



# 面向对象程序设计（Java）

黄绍辉

厦门大学计算机系

[hsh@xmu.edu.cn](mailto:hsh@xmu.edu.cn)

# 教材

- 👉 书名: Introduction to Java Programming (Brief Version or Comprehensive Version)
- 👉 版本: 10th Edition
- 👉 作者: Y. Daniel Liang
- 👉 网上有很多电子书, 自行搜索下载。也可以购买实体书, 不过巨贵, 不推荐.....
- 👉 由于尚未有中文译本, 本PPT尽量使用中文。



# 推荐参考书

- 👉 书名：Thinking in Java
- 👉 中文译名：Java编程思想
- 👉 版本：Fourth Edition
- 👉 作者：Bruce Eckel
- 👉 网上很容易搜到电子书，中英文都有。实体书也有卖，价格不详。
- 👉 早期版本中文翻译得比较吐血，不如直接看英文，不知目前版本如何。



# 学习指南

## ☞ 英文不好，能看懂英文教材吗？

- 首先，计算机编程类的教材词汇量比较少，基本上可以猜出来意思；其次，可以在线翻译或者词霸；最后，你还可以找几本中文的Java教材先入门。不过因为本课程用的是JDK1.8版本，你能找到的教材很可能过时了.....

## ☞ 要怎么样才能学好Java？

- 其实编程类的课程，学好的诀窍都是一样的：**上机多练**。练就一个字，我只说一次.....



# Java集成开发环境（Integrated Development Environment, IDE）

- ☞ NetBeans，源自Oracle公司的开源IDE，有中文版的，可捆绑JDK下载，推荐使用。
- ☞ Eclipse，源自IBM公司的开源IDE，因为需要配置插件，且是全英文菜单，慎重推荐使用。
- ☞ 如果不愿意使用IDE，UltraEdit/EditPlus之类的也可以将就用用，虽然会极大增加你的开发工作量。至于记事本就算了，那只是哥上课的时候才偶尔用用。



# 工具下载

☞ JDK下载链接:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>



该页面有2个下载，建议下载右边的NetBeans+JDK8，这样连开发工具也解决了。

☞ Eclipse IDE for Java Developers 下载链接:

<http://www.eclipse.org/downloads/>

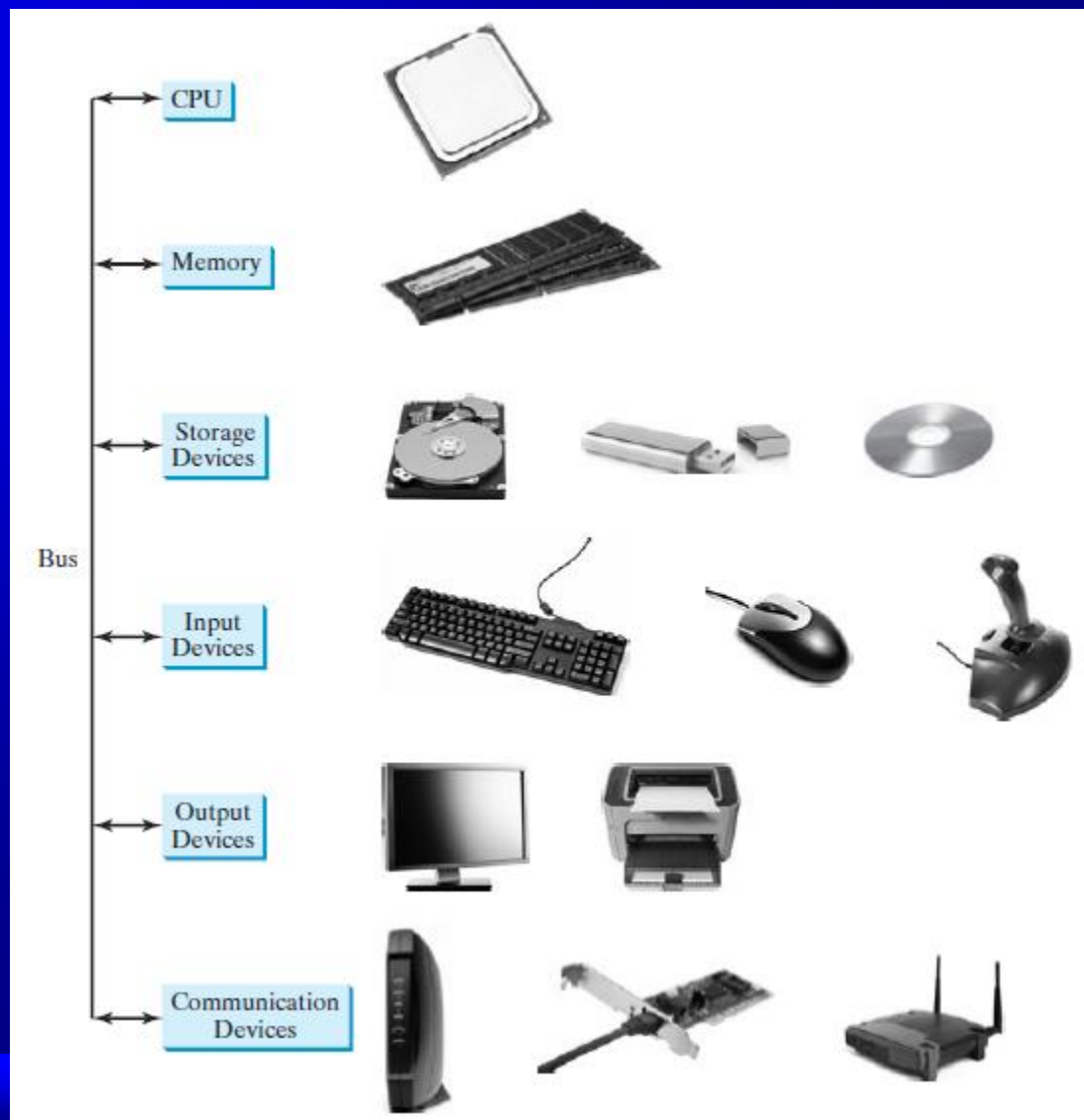


# Chapter 1 Introduction to Computers, Programs, and Java



# 什么是计算机？

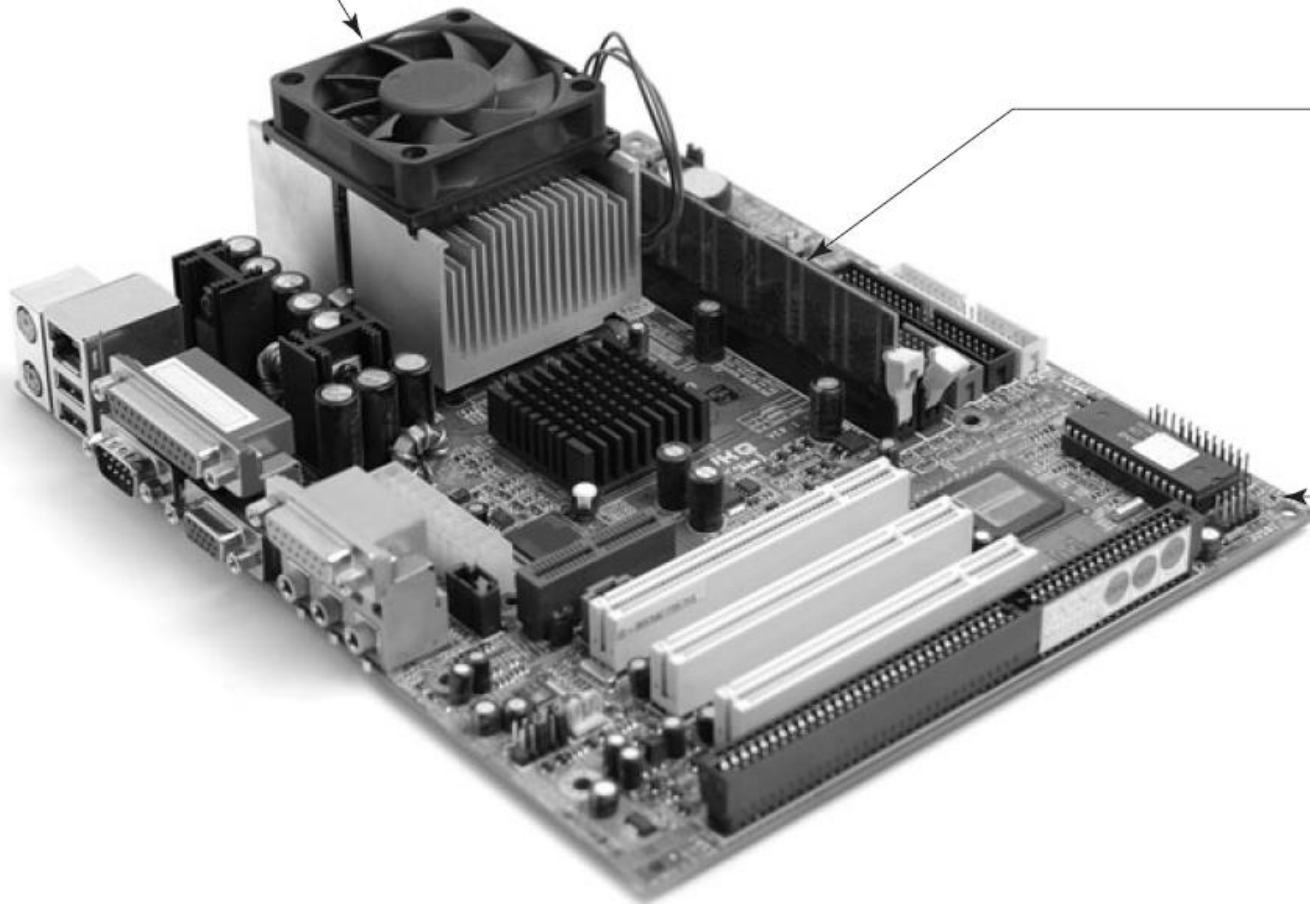
👉 计算机由软硬件组成，硬件是那些看得见的物理设备。这些硬件通过总线连接。软件则通过看不见的指令控制硬件工作。





# 主板把所有的部件连接在一起

CPU is placed  
under the fan

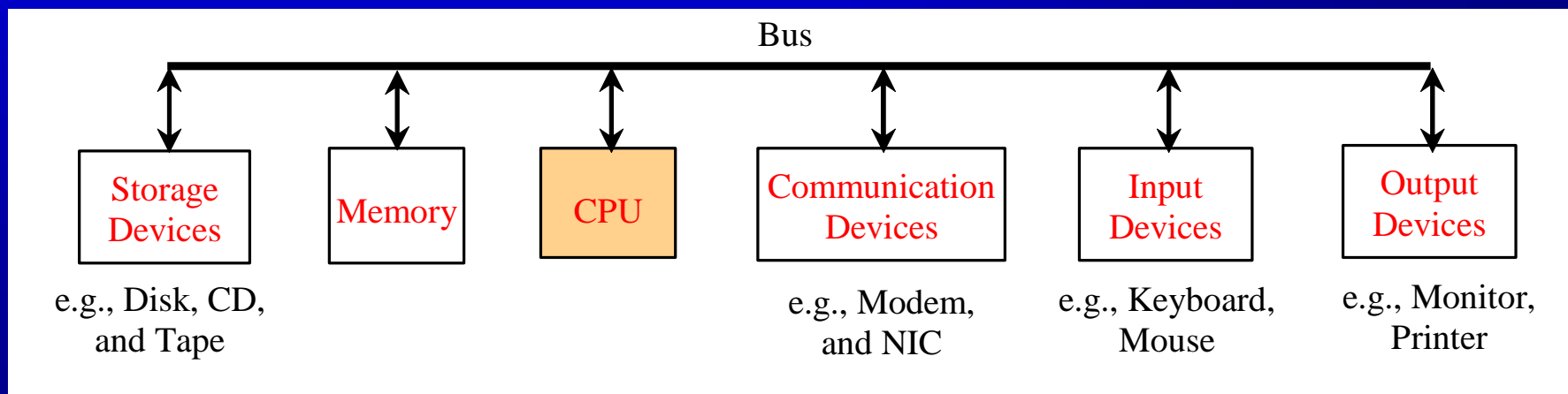


Memory

Motherboard

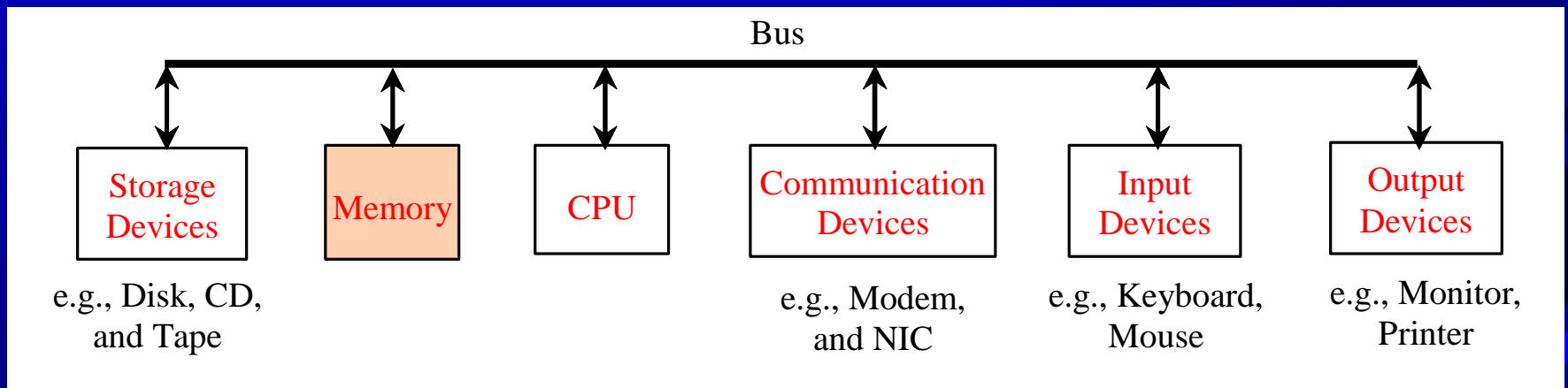
# CPU (Central Processing Unit)

- CPU由两个部件组成：控制单元（control unit）和算术/逻辑单元（arithmetic/logic unit）
- CPU有个内部时钟，可以定时发出脉冲。这些脉冲用于控制计算机运行。显然时钟越快，CPU就越快。衡量脉冲频率的指标是主频，单位是MHz。



# 主存 (Memory)

- 主存用于在计算机运行时存储数据。任何数据在进入CPU运算之前，都必须存储在主存。
- 主存以字节 (byte) 为单位存储数据。不过表示存储容量的最小单位是位 (bit)，表示1个的1或者0。
- 1字节=8位。
- 为了表示主存的每一个存储位置，主存必须按字节编址。



# 如何存储数据？

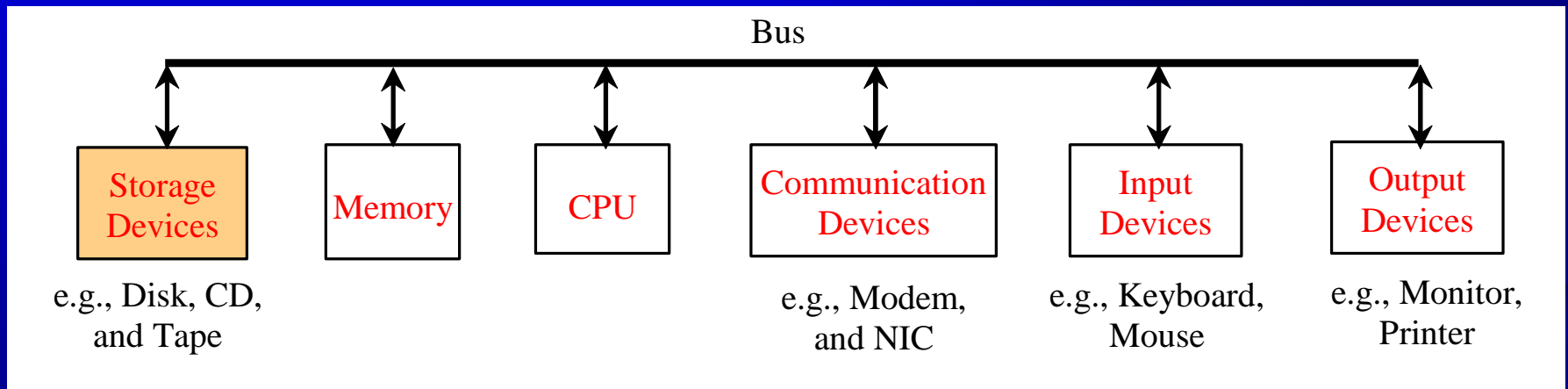
- 主存按照字节为单位，将存储空间从0开始编址。不使用位来编址，是因为这个单位实在太小，难以使用。
- 例如，字符‘J’在计算机中用01001010这个二进制串表示，占1个字节大小。一个比较小的数，例如3，也只需要1个字节即可存储。更大的数需要多个字节才能存储，这时候计算机会连续分配几个字节给它。
- 字节是最小的存储单位。换句话说，计算机存储一个数，至少需要1字节。

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



# 存储设备（外存）

- 主存是易失性存储设备，掉电后所有信息就丢失了。为了长期保存数据，需要用外部存储设备，例如：磁盘（Disk drives），CD（CD-R或CD-RW），磁带，U盘等等。



# 磁盘（Magnetic disk drives）

- ➡ 磁盘以磁介质作为存储的媒介。容量从几个G到几个T都有。
- ➡ 典型的磁盘有硬盘（hard disks）以及软盘（floppy disks）。



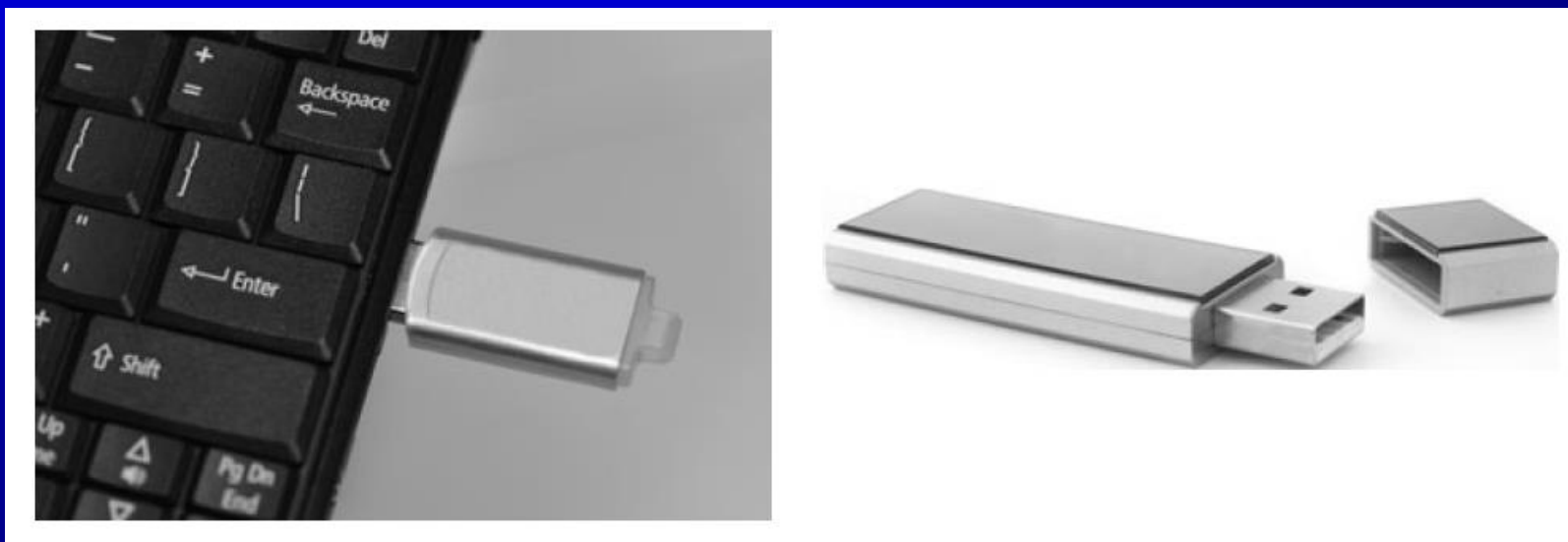
# CD和DVD

- ☞ CD指的是光盘，有只读的CD-R和可读写的CD-RW，容量在700MB。
- ☞ DVD也有两种，只读的DVD-R和可读写的DVD-RW，容量可以达到4.7GB。



# U盘（USB Flash Drives）

- ☞ U盘是通过USB（Universal serial bus）线连接到计算机中的。容量可以达到256GB。





# 编程语言

机器语言→汇编语言→高级语言

- 机器语言是计算机能直接运行的唯一语言，它是二进制的指令形式。
- 机器语言的可读性显然是很差的，例如，把两个数相加，你可能要编写这么一段代码：

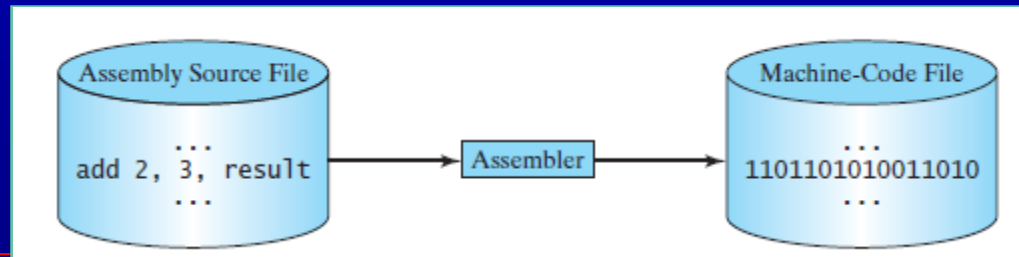
```
1101101010011010
```



# 编程语言

机器语言→汇编语言→高级语言

- 汇编语言是机器语言的符号版，由一组助记符组成，所以可读性大大提高。
- 不过显然计算机并不认识汇编语言，所以汇编语言需要借助汇编程序，将源代码翻译成机器语言之后才能够运行。
- 仍然以加法为例，这时候你可以这样写代码了：  
**ADDF3 R1, R2, R3**



# 编程语言

机器语言→汇编语言→高级语言

- 高级语言看上去已经和英文差不多了，所以容易学习和使用。例如计算半径为5的圆的面积，你可以直接这样写：

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



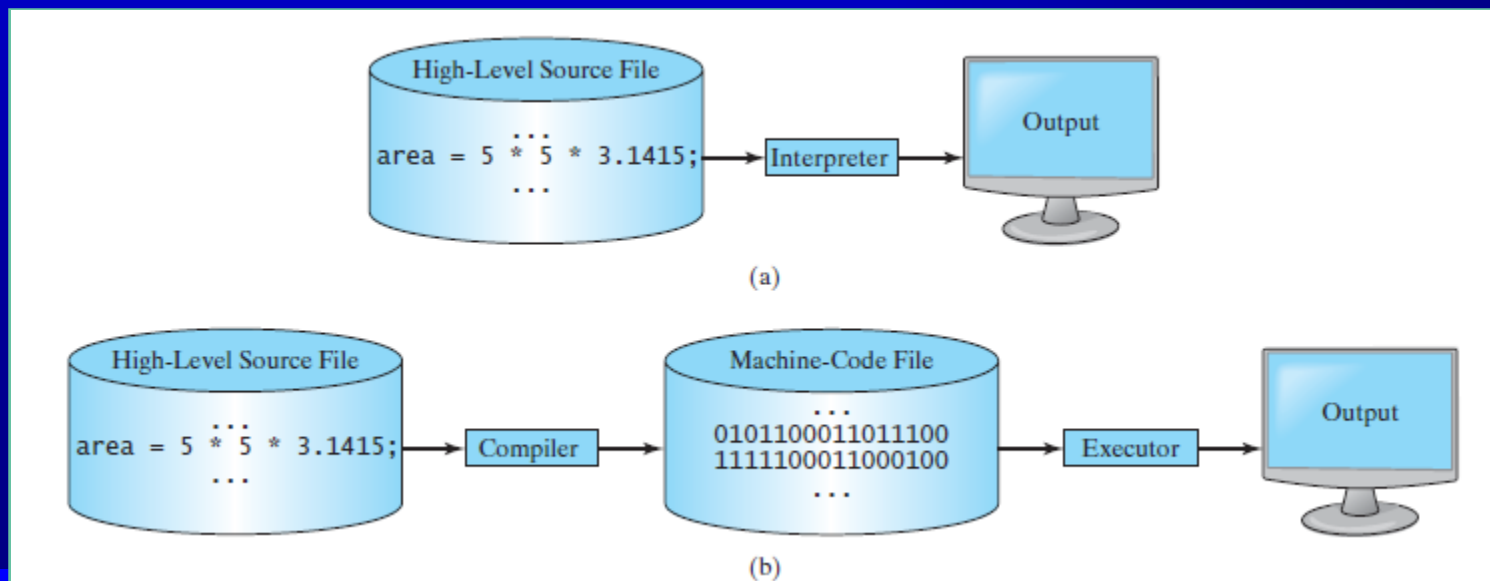
# 比较流行的高级语言

- ☞ COBOL (COmmon Business Oriented Language)
- ☞ FORTRAN (FORmula TRANslation)
- ☞ BASIC (Beginner All-purpose Symbolic Instructional Code)
- ☞ Pascal (named for Blaise Pascal)
- ☞ Ada (named for Ada Lovelace)
- ☞ C (whose developer designed B first)
- ☞ Visual Basic (Basic-like visual language developed by Microsoft)
- ☞ Delphi (Pascal-like visual language developed by Borland)
- ☞ C++ (an object-oriented language, based on C)
- ☞ C# (a Java-like language developed by Microsoft)
- ☞ Java (We use it in the book)



# 源代码的编译

- ➡ 高级语言需要转换为机器语言才能运行。通常有两种做法：
- 解释执行：利用解释器，将源代码逐行解释成机器语言，边解释边运行，如图(a)所示。
  - 编译执行：利用编译器，将源代码一次性编译成机器语言然后再运行，如图(b)所示。这种方式运行速度比较快。

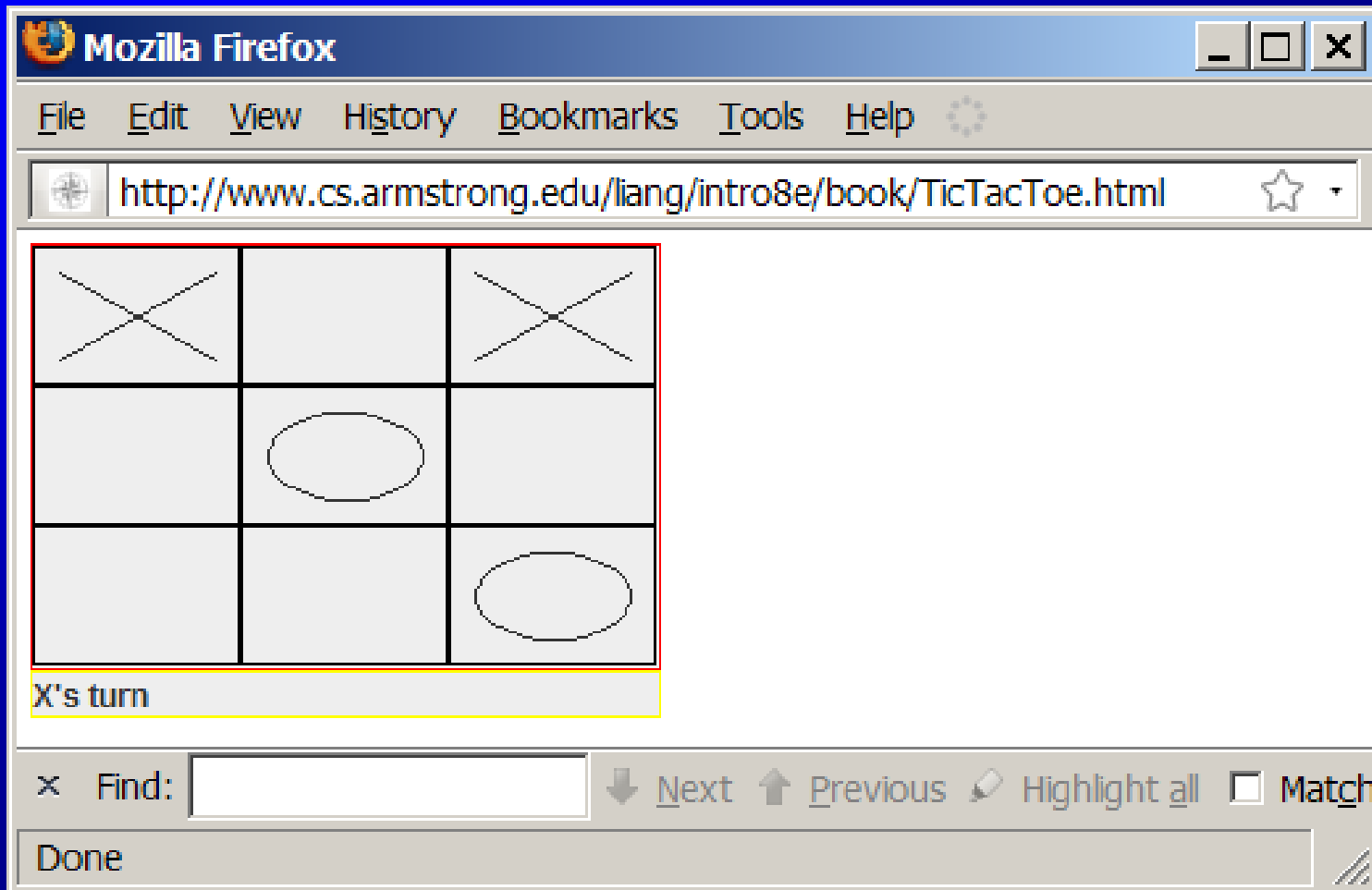


# Java可以用来做什么？

- ☞ Java天生是网络型的语言，特别适合开发和网络相关的应用。
- ☞ Java能够嵌入网页运行，这个叫Applet。
- ☞ Java可以开发大型Web应用，例如网站。
- ☞ Java可以做嵌入式开发，例如运行在智能手机上。



# Java开发的小应用程序(Applets)



# 在Android上使用Java开发的例子





# Java的历史

- ☞ James Gosling and Sun Microsystems
- ☞ Oak, 1991
- ☞ Java, May 20, 1995, Sun World
- ☞ Oracle, 2010



# Java的特点

- ☞ 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的特点

- ➡ 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脱胎于C++，吸收了C++的长处，并且对C++做了简化和提高。不少人更愿意把Java叫做“C++-”，原因是它长得象C++，但是明显比C++简单。



# Java的特点

- ☞ 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 是天生面向对象的语言。面向对象编程（Object-oriented programming, OOP）是区别于传统的面向过程编程的全新的思维方式。C语言是经典的面向过程的编程语言，以函数为单位来搭建程序。

面向对象编程是以对象为单位来搭建程序的，有助于代码的重用，并且大大提高了编程的效率。

面向对象编程有三个特性：封装性、继承性和多态性。

# Java的特点

- ➡ 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

分布式计算指的是利用网络联合多台计算机一起计算。**Java**的网络功能，使得分布式计算的程序设计变得容易。

**Java**是天生的网络型语言，对它来说，从网上收发数据，和从文件读取数据，编程思路上是没有任何区别的。



# Java的特点

- ➡ 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是解释型的语言，任何Java程序会被编译成字节码文件，这个文件需要安装Java虚拟机（Java Virtual Machine, JVM）才能运行。



# Java的特点

- ➡ 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**的编译器能够检测很多其它编译器忽略的错误，这是因为它的语法规则更加严格和规范。

**Java**还提供了运行时的异常处理机制，用来增强程序的健壮性。



# Java的特点

- ☞ 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实现了很多的安全机制，用来保护你的系统。所以一个Java程序要搞死计算机是几乎不可能的。





# Java的特点

- ☞ 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的口号是：一次编写，到处运行（Write once, run anywhere）

这是由于Java虚拟机（JVM）是一个虚拟的机器，这个虚拟的机器在任何平台上的工作特点是完全一致的，是一台理想的机器。

# Java的特点

- ☞ 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

由于Java是体系中立的理想机器，所以它编译出来的字节码文件在各个平台上都没有区别。因此Java程序可以完全跨平台运行而无需重编译。

# Java的特点

- ☞ 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**的性能大约是你能够批评它的唯一地方了。由于它毕竟是一种半编译半解释型的语言，速度上比起纯编译型的语言如C/C++肯定要慢一些，但是比起纯解释型的语言还是会快不少，例如BASIC。

很多时候，稳定性比运行速度要重要得多，因为后者毕竟可以通过提高硬件水平来弥补。所以**Java**适合用来做大型的应用。



# Java的特点

- ☞ 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**天生支持多线程，所以在**Java**上面开发多线程的程序是很轻松愉快的一件事情。

界面程序（GUI）是一定要用多线程实现的，否则一些较大的运算量经常会造成界面假死。如果使用MFC做界面，你就不得不要自己写多线程。在**Java**中你完全不用考虑这个问题。

# Java的特点

- ☞ 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程序的升级特别容易，新的代码会被自动加载而无需重编译原有代码。开发者无需特意写一个升级程序，而用户也无需重新安装，即可完成程序的透明升级。



# JDK的版本号

- ☞ 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的版本

## ☞ Java Standard Edition (J2SE)

- J2SE用于开发客户端或单机版的应用程序，也可以用来写小应用程序（applet）

## ☞ Java Enterprise Edition (J2EE)

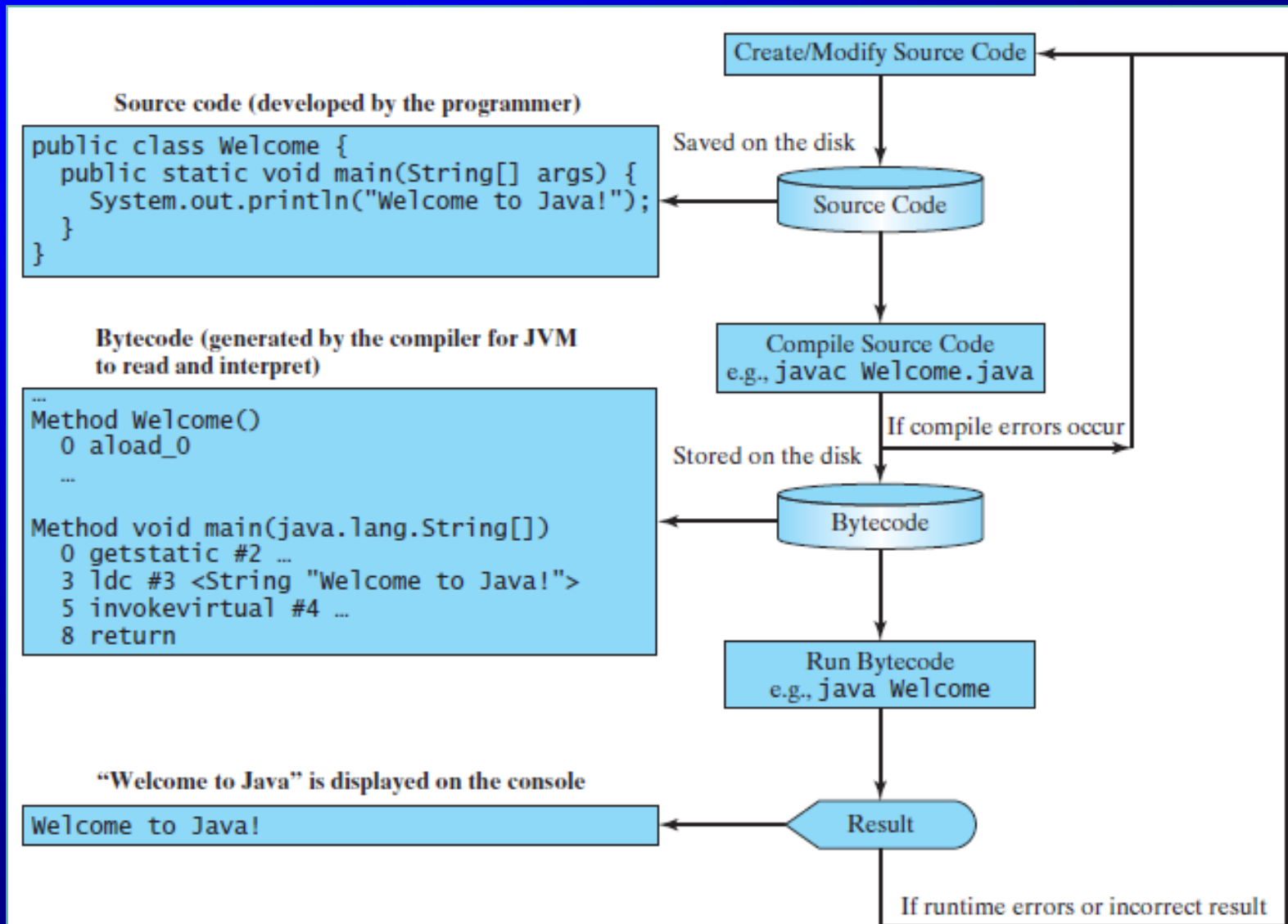
- J2EE用于开发服务端的大型应用程序，如Java servlets和Java Server Pages（JSP）

## ☞ Java Micro Edition (J2ME).

- J2ME用于开发移动设备的应用程序，如手机游戏。

本课程讲解J2SE版本。其实三个版本在语法和编程思路上完全没有区别，只是可使用的支持库有所不同。

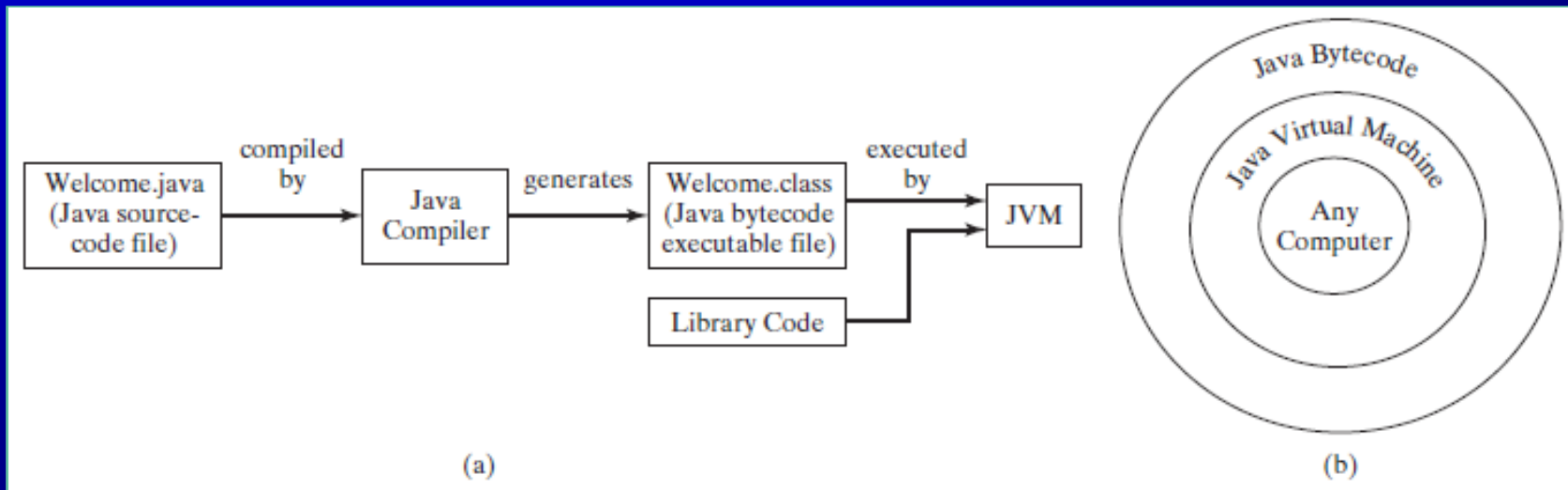
# Java开发的三步骤：编辑，编译，运行程序





# 源代码的编译

- ➡ 为了跨平台，Java只将源代码编译成字节码文件，然后由Java虚拟机（Java Virtual Machine, JVM）直接运行。
- ➡ 不同平台下的字节码文件，是完全相同的，所以你的Java程序完全无需考虑跨平台的问题。当然，为了运行字节码文件，不同平台需要安装对应的JVM。



# Java环境变量的设置

☞ 为了方便手工编译运行Java程序，只需正确设置一个环境变量（使用IDE的话不设置也行，低版本的Java还要设置其它环境变量）：

✓ 变量名：**path**（如已存在，请修改原值）

变量值：在原值的最前面加 **C:\Program Files\Java\jdk1.8.0\_25\bin;**

注意：(1)上述路径其实是你机器上JDK的实际安装路径加上\bin，所以请根据实际情况修改；  
(2) 环境变量有两种：用户变量和系统变量。随便设置一个就行。如path不存在，自己新建。

# 找个简单的Java程序练练手

## Listing 1.1

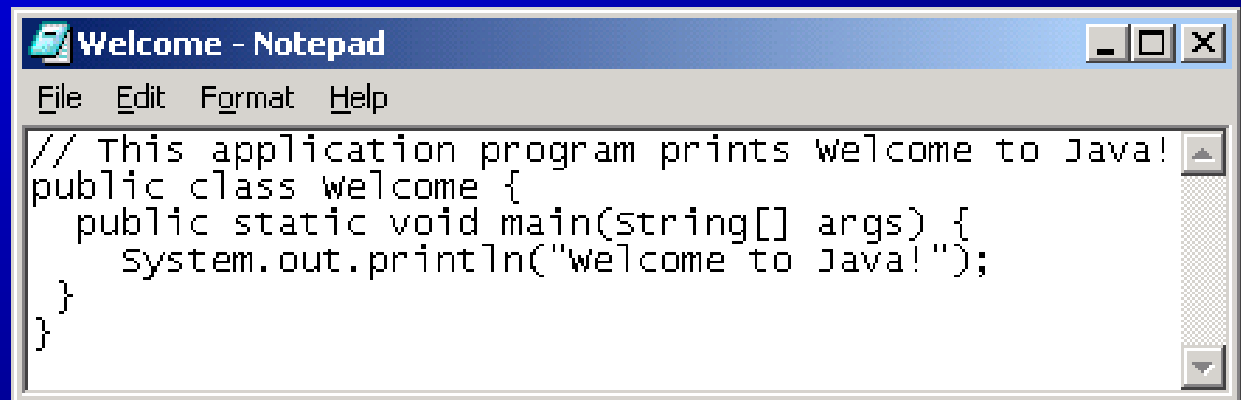
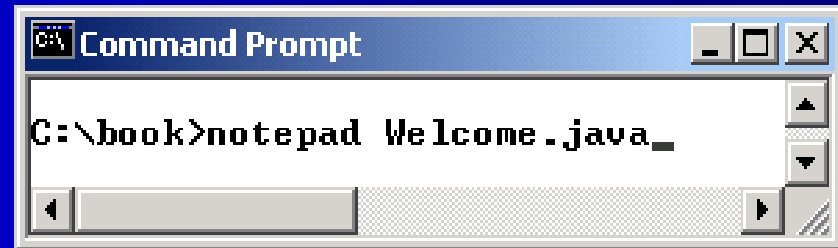
```
//This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```



# 使用记事本编辑源代码

☞ 在DOS窗口下输入这个命令：

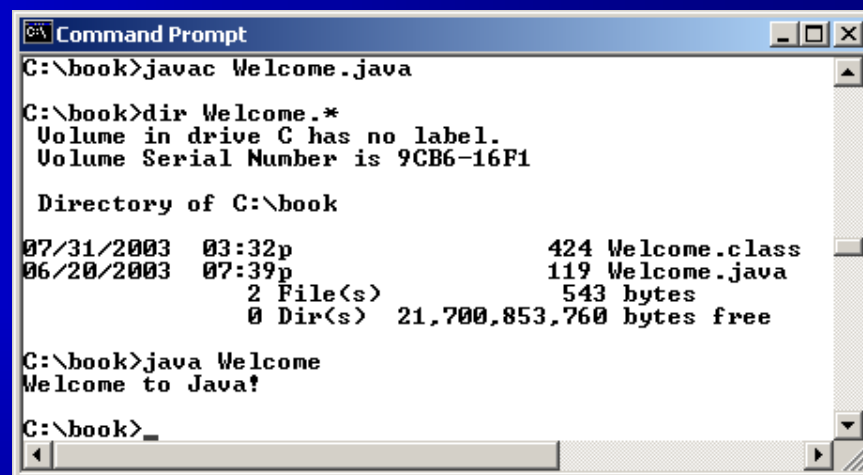
notepad Welcome.java



# 编译运行Java程序

## 编译

- **javac** Welcome.java
- 一切顺利的话，会生成Welcome.class文件。



```
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>_
```

## 运行

- **java** Welcome
- 运行Welcome.class字节码文件，**注意不要写.class**。



# 解剖一下我们的第一个程序

- ➡ 注释
- ➡ 关键字
- ➡ 修饰符
- ➡ 语句
- ➡ 语句块
- ➡ 类
- ➡ 方法
- ➡ main方法



# 注释

- Java支持三种注释
  1. 行注释：`//` 表示注释到行尾
  2. 段注释：`/*` 中间的所有文字都是注释，可以跨行，也可以跨段 `*/`
  3. 文档注释：`/**` 中间的所有文字都是注释，可以跨行，也可以跨段，用于javadoc命令自动生成帮助文档。 `*/`



# 关键字

关键字又叫保留字，是Java留用的一些单词，有特定的含义，如class, public, static, void.....





# 修饰符

修饰符用来修饰变量、方法等，如本例的**public**和**static**，它们能够对变量起限制作用，如限定变量的作用范围。



# 语句

语句表示一个具体的操作动作，Java的语句以分号;  
作为结束。



# 语句块

夹在一对大括号 `{ }` 中的所有语句合起来叫做一个语句块。

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

Class block

Method block



# 类

类是Java的核心，也是面向对象编程的最基本概念。关于类的介绍后面会详细展开，现在你只要知道：无论多小的Java程序，总是由一个或多个类组成。没有定义类的Java程序是不存在的。



# 方法

`System.out.println`是什么？其实它是一个方法，或者说是一个功能模块。Java的方法和C语言的函数相比，概念上是完全一样的。只是Java非要叫method，不愿意叫function，所以你同样可以把它叫函数，不过这不是Java的术语。



# main方法

main方法是Java程序的运行入口，不过Java的main比起C要复杂那么一点点，它必须写成下面这个形式才可以：

```
public static void main(String[] args) {  
    // 各种语句;  
}
```



# Java源文件命名

- ☞ Java源文件不能随便取名，它一定要取这个文件中**public class**的那个**class**名，包括大小写也必须是一样的。例如上面的例子，源文件名一定要叫Welcome.java。
- ☞ 为什么有这么奇怪的规定？因为每一个class都可以有自己的main函数，而main函数就是程序的入口，因此JVM只会进入public class所拥有的那个main函数开始运行。

# 用Java显示一个对话框

Java做界面是非常容易的事情，例如下面这个例子：

```
import javax.swing.*;
```

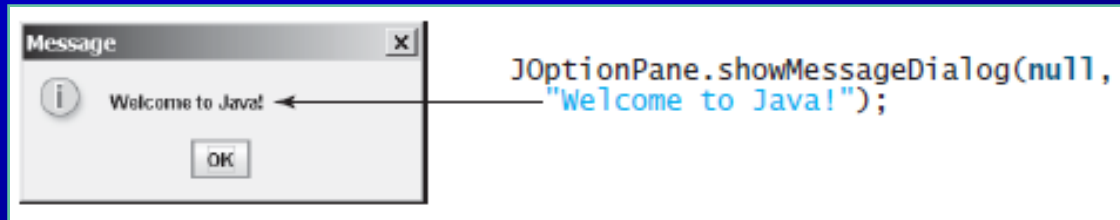
```
public class Welcome2 {
```

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

```
        JOptionPane.showMessageDialog(null, "Welcome to  
Java!");
```

```
    }
```

```
}
```



高亮的那两行就是要显示对话框的两个要素。你照着抄就可以了。



# 编程风格的忠告

## ☞ 适当的注释

- 多使用注释可以有助于自己 and 他人理解你的程序

## ☞ 适当的缩进和空格

- 可读性是很重要的，有助于查错

```
System.out.println(3+4*4);
```

← Bad style

```
System.out.println(3 + 4 * 4);
```

← Good style



# 代码风格

- ☞ Java有两种代码风格。一种是Next-line，一种是End-of-line，区别在于左括号{是否另起一行。
- ☞ 第一种风格是Windows下面的C/C++程序员比较喜欢的。一般人写Java用第二种风格，以便和Java库函数的写法保持一致。

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

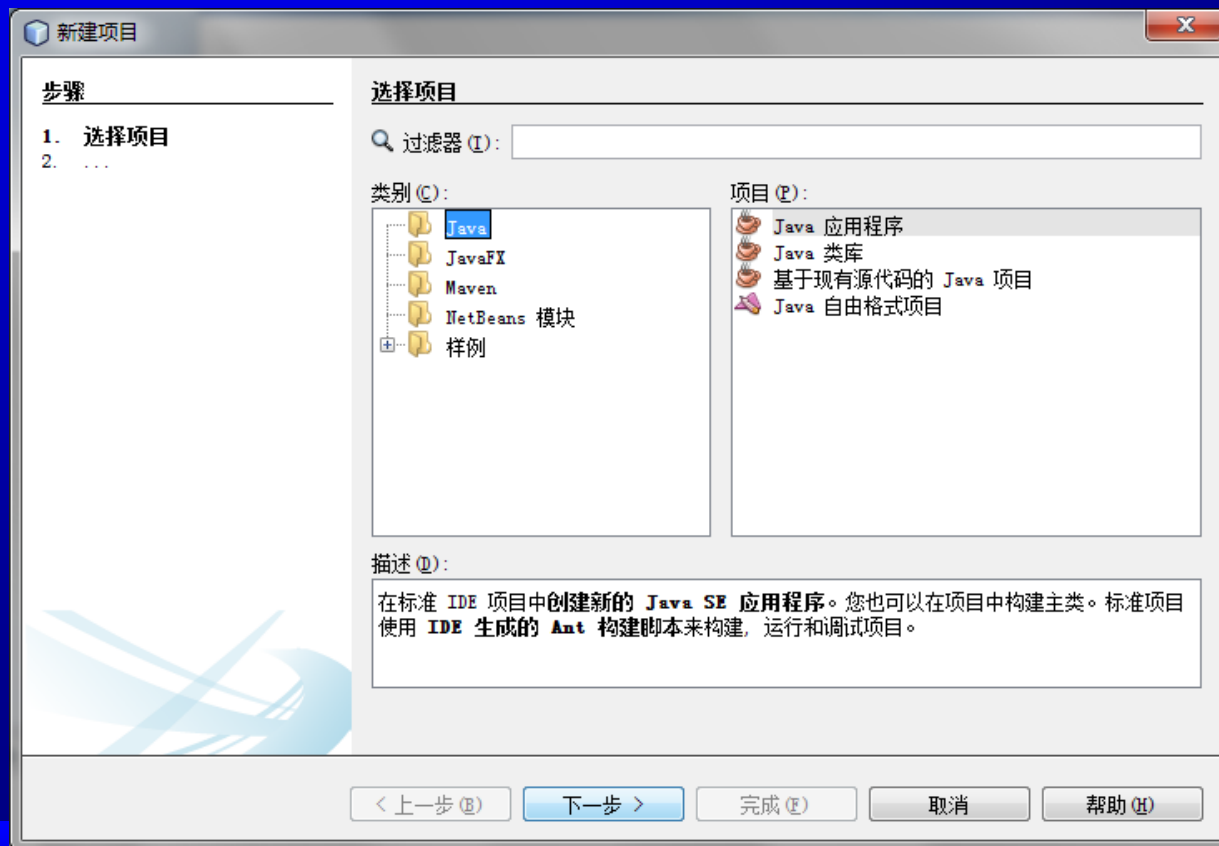
Next-line style

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

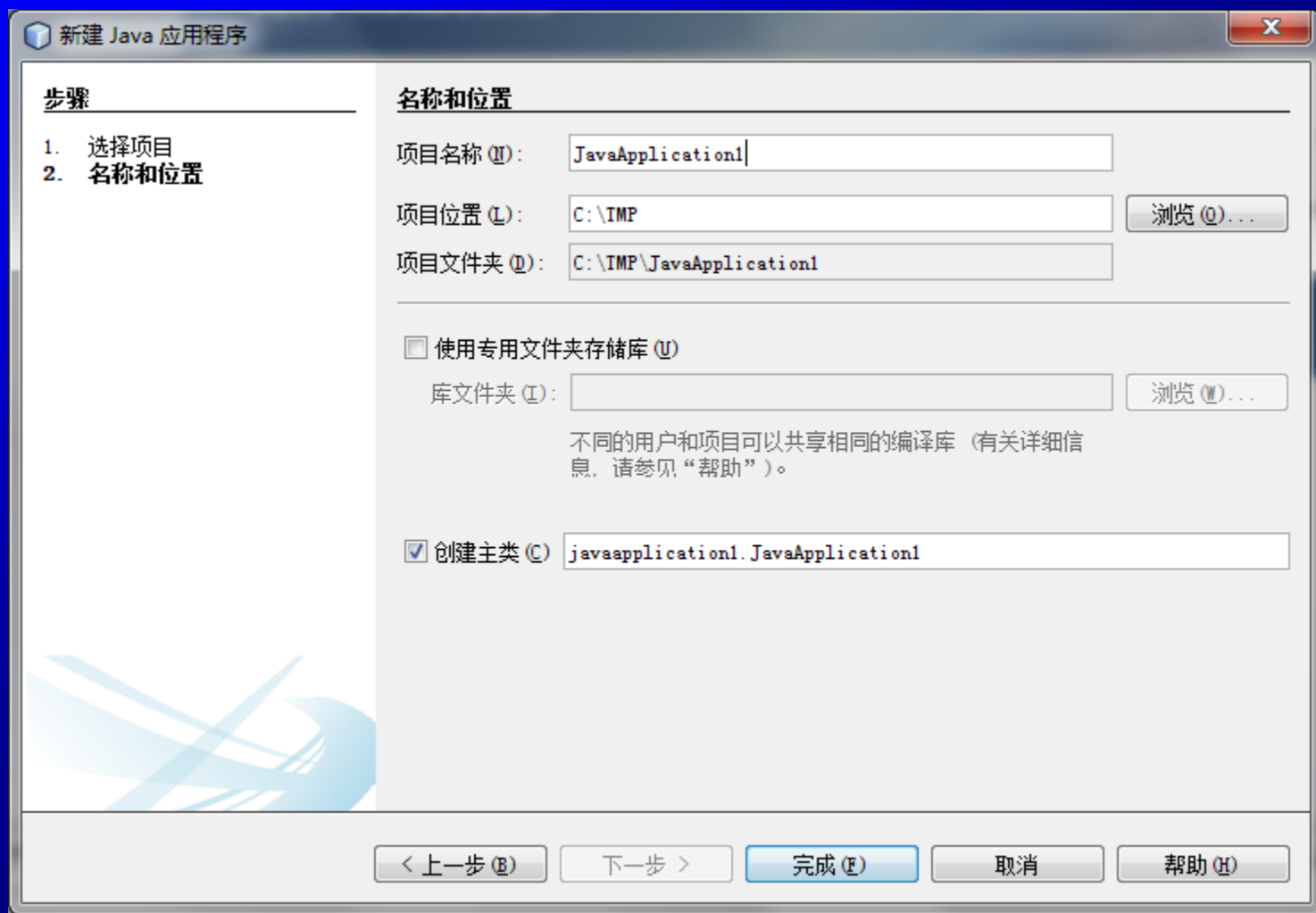
End-of-line style

# 使用NetBeans开发

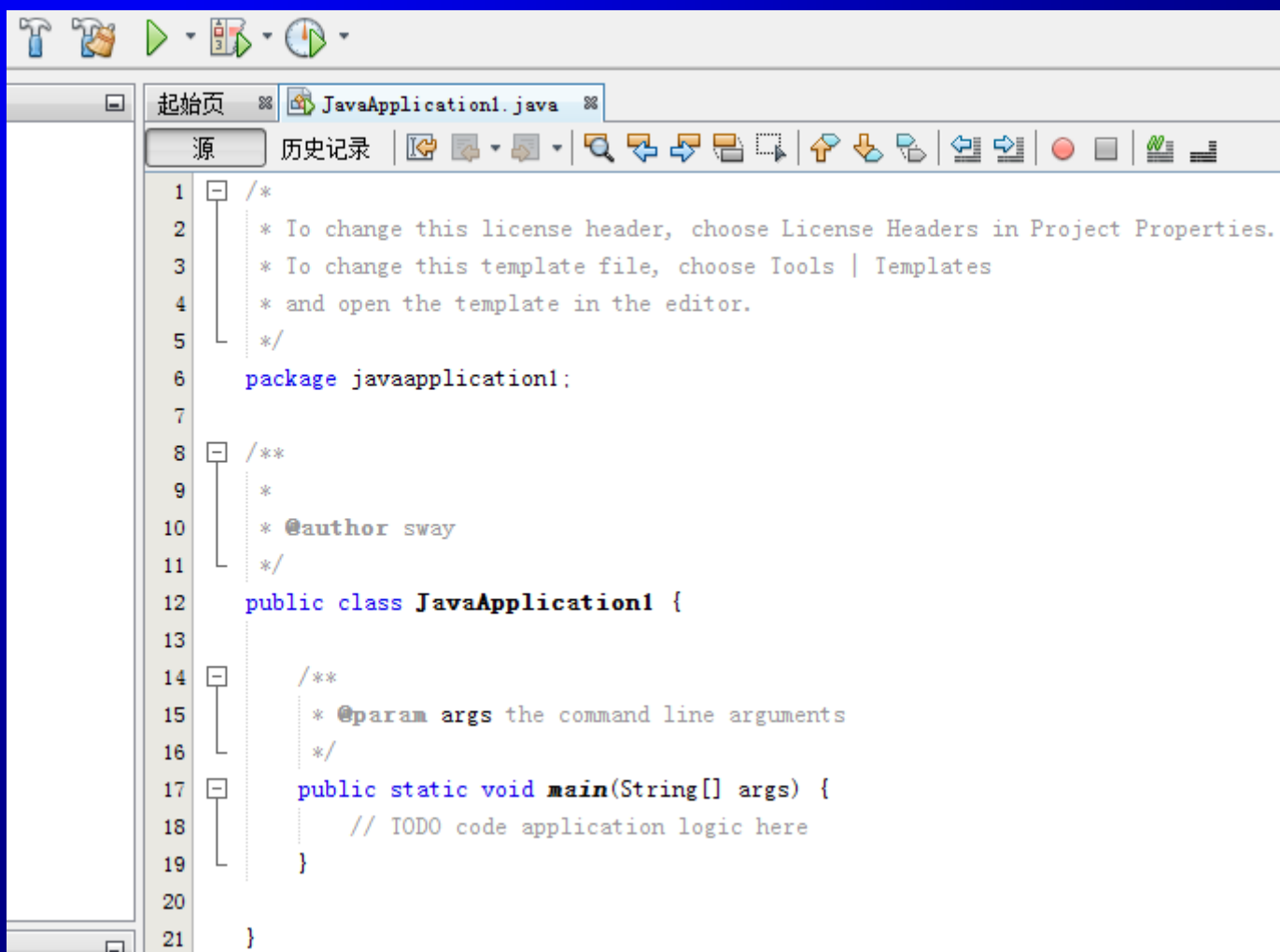
1. 文件 → 新建项目。类别的话，没界面的程序，选Java；有界面的程序，选JavaFX。



## 2. 指定项目名称、路径后，直接点完成按钮。



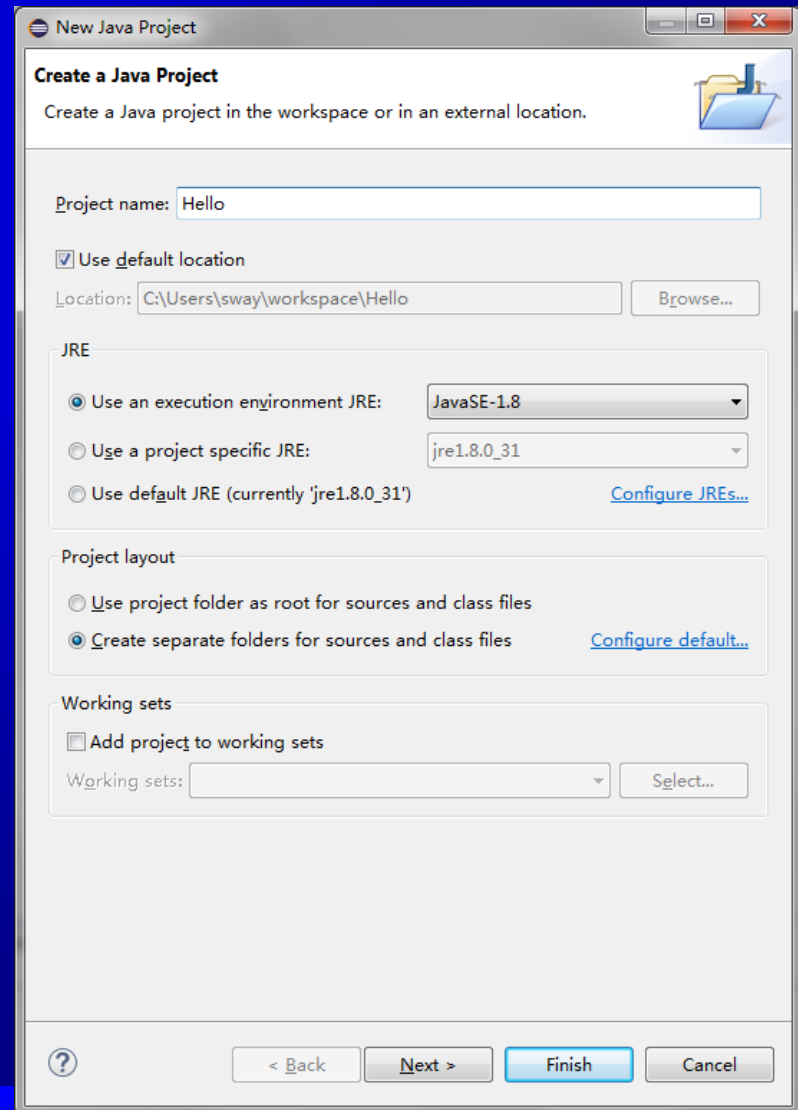
3. 下面是默认生成的框架，在此基础上添加代码即可。写完后，点击工具条上的“运行项目”按钮（左上第三个）或快捷键F6，启动编译运行。

A screenshot of an IDE window showing a Java application template. The window has a title bar with standard OS icons. Below the title bar is a tab bar with two tabs: '起始页' (Start Page) and 'JavaApplication1.java'. The 'JavaApplication1.java' tab is active. Below the tab bar is a toolbar with various icons for file operations, editing, and running. The main editor area displays the following Java code:

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6   package javaapplication1;
7
8   /**
9    *
10   * @author sway
11   */
12   public class JavaApplication1 {
13
14       /**
15        * @param args the command line arguments
16        */
17       public static void main(String[] args) {
18           // TODO code application logic here
19       }
20
21   }
```

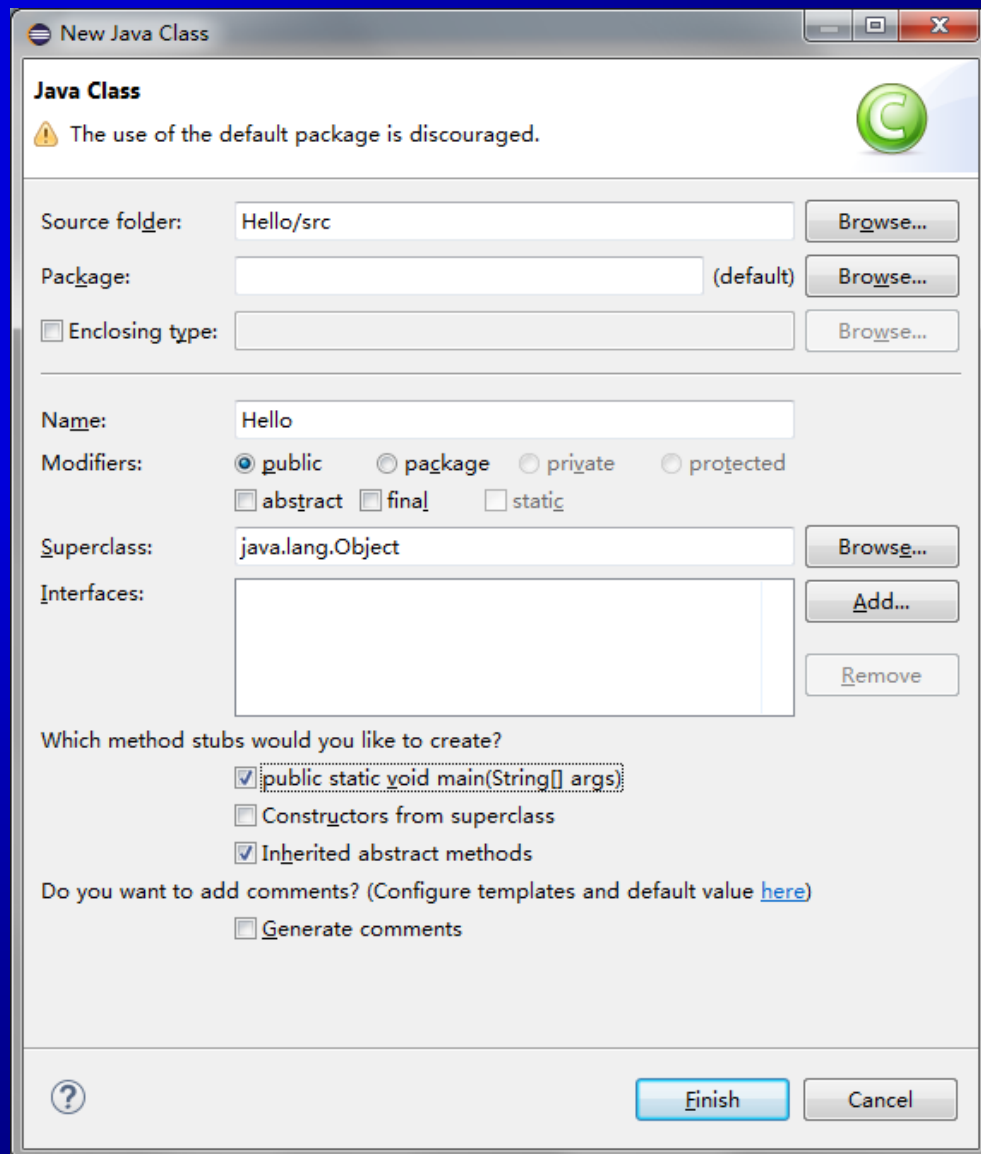
# 使用Eclipse开发

1. File → New  
→ Java Project,  
填写工程名字,  
指定工程所在路  
径, 然后直接点  
Finish, 生成一  
个空工程。



2. File → New → Class, 填写类型 Name, 勾上 public static void main 前面那个勾, 然后 Finish。

➤ 有兴致的话还可以顺便填写 Package 名。



3. 下面是默认生成的框架，在此基础上添加代码即可。写完后，点击工具条上的“Run”按钮（左上第二个），启动编译运行。

