**《Java面向对象编程》第二版的勘误以及更新**

备注：以下页数以第一次印刷的书本为准。以下红色字体是原书中的需更新的内容，绿色字体是修改后的内容。

以下勘误信息多数为笔误，不影响阅读。只有第8章的8.4(小结)对知识作为了稍微修改。

在本书以后的重印中，会直接修改纸质书中的内容。

2017/10/26日

孙卫琴

目录

[1. Page VI 前言部分 1](#_Toc496793271)

[2. Page 138 ,第4章，4.9.1节 3](#_Toc496793272)

[3. Page150,第5章，5.1.2节 3](#_Toc496793273)

[4. Page174,第6章，6.1节，图6-1 3](#_Toc496793274)

[5. Page210，7.1节，表7-2 4](#_Toc496793275)

[6. Page258，8.4节 4](#_Toc496793276)

[7. Page273，第9章，9.2.5节 5](#_Toc496793277)

[8. Page282,第9章，9.3.1节开头 5](#_Toc496793278)

[9. Page334，第11章，11.2.3节 6](#_Toc496793279)

[10. Page351，第11章，11.3.5节，图11-5 7](#_Toc496793280)

[11. Page373，第12章，12.1.1节 7](#_Toc496793281)

[12. Page390,第12章,第4道思考题 8](#_Toc496793282)

[13. Page409，第3章，13.4.3节开头文字 8](#_Toc496793283)

[14. Page415，第13章，例程13-16 9](#_Toc496793284)

[15. Page416,第13章，13.8节 9](#_Toc496793285)

[16. Page445，第13章，13.15节 10](#_Toc496793286)

[17. Page511，第15章，15.10节 10](#_Toc496793287)

[18. Page573，第18章，18.8.4节 10](#_Toc496793288)

[19. Page598，第18章，例程19-30 11](#_Toc496793289)

[20. Page601，第18章，例程18-31 11](#_Toc496793290)

[21. Page626，第19章，19.4.1节 11](#_Toc496793291)

[22. Page636，第19章，表19-4 12](#_Toc496793292)

[23. Page669,第20章，20.9节 12](#_Toc496793293)

[24. Page669,第20章，20.9节 13](#_Toc496793294)

[25. 本书的反面封面 13](#_Toc496793295)

## Page VI 前言部分

（1）把前言靠末尾的以下内容：

本书中实例源文件以及思考题答案的下载网址为：www.javathinker.net/download.htm。

改为：

本书中实例源文件、思考题答案、视频课程以及PPT讲义和教学大纲等的下载网址为：www.javathinker.net/javabook.jsp。

（2）在“这本书是否适合您”后面再增加一个栏目：

本书使用指南

这本书涵盖了Java从入门到精通的所有知识，既详细地介绍了基本的Java语法和创建程序的过程，又深入介绍了按照面向对象的思想来开发软件程序的高级技巧和设计模式。根据读者的不同技术背景，提供以下学习建议：

（1）针对没有任何编程经验的读者

 建议先学习本书的配套视频的第3、4、5节课，按照视频的指导在本地机器上安装JDK，并且创建和运行第一个程序。在学习视频和实践的过程中，同时学习第2章的内容，对Java语言的基本语法获得更全面和深入的认识。

建议一直阅读完第12章，再回过头来学习第1章的内容。第1章的内容高屋建瓴，站在开发整个软件系统的角度，介绍了如何运用面向对象的开发思想来创建可扩展、可重用和可维护的软件系统。

 始终把视频和书结合起来学习，这样会更加轻松省力。总的说来，视频讲授的深度要低于书的深度。通过观看视频，可以帮助您顺利地对特定知识点有概要和基础的了解，掌握其中的核心内容，同时阅读书中相关章节，对该知识点获得更深入和全面的认识。

 按照由浅入深地循序来学习，要一次读懂本书不是很现实。可以多读几遍，先粗读，再精读，每次学习都会有新的收获。

（2）针对已经编写过简单程序的读者

 您可以按照书的先后循序来阅读。对书中从第2章开始的一些基础内容，如果您已经熟悉，那可以跳过这些章节，去阅读您感兴趣的内容。在阅读过程中，多留意书中对设计模式、Java虚拟机执行Java程序的原理等高级技术的阐述，这样可以帮助您提升Java编程的能力，开发出更加健壮、具有良好性能和架构的Java应用。

（3）针对已经对Java编程比较熟练的读者

对于您来说，这本书是一本内容非常全面的Java编程参考手册。本书对许多语法细节、运行原理和类库的用法都讲得很透彻。当您在开发程序的过程中，如果对特定语法细节不太清楚，或者调试中遇到困难，或者不熟悉JDK类库中一些实用类和接口的用法，都可以参阅本书，从中会获得满意的答案。

(4)针对Java培训老师

本书是一本优秀的Java教材。在本书的技术支持网址中，提供了详细的教学大纲，以及全面的试题库和答案。此外，您还可以安排学生反复观看本书的配套视频课程，让您的教学过程更加省力有效。

## Page 138 ,第4章，4.9.1节

倒数第4行的程序代码：

byte b=(int)a; //数据溢出，变量b的值为0

改为：

byte b=(byte)a; //数据溢出，变量b的值为0

## Page150,第5章，5.1.2节

对于第一段程序代码：

switch语句是多路分支语句，它的基本语法为：

switch (expr){

 case value1

 statements;

 break;

 …

 case valueN

 statements;

在以上value1和valueN后面要加上冒号“:”，改为：

switch (expr){

 case value1:

 statements;

 break;

 …

 case valueN:

 statements;

## Page174,第6章，6.1节，图6-1

原图：



改为：



## Page210，7.1节，表7-2

把表7-2中第二列中的“Public”、“Protected”、“Private”改为小写，修改后为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 访问级别 | 访问控制修饰符 | 同类 | 同包 | 子类 | 不同的包 |
| 公开 | public | √ | √ | √ | √ |
| 受保护 | protected | √ | √ | √ | － |
| 默认 | 没有访问控制修饰符 | √ | √ | － | － |
| 私有 | private | √ | － | － | － |

## Page258，8.4节

原文：

接口与抽象类都位于系统的抽象层，但两者有着不同的特点和用处。抽象类的优势在于可以为部分方法提供默认的实现，避免子类重复实现它们，从而提高代码的可重用性。但抽象类的这一优势限制了多继承。因为子类有可能覆盖抽象类中方法的实现，还有可能隐藏抽象类的成员变量和静态方法，为了简化Java虚拟机的绑定机制，Java语言不允许一个类有多个直接的父类。

接口的优势在于一个类可以实现多个接口。接口获得这一优势是以不允许为任何方法提供实现作为代价的。

以上原文中部分内容在知识点上说法有误，对这两段正文全部改为：

接口与抽象类都位于系统的抽象层，但两者有着不同的特点和用处。抽象类的优势在于可以为部分方法提供默认的实现，避免子类重复实现它们，并且可以包含实例成员变量，从而提高代码的可重用性。但抽象类的这一优势会使得多继承变得错综复杂，为了简化Java虚拟机的绑定机制，Java语言不支持多继承，即不允许一个类有多个直接的父类。

接口的优势在于一个类可以实现多个接口，使得一个类可以身兼数职，拥有多种功能，提供多种服务。并且在JDK的高版本中，还允许为接口的一些方法提供默认实现，从而提高代码的可重用性。

## Page273，第9章，9.2.5节

当前页从倒数第3行开始的程序代码：

void method5(){

 try{

 method1();

 }**catch(IOException e){…}**

}

改为：

void method5(){

 try{

 method1();

 }**catch(IOException e){ System.out.println("IOException");}**

}

## Page282,第9章，9.3.1节开头

原文：

RuntimeException类及其子类都称为运行时异常，这种异常的特点是Java编译器不会检查它，也就是说，当程序中可能出现这类异常，即使没有用try-catch语句捕获它，也没有用throws子句声明抛出它，也会编译通过。例如当以下divide()方法的参数b为0，执行“a/b”操作时会出现ArrithmeticException异常，它属于运行时异常，Java编译器不会检查它：

public int divide(int a,int b){

 return a/b; //当参数b为0，抛出ArrithmeticException

}

把以上的“ArrithmeticException”改为“ArithmeticException”：

RuntimeException类及其子类都称为运行时异常，这种异常的特点是Java编译器不会检查它，也就是说，当程序中可能出现这类异常，即使没有用try-catch语句捕获它，也没有用throws子句声明抛出它，也会编译通过。例如当以下divide()方法的参数b为0，执行“a/b”操作时会出现ArithmeticException异常，它属于运行时异常，Java编译器不会检查它：

public int divide(int a,int b){

 return a/b; //当参数b为0，抛出ArithmeticException

}

## Page334，第11章，11.2.3节

当前页的第一段程序代码：

public class Sub extends Base{

 private int b;

 //在Bas父类的构造方法中初始化实例变量a，在Sub子类的构造方法中初始化实例变量b

 public Base(int a,int b){**super(a);** this.b=b;}

 public int getB(){return b;}

 public static void main(String args[]){

 Sub sub=**new Sub(1,2);**

 System.out.println("a="+sub.getA()+" b="+sub.getB()); //打印a=1 b=2

 }

}

对以上程序代码做两处修改，第一个“Bas”改为“Base”，第二个“Base”改为“Sub”：

public class Sub extends Base{

 private int b;

 //在Base父类的构造方法中初始化实例变量a，在Sub子类的构造方法中初始化实例变量b

 public Sub(int a,int b){**super(a);** this.b=b;}

 public int getB(){return b;}

 public static void main(String args[]){

 Sub sub=**new Sub(1,2);**

 System.out.println("a="+sub.getA()+" b="+sub.getB()); //打印a=1 b=2

 }

}

## Page351，第11章，11.3.5节，图11-5

修改图11-5，把图中的一个“ServiceIml3”改为“ServiceImpl2”。原图为：



修改为：



## Page373，第12章，12.1.1节

12.1.1节开头的程序代码：

1．在创建实例内部类的实例时，外部类的实例必须已经存在。例如要创建InnerTool类的实例，必须先创建Outer外部类的实例：

Outer.InnerTool tool=new Outer().new InnerTool();

以上代码等价于：

Outer outer=new Outer();

Outer.InnerTool tool =outer.new InnerTool()

以上最后一行代码需在末尾增加一个分号“;”，改为：

以上代码等价于：

Outer outer=new Outer();

Outer.InnerTool tool =outer.new InnerTool();

## Page390,第12章,第4道思考题

原文:

4．外部类不能被private、protected修饰，有些内部类可以被private、protected修饰。这句话对吗？

把“外部类”改为“顶层类”：

4．顶层类不能被private、protected修饰，有些内部类可以被private、protected修饰。这句话对吗？

## Page409，第3章，13.4.3节开头文字

原文：

当线程在运行中执行了Thread类的yield()静态方法，如果此时具有相同优先级的其他线程处于就绪状态，yield()方法将把当前运行的线程放到可运行池中并使另一个线程运行。如果没有相同优先级的可运行进程，yield()方法什么都不做。

修改两处，增加“或更高”，把“进程”改为“线程”

当线程在运行中执行了Thread类的yield()静态方法，如果此时具有相同或更高优先级的其他线程处于就绪状态，yield()方法将把当前运行的线程放到可运行池中并使另一个线程运行。如果没有相同优先级的可运行线程，yield()方法什么都不做。

## Page415，第13章，例程13-16

原文：

package countbirds;

public class Machine implements Runnable {

 private int a=1; //共享数据

 public void run() {

 for(int i=0;i<1000;i++){

 a+=i;

 Thread.yield(); //给其他线程运行的机会

 a-=i;

 System.out.println(a);

 }

 }

修改其中的for语句，把“i=0”改为“i=1”，并把“i<1000”改为“i<=1000”：

package countbirds;

public class Machine implements Runnable {

 private int a=1; //共享数据

 public void run() {

 for(int i=1;i<=1000;i++){

 a+=i;

 Thread.yield(); //给其他线程运行的机会

 a-=i;

 System.out.println(a);

 }

 }

## Page416,第13章，13.8节

当前页第3段正文：

以上操作被称为原子操作。原子操作由相关的一组操作完成，这些操作可能会操纵与其他线程共享的资源。为了保证得到正确的运算结果，一个线程在执行原子操作的期间，应该采取措施使得其他线程不能操纵共享资源，这里的共享资源是指Machine对象的实例变量a。

正文中插入“业务逻辑上”，修改后为：

以上操作被称为原子操作。原子操作由业务逻辑上相关的一组操作完成，这些操作可能会操纵与其他线程共享的资源。为了保证得到正确的运算结果，一个线程在执行原子操作的期间，应该采取措施使得其他线程不能操纵共享资源，这里的共享资源是指Machine对象的实例变量a。

## Page445，第13章，13.15节

13.15节开头：

编写多线程的程序代码时，既要保证线程的同步，又要避免死锁，还要考虑并发性能，因此对开发人员的要求很高。为了帮助开发人员编写出高效安全的多线程代码，从JDK5开始，增加了java.util.cuncurrent并发包，它提供了许多实用的处理多线程的接口和类，主要包括：

把“java.util.cuncurrent”改为“java.util.concurrent”：

编写多线程的程序代码时，既要保证线程的同步，又要避免死锁，还要考虑并发性能，因此对开发人员的要求很高。为了帮助开发人员编写出高效安全的多线程代码，从JDK5开始，增加了java.util.concurrent并发包，它提供了许多实用的处理多线程的接口和类，主要包括：

## Page511，第15章，15.10节

当前页倒数第一段代码：

因此，在通过Arrays.asList()方法得到了一个List对象后，还可以把它转换为其他类型的集合。例如：

Integer[] array={11,22,33};

List<Integer> list=Arrays.asList(array); ////把数组转换为List

把“////”改为“//”：

因此，在通过Arrays.asList()方法得到了一个List对象后，还可以把它转换为其他类型的集合。例如：

Integer[] array={11,22,33};

List<Integer> list=Arrays.asList(array); //把数组转换为List

## Page573，第18章，18.8.4节

本小节开头：

 BufferedWriter带有缓冲区，它可以先把一批数据写到缓冲区内，当缓冲区满的时候，再把缓冲区的数据写到字符输出流中。这可以避免每次都执行物理写操作，从而提高I/O操作的效率。BufferedReader有一个readLine()方法，而BufferedWriter没有相应的writerLine()方法。…….

把“writerLine”改为“writeLine”:

 BufferedWriter带有缓冲区，它可以先把一批数据写到缓冲区内，当缓冲区满的时候，再把缓冲区的数据写到字符输出流中。这可以避免每次都执行物理写操作，从而提高I/O操作的效率。BufferedReader有一个readLine()方法，而BufferedWriter没有相应的writeLine()方法。......

## Page598，第18章，例程19-30

原代码：

import java.io.\*;

impirt java.io.Date;

public class DirUtil{

把“impirt”改为“import”:

import java.io.\*;

import java.io.Date;

public class DirUtil{

## Page601，第18章，例程18-31

本例程末尾的原代码：

 //遍历循环C:\\dollapp目录下的内容

 tool.createAndShowDir("C:\\dollapp");

把“遍历循环”改为“遍历访问”：

 //遍历访问C:\\dollapp目录下的内容

 tool.createAndShowDir("C:\\dollapp");

## Page626，第19章，19.4.1节

本小节开头：

 每个具体的事件都是某种事件类的实例，事件类包括：ActionEvent、ItemEvent、MouseEvent、KeyEvent、FocusEven和WindowEvent等。……

把“FocusEven”改为“FocusEvent”：

 每个具体的事件都是某种事件类的实例，事件类包括：ActionEvent、ItemEvent、MouseEvent、KeyEvent、FocusEvent和WindowEvent等。……

## Page636，第19章，表19-4

在“ContainerListener”一行后面增加一行，修改后的表为：

|  |  |
| --- | --- |
| 监听接口 | 适配器 |
| ComponentListener | ComponentAdapter |
| FocusListener | FocusAdapter |
| KeyListener | KeyAdapter |
| MouseListener | MouseAdapter |
| MouseMotionListener | MouseMotionAdapter |
| ContainerListener | ContainerAdapter |
| WindowListener | WindowAdapter |
| ItemListener | 没有适配器 |
| ActionListener | 没有适配器 |
| InputMethodListener | 没有适配器 |

## Page669,第20章，20.9节

修改图20-13,修改后的图为：



## Page669,第20章，20.9节

在当前页末尾一段正文前再增加一段正文。以下绿色字体是增加的正文，黑色字体是书中的原文。

JMenu类继承了JMenuItem类，所以，一个菜单可以像菜单项一样，加入到另个一个菜单中，这样就形成了多级菜单。

JMenuItem有两个子类：JRadioButtonMenuItem和JCheckBoxMenuItem，它们分别表示单选菜单项和复选菜单项。当用户选择了某个菜单项，就会触发一个ActionEvent事件，该事件由ActionListener负责处理。

## 本书的反面封面

把原来的“艺术资讯”二维码改为本书的技术支持网址的二维码：



扫一扫访问本书的更多技术支持信息