory and executes them.* The CPU speed is measured in megahertz (MHz), with 1 megahertz equaling 1 million pulses per second. The speed of the CPU has been improved continuously. If you buy a PC now, you can get an Intel Pentium 4 Processor at 3 gigahertz (1 gigahertz is 1000 megahertz).



# Memory

*Memory* is to store data and program instructions for CPU to **execute**. A memory unit is an ordered sequence of bytes, each holds eight bits. **A program and its data must be brought to memory before they can be executed**. A memory byte is never empty, but its initial content may be meaningless to your program. The current content of a memory byte is lost whenever new information is placed in it.



# How Data is Stored?

Data of various kinds, such as numbers, characters, and strings, are encoded as a series of bits (zeros and ones). Computers use zeros and ones because digital devices have two stable states, which are referred to as *zero* and *one* by convention. The programmers need not to be concerned about the **encoding and decoding of data**, which is performed automatically by the system based on the **encoding scheme**. The encoding scheme varies. For example, character 'J' is represented by 01001010 in one byte. A small number such as three can be stored in a single byte. If computer needs to store a large number that cannot fit into a single byte, it uses a number of adjacent bytes. No two data can share or split a same byte. A byte is the minimum storage unit.

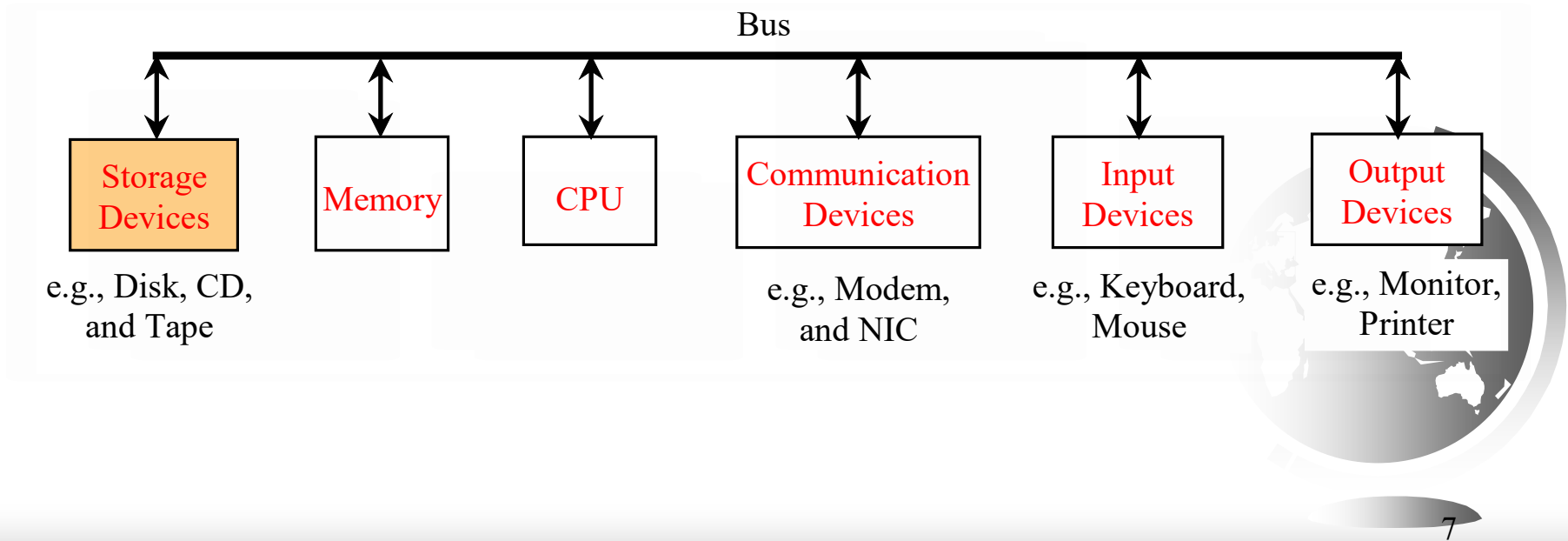
Memory address	Memory content	
.	.	
.	.	
.	.	
2000	01001010	Encoding for character 'J'
2001	01100001	Encoding for character 'a'
2002	01110110	Encoding for character 'v'
2003	01100001	Encoding for character 'a'
2004	00000011	Encoding for number 3



# Storage Devices

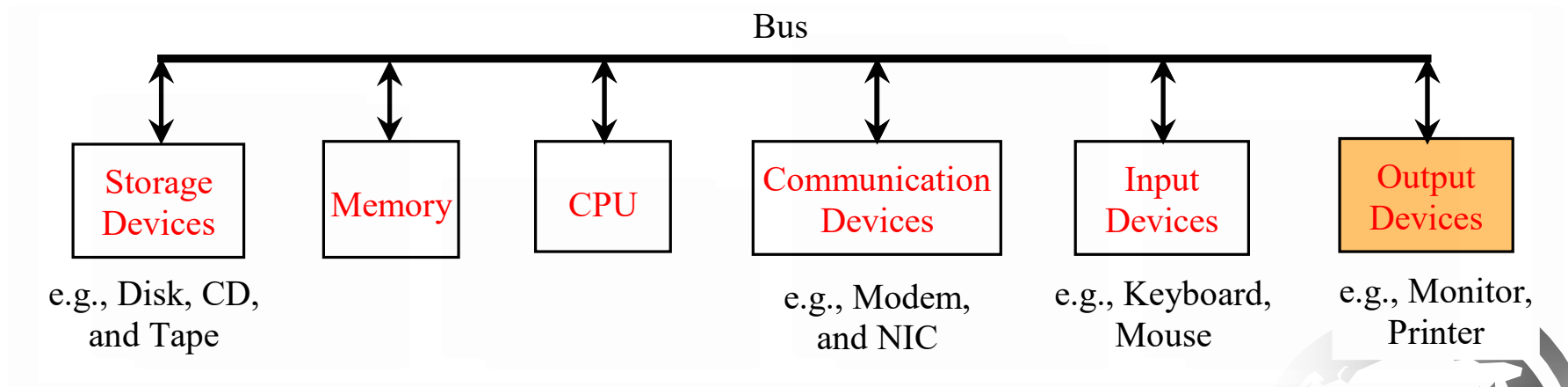
Memory is volatile, because information is lost when the power is off. *Programs and data are permanently stored on storage devices and are moved to memory when the computer actually uses them.*

There are three main types of storage devices: Disk drives (hard disks and floppy disks), CD drives (CD-R and CD-RW), and Tape drives.



# Output Devices: Monitor

The monitor displays information (text and graphics). The resolution and dot pitch determine the quality of the display.





# Monitor Resolution and Dot Pitch

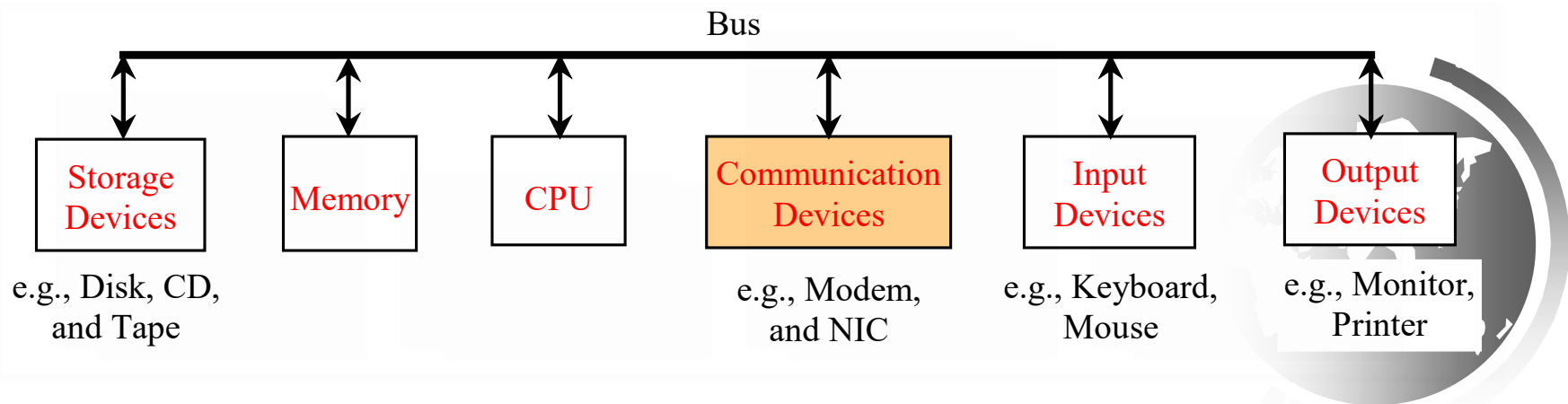
*resolution* The *screen resolution* specifies the number of pixels in horizontal and vertical dimensions of the display device. *Pixels* (short for “picture elements”) are tiny dots that form an image on the screen. A common resolution for a 17-inch screen, for example, is 1,024 pixels wide and 768 pixels high. The resolution can be set manually. The higher the resolution, the sharper and clearer the image is.

*dot pitch* The *dot pitch* is the amount of space between pixels, measured in millimeters. The smaller the dot pitch, the sharper the display.



# Communication Devices

A *regular modem* uses a phone line and can transfer data in a speed up to 56,000 bps (bits per second). A *DSL* (digital subscriber line) also uses a phone line and can transfer data in a speed 20 times faster than a regular modem. A *cable modem* uses the TV cable line maintained by the cable company. A cable modem is as fast as a DSL. Network interface card (*NIC*) is a device to connect a computer to a local area network (LAN). The LAN is commonly used in business, universities, and government organizations. A typical type of NIC, called *10BaseT*, can transfer data at 10 mbps (million bits per second).



# 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
```



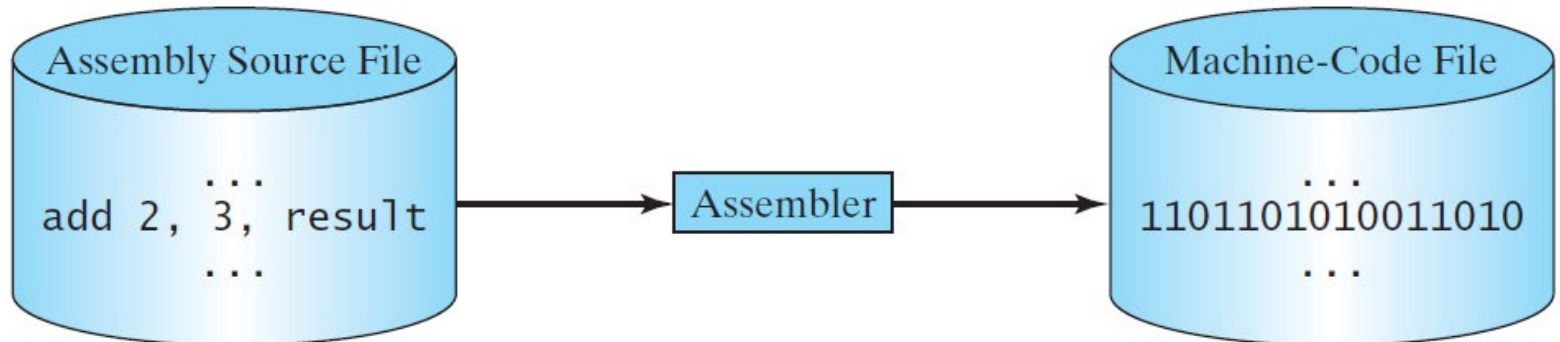
# Programming Languages

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.*

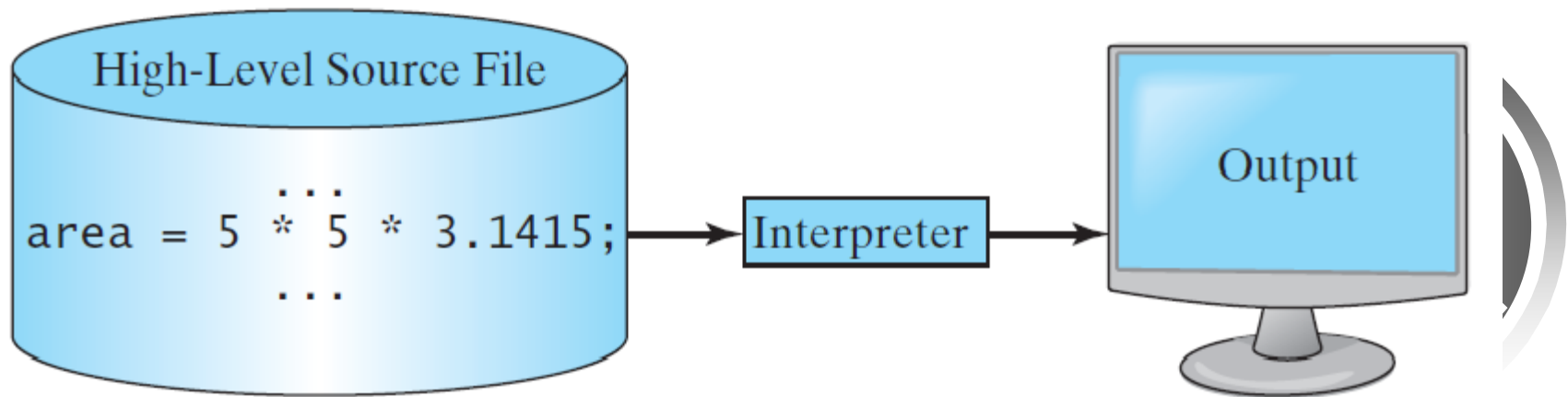
The translation can be done using another programming tool called an *interpreter* or a *compiler*.





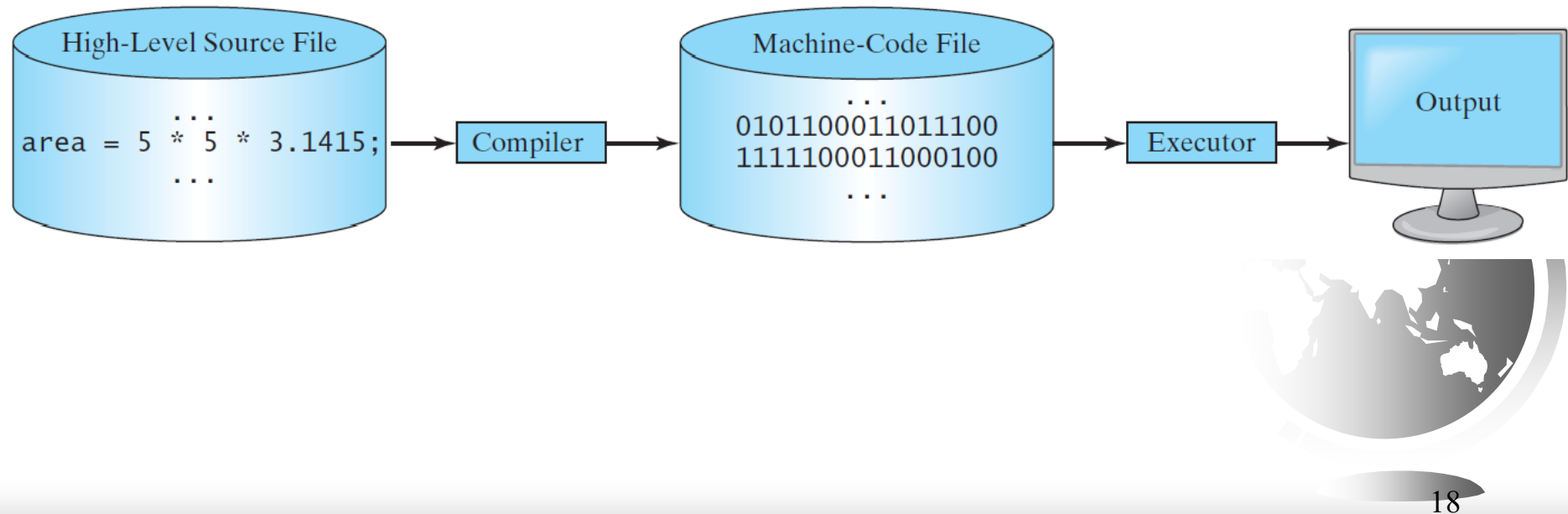
# Interpreting Source Code

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.**



# Java's History

- 1991 - James Gosling begins work on Java project (Originally named “Oak” for the oak tree outside his office.)
- 1995 - Sun releases first public implementation as Java 1.0
- 1998 - JDK 1.1 release downloads tops 2 million
- 1999 - Java 2 is released by Sun
- 2005 - Approximately 4.5 million developers use Java technology
- 2007 - Sun makes all of Java's core code available under open-source distribution terms.
- Early History Website:

<http://www.java.com/en/javahistory/index.jsp>



# Java's History

- Java SE 6 (December 11, 2006)
- Java SE 7 (July 28, 2011)
- Java SE 8 (March 18, 2014)
- Java SE 9 (September 21, 2017)
- Java SE 10 (March 20, 2018)
- Java SE 11 (September 25, 2018)
- Java SE 12 (March 19, 2019)
- Java SE 13.0.2 (January 14, 2020)
- Java SE 14.0.2 (July 14, 2020)
- Java SE 15.0.2 (January 19, 2021)
- Java SE 16.0.2 (July 20, 2021)



# Java SE

## Java Client Technologies

Java 3D, Java Access Bridge, Java Accessibility, Java Advanced Imaging, Java Internationalization and Localization Toolkit, Java Look and Feel, Java Media Framework (JMF), Java Web Start (JAWS), JIMI SDK

## Java Platform Technologies

Java Authentication and Authorization Service (JAAS), JavaBeans, Java Management Extension (JMX), Java Naming and Directory Interface, RMI over IIOP, Java Cryptography Extension (JCE), Java Secure Socket Extension

## Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files

The Java Cryptography Extension enables applications to use stronger versions of cryptographic algorithms. JDK 9 and later offer the stronger cryptographic algorithms by default.

The unlimited policy files are required only for JDK 8, 7, and 6 updates earlier than 8u161, 7u171, and 6u181. On those

## Java SE downloads

- [Java SE 22](#)
- [Java SE 21](#)
- [Java SE 20](#)
- [Java SE 19](#)
- [Java SE 18](#)
- [Java SE 17](#)
- [Java SE 16](#)
- [Java SE 15](#)
- [Java SE 14](#)
- [Java SE 13](#)
- [Java SE 12](#)
- [Java SE 11](#)

# Java SE 22 Archive Downloads

Go to the [Oracle Java Archive](#) page.

The JDK is a development environment for building applications using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java™ platform.

**WARNING:** Older versions of the JDK are provided to help developers debug issues in older systems. **They are not updated with the latest security patches and are not recommended for use in production.**

For production use Oracle recommends downloading the latest JDK version.

Only developers and enterprise administrators should download these releases.

For current Java releases, please visit [Oracle Java SE Downloads](#).

## Major New Functionality

### 1. Language

#### → Unnamed Variables & Patterns

Enhance the Java programming language with unnamed variables and unnamed patterns, which can be used when variable declarations or nested patterns are required but never used. Both are denoted by the underscore character, `_`.

[See JEP 456](#)

### 1.1 Language Previews

#### → Statements before `super(...)` (Preview)

In constructors in the Java programming language, allow statements that do not reference the instance being created to appear before an explicit constructor invocation. This is a [preview language feature](#).

[See JEP 447](#)



1996  
01-23

开发代号为Oak(橡树)

引入JDBC(Java DataBase Connectivity)  
支持内部类  
引入Java Bean;  
引入RMI(Remote Method Invocation)  
引入反射(仅用于内省)

1997  
02-19

1998  
12-08

引入的新特性包括:  
引入集合框架;  
对字符串常量做内存映射;  
引入JIT(Just In Time)编译器  
引入对打包的Java文件进行数字签名;  
引入控制授权访问系统资源的策略工具;  
引入JFC(Java Foundation Classes), 包括Swing 1.0, 拖放和Java2D类库;  
引入Java插件;  
在JDBC中引入可滚动结果集,BLOB,CLOB,批量更新和用户自定义类型;  
在Applet中添加声音支持.

引入的新特性包括:  
引入Java Sound API;  
jar文件索引;  
对Java的各个方面都做了大量优化和增强

2000  
05-08

2004  
02-06

XML处理;  
Java打印服务;  
引入Logging API;  
引入Java Web Start;  
引入JDBC 3.0 API;  
引入断言;  
引入Preferences API;  
引入链式异常处理;  
支持IPv6;  
支持正则表达式;  
引入Image I/O API.

2004  
09-30

引入泛型;  
增强循环,可以使用迭代方式;  
自动装箱与自动拆箱;  
类型安全的枚举;  
可变参数;  
静态引入;  
元数据(注解);  
引入Instrumentation

2006  
12-11

WebService元数据  
脚本语言支持  
JTable的排序和过滤  
更简单,更强大的JAX-WS  
轻量级Http Server  
嵌入式数据库 Derby

2011  
07-28

switch语句块中允许以字符串作为分支条件;  
在创建泛型对象时应用类型推断;  
在一个语句块中捕获多种异常;  
支持动态语言;  
运用List<String> tempList = new ArrayList<>(); 即泛型实例化  
类型自动推断  
语法上支持集合,而不一定是数组  
新增一些取环境信息的工具方法  
Boolean类型反转,空指针安全,参与位运算  
两个char间的equals  
安全的加减乘除  
map集合支持并发请求,且可以写成 Map map = {name:"xxx",age:  
18};  
引入Java NIO.2开发包;  
数值类型可以用二进制字符串表示,并且可以在字符串表示中添加下划  
线;  
钻石型语法(在创建泛型对象时应用类型推断);  
null值得自动处理.

2014  
03-19

引入Lambda 表达式;  
管道和流;  
新的日期和时间 API;  
默认的方法;  
类型注解;  
Nashorn javascript 引  
擎;  
并行累加器;  
并行操作  
内存错误移除

2016  
09-22

Jigsaw 项目;模块化源码  
简化进程API  
轻量级 JSON API  
钱和货币的API  
改善锁争用机制  
代码分段缓存  
智能Java编译, 第二阶段  
HTTP 2.0客户端  
Knulla计划: Java的REPL实现



# 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

Sun describes  
it as

[www.cs.armstrong.edu/liang/JavaCharacteristics.pdf](http://www.cs.armstrong.edu/liang/JavaCharacteristics.pdf)



# 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.

没有C、C++语言中的指针；没有多重继承；没有操作符重载；



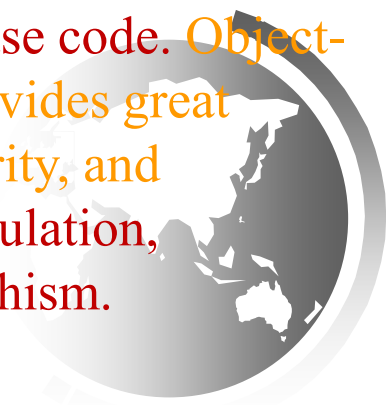
# Characteristics of Java

- 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.



# 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.



# 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).



# 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.

强类型机制、异常处理、  
垃圾内存自动搜集机制等





# 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.

解释性语言  
无指针、数组越界检查等



# 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.



# 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.



# 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 a trade off between speed and security



# 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.



# 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 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.

We use J2SE to introduce Java programming.



# Popular Java IDEs

- Eclipse
- IntelliJ IDEA
- NetBeans
- VS Code





# A Simple Java Program

## Listing 1.1

//This program prints Welcome to Java!

注释

public class Welcome {

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

}

类声明

方法声明

Animation

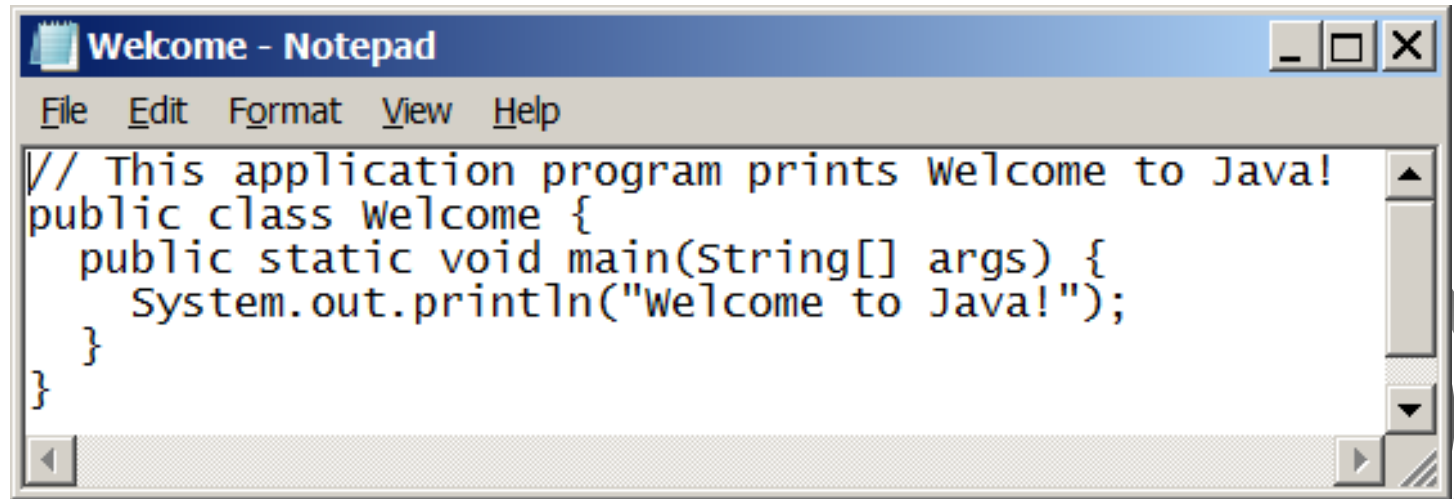
Welcome

Run

IMPORTANT NOTE: If you cannot run the buttons, see [www.cs.armstrong.edu/liang/javaslidenote.doc](http://www.cs.armstrong.edu/liang/javaslidenote.doc).

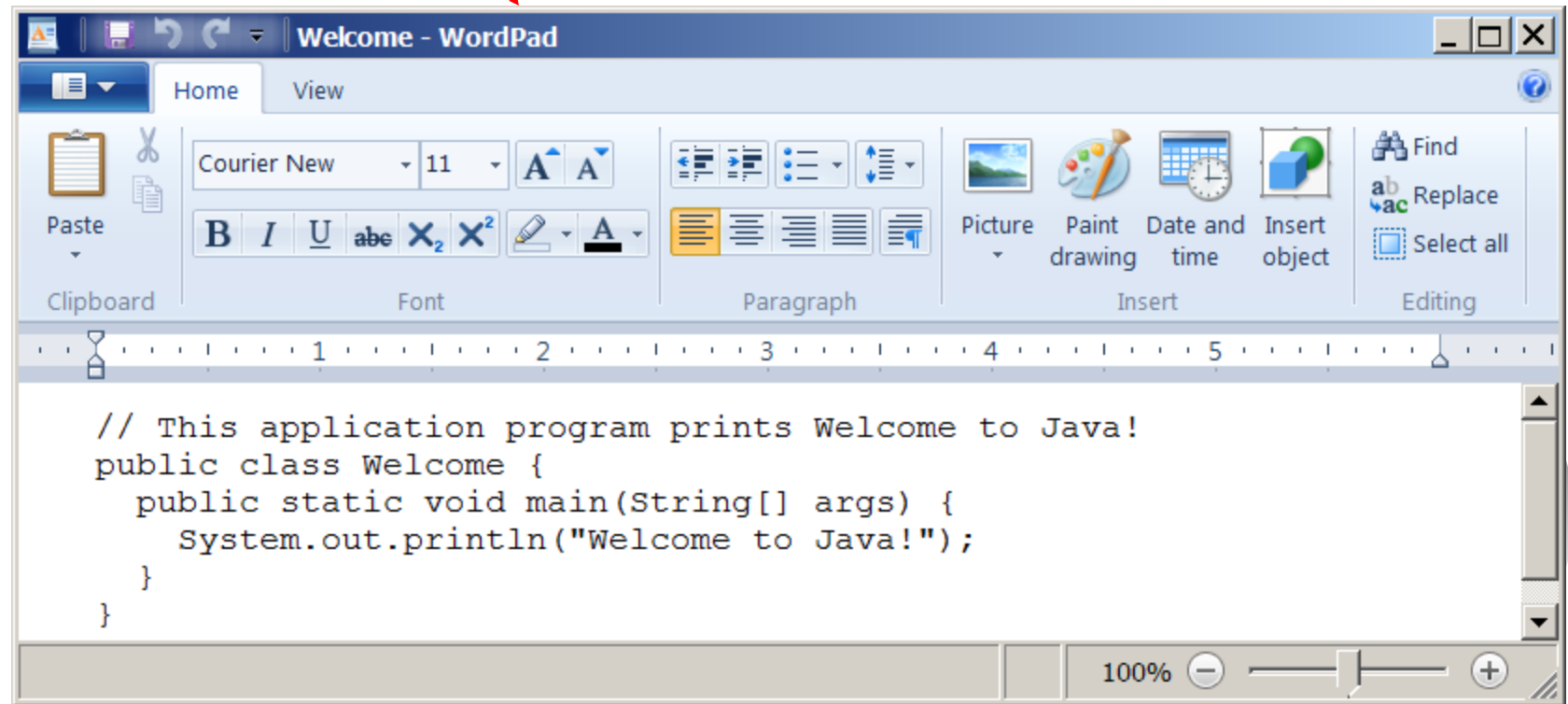
# Creating and Editing Using NotePad

To use NotePad, type  
notepad Welcome.java  
from the DOS prompt.



# Creating and Editing Using WordPad

To use WordPad, type  
write Welcome.java  
from the DOS prompt.



```
File Edit Format View Help
// 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)

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

Bytecode (generated by the compiler for JVM to read and interpret)

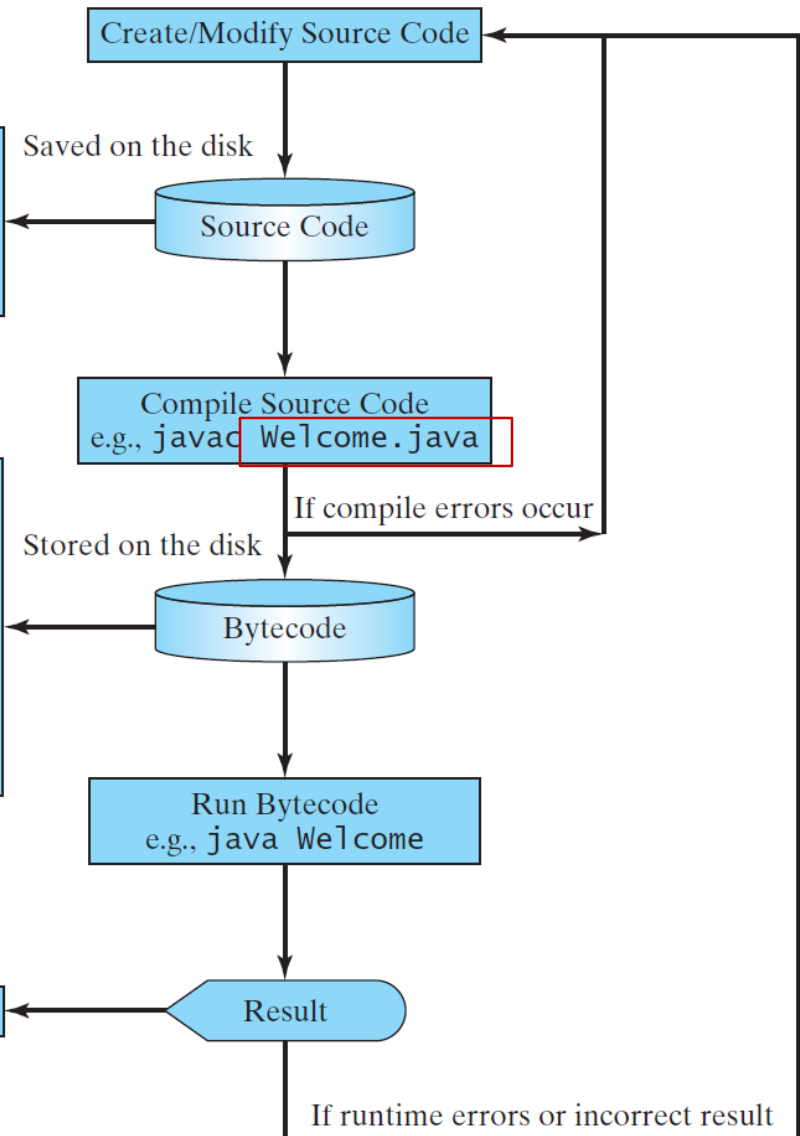
```
...
Method Welcome()
  0 aload_0
  ...

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

“Welcome to Java” is displayed on the console

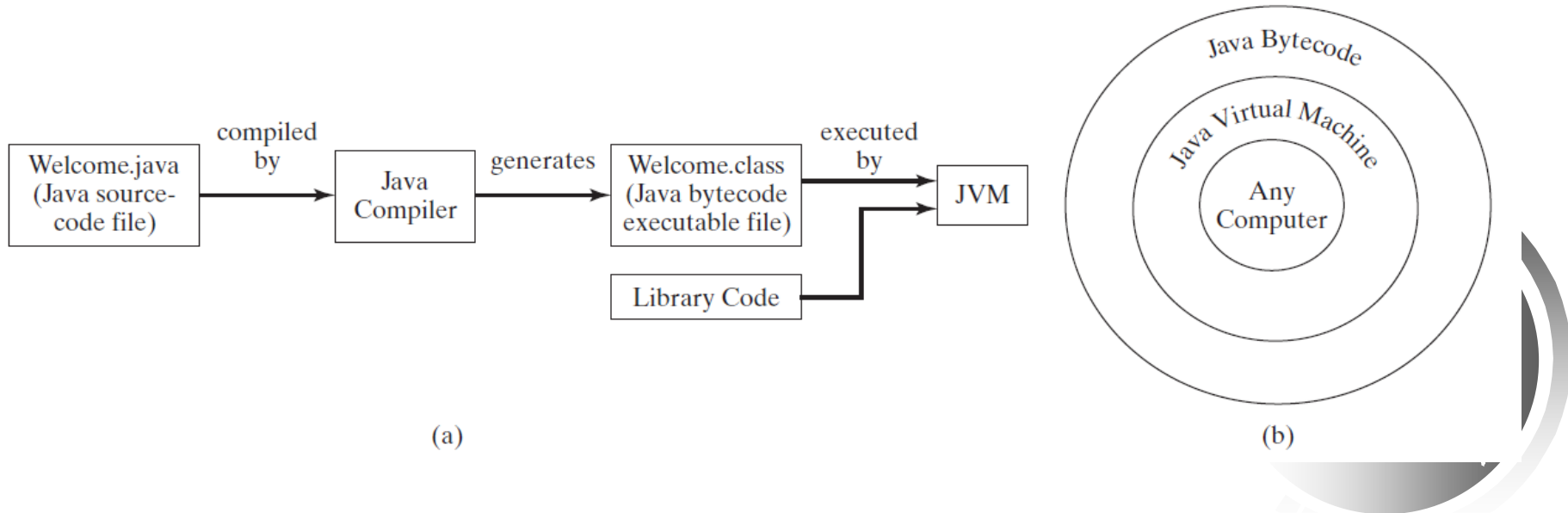
Welcome to Java!

# Creating, Compiling, and Running Programs



# Compiling Java Source Code

- 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.**



# 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!");  
    }  
}
```



# Trace a Program Execution

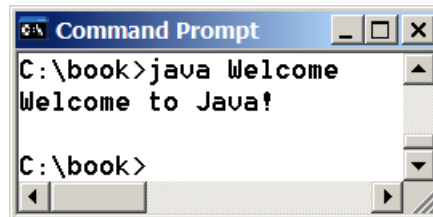
Execute statement

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



# 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!");  
    }  
}
```



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

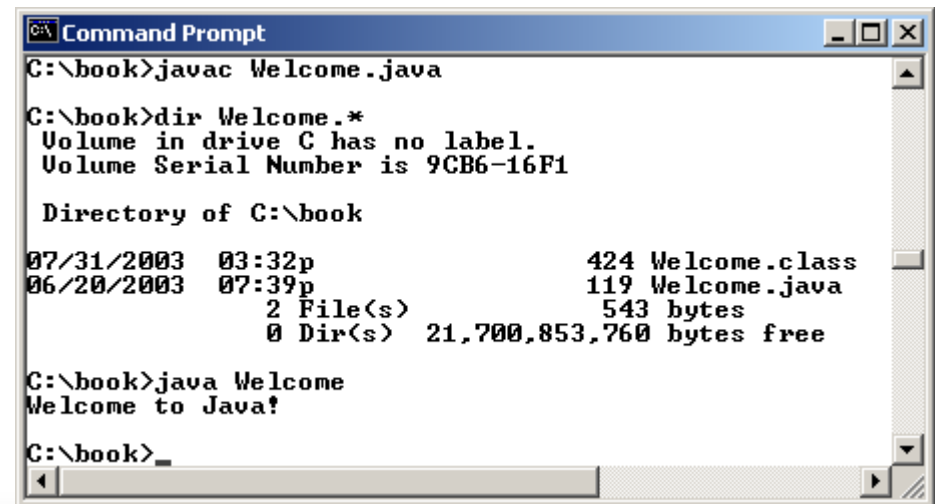
print a message to the  
console



# Compiling and Running Java from the Command Window

- Set path to JDK bin directory
  - set path=c:\Program Files\java\jdk1.8.0\bin
- Set classpath to include the current directory
  - set classpath=.
- Compile
  - javac Welcome.java
- Run
  - java Welcome

变量	值
CLASSPATH	.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOM...
JAVA_HOME	D:\Program Files\Java\jdk1.8.0_31
Path	%JAVA_HOME%\bin;%JAVA_HOME%\jre...



```
C:\book>javac Welcome.java
C:\book>dir Welcome.*
Volume in drive C has no label.
Volume Serial Number is 9CB6-16F1

Directory of C:\book

07/31/2003  03:32p                424 Welcome.class
06/20/2003  07:39p                119 Welcome.java
                2 File(s)                543 bytes
                0 Dir(s)  21,700,853,760 bytes free

C:\book>java Welcome
Welcome to Java!

C:\book>
```

# 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!");  
    }  
}
```

Diagram illustrating blocks in the code:

- Class block:** Indicated by a long arrow pointing from the opening brace of `public class Test {` to the closing brace of `}`.
- Method block:** Indicated by a shorter arrow pointing from the opening brace of `public static void main(String[] args) {` to the closing brace of `}`.






# 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!");  
    }  
}
```

// ...

还有其他两种注释

1./\* ....\*/

2.\*\* ...\*/

```
// 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.





# 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 two spaces.
- 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
  - Detected by the compiler
- Runtime Errors
  - Causes the program to abort
- Logic Errors
  - Produces incorrect result



# Syntax Errors

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



# Runtime Errors

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



# Logic Errors

```
public class ShowLogicErrors {  
    public static void main(String[] args) {  
        System.out.println("Celsius 35 is Fahrenheit degree ");  
        System.out.println((9 / 5) * 35 + 32);  
    }  
}
```

