《面向对象程序设计》电子教案

《Java 基础入门》教案

第 01 讲　　环境构建与简单程序开发　　2 学时 110 分钟

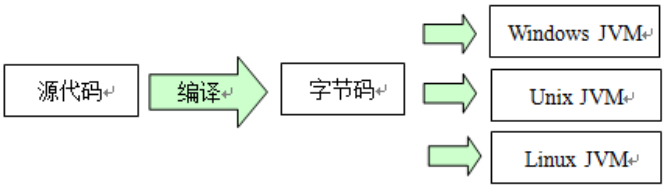
章节内容	**第 1 章　Java 起步入门** 1.1 Java 概述 1.2.Java 程序运行机制 1.3.开发环境搭建、集成开发环境的使用 1.4 .简单 Java 程序的开发和运行
教学目标	• 了解 Java 的起源与发展 • 了解 Java 程序的运行机制 • 掌握开发环境的搭建 • 重点掌握 Java 程序的编辑、编译和运行
重点 难点	**重点**：重点掌握 Java 程序的编辑、编译和运行。 **难点**：在字符界面下编译和运行程序。
教学环节 教学方法 及其它说 明事项	1.采用讲授法、演示法、讨论法。 2.案例+项目式教学。 3.采用线上线下混合教学模式。配套线上课程地址： www.qingline.net 4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。
导言 或复习	编程语言大致分为三个层次：机器语言、汇编语言和高级语言。 **机器语言**是 CPU 唯一能理解的编程语言。机器语言指令是用二进制编写的。这种方式书写和记忆程序都很困难，因此称为低级语言。 **汇编语言**比机器语言高一级的语言，它允许程序员使用符号操作码来编写程序，而不是将程序写成二进制序列。 **高级语言**的开发是为了使程序员能够比使用汇编语言更快地编写程序。 **C 语言**是在 20 世纪 70 年代初由 AT&T 贝尔实验室开发的。该语言可以说是最伟大的编程语言，直到今日，仍然具有巨大的生命力。 现代的编程语言大多支持**面向对象**。C++是在 20 世纪 80 年代早期作为 C 语言的继承者开发的，目的是增加对面向对象编程的支持。Java 是目前最流行、应用最广泛的面向对象编程语言。
课堂讲授	1. Java 语言及其有点 Java 是一种编程语言和计算平台，由 Sun Microsystems 的 James Gosling 和他的团队开发，于 1995 年发布。 Java 有三个版本：Java 标准版、Java 企业版、Java 微型版 **Java 语言的优点** 其中包括：Java 是面向对象的、解释型的、平台独立的、简单的、可移植的、健壮的、

分布式的、安全的、高性能的、多线程的和动态的。

1.2 Java 程序运行机制

一、Java 字节码与平台独立

在 Java 编程中，源程序要编译成**字节码**（bytecode）才能运行。字节码不是本地机代码，所以它不能直接运行。字节码只能在 **Java 虚拟机**（Java Virtual Machine，JVM）上运行。JVM 是一种解释字节码的本机应用程序。



二、JVM、JRE 和 JDK

JVM 是一个运行字节码的应用程序。

JRE（Java Runtime Enviroment）称为 Java 运行时环境，它包括 JVM 和 Java 类库。

JDK（Java Development Kit）是 Java 开发工具包，它包含进行 Java 开发所需的最少软件。

演示或观看教学视频

1.3 建立开发环境

JDK 安装完后，会在硬盘上创建一个目录，该目录被称为 JDK 安装目录。

重点强调“”

bin 目录：存放编译、执行和调试 Java 程序的工具。如 javac.exe 是 Java 编译器、java.exe 是 Java 解释器。

1.3.3 关于环境变量

输入 java -version，如果显示 Java 版本号信息，说明解释器正常。如图所示。

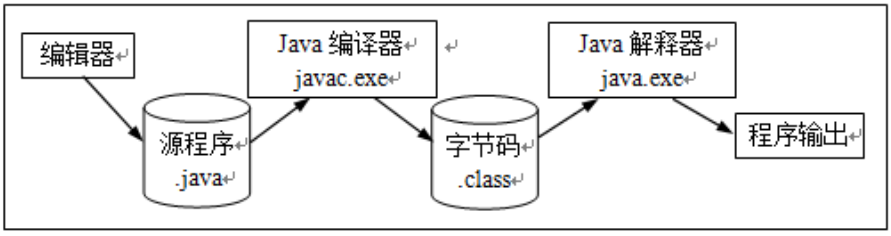
提示：

如果读者下载的 JDK 是压缩文件，那么解压后就需要自己设置 path 环境变量，将 Java 安装目录的 bin 目录添加到 path 环境变量中。

讲授内容

1.4 第一个 Java 程序

开发 Java 程序通常分三步：编写源程序；编译源程序；执行程序，得到程序输出结果。

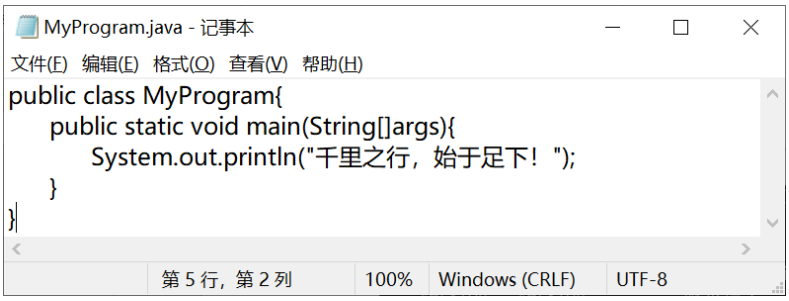


【案例 1-1】第一个 Java 程序，在控制台输出一个字符串。

【程序 1-1】MyProgram.java

```
public class MyProgram{
    public static void main(String[] args){
        System.out.println("千里之行，始于足下！");
    }
}
```

1.4.1 编写 Java 程序



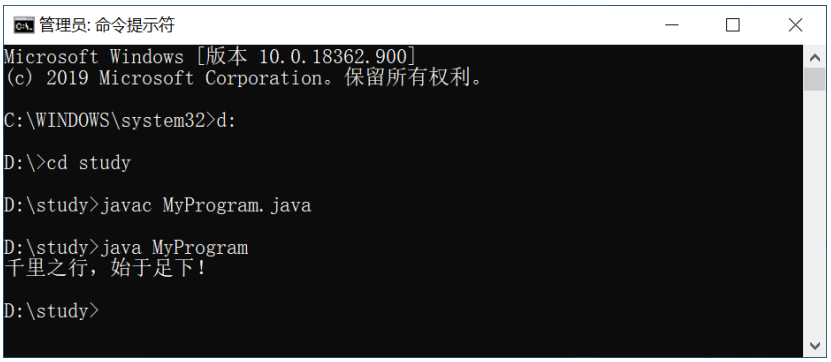
1.4.2 编译 Java 程序

打开一个命令提示符窗口，并将目录更改为保存 MyProgram.java 文件的目录。输入以下命令。

```
D:\study>javac MyProgram.java
```

1.4.3 执行 Java 程序

```
D:\study>java MyProgram
```



1.5 Eclipse IDE

1.5.1 Eclipse 的安装和启动

1.5.3 使用 Eclipse 开发程序

演示或看教学视频

1. 创建 Java 项目
2. 在项目中创建包
3. 在包中创建 Java 类
4. 编写程序代码
5. 运行程序

课堂讨论与互动

一、关于 Java 程序的 main()方法，回答下面问题。

（1）说明 main()方法声明中各部分的含义。

（2）public static void main(String[]args)写成 static public void main(String[]args)会怎

	样？
	（3）public static void main(String[]args)写成 public void main(String args)会怎样？
	二、程序错误大致可以分为三类：编译错误、运行时错误和逻辑错误。
	下面介绍在 Eclipse 中如何发现和处理编译错误。下面代码中有 5 处错误，现在你能都找出来吗？
	`public class MyProgram{`
	`public static void main(String args){`
	`system.out.print1("千里之行，始于足下！")`
	`}`
	`}`

总结 **课后作业**	**一、知识点总结** 1. 编写和运行 Java 程序需要使用 JDK，目前的最新版本是 JDK 17。 2. 开发 Java 程序的一般步骤是：使用编辑器编写源程序，保存为.java 文件；使用 Java 编译器 javac.exe 将源程序编译成.class 字节码文件；最后使用 Java 解释器 java.exe 执行程序。 3. 为加快程序开发速度，可以使用 IDE（集成开发环境），两个最流行的 IDE 是 IntelliJ IDEA 和 Eclipse，它们都是开源免费的。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2.编写程序，打印输出你最喜欢的一首古诗。 3.编写程序，计算并输出 1+3+5+7+9 的结果。

《Java 基础入门》教案

第 02 讲　　数据类型　　2 学时 110 分钟

章节内容	第 2 章 数据类型与运算符 2.1 Java 类型系统 2.2 变量与赋值 39 2.3 文档风格和注释 2.4 字面值 2.5 字符串与文本块 2.7 数据类型转换 2.6 案例研究：计算身体质量指数
教学目标	• 了解 Java 类型系统 • 了解文档风格和注释 • 掌握变量与赋值，各种类型字面值 • 了解字符串与文本块，数据类型转换
重点	**重点**：掌握各种基本数据类型的使用。
难点	**难点**：数据类型转换。
教学环节 教学方法 及其它说 明事项	1.采用讲授法、演示法、讨论法。 2.案例+项目式教学。 3.采用线上线下混合教学模式。配套线上课程地址： www.qingline.net 4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。
导言 或复习	在程序设计中，数据是程序的必要组成部分，也是程序处理的对象。不同的数据有不同的类型，不同的数据类型有不同的数据结构、不同的存储方式，并且参与的运算也不同。 Java 数据类型可分为**基本数据类型**（primitive data type）和**引用数据类型**（reference data type）两大类。下面简单介绍这两种数据类型。
讲授内容	2.1.1 基本数据类型 Java 共有 8 种基本数据类型。 前 6 种基本类型（byte、short、int、long、float 和 double）用于存储数字，它们能够存储数值的大小不同。 对于带有小数点的数字，需要 float 或 double。float 是用 32 位存储的浮点值。 char 类型可以保存一个 Unicode 字符，比如'A'、'9 '或'&'。Unicode 也允许使用不包含在英语字母表中的字符。boolean 类型值可以包含两种可能状态之一（false 或 true）。 2.1.2 引用数据类型 Java 还支持 6 种引用数据类型，其中包括类、接口、枚举、记录、注解和数组。 • 类是 Java 语言最重要的引用类型，任何 Java 程序都离不开类的使用。 • 接口是对类的一种扩展，它也是一种引用类型，但接口不能实例化。

- 枚举是一种特殊的引用类型，它用来定义具有确定几个值的类型，比如交通灯有绿色、红色、黄色三种颜色。
- 注解类型主要用于以结构化的方式为程序元素（类、方法等）提供信息。
- 记录在 Java 15 中是预览特性，它用来声明主要用于数据存储的类。
- 数组是一种特殊的引用类型，它不需要程序员自己定义类型。

2.2 关键字、变量与赋值

一、Java 关键字

每种语言都定义了自己的关键字。所谓**关键字**（keywords）是该语言事先定义的一组词汇，这些词汇具有特殊的用途，用户不能将它们定义为标识符。

二、Java 标识符

标识符必须以字母、下划线（_）或美元符（$）开头，其后可以是字母、下划线、美元符或数字，长度没有限制。如下面是一些合法的标识符：

```
intTest, Manager_Name, _var, $Var
```

三、变量

变量（variable）是在程序运行中其值可以改变的量。一个变量通常由三个要素组成，数据类型、变量名和变量值。

```
int  age;
double  d1, d2;
char  letter, ch2;
```

四、变量赋值

```
age = 21;
letter = 'A';
d1 = d2 = 0.618;
```

一次给多个变量赋值

也可以在声明变量的同时给变量赋值，例如：

```
boolean b = false;
```

声明变量同时赋值

五、语句

程序由一系列指令组成，这些指令称为**语句**（statement）。Java 有许多类型的语句，有简单的语句（如声明语句、赋值语句），也有控制流程语句（如 if、while、for、switch）等语句。在 Java 中，语句以分号结束。

下面是给变量赋值的语句。

```
x = z + 5;
```

下面是一个变量声明语句。

```
long secondsElapsed;
```

观看教学视频

2.3 文档风格和注释

代码块是由大括号围起来的一组语句，如类体、方法体、初始化块等。代码块的大括号有两种写法，一是**行末格式**，即左大括号写在上一行的末尾，右大括号写在下一行开头，如程序 1-1 所示。另一种格式称**为次行格式**，即将左大括号单独写在下一行，右大括号与左大括号垂直对齐。

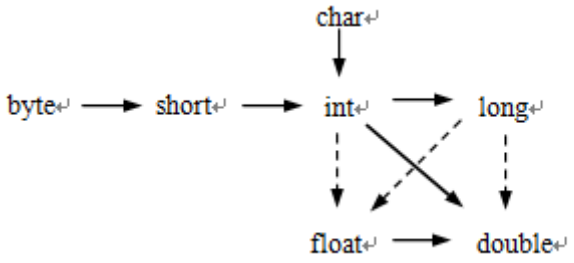
	保持一致的缩进会使程序更加清晰、易读、易于调试和维护。
	二元操作符的两边也应该各加一个空格，如下面语句所示：
	`System.out.println(3+4*5);  // 不好的风格`
	`System.out.println(3 + 4 * 5);  // 好的风格`
	2.3.3 程序注释
	Java 源程序支持三种类型的注释。
	（1）单行注释
	`// 这里是注释内容`
	（2）多行注释
	`/*`
	`该文件的文件名必须为：MyProgram.java`
	`*/`
	（3）文档注释，以/**开始，以*/结束的多行。

讲授内容	**2.4 字面值与基本类型**
	字面值（literals）是某种类型值的表示形式，比如，99 是 int 类型的字面值。
	字面值有三种类型：基本类型的字面值、字符串字面值以及 null 字面值。基本类型的字面值有 4 种类型：整数型、浮点型、布尔型、字符型。
	2.4.1 整数字面值
	Java 提供 4 种整数类型，分别是 byte 型（字节型）、short 型（短整型）、int 型（整型）和 long 型（长整型）。
	整型字面值有四种表示方法。
	（1）十进制数。（2）二进制数。（3）八进制数。（4）十六进制数。
	整型变量使用 byte、short、int、long 等声明。
	`byte a = 0b00101010;`
	`System.out.println("a = "  + a);  //  42`
	【案例 2-1】计算一光年的距离。在表示较大的整数时，可能需要用到长整型 long。
	【程序 2-1】LightYear.java
	`package com.boda.xy;`
	`public class LightYear{`
	`public static void main(String[] args){`
	`int speed = 300000;  // 光速为每秒 300000 公里`
	`long seconds = 365 * 24 * 60 * 60; // 假设一年为 365 天`
	`long distance = speed * seconds;`
	`System.out.println("一光年的距离是 "+distance+" 公里。" );`
	`}`
	`}`
	2.4.2 浮点字面值
	Java 中有两种浮点类型的数据：float 型和 double 型。
	浮点型字面值有两种表示方法。

	（1）十进制数形式。
	（2）科学记数法形式。
	`float pi = 3.1415926F;` ← float 型值必须加 F 或 f
	`System.out.println("float pi = " + pi);`
	如果一个数值字面值太长，读起来会比较困难。从 Java 7 开始，对数值型字面值的表示可以使用下划线（_）将一些数字进行分组，这可以增强代码的可读性。
	`210703_19901012_2415 // 表示一个身份证号`
	注意：浮点数计算可能存在舍入误差，因此，浮点数不适合做财务计算，而财务计算中的舍入误差是不能接受的。例如，下面语句的输出结果是 0.8999999999999999，而不是所期望的 0.9。
	`System.out.println(2.0 - 1.1);` ← 结果不是 0.9!
	### 2.4.3 字符字面值
	字符型字面值用英文单引号定界，大多数可见的字符都可用这种方式表示，如'a'、'@'、'我'等。
	常用的转义序列如表 2-4 所示。
	字符型变量使用 char 定义，在内存中占 16 位，表示的数据范围是 0~65 535。字符型变量的定义如：
	`char c = 'A';`
	### 2.4.4 布尔字面值
	布尔型变量使用 boolean 关键字声明，如下面语句声明了布尔型变量 t 并为其赋初值 true：
	`boolean t = true;`

演示或看教学视频	2.6 案例研究：计算身体质量指数 问题描述 身体质量指数（Body Mass Index，BMI）是衡量一个人体重是否超重的指标。计算公式为： BMI = 体重 / 身高的平方 编写程序，从键盘上输入一个人的体重（单位：公斤）和身高（单位：米），计算他的 BMI。 **完整代码见教材**，《Java 基础入门-项目案例+微课视频+题库》，沈泽刚，清华大学出版社，2021 年。
讲授内容	## 2.7 数据类型转换 在 Java 中，基本数据类型的转换分为自动类型转换和强制类型转换两种。 ### 2.7.1 自动类型转换 自动类型转换也称加宽转换，它是指将具有较少位数的数据类型转换为具有较多位数的数据类型。例如： `byte b = 120;`

```
int   i = b;     // 字节型数据 b 自动转换为整型
```



2.7.2 强制类型转换

将位数较多的数据类型转换为位数较少的数据类型。

其语法是在圆括号中给出要转换的目标类型，随后是待转换的表达式。例如：

```
double d = 200.5;
byte b = (byte)d;        ← 将 double 型值强制转换成 byte 型值
System.out.println(b);   // 输出-56
```

2.7.3 表达式类型自动提升

除了赋值可能发生类型转换外，在含有变量的表达式中也有类型转换的问题，如下所示：

```
byte  a = 40;
byte  b = 50;
byte  c = a + b;        ← 这里表达式 a + b 结果类型提升为 int 型
c = (byte)(a + b);   // 正确
int i = a + b;
```

课堂互动与讨论

【案例 2-3】浮点数四舍五入。该案例通过使用类型转换实现四舍五入功能。比如，圆周率在 Java 中用 Math.PI 常量表示，默认情况下输出它的值是 3.141592653589793。假设希望将该值四舍五入保留 4 位小数，可以通过下面程序实现。

【程序 2-4】CastDemo.java

```
package com.boda.xy;
public class CastDemo{
    public static void main(String[] args){
        System.out.println(Math.PI);   // 输出 PI
        double pi = Math.PI;
        pi = pi * 10000 + 0.5;       // 保证四舍五入
        pi = (int) pi;               // 保留整数部分
        pi = pi / 10000;             // 得到结果
        System.out.println(pi);
    }
}
```

作业预习

一、知识点总结

1. Java 数据类型可分为基本数据类型（byte、short、int、long、float、double、char、

	boolean）和引用数据类型（数组、类、接口、记录、枚举、注解）。 2. 使用 java.util.Scanner 类可以从键盘读取除 char 外的各种类型数据（包括字符串）。 **二、课后作业** 1.**登录本书在线作业网站，完成本章作业题目**。地址：https://www.qingline.net/ 2.编写程序，从键盘上输入一个 double 型的华氏温度，然后将其转换为摄氏温度输出。转换公式如下：摄氏度 =(5/9)×（华氏度-32） 3.编写程序，从键盘输入圆柱底面半径和高，计算并输出圆柱的体积。

《Java 基础入门》教案

第 03 讲　　运算符　　2 学时 110 分钟

章节内容	**第 2 章 数据类型与运算符** 2.8　运算符 2.8.1　算术运算符 2.8.2　比较运算符 2.8.3　逻辑运算符 2.8.4　赋值运算符 2.8.5　位运算符 2.8.6　运算符的优先级 2.9　案例研究：显示当前时间
教学目标	• 掌握 Java 常用运算符 • 了解云素服优先级和位运算符
重点	**重点**：掌握算符、比较、逻辑和赋值运算符。
难点	**难点**：逻辑运算符和位运算符的使用。
教学环节 教学方法 及其它说 明事项	1.采用讲授法、演示法、讨论法。 2.案例+项目式教学。 3.采用线上线下混合教学模式。配套线上课程地址： www.qingline.net 4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。

导言
或复习
	运算符和表达式是 Java 程序的基本组成要素。把表示各种不同运算的符号称为**运算符**（operator），参与运算的各种数据称为**操作数**（operand）。为了完成各种运算，Java 提供了多种运算符，不同的运算符用来完成不同的运算。
	表达式是由运算符和操作数按一定语法规则组成的符号序列。以下是合法的表达式：
	`(7 + 3) * (8 - 5)`
	`a > b && c < d`
	一个字面值或一个变量是最简单的表达式。每个表达式经过运算后都会产生一个确定的值。

讲授内容	2.8.1　算术运算符 算术运算符一般用于对整型数和浮点型数运算。算术运算符有加（+）、减（-）、乘（*）、除（/）和取余数（%）5 个二元运算符和正（+）、负（-）、自增（++）、自减（--）4 个一元运算符。 **注意**，除法运算符（/）时，如果两个操作数都是整数，商为整数，例如：5 / 2 的结果是 2 而不是 2.5，而 5.0 / 2 的结果是 2.5。 “%”运算符用来求两个操作数相除的余数，操作数可以为整数，也可以为浮点

数。例如，7 % 4 的结果为 3，10.5 % 2.5 的结果为 0.5。当操作数含有负数时，情况有点复杂。这时的规则是余数的符号与被除数相同且余数的绝对值小于除数的绝对值。例如：

```
10 % 4 = 2
10 % -4 = 2
-10 % 4 = -2
-10 % -4 = -2
```

在操作数涉及负数求余运算中，可通过下面规则计算：先去掉负号，再计算结果，结果的符号取被除数的符号。如求-10 % -4 的结果，去掉负号求 10 %4，余数为 2。由于被除数是负值，因此最终结果为-2。

“+”运算符不但用于计算两个数值型数据的和，还可用于字符串对象的连接。例如，下面的语句输出字符串"abcde"。

```
System.out.println("abc" + "de");
```

当+运算符的两个操作数一个是字符串而另一个是其他数据类型，系统会自动将另一个操作数转换成字符串，然后再进行连接。例如下面代码输出“sum = 123”。

```
int a = 1, b = 2, c = 3;
System.out.println("sum = " + a + b + c);
```

但要注意，下面代码输出“sum = 6”。

```
System.out.println("sum = " + (a + b + c));
```

先计算括号内表达式

自增（++）和自减（--）运算符

“++”和“--”运算符主要用于对变量的操作，分别称为自增和自减运算符，“++”表示加 1，“--”表示减 1。它们又都可以使用在变量的前面或后面，如果放在变量前，表示给变量加 1 后再使用该变量；若放在变量的后面，表示使用完该变量后再加 1。

变量 x 的值为 5，执行下面语句后 y 和 x 的值如下所示：

```
y = x++;       //语句执行后 y = 5   x = 6
y = ++x;       //语句执行后 y = 6   x = 6
y = x--;       //语句执行后 y = 5   x = 4
y = --x;       //语句执行后 y = 4   x = 4
```

2.8.2 比较运算符

比较运算符（也称关系运算符）用来比较两个值的大小或是否相等。一般用来构成条件表达式，比较的结果返回 true 或 false。假设定义了下面的变量。

```
int x = 99;
int y = 108;
```

下面的语句的输出是 true。

```
System.out.println(x < y);
```

比较结果通常作为判断条件，如下所示。

```
if (n % 2 == 0)
   System.out.println( n + "是偶数");
```

2.8.3 逻辑运算符

逻辑运算符包括以下几种：逻辑非（!）、短路与（&&）、短路或（||）、逻辑与（&）、逻辑或（|）、逻辑异或（^）。假设 A、B 是两个布尔型数据，则逻辑运算的规则如表 2-6 所示。

【案例 2-5】逻辑运算符使用。该案例说明在相同的初始条件下，使用短路逻辑运算符和非短路逻辑运算符，对同一表达式的计算结果可能不同。

【程序 2-6】LogicalDemo.java

```
package com.boda.xy;
public class LogicalDemo{
   public static void main(String[] args){
      int a = 1, b = 2, c =3 ;
        boolean u = false;
      u = (a >= --b || b++ < c--) && b == c;
        System.out.println("u = " + u);
        // 使用&和|运算符
        b = 2;
      u = ( a >= --b | b++ < c--) & b == c;
        System.out.println("u = "+u);
   }
}
```

这个表达式未被计算

这个表达式被计算

2.8.4 赋值运算符

赋值运算符用来为变量指定新值。

```
int x = 10;
int y = x + 20;
```

赋值运算必须是类型兼容的，即左边的变量必须能够接受右边的表达式的值，否则会产生编译错误。如下面的语句产生编译错误。

```
int pi = 3.14;
```

编译错误

复合赋值运算符

用变量当前值与右侧表达式值进行运算，最后将运算结果赋给变量。例如，下面两行是等价的：

```
a += 3;
a = a + 3;
```

观看视频自学

2.8.5 位运算符

位运算有两类：**位逻辑**运算（bitwise）和**移位**运算（shift）。位逻辑运算符包括按位取反（~）、按位与（&）、按位或（|）和按位异或（^）4 种。移位运算符包括左移（<<）、右移（>>）和无符号右移（>>>）3 种。

课堂讲授

2.8.6 运算符的优先级

假设有下面一个表达式：

```
3 + 4 * 5 > (5 * (2 + 4) - 10)&& 8 - 4 > 5
```

首先计算括号中的表达式（如果有嵌套括号，先计算里层括号中的表达式）。当计算没有括号的表达式时，会按照运算符的优先级和结合性进行运算。

不必死记硬背运算符的优先级。必要时可以在表达式中使用圆括号，圆括号的优先级最高。圆括号还可以使表达式显得更加清晰。例如，考虑以下代码：

```
int x = 5;
int y = 5;
boolean z = x * 5 == y + 20;
```

因为“*”和“+”的优先级比“==”高，比较运算之后，z 的值是 true。但是，这个表达式的可读性较差。使用圆括号把最后一行修改如下：

```
boolean z = (x * 5) == (y + 20);
```

讲授内容

2.9 案例学习：显示当前时间

1. 问题描述

编写程序，以 GMT（格林尼治时间）来显示当前的时间，以“小时:分钟:秒”的格式来显示。要求得到类似下面的输出结果。

当前时间：11:10:27

2. 运行结果

下面是程序一次运行结果，如图 2-11 所示。

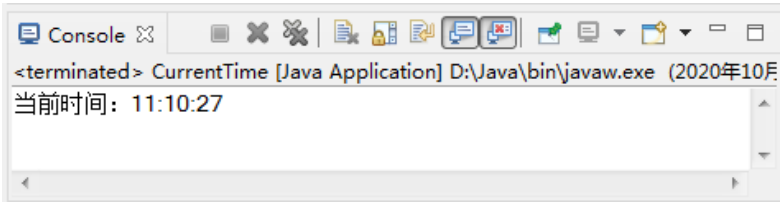


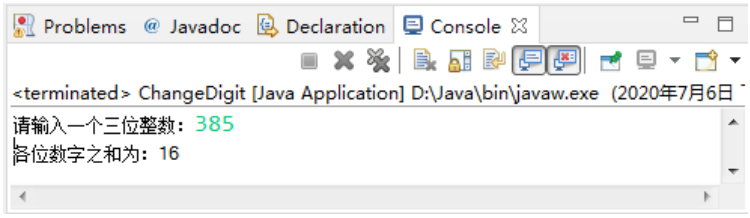
图 2-11 案例运行结果

【程序 2-7】CurrentTime.java

```
package com.boda.xy;
public class CurrentTime {
    public static void main(String[] args) {
        long t = System.currentTimeMillis();
        long seconds = t / 1000;      // 总秒数
        long s = seconds % 60;
        long minutes = seconds / 60;   // 总分钟数
        long m = minutes % 60;
```

<table>
<tr><td></td><td>

```
            long hours = minutes / 60;     // 总小时数
            long h = hours % 24;
            System.out.println("当前时间："+h +":" + m + ":" + s);
        }
    }
```

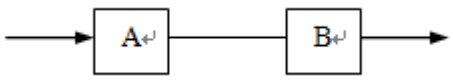
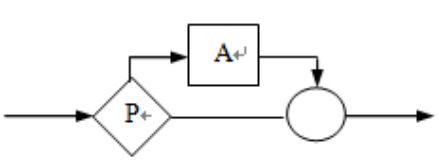
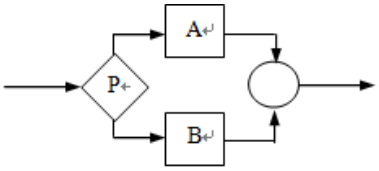
</td></tr>
<tr><td>课堂互动与讨论</td><td>编写程序，接受用户从键盘输入一个三位整数（如 385），计算并输出各位数字之和。运行结果如图 2-13 所示。
 
</td></tr>
<tr><td>总结

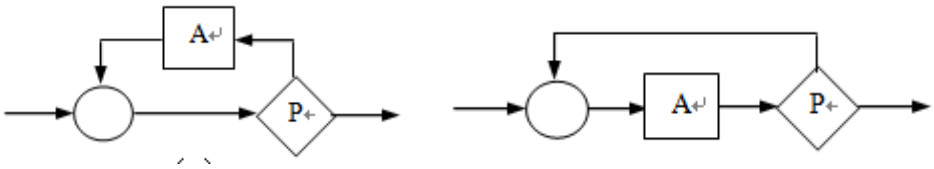
课后作业</td><td>一、知识点总结
1. 算术运算符包括：加（+）、减（-）、乘（*）、除（/）、求余（%）、自增（++）、自减（--），关系运算符包括：大于（>）、小于（<）、大于或等于（>=）、小于或等于（<=）、等于（==）、不等于（!=），
2. 位运算符包括：按位取反（~）、按位与（&）、按位或（|）、按位异或（^）、左移（<<）、右移（>>）、无符号右移（>>>），位运算符只能应用在整型数据上，逻辑运算符包括：逻辑非（!）、短路与（&&）、短路或（||）、逻辑与（&）、逻辑或（|）、逻辑异或（^），赋值运算符包括（=）和扩展的赋值运算符（如+=、<<=等）。

二、课后作业
1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/
2. 编写程序，从键盘输入一个两位数，按数位逆序输出。提示：使用“%”和“/”运算符可求出每一位数字。
3.编写程序，要求用户从键盘输入 a、b 和 c 的值，计算下列表达式的值。
$$\frac{-b+\sqrt{b^2-4ac}}{2a}$$</td></tr>
</table>

《Java 基础入门》教案

第 04 讲　　选择结构　　2 学时 110 分钟

章节内容	**第 3 章 结构化编程** 3.1　编程方法 3.2　选择结构 3.2.1　if 语句与 if~else 语句 3.2.3　多分支 if-else 语句 3.4　switch 语句与 switch 表达式 3.4.1　switch 语句 3.4.2　switch 表达式
教学目标	• 了解编程方法 • 掌握各种选择结构 • 重点掌握 if-else 结构和 switch 结构
重点 难点	**重点**：掌握各种选择结构的使用，包括 if-else 结构和 switch 结构。 **难点**：新的 switch 结构和 switch 表达式的使用。
教学环节 教学方法 及其它说 明事项	1.采用讲授法、演示法、讨论法。 2.案例+项目式教学。 3.采用线上线下混合教学模式。配套线上课程地址： www.qingline.net 4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。
导言 或复习	有多种编程方法，比如结构化编程方法、面向对象编程方法、函数式编程、反应式编程。 **结构化编程**（structured programming）方法在 1965 年提出的，是软件发展的一个重要的里程碑。在结构化编程中，只允许三种基本的程序结构，它们是顺序结构、分支结构（包括多分支结构）和循环结构，这三种基本结构的共同特点是只允许有一个入口和一个出口，仅由这三种基本结构组成的程序称为**结构化程序**。 顺序结构。顺序结构表示程序中的各操作是按照它们出现的先后顺序执行的。  选择结构。程序的处理步骤出现了分支，它需要根据某一特定的条件选择其中的一个分支执行。   循环结构。程序反复执行某个或某些操作，直到某条件为假（或为真）时才可终止循环。



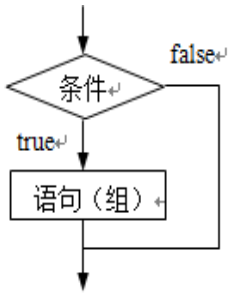
讲授内容

顺序结构比较简单，程序按语句的顺序依次执行。前面章节编写的程序都是顺序结构的，本章重点讨论选择结构和循环结构。

3.2.1 if 语句

if 语句的格式如下：

```
if (条件){
    语句（组）;
}
```



【案例 3-1】if 语句的使用。编写程序，要求用户从键盘输入两个整数，分别存入变量 a 与 b，如果 a 大于 b，则交换 a 和 b 的值，也就是保证 a 小于或等于 b，最后输出 a 和 b 的值。

【程序 3-1】ExchangeDemo.java

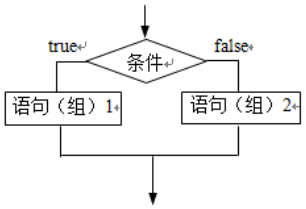
```
if(a > b){
    int t = b;
    b = a;
    a = t;
}
```

交换 a 与 b 的值

3.2.2 if~else 语句

if-else 结构的一般格式如下：

```
if (条件){
    语句（组）1;
}else{
    语句（组）2;
}
```



【案例 3-2】if-else 语句的使用。该案例要求用户从键盘输入一个年份，输出该年是否是闰年。符合下面两个条件之一的年份即为闰年：（1）能被 400 整除；（2）能被 4 整除，但不能被 100 整除。

【程序 3-2】LeapYear.java

```
if(year % 400 == 0||(year % 4 == 0 && year % 100 != 0)){
    System.out.println(year + " 年是闰年。") ;
}else{
    System.out.println(year + " 年不是闰年。") ;
```

```
        }
```

3.2.3 多分支 if-else 语句

【案例 3-3】多分支 if-else 语句的使用。要求输入一个人的身高和体重，计算并打印出他的 BMI，同时显示 BMI 是高还是低。对于一个成年人，BMI 值的含义如下：

- BMI<18.5，表示偏瘦；
- 18.5≤BMI<25.0，表示正常；
- 25.0≤BMI<30.0，表示超重；
- BMI≥30.0，表示过胖。

【程序 3-3】ComputeBMI.java

```
if(bmi < 18.5){
    System.out.println("你的体重偏瘦。");
}else if(bmi < 25.0) {
    System.out.println("你的体重正常。");
}else if(bmi < 30.0) {
    System.out.println("你的体重超重。");
}else {
    System.out.println("你的体重过胖。");
}
```

3.2.4 条件运算符

条件运算符（conditional operator）的格式如下：

```
条件 ? 表达式1 : 表达式2
```

```
max = (a > b)? a : b ;   等价于→   if (a > b) {
                                       max = a;
                                   }else {
                                       max = b;
                                   }
```

课堂讨论

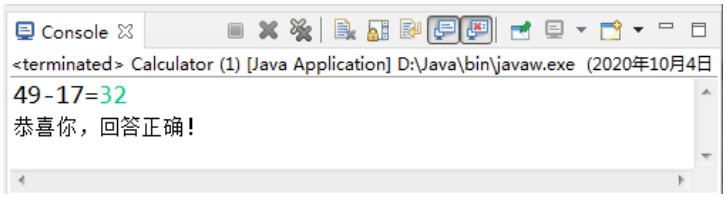
3.3 案例学习：两位数加减运算

1. 问题描述

开发一个让小学生练习两位整数加减法的程序，要求程序运行随机生成两个两位数及加减号（要保证减法算式的被减数大于减数），显示题目让学生输入计算结果，程序判断结果是否正确。

2. 运行结果

案例运行结果如图 3-9 所示，这里产生一个减法题目。



该案例的设计思路主要如下：

（1）要实现加减法运算，首先应该随机产生两个两位整数。随机生成整数有多种方法，可以使用 Math.random()方法生成一个随机浮点数，然后将它扩大再取整。random()方法返回 0.0~1.0（不包括）之间的浮点数，要得到 10~99 之间的整数，可以使用下面表达式：

```
int number1 = 10 + (int)(Math.random()*90);
```

（2）确定加或减运算。这也可以通过产生 2 个随机数（比如，0 和 1，0 表示加法，1 表示减法）确定。

```
int operator =  (int)(Math.random()*2);
```

（3）设学生没有学过负数概念，如果做减法运算，要保证第一个数大于第二个数。也就是如果 number1 小于 number2，应该交换这两个数。

```
if(number1 < number2){
    int temp = number2;
    number2 = number1;
    number1 = temp;
}
```

（4）最后根据运算符决定做何种运算。将计算结果保存到 result 变量中，然后与用户输入的答案 answer 比较，判断用户答题是否正确。

课堂讲授

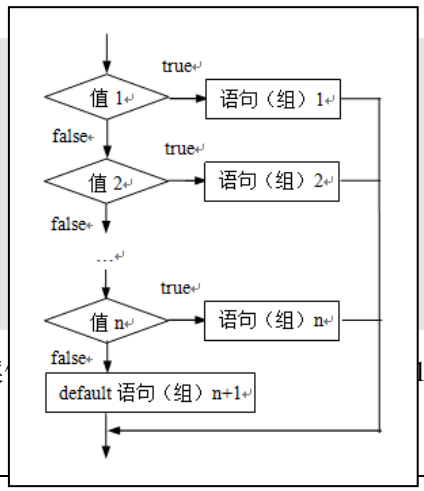
3.4 switch 语句与 switch 表达式

从 Java 12 开始对 switch 语句进行了修改并支持 switch 表达式。尽管 Java 仍然支持旧的 switch 结构，但建议读者熟悉并使用新的 switch 语句和 switch 表达式。

3.4.1 switch 语句

如果需要从多个选项选择其中一个，可以使用 switch 语句。switch 语句主要实现多分支结构，一般格式如下：

```
switch (表达式){
   case 值 1 ->  语句（组）1;
   case 值 2 ->  语句（组）2;
   …
   case 值 n ->  语句（组）n;
   [default ->  语句（组）n+1;]
}
```



【案例 3-4】用 switch 结构实现多重选择。该案例 1, 2, 3, 4），程序根据输入的数输出一句话。

【程序 3-5】SwitchDemo.java

```
        int season = input.nextInt();
        switch (season) {
            case 1 -> System.out.println("春雨惊春清谷天");
            case 2 -> System.out.println("夏满忙夏暑相连");
            case 3 -> System.out.println("秋处露秋寒霜降");
            case 4 -> System.out.println("冬雪雪冬小大寒");
            default ->
                System.out.println("季节输入非法.");
          }
```

从 Java 7 开始，可以在 switch 语句的表达式中使用 String 对象。

3.4.2 switch 表达式

可以使用 switch 表达式，即通过 switch 结构返回一个值，并将该值赋给变量。例如，下面代码根据 day 的值返回一个数值赋给变量 numLetters。

```
DayOfWeek day = DayOfWeek.SATURDAY;
int numLetters = switch(day){
   case MONDAY, FRIDAY,SUNDAY -> 6;
   case TUESDAY               -> 7;
   case THURSDAY, SATURDAY    -> 8;
   case WEDNESDAY             -> 9;
  };
System.out.println(numLetters);       // 输出 8
```

根据表示星期的枚举常量返回单词的字母数

分号是赋值语句的结束

【案例 3-6】switch 表达式应用。下面程序从键盘输入一个年份（如 2000 年）和一个月份（如 2 月），用 switch 表达式返回该月的天数（29），将其存入一个变量。

【程序 3-7】SwitchExprDemo.java

```
        int year = input.nextInt();
        int month = input.nextInt();
        int numDays = switch (month) {
           case 1, 3, 5, 7, 8, 10, 12 -> 31;
           case 4, 6, 9, 11 -> 30;
           // 对 2 月需要判断是否是闰年
           case 2 -> {
               if (((year % 4 == 0) && !(year % 100 == 0)) || (year % 400
== 0))
                   yield 29;
               else
                   yield 28;
           }
           default -> 0;
        };
        System.out.println("该月的天数为: " + numDays);
```

switch 表达式

yield 是受限标识符，生成一个值

分号是赋值语句的结束

课堂互动与讨论	从一副纸牌中任意抽取一张，并打印出抽取的是哪一张牌。一副牌有 4 种花色，黑桃、红桃、方块和梅花。每种花色有 13 张牌，共有 52 张牌。可以将这 52 张牌编号，从 0 到 51。规定编号 0 到 12 为黑桃，13 到 25 为红桃，26 到 38 为方块，39 到 51 为

	梅花。
	可以使用整数的除法运算来确定是哪一种花色，用求余数运算确定是哪一张牌。例如，假设抽出的数是 n，计算 n/13 的结果，若商为 0，则牌的花色为黑桃，若商为 1，则牌的花色为红桃，若商为 2，则牌的花色为方块，若商为 3，则牌的花色为梅花。计算 n%13 的结果可得到第几张牌。
	`int card =(int) (Math.random()*52);`
	`String suit="",rank="";`
	`suit = switch(card / 13){ // 确定牌的花色`
	`case 0 -> "♠";`
	`case 1 -> "♥";`
	`case 2 -> "♦";`
	`case 3 -> "♣";`
	`default -> "";` ← 在 switch 表达式中必须包含 default 语句
	`};`
	`rank = switch(card % 13){ // 确定是第几张牌`
	`case 0 -> "A";`
	`case 10 -> "J";`
	`case 11 -> "Q";`
	`case 12 -> "K";`
	`default -> ""+(card %13 +1);`
	`};`
	`System.out.println("你抽取的牌是：" + suit + rank);`

总结 **课后作业**	**一、知识点总结** 1. Java 的选择结构有以下几种类型：单分支 if 语句、双分支 if-else 语句、嵌套 if 语句、多分支 if-else 语句、switch 语句等。 2. switch 语句根据 char、byte、short、int、String 或者 enum 类型的 switch 表达式来进行控制决定。 3.从 Java 12 开始还可以使用 switch 表达式，它可将 switch 的 case 子句的结果赋给变量，或者作为方法的返回值。 4.条件运算符（?:）是 Java 唯一的三元运算符，它可以简化 if-else 语句的编写。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2.编写程序，要求用户从键盘上输入一个正整数，程序判断该数是奇数还是偶数。 3.从键盘输入一个百分制的成绩，输出五级制的成绩，如输入 85，输出“良好”，要求使用 switch 结构实现。

《Java 基础入门》教案

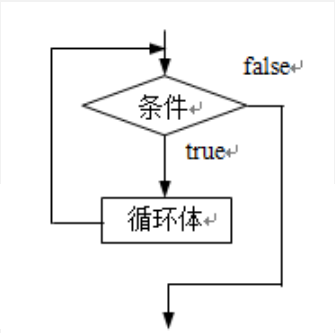
第 05 讲　　循环结构　　2 学时 110 分钟

章节内容	**第 3 章 结构环编程** 3.5　循环结构 3.5.1　while 循环 3.5.2　do-while 循环 3.5.3　for 循环 3.5.4　循环的嵌套 3.5.5　break 语句和 continue 语句 3.6　案例研究：求最大公约数 3.7　案例研究：打印输出若干素数
教学目标	• 掌握 Java 各种循环结构 • 掌握循环的嵌套 • 掌握 break 和 continue 语句 • 了解带标签的 break 和 continue 语句
重点	**重点**：掌握 Java 各种循环结构，包括 while 循环、do-while 循环和 for 循环。
难点	**难点**：循环的嵌套。
教学环节 教学方法 及其它说 明事项	1.采用讲授法、演示法、讨论法。 2.案例+项目式教学。 3.采用线上线下混合教学模式。配套线上课程地址： www.qingline.net 4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。
导言 或复习	上一次课学习了选择结构： 包括：if 语句、if-else 语句、嵌套的 if-else 语句 switch 结构和 switch 表达式 在程序设计中，有时需要反复执行一段相同的代码，这时就需要使用循环结构来实现。Java 语言提供了 4 种循环结构：while 循环、do-while 循环、for 循环和增强的 for 循环。 一般情况下，一个循环结构包含四部分内容： （1）初始化部分：设置循环开始时变量初值。 （2）循环条件：一般是一个布尔表达式，当表达式值为 true 时执行循环体，为 false 时退出循环。 （3）迭代部分：改变变量的状态。 （4）循环体：需要重复执行的代码。
讲授内容	**3.5.1　while 循环** while 循环是 Java 最基本的循环结构，这种循环是在某个条件为 true 时，重复执

行一个语句或语句块。它的一般格式如下：

```
[初始化部分]
while(条件){
    // 循环体
    [迭代部分]
}
```

大括号内为循环体



while 循环的执行流程如图 3-15 所示。

【案例 3-8】使用 while 循环计算 1 到 100 之和。

【程序 3-9】WhileDemo.java

```
int n = 1;
int sum = 0;
while(n <= 100){
    sum = sum + n;
    n = n + 1;
}
System.out.println("sum = " + sum);   // 输出 sum = 5050
```

初始化部分

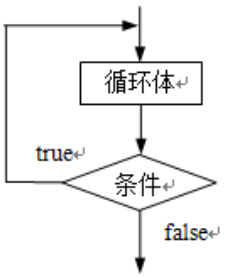
迭代语句

3.5.2 do-while 循环

do-while 循环的一般格式如下：

```
[初始化部分]
do{
   // 循环体
   [迭代部分]
}while(条件);
```

大括号内为循环体

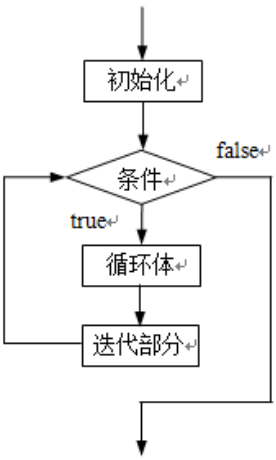


do-while 循环执行过程如图 3-18 所示。

【案例 3-10】用 do-while 循环计算 1 到 100 之和。

【程序 3-11】Sum100.java

```
int n = 1;
int sum = 0;
do{
    sum = sum + n;
    n = n + 1;
}while(n <= 100);
System.out.println("sum = " + sum);
// 输出 sum = 5050
```



3.5.3 for 循环

for 循环是 Java 程序中使用最广泛的，也是功能最强的循环结构。它的一般格式如下：

```
for (初始化部分; 循环条件; 迭代部分){
    // 循环体
}
```

这里，初始化部分、循环条件和迭代部分用分号隔开，大括号内为循环体。for 循环的

执行流程如图 3-20 所示。

下面代码使用 for 循环计算 1 到 100 之和。

```
int sum = 0;
for(int n = 1; n <= 100; n++){
    sum = sum + n;
}
System.out.println("sum = " + sum);   // 输出 sum = 5050
```

3.5.4 循环的嵌套

在一个循环的循环体中可以嵌套另一个完整的循环，称为循环的嵌套。内嵌的循环还可以嵌套循环，这就是多层循环。同样，在循环体中也可以嵌套另一个选择结构，选择结构中也可以嵌套循环。

【案例 3-11】用嵌套的 for 循环打印输出如图 3-21 所示图形。这里，第一行输出一个星号，第二行输出 3 个星号，…，第 8 行输出 15 个星号。

【程序 3-12】PrintStars.java

```
// n 记录行数
      for (int n = 1; n <= 8; n++) {
          // 打印每行的前导空格
       for (int k = 1; k <= (8 - n); k++) {
              System.out.print(" ");
           }
          // 每行打印 2*n-1 个星号
          for (int j = 1; j <= (2*n - 1); j++) {
              System.out.print("*");
          }
          System.out.println();      // 换行
      }
```

3.5.5 break 语句和 continue 语句

1. break 语句

break 语句是用来结束 while、do、for 结构的执行，该语句有两种格式：

```
break;
break 标签名;
```

2. continue 语句

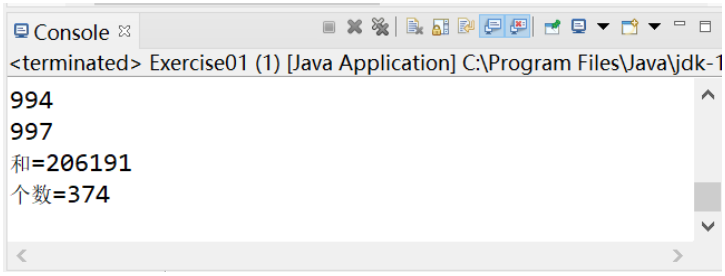
continue 语句与 break 语句类似，但它只终止执行当前的迭代，导致控制权从下一次迭代开始。该语句有下面两种格式：

```
continue;
continue  标签名;
```

观看视频自学	**3.6 案例研究：求最大公约数**

	1. 问题描述
	两个整数的最大公约数（Greatest Common Divisor，GCD）是能够同时被两个数整除的最大整数。例如，4 和 2 的最大公约数是 2，16 和 24 的最大公约数是 8。
	编写程序，要求从键盘输入两个整数，程序计算并输出这两个数的最大公约数。
	求两个整数的最大公约数有多种方法。一种方法是，假设求两个整数 m 和 n 的最大公约数，显然 1 是一个公约数，但它可能不是最大的。可以依次检查 k（k=2,3,4,…）是否是 m 和 n 的最大公约数，直到 k 大于 m 或 n 为止。
	该案例的设计思路主要如下：
	（1）从键盘输入两个整数，分别存入变量 m 和变量 n。
	（2）用下面循环结构计算能同时被 m 和 n 整除的数，循环结束后得到的 gcd 就是最大公约数。
	`while(k <= m && k <= n){`
	`if(m % k == 0 && n % k == 0)`
	`gcd = k;`
	`k++;`
	`}`
	完整代码见教材。

讲授内容	**3.7 案例研究：打印输出若干素数** 1. 问题描述 素数（prime number）又称质数，有无限个，在计算机密码学中有重要应用。素数定义为在大于 1 的自然数中，除了 1 和它本身以外不再有其他因数的数。编写程序计算并输出前 50 个素数，每行输出 10 个。 2 设计思路 该案例的设计思路主要如下： （1）要输出 50 个素数，首先用 while 循环对素数计数（用 count 变量）。 （2）要判断的数 number 从 2 开始，这里用一个 for 循环。根据定义，使用从 2 开始的数（divisor）去除要判断的数，如果直到 divisor = number - 1 仍不能整除，则 number 就是一个素数。 （3）打印输出素数。为了输出美观，可以使用格式化输出方法，这里使用了 System 类的 printf()方法，它可以对输出数据进行格式化。 `System.out.printf("%5d%n",number);` 该语句用宽度 5 个字符位置（%5d）输出 number，输出数字右对齐，输出后换行（%n）。 完整代码见教材。
课堂互动与讨论	编写程序，分别使用 while 循环、do~while 循环和 for 循环结构，计算并输出 1-1000 之间含有 7 或者是 7 倍数的整数之和及个数。运行结果如图 3-26 所示。

<table>
<tr><td></td><td> 
</td></tr>
<tr><td>总结

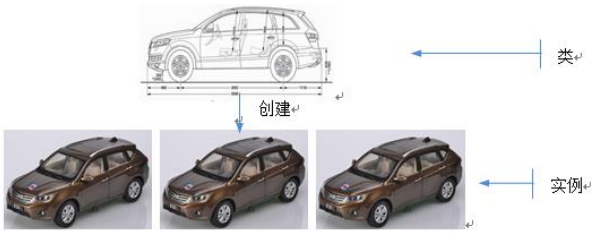
课后作业</td><td>
一、知识点总结
1. 循环语句有四种：while 循环、do-while 循环、for 循环和增强的 for 循环。循环中包含重复执行语句的部分称为循环体。
2. while 循环首先检查循环继续条件，如果条件为 true，则执行循环体；如果条件为 false，则循环结束。
3. do-while 循环与 while 循环类似，只是 do-while 循环先执行循环体，然后再检查循环继续条件，以确定是继续循环还是终止循环。
4. for 循环控制由三部分组成。第一部分是初始操作，通常用于初始化控制变量。第二部分是循环继续条件，决定是否执行循环体。第三部分是每次迭代后执行的操作，经常用于调整控制变量。for 循环一般用在循环体执行次数固定的情况。
5. 在循环体中可以使用 break 和 continue 这两个关键字。break 立即终止包含 break 的最内层循环。continue 只是终止当前迭代。
二、课后作业
1.登录本书在线作业网站，完成本章作业题目
2.编写程序，提示用户输入一个十进制整数，然后显示对应的二进制值。在这个程序中不要使用 Integer.toBinaryString(int)方法。
3.编写程序，求出 1~1000 之间的所有完全数。完全数是其所有因子（包括 1 但不包括该数本身）的和等于该数。例如 28=1+2+4+7+14，28 就是一个完全数。
</td></tr>
</table>

《Java 基础入门》教案

第 06 讲　　类的定义与对象的创建　　2 学时 110 分钟

章节内容	**第 4 章 类、对象和方法** 4.1　面向对象概述 4.2　类的定义与对象创建 4.2.1　类的定义 4.2.2　创建和使用对象 4.2.3　用 UML 图表示类 4.2.4　对象引用赋值 4.2.5　理解栈与堆
教学目标	• 了解面向对象基本概念。 • 掌握类的定义与对象的创建。 • 了解 UML 图、对象栈和堆的概念。 • 掌握对象引用赋值。
重点 难点	**重点**：类的定义，对象的创建和使用。 **难点**：理解栈与堆，以及对象在内存中表示。
教学环节 教学方法 及其它说 明事项	1.采用讲授法、演示法、讨论法。 2.案例+项目式教学。 3.采用线上线下混合教学模式。配套线上课程地址： www.qingline.net 4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。
导言 或复习	什么是 OOP? 面向对象编程（Object Oriented Programming，简称 OOP）是一种功能强大的程序设计方法，也是一种软件开发的新方法，使用这种方法开发的软件具有可复用、易维护和可扩展等特性。 OOP 的优势：OOP 的优势包括代码可复用、代码易维护和可扩展性。 **4.1.2　OOP 基本概念** 1.　对象 存在的一切事物都是对象 一个对象一般具有两方面的特征：状态和行为。状态用来描述对象的静态特征，行为用来描述对象的动态特征。 2.　类 所有的事物都可以归到某类中。例如，汽车属于交通工具类，手机属于通信工具类。 属于某个类的一个具体的对象称为该类的一个实例（instance）。例如，我的汽车是汽车类的一个实例。实例与对象是同一个概念。

类与实例的关系是抽象与具体的关系。类是多个实例的综合抽象，实例是某个类的个体实物。比如，汽车类相当于汽车的设计图纸。



3. 消息和方法

一个对象发送的消息包含三方面的内容：接收消息的对象；接收对象采用的方法（操作）；方法所需要的参数。

讲授内容

4.2 类的定义与对象创建

4.2.1 类的定义

一个类的定义包括两个部分：类声明和类体的定义。

1. 类声明

类声明的一般格式为：

```
[修饰符] class 类名 {
    // 1.成员变量
    // 2.构造方法
    // 3.成员方法
}
```

大括号内部称为类体，其中的每个部分都是可选的

【案例 4-1】假设开发一个处理银行业务的应用程序，就需要设计一个表示账户的类。一个账户应该有账号、姓名以及余额等属性，它们定义为成员变量。另外，账户应该有取款操作和存款操作，它们定义为方法。程序 4-1 定义一个账户类 Account。

【程序 4-1】Account.java

```
package com.boda.xy;
public class Account {
   public int id;
   public String name;
   public double balance;

   public Account (){}

   public void deposit(double amount) {
      balance = balance + amount;
      System.out.println("目前账户余额是：" + balance);
   }
   public void withdraw(double amount) {
       balance = balance - amount;
       System.out.println("目前账户余额是：" + balance);
```

成员变量的定义

默认构造方法

存款方法

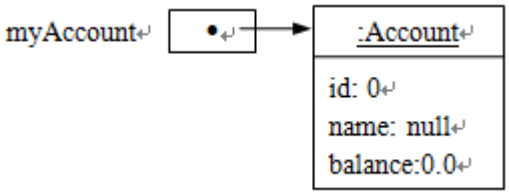
取款方法

```
    }
}
```

4.2.2 创建和使用对象

一般还要声明一个对象名，即声明对象的**引用**（reference），然后使用 new 运算符调用类的构造方法创建对象。例如，下面两行声明并创建 Account 类的一个实例：

```
Account myAccount;
myAccount = new Account();
```



若要声明多个同类型的对象名，可用逗号分开。

```
Account myAccount, yourAccount;
```

也可以将对象的声明和创建对象使用一个语句完成。

```
Account myAccount = new Account();
```

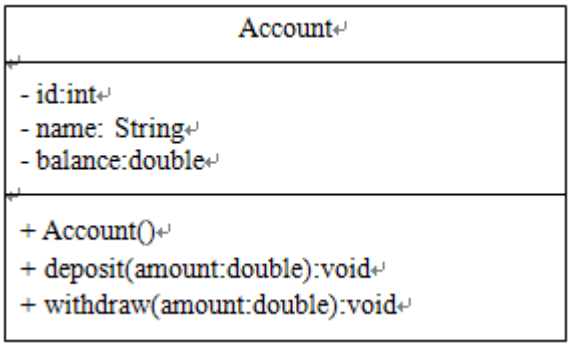
【案例 4-2】使用 Account 类创建一个对象并访问它的变量和方法。

【程序 4-2】AccountDemo.java

```
Account myAccount;            // 声明一个 Account 类型的引用变量
myAccount = new Account();  // 调用构造方法创建对象
myAccount.id = 1001;
myAccount.name = "李明月";          访问对象成员
myAccount.deposit(5000.00);         调用对象方法
myAccount.withdraw(3000.00);
// 输出账户信息
System.out.println("账户 ID = " + myAccount.id);
System.out.println("姓名 = " + myAccount.name);
System.out.println("余额 = " + myAccount.balance);
```

4.2.3 用 UML 图表示类

UML（Unified Modeling Language）称为**统一建模语言**，它是一种面向对象的建模语言，它用统一的、标准化的标记和定义实现对软件系统进行面向对象的描述和建模。



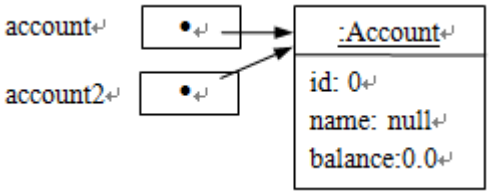
从图可以看到，Account 类包含三个私有成员变量，一个构造方法和两个普通方法。在 UML 图中，成员变量和类型之间用冒号分隔。方法的参数列表放在圆括号中，参数需指定名称和类型，它的返回值类型写在一个冒号后面。

4.2.4 对象引用赋值

对象的赋值是将对象的引用（地址）赋给变量。

```
Account account = new Account();
Account account2 = account;  // 将 account 的引用赋给 account2
```

上面的赋值语句执行结果是把 account 的引用赋值给了 account2，即 account 和 account2 的地址相同，也就是 account 和 account2 指向同一个对象，如图所示。



由于引用变量 account 和 account2 指向同一个对象，这时如果修改了 account 对象 name 属性值，输出 account2.name 值与 account.name 值相同。

```
account.name="李明月";
System.out.println(account2.name);    // 输出结果是“李明月”
```

4.2.5 理解栈与堆

当 Java 程序运行时，JVM 需要给数据和代码分配内存空间。JVM 将内存大致分为四个部分：栈区、堆区、代码区和静态区。

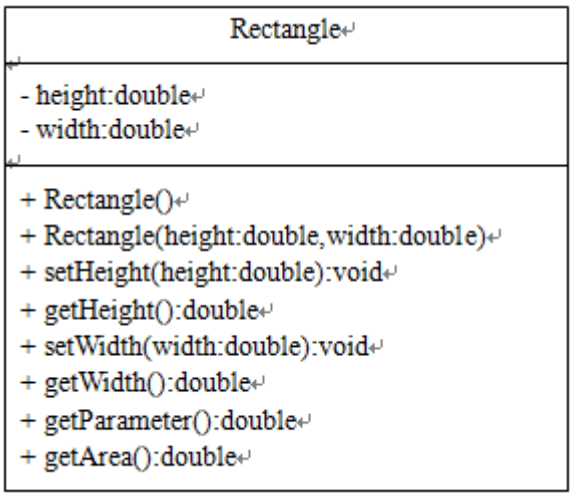
栈区：由编译器自动分配释放，方法的参数值、局部变量的值等存储在栈区，方法执行结束后，系统自动释放栈区内存。

堆区：由程序员分配释放，存放 new 创建的对象和数组，JVM 不定时检查堆区，如果对象没有引用指向它，内存将被回收。

静态区：存放全局变量、静态变量和字符串常量，不释放。

代码区：存放程序中方法的二进制代码，且多个对象共享一个代码空间区域。

讲授内容	
课 堂 互 动	1.定义一个名为 Person 的类，其中含有一个 String 类型的成员变量 name 和一个 int

与讨论	类型的成员变量 age，分别为这两个变量定义访问方法和修改方法， 2.另外再为该类定义一个名为 speak 的方法，在其中输出其 name 和 age 的值。 3.编写程序，使用上面定义的 Person 类，实现数据的访问、修改。
总结与作业	**一、知识点总结** （1）类是对象的模板。它定义对象的属性，并提供用以创建对象的构造方法以及操作对象的普通方法。 （2）类是一种引用数据类型。可以用它声明对象引用变量。对象引用变量包含的只是对该对象的引用，对象实际存储在内存堆中。 （3）对象是类的实例。可以使用 new 运算符创建对象，使用点运算符（.）通过对象的引用变量来访问该对象的成员。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2 .定义一个名为 Rectangle 的类表示矩形，该类的 UML 图如图 4-22 所示。其中求矩形周长的方法是 getPerimeter()，求面积的方法是 getArea()。 编写一个 RectangleDemo 应用程序，在 main()方法中用默认构造方法创建矩形对象后通过方法设置其高为 1 和宽为 2，然后输出其周长和面积。 

《Java 基础入门》教案

第 07 讲　　构造方法与方法设计　　2 学时 110 分钟

章节内容	**第 4 章 类、对象和方法** 4.3　构造方法 4.4　案例研究：使用 MyDate 日期类 4.5　方法设计 4.5.3　方法重载 4.5.4　方法参数的传递
教学目标	• 掌握构造方法设计。 • 掌握实例方法设计。 • 了解什么是方法重载。 • 掌握方法参数的传递。
重点 难点	**重点**：构造的定义，方法的定义与调用。 **难点**：方法的重载与方法参数的传递。
教学环节 教学方法 及其它说 明事项	1.采用讲授法、演示法、讨论法。 2.案例+项目式教学。 3.采用线上线下混合教学模式。配套线上课程地址： www.qingline.net 4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。

导言
或复习
	每个类都有构造方法，构造方法用来创建类的对象或实例。构造方法也有名称、参数和方法体。构造方法与普通方法的区别是：
	• 构造方法的名称必须与类名相同。
	• 构造方法不能有返回值，也不能返回void。
	• 构造方法必须在创建对象时用new运算符调用。
	构造方法定义的格式为：
	`[public|protected|private] 类名([参数列表]) {`
	`  // 方法体`
	`}`

讲授内容	**4.3.1　无参数构造方法** 如果在定义类是时没有为类定义任何构造方法，则编译器自动为类添加一个**默认构造方法**（default constructor）。默认构造方法是**无参数构造方法**（no-args constructor），方法体为空。假设没有为 Account 类定义构造方法，编译器提供的默认构造方法如下： `public Account(){}  // 默认构造方法` 用户也可以为类定义无参数构造方法，并在方法体中初始化对象。例如，在 Account 类中，可以定义下面的无参数构造方法：

```
public Account(){
    id = 0;
    name = "";          ← 初始化成员变量
    balance = 100;
}
```

使用无参数构造方法创建对象，只需在类名后使用一对括号即可，如下所示：

```
Account myAccount = new Account();
```

4.3.2 带参数构造方法

如果希望在创建一个对象时就将其成员变量设置为某个值，而不是采用默认值。这时可以定义带参数构造方法。例如，在创建一个 Account 对象时就指定账户 ID、姓名和余额，则可以定义如下带三个参数的构造方法。

```
public Account(int i, String n , double b){
    id = i ;
    name = n ;          ← 用参数初始化成员变量
    balance = b ;
}
```

然后，在创建 Account 对象时可以指定账户 ID、姓名和余额，如下代码创建一个 ID 为 1002，姓名为“李清泉”，余额为 8000 的账户对象。

```
Account account  = new Account(1002, "李清泉", 8000.00);
```

4.3.3 构造方法的重载

构造方法和普通方法都可以重载，所谓重载是名称相同、参数不同的方法。4.7 节将详细讨论方法重载。下面案例为 Account 类定义了三个重载的构造方法，其中包含一个无参数构造方法和两个带参数的构造方法。

【案例 4-4】带多个构造方法的账户类。

【程序 4-4】Account.java

4.3.4 this 关键字

this 表示对象本身，它自动传递给实例方法和构造方法。在一个方法的方法体或参数中，也可能声明与成员变量同名的局部变量，此时的局部变量会隐藏成员变量。要使用成员变量就需要在前面加上 this 关键字，例如：

```
public Account(int id, String name, double balance) {
    this.id = id;
    this.name = name;
    this.balance = balance;
}
```

this 关键字的另一个用途是在一个构造方法中调用该类的另一个构造方法。例如，假设在 Account 类定义了一个构造方法 Account (int id, String name,double balance)，现在又要定义一个无参数的构造方法，这时可以在下面的构造方法中调用该构造方法，如下所示：

```
public Account(){
```

```
        this(1002, "张明月", 5000);
    }
```

案例学习课堂互动与讨论

4.4 案例研究：使用 MyDate 日期类

1. 问题描述

在 Java 程序中经常需要使用其他的类，包括核心类库的类、用户自定义的类和第三方类库的类。这里笔者定义了一个名为 MyDate 的日期类，并且将它打包到一个 JAR 文件中，现在看如何使用这个类。MyDate 类的 UML 图如图 4-7 所示。该类定义了 3 个私有成员变量，2 个构造方法，8 个普通方法。针对上面给出的 MyDate 类，完成下面操作：

（1）创建一个日期对象 today 表示今天的日期（用默认构造方法），调用 today 的 toString()方法输出今天的日期。

（2）创建一个日期对象 birthday 表示你的出生日期（用带参数构造方法）。

（3）输出你的出生日期的年是否是闰年。

（4）计算并输出从你出生到今天一共过去多少天。

【程序 4-5】MyDateDemo.java

```
package com.boda.xy;
import com.boda.MyDate;
public class Exercise04{
    public static void main(String[]args){
        MyDate today = new MyDate();
        System.out.println(today.toString());
        MyDate birthday = new MyDate(2002,10,26);
        System.out.println(birthday.getYear()+"年"
             +(birthday.isLeapYear()?"是闰年":"不是闰年") );
        System.out.println("你已出生" + today.between(birthday) + "天。");
    }
```

```
    }
```

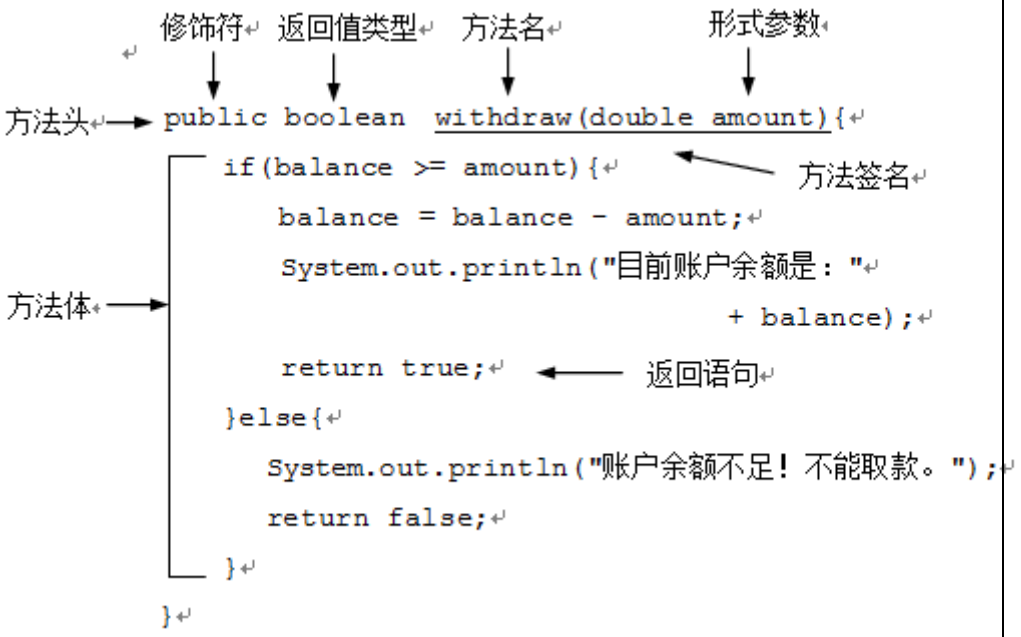
课堂讲授

4.5 方法设计

本节将学习如何设计方法、方法的重载、方法的参数传递等。

4.5.1 如何设计方法

设计方法包括方法的返回值、参数以及方法的实现等。下面重新设计了 Account 类中定义的 withdraw()方法，这里考虑了取款时账户余额不足的问题，如图 4-9 所示。



访问方法和修改方法

一般地，把能够返回成员变量值的方法称为**访问方法**（accessor method），把能够修改成员变量值的方法称为**修改方法**（mutator method）。访问方法名一般为 getXxx()，因此，访问方法也称 getter 方法。修改方法名一般为 setXxx()，因此，修改方法也称 setter 方法。访问方法的返回值一般与原来的变量值类型相同，而修改方法的返回值为 void。例如，在 Account 类中有一个 name 成员变量，那么就可以定义下面两个方法：

```
public void setName(String name){
    this.name = name;
}
public String getName(){
    return name;
}
```

修改方法

访问方法

当然，也可以为一个成员只定义访问方法而不定义修改方法，那么这个成员变量就是只读的，也可以只定义修改方法。

4.5.2 方法的调用

一般来说，要调用类的实例方法应先创建一个对象，然后通过对象引用调用，例如：

```
Account myAccount = new Account();
myAccount.deposit(3000.00);
```

	如果要调用类的静态方法，通常使用类名调用，例如：
	`double rand = Math.random();  // 返回一个随机浮点数`
	在调用没有参数的方法时，只使用圆括号即可，对于有参数的方法需要提供实际参数。关于方法参数的传递将在 4.5.4 节讨论。
	方法调用主要使用在三种场合：（1）用对象引用调用类的实例方法。（2）类中的方法调用本类中的其他方法。（3）用类名直接调用 static 方法。
	4.5.3 方法重载
	Java 语言提供了方法重载的机制，它允许在一个类中定义多个同名的方法，这称为**方法重载**（method overloading）。实现方法重载，要求同名的方法要么参数个数不同，要么参数类型不同，仅返回值不同不能区分重载的方法。方法重载就是在类中允许定义签名不同的方法。
	【案例 4-5】在类中定义多个重载的方法。这里，在 OverloadDemo 类中定义了 4 个重载的 display()方法。定义了重载方法后，对重载方法的调用与一般方法的调用相同。
	【程序 4-6】OverloadDemo.java
	4.5.4 方法参数的传递
	对带参数的方法，调用方法时需要向它传递参数。在 Java 语言中，方法的参数传递是**按值传递**（pass by value），即在调用方法时将实际参数的值的一个拷贝传递给方法中的形式参数，方法调用结束后实际参数的值并不改变。形式参数是局部变量，其作用域只在方法内部，离开方法后自动释放。
	尽管参数传递是按值传递的，但对于基本数据类型的参数和引用数据类型的参数的传递还是不同的。对于基本数据类型的参数，是将实际参数值的一个拷贝传递给方法，方法调用结束后，对原来的值没有影响。当参数是引用类型时，实际传递的是引用值，因此在方法的内部有可能改变原来的对象。
	【案例 4-6】两种参数传递，按值传递和按引用传递。
	【程序 4-7】PassByValue.java
	【注意】 如果为方法传递的是不可变的引用类型对象（如 String 对象），对象在方法内部不可能被改变。

总结 **课后作业**	**一、知识点总结** （1）方法头指定方法的修饰符、返回值类型、方法名和参数。方法可以返回一个值，如果方法不返回值，则返回值类型使用关键字 void。有返回值的方法必须使用 return 语句返回一个值。无返回值的方法也可以使用 return 语句。 （2）参数列表是指方法中参数的类型、个数和次序。方法名和参数列表一起构成方法签名。参数是可选的，即一个方法可以不包含参数。 （3）传递给方法的实际参数应该与方法签名中的形式参数具有相同的个数、类型和顺序。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2..为一元二次方程 $ax^2+bx+c=0$ 设计一个名为 QuadraticEquation 的类。这个类包括： • 代表三个系数的私有数据域a、b和c。 • 一个参数为a、b和c的构造方法。

	• a、b、c的三个getter方法。 • 一个名为getDiscriminant()的方法返回判别式，b^2-4ac。 • 名为getRoot1()和getRoot2()的方法返回方程的两个根。 $$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a} \text{ 和 } x_2=\frac{-b-\sqrt{b^2-4ac}}{2a}$$ 这些方法只有在判别式为非负数时才有用，如果判别式为负，这些方法返回 0。 编写一个测试程序，提示用户输入 a、b 和 c 的值，然后显示判别式的结果。如果判别式为正数，显示两个根；如果判别式为 0，显示一个根；否则显示“方程无根”。

第 08 讲　　静态成员与对象初始化　　2 学时 110 分钟

章节内容	**第 4 章 类、对象和方法** 4.7　静态变量和静态方法 4.8　方法递归调用 4.10　对象初始化 4.11　变量的作用域 4.12　局部变量类型推断 4.13　垃圾回收　1
教学目标	• 重点掌握静态成员的定义和使用。 • 掌握方法的递归调用。 • 了解对象初始化和变量作用域。 • 掌握局部变量类型推断 var 的使用。 • 了解垃圾回收的概念。
重点 难点	重点：静态成员的定义和使用。方法的递归调用。 难点：方法的递归调用。
教学环节 教学方法 及其它说明事项	1.采用讲授法、演示法、讨论法。 2.案例+项目式教学。 3.采用线上线下混合教学模式。配套线上课程地址： www.qingline.net 4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。
导言 或复习	前面学习了实例变量和实例方法的定义和使用，这节来学习静态变量和静态方法的使用。另外还将学习方法的递归调用以及对象初始化等内容。

讲授内容	**4.7　静态变量和静态方法**
	如果成员变量用 static 修饰，则该变量称为**静态变量**或**类变量**（class variable），否则称为**实例变量**（instance variable）。如果成员方法用 static 修饰，则该方法称为**静态方法**或**类方法**（class method），否则称为**实例方法**（instance method）。
	4.7.1　静态变量
	实例变量和静态变量的区别是：在创建类的对象时，Java 运行时系统为每个对象的实例变量分配一块内存，然后通过该对象来访问该实例变量。不同对象的实例变量占用不同的存储空间，因此它们是不同的。而对于静态变量，Java 运行时系统在类装载时为这个类的每个静态变量分配一块内存，以后再生成该类的对象时，这些对象将共享同名的静态变量，每个对象对静态变量的改变都会影响到其他对象。
	【案例 4-7】类的实例变量和静态变量区别。Counter 类定义了一个实例变量 x 和一个静态变量 y。
	【程序 4-9】Counter.java
	`package com.boda.xy;`
	`public class Counter{`

```
    public int x ;                  // 实例变量
    public static int y = 0 ;       // 静态变量
    public Counter(){
        x = 100;
    }
}
```

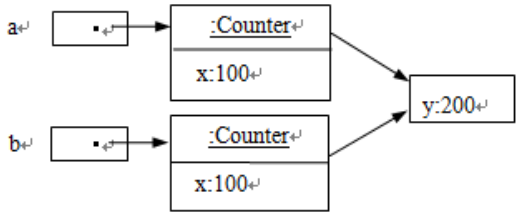
对于静态变量通常使用类名访问，如下所示：

```
Counter.y = 100;
Counter.y = 200;
System.out.println(Counter.y);   // 输出 200
```

可以通过实例名访问静态变量，但这种方法可能产生混乱的代码，不推荐通过实例访问静态变量。

```
Counter a = new Counter();
Counter b = new Counter();
a.y = 100;
b.y = 200;
System.out.println(a.y);   // 输出结果为 200
```

上述语句执行后的效果如图 4-14 所示。a 和 b 两个对象共享一个 y 存储空间。



4.7.2 静态方法

静态方法和实例方法的区别是：静态方法属于类，它只能访问静态变量。实例方法可以对当前的实例变量进行操作，也可以对静态变量进行操作。静态方法通常用类名调用，也可以用实例变量调用，实例方法必须由实例来调用。注意，在静态方法中不能使用 this 和 super 关键字。

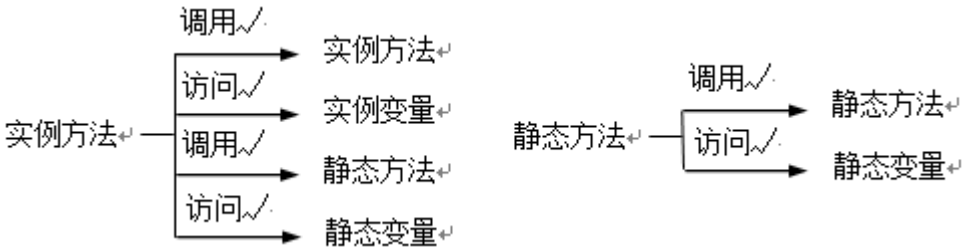
【案例 4-8】类的静态方法。静态方法属于类，可以访问类的静态成员，但不能方法实例成员。

【程序 4-10】SomeClass.java

```
package com.boda.xy;
public class SomeClass{
   int x = 5;
   static int y = 48;
   // 静态方法的定义
   public static void display(){
      y = y + 100;                          ← 可以访问静态变量
      System.out.println("y = "+ y);
      x = x * 5 ;                           ← 编译错误，不能访问实例成员
      System.out.println("x = "+ x);
   }
```

```
}
```

关于类的静态成员和实例成员总结如下：实例方法可以调用实例方法和静态方法，以及访问实例变量和静态变量。静态方法可以调用静态方法以及访问静态变量。静态方法不能调用实例方法或访问实例变量，因为静态方法和静态变量不属于某个特定对象。静态成员和实例成员的关系如图 4-16 所示。



4.8 方法递归调用

递归（recursion）是解决复杂问题的一种常见方法。其基本思想就是把问题逐渐简单化，最后实现问题的求解。例如，求正整数 n 的阶乘 n!，就可以通过递归实现。n!可按递归定义如下：

```
0! = 1;
n! = n ×（n-1）!; n > 0
```

Java 语言支持方法的递归调用。所谓方法的递归调用就是方法自己调用自己。设计算 n!的方法为 factor(n)，则该算法的简单描述如下：

```
if(n == 0)
   return 1;
else
   return factor(n-1) * n;
```

【案例 4-9】用递归方法计算整数的阶乘。

【程序 4-11】RecursionDemo.java

```
package com.boda.xy;
public class RecursionDemo{
   public static long factor(int n){
      if(n == 0)
       return 1 ;                          ← 计算 n 阶乘的递归方法
      else
          return n * factor(n-1);          ← 调用方法自己，但参数不同
   }
   public static void main(String[] args){
      int k = 20 ;
      System.out.println(k+"!="+factor(k));
      System.out.println("Long.MAX_VALUE = "
             + Long.MAX_VALUE); // long 型数的最大值
   }
}
```

课堂互动与讨论

4.9 案例研究：打印斐波那契数列

1. 问题描述

斐波那契数列（Fibonacci sequence），又称黄金分割数列，是意大利数学家列昂纳多·斐波那契（Leonardo Fibonacci）提出，它指的是这样一个数列：1、1、2、3、5、8、13、21、34、……，依次类推下去，后面的每个数是前面两个数的和。在这个数列中的数，就被称为斐波那契数。

斐波那契数列在很多领域都有应用，自然界存在很多斐波那契数，如树的叶子数、向日葵的花瓣数等。研究发现，斐波那契数还有很多有趣的现象，比如当数较大时，前一项与后一项的比值越来越逼近黄金分割比 0.618。将杨辉三角形的数按左对齐排列（见案例 5-4 节结果），将斜行的数相加就可得到斐波那契数列。

本案例要求使用方法递归，计算并输出前 20 项斐波那契数，并计算最后两项的比值。

2. 设计思路

在数学上，斐波那契数列可按如下以递归的方式定义：

$F(1)=1$，$F(2)=1$, $F(n)=F(n-1)+F(n-2)$，（$n \geq 3$，n 为自然数）

根据上述定义，我们可以编写下面方法计算第 n 个斐波那契数。

```
public static long fib(int n){
        if(n ==1 || n == 2){
            return 1;
        }else{
            return fib(n-1) + fib(n-2);
        }
}
```

这是方法递归调用

这里利用了方法的递归调用，当 n 的值大于等于 3 时，使用所求出的前两项之和计算出下一项的斐波那契数。

有了求斐波那契数的递归方法 fib(int n)，要计算前 20 个数就只需用一个循环即可，代码如下。

【程序 4-12】FibonacciNumber.java

讲授内容

4.10 对象初始化

在 Java 程序中需要创建许多对象。为对象确定初始状态称为对象初始化。对象初始化主要是指初始化对象的成员变量。实例变量和静态变量的初始化略有不同。当一个对象不再使用，应该清除以释放它所占的空间，通过垃圾回收器清除对象。

4.10.1 实例变量的初始化

Java 语言能够保证所有的对象都被初始化。实例变量的初始化有下面几种方式：（1）声明时初始化。（2）使用初始化块。（3）使用构造方法初始化。

1. 成员变量默认值

在类的定义中如果没有为变量赋初值，则编译器为每个成员变量指定一个默认值。对引用类型的变量，默认值为 null。对基本数据类型的变量，默认值如表 4-1

所示。

【案例 4-10】使用成员变量默认值初始化对象状态。

【程序 4-13】Student.java

```
package com.boda.xy;
public class Student{
   int id;
   String name;
   double marks;
   boolean pass;
   // 定义成员方法
   public void display(){
      System.out.println("id = "+id);
      System.out.println("name = "+name);
      System.out.println("marks = "+marks);
      System.out.println("pass = "+pass);
   }
   public static void main(String[] args){
      Student s = new Student();
      s.display();
  }
}
```

成员变量没有赋初值，它们将使用默认值。

2. 在变量声明时初始化

可以在成员变量声明的同时为变量初始化，如下所示。

3. 使用初始化块初始化

在类体中使用一对大括号定义一个初始化块，在该块中可以对实例变量初始化。例如：

```
int id;
String name;
double marks;
boolean pass ;
{
    id = 1001;
    name = "李明";
    marks = 90.5;
    pass = true;

}
```

这里是初始化块，由一对大括号定界。

4. 使用构造方法初始化

可以在构造方法中对变量初始化，例如，对于 Student 类可以定义下面的构造方法。

```
public Student(int id, String name, double marks, boolean pass){
    this.id = id;
    this.name = name;
```

用构造方法参数初始化成员变量。

```
        this.marks = marks;
        this.pass = pass;
    }
```

5. 初始化次序

从上面程序输出结果可以看到，构造方法被最后执行，这与初始化块和构造方法在源代码中的位置无关。实际上，程序是按下面顺序为实例变量初始化的。

（1）首先使用默认值或指定的初值初始化。

（2）接下来执行初始化块。

（3）最后再执行构造方法。

4.10.2 静态变量的初始化

静态变量的初始化与实例变量的初始化类似，静态变量如果在声明时没有指定初值，编译器也将使用默认值为其赋初值。主要方法有：（1）声明时初始化。（2）使用静态初始化块。（3）使用构造方法初始化。

1. 静态初始化块

对于 static 变量除了可以使用前两种方法初始化外，还可以使用静态初始化块。静态初始化块是在初始化块前面加上 static 关键字。例如，下面的类定义就使用了静态初始化块。

```
public class StaticDemo{
   static int x = 100;
   static{
       x = 48;
   }
   public StsticDemo{
      x = 88;
   }
   //其他代码
}
```

静态初始化块。

2. 初始化顺序

当一个类有多种初始化方法时，执行顺序是：

（1）用默认值给 static 变量赋值，然后执行静态初始化块为 static 变量赋值。

（2）用默认值给实例变量赋值，然后执行初始化块为实例变量赋值。

（3）最后使用构造方法初始化实例变量和静态变量。

4.11 变量的作用域

在 Java 中有多个地方可以使用变量，如类的成员变量、方法的局部变量、代码块中声明的变量（如 for 语句以及异常处理块等）。

变量的**作用域**（scope）是指一个变量可以在程序的什么范围内被使用。Java 程序的作用域是通过块实现的，块（block）是使用一对大括号封装的语句序列，块可对语句进行分组并定义变量的作用域。一般来说，变量只在其声明的块中可见，在块外不可见。若一个变量属于某个作用域，它在该作用域可见，即可被访问，否则不能被访问。

变量的**生存期**（lifetime）是指变量被分配内存的时间期限。当声明一个方法局

部变量时，系统将为该变量分配内存，只要方法没有返回，该变量将一直保存在内存中。一旦方法返回，该变量将从内存栈中清除，它将不能再被访问。

对于对象，当使用 new 创建对象时，系统将在堆中分配内存。当一个对象不再被引用时，对象和内存将被回收。但不是对象离开作用域就立即被回收。实际上是在之后某个时刻当垃圾回收器运行时才被回收。

4.12 局部变量类型推断

在方法中声明一个引用类型变量并使用构造方法创建对象时，需要两次输入变量类型，第一次用于声明变量类型，第二次用于构造方法，比如：

```
Account myAccount = new Account();
```

在声明变量时使用一次类型名 Account，在创建实例时又使用一次类型名，显然存在代码冗余问题。从 Java 10 开始，开发人员可以使用 var 声明一个局部变量，然后通过构造方法或工厂方法创建对象，在创建对象时让编译器自己去推断类型，从而确定变量的类型：

```
var myAccount = new Account();
```

在处理 var 时，编译器先是查看表达式右边部分，也就是所谓的构造器，并将它作为变量的类型。这样可以少输入一些字符，更重要的是，它避免了信息冗余，而且对齐了变量名，更容易阅读。

不是在任何时候都可使用 var 声明变量，它只能用在局部变量声明并创建时，更准确地说，是那些带有构造器的局部变量声明。下面用法不正确。

```
var myAccount;
myAccount = new Account();
```

不可以,现在不能推断 myAccount 的类型

字符串是引用类型，下面用法是正确的：

```
var name = "张大海";
```

4.13 垃圾回收

在 Java 程序中允许创建尽可能多的对象，而不用担心销毁它们。当程序使用完一个对象，该对象不再被引用时，Java 运行系统就在后台自动运行一个线程，它将销毁该对象并释放其所占的内存空间，这个过程称为**垃圾回收**（Garbage Collection，GC）。

后台运行的线程称为**垃圾回收器**（garbage collector）。垃圾回收器自动完成垃圾回收操作，因此，这个功能也称为自动垃圾回收。所以，在一般情况下，程序员不必关心对象不被清除而产生内存泄露问题。

1. 对象何时有可能被回收

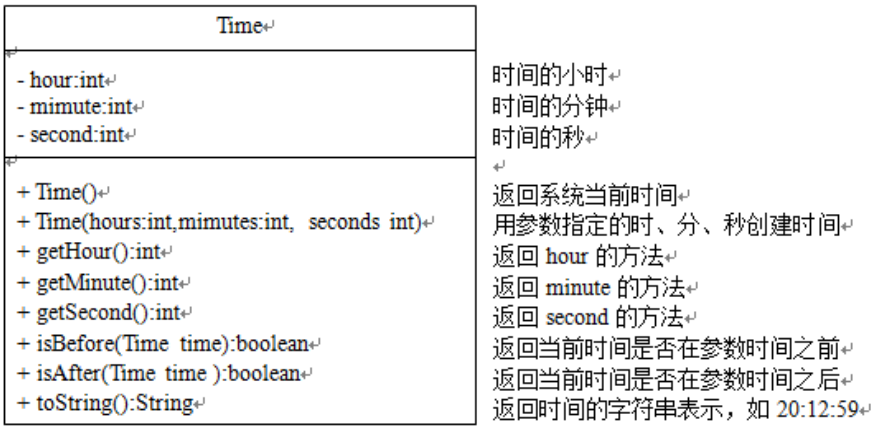
当一个对象不再被引用时，该对象才有可能被回收。请看下面代码。

```
Account account = new Account (),account2 = new Account ();
account2 = account;
```

上面代码段创建了两个 Account 对象 account、account2，然后让 account2 指向 account，这时 account2 原来指向的对象没有任何引用指向它了，也没有任何办法得到或操作该对象了，该对象就有可能被回收了。

	另外，也可明确删除一个对象的引用，这通过为对象引用赋 null 值即可，如下所示：
	`account2 = null ; // 原来的 account2 对象可被回收，注意与上面代码的区别`
	一个对象可能有多个引用，只有在所有的引用都被删除，对象才有可能被回收。
	2. 强制执行垃圾回收
	尽管 Java 提供了垃圾回收器，但不能保证不被使用的对象及时被回收。如果希望系统运行垃圾回收器，可以直接调用 System 类的 gc()方法，如下所示：
	`System.gc();`
	另一种调用垃圾回收器的方法是通过 Runtime 类的 gc()实例方法，如下所示：
	`Runtime rt = Runtime.getRuntime();`
	`rt.gc();`
	要注意，启动垃圾回收器并不意味着马上能回收无用的对象。因为，执行垃圾回收器需要一定的时间，且受各种因素如内存堆的大小、处理器的速度等的影响，因此垃圾回收器的真正执行是在启动垃圾回收器后的某个时刻才能执行。

总结与 课后作业	**一、知识点总结** （1）静态成员使用 static 修饰符修饰。类的每个实例都能访问这个类的静态变量和静态方法。然而，为清晰起见，最好使用“类名.变量”和“类名.方法”来访问静态变量和静态方法。 （2）实例变量的初始化顺序是在声明时初始化、使用初始化块初始化、使用构造方法初始化。静态变量的初始化顺序是声明时初始化、使用静态初始化块初始化、使用构造方法初始化。 （3）当一个对象不再被使用，自动调用后台垃圾回收器销毁对象，也可以调用 System.gc()方法或者 Runtime 实例的 gc()方法强制执行垃圾回收器。但这些方法都不保证系统立即回收无用对象。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2.定义 Time 类表示时间，该类有三个数据成员：hour（小时）、minute（分钟）和 second（秒），为该类定义两个构造方法和其他方法，这些方法如的 UML 图如图所示。提示：类的默认构造方法创建的时间为当前系统时间，请参阅 2.10 节的案例研究。关于时间对象还应满足下面条件：①三个数据成员都必须是非负整数；②hour 值应该在 0 到 23 之间，minute 和 second 值应该在 0 到 59 之间。编写应用程序测试 Time 类的使用。



2　编写一个方法 gcd(m,n)使用递归求 m 和 n 的最大公约数。gcd()方法可以递归地定义如下：

- 如果m%n结果为0，那么gcd(m,n)的结果为n；
- 否则，gcd(m,n)就是gcd(n,m%n)。

编写测试程序，提示用户输入两个整数，显示它们的最大公约数。

第 09 讲　　数组　　2 学时 110 分钟

<table>
<tr><td>章节内容</td><td>第 5 章 数组
5.1　创建和使用数组
5.1.4　增强的 for 循环
5.2　数组的应用
5.2.1　数组元素的复制
5.2.2　数组参数与返回值
5.2.3　可变参数方法
5.4　java.util.Arrays 类</td></tr>
<tr><td>教学目标</td><td>• 了解数组的应用。
• 掌握数组的定义和初始化以及元素的访问。
• 掌握增强的 for 循环，与可变参数方法。
• 了解 Arrays 类的使用。</td></tr>
<tr><td>重点
难点</td><td>重点：数组的定义与应用，增强 for 循环。
难点：可变参数方法的定义和使用。数组参数与返回值。</td></tr>
<tr><td>教学环节
教学方法
及其它说明
事项</td><td>1.采用讲授法、演示法、讨论法。
2.案例+项目式教学。
3.采用线上线下混合教学模式。配套线上课程地址：
www.qingline.net
4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。</td></tr>
<tr><td>导言
或复习</td><td>数组是几乎所有程序设计语言都提供的一种数据存储结构。Java 语言的数组是一种引用数据类型，即数组是对象。数组在程序设计中具有广泛应用，它通常用来存储和操作一组类型相同的数据。本章首先介绍数组的创建和使用，然后介绍数组的有关应用、Arrays 类的使用和二维数组。面向对象编程（Object Oriented Programming，简称 OOP）是一种功能强大的程序设计方法，也是一种软件开发的新方法，使用这种方法开发的软件具有可复用、易维护和可扩展等特性。
统计学生成绩问题，使用数组就很方便。
有 5 名学生学年考试成绩如下表所示：
科目 1　科目 2　科目 3　科目 4
80　75　78　93
67　87　98　65
86　72　60　76
76　80　76　63
82　70　90　67</td></tr>
<tr><td>讲授内容</td><td>5.1　创建和使用数组
所谓数组（array）是名称相同，下标不同的一组变量，它用来存储一组类型相同的数据。</td></tr>
</table>

5.1.1 数组定义

两个步骤：（1）声明数组：声明数组名称和元素的数据类型。（2）创建数组：为数组元素分配存储空间。

1. 声明数组

使用数组之前需要声明，声明数组就是告诉编译器数组名和数组元素类型。

例如，下面代码声明了两个数组。

```
double []marks;
Account []accounts;
```

2. 创建数组

数组声明仅仅声明一个数组对象引用，而创建数组是为数组的每个元素分配存储空间。创建数组使用 new 语句。

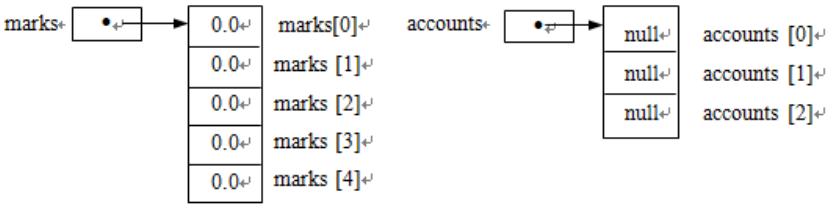
下面代码创建 marks 数组和 accounts 数组。

```
marks = new double[5];          // 数组包含 5 个 double 型元素
accounts = new Account[3];      // 数组包含 3 个 Account 型元素
```

数组的声明与创建可以写在一个语句中，例如：

```
double []marks = new double[5];
Account []accounts = new Account[3];
```

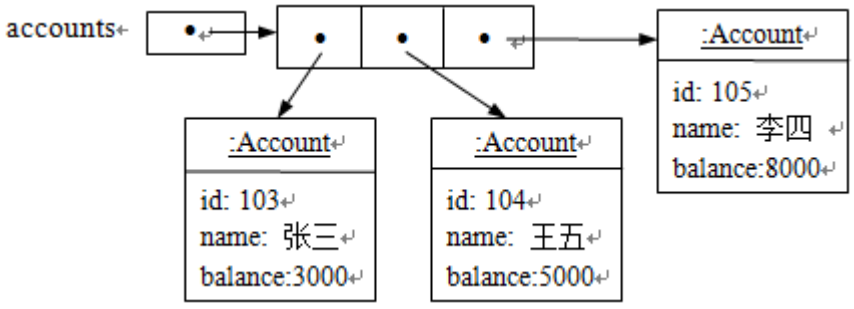
声明同时创建数组



对于引用类型数组（对象数组），它的每个元素初值为 null，因此，还需要创建数组元素对象。

```
accounts[0] = new Account(103,"张三",3000.0);
accounts[1] = new Account(104,"王五",5000.0);
accounts[2] = new Account(105,"李四",8000.0);
```

上面语句执行后效果如图 5-2 所示。



5.1.2 访问数组元素

声明了一个数组，并使用 new 运算符为数组元素分配内存空间后，就可以使用

	数组中的每一个元素。
	通过数组名和下标访问数组元素，下标从 0 开始，到数组的长度减 1。例如，上面定义的 accounts 数组包含 3 个元素，所以只能使用 accounts[0]、accounts[1]和 accounts[2]这三个元素。数组一经创建大小不能改变。
	数组作为对象提供一个 length 成员变量，它的值是数组元素个数，访问该成员变量的方法为“数组名.length”。
	【案例 5-1】数组的声明、创建以及元素和 length 成员的使用。
	【程序 5-1】ArrayDemo.java
	为了保证安全性，Java 运行时系统要对数组元素的范围进行越界检查，若数组元素下标超出范围，将抛出 ArrayIndexOutOfBoundsException 运行时异常。例如，下面代码抛出异常。
	`System.out.println(marks[5]);`
	5.1.3 数组初始化器
	声明数组同时可以使用初始化器对数组元素初始化，它是在一对大括号中给出数组的每个元素值。这种方式适合数组元素较少的情况，这种初始化也称为静态初始化。
	`double[] marks = {79, 84.5, 63, 90, 98};`
	`Account[] accounts = {new Account(103, "张三", 3000),`
	`new Account(104, "王五", 5000), new Account(105,"李四",8000)};`
	5.1.4 增强的 for 循环
	如果程序只需顺序访问数组中每个元素，可以使用增强的 for 循环，它是 Java 5 新增功能。增强的 for 循环可以用来迭代数组和对象集合的每个元素。它的一般格式为：
	`for(type identifier: expression) {`
	`// 循环体`
	`}`
	该循环的含义为：对 expression（数组或集合）中的每个 type 类型的元素 identifier，执行一次循环体中的语句。这里，type 为数组或集合中的元素类型。expression 必须是一个数组或集合对象。
	下面使用增强的 for 循环实现求数组 marks 数组中各元素的和，代码如下：
	`double sum = 0;`
	`for(var score : marks){`
	`sum = sum + score;`
	`}`
	`System.out.println("总成绩=" + sum);`

讲授内容	**5.2 数组的应用** **5.2.1 数组元素的复制**

经常需要将一个数组的元素复制到另一个数组中。首先应该想到将数组元素一个一个复制到目标数组中。设有一个数组 source，其中有 4 个元素，现在定义一个数组 target，与原来数组类型相同，元素个数相同。使用下面方法将源数组的每个元素复制到目标数组中。

```
int[] source = {10,30,20,40};                // 源数组
int[] target = new int[source.length];       // 目标数组
for(var i = 0; i < source.length; i++)
    target[i] = source[i] ;
```

除上述方法外，还可以使用 System 类的 arraycopy()方法，格式如下：

```
public static void arraycopy(Object src, int srcPos,
                             Object dest, int destPos,int length)
```

下面代码实现将 source 中的每个元素复制到数组 target 中。

```
int[] source = {10,30,20,40};
int[] target = new int[source.length];
System.arraycopy(source, 0, target, 0, 4);
```

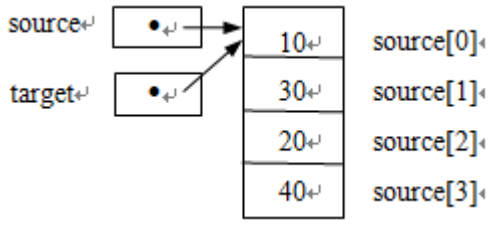
【案例 5-2】数组元素的复制与异常。

【程序 5-2】ArrayCopyDemo.java

注意，不能使用下列方法试图将数组 source 中的每个元素复制到 target 数组中。

```
int[] source = {10,30,20,40};
int[] target = source;    // 这是引用赋值
```

上述两条语句实现对象的引用赋值，两个数组引用指向同一个数组对象，如图 5-5 所示。



5.2.2 数组参数与返回值

数组对象可以作为参数传递给方法，例如下面代码定义了一个求数组元素和的方法。

```
public static double sumArray(double array[]){
   var sum = 0;
   for(var i = 0; i < array.length; i++){
      sum = sum + array[i];
   }
   return sum;
}
```

注意，由于数组是对象，因此将其传递给方法是按引用传递。当方法返回时，数组对象不变。但要注意，如果在方法体中修改了数组元素的值，则该修改反映到返回

	的数组对象。
	一个方法也可以返回一个数组对象，例如，下面的方法返回参数数组的元素反转后的一个数组。
	`public static int[] reverse(int[] list){`
	`  var result = new int[list.length];  // 创建一与参数数组大小相同的数组`
	`  for(var i = 0, j = result.length - 1; i < list.length; i++ , j--){`
	`  result[j] = list[i];  // 实现元素反转`
	`  }`
	`  return result;  // 返回数组`
	`}`
	有了上述方法，可以使用如下语句实现数组反转。
	`int[]  list = {6, 7, 8, 9, 10};`
	`int[] list2 = reverse(list);`
	5.2.3　可变参数方法
	Java 语言允许定义方法（包括构造方法）带可变数量的参数，这种方法称为可变参数（variable argument）方法。具体做法是在方法参数列表的最后一个参数的类型名之后、参数名之前使用省略号，例如：
	`public static double average(double … values){`
	`  // 方法体`
	`}`
	这里，参数 values 被声明为一个 double 型值的序列。其中参数的类型可以是引用类型。对可变参数的方法，调用时可以为其传递任意数量的指定类型的实际参数。在方法体中，编译器将为可变参数创建一个数组，并将传递来的实际参数值作为数组元素的值，这就相当于为方法传递一个指定类型的数组。
	【案例 5-3】可变长度参数方法的应用。
	【程序 5-3】VarargsDemo.java

课堂互动与讨论	**5.3　案例学习：数组起泡排序** 1.　问题描述 在程序设计中，排序是一种常见的操作。例如，一个数组可能存放某个学生的各科成绩，现要求将成绩按从低到高的顺序输出，这就需要为数组排序。排序有多种方法，如起泡排序、选择排序、插入排序、快速排序等。本案例采用起泡排序法（bubble sort）对一个数组按升序排序。 2.　设计思路 假设给定下面整型数组： `int[]a = {75, 53, 32, 12, 46, 199, 17,  54};` 根据问题要求，采用起泡排序法对该数组升序排序，设计思路如下：

	从数组的第一个元素开始，与它后面的元素比较，如果逆序（即 a[j] > a[j+1]），则交换两个元素的位置，如果正序，则不交换。经过第一趟排序后，结果如下。 `53, 32, 12, 46, 75, 17, 54, 199` 可以看到，经过第一趟排序，找到了数组中的最大值，而经过比较后，小的元素交换到前面，就像水泡向上冒。第二趟排序时，只需比较到倒数第二个元素为止，即可找到次大的元素，因此这里使用二重循环结构。
讲授内容	**5.4 java.util.Arrays 类** java.util.Arrays 类定义了若干静态方法对数组操作，包括对数组排序、在已排序的数组中查找指定元素、数组元素的拷贝、比较两个数组是否相等、将一个值填充到数组的每个元素中。 • public static int binarySearch (int[] a, int key)：根据给定的键值，查找该值在数组中的位置，如果找到指定的值，则返回该值的下标值。如果查找的值不包含在数组中，方法的返回值为（-插入点-1）。插入点为指定的值在数组中应该插入的位置。 • public static void sort(int[] a)：对数组 a 按自然顺序排序。 • public static void sort(int[] a, int fromIndex, int toIndex)：对数组 a 中的元素从起始下标 fromIndex 到终止下标 toIndex 之间的元素排序。 • public static double[] copyOf(double[] original, int newLength)：方法的 original 参数是原数组，newLength 参数是新数组的长度。 • public static void fill (int[] a, int val)：用指定的 val 值填充数组 a 中的每个元素。 • public static boolean equals(boolean[] a, boolean[] b)：比较布尔型数组 a 与 b 是否相等。 • public static String toString(int[] a)：将数组 a 的元素转换成字符串，它有多个重载的版本，方便对数组的输出。 【案例 5-4】用 Arrays 的 sort()方法排序数组。 【程序 5-5】SortDemo.java
总结 **课后作业**	**一、知识点总结** （1）使用 elementType[] arrayName 或者 elementType arrayName[]声明一个数组类型的变量，尽管这两种语法都是合法的，但推荐使用前者的风格。 （2）只有创建数组后才能给元素赋值。使用 new 操作符创建数组，语法如下：new elementType[arraySize]。 （3）数组变量是引用类型的变量。数组变量包含的是对数组的引用。 （4）数组中的每个元素都使用 arrayName[index]语法表示。下标必须是一个整数或整数表达式。 （5）创建数组后，它的大小不能改变，可以使用 arrayName.length 得到数组的大小。由于数组的下标是从 0 开始，所以，最后一个下标是 arrayName.length -1。如果试图访问数组界外的元素，就会发生越界错误。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2.编写程序，用数组初始化器初始化一个有 5 个 double 型数的数组 numbers，求数组中的最大、最小元素值及各元素的和与平均值。

	3.编写程序，用数组初始化器初始化一个有 5 个 int 型数的数组 source，分别使用下面方法将数组元素复制到目标数组 target 中。 （1）用 for 循环一个一个元素复制。 （2）使用 System 类的 arraycopy()方法复制。 （3）用 Arrays 类的 copyOf()或 copyOfRange()方法。

第 10 讲　　二维数组　　2 学时 110 分钟

<table>
<tr><td>章节内容</td><td>第 5 章　数组
5.6　二维数组
5.6.1　二维数组定义
5.6.2　数组元素的使用
5.6.3　数组初始化器
5.6.4　不规则二维数组
5.8　案例研究：打印输出魔方数</td></tr>
<tr><td>教学目标</td><td>• 了解二维数组的应用。
• 掌握二维数组的定义和使用。
• 了解数组初始化器。
• 了解不规则二维数组。</td></tr>
<tr><td>重点</td><td>重点：二维数组的定义和使用。</td></tr>
<tr><td>难点</td><td>难点：不规则二维数组的使用。</td></tr>
<tr><td>教学环节
教学方法
及其它说
明事项</td><td>1.采用讲授法、演示法、讨论法。
2.案例+项目式教学。
3.采用线上线下混合教学模式。配套线上课程地址：
www.qingline.net
4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。</td></tr>
<tr><td>导言
或复习</td><td>Java 语言中数组元素还可以是一个数组，这样的数组称为数组的数组或二维数组。

有 5 名学生学年考试成绩如下表所示：
科目 1　科目 2　科目 3　科目 4
80　75　78　93
67　87　98　65
86　72　60　76
76　80　76　63
82　70　90　67

编写程序，使用二维数组存储学生成绩，并完成下列操作：
(1) 计算并输出每名学生的总成绩。
(2) 打印输出总成绩最高的行号及总成绩。
(3) 打印输出每科最高分及所在行号。

这个问题就应该使用二维数组实现。</td></tr>
<tr><td>讲授内容</td><td>5.6.1　二维数组定义

二维数组的使用也分为声明、创建两个步骤。

1.　二维数组声明

二维数组有下面三种等价的声明格式：</td></tr>
</table>

```
元素类型 [][]数组名;
元素类型 数组名[][];
元素类型 []数组名[];
```

推荐使用第一种格式

这里，元素类型可以是基本类型，也可以是引用类型，数组名为合法的变量名。推荐使用第一种格式声明二维数组。下面语句声明了一个整型二维数组 matrix 和一个 String 型二数组 cities。

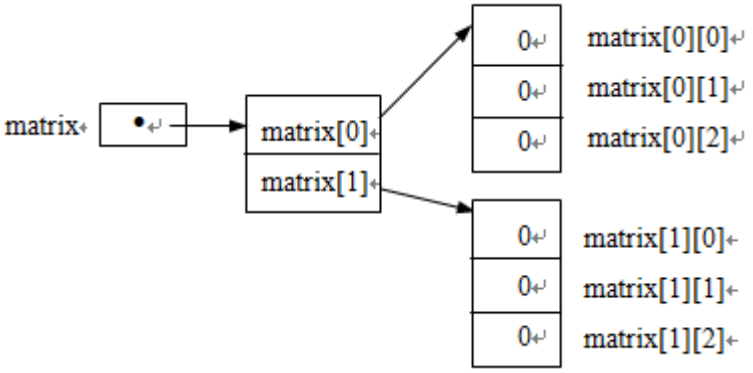
```
int [][] matrix;
String [][] cities;
```

2. 创建二维数组

创建二维数组就是为二维数组的每个元素分配存储空间。系统先为高维分配引用空间，然后再顺次为低维分配空间。二维数组的创建也使用 new 运算符，分配空间有两种方法，下面是直接为每一维度分配空间。

```
var matrix = new int[2][3];   // 直接为每一维分配空间
```

这种方法适用于数组的低维具有相同个数的数组元素。在 Java 中，二维数组是数组的数组，即数组元素也是一个数组。上述语句执行后创建的数组如图 5-11 所示，二维数组 matrix 有两个元素，matrix[0]和 matrix[1]，它们又都是数组，各有三个元素。图 5-11 中共有三个对象：matrix、matrix[0]和 matrix[1]。



创建了二维数组后，它的每个元素被指定为默认值。上述语句执行后，数组 matrix 的 6 个元素值都被初始化为 0。

在创建二维数组时，也可以先为第一维分配空间，然后再为第二维分配空间。

```
var matrix = new int[2][];      // 先为第一维分配空间
matrix[0] = new int[3];         // 再为第二维分配空间
matrix[1] = new int[3];
```

5.6.2 数组元素的使用

访问二维数组的元素，使用下面的形式：

```
数组名[下标 1][下标 2]
```

其中，下标 1 和下标 2 可以是整型常数或表达式。同样，每一维的下标也是从 0 到该维的长度减 1。

下面代码给 matrix 数组元素赋值：

```
matrix[0][0] = 80;
matrix[0][1] = 75;
matrix[0][2] = 78;
matrix[1][0] = 67;
matrix[1][1] = 87;
matrix[1][2] = 98;
```

下面代码输出 matrix[1][2]元素值：

```
System.out.println(matrix[1][2]);
```

对二维数组的第一维通常称为行，第二维称为列。要访问二维数组的所有元素，应该使用嵌套的 for 循环。如下面代码输出 matrix 数组中所有元素。

```
for(var i = 0; i < matrix.length; i++){
    for(var j = 0; j < matrix[0].length; j ++){
       System.out.print(matrix[i][j] + "  ");
    }
    System.out.println();     // 换行
}
```

5.6.3 数组初始化器

对于二维数组也可以使用初始化器在声明数组的同时为数组元素初始化，例如：

```
int[][] matrix = {{15,56,20,-2},
                  {10,80,-9,31},
                  {76,-3,99,21},};
```

matrix 数组是 3 行 4 列的数组。多维数组每一维也都有一个 length 成员表示数组的长度。matrix.length 的值是 3，matrix[0].length 的值是 4。

5.6.4 不规则二维数组

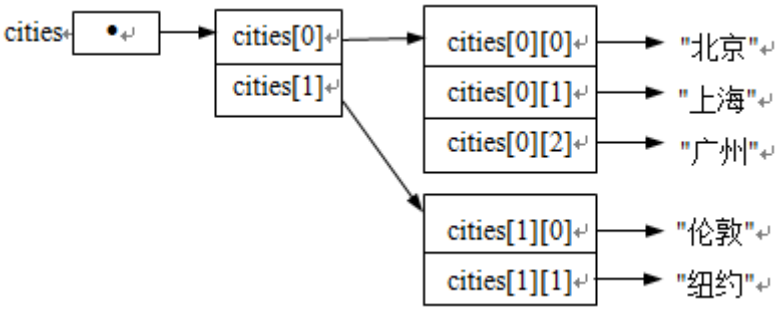
Java 的二维数组是数组的数组，对二维数组声明时可以只指定第一维的大小，第二维的每个元素可以指定不同的大小，例如：

```
var cities = new String[2][];     // cities 数组有 2 个元素
cities[0] = new String[3];        // cities[0]数组有 3 个元素
cities[1] = new String[2];        // cities[1]数组有 2 个元素
```

这种方法适用于低维数组元素个数不同的情况，即每个数组的元素个数可以不同。对于引用类型的数组，除了为数组分配空间外，还要为每个数组元素的对象分配空间。

```
cities[0][0] = new String("北京");
cities[0][1] = new String("上海");
cities[0][2] = new String("广州");
cities[1][0] = new String("伦敦");
cities[1][1] = new String("纽约");
```

cities 数组元素空间的分配情况如图 5-13 所示，图中共有 8 个对象。



课堂互动与讨论

5.8 案例研究：打印输出魔方数

1. 问题描述

在中国传统文化中，有一种图表叫九宫图，它是在 9 个格子中放置 1 到 9 数字，使得结果的每行、每列以及对角线上的 3 个数字之和相等，如图 5-16 所示。对于该图，古人还给出了图中数字的记忆方法，“戴九履一，左三右七，二四为肩，六八为足，五居中央”。

8	1	6
3	5	7
4	9	2

数学上已经证明，对于任何 n×n 个自然数（1，2，…，n^2，n 为奇数）都有这样的排列，使得每行、每列和每条对角线上数的和都相等。这些数称为魔方数。编写程序，要求用户从键盘输入要打印的魔方数（比如，5），程序计算并输出魔方数。

2. 设计思路

对于 n 为奇数的魔方数，可以按下列方法将 1 到 n^2 的数填充到二维数组的元素中。首先把 1 放在第 0 行的中间，剩下的数 2，3，…，n^2 依次向上移动一行，并向右移动一列。当可能越过数组边界时需要“绕回”到数组的另一端。例如，如果需要把下一个数放到-1 行，就将其存储到 n-1 行（最后一行）；如果需要把下一个数放到第 n 列，就将其到第 0 列。如果某个特定的数组元素已被占用，就把该数存储在前一个数的正下方。

根据上述规则，可按下列步骤设计实现本案例。

（1）首先从键盘输入一个正奇数 size，然后定义一个 size×size 的二维数组，代码如下。

```
int [][] magic = new int[size][size];
```

（2）将 1 填充到数组的第一行中间元素中，代码如下。

```
int row = 0, col = size/2;
magic[row][col] = 1;  // 将 1 填到第一行中间元素
```

（3）将 2 到 size*size 填充到数组其他元素中，这里重点是确定下一个元素填充的位置。通过定义临时变量存放元素的行和列，如果下一个位置超出数组元素范围，将改变其行或列的值，最终确定正确位置。

	（4）输出填充了数值的二维数组的每个元素。
总结 **课后作业**	**一、知识点总结** （1）可以使用二维数组存储表格数据。二维数组的使用也需要声明和创建数组。可以使用数组初始化器创建二维数组。 （2）二维数组可以是长度不同的。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2.有下面矩阵 A，编写程序求矩阵 A 的转置，即得到下面 B 的结果。 $A=\begin{pmatrix}1 & 3 & 5\\ -3 & 6 & 0\\ 13 & -5 & 7\\ -2 & 19 & 25\end{pmatrix}$　　$B=\begin{pmatrix}1 & -3 & 13 & -2\\ 3 & 6 & -5 & 19\\ 5 & 0 & 7 & 25\end{pmatrix}$

第 11 讲　　面向对象特征　　2 学时 110 分钟

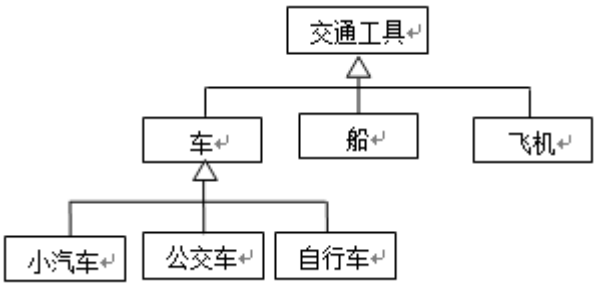
<table>
<tr><td>章节内容</td><td>第 6 章　面向对象特征
6.1　面向对象特征
6.2　包与类库
6.3　案例研究：开发自定义类库
6.4　封装性与访问修饰符
6.5　类的继承
6.5.2　方法覆盖
6.6　final 修饰符</td></tr>
<tr><td>教学目标</td><td>• 了解面向对象特征。
• 掌握包、类库、封装性和访问修饰符。
• 掌握类继承的实现、方法覆盖、super 的使用。
• 了解 final 关键字的使用。</td></tr>
<tr><td rowspan="2">重点
难点</td><td>重点：包、类库、封装性和访问修饰符。类继承的实现、方法覆盖、super 的使用。</td></tr>
<tr><td>难点：如何调用超类的方法，对象的创建过程。</td></tr>
<tr><td>教学环节
教学方法
及其它说
明事项</td><td>1.采用讲授法、演示法、讨论法。
2.案例+项目式教学。
3.采用线上线下混合教学模式。配套线上课程地址：
www.qingline.net
4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。</td></tr>
<tr><td>导言
或复习</td><td>本章主要学习 Java 语言的封装性、继承性和多态性等面向对象特征，包括包与类库、封装性和访问修饰符、定义类的子类、如何覆盖超类的方法、final 关键字、类的关系、抽象类、对象转换与多态等。

6.1　面向对象特征

为支持面向对象的设计原则，所有 OOP 语言，包括 Java 在内，都有三个特性：封装性、继承性和多态性。

封装（encapsulation）就是把对象的状态（属性）和行为（方法）结合成一个独立的系统单位，并尽可能地隐藏对象的内部细节。例如，一辆汽车就是一个封装体，它封装了汽车的状态和操作。

继承（inheritance）的概念普遍存在于现实世界中。例如，小汽车是一种车，公交车是一种车，自行车也是一种车，它们都具有车的特性，因此它们是车的子类。</td></tr>
</table>



所谓**多态**（polymorphism），是指一个程序中相同的名字表示不同含义的情况。面向对象的程序中的多态有多种情况。在简单的情况下，在同一个类中定义了多个名称相同的方法，即方法重载，另一种情况是子类中定义的与父类中的方法同名的方法，即方法覆盖。这两种情况都称为多态，且前者称为静态多态，后者称为动态多态。

讲授内容

6.2 包与类库

Java 语言使用包来组织类库。**包**（package）实际是一组相关类或接口的集合。Java 类库中的类都是通过包来组织的。

6.2.1 包与 package 语句

用户自定义的类通常也应存放到某个包中，要将某个类放到包中，需在定义类时使用 package 语句指明属于哪个包，如下所示：

```
package com.boda.xy;
public class Account{
  …
}
```

为了保证自己创建的类不与其他人创建的类冲突，需要将类放入包中，这就需要给包取一个独一无二的名称。为了使你的包名与别人的包名不同，建议将域名反转过来，然后中间用点（.）号分隔作为包的名称。因为域名是全球唯一的，以这种方式定义的包名也是全球唯一的。

例如，假设一个域名为 xy.boda.com，那么创建的包名可以为 com.boda.xy。创建的类都存放在这个包下，这些类就不会与任何人的类冲突。

许多 IDE 工具（如 Eclipse 或 IntelliJ IDEA 等）创建带包的类时自动创建包的路径，并将编译后的类放入指定的包中。

如果在命令提示符下使用 javac 编译程序，可以使用带–d 选项的编译命令创建包。

为了方便程序设计和运行，Java 类库中的类都是以包的形式组织的，这些类通常称为 Java API。有关 API 的详细信息请参阅 Java API 文档。

如果一个类属于某个包，可以用类的**完全限定名**（fully qualified name）来表示。例如，Account 类属于 com.boda.xy 包，则 Account 类的完全限定名是 com.boda.xy.Account。

6.2.2 类的导入

在 Java 语言中可以使用两种导入：一是使用 import 语句导入指定包中的类或接口。二是使用 import static 导入类或接口中的静态成员。

	1. import 语句
	import 语句的一般格式为：
	`import package1[.package2[.package3[…]]].类名|*;`
	如果指定具体的类名将导入指定的类，若选用“*”号，表示导入包中所有类。如果一个源程序中要使用某个包中的多个类，用第二种方式比较方便，否则要写多个 import 语句。导入某个包中所有类并不是将所有的类都加到源文件中，而是使用到哪个类才导入哪个类。
	也可以不用 import 语句而在使用某个类时指明该类所属的包。
	`java.util.Scanner sc = new java.util.Scanner(System.in);`
	2. import static 语句
	从前面的例子看到，使用类的静态常量或静态方法，需要在常量名前或方法名前加上类名，如 Math.PI、Math.random()等。这样如果使用的常量或方法较多，代码就显得冗长。因此从 Java 5 开始，允许使用 import static 语句导入类中的常量和静态方法，然后再使用这些类中的常量或方法就不用加类名前缀了。
	例如，要使用 Math 类的 random()等方法，可以先使用下列静态导入语句。之后，在程序中就可以直接使用 random()了。
	`import static java.lang.Math.*;`
	6.2.3 Java 类库
	程序员除了使用自己定义的类外，还可以使用 **Java 标准类库**（Java Class Library，JCL）中定义的类或者第三方定义的类。
	JCL 是 Java 语言实现的包的集合。简单地说，它是 JDK 中的可用.class 文件集合。一旦安装了 Java，它们就将作为安装的一部分，并可以使用 JCL 类作为构建块来构建应用程序代码，这些构建块负责完成许多底层开发。JCL 的丰富性和易用性极大地促进了 Java 的流行。

课堂互动与讨论	**6.3 案例研究：开发自定义类库**
	1. 问题描述
	在 Java 应用开发中通常需要开发自己的类库，然后在应用程序中使用。本案例学习如何开发一个简单的类库，然后将它打包成.jar 文件，并且在程序中使用它。
	要求定义一个名为 com.boda.xy.MathUtils 类，在该类中定义如下两个静态方法：
	`public static boolean isPrime(int n)`
	`public static boolean isPalindrome(int n)`
	isPrime(int n)方法返回 n 是否是素数，isPalindrome(int n)方法返回 n 是否是回文数，如 363 是一个回文数。最后编写程序导入类库，并编写 main()方法，求出 2-1000 之间的所有回文素数。
	2. 设计思路
	对于自定义类库，按下面思路设计。
	（1）按要求编写 com.boda.xy.MathUtils 类，其中定义两个静态方法 isPrime(int n)和 isPalindrome(int n)。

（2）将编译好的类文件打包到.jar 文件中。

（3）在应用程序中添加.jar 文件。

3. 代码实现

程序 6-3 是类库的 com.boda.xy.MathUtils 类，该类定义了两个静态方法，其中 isPrime()用于返回参数是否是素数，isPalindrome()返回参数是否是回文数。

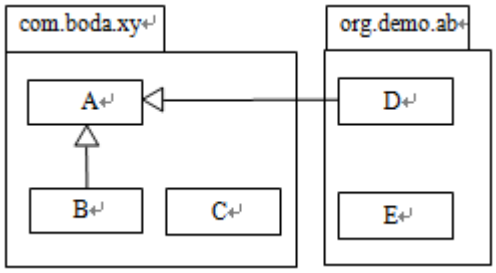
【程序 6-3】MathUtils.java

课堂讲授

6.4 封装性与访问修饰符

封装性是面向对象的一个重要特征。对象的封装是通过两种方式实现的：（1）通过包实现封装性。在定义类时使用 package 语句指定类属于哪个包。（2）通过类和类成员的访问权限实现封装性。

包是 Java 语言最大的封装单位，它定义了程序对类的访问权限。图 6-7 给出了两个包 com.boda.xy 和 org.demo.ab 的结构，其中 A、B、C 类属于 com.boda.xy 包，D 和 E 类属于 org.demo.ab 包。箭头表示类继承关系，其中 B 是 A 的子类，且与 A 在同一个包中，D 也是 A 的子类，但与 A 不在同一个包中。



6.4.1 类的访问权限

类（包括接口和枚举等）的访问权限通过修饰符 public 实现。它定义哪些类可以使用该类。public 类可以被任何其他类使用，而缺省访问修饰符的类仅能被同一包中的类使用。

【案例 6-3】类的访问权限。下面的 Bicycle 类定义在 com.boda.xy 包中，该类缺省访问修饰符。

【程序 6-4】Bicycle.java

```
package com.boda.xy;
class Bicycle{          <-- 这里没有用 public 修饰类
   Bicycle(){
      System.out.println("生产一辆自行车。");
   }
}
```

下面的 BicycleDemo 类定义在 org.demo.ab 包中，它与 Bicycle 类不在同一个包，在该类中试图使用 com.boda.xy 包中的 Bicycle 类。

【程序 6-5】BicycleDemo.java

```
package org.demo.ab;
import com.boda.xy.Bicycle;     <-- 发生编译错误，Bicycle 不可见
public class BicycleDemo {
    public static void main(String[] args){
       var myBike = new Bicycle();
```

```
        }
    }
```

6.4.2 类成员的访问权限

类成员的访问权限包括成员变量和成员方法的访问权限。共有 4 个修饰符，它们分别是 private、缺省的、protected 和 public，这些修饰符控制成员可以在程序的哪些部分被访问，也称为成员的可见性（visibility）。

1. private 访问修饰符

用 private 修饰的成员称为私有成员，私有成员只能被这个类本身访问，外界不能访问。private 修饰符最能体现对象的封装性，从而可以实现信息的隐藏。

2. 缺省访问修饰符

缺省访问修饰符的成员，一般称为包可访问的。这样的成员可以被该类本身和同一个包中的类访问。其他包中的类不能访问这些成员。对于构造方法，如果没有加访问修饰符，也只能被同一个包的类产生实例。

对于案例 6-4 中的 Animal 类，如果将成员变量和方法的修饰符 private 去掉，它们就是包可访问的，程序不会产生编译错误。因为 AnimalTest 类与 Animal 类在同一个包中。

3. protected 访问修饰符

当成员被声明为 protected 时，一般称为保护成员。该类成员可以被这个类本身、同一个包中的类以及该类的子类（包括同一个包以及不同包中的子类）访问。

如果一个类有子类且子类可能处于不同的包中，为了使子类能直接访问父类的成员，那么应该将其声明为保护成员，而不应该声明为私有或默认的成员。

4. public 访问修饰符

用 public 修饰的成员一般称为公共成员，公共成员可以被任何其他的类访问，但前提是类是可访问的。

表 6-2 总结了各种修饰符的访问权限。

修饰符	同一个类	同一个包的类	不同包的子类	任何类
private	允许			
缺省	允许	允许		
protected	允许	允许	允许	
public	允许	允许	允许	允许

6.5 基本概念

将人定义为一个类（Person），因为员工具有人的所有的特征和行为，则可以将员工类（Employee）定义为人的子类，这就叫继承。进一步，还可以将经理类（Manager）再定义为员工类的子类，经理也继承了员工的特征和行为，这样形成类的层次结构。在类的层次结构中，被继承的类称为**父类**（parent class）或**超类**（super class），而继承得到的类称为**子类**（sub class）或**派生类**（derived class）。子类继承父类的状态和行为，同时也可以具有自己的特征。

6.5.1 类继承的实现

实现类的继承，使用 extends 关键字，格式如下：

```
[public] class 子类名 extends 父类名{
    // 类体定义
}
```

这样声明后就说子类继承了超类或者说子类扩展了父类。如果父类又是其他类的子类，则这里的子类就为那个类的间接子类。

【案例 6-5】定义 Person 类的子类 Employee 类。设现有 Person 类，要设计一个 Employee 类。因为 Employee 也是 Person，那么就没有必要从头定义 Employee 类，可以继承 Person 类。因为 Employee 除具有人员 Person 的特征，还有一些自己的特征（如描述员工工资，计算工资的操作等）。

程序 6-7 是 Person 类，

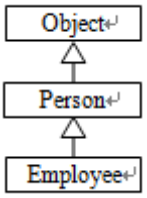
程序 6-8 是 Employee 类表示员工。

关于类继承的几点说明：

（1）子类继承父类中非 private 的成员变量和成员方法。例如，在 Employee 类中可以使用父类中继承来的 name 和 age 属性，还可以调用从父类中继承来的方法，如 sayHello()方法。子类中还可以定义自己的成员变量和成员方法，如 Employee 类中定义了一个表示工资的变量 salary，还定义了 computeSalary()方法。

（2）定义类时若缺省 extends 关键字，则所定义的类为 java.lang.Object 类的直接子类。在 Java 程序中，所有类都是 Object 类的直接或间接子类。如，Person 类就是 Object 类的子类，Person 类也继承了 Object 类中定义的方法。Employee 类、Person 类和 Object 之间的类层次关系如图所示，箭头号表示继承关系。前面定义的所有类也都是 Object 的子类。

（3）Java 仅支持单继承，即一个类至多只有一个直接父类。在 Java 中可以通过接口实现其他语言中的多继承。

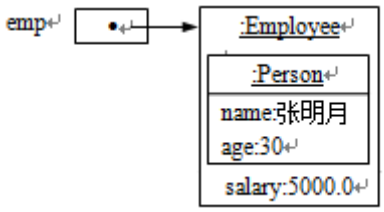


【案例 6-6】创建 Employee 类的实例。定义了子类后，就可以使用其构造方法创建子类对象。

【程序 6-9】EmployeeTest.java

```
package com.boda.xy;
public class EmployeeTest{
   public static void main(String[] args){
    var emp = new Employee("张明月",30,5000);
      System.out.println("姓名 = " + emp.name);
    System.out.println("年龄 = " + emp.age);
    emp.sayHello();       // 调用从父类继承的方法
      System.out.println(emp.computeSalary(10, 50.0)); // 调用子类定义的方法
   }
}
```

这里创建的 Employee 类的对象 emp 的 UML 图如图 6-10 所示。



6.5.2 方法覆盖

在子类中可以定义与父类中的名字、参数列表、返回值类型都相同的方法，这时子类的方法就叫做**覆盖**（overriding）或**重写**了父类的方法。

假设要在 Employee 类中也定义一个 sayHello()方法，它用来输出员工信息，可定义如下：

```
public void sayHello(){
    System.out.println("你好! 我是 " + name);
    System.out.println("我的工资是  " + salary);
}
```

该方法就是对 Person 类的 sayHello()方法的覆盖。如果子类覆盖了超类的方法，再调用相同的方法时，调用的是子类的方法。

为了避免在覆盖方法时写错方法头，可以使用@Override 注解语法，即在要覆盖的方法前面添加@Override 注解。例如，假设一个 Employee 类要覆盖 Object 类的 toString()方法，代码如下：

```
@Override
public String toString(){
    return "姓名: " + name +"年龄: " + age ;
}
```

@Override 注解表示其后的方法必须是覆盖父类的一个方法。如果具有该注解的方法没有覆盖父类的方法，编译器将报告一个错误。例如，toString 如果被错误地写成 tosrting，将报告一个编译错误。如果没有使用注解，编译器不会报告错误。使用注解可以避免错误。

关于方法覆盖，有下面两点值得注意：

（1）父类中 private 方法不能被覆盖。只有非 private 的实例方法才可以覆盖，如果在子类中定义了一个方法在父类中是 private 的，则这两个方法完全无关。

（2）父类中 static 方法也不能被覆盖，但可以被继承。如果子类中定义了与父类中的 static 方法完全一样的方法，那么父类中的方法被隐藏。父类中被隐藏的 static 方法仍然可以使用“类名.方法名()”形式调用。

6.5.3 super 关键字

在子类中可以使用 super 关键字，它用来引用当前对象的父类对象，它可用于下面三种情况。

（1）在子类中调用父类中被覆盖的方法，格式为：

```
    super.方法名([参数列表])
```

（2）在子类中调用父类的构造方法，格式为：

```
    super([参数列表])
```

（3）在子类中访问父类中被隐藏的成员变量，格式为：

```
    super.变量名
```

这里，变量名是要访问的父类中被隐藏的变量名，方法名表示要调用的父类中被覆盖的方法名，参数列表是为方法传递的参数。

6.5.4 调用父类的构造方法

子类不能继承父类的构造方法。要创建子类对象，需要使用默认构造方法或为子类定义构造方法。

1. 子类的构造方法

Java 语言规定，在创建子类对象时，必须先创建该类的所有父类对象。因此，在编写子类的构造方法时，必须保证它能够调用父类的构造方法。

在子类的构造方法中调用父类的构造方法有两种方式：

（1）使用 super 来调用父类的构造方法

```
    super([参数列表]);
```

这里 super 指直接父类的构造方法，参数列表指调用父类带参数的构造方法。不能使用 super 调用间接父类的构造方法，如 super.super()是不合法的。

（2）调用父类的默认构造方法

在子类构造方法中若没有使用 super 调用父类的构造方法，则编译器将在子类的构造方法的第一句自动加上 super()，即调用父类无参数的构造方法。

另外，在子类构造方法中也可以使用 this 调用本类的其他构造方法。不管使用哪种方式调用构造方法，this 和 super 语句必须是构造方法中的第一条语句，并且最多只有一条这样的语句，不能既调用 this，又调用 super。

2. 构造方法的调用过程

在任何情况下，创建一个类的实例时，将会沿着继承链调用所有父类的构造方法，这叫做构造方法链。

【案例 6-7】调用父类的构造方法。该案例定义了 Vehicle 类、Bicycle 类和 ElectricBicycle 类，代码演示了子类和父类构造方法的调用。

【程序 6-10】ElectricBicycle.java

6.6 final 修饰符

final 修饰符可以修饰类、方法、变量以及方法参数。final 的含义是最终的、不能改变的。

6.6.1 final 修饰类

如果一个类使用 final 修饰，则该类就为最终类（final class），最终类不能被继承。下面代码会发生编译错误：

```
final class AA{
    //…
}
class BB extends AA{        ← 不能继承 final 类
    //…
```

```
}
```

定义为 final 的类隐含定义了其中的所有方法都是 final 的。因为类不能被继承，因此也就不能覆盖其中的方法。有时为了安全的考虑，防止类被继承，可以在类的定义时使用 final 修饰符。在 Java 类库中就有一些类声明为 final 类，如 Math 类和 String 类都是 final 类，它们都不能被继承。

6.6.2 final 修饰方法

如果一个方法使用 final 修饰，则该方法不能被子类覆盖。例如，下面的代码会发生编译错误。

```
class AA{
    public final void method(){}
}
class BB extends AA{
    public void method(){}    ← 不能覆盖超类的 final 方法
}
```

6.6.3 final 修饰变量

用 final 修饰的变量包括类的成员变量、方法的局部变量和方法的参数。一个变量如果用 final 修饰，则该变量为常值变量，一旦赋值便不能改变。

对于类的成员变量一般使用 static 与 final 组合定义类常量。这种常量称为编译时常量，编译器可以将该常量值代入任何可能用到它的表达式中，这可以减轻运行时的负担。

如果使用 final 修饰方法的参数，则参数的值在方法体中只能被使用而不能被改变，请看下面代码：

```
class Test{
   public static final int SIZE = 50;
   public void methodA(final int i){
      i = i + 1;    ← 语句产生编译错误，不能改变 i 的值
   }
   public int methodB(final int i){
      final int j = i + 1;    ← 该语句没有错误，可以使用 i 的值
      return j ;
   }
}
```

注意，如果一个引用变量使用 final 修饰，表示该变量的引用（地址）不能被改变，一旦引用被初始化指向一个对象，就无法改变使它指向另一个对象。但对象本身是可以改变的，Java 没有提供任何机制使对象本身保持不变。

总结

课后作业

一、知识点总结

（1）Java 是面向对象的语言，它具有面向对象的所有特征。其中包括封装性、继承性和多态性。

（2）包用来组织类库，包实际是一组相关类或接口的集合。在程序中使用 package 语句定义类属于哪个包。用 import 语句导入包中的类。类库是包的集合，模块是类库

	的集合。 （3）类成员（变量和方法）使用访问权限修饰符 public、protected、缺省、private 等实现封装性。 （4）可以使用 extends 通过现有类定义新类，这称为类的继承。新类称为子类或派生类，现有类称为父类、超类或基类。 （5）子类继承父类中非 private 成员变量和成员方法，子类可以覆盖父类中的实例方法，子类不能继承父类的构造方法。 （6）要覆盖一个方法，必须使用与它父类中的方法相同的签名来定义类中的方法。私有方法不能被覆盖。静态方法不能被覆盖，如果父类中定义的静态方法在子类中重新定义，那么父类中定义的方法被隐藏。 （7）可以使用 super 关键字访问父类的变量、方法和构造方法。如果没有显式地调用父类构造方法，编译器就会把 super()作为构造方法的第一条语句，它调用的是父类的无参构造方法。 （10）使用 final 修饰的类是最终类，不能被继承；使用 final 修饰的方法是最终方法，不能被覆盖；若使用 final 修饰变量，则变量一旦赋值便不能被改变。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2. 定义一个名为 Cuboid 的长方体类，使其继承 Rectangle 类，其中包含一个表示高的 double 型成员变量 height；定义一个构造方法 Cuboid(double length, double width, double height)；再定义一个求长方体体积的 volume()方法。编写程序，求一个长、宽和高分别为 10、5、2 的长方体的体积。

《Java 基础入门》教案

第 12 讲　　抽象类与多态　　2 学时 110 分钟

章节内容	**第 6 章　面向对象特征** 6.8　抽象类 6.9　对象转换 6.9.1　对象转换 6.9.2　instanceof 运算符 6.10　理解多态
教学目标	• 了解抽象类的概念及定义。 • 掌握对象转换。 • 了解 instanceof 运算符。 • 掌握多态的实现。
重点	**重点**：对象转换与替换原则，包括自动转换与强制转换。
难点	**难点**：多态的理解。
教学环节 教学方法 及其它说 明事项	1.采用讲授法、演示法、讨论法。 2.案例+项目式教学。 3.采用线上线下混合教学模式。配套线上课程地址： www.qingline.net 4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。
导言 或复习	前面章节中定义的类可以创建对象，它们都是具体的类。在 Java 中还可以定义抽象类。**抽象类**（abstract class）是包含抽象方法的类。 假设要开发一个图形绘制系统，需要定义圆（Circle）类、矩形（Rectangle）类和三角形（Triangle）类等，这些类都需要定义求周长和面积的方法，这些方法对不同的图形有不同的实现。这时就可以设计一个更一般的类，比如几何形状（Shape）类，在该类中定义求周长和面积的方法。由于 Shape 不是一个具体的形状，这些方法就不能实现，因此要定义为**抽象方法**（abstract method）。

讲授内容	**6.8　抽象类**
	定义抽象方法需要在方法前加上 abstract 修饰符。抽象方法只有方法的声明，没有方法的实现。包含抽象方法的类必须定义为抽象类，定义抽象类需要的类前加上 abstract 修饰符。
	【案例 6-8】抽象类的定义与继承。该案例定义的 Shape 类即为抽象类，其中定义了两个抽象方法。程序 6-12 定义了 Circle 类，它继承了 Shape 类并实现了其中的抽象方法。
	【程序 6-11】Shape.java
	`package com.boda.xy;`
	`public abstract class Shape{`

```
    String name;
    public Shape(){}                                    抽象类可以定义构造方法
    public Shape(String name){
        this.name = name;
    }
    public abstract double getArea();                   两个抽象方法
    public abstract double getPerimeter();
}
```

抽象类不能被实例化，即不能生成抽象类的对象，如下列语句将会产生编译错误：

```
Shape sh = new Shape();    // 抽象类不能实例化
```

【程序 6-12】Circle.java

完整代码见教材。

这里定义的 Circle 类继承了 Shape 抽象类，由于 Circle 类不是抽象类，因此它必须实现抽象类中 getArea()和 getPerimeter()两个方法，此外，它还定义了构造方法和其他普通方法。

6.9 对象转换

为了讨论方便，先介绍两个术语：子类型和父类型。一个类实际上定义了一种类型。子类定义的类型称为子类型，而父类（或接口）定义的类型称为父类型。因此，可以说 Circle 是 Shape 的子类型，Shape 类是 Circle 的父类型。

6.9.1 对象转换

子类对象和父类对象在一定条件下也可以相互转换，这种类型转换一般称为对象转换或**造型**（casting）。对象转换也有自动转换和强制转换之分。

由于子类继承了父类的数据和行为，因此子类对象可以作为父类对象使用，即子类对象可以自动转换为父类对象。可以将子类型的引用赋值给父类型的引用。假设 parent 是一个父类型引用，child 是一个子类型（直接或间接）引用，则下面的赋值语句是合法的：

```
parent = child;     // 子类对象自动转换为父类对象
```

这种转换称为**向上转换**（up casting)。向上转换指的是在类的层次结构图中，位于下方的类（或接口）对象都可以自动转换为位于上方的类（或接口）对象，但这种转换必须是直接或间接类（或接口）。

反过来，也可以将一个父类对象转换成子类对象，这时需要使用强制类型转换。强制类型转换需要使用转换运算符“（）”。

【案例 6-9】对象自动转换和强制转换。

【程序 6-13】CastDemo.java

```
package com.boda.xy;
public class CastDemo{
    public static void main(String[] args){
        var emp = new Employee("张明月",30,5000);
        System.out.println(emp);
        Person p = emp;             // 自动类型转换
        System.out.println(p);
        p.sayHello();
```

```
        System.out.println(p.getClass().getName());
        //System.out.println(p.computeSalary(5,50));  编译错误
        emp = (Employee)p;        // 强制类型转换
        System.out.println(emp.computeSalary(5,50));
    }
}
```

向上转换可以将任何对象转换为继承链中任何一个父类型对象，包括 Object 类的对象，下面语句是合法的。

```
Object obj = new Employee();
```

注意，不是任何情况下都可以进行强制类型转换，请看下面代码：

```
var p = new Person();
var emp = (Employee) p;  // 不能把父类对象强制转换成子类对象
```

6.9.2 instanceof 运算符

instanceof运算符用来测试一个实例是否是某种类型的实例，这里的类型可以是类、抽象类、接口等。instanceof 运算符的格式为：

```
变量名 instanceof 类型名
```

该表达式返回逻辑值。如果变量名引用的变量类型是指定类型名的类型的实例，表达式返回 true，否则返回 false。

如果一个实例是某种类型的实例，那么该实例也是该类型的所有父类型的实例。给定图 6-18 定义的 Fruit 类及子类的层次结构，假设有下面代码：

```
Fruit fruit = new Apple();
Orange orange = new Orange();
```

表达式 fruit instanceof Apple 的结果是 true，表达式 orange instanceof Apple 的结果是 false，表达式 fruit instanceof Object 的结果是 true。

讲授内容

6.10 理解多态

多态（polymorphism）是指由继承而产生的相关的不同的类，其对象对同一消息会做出不同的响应。多态性是指在运行时系统判断应该执行对象哪个方法的代码的能力。

将方法调用与方法体关联起来称方法**绑定**（binding）。若在程序执行前进行绑定，叫前期绑定，如 C 语言的函数调用都是前期绑定。若在程序运行时根据对象的类型进行绑定，则称后期绑定或动态绑定。比如，当调用实例方法时，变量的实际类型在运行时决定使用方法的哪个实现，这称为动态绑定。Java 中除 static 方法和 final 方法外都是动态绑定。

对子类的一个实例，如果子类覆盖了父类的方法，运行时系统调用子类的方法，如果子类继承了父类的方法，则运行时系统调用父类的方法。

有了方法的动态绑定，就可以编写只与基类交互的代码，并且这些代码对所有的子类都可以正确运行。

【案例 6-10】多态和方法动态绑定。假设抽象类 Shape 定义了 getArea()方法，其子类 Circle、Rectangle 都各自实现了 getArea()方法。下面的程序演示了多态概念。

【程序 6-14】PolymorphismDemo.java

完整代码见教材。

知识总结

课后作业

一、知识点总结

（1）抽象类使用 abstract 修饰，抽象方法是只有方法声明，没有方法实现。抽象类不能被实例化，它必须被继承，在非抽象类中抽象方法必须被实现。

（2）因为子类的实例总是它的父类的实例，所以总是可以把一个子类的实例转换成一个父类的变量。当把父类实例转换类子类变量时，必须使用强制类型转换。

（3）可以使用 instanceof 运算符测试一个对象是否是某种类型的一个实例。

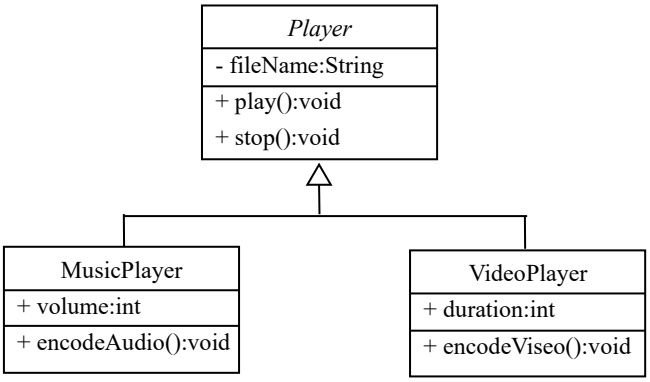
（4）如果一个方法的参数类型是父类（例如，Shape），可以向该方法的参数传递任何子类（例如，Circle）对象，这称为多态。

（5）当调用实例方法时，变量的实际类型在运行时决定使用方法的哪个实现，这称为动态绑定。

二、课后作业

1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/

2 给定如图所示的 Player 类及其子类 MusicPlayer 和 VideoPlayer 的继承关系 UML 图。其中，抽象类 Player 表示播放器，fileName 是播放的文件，play()和 stop()表示播放和停止方法；MusicPlayer 表示音频播放器，volume 表示音量，encodeAudio()表示音频解码；VideoPlayer 表示视频播放器，duration 表示视频持续时间，encodeVideo()表示视频解码。



（1）编写代码实现这些类，为这些类定义无参数构造方法，在构造方法中输出一句话，实现每个类中定义的方法。

（2）在 main()方法中创建一个 MusicPlayer 对象，访问该对象的 volume 成员、play()与 stopt()和 encodeAudio()方法。访问 toString()方法，该方法是在哪里定义的？

（3）在 main()方法中编写下面代码，该段代码是否有编译错误，为什么？

```
Player player;
MusicPlayer mplayer = new MusicPlayer();
mplayer.play();
player = mplayer;
player.encodeAudio();
MusicPlayer mplayer2 = (MusicPlayer)player;
player2.encodeAudio();
```

《Java 基础入门》教案

第 13 讲　　Java 核心类 库　　2 学时 110 分钟

<table>
<tr><td>章节内容</td><td>第 7 章　Java 核心类
7.1　Object 类
7.5　基本类型包装类
7.7　Math 类
7.8　BigInteger 和 BigDecimal 类
7.9　日期-时间 API</td></tr>
<tr><td>教学目标</td><td>• 了解 Java 核心类的使用。
• 掌握 Object 类定义的方法，以及如何覆盖常用方法。
• 掌握基本类型包装类及其使用。
• 了解 Math 类，BigInteger 和 BigDecimal 类。
• 掌握常用日期-时间 API 的使用。</td></tr>
<tr><td rowspan="2">重点
难点</td><td>重点：Object 类、基本类型包装类、日期-时间 API。</td></tr>
<tr><td>难点：BigInteger 和 BigDecimal 类、日期-时间 API。</td></tr>
<tr><td>教学环节
教学方法
及其它说
明事项</td><td>1.采用讲授法、演示法、讨论法。
2.案例+项目式教学。
3.采用线上线下混合教学模式。配套线上课程地址：
www.qingline.net
4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。</td></tr>
<tr><td>导言
或复习</td><td>Java 语言本身定义了许多类，称为 Java 类库（Java Class Library，JCL）。编写程序经常需要使用这些类。
本次课介绍几个最常用的类库。</td></tr>
<tr><td>讲授内容</td><td>
<h3>7.1　Object 类</h3>
java.lang.Object 类是 Java 语言中所有类的根类，定义类时若没有用 extends 指明继承哪个类，编译器自动加上 extends Object。Object 类中共定义了 9 个方法，所有的类（包括数组）都继承该类中的方法。表 7-1（见教材）给出了几个常用方法。
<h4>7.1.1　toString()方法</h4>
toString()方法是 Object 类的一个重要方法，调用对象的 toString()方法可以返回对象的字符串表示。该方法在 Object 类中的定义是返回类名加一个@符号，再加一个十六进制整数。如果在 Account 类中没有覆盖 toString()方法，执行下面的代码：
<pre>
var account = new Account(108, "张明月",5000);
System.out.println(account.toString());
</pre>
</td></tr>
</table>

可能产生类似下面的输出：

```
com.boda.xy.Account@2f92e0f4
```

这些信息没有太大的用途，因此通常在类中覆盖 toString()方法，使它返回一个有意义的字符串。例如，在 Account 类中按如下覆盖 toString()方法：

```
@Override
public String toString(){
    return  "账号:" + id +"  姓名："+ name + "  余额："+ balance;
}
```

这时，语句 System.out.println(account.toString());的输出结果为：

```
账号:108  姓名：张明月  余额：5000.0
```

实际上，还可以仅使用对象名输出对象的字符串表示形式，而不用调用 toString()方法，这时 Java 编译器将自动调用 toString()方法，例如，下面两行等价。

```
System.out.println(account);
System.out.println(account.toString());
```

7.1.2 equals()方法

equals()方法用来比较两个对象是否相等，使用格式为：

```
obj1.equals(obj2)
```

上述表达式用来比较两个对象 obj1 和 obj2 是否相等，若相等则返回 true，否则返回 false。但两个对象比较的是什么呢？首先来看一下在 Object 类中 equals()方法的定义：

```
public boolean equals(Object obj){
    return (this == obj);
}
```

由此可见，该方法比较的是两个对象的引用，即相当于两个对象使用“==”进行比较。假设要比较两个 Account 对象是否相等，如果使用下面代码，将输出 false。

```
var account1 = new Account(108,"张明月",5000);
var account2 = new Account(108,"张明月",5000);
System.out.println(account1.equals(account2));   // false
```

然而，经常需要比较两个对象的内容是否相等，比如对于账户来说，如果两个账户的 id、name 和 balance 属性值都相等，则认为它们相等。要达到这个目的就需要在 Account 类中覆盖 equals()方法。

在 Account 类中可以这样覆盖 equals()方法：

```
@Override
public boolean equals(Object obj){
   if(obj instanceof Account)
      return this.id==(( Account)obj).id &&
             this.name.equals(((Account)obj).name) &&
             this.balance==(( Account)obj).balance;
   else
```

```
        return false;
    }
```

如果在 Account 类中按上面方式覆盖了 equals()方法，再使用该方法比较两个 Account 对象就是比较它们的各属性值是否相等。

7.1.3 hashCode()方法

hashCode()方法返回一个对象的**哈希码**（hash code）值，它是一个整数，主要用来比较对象的大小。在 Object 类中 hashCode()方法的实现是返回对象在计算机内部存储的十进制内存地址。例如代码：

```
var account = new Account(108,"张明月",5000);
System.out.println(account.hashCode());
```

可能的输出结果：798154996

在覆盖 hashCode()方法时，要保证相同对象的哈希码必须相同。可以使用不同算法生成对象的哈希码。可以使用 java.util.Objects 类的 hash()方法直接联合类的每个实例变量的哈希码。

```
@Override
public int hashCode() {
   return Objects.hash(id, name,balance);
}
```

Objects 类的 hash()方法的参数是可变参数，该方法计算每个参数的哈希码，并将它们组合起来。这个方法是空指针安全的。

如果类包含数组类型的实例变量，比较它们的哈希码时，首先使用静态方法 Arrays.hashCode()计算数组的每个元素哈希码组成的哈希码，然后将结果传给 Objects 的 hash()方法。

【案例 7-1】重写 Account 类，覆盖 toString()方法、equals()方法和 hashCode()方法。

【程序 7-1】Account.java

讲授内容

7.5 什么是基本类型包装类?

Java 语言提供了八种基本数据类型，如整型（int）、字符型（char）等。这些数据类型不属于 Java 的对象层次结构。Java 语言保留这些数据类型主要是为了提高效率。这些类型的数据在方法调用时是采用值传递的，不能采用引用传递。

有时需要将基本类型数据作为对象处理，如许多 Java 方法需要对象作参数。因此，Java 为每种基本数据类型提供了一个对应的类，这些类通常称为基本数据类型**包装类**，通过这些类，可以将基本类型的数据包装成对象。

基本数据类型与包装类的对应关系如表 7-3 所示（见教材）。

7.5.1 Character 类

Character 类对象封装了单个字符值。可以使用 Character 类的静态工厂方法创建 Character 对象，其格式为：

```
public static Character valueOf(char c)
```

7.5.2 Boolean 类

	Boolean 类的对象封装了一个布尔值（true 或 false），可以使用 boolean 值或字符串创建 Boolean 实例：
	• public static Boolean valueOf(boolean b)：用一个 boolean 型值创建一个 Boolean 对象。
	• public static Boolean valueOf(String s)：将参数 s 的值转换为 Boolean 对象。
	7.5.3 创建数值类对象
	六种数值型包装类（Byte、Short、Integer、Long、Float 和 Double）都有静态工厂方法 valueOf()和 parseXxx()。一个是以该类型的基本数据类型作为参数，另一个以一个字符串作为参数。
	例如，Integer 类有下面方法：
	`public static Integer valueOf (int value)`
	使用 int 类型的值创建包装类型 Integer 对象。要构造一个包装了 int 型值 314 的 Integer 型对象，可以使用下面方法：
	`var intObj = Integer.valueOf(314);`
	下面代码使用 Double 类的 parseDouble()静态方法将一个字符串转换为 double 型值：
	`var pi = Double.parseDouble("3.14159");`
	7.5.4 什么是自动装箱与自动拆箱?
	为方便基本类型和包装类型之间转换，Java 5 版提供了一种新的功能，称为自动装箱和自动拆箱。自动装箱（autoboxing）是指基本类型的数据可以自动转换为包装类的实例，自动拆箱（unboxing）是指包装类的实例自动转换为基本类型的数据。例如，下面表达式就是自动装箱：
	`Integer value = 300;  // 自动装箱`
	该赋值语句将基本类型数据 300 自动转换为包装类型，然后赋值给包装类的变量 value。下面的语句是自动拆箱：
	`int x = value;  // 自动拆箱`
	它把 Integer 类型的变量 value 中的数值 300 解析出来，然后赋值给基本类型的整型变量 x。

课堂互动与讨论	下面程序中共有 6 处使用了自动装箱或自动拆箱，请指出来。
	`public class UseBoxing {`
	`  boolean go(Integer i){`
	`  Boolean ifSo = true;`
	`  Short s = 300;`
	`  if(ifSo){`
	`  System.out.println(++s);`
	`  }`
	`  return !ifSo;`
	`  }`
	`  public static void main(String []args){`
	`  UseBoxing u = new UseBoxing();`
	`  u.go(5);`
	`  }`

```
    }
```

课堂讲授

7.7 Math 类

java.lang.Math 类定义了一些方法完成基本算术运算，如指数函数、对数函数、平方根函数以及三角函数等。Math 类是 final 类，因此不能被继承。它的构造方法是 private 的，因此不能实例化。Math 类中定义的两个常量 PI 和 E 以及所有的方法都是 static 的，因此仅能通过类名访问。Math 类的常用方法如表 7-4（见教材）所示。

Math 类中的 random()方法用来生成大于等于 0.0 小于 1.0 的 double 型随机数（0<=Math.random()<1.0）。该方法十分有用，可以用它来生成任意范围的随机数。例如：

```
(int)(Math.random() * 10)          // 返回 0 到 9 之间的随机整数
50  + (int)(Math.random() * 51)     // 返回 50 到 100 之间的随机整数
```

一般地，a + (int)(Math.random() * (b+1)) 返回 a 到 a+b 之间的随机数，包括 a+b。

【案例 7-8】随机生成 100 个小写字母。由于小写字母的 ASCII 码值在 97（'a'）和 122（'z'）之间，因此本题只需随机产生 100 个 97 到 122 之间的整数，然后把它们转换成字符即可，代码如下。

【程序 7-13】RandomCharacter.java

```
package com.boda.xy;
public class RandomCharacter {
   public static char getLetter(){
       return (char)(97 + Math.random() * (26));
   }
   public static void main (String[] args) {
      for(var i = 1 ;i <= 100 ; i ++){
         System.out.print(getLetter()+" ");
        if( i % 20 ==0)    // 每输出 20 个字母换行
          System.out.println();
      }
   }
}
```

7.8 BigInteger 和 BigDecimal 类

如果在计算中需要非常大的整数或非常高精度的浮点数，可以使用 java.math 包中定义的 BigInteger 类和 BigDecimal 类。这两个类都扩展了 Number 类并实现了 Comparable 接口，它们的实例都是不可变的。

BigInteger 的实例可以表示任何大小的整数。使用 BigInteger 类构造方法创建大整数对象。

- public BigInteger(byte [] val)：用字节数组的二进制位构成的整数创建 BigInteger 对象。
- public BigInteger(String val)：用包含数字的字符串参数 val 创建 BigInteger。

BigInteger 类定义了 ZERO、ONE、TWO 和 TEN 几个常量，它们分别表示 0、1、2 和 10。

BigDecimal 的实例可以表示任何大小的浮点数。使用 BigDecimal 类构造方法创建 BigDecimal 实例，常用构造方法如下：

- public BigDecimal(char[] in)：使用字符数组字符创建 BigDecimal 对象。

- public BigDecimal(double val)：使用参数 val 值创建 BigDecimal 对象。
- public BigDecimal(String val)：使用字符串创建 BigDecimal 对象。

BigDecimal 类中定义了与 BigInteger 类类似的方法，如 add()、subtract()、multiply()、divide()以及 remainder()等方法执行算术运算，还可以使用 compareTo()方法比较它们的大小。下面代码创建两个 BigInteger 实例然后对它们相乘：

```
var a = new BigInteger("92233720368547758O7");  // long 型最大值
var b = new BigInteger("2");
var c = a.multiply(b);
System.out.println(c);              // 输出 18446744073709551614
```

7.9 日期-时间 API

Java SE 8 开始提供了一个新的日期-时间 API，它们定义在 java.time 包中。常用的类包括 LocalDate、LocalTime、LocalDateTime、YearMonth、MonthDay、Year、Instant、Duration 及 Period 等类。

7.9.1 LocalDate 类

LocalDate 对象用来表示带年月日的日期，它不带时间信息。使用 LocalDate 对象可记录重要的事件，如人的出生日期，产品的出厂日期等。表 7-5（见教材）给出了 LocalDate 类的常用方法。

下面语句创建两个 LocalDate 实例。

```
var today = LocalDate.now();
var birthDay = LocalDate.of(2002, Month.OCTOBER, 20);
```

下面代码使用 plusYears 和 plusDays()方法创建 LocalDate 实例。

```
var newBirthday = birthDay.plusYears(18);
var pday = LocalDate.of(2020,1,1).plusDays(255);    // 2020 年的程序员日
```

LocalDate 类提供了一些访问方法获取日期的有关信息，如 getDayOfWeek()方法可以获得日期中星期，下列代码返回“SUNDAY”。

```
var dofw = LocalDate.of(2018, Month.SEPTEMBER, 2).getDayOfWeek();
```

【案例 7-10】LocalDate 类的使用。
【程序 7-15】DateDemo.java

7.9.2 LocalTime 类

LocalTime 对象表示本地时间，包含时、分和秒，它是不可变对象，最小精度是纳秒。例如，时间“13:45.30.123456789”可以用 LocalTime 对象存储。时间对象中不包含日期和时区。使用下面方法创建 LocalTime 对象。

- public static LocalTime now()：获得默认时区系统时钟的当前时间。

表 7-6（见教材）给出了 LocalTime 类的常用方法。

下面代码创建了两个 LocalTime 对象：

```
var rightNow = LocalTime.now();
```

```
var bedTime = LocalTime.of(22, 30); // 或者 LocalTime.of(22, 30, 0)
```

LocalTime 类 now()方法创建的对象默认带纳秒时间，也可以用 truncatedTo()方法将其截短，不保留纳秒，如下所示。

```
var time = LocalTime.now();
var truncatedTime = time.truncatedTo(ChronoUnit.SECONDS);
```

课堂互动与讨论

7.10 案例研究：打印输出年历

1. 问题描述

打印输出年历是编程经常实现的功能，Java 语言也有多种方法打印年历。本案例要求使用日期-时间 API 的相关类实现年历打印。

2. 设计思路

使用 LocalDate 类打印输出年历的基本思路如下：

（1）年历是按每月打印的。根据输入的年份和要打印的月份，可以创建该月的第一天的 LocalDate 对象，代码如下。

```
var dates = LocalDate.of(year,month,1);
```

（2）根据该月第一天日期可以返回当月的天数 dayOfMonth 和第一天是星期几 dayOfWeek，返回 1 是周一，返回 7 是周日，代码如下。

```
var daysOfMonth = dates.lengthOfMonth();
var dayOfWeek = dates.getDayOfWeek().getValue();
```

（3）有了上面两个值，就可以打印月历。如果 dayOfWeek 值不是 1（星期一），则要输出前导空格，否则不输出空格。然后使用下面代码输出该月的每天日期。

```
for(var i = 1; i <= daysOfMonth;i++){
      System.out.printf("%4d",i);
      if((dayOfWeek + i-1)%7==0)
          System.out.println();     //换行
}
```

3. 代码实现

程序实现代码如下所示，程序运行首先要求用户输入一个年份。

【程序 7-20】PrintCalendar.java

具体代码见教材。

总结

课后作业

一、知识点总结

（1）Object 类是 Java 语言中所有类的根类，定义类时若没有用 extends 指明继承哪个类，编译器自动加上 extends Object。

（2）Object 类中定义了 toString()、equals()等方法，这些方法被子类继承，子类也可以覆盖这些方法。

（3）Java 在 Math 类中提供了数学方法 sin、cos、tan、asin、acos、atan、toRadians、toDegree、exp、log、log10、pow、sqrt、floor、ceil、rint、round、min、max、abs 以及 random，用于执行数学函数。

（4）使用 Math 类的 random()方法可以随机生成 0.0 到 1.0（不包含）之间的一个 double 型数。在此基础上通过编写简单的表达式，生成任意范围的随机数。

	（5）java.lang.System 类是一个系统类，它定义了一些有用的 static 字段和 static 方法，可以帮助你完成常见任务。 （7）每种基本数据类型都有一个对应的包装类型，包装类型提供了常用方法对包装的对象操作。基本数据类型与包装类型可以自动转换，称为自动装箱和自动拆箱。 （8）使用包装类型的 parseXxx()静态方法可以将字符串转换成基本类型值，使用 String 类的 valueOf()静态方法可以把基本类型值转换成字符串。 （9）使用 BigInteger 类和 BigDecimal 类可以对大整数和大浮点数执行有关计算。 （10）使用 LocalDate 类和 LocalTime 类可以对本地日期和本地时间进行操作。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2.编写程序，计算 98765432101234567 89 乘以 98765432101234567 89 的结果。 3.程序员日是每年的第 256 天，编写程序计算 2022 年的程序员日是哪一天？

《Java 基础入门》教案

第 14 讲　　字符串类　　2 学时 110 分钟

章节内容	**第 7 章　Java 核心类** 7.2　String 类 7.3　StringBuilder 类 7.4　案例研究：字符串加密解密
教学目标	• 掌握 String 类的使用和常用方法。 • 掌握 StringBuilder 类的使用和常用方法。 • 了解命令行参数和格式化。
重点 难点	**重点**：String 类的常用方法。 **难点**：字符串拆分与组合。格式化输出。
教学环节 教学方法 及其它说 明事项	1.采用讲授法、演示法、讨论法。 2.案例+项目式教学。 3.采用线上线下混合教学模式。配套线上课程地址： www.qingline.net 4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。
导言 或复习	字符串是字符的序列，它是许多程序设计语言的基本数据结构。Java 语言是通过字符串类实现的。Java 提供了三个字符串类：String 类、StringBuilder 类和 StringBuffer 类。 String 类是不变字符串，StringBuilder 和 StringBuffer 是可变字符串，这三种字符串都是 16 位的 Unicode 字符序列，这三个类都被声明为 final，因此不能被继承。这三个类各有不同的特点，应用于不同场合。

讲授内容	**7.2.1　创建 String 类对象**
	有一种特殊的创建 String 对象的方法，这是直接利用字符串字面值创建字符串对象。
	`var str = "Java is cool";`
	一般使用 String 类的构造方法创建一个字符串对象。String 类有十多个重载的构造方法，可以生成一个空字符串，也可以由字符或字节数组生成字符串。常用的构造方法见教材。
	下面代码说明了使用字符串的构造方法创建字符串对象。
	`char []chars1 = {'A','B','C'};`
	`char []chars2 = {'中','国','π','α','M','N'};`
	`var s1 = new String(chars1);`
	`var s2 = new String(chars2, 0, 4);`

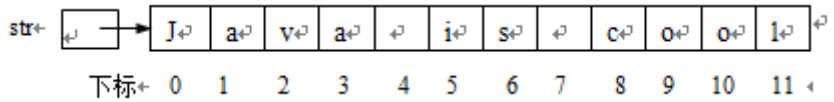
```
System.out.println("s1 = " + s1);    // s1 = ABC
System.out.println("s2 = " + s2);    // s2 = 中国πα
```

7.2.2 字符串基本操作

首先看一下字符串在内存的表示。假设有下面声明：

```
var str = new String("Java is cool");
```

该字符串对象在内存中状态如图 7-3 所示。



该字符串共包含 12 个字符，即长度为 12，其中每个字符都有一个下标，下标从 0 开始，可以通过下标访问每个字符。可以调用 String 类的方法操作字符串。常用方法见教材。

下面代码演示了 String 类的几个方法的使用。

```
var s = "Java is cool";
System.out.println(s.length());           // 12
System.out.println(s.substring(5,7));     // is
System.out.println(s.substring(8));       // cool
var s1 = "Write Once,";
var s2 = "Run Anywhere.";
s1 = s1.concat(s2);
System.out.println(s1);                  // Write Once,Run Anywhere
System.out.println(s.isEmpty());          // false
```

【案例 7-2】要求从键盘输入一个字符串，判断该字符串是否是回文串。一个字符串，如果从前向后读和从后向前读都一样，则称该串为回文串。例如，“mom”和“上海海上”都是回文串。

对于一个字符串，先判断该字符串的第一个字符和最后一个字符是否相等，如果相等，检查第二个字符和倒数第二个字符是否相等。这个过程一直进行，直到出现不相等的情况或者串中所有字符都检测完毕。当字符串有奇数个字符时，中间的字符不用检查。

【程序 7-3】Palindrome.java

完整代码见教材。

7.2.3 字符串查找

String 类常用查找方法如下。

- public int indexOf(int ch)：查找字符 ch 第一次出现的位置。如果查找不成功则返回-1，下述方法相同。

下列代码演示了几个查找方法：

```
var s = new String("This is a Java string.");
System.out.println(s.length());                  // 22
System.out.println(s.indexOf('a'));              // 8
System.out.println(s.lastIndexOf('a',12));       // 11
```

```
System.out.println(s.indexOf("is"));              // 2
System.out.println(s.lastIndexOf("is"));           // 5
System.out.println(s.indexOf("my"));              // -1
```

7.2.4 字符串比较

在 Java 程序中，经常需要比较两个字符串是否相等、或比较两个字符串的大小，下面介绍如何比较字符串。

1. 字符串相等比较

如果要比较两个字符串对象的内容是否相等，可以使用 String 类的 equals()方法或 equalsIgnoreCase()方法。

- public boolean equals(String anotherString)：比较两个字符串内容是否相等。
- public boolean equalsIgnoreCase(String anotherString)：比较两个字符串内容是否相等，不区分大小写。

下面代码演示了这两个方法的使用。

```
var s1 = new String("Hello");
var s2 = new String("Hello");
System.out.println(s1.equals(s2));                    // 输出 true
System.out.println(s1.equals("hello"));               // 输出 false
System.out.println(s1.equalsIgnoreCase("hello"));     // 输出 true
```

实际上 equals()是从 Object 类中继承来的，原来在 Object 类中，该方法比较对象的是对象的引用，而在 String 类中覆盖了该方法，比较的是字符串的内容。

特别注意，不能使用“==”号来比较字符串内容是否相等，请看下面代码。

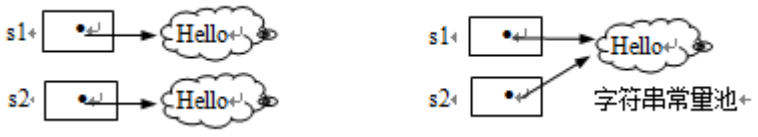
```
var s1 = new String("Hello");
var s2 = new String("Hello");
System.out.println(s1 == s2);     // 输出 false
```

这是因为在使用“==”比较引用类型的数据（对象）时，比较的是引用（地址）是否相等。只有两个引用指向同一个对象时，结果才为 true。上面使用构造方法创建的两个对象是不同的，因此 s1 和 s2 的引用是不同的，如图 7-5(a)所示。

再请看下面一段代码：

```
var s1 = "Hello";           // 不是用构造方法创建的对象
var s2 = "Hello";
System.out.println(s1 == s2);  // 输出 true
```

这次输出结果为 true。这两段代码的不同之处在于创建 s1、s2 对象的代码不同。这里的 s1、s2 是用字符串常量创建的两个对象。字符串常量存储和对象存储不同，字符串常量是存储在常量池中，对内容相同的字符串常量在常量池中只有一个副本，因此 s1 和 s2 是指向同一个对象，如图 7-5(b)所示。



2. 字符串大小比较

使用 equals()方法只能比较字符串的相等与否，要比较大小，可以使用 String 类的 compareTo()方法，格式为：

```
public int compareTo(String another)
```

方法将当前字符串与参数字符串比较，并返回一个整数值。使用字符的 Unicode 码值进行比较。若当前字符串小于参数字符串，方法返回值小于 0，若当前字符串大于参数字符串，方法返回值大于 0，若当前字符串等于参数字符串，方法返回值等于 0。

下面语句输出-2，因为'C'的 Unicode 值比'E'的 Unicode 值小 2。

```
System.out.println("ABC".compareTo("ABE"));
```

如果在字符串比较时忽略大小写，可使用 compareToIgnoreCase(String anotherString)方法。

7.2.6 字符串拆分与组合

使用 String 类的 split()方法可以将一个字符串分解成子字符串或令牌（token），使用 join()方法可以将 String 数组中字符串连接起来，使用 matches()方法返回字符串是否与正则表达式匹配。

- public String[] split(String regex)：参数 regex 表示正则表达式，根据给定的正则表达式将字符串拆分成字符串数组。
- public boolean matches(String regex)：返回字符串是否与给定的正则表达式匹配。
- public static String join(CharSequence delimiter, CharSequence... elements)：使用指定的分割符将 elements 的各元素组合成一个新字符串。

【案例 7-4】字符串的拆分和组合。本案例使用 split()方法拆分字符串，使用 join()方法将一个字符串数组按指定的分割符组合成一个字符串。

【程序 7-5】SplitJoinDemo.java

完整代码见教材。

7.2.8 命令行参数

Java 应用程序从 main()方法开始执行，main()方法的声明格式为：

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

参数 String[] args 称为命令行参数，它是一个字符串数组，该参数是在程序运行时通过命令行传递给 main()方法。

【案例 7-5】命令行参数传递。要求从命令行为程序传递 3 个参数，在 main()方法中通过 args[0]、args[1]、args[2]输出这 3 个参数的值。

【程序 7-6】HelloProgram.java

7.2.9 格式化输出

可以使用 System.out.printf()方法在控制台上显示格式化输出，格式如下：

```
public PrintStream printf(String format, Object... args)
```

参数 format 是格式控制字符串，其中可以嵌入格式符（specifier）指定参数如何输出。args 为输出参数列表，参数可以是基本数据类型，也可以是引用数据类型。

格式符的一般格式如下：

	`%[argument_index$][flags][width][.precision]conversion` 格式符以百分号（%）开头，至少包含一个转义字符，其他为可选内容。其中，argument_index 用来指定哪个参数应用该格式。例如，“%2$”表示列表中的第 2 个参数。flags 用来指定一个选项，如“+”表示数据前面添一个加号，“0”表示数据前面用 0 补充。width 和 precision 分别表示数据所占最少的字符数和小数的位数。conversion 为指定的格式符，表 7-2 列出了常用的格式符（见教材）。

课堂互动与讨论	**7.4 案例研究：字符串加密解密**
	1. 问题描述
	在密码学中，凯撒密码（Caesar cipher）是一种最简单且最广为人知的加密技术。它是一种替换加密技术，明文中的所有字母都在字母表上向后（或向前）按照一个固定数目进行偏移后被替换成密文。这个加密方法是以罗马时期恺撒的名字命名。
	编写程序，实现简单的加密功能：从键盘上输入一个英文字符串（明文），程序运行将该字符串加密后输出（密文）。加密规则为，字符串中的每个字母都用它后面的第 5 个字母代替，其他字符不变，如 A 用 F 代替，a 用 f 代替，对字母表后面的字母，如 V 用 A 代替、Z 用 E 代替。编写解密程序，从键盘输入密文字符串，程序打印输出解密后的明文字符串。
	4. 设计思路
	根据题目要求，使用 Scanner 类的方法从键盘读取字符串，然后创建 StringBulder 对象，在其上取出每个字符，然后对其变换再写入字符串，具体如下。
	（1）对于加密程序，根据取出字符的范围，对字符变换，代码如下：
	`if(c >='A' && c <='Z' || c >'a' && c <='z'){`
	`  if(c >'U' && c <='Z' || c >'u' && c <='z'){`
	`  c = (char)(c - 21);`
	`  }else{`
	`  c = (char)(c + 5);`
	`  }`
	`}`
	（2）对于解密程序，根据取出字符的范围，对字符变换，代码如下：
	`if(c >='A' && c <='Z' || c >'a' && c <='z'){`
	`  if(c >='A' && c <='E' || c >='a' && c <='e'){`
	`  c = (char)(c + 21);`
	`  }else{`
	`  c = (char)(c - 5);`
	`  }`
	`}`
	5. 代码实现
	字符串加密如程序 7-8 所示。注意，这里只对英文字符加密和解密，其他字符不变。
	【程序 7-8】Encryption.java
	字符串解密如程序 7-9 所示。

【程序 7-9】Decryption.java
完整代码见教材 P201。

课堂讲授

7.3 StringBuilder 类

StringBuilder 表示可变字符串，即该类的对象内容是可以修改的。一般来说，只要使用字符串的地方，都可以使用 StringBuilder 类，它比 String 类更灵活。

7.3.1 创建 StringBuilder 对象

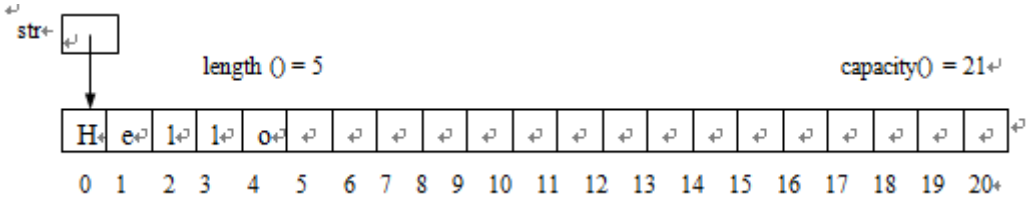
StringBuilder 类是 Java 5 新增加的，它表示可变字符串。下面是 StringBuilder 类常用的构造方法。

- public StringBuilder()：创建一个没有字符的字符串缓冲区，初始容量为 16 个字符。此时 length()方法的值为 0，而 capacity()方法的值为 16。
- public StringBuilder(int capacity)：创建一个没有字符的字符串缓冲区，capacity 为指定的初始容量。
- public StringBuilder(String str)：利用一个已存在的字符串对象 str 创建一个字符串缓冲区对象，另外再分配 16 个字符的缓冲区。

设有下列代码：

```
var str = new StringBuilder("Hello");
```

str 对象在内存中的状态如图 7-9 所示。



在创建 StringBuilder 对象时，系统除为字符串分配空间外，还额外分配 16 个字符的缓冲区。缓冲区主要是为方便 StringBuilder 对象的修改。StringBuilder 对象是可变对象，即修改是在原对象上完成的。如果修改后的长度超过容量，则将容量修改为（原容量+1）的 2 倍。

```
System.out.println(str.length());       // 输出 5
System.out.println(str.capacity());     // 输出 21
```

7.3.2 StringBuilder 的常用方法

StringBuilder 类除定义了 length()、charAt()、indexOf()、getChars()等方法外，还提供了许多方法（见教材 P198 页）。

【案例 7-6】StringBuilder 常用方法的使用。
【程序 7-7】StringBuilderDemo.java
完整程序见教材 P198。

7.3.3 运算符“+”的重载

在 Java 语言中不支持运算符重载，但有一个特例，即“+”运算符（包括+=），它是唯一重载的运算符。该运算符除用于计算两个数之和外还用于连接两个字符串。当用“+”运算符连接的两个操作数其中有一个是 String 类型时，该运算即为字符串连

	接运算，如：
	`int age = 18 ;`
	`var  s = "我今年" + age + "岁。";`
	上述连接运算过程实际上是按如下方式进行的：
	`var s = new StringBuilder("我今年").append(age).append(`
	`"岁。").toString();`

总结与作业	**一、知识点总结** （1）字符串是一个字符序列。字符串字面值使用一对匹配的双引号（"）定界，而字符字面值使用一对匹配的单引号（'）定界。可以用字符串字面值或字符数组创建一个字符串对象。 （2）字符串在 Java 中是对象，通常使用字符串的实例方法操作字符串对象。例如，length()方法返回字符串的长度，charAt()方法返回特定位置的字符。 （3）字符串对象是不可变的，像 replace()、substring()、toLowerCase()等方法都是在操作后创建一个新字符串，原来的字符串不改变。 （4）使用 equals()方法比较字符串是否相等，compareTo()方法比较两个字符串的大小。split()方法拆分字符串，matches()方法实现字符串与正则表达式匹配，join()方法用于按指定分隔符连接字符串。 （5）Java 应用程序的 main()方法带一个字符串数组参数，称为命令行参数。当调用 main()方法时，可以为它传递若干参数，Java 解释器将这些参数存储在 args 参数中。 （6）使用 PrintStream 类的 printf()方法可以对各种类型的数据以指定格式输出。 （7）StringBuilder/StringBuffer 类可以用来替代 String 类。String 类是不可变的，但是可以向 StringBuilder/StringBuffer 对象中添加、插入或追加新的内容。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2.使用下面的方法签名编写一个方法，统计一个字符串中包含字母的个数。 `public static int countLetters(String s)` 编写测试程序调用 countLetters("Beijing 2008")方法并显示它的返回值 7。 3.编写程序，将字符串“no pains,no gains.”解析成含有 4 个单词的字符串数组。

《Java 基础入门》教案

第 15 讲　　接口　　2 学时 110 分钟

章节内容	**第 8 章　接口与内部类** 8.1　接口 8.2　接口方法 8.3　接口示例 8.4　案例研究：比较员工对象大小 2
教学目标	• 了解接口的作用。 • 掌握接口的定义、实现、继承以及接口常量。 • 重点掌握接口的各种方法，包括私有方法、默认方法、静态方法。 • 了解常用接口的使用。
重点 难点	**重点**：掌握接口的各种方法，包括私有方法、默认方法、静态方法。 **难点**：Comparable<T>接口和 Comparator<T>接口的实现。
教学环节 教学方法 及其它说 明事项	1.采用讲授法、演示法、讨论法。 2.案例+项目式教学。 3.采用线上线下混合教学模式。配套线上课程地址： www.qingline.net 4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。
导言 或复习	Java 语言中所有的类都处于一个类层次结构中，除 Object 类以外，所有的类都只有一个直接父类，即子类与父类之间是单继承的关系，而不允许多重继承。 而现实问题类之间的继承关系往往是多继承的关系，为了实现多重继承，Java 语言通过接口使得处于不同层次、甚至互不相关的类具有相同的行为。

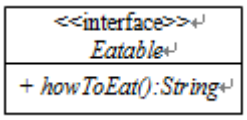
讲授内容	**8.1.1　接口定义**
	接口（interface）定义了一种可以被类层次中的任何类实现的行为的协议。在 Java 8 之前，接口中只能定义常量和抽象方法，从 Java 8 开始，接口中还可以定义默认方法、静态方法和私有方法。接口主要为实现类提供一种操作契约，接口可以用来实现多重继承。
	接口的定义与类的定义类似，包括接口声明和接口体两部分。接口声明使用 interface 关键字，格式如下：
	`[public] interface 接口名 [extends 超接口]{`
	`  // 接口体`
	`}`
	【案例 8-1】一个简单的接口 Eatable（可吃的）。
	【程序 8-1】Eatable.java

```
package com.boda.xy;
public interface Eatable{
    public abstract String howToEat();      ← 抽象方法
}
```

接口中的抽象方法只有声明，没有实现。抽象方法也可以省略 public、abstract 修饰符，省略修饰符编译器自动加上，下面两行代码等价。

```
public abstract String howToEat();
String howToEat();
```

在 UML 中，接口的表示与类图类似，图 8-1 是 Eatable 接口的 UML 图，其中接口名上方使用<<interface>>表示接口，接口名和抽象方法名使用斜体表示。



8.1.2 接口的实现

实现接口就是实现接口中定义的抽象方法，这需要在类声明中用 implements 子句来表示实现接口，一般格式如下：

```
[public] class 类名 implements 接口列表{
    // 类体定义
}
```

一个类可以实现多个接口，这需要在 implements 子句中指定要实现的接口并用逗号分隔。在这种情况下如果把接口理解成特殊的类，那么这个类利用接口实际上实现了多继承。

如果实现接口的类不是 abstract 类，则在类的定义部分必须实现接口中的所有抽象方法，即必须保证非 abstract 类中不能存在 abstract 方法。

【案例 8-2】定义 Mutton 类实现 Eatable 接口。

【程序 8-2】Mutton.java

```
package com.boda.xy;
public class Mutton implements Eatable{
    @Override
    public String howToEat(){      ← 实现接口的抽象方法
       return "烤羊肉串";
    }
 }
```

由于 Mutton 类不是抽象类，它必须实现 Eatable 接口中的 howToEat()方法。为了保证实现的方法是接口中定义的方法，也应该使用@Override 注解。

8.1.3 接口的继承

一个接口可以继承一个或多个接口。与类的继承类似，子接口继承父接口中的常量、抽象方法、默认方法。定义接口的继承仍使用 extends 关键字，一般格式如下：

```
[public] interface 接口名 extends 接口列表{
```

```
    // 接口体定义
}
```

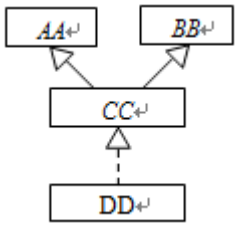
接口的继承是子接口继承父接口中的抽象方法和默认方法。与类的继承不同，一个接口可以继承多个父接口。

【案例 8-3】本案例定义了 AA 接口和 BB 接口，然后定义 CC 接口继承了 AA 接口和 BB 接口。

【程序 8-3】CC.java

完整代码见教材《Java 基础入门-项目案例+微课视频+题库》，沈泽刚，清华大学出版社，2021 年。

上述 AA、BB、CC 接口与 DD 类之间的关系如图 8-3 所示，图中虚线表示接口实现。可以看到，接口允许多继承，而类的继承只能是单继承。



一个类也可以实现多个接口，下面的 AB 类实现了 AA 接口和 BB 接口。

```
package com.boda.xy;
public class AB implements AA, BB{
      public void display(){          ← 实现接口 AA 的 display 方法
        System.out.println("接口 AA 的 display 方法");
    }

    public  void show(){              ← 实现接口 BB 中的 show 方法
         System.out.println("接口 BB 的 show 方法");
      }
}
```

一个类实现多个接口就要实现每个接口中的抽象方法。接口中的常量和默认方法都被实现类继承，但接口中的静态方法不被继承。

8.1.4 接口类型的使用

接口也是一种引用类型，任何实现该接口的实例都可以存储在该接口类型的变量中。当通过接口对象调用某个方法时，Java 运行时系统确定该调用哪个类中的方法。对上一小节中定义的 AA、BB、CC 接口和 DD 类，下面代码是合法的。

```
AA aa = new DD();          // 向上自动类型转换
BB bb = new DD();
CC cc = new DD();
aa.display();               // 调用实现类的方法
bb.show();
cc.print();                 // 调用继承的默认方法
```

8.1.5 常量

定义在接口中的任何变量都自动加上 public、final、static 属性，因此它们都是常

量，常量的定义可以省略修饰符，下面三行代码效果相同。

```
int STATUS = 100;
public int STATUS = 100;
public final static int STATUS = 100;
```

按照 Java 标识符命名惯例，常量名都使用大写字母命名。接口中的常量应该使用接口名引用。不推荐在接口中定义常量，因为使用枚举类型描述一组常量集合比接口中定义常量更好。关于枚举类型的定义和使用请参考 9.2 节。

由于接口可以多继承，可能出现常量冲突问题。如果常量名不冲突，子接口可以继承父接口的常量。如果多个父接口中有同名的常量，则子接口中不能继承。但子接口可以重新定义一个同名的常量。

讲授内容

8.2 接口方法

在 Java 的早期版本中，接口的所有方法都必须是抽象的。从 Java 8 开始可以在接口中添加两种有具体实现的方法：静态方法和默认方法。在 Java 9 中，还可以定义私有（private）方法。

8.2.1 默认方法

可以给接口中任何方法提供一个默认实现，这称为默认方法（default method）。默认方法需要使用 default 关键字定义。

```
public interface BB {
   public void show();           // 一个抽象方法
   public default void print(){      <-- 一个默认方法，有方法体
      System.out.println("这是 BB 接口的默认方法");
   }
}
```

默认方法需要通过引用变量调用。默认方法可以被子接口和实现类继承，但子接口中若定义相同的默认方法，父接口的默认方法被隐藏。

8.2.2 私有方法

在 Java 9 中，除了静态方法和默认方法外，还可以在接口中定义私有方法。私有方法通常实现某种行为，这些行为可以被默认方法调用。

```
public interface MyInterface {
    void normalMethod();   // 抽象方法
    private void init() {                <-- 私有方法实现某种操作
       System.out.println("完成某些初始化操作");
    }
    default void defaultMethodA(){
       init();          <-- 在默认方法中调用私有方法
     }
    default void defaultMethodB() {
       init();          <-- 在默认方法中调用私有方法
    }
```

```
}
```

如果使用默认方法开发 API，那么接口私有方法可能有助于实现其部分功能。

8.2.3 静态方法

在一个类中可以定义静态方法，它被该类的所有实例共享。在 Java 8 中，可以在接口中定义静态方法，与接口有关的静态方法都可以在接口中定义，而不再需要辅助类。定义静态方法使用 static 关键字，默认的访问修饰符是 public。

```
public interface SS {
   int STATUS = 100;
   public static void display(){   <-- 静态方法也是实现的方法
      System.out.println(STATUS);
   }
}
```

接口的静态方法使用“接口名.方法名()”的形式访问。接口的静态方法不能被子接口继承，也不被实现类继承。静态方法在哪个接口中定义，就用哪个接口名访问。

8.2.4 关于接口与抽象类

在 Java 8 之前的时代，在抽象类和接口之间做出选择比现在容易。在那时，如果需要在扩展/实现类之间共享一些实现，就选择抽象类。否则，就使用接口。今天，也可以将实现放在接口中，因此创建抽象类的动机肯定会减少。然而，仍然有一些理由使用抽象类：

- 在抽象类中可以添加最终（final）方法，但在接口中不能。如果想阻止一个方法被覆盖，这一点非常重要。
- 在抽象类中，可以使用字段保存对象的状态。相反，接口中的字段是 public static 的，这意味着它们的值在实现类的所有实例之间共享。
- 在抽象类中可以定义构造方法，接口中不能。

课堂互动与讨论

8.3 接口示例

Java 类库中也定义了许多接口，有些接口中没有定义任何方法，这些接口称为标识接口，如 java.lang 包中定义的 Cloneable 接口、java.io 包中的 Serializable 接口。有些接口中定义了若干方法，如 java.lang 包中的 Comparable<T>接口中定义了 comapreTo()方法，AutoClosable 接口定义了 close()方法，Runnable 接口中定义了 run()方法。下面介绍两个用于对象比较的接口：Comparable 接口和 Comparator 接口。

8.3.1 Comparable<T>接口

我们知道，要比较两个 String 对象的大小，可以使用它的 compareTo()方法。现在假设要比较两个 Circle 对象的大小，该如何比较呢？Circle 类是用户定义的类，它无法比较大小。要想比较 Circle 对象的大小，如按面积比较，需要实现 Comparable<T>接口的 compareTo()方法。Comparable<T>接口的定义如下：

```
package java.lang;
public interface Comparable<T>{
   int compareTo(T other);
```

```
}
```

Comparable<T>是泛型接口。在实现该接口时，将泛型类型 T 替换成一种具体的类型。如果希望一个类的对象能够比较大小，类必须实现 Comparable<T>接口的 compareTo()方法。该方法实现当前对象与参数对象比较，返回一个整数值。当调用对象小于、等于、大于参数对象时，该方法分别返回负整数、0 和正整数。按这种方法比较出的对象顺序称为**自然顺序**（natural order）。

【案例 8-6】下面案例说明了如何通过实现 Comparable<T>接口对 Circle 类的对象根据其面积大小进行比较。

【程序 8-6】Circle.java

完整代码参见《Java 基础入门-项目案例+微课视频+题库》，沈泽刚，清华大学出版社，2021 年。

8.3.2 Comparator<T>接口

假设需要根据长度而不是字典顺序对字符串排序，我们可以使用 Arrays 类的带两个参数的 sort()方法，格式如下：

```
public static <T>void sort(T[] a, Comparator<? super T>c)
```

第一个参数 a 是任意类型的数组，第二个参数 c 是一个实现了 java.util.Comparator 接口的实例，它称为比较器对象。Comparator<T>接口中声明了 compare()抽象方法，如下所示。

```
public interface Comparator<T> {
    int compare(T first, T second);
    // 其他静态方法和默认方法
}
```

compare()方法用来比较它的两个参数对象。当第一个参数小于、等于、大于第二个参数时，该方法分别返回负整数、0、正整数。

【案例 8-7】按字符串的长度比较大小，长度小的排在前面。这可以定义一个比较器对象，也就是实现 Comparator<String>接口的类。

【程序 8-7】LengthComparator.java

完整代码参见《Java 基础入门-项目案例+微课视频+题库》，沈泽刚，清华大学出版社，2021 年。

总结

课后作业

一、知识点总结

（1）接口是一种与类相似的结构，它为相关或不相关的类的多个对象指定共同行为。使用 interface 定义接口。

（2）在接口中可以定义常量、抽象方法、静态方法、默认方法和私有方法。

（3）在 Java 中接口被认为是一种特殊的类型。就像常规类一样，每个接口都被编译成独立的字节码文件。

（4）一个接口可以继承一个或多个接口。类实现接口就是实现接口中定义的抽象方法。一个类可以实现一个或多个接口。

（5）接口中定义的默认方法可以被子接口或实现类继承，静态方法不能被继承，静态方法使用接口名调用。

（6）java.lang.Comparable<T>接口定义了 compareTo()方法用来比较对象，Java 类库中很多类实现了 Comparable<T>接口。

（7）java.util.Comparator<T>接口定义了 compare()方法用来实现比较器对象，可

	以将它传递给有关方法实现非自然顺序的比较。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2.编写程序，定义一个名为 Swimmable（可游泳的）的接口，其中包含 void swim()方法，定义另一个名为 Flyable（可飞的）的接口，其中包含 void fly()方法。 3.定义一个名为 Animal 的抽象类，其中包含一个 void eat()的抽象方法。定义一个名为 WildDuck（野鸭）的类实现上述两个接口，并继承 Animal 类。 4.编写 main()方法，在其中创建一个 WildDuck 实例，调用它的各种方法。将 WildDuck 实例分别赋值给一个 Swimmable 引用、Flyable 引用和 Animal 引用，测试通过这些引用能否调用相应接口或抽象类中定义的方法。

《Java 基础入门》教案

第 16 讲　　内部类　　2 学时 110 分钟

章节内容	**第 8 章　接口与内部类** 8.7　内部类 8.7.1　成员内部类 8.7.2　静态内部类 8.7.3　匿名内部类 8.7.4　局部内部类
教学目标	• 了解什么是内部类。 • 掌握成员内部类、静态内部类和匿名内部类的定义。 • 了解局部内部类定义。
重点 难点	**重点**：成员内部类、静态内部类和匿名内部类定义和使用。 **难点**：匿名内部类的定义和使用。
教学环节 教学方法 及其它说 明事项	1.采用讲授法、演示法、讨论法。 2.案例+项目式教学。 3.采用线上线下混合教学模式。配套线上课程地址： www.qingline.net 4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。
导言 或复习	Java 允许在一个类的内部定义另一个类（接口、枚举或注解），这种类称为**内部类**（inner class）或**嵌套类**（nested class）。有多种类型的内部类。大致可分为成员内部类、静态内部类、匿名内部类和局部内部类。下面分别讨论这几种内部类的定义和使用。
讲授内容	**8.7.1　成员内部类** 成员内部类是没有用 static 修饰且定义在外层类的类体中。在成员内部类中可以定义自己的成员变量和方法，也可以定义自己的构造方法。成员内部类的访问修饰符可以是 private、public、protected 或缺省。成员内部类可以看成是外层类的一个成员，因此可以访问外层类的所有成员（包括私有成员）。 【案例 8-13】下面案例在 Outer 类中定义了一个成员内部类 Inner。 【程序 8-18】Outer.java **完整代码见教材** P253。 程序中 Inner 是 Outer 的成员内部类。内部类编译后将单独生成一个类文件，如上述代码编译后将生成两个类文件：Outer.class 和 Outer$Inner.class。 在外层类的方法中（如 makeInner）可以直接创建内部类的实例。在外层类的外面要创建内部类的实例必须先创建一个外层类的实例，因为内部类对象对外层类对象有一个隐含的引用。 创建内部类对象也可以使用下面的语句实现：

```
var inner = new Outer().new Inner();
```

在使用成员内部类时需要注意下面几个问题：

- 成员内部类中不能定义 static 变量和 static 方法。
- 成员内部类也可以使用 abstract 和 final 修饰，其含义与其他类一样。
- 成员内部类还可以使用 private、public、protected 或包可访问修饰符。

8.7.2 静态内部类

静态内部类使用 static 修饰，静态内部类也称**嵌套类**（nested class），静态内部类与成员内部类的行为完全不同，下面是它们的不同之处：

- 静态内部类中可以定义静态成员，而成员内部类不能。
- 静态内部类只能访问外层类的静态成员。成员内部类可以访问外层类的实例成员和静态成员。
- 创建静态内部类的实例不需要先创建一个外层类的实例。相反，创建成员内部类实例，必须先创建一个外层类的实例。

【案例 8-14】静态内部类的使用。

【程序 8-19】Outer.java

完整代码见教材 P254。

静态内部类实际是一种外部类，它不存在对外部类的引用，不通过外部类的实例就可以创建一个对象。程序中 Outer.Inner 就是静态内部类的完整名称，此时必须使用完整的类名（如 Outer.Inner）创建对象。

静态内部类不具有任何对外层类实例的引用，因此静态内部类中的方法不能使用 this 关键字访问外层类的实例成员，然而这些方法可以访问外层类的 static 成员。这一点与一般类的 static 方法的规则相同。

8.7.3 匿名内部类

定义类最终目的是创建一个类的实例，但如果某个类的实例只使用一次，可以将类的定义和实例的创建在一起完成，或者说在定义类的同时就创建一个实例。以这种方式定义的没有名字的类称为**匿名内部类**（anonymous inner class）。

声明和构建匿名内部类的一般格式如下：

```
new TypeName(){
    /* 此处为类体 */
}
```

匿名内部类可以继承一个类或实现一个接口，这里 TypeName 是匿名内部类所继承的类或实现的接口。如果实现一个接口，该类是 Object 类的直接子类。匿名类继承类或实现接口不需要使用 extends 或 implements 关键字。匿名内部类不能同时继承一个类和实现一个接口，也不能实现多个接口。

由于匿名内部类没有名称，所以类体中不能定义构造方法。由于不知道类名，所以只能在定义类的同时用 new 关键字创建类的实例。实际上，匿名内部类的定义、创建对象发生在同一个地方。

另外，上式是一个表达式，它返回一个对象的引用，所以可以直接使用或将其赋给一个引用变量。

```
TypeName obj = new TypeName(){
    /* 此处为类体 */
};
```

也可以将构建的匿名类对象作为方法的参数。

```
someMethod(new TypeName() {
             /* 此处为类体 */
           });
```

【案例 8-15】下面案例中的匿名内部类实现一个 Printable 接口。

【程序 8-20】PrintableTest.java

```
package com.boda.xy;
interface Printable{
     public void print(String message);
}
public class PrintableTest{
   public static void main(String[]args){
        var printer = new Printable(){          ← 创建一个匿名内部类实例，该类实现了 Printable 接口
            @Override
          public void print(String message){
                System.out.println(message);
            }
        };                                      ← 这里的分号是赋值语句的结束
        printer.print("这是惠普打印机");
   }
}
```

运行结果如图 8-18 所示。

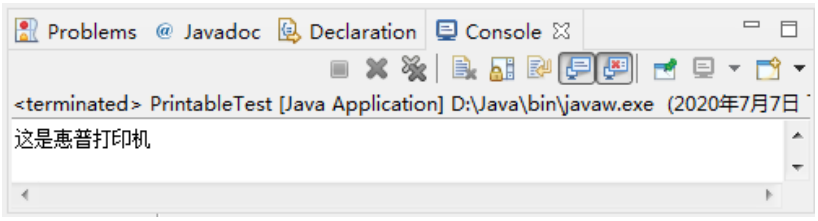


图 8-18　案例运行结果

Printable 是一个接口，其中声明了一个 print()抽象方法。在 PrintableTest 类的 main()方法中声明了一个 Printable 接口变量，然后用 new Printable()创建一个实现该接口的对象。

【案例 8-16】下面案例通过继承 Animal 类并覆盖它的 eat()方法定义一个匿名内部类并创建一个对象。

【程序 8-21】AnimalTest.java

```
package com.boda.xy;
class Animal{
     public void eat(){
       System.out.println("I like eat anything.");
```

```
    }
}
public class AnimalTest{
   public static void main(String[]args){
        var dog = new Animal(){      // 继承 Animal 类
            @Override
          public void eat(){
                System.out.println("I like eat bones.");
            }
        };          这里的分号是赋值语句的结束
        dog.eat();
   }
}
```

运行结果如图 8-19 所示。

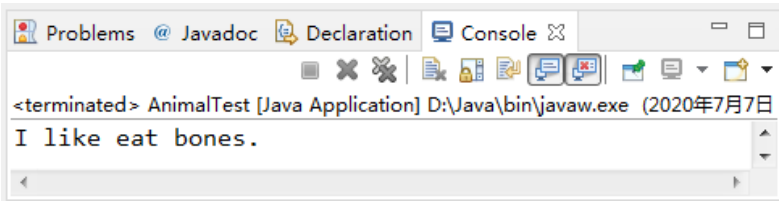


图 8-19　案例运行结果

这里创建的匿名内部类实际上继承了 Animal 类，是 Animal 类的子类，并覆盖了 Animal 类的 eat()方法。同时代码创建一个匿名类的实例，并用 dog 指向它。

匿名内部类的一个重要应用是编写 Java 图形界面的事件处理程序。如为按钮对象 button 注册事件处理器，就可以使用匿名内部类。关于 Java 图形界面的事件处理请参阅其他教材。

8.7.4　局部内部类

可以在方法体或语句块内定义类。在方法体或语句块 （包括方法、构造方法、局部块、初始化块或静态初始化块) 内部定义的类称为**局部内部类**（local inner class）。

局部内部类不能看作外部类的成员，它只对局部块有效，如同局部变量一样，在说明它的块之外完全不能访问，因此也不能有任何访问修饰符。下面案例演示了局部内部类的定义。

【案例 8-17】在方法中定义局部内部类。

【程序 8-22】Outer.java

完整代码见教材 P257。

使用局部内部类时要注意下面问题：

- 局部内部类同方法局部变量一样，不能使用 private、protected 和 public 等访问修饰符，也不能使用 static 修饰，但可以使用 final 或 abstract 修饰。
- 局部内部类可以访问外层类的成员，若要访问其所在方法的参数和局部变量，这些参数和局部变量不能修改。
    - static 方法中定义的局部内部类，可以访问外层类定义的 static 成员，不能访问外层类的实例成员。

总结与作业	**一、知识点总结** （1）成员内部类是没有用 static 修饰且定义在外层类的类体中。 （2）在外层类的外面要创建内部类的实例必须先创建一个外层类的实例，因为内部类对象对外层类对象有一个隐含的引用。 （3）静态内部类使用 static 修饰，静态内部类也称**嵌套类**（nested class）。 （4）创建静态内部类的实例不需要先创建一个外层类的实例。 （5）可以将类的定义和实例的创建在一起完成，或者说在定义类的同时就创建一个实例。以这种方式定义的没有名字的类称为匿名内部类。匿名内部类可以继承一个类或实现一个接口。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2 定义一个类，类中包含私有数据成员和私有方法。在这个类中定义一个内部类，内部类中定义一个方法修改外部类的数据成员值，并调用外部类的私有方法。在外部类的公共静态方法中创建内部类对象，并调用内部类的方法。

《Java 基础入门》教案

第 17 讲　　枚举类型和注解类型　　2 学时 110 分钟

<table>
<tr><td>章节内容</td><td>第 8 章　接口与内部类
8.5　枚举类型
8.5.1　枚举的定义和使用
8.5.2　在 switch 中使用枚举
8.6　注解类型
8.6.1　注解概述
8.6.2　标准注解</td></tr>
<tr><td>教学目标</td><td>• 掌握枚举类型的定义和使用。
• 了解注解类型的使用。</td></tr>
<tr><td>重点</td><td>重点：枚举类型。</td></tr>
<tr><td>难点</td><td>难点：注解类型。</td></tr>
<tr><td>教学环节
教学方法
及其它说
明事项</td><td>1.采用讲授法、演示法、讨论法。
2.案例+项目式教学。
3.采用线上线下混合教学模式。配套线上课程地址：
www.qingline.net
4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。</td></tr>
<tr><td>导言
或复习</td><td>在实际应用中，有些数据的取值被限定在几个确定的值之内。例如，一年有 4 个季度，一周有 7 天、一副纸牌有 4 种花色等。对这种类型的数据，以前通常在类或接口中定义常量实现。
Java 5 中增加了枚举类型，这种类型的数据可以定义为枚举类型。</td></tr>
<tr><td>讲授内容</td><td>8.5.1　枚举的定义和使用

枚举类型是一种特殊的引用类型，它的声明和使用与类和接口有类似的地方。它可以作为顶层的类型声明，也可以像内部类一样在其他类的内部声明。

【案例 8-8】枚举类型的定义和使用。下面程序定义了一个名为 Direction 的枚举类型，它表示 4 个方向。
【程序 8-11】Direction.java
<pre>package com.boda.xy;
public enum Direction{
    EAST, SOUTH, WEST, NORTH;
}</pre>枚举类型的声明使用 enum 关键字，Direction 为枚举类型名，其中声明了 4 个常量，分别表示 4 个方向。由于枚举类型的实例是常量，因此按照命名惯例它们都用大写字母表示。上面的程序经过编译后产生一个 Direction.class 类文件。</td></tr>
</table>

为了使用枚举类型，需要创建一个该类型的引用，并将某个枚举实例赋值给它。下面代码可以输出每个枚举常量名和它们的顺序号。

【程序 8-12】EnumDemo.java

```
package com.boda.xy;
public class EnumDemo {
   public static void main(String[] args){
      // 声明一个枚举类型变量，并用一个枚举赋值
      var left = Direction.WEST;
      System.out.println(left);   // 输出 WEST
      // 输出每个枚举对象的序号
      for(var d : Direction.values()){
          System.out.println(d.name()+",序号"+d.ordinal());
      }
   }
}
```

8.5.2 在 switch 中使用枚举

枚举类型有一个特别实用的特性，它可以在 switch 语句中使用。java.time.DayOfWeek 是一个枚举类型，其中包括一周的 7 天，分别为 MONDAY、TUESDAY、WEDNESDAY、THURSDAY、FRIDAY、SATURDAY 和 SUNDAY，序号从 0 到 6。

【案例 8-9】下面案例在 switch 结构中使用 DayOfWeek 枚举。

【程序 8-13】EnumSwitch.java

```
package com.boda.xy;
import java.time.DayOfWeek;
public class EnumSwitch{
    public static void describe (DayOfWeek day) {
       switch (day) {
            case MONDAY ->
              System.out.println("Mondays are bad.");
            case FRIDAY ->
              System.out.println("Fridays are better.");
          case SATURDAY,SUNDAY ->
             System.out.println("Weekends are best.");
            default ->
             System.out.println("Midweek days are so-so.");
        }
       }

     public static void main(String[] args) {
       var firstDay = DayOfWeek.MONDAY;
        describe (firstDay);
        var thirdDay = DayOfWeek.WEDNESDAY;
        describe (thirdDay);
        var seventhDay = DayOfWeek.SUNDAY;
```

```
            describe(seventhDay);
        }
    }
```

8.6 注解类型

注解类型（annotation type）以结构化的方式为程序元素提供信息，这些信息能够被外部工具（编译器、解释器等）自动处理。

注解有许多用途，其中包括：

- 为编译器提供信息。编译器可以使用注解检测错误或阻止编译警告。
- 编译时或部署时处理。软件工具可以处理注解信息生成代码、XML 文件等。
- 运行时处理。有些注解在运行时可以被检查。

像使用类一样，要使用注解必须先定义注解类型（也可以使用语言本身提供的注解类型）。

8.6.1 注解概述

注解是为 Java 源程序添加的说明信息，这些信息可以被编译器等工具使用。可以给 Java 包、类型（类、接口、枚举）、构造方法、方法、成员变量、参数及局部变量进行标注。例如，可以给一个 Java 类进行标注，以便阻止 javac 程序可能发出的任何警告，也可以对一个想要覆盖的方法进行标注，让编译器知道你是要覆盖这个方法而不是重载它。

1. 注解和注解类型

学习注解会经常用到下面两个术语：注解（annotation）和注解类型（annotation type）。注解类型是一种特殊的接口类型，注解是注解类型的一个实例。就像接口一样，注解类型也有名称和成员。注解中包含的信息采用“键/值”对的形式，可以有零或多个“键/值”对，并且每个键有一个特定类型。它可以是一个 Stirng、int 或其他 Java 类型。没有“键/值”对的注解类型称作标记注解类型（marker annotation type）。如果注解只需要一个“键/值”对，则称为单值注解类型。

2. 注解语法

在 Java 程序中为程序元素指定注解的语法如下：

```
@AnnotationType
```

或者

```
@AnnotationType(elementValuePairs)
```

8.6.2 标准注解

注解的功能很强大，但程序员很少需要定义自己的注解类型。大多数情况下是使用语言本身定义的注解类型。下面介绍几个 Java API 中定义的注解类型。

Java 语言规范中定义了 3 个注解类型，它们是供编译器使用的。这 3 个注解类型定义在 java.lang 包中，分别为@Override、@Deprecated 和@SuppressWarnings。

1. Override

Override 是一个标记注解类型，可以用在一个方法的声明中，它告诉编译器这个方法要覆盖父类中的某个方法。使用该注解可以防止程序员在覆盖某个方法时出错。

例如，考虑下面的 Parent 类：

```
class Parent{
   public double calculate(double x,double y){
       return x * y;
   }
 }
```

假设现在要扩展 Parent 类，并覆盖它的 calculate()方法。下面是 Parent 类的一个子类：

```
class Child extends Parent{
   public int calculate(int x,int y){
       return (x + 1) * y;
   }
}
```

Child 类可以编译。然而，Child 类中的 calculate()方法并没有覆盖 Parent 中的方法，因为它的参数是 2 个 int 型，而不是 2 个 double 型。使用 Override 注解就可以很容易防止这类错误。每当你想要覆盖一个方法时，就在这个方法前声明 Override 注解类型：

```
class Child extends Parent{
   @Override
   public int calculate(int x,int y){
       return (x + 1) * y;
   }
 }
```

这样，如果要覆盖的方法不是父类中的方法，编译器会产生一个编译错误，并指出 Child 类中的 calculate()方法并没有覆盖父类中的方法。如图 8-13 所示。

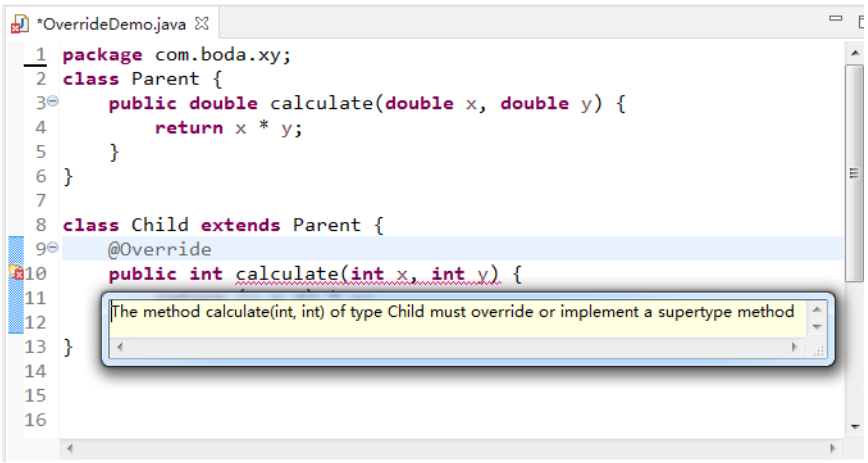
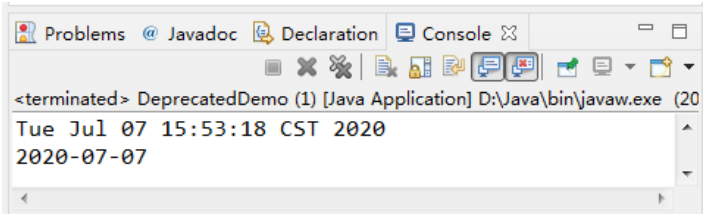


图 8-13　代码运行结果

2. Deprecated

Deprecated 是一个标记注解类型，可以应用于某个方法或某个类型，指明方法或类型已被弃用。标记已被弃用的方法或类型，是为了警告其代码用户，不应该使用或者覆盖该方法，或者不该使用或扩展该类型。一个方法或类型被标记弃用通常是因为有了更好的方法或类型。当前的软件版本中保留这个被弃用的方法或类型是为了向后

	兼容。 【案例 8-11】Deprecated 注解的使用。在 Eclipse 中，标注 Deprecated 注解的方法被加上删除线，使用弃用方法的代码也被加上删除线。 【程序 8-15】DeprecatedDemo.java 完整代码见教材《Java 基础入门-项目案例+微课视频+题库》，沈泽刚，清华大学出版社，2021 年。 编译该文件，编译器将发出警告。在 Eclipse 中，废弃的方法加删除线。运行结果如图 8-14 所示。  图 8-14　案例运行结果 3.　SuppressWarnings 使用 SuppressWarnings 注解指示编译器阻止某些类型的警告，具体的警告类型可以用初始化该注解的字符串来定义。该注解可应用于类型、构造方法、方法、成员变量、参数以及局部变量。它的用法是传递一个 String 数组，其中包含需要阻止的警告。语法如下： `SuppressWarnings（value={string-1,…,string-n}）`
课堂互动与讨论	编写程序，定义一个名为 TrafficLight 的 enum 类型，它包含三个常量：GREEN、RED 和 YELLOW 表示交通灯的三种颜色。通过 values()方法和 ordinal()方法循环并打印每一个值及其顺序值。编写一个 switch 语句，为 TrafficLight 的每个常量输出有关信息。
总结与作业	**一、知识点总结** （1）枚举类型是一种特殊的引用类型，枚举类型的声明使用 enum 关键字。 （2）注解类型是一种特殊的接口类型，注解是注解类型的一个实例。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2 ..定义一个名为 Enhancement 的注解类型，包含 id、synopsis、engineer 和 date 等 4 个元素，为 engineer 和 date 分别指定默认值"unsigned"和"unknown"。

《Java 基础入门》教案

第 18 讲　　异常处理　　2 学时 110 分钟

<table>
<tr><td>章节内容</td><td>第 9 章　异常处理
9.1　异常与异常类
9.1.1　异常的概念
9.1.2　异常类型
9.2　用 try-catch 捕获异常
9.3　捕获多个异常
9.4　throws 和 throw 关键字
9.5　try-with-resources 语句
9.6　自定义异常类</td></tr>
<tr><td>教学目标</td><td>• 了解异常类和异常类型。
• 掌握异常处理的各种方法。
• 了解自定义异常。</td></tr>
<tr><td>重点</td><td>重点：异常处理的各种方法。</td></tr>
<tr><td>难点</td><td>难点：throw 和 throws 的使用以及 try-with-resources 语句。</td></tr>
<tr><td>教学环节
教学方法
及其它说
明事项</td><td>1. 采用讲授法、演示法、讨论法。
2. 案例+项目式教学。
3. 采用线上线下混合教学模式。配套线上课程地址：
www.qingline.net
4. 软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。</td></tr>
<tr><td>导言
或复习</td><td>所谓异常（exception）是在程序运行过程中产生的使程序终止正常运行的错误对象。如数组下标越界、整数除法中零作除数、文件找不到等都可能使程序终止运行。
为了理解异常的概念，首先看下面的案例。
【案例 9-1】数组下标越界异常。
【程序 9-1】ArrayExceptionDemo.java
<pre>
package com.boda.xy;
public class ArrayExceptionDemo {
    public static void main(String[] args) {
        int []a = new int[5];
        System.out.println(a[5]); ← 不存在下标是 5 的元素
        System.out.println("程序正常结束");
    }
}
</pre>
该段代码编译不会发生错误，但运行时在控制台输出错误信息。
程序执行没有结束，而是发生了异常。在控制台显示了异常信息，这里的信息表示，在 main 线程中发生 java.lang.ArrayIndexOutOfBoundsException 异常，它发生在程序的第 5 行。</td></tr>
</table>

Java 语言规定在使用数组元素时，下标范围是 0 到数组的 length-1，超出这个范围将发生 ArrayIndexOutOfBoundsException 异常，它称为数组下标越界异常。

再看下面案例，该案例试图从键盘上输入一个字符，然后输出。但调用 System.in.read()方法发生编译错误。

【案例 9-2】输入/输出异常。

【程序 9-2】InputCharDemo.java

```
package com.boda.xy;
public class InputCharDemo{
   public static void main(String[] args){
      System.out.print("请输入一个字符：");
      var c = (char)System.in.read();
      System.out.println("c = " + c);
   }
}
```

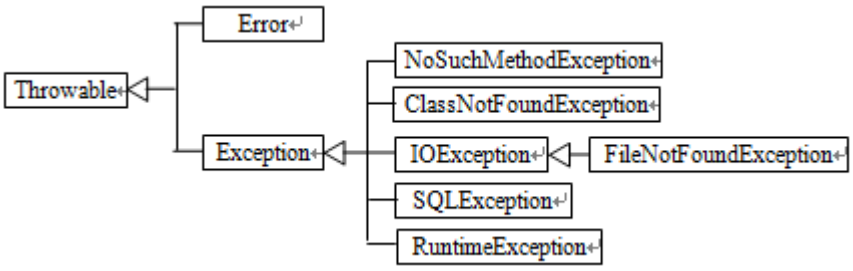
该行发生编译错误

编译程序报错，

讲授内容

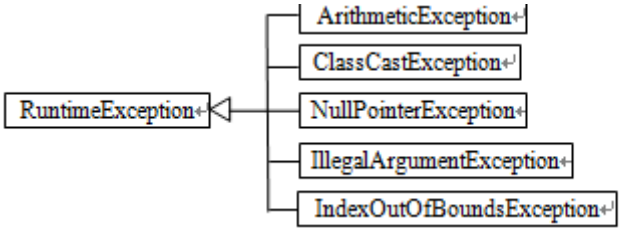
9.1.2 异常类型

Exception 类的子类一般又可分为两种类型：非检查异常和检查异常。



1. 非检查异常

非检查异常（unchecked exception）是 RuntimeException 类及其子类异常，也称为运行时异常。常见的非检查异常如图所示。



非检查异常是在程序运行时检测到的，可能发生在程序的任何地方且数量较大，因此编译器不对非检查异常（包括 Error 类的子类）处理，这种异常又称免检异常。

程序运行时发生非检查异常时运行时系统会把异常对象交给默认的异常处理程序，在控制台显示异常的内容及发生异常的位置。

下面介绍几种常见的非检查异常。

- NullPointerException：空指针异常，即当某个对象的引用为 null 时调用该对象的方法或使用对象时就会产生该异常，如：

```
String name = null;
System.out.println(name.length());     // 该语句发生异常
```

- ArithmeticException：算术异常，在做整数的除法或整数求余运算时可能产生的异常，它是在除数为零时产生的异常。

```
int a = 5;
int b = a / 0;                // 该语句发生异常
```

注意：浮点数运算不会产生该类异常。如，1.0/0.0 的结果为 Infinity。

- ClassCastException：对象转换异常，Java 支持对象类型转换，若不符合转换的规定，则产生类转换异常，例如：

```
Object o = new Object();
String s = (String)o;        // 该语句发生异常
```

- ArrayIndexOutOfBoundsException：数组下标越界异常，当引用数组元素的下标超出范围时产生的异常，例如：

```
int a[] = new int[5];
a[5] = 10;           // 该语句发生异常
```

因为定义的数组 a 的长度为 5，不存在 a[5]这个元素，因此发生数组下标越界异常。

- NumberFormatException：数字格式错误异常，在将字符串转换为数值时，如果字符串不能正确转换成数值则产生该异常，例如：

```
double d = Double.parseDouble("5m7.8"); // 该语句发生异常
```

异常的原因是字符串"5m7.8"不能正确转换成 double 型数据。

2. 检查异常

检查异常（checked exception）是除 RuntimeException 类及其子类以外的异常类，有时也称为必检异常。对这类异常，程序必须捕获或声明抛出，否则编译不能通过。程序 9-2 中的 read()方法声明抛出 IOException 异常就是必检异常。

9.2 用 try-catch 捕获异常

捕获并处理异常最常用的方法是用 try-catch-finally 语句，一般格式为：

```
try{
    // 需要处理的代码                    ← 可能发生异常的代码
} catch (ExceptionType1 exceptionObject){
    // 异常处理代码                      ← 处理异常的代码
}[catch (ExceptionType2 exceptionObject){
    // 异常处理代码                      ← 可有多个 catch 块
}]
[finally{                              ← finally 块是可选的
    // 最后处理代码
}]
```

1）try 块将程序中可能产生异常的代码段用大括号括起来，该块内可能抛出一种

或多种异常。

2）catch 块用来捕获异常，括号中指明捕获的异常类型及异常引用名，类似于方法的参数，它指明了 catch 语句所处理的异常。大括号中是处理异常的代码。catch 语句可以有多个，用来处理不同类型的异常。

3）finally 块是可选项。异常的产生往往会中断应用程序的执行，而在异常产生前，可能有些资源未被释放。有时无论程序是否发生异常，都要执行一段代码，这时就可以通过 finally 块实现。

下面是对程序 9-1 的修改，使用 try-catch 结构捕获异常。

【程序 9-3】ArrayExceptionDemo.java

```
package com.boda.xy;
public class ArrayExceptionDemo {
      public static void main(String[] args) {
           int []a = new int[5];
         try{
            System.out.println(a[5]);   <-- 抛出异常
         }catch(Exception e){
            System.out.println(e.toString());   <-- 处理异常
         }
         System.out.println("程序正常结束");
      }
}
```

9.3 捕获多个异常

有时捕获异常的两个或多个 catch 语句可能执行相同的代码序列。现在可以使用 JDK 7 提供的一个新功能，用一个 catch 语句处理多个异常，而不必单独捕获每个异常类型，这就减少了代码重复。

【案例 9-3】捕获处理多个异常。要在一个 catch 语句中处理多个异常，需要使用“或”运算符（|）分隔多个异常。下面的程序演示了捕获多个异常的方法。

【程序 9-5】MultiCatchDemo.java

```
// 这里捕获多个异常
catch(ArithmeticException | ArrayIndexOutOfBoundsException me){
     System.out.println("捕获到异常: " + me);
}
```

9.4 throws 和 throw 关键字

可以在声明方法时用 throws 子句声明抛出异常，将异常传递给调用该方法的方法处理。

声明方法抛出异常的格式如下：

```
返回值类型 方法名([参数列表]) throws 异常列表{
   // 方法体
}
```

按上述方式声明的方法，就可以对方法中产生的异常不作处理，若方法内抛出了

异常，则调用该方法的方法必须捕获这些异常或者再声明抛出。

【案例 9-4】声明方法抛出异常。程序 9-3 的例子是在 method()方法中处理异常，若不在该方法中处理异常，而由调用该方法的 main()方法处理，程序修改如下。

【程序 9-6】ThrowsExceptionDemo.java

完整代码请参见教材。

在程序中也可以用创建一个异常对象，然后用 throw 关键字抛出，或者将捕获到的异常对象用 throw 语句再次抛出，throw 语句的格式如下：

```
throw 异常实例;
```

这里，异常实例可以是用户创建的异常对象，也可以是程序捕获到的异常对象，该实例必须是 Throwable 类或其子类的实例。

【案例 9-5】使用 throw 语句抛出异常。

【程序 9-7】ThrowExceptionDemo.java

完整代码请参见教材。

9.5 try-with-resources 语句

try-with-resources 语句的基本形式如下：

```
try(resource-specification){
   // 使用资源                          ← 控制离开 try 块后，创建的资源将自动调用 close()方法关闭，代码简洁
}[
  catch(Exception e){                   ← catch 子句是可选的
  }
]
```

这里，resource-specification 是声明并初始化资源（如文件）的语句，它包含变量声明，用被管理的对象的引用初始化该变量。这里可以创建多个资源。当 try 块结束时，资源会自动释放。如果是文件，文件将被关闭，因此不需要显式调用 close()方法。try-with-resources 语句也可以包含 catch 语句和 finally 语句。

9.6 自定义异常类

尽管 Java 已经预定义了许多异常类，但有时还需要定义自己的异常。编写自定义异常类实际上是继承一个 API 标准异常类，用新定义的异常处理信息覆盖原有信息的过程。常用的编写自定义异常类的模式如下：

```
public class CustomException extends Exception {
   public CustomException(){}
   public CustomException(String message) {
        super(message);
   }
}
```

当然也可选用 Throwable 作为父类。其中无参数构造方法为创建缺省参数对象提供了方便。第二个构造方法将在创建这个异常对象时提供描述这个异常信息的字符串，通过调用超类构造方法向上传递给父类，对父类中的 toString()方法中返回的原有信息进行覆盖。

【案例 9-6】自定义异常及应用。假设程序中需要验证用户输入的数据值必须是正值。我们可以按照以上模式编写自定义异常类。

【程序 9-8】NegativeValueException.java

```
package com.boda.xy;
public class NegativeValueException extends Exception {
    public NegativeValueException()  {}
    public NegativeValueException(String message) {
          super(message);
     }
}
```

有了上述自定义异常，我们在程序中就可以使用它。假设编写程序要求用户输入圆半径，计算圆面积。该程序要求半径值应该为正值。

【程序 9-9】CustomExceptionDemo.java

```
package com.boda.xy;
import java.util.Scanner;
public class CustomExceptionDemo{
   public static void main(String[] args){
       var input = new Scanner(System.in);
       var radius = 0,area = 0;
       System.out.print("请输入半径值：");
       try{
          radius = input.nextDouble();
          if(radius < 0){
              throw new NegativeValueException("半径值不能小于 0.");
       } else{
          area = Math.PI * radius * radius;
          System.out.println("圆的面积是：" + area);
       }
     }catch(NegativeValueException nve){
         System.out.println(nve.getMessage());
     }
   }
}
```

课堂互动与讨论	要求从键盘输入一个 double 型的圆的半径，计算并输出其面积。测试当输入的数据不是 double 型数据（如字符串“abc”）会抛出什么异常？试用异常处理方法修改程序。
总结与作业	**一、知识点总结** （1）异常是在 Java 程序运行过程中产生的使程序终止正常运行的错误对象。Java 定义了各种异常类处理异常。 （2）Java 异常是扩展自 java.lang.Throwable 的类的实例。Java 提供大量的预定义的异常，例如：Error、Exception、RuntimeException、ClassNotFoundException、NullPointerException 和 ArithmeticException。 （3）Java 异常一般分为两类：非检查异常（运行时异常），它是 RuntimeException 类及其子类；检查异常（非运行时异常），Exception 类除 RuntimeException 类外的子

	类。对运行时异常可以不处理，但程序运行若产生该类异常，运行时系统同样抛出。对非运行时异常，要求必须捕获或声明抛出。 （4）所有异常都是在方法中产生的，称为抛出异常。程序应该处理各种异常。最常用的方法是使用 try-catch-finally 结构。 （5）可以用一个 catch 块捕获多个异常，也可以用多个 catch 块捕获多个异常，此时异常的指定顺序是非常重要的，应该先指定捕获子类异常，后捕获父类异常，否则会导致编译错误。 （6）不管 try 块中是否出现了异常，在任何情况下，finally 块中的代码都将被执行，除非调用 System.exit()结束程序运行。 （7）在方法中捕获到的异常可以使用 throw 语句再次抛出，也可以使用 throws 声明方法抛出异常。 （8）使用 try-with-resources 结构可以打开实现了 java.lang.AutoCloseable 接口的那些资源，资源使用完后可自动关闭，从而避免程序员忘记关闭资源的错误。 （9）根据需要可以定义自己的异常类，这需要继承 Exception 类或其子类。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2．4.编写程序，在 main()方法中使用 try 块抛出一个 Exception 类的对象，为 Exception 的构造方法提供一个字符串参数，在 catch 块内捕获该异常并打印出字符串参数。添加一个 finally 块并打印一条消息。 5.编写程序，定义一个 static 方法 methodA()，令其声明抛出一个 IOException 异常，再定义另一个 static 方法 methodB()，在该方法中调用 methodA()方法，在 main()方法中调用 methodB()方法。试编译该类，看编译器会报告什么？对于这种情况应如何处理？由此可得到什么结论？。

《Java 基础入门》教案

第 19 讲　　泛型　　2 学时 110 分钟

章节内容	**第 10 章　泛型与集合** 10.1　泛型 10.1.1　泛型类型 10.1.2　泛型方法 10.1.3　通配符（?）的使用 10.1.4　方法中有界参数
教学目标	• 了解泛型的概念。 • 掌握泛型的定义和使用。 • 了解通配符（?）的使用和方法中有界参数。
重点	**重点**：泛型的定义和使用。
难点	**难点**：通配符（?）的使用和方法中有界参数。
教学环节 教学方法 及其它说 明事项	1.采用讲授法、演示法、讨论法。 2.案例+项目式教学。 3.采用线上线下混合教学模式。配套线上课程地址： www.qingline.net 4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。
导言 或复习	回顾一下，我们在 7.6 节中定义了一个整数栈类 IntStack，该类使用 Integer 作为栈的元素，这就限制了该类只能对 Integer 元素操作。如果要使这个栈类更具有通用性，我们可以使用 Object 作为栈的元素，因为 Object 类是所有类的超类，所以 Object 可以引用任何对象类型。 然而，这种做法无法提供类型的安全性，在进行类型转换时可能发生类型不匹配异常。使用泛型就可以提高类型安全性，因为，它可以使类型转换自动地、隐式地进行。 所谓**泛型**（generics）就是带一个或多个**类型参数**（type parameter）的类或接口。对于上述讨论的对象栈，可以使用泛型定义。

讲授内容	**10.1.1　泛型类型**
	泛型（generics）是带一个或多个类型参数的类或接口。
	【案例 10-1】定义一个泛型 Node 类表示节点，类型参数 T 表示节点中存放的值。
	【程序 10-1】Node.java
	`package com.boda.xy;`
	`public class Node<T> {`
	`  private T data;  // 泛型成员`
	`  public  Node(){}  // 默认构造方法`
	`  public  Node(T data){  // 带参数构造方法`

```
        this.data = data;
    }
    public T getData() {          // 访问方法定义
        return data;
    }
    public void setData(T data) {     // 修改方法定义
        this.data = data;
    }
    // 显示类型名
    public void showType(){
        System.out.println("T 的类型是："+ data.getClass().getName());
    }
}
```

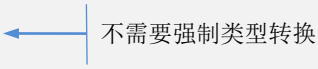
泛型类型的使用与方法调用类似，方法调用需向方法传递参数，使用泛型需传递一个类型参数，即用某个具体的类型替换 T。例如，如果要在 Node 对象中存放 Integer 对象，就需要在创建 Node 对象时为其传递 Integer 类型参数。要实例化泛型类对象，也使用 new 运算符，但在类名后面需加上要传递的具体类型。

```
var intNode = new Node<Integer>();
```

一旦创建了 intNode 对象，就可以调用 setData()方法设置其中的 Integer 对象，调用 getData()方法返回其中的 Integer 对象，如下代码所示。

【程序 10-2】NodeTest.java

```
package com.boda.xy;
public class NodeTest{
    public static void main(String[]args){
        var intNode = new Node<Integer>();
        intNode.setData(999);
        var value = intNode.getData();
        System.out.println(value);
        intNode.showType();
    }
}
```



按照约定，类型参数名使用单个大写字母表示。常用的类型参数名有：E 表示元素，K 表示键，N 表示数字，T 表示类型，V 表示值等。

还可以定义泛型接口。泛型可能具有多个类型参数，但在类或接口的声明中，每个参数名必须是唯一的。

【案例 10-2】带两个类型参数的泛型接口，以及实现泛型接口的泛型类。程序 10-3 定义了带两个参数的泛型接口 Entry，程序 10-4 定义了实现 Entry 接口的泛型类 Pair。

【程序 10-3】Entry.java

```
package com.boda.xy;
public interface Entry<K, V> {
    public K getKey();
    public V getValue();
}
```

【程序 10-4】Pair.java

```
package com.boda.xy;
public class Pair<K, V> implements Entry<K, V>{
    private K key;
    private V value;
    public Pair(K key, V value) {    // 构造方法
        this.key = key;
        this.value = value;
    }
    public void setKey(K key) { this.key = key; }
    public K getKey() { return key; }
    public void setValue(V value) { this.value = value; }
    public V getValue() { return value; }
}
```

下面语句创建两个 Pair 类实例：

```
var p1 = new Pair<Integer,String>(20,"twenty");
var p2 = new Pair<String,String>("china", "Beijing");
```

10.1.2 泛型方法

泛型方法（generic method）是带类型参数的方法。类的成员方法和构造方法都可以定义为泛型方法。泛型方法的定义与泛型类型的定义类似，但类型参数的作用域仅限于声明的方法和构造方法内。泛型方法可以定义为静态的和非静态的。

【案例 10-3】泛型方法的定义和使用。下面的 MathUtil 类中定义两个 static 的泛型方法 swap()和 compare()。swap()方法用于交换任何数组中两个元素（数组元素类型不是基本类型），compare()方法用于比较两个泛型类 Pair 对象的参数 K 和 V 是否相等。特别注意，对于泛型方法必须在方法返回值前指定泛型，如<K,V>。

【程序 10-5】MathUtil.java

10.1.3 通配符（?）的使用

泛型类型本身是一个 Java 类型，就像 java.lang.String 和 java.time.LocalDate 一样，为泛型类型传递不同的类型参数会产生不同的类型。例如，下面的 list1 和 list2 就是不同的类型对象。

```
List<Object> list1 = new ArrayList<Object>();
List<String> list2 = new ArrayList<String>();
```

这里 List 和 ArrayList 是泛型接口和泛型类。尽管 String 是 Object 的子类，但 List<String>与 List<Object>却没有关系，List<String>并不是 List<Object>的子类型。因此，把一个 List<String>对象传递给一个需要 List<Object>对象的方法，将会产生一个编译错误。请看下面代码。

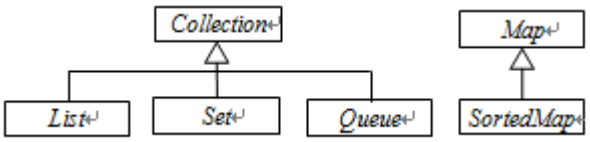
```
public static void printList(List<Object> list){
   for(Object element : list){
      System.out.println(element);
   }
}
```

	该方法的功能是打印传递给它的列表的所有元素。如果传递给该方法一个List<String>对象，将发生编译错误。如果要使上述方法可打印任何类型的列表，可将其参数类型修改为List<?>，如下所示：
	`public static void printList(List<?> list){`
	`  for(Object element : list){`
	`  System.out.println(element);`
	`  }`
	`}`
	这里，问号（?）就是通配符，它表示该方法可接受任何类型的List对象。
	【案例10-4】声明方法参数使用问号（?）通配符。List<?> list表示元素是任何类型的List对象。
	【程序10-6】WildCardDemo.java
	10.1.4 方法中有界参数
	有时需要限制传递给类型参数的类型种类，例如，要求一个方法只接受Number类或其子类的实例，这就需要使用**有界类型参数**（bounded type parameter）。
	有界类型分为上界和下界，上界用extends指定，下界用super指定。例如，要声明上界类型参数，应使用问号（？），后跟extends关键字，然后是上界类型。这里，extends具有一般的意义，对类表示扩展（extends），对接口表示实现（implements）。
	假如要定义一个getAverage()方法，它返回一个列表中所有数字的平均值，我们希望该方法能够处理Integer列表、Double列表等各种数字列表。但是，如果把List<Number>作为getAverage()方法的参数，它将不能处理List<Integer>列表或List<Double>列表。为了使该方法更具有通用性，可以限定传递给该方法的参数是Number对象或其子类对象的列表，这里Number类型就是列表中元素类型的**上界**（upper bound）。下面案例的getAverage()方法就是这样的参数。
	【案例10-5】方法参数使用有界通配符。
	【程序10-7】BoundedTypeDemo.java

总结与作业	**一、知识点总结** （1）泛型是带一个或多个类型参数的类或接口。创建泛型类的实例时需要为其传递类型参数。 （2）泛型方法是带类型参数的方法。类的成员方法和构造方法都可以定义为泛型方法。泛型类和非泛型类都可以定义泛型方法。 （3）泛型类型是不变的，当S是T的子类型时，G<S>并不是G<T>的子类型，二者没有关系。 （4）通过使用通配符G<? extends T>或者G<? super T>，可以指定一个方法接受一个带子类型或者父类型参数的泛型类型实例。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2 .定义一个泛型类Point<T>，其中包含x和y两个类型为T的成员，定义带两个参数的构造方法，为x和y定义setter和getter，另外定义translate()方法将点移到新的坐标。编写main()方法，创建Point<Integer>对象，调用translate()方法。
教学反思	

《Java 基础入门》教案

第 20 讲　　集合　　2 学时 110 分钟

章节内容	**第 10 章　泛型与集合** 10.2　集合框架 10.3　List 接口及实现类 10.4　Set 接口及实现类 10.4.3　对象顺序 10.5　Queue 接口及实现类 10.7　Map 接口及实现类 10.8　Collections 类
教学目标	• 了解集合框架。 • 掌握 List 和 Set 接口及实现类。 • 掌握 Map 接口及实现类。 • 了解 Queue 接口和实现类，Collections 类。
重点	**重点**：List 接口及实现类、Set 接口及实现类、Map 接口及实现类。
难点	**难点**：Map 接口及实现类。
教学环节 教学方法 及其它说 明事项	1.采用讲授法、演示法、讨论法。 2.案例+项目式教学。 3.采用线上线下混合教学模式。配套线上课程地址： www.qingline.net 4.软件环境：Windows 10，JDK 16 以上，Eclipse 2020 以上。
导言 或复习	在编写面向对象的程序时，经常要用到一组类型相同的对象。可以使用数组来集中存放这些类型相同的对象，但数组一经定义便不能改变大小。因此，Java 提供了一个集合框架（Collections Framework），该框架定义了一组接口和类，使得处理对象组更容易。 集合是指集中存放一组对象的一个对象。集合相当于一个容器，它提供了保存、获取和操作其他元素的方法。集合能够帮助 Java 程序员轻松地管理对象。Java 集合框架由两种类型构成，一个是 Collection，另一个是 Map。Collection 对象用于存放一组对象，Map 对象用于存放一组“关键字/值”的对象。Collection 和 Map 是最基本的接口，它们又有子接口，这些接口的层次关系如图 10-5 所示。 
讲授内容	**10.3　List 接口及实现类** List 接口实现一种线性表的数据结构。存放在 List 中的所有元素都有一个下标（从 0

开始），可以通过下标访问 List 中的元素。List 中可以包含重复元素。List 接口的实现类包括 ArrayList、LinkedList、Vector 和 Stack。

10.3.1 List 的操作

List 接口除继承 Collection 的方法外，还定义了一些自己的方法。使用这些方法可以实现定位访问、查找、迭代和返回子线性表。

10.3.2 ArrayList 类

ArrayList 是最常用的列表实现类，它通过数组实现的集合对象。ArrayList 类实际上实现了一个变长的对象数组，其元素可以动态地增加和删除。它的定位访问时间是常量时间。

下列代码创建一个 ArrayList 对象向其中插入几个元素，并使用 ArrayList 的有关方法对它操作。

```
var bigCities = new ArrayList<String>();
bigCities.add("北京");
bigCities.add("上海");
bigCities.add("广州");
System.out.println(bigCities.size());
bigCities.add(2,"伦敦");                          // 插入元素
bigCities.set(2,"纽约");                          // 修改元素
System.out.println(bigCities.contains("北京"));
System.out.println(bigCities);
System.out.println(bigCities.indexOf("巴黎"));
```

10.3.3 遍历集合元素

在使用集合时，遍历集合元素是最常见的任务。遍历集合中的元素有多种方法：用简单的 for 循环、用增强的 for 循环和用 Iterator 迭代器对象。

1. 使用简单的 for 循环

使用简单的 for 循环可以遍历集合中的每个元素。

```
for(var i = 0; i < bigCities.size(); i++){
     System.out.print(bigCities.get(i) + "  ");
}
```

2. 使用增强的 for 循环

使用增强的 for 循环不但可以遍历数组的每个元素，还可以遍历集合中的每个元素。下面的代码打印集合的每个元素：

```
for (var city : bigCities)
  System.out.println(city);
```

上述代码的含义是：将集合 bigCities 中的每个对象存储到 city 变量中，然后打印输出。使用这种方法只能按顺序访问集合中的元素，不能修改和删除集合元素。

3. 使用迭代器

迭代器是一个可以遍历集合中每个元素的对象。调用集合对象的 iterator()方法可以得到 Iterator 对象，再调用 Iterator 对象的方法就可以遍历集合中的每个元素。Iterator 接口定义了如下 3 个方法。

- boolean hasNext()：返回迭代器中是否还有对象。
- E next()：返回迭代器中下一个对象。
- void remove()：删除迭代器中的当前对象。

Iterator 使用一个内部指针，开始它指向第一个元素的前面。如果在指针的后面还有元素，hasNext()方法返回 true。调用 next()方法，指针将移到下一个元素，并返回该元素。remove()方法将删除指针所指的元素。假设 myList 是 ArrayList 的一个对象，要访问 myList 中的每个元素，可以按下列方法实现：

```
Iterator iterator = myList.iterator();   // 得到迭代器对象
while (iterator.hasNext()){
    System.out.println(iterator.next());
}
```

使用 Iterator 也可以用 for 循环访问集合元素。

```
for(var iterator = myList.iterator();iterator.hasNext();){
    System.out.println(iterator.next());
}
```

10.4 Set 接口及实现类

Set 接口对象类似于数学上的集合概念，其中不允许有重复的元素。Set 接口没有定义新的方法，只包含从 Collection 接口继承的方法。Set 接口的常用实现类有：HashSet 类、TreeSet 类和 LinkedHashSet 类。

10.4.1 HashSet 类

HashSet 类用散列方法存储元素，具有最好的存取性能，但元素没有顺序。

下面代码演示了 HashSet 的使用。

```
var words = new HashSet<>();
words.add("one");
words.add("two");
words.add("three");
words.add("four");
words.add("one");    // 不能将重复的元素添加到集合中
for(var w : words)
  System.out.print(w + " ");    // four one two three
```

从结果可以看到，在向 Set 对象中添加元素时，重复的元素不能添加到集合中。另外，由于程序中使用的实现类为 HashSet，它并不保证集合中元素的顺序。

10.4.2 TreeSet 类

TreeSet 实现一种树集合，它使用红-黑树为元素排序，添加到 TreeSet 中的元素必须是可比较的，即元素的类必须实现 Comparable<T>接口。它的操作要比 HashSet 慢。

TreeSet 类的默认构造方法创建一个空的树集合，其他构造方法如下。

	• TreeSet(Collection c)：用指定集合 c 中的元素创建一个新的树集合，集合中的元素按自然顺序排序。 • TreeSet(Comparator c)：创建一个空的树集合，元素的排序规则按给定的比较器 c 的规则排序。 【案例 10-7】下面案例创建一个 TreeSet 对象，其中添加了 4 个字符串对象。 【程序 10-9】TreeSetDemo.java
案例学习	**10.6 案例研究：用集合存储、遍历学生信息** 1. 问题描述 设计学生类 Student，属性：学号（整型）；姓名（字符串），语文成绩，数学成绩，英语成绩和总成绩，成绩均为 int 型。程序创建多个学生信息存储到集合中，学号相同时视为同一个对象，不重复存储，遍历集合将学生信息按总成绩降序输出到控制台。 2. 设计思路 （1）定义学生类，重写方法 hashCode()和 equals()，用学号识别是否为重复的对象。 （2）创建 HashSet 集合对象，使用泛型限定只存储学生对象。 （3）创建若干学生对象，把学生对象添加到集合。 （4）遍历集合，输出学生信息到控制台。 （5）修改学生类使其实现 Comparable 接口，让对象拥有自然序。实现接口中的抽象方法 compareTo，按总成绩降序排序学生对象。 （6）定义 TreeSet 集合，将 HashSet 集合中的学生对象添加到此 TreeSet 集合中，实现学生对象按总成绩降序存储。 3. 代码实现 下面程序是该案例的实现代码。Student 类的代码如程序 10-13 所示，录入和输出学生代码如程序 10-14 所示。 【程序 10-13】Student.java 【程序 10-14】StudentSet.java 完整代码请见教材，《Java 基础入门-项目案例+微课视频+题库》，沈泽刚，清华大学出版社，2021 年。
课堂讲授	**10.7 Map 接口及实现类** Map 是用来存储“键/值”对的对象，通常称为映射。在 Map 中存储的关键字和值都必须是对象，并要求关键字是唯一的，而值可以重复。 **10.7.1 Map 接口** Map 接口实现基本操作的方法包括添加“键/值”对、返回指定键的值、删除“键/值”对等。 • public V put(K key, V value)：向映射对象中添加一个“键/值”对。 • public V get(Object key)：返回指定键的值。 • public V remove(Object key)：从映射中删除指定键的“键/值”对。

10.7.2 HashMap 类

HashMap 类以散列方法存放“键/值”对。

【案例 10-11】下面案例使用 HashMap 存放几个国家名称和首都名称对照表，国家名称作为键，首都名作为值，然后对其进行各种操作。

【程序 10-13】MapDemo.java

完整代码请见教材，《Java 基础入门-项目案例+微课视频+题库》，沈泽刚，清华大学出版社，2021 年。

10.7.3 TreeMap 类

TreeMap 类实现了 SortedMap 接口，它保证 Map 中的“键/值”对按关键字升序排序。

对程序 10-13 的例子，假设希望按国家名的顺序输出 Map 对象，仅将 HashMap 改为 TreeMap 即可。

10.8 Collections 类

java.util.Collections 类提供了若干 static 方法实现集合对象的操作。这些操作大多对 List 操作，主要包括下面几个方面：排序、重排、查找、求极值以及常规操作等。

使用 sort()方法可对列表中元素排序，它有下面两种格式：

- public static<T> void sort(List<T> list)
- public static<T> void sort(List <T>list, Comparator<? super T> c)

该方法实现对 List 的元素按升序或指定的比较器顺序排序。该方法使用优化的归并排序算法，因此排序是快速的和稳定的。在排序时如果没有提供 Comparator 对象，则要求 List 中的对象必须实现 Comparable 接口。

下面代码对元素为 Integer 的 List 排序。

```
var numbers = Arrays.asList(20, -5, 49, 8);
Collections.sort(numbers);
for(var n :numbers){
    System.out.print(n + "  ");
}
```

总结与作业

一、知识点总结

（1）Collection 接口为所有集合类提供类共同方法。列表是一个有序集合，其中的每个元素都有一个整数索引，ArrayList 是它的常用实现类。Set 是不重复元素的集合，HashSet 和 TreeSet 是它的两个常用实现类。

（2）Queue 实现一种队列的数据结构，它是以先进先出（First-In-First-Out，FIFO）的方式排列其元素。Deque 对象实现双端队列，ArrayDeque 和 LinkedList 是它的两个实现类。

（3）java.util.Collections 工具类中针对 List 定义了大量操作算法，如排序、查找、重排、求极值等操作。

（4）使用迭代器可以遍历集合中的元素，但无法实现高效的并发执行。使用 Stream API 可以高效对集合操作。

	（5）Map 是用来存储“键/值”对的对象，要求关键字是唯一的，而值可以有重复的。Map 接口常用的实现类有 HashMap、TreeMap 类。 **二、课后作业** 1.登录本书在线作业网站，完成本章作业题目。地址：https://www.qingline.net/ 2.编写程序，随机生成 20 个两位整数，将它们分别添加到 HashSet 对象和 TreeSet 对象中。 （1）使用增强的 for 循环访问集合中的每个元素。 （2）使用 Iterator 迭代器访问集合中的每个元素。为什么集合中的元素不是 20 个？