# 第十一章 继承和多态

## 11.1引言

1.要点提示：面向对象的编程允许你从已经存在的类中定义新的类，这称为“继承”。

2.继承是Java在软件重用方面一个重要且功能强大的特征。假设要定义一个类，对圆、矩形和三角形建模。这些类有很多共同特征。设计这些类来避免冗余并使系统更易理解和易于维护的最好方式是什么？答案就是使用继承。

## 11.2父类和子类

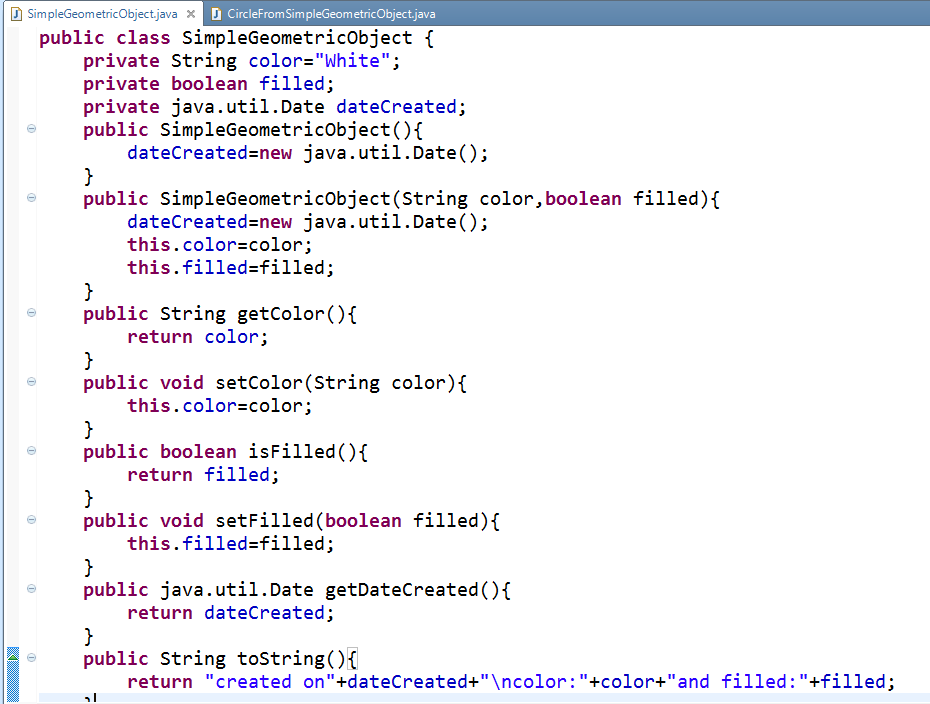
1.要点提示：继承使得你可以定义一个通用的类（即父类），之后扩充该类为一个更加特定的类（即子类）。

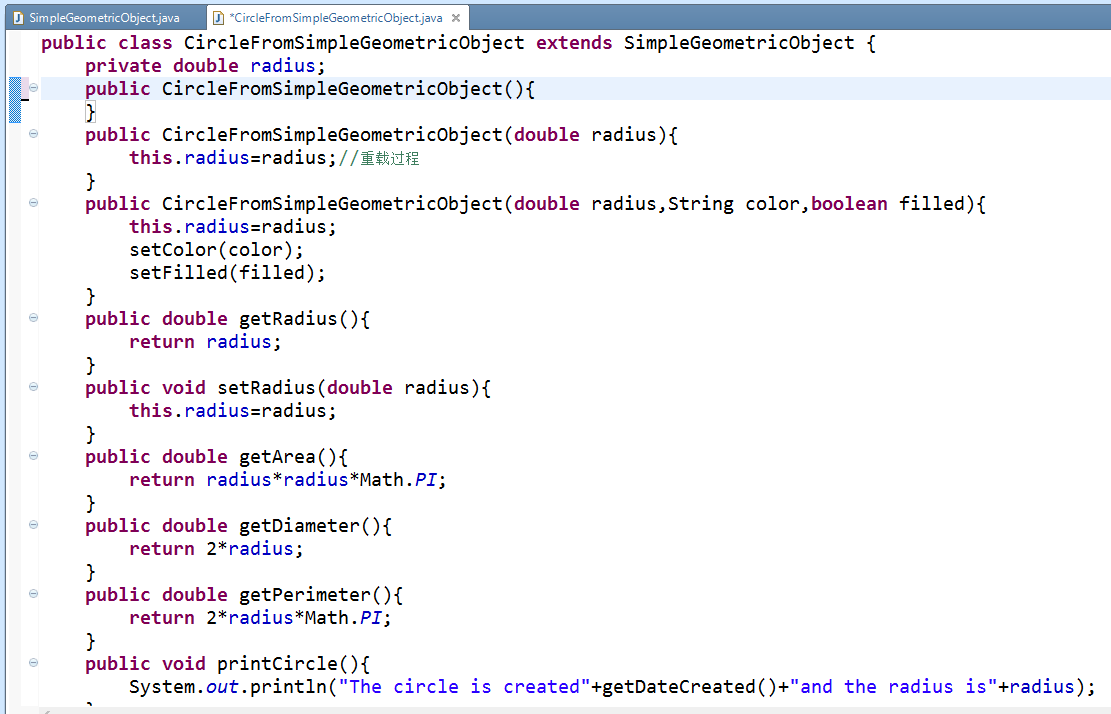
2.toString( )方法返回代表该对象的字符串。

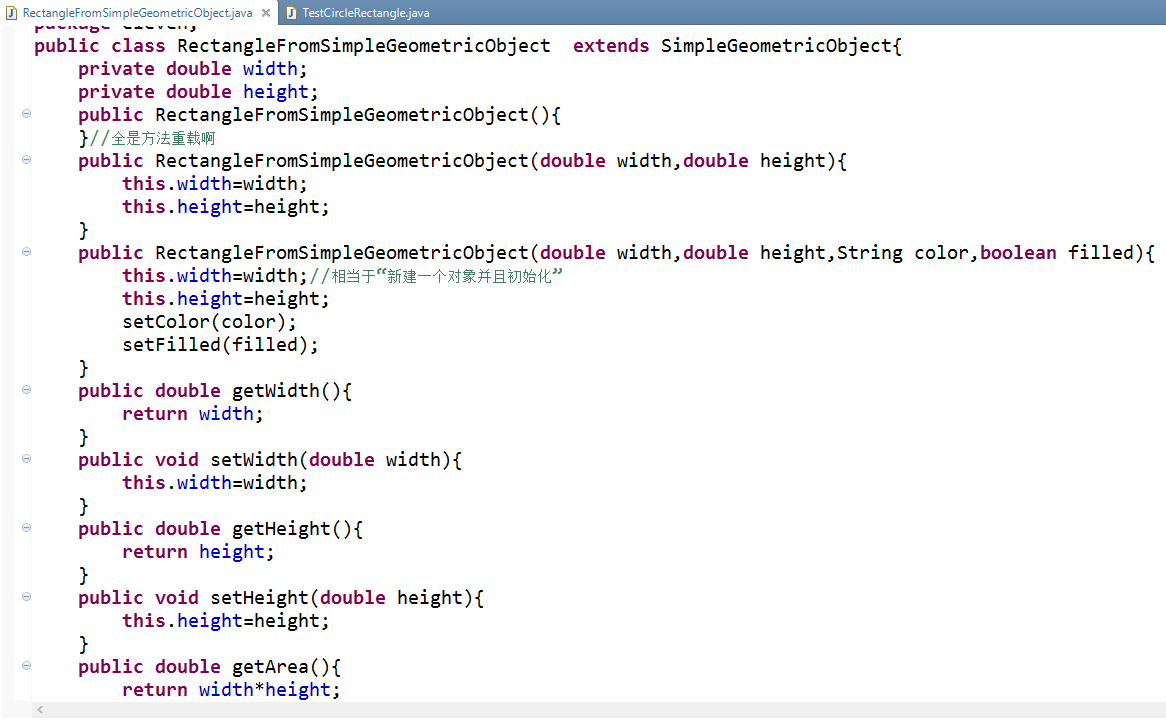
3.在Java术语中，如果C1扩展自另一个类C2，那么就将C1称为次类，将C2称为超类。超类也称为父类或基类，次类又称为子类、扩展类或派生类。子类从它的父类中继承可以访问的数据域和方法，还可以添加新的数据域和方法。

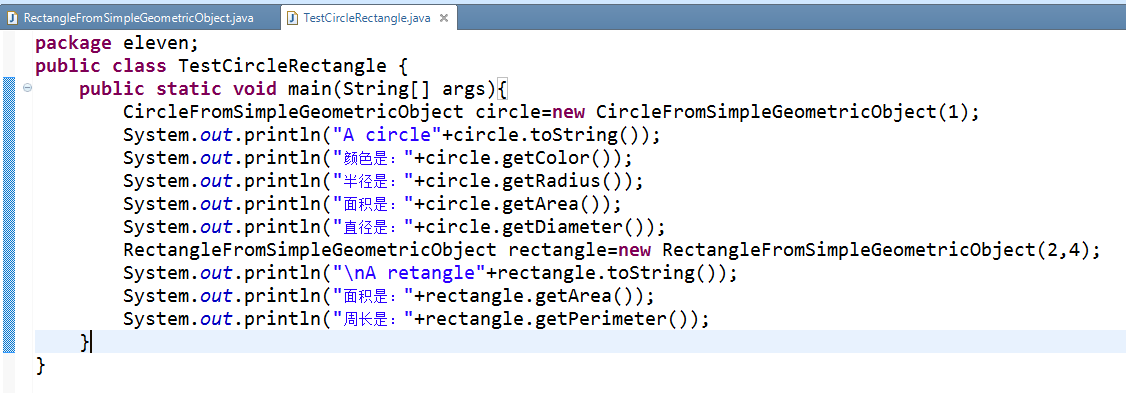
4.避免命名冲突最好的方法是将类放在不同的包里。

5.以下是类的继承的实例：（自己看看吧，也就那样。其中个别的原理我都备注出来了）









6.关于类的继承的几个关键点：

第一：和传统的理解不同，子类并不是父类的一个子集。实际上，一个子类通常比它的父类包含更多的信息和方法。

第二：父类之中的私有数据在该类之外是不能访问的。因此不能在子类中直接应用。但是，如果父类中如果定义了公共的访问器/修改器（get、set方法），那么则可以通过这些访问器/修改器来修改它们。

第三：继承是用来为“是一种（is-a）”关系建模的，但是不是所有“是一种”关系都需要用到继承。

第四：某些程序设计语言是允许从几个类派生出一个子类的，这种能力称为“多重继承”。但是Java中不允许多重继承。一个Java类只可能直接继承自一个父类，这种限制称为“单一继承”。如果使用extends关键字来定义一个子类，它只允许有一个父类。然而，多重继承是可以通过接口来实现的。

## 11.3使用super关键字

1.要点提示：关键字super代指父类，可以用于调用父类中的普通方法和构造方法。

2.关键字this的作用，它是对调用对象的引用。

关键字super是指这个super关键字所在类的父类。

3.关键字super可以用于两种途径：

（1）调用父类的构造方法；

（2）调用父类的方法。

### 11.3.1调用父类的构造方法

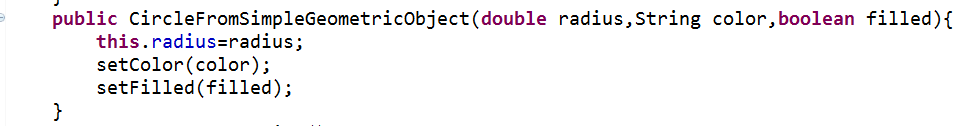
1.构造方法用于构建一个类的实例。不同于属性和普通方法，父类的构造方法不会被子类继承。它们只能使用关键字super从子类的构造方法中调用。

2.调用父类构造方法的语法是：

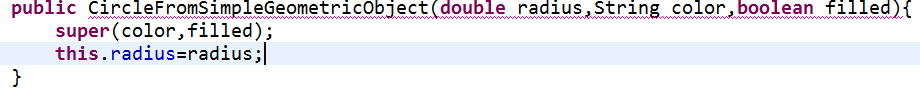
super 或者 super(parameter);

语句super( )调用父类的无参构造方法，而语句super(arguments)调用与参数匹配的父类的构造方法。

语句super( )和super(arguments)必须出现在子类构造方法的第一行，这是显示调用父类构造方法的唯一方式。



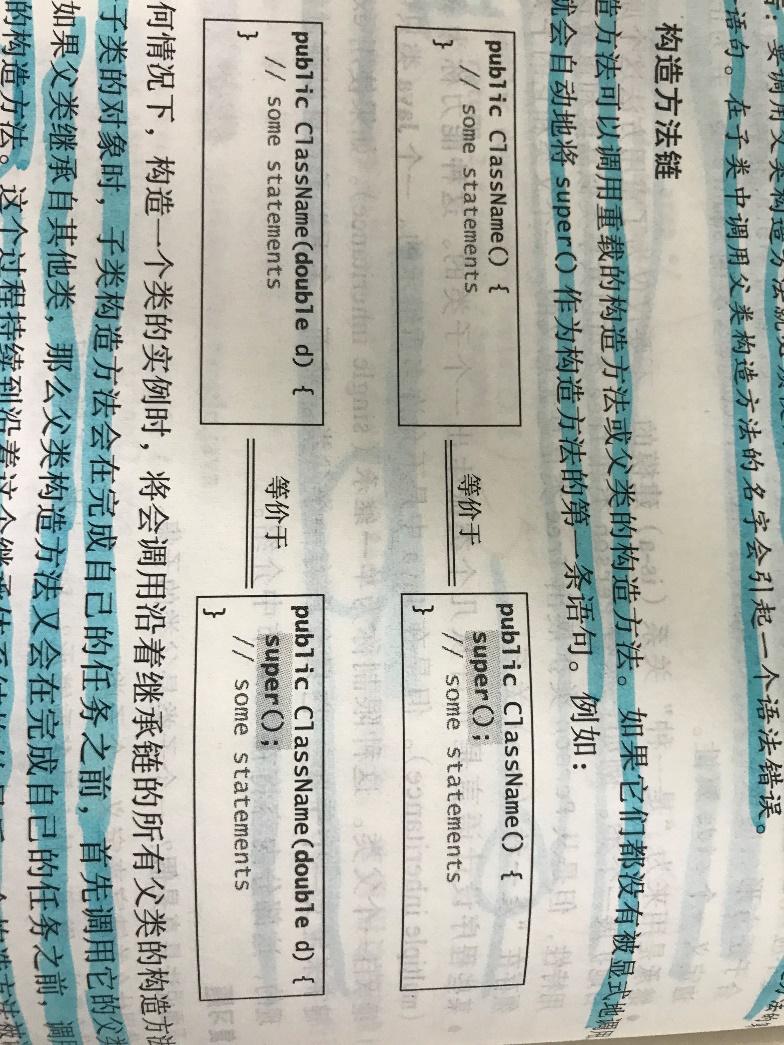
可以替换为：



3.警告：要构造父类的构造方法就必须使用关键字super，而且这个调用必须是构造方法的第一条语句。在子类中调用父类的构造方法的名字会引起一个语法错误。

### 11.3.2构造方法链

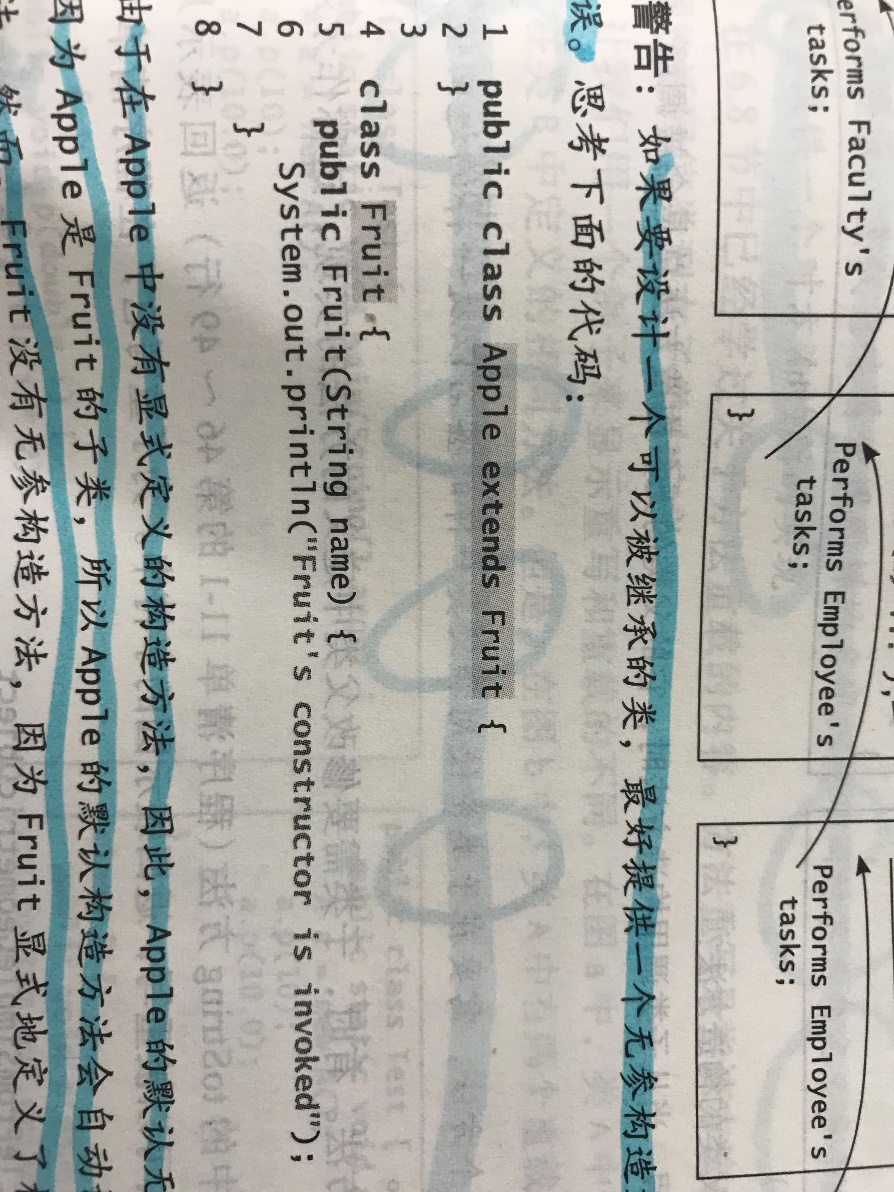
1.构造方法可以调用重载的构造方法或父类的构造方法。如果它们都没有被显示的调用，编译器就会自动的将super（）作为构造方法的第一条语句。例如：



2.当构造一个子类的对象时，子类的构造方法会在完成自己的任务之前，首先调用它的父类的构造方法。如果父类继承自其它类，那么父类的构造方法又会在完成自己的任务之前，调用它自己父类的构造方法。

这个过程持续到沿着这个继承体系结构的最后一个构造方法被调用为止，这个就是“构造方法链”。

3.警告：如果要设计一个可以被继承的类，最好提供一个无参构造方法以避免程序设计错误。思考下面的代码：



由于在Apple中没有显式定义的构造方法，因此，Apple的默认无参构造方法被隐式调用。因为Apple是Fruit的子类，所以Apple的默认构造方法会自动调用Fruit的无参构造方法。然而，Fruit没有无参构造方法，因为Fruit显式地定义了构造方法。因此，程序不能被成功编译。

4.设计指南：一般情况下，最好能为每个类提供一个无参构造方法，以便于对该类进行扩展，同时避免出错。

### 11.3.3调用父类的方法

1.关键字super不仅可以引用父类的构造方法，也可以引用父类的方法。所用语法如下：

Super.方法名（参数）；

## 11.4方法重写

1.要点提示：要重写一个方法，需要在子类中使用和父类一样的签名以及一样的返回值类型来对该方法进行定义。

2.以下几点值得注意：

第一，仅当实例方法是可访问时，它才能被覆盖。因为私有方法在它的类本身以外是不能访问的，所以它不能被覆盖。如果子类中定义的方法在父类中是私有的，那么这两个方法完全没有关系。

第二，与实例方法一样，静态方法也能被继承。但是静态方法不能被覆盖。

## 11.5方法重写与重载

1.方法重写与重载的不同

自己理解：方法重载只是参数不同，方法名是相同的；方法重写是在子类中进行的，方法名、参数以及返回值都是与重写方法一致的。

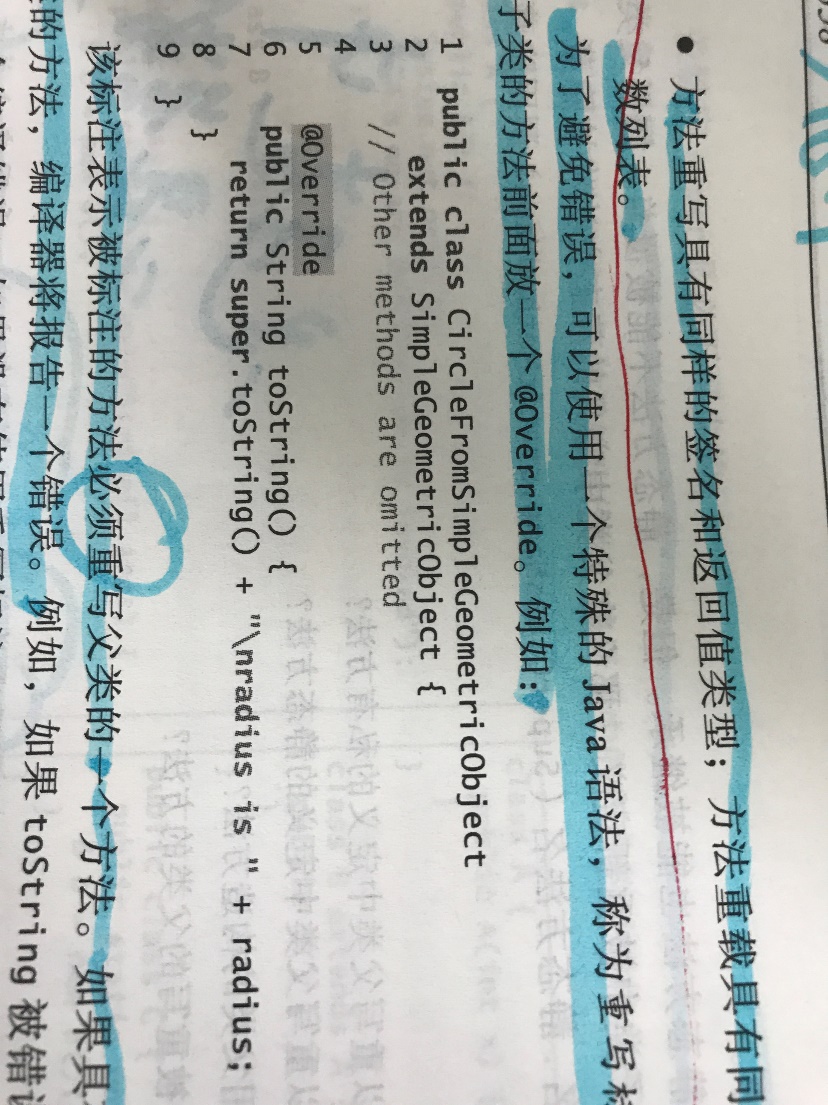
课本官方说明：重载意味着使用相同的名字但是不同的签名来定义多个方法。重写意味着在子类中提供一个对方法的新的实现。

2.注意以下问题：

第一，方法重写发生在通过继承而相关的不同类中；方法重载可以发生在同一个类中，也可以发生在由于继承而相关的不同类中。

第二，方法重写具有同样的签名和返回值类型；方法重载具有相同的名字，但是不同的参数列表。

3.为了避免错误，可以使用一个特殊的Java语法，称为**“重写标注”**，在子类的方法面前放一个@Override。例如：



该标注表示被标注的方法必须重写父类的一个方法。如果具有该标注的方法没有重写父类的方法，编译器将报告一个错误。

## 11.6 Object类以及toString( )方法

1.要点提示：Java中所有的类都继承自java.lang.Object类。

如果在定义一个类时没有指定其继承性，那么这个类的父类就被默认为是Object。

2.调用一个对象的toString（）会返回一个描述该对象的字符串。默认情况下，它返回一个由该对象所属的类名、@符号以及十六进制形式的内存地址组成的字符串。例如：

Loan loan =new Loan( )；

System.out.println(loan.toString( ))；

这些代码会显示像Loan@15037e5这样的字符串。这个信息不是很有用，或者说没有什么信息量。通常，应该重写这个toString方法，这样，它可以返回一个代表该对象的描述性字符串。

3.注意：也可以传递一个对象来调用System.out.print(object)，这相当于调用System.out.print(object.toString( ))。

## 11.7多态？？（不太懂）

1.要点提示：多态意味着父类的变量可以指向子类对象。

2.面向对象程序设计的三大支柱是封装、继承和多态。

3.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，而父类定义的类型称为父类型。

4.继承关系使得一个子类继承父类的特性，并且附加一些新的特性。子类是它的父类的特殊化，每个子类的实例都是其父类的实例，但是反过来是不成立的。例如：每个圆都是一个几何对象，但并非每个集合对象都是圆。因此，总可以将子类的实例传给需要的父类型的参数。

**5.使用父类对象的地方都可以使用子类的对象，这就是所说的多态（它源于希腊文，多种样式的意思）。简单来说，多态意味着父类型的变量可以引用子类型的对象。**

## 11.8动态绑定（解决方法重写过程中的调用顺序问题）

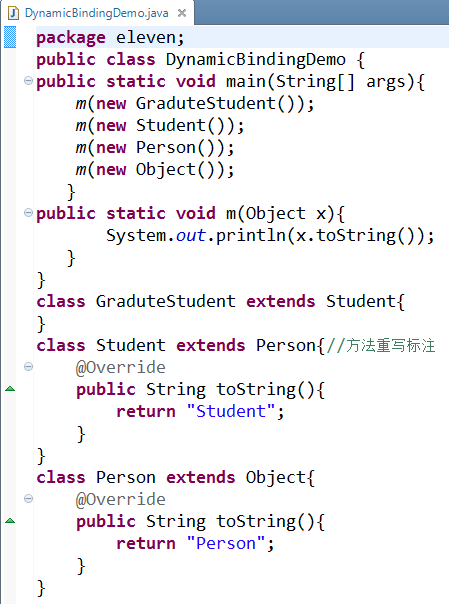
1.要点提示：方法可以在沿着继承链的多个类中实现。JVM决定运行时调用哪个方法。

2.动态绑定解决的问题？

方法可以在父类中定义，并且可以在子类中重写。如此多重写的方法，调用顺序是怎样的呢？动态绑定就是解决它们之间的调用顺序的问题。

3.动态绑定的机制？

动态绑定的工作机制如下：假设对象o是类C1、C2、…CN的实例，其中C1是C2的子类，C2是C3的子类，也就是说，CN是最通用的类，C1是最特殊的类。在Java中CN就相当于Object类。如果o调用一个方法，那就从C1开始，逐步在其父类中找方法，直到找到为止。



## 11.9对象转换和instanceof运算符

1.自己理解的对象转换：对象类型的转换。

2.对象的引用 可以类型转换为 对另一种对象的引用，这称为对象转换。

3.语句 Object o=new Student( )是合法的，它称为隐式转换。

4.假设想使用下面的语句把对象引用o赋值给Student类型的变量：Student b=o

在这种情况下会发生编译错误，因为Student对象总是Object的实例，但是Object对象不一定是Student的实例。为了告诉编译器o就是一个Student对象，就要使用显示转换。

5.显示转换的语法与基本类型转换的语法很相似，用圆括号把目标对象的类型括住，然后放到要转换的对象前面，如下所示：

Student b=(Student) o ;

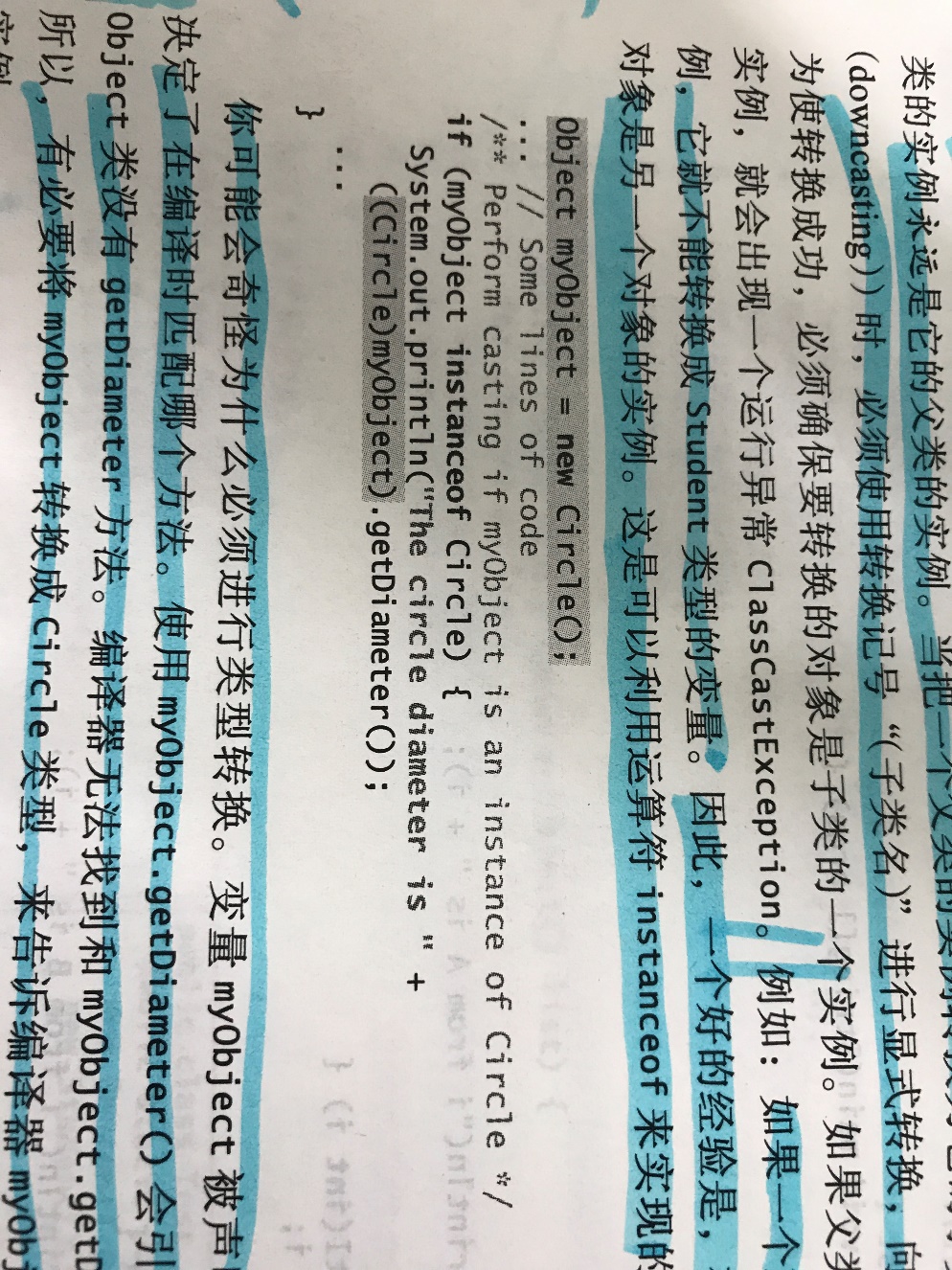
6.向上转换/向下转换+代码及其解释

向上转换：总是可以将一个子类的实例转换成一个父类的变量，称为向上转换（因为子类的实例永远是其父类的实例）。

向下转换：当把一个父类的实例转化为它的子类变量（向下转换）时，必须使用转换记号“（子类名）”进行显示转换。

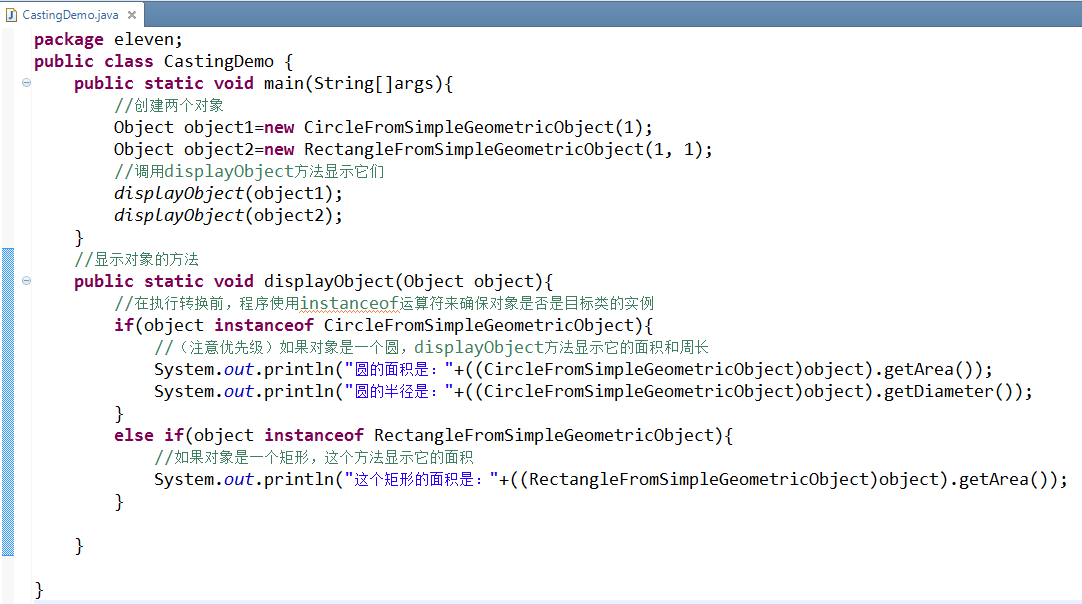
7.instanceof运算符

例如：如果一个对象不是Student的实例，它就不能转换成Student类型的变量。因此，一个好的经验是，在转换之前确保该对象是另一个对象的实例。这里可以利用运算符instanceof来实现：



解释：你可能会奇怪为什么必须进行类型转换。变量myObject被声明为Object。声明类型决定了在编译时匹配哪个方法。使用myObject.getDiameter（）会引起一个编译错误，因为Object类没有getDiameter方法。编译器无法找到和

myObject.getDiameter( )匹配的方法。所以，有必要将myObject转换成Circle类型，告诉编译器myObject也是Circle的一个实例。

8. 

优先级说明：对象成员访问运算符（.）优先于类型转换运算符。使用圆括号保证在点运算符（.）之前进行转换，例如：

（（Circle）object）.getArea( );

9.基本类型值转换 与 对象转换 的不同：

对基本类型值进行转换不同于对对象引用进行转换。转换基本类型值返回一个新的值；而转换一个对象不会创建一个对象。

## 11.10 Object类的equals方法

1.要点提示：如同toString( )方法，equals(Object)方法是定义在Object类中的另一个有用的方法，这个方法测试两个对象是否相等，调用它的语法是：

Object1.equals(object2);

2.Object类中equals方法的默认实现是：

public Boolean equals(Object obj) { return (this==obj) }

这个实现使用==运算符检测两个引用变量是否指向同一个对象。因此，应该在自己的客户类中重写这个方法，以测试两个不同的对象是否具有相同的内容。

3.equals方法在Java API的许多类中被重写，比如java.lang,String 和 Java.util.Date，用于比较两个对象的内容是否相等。

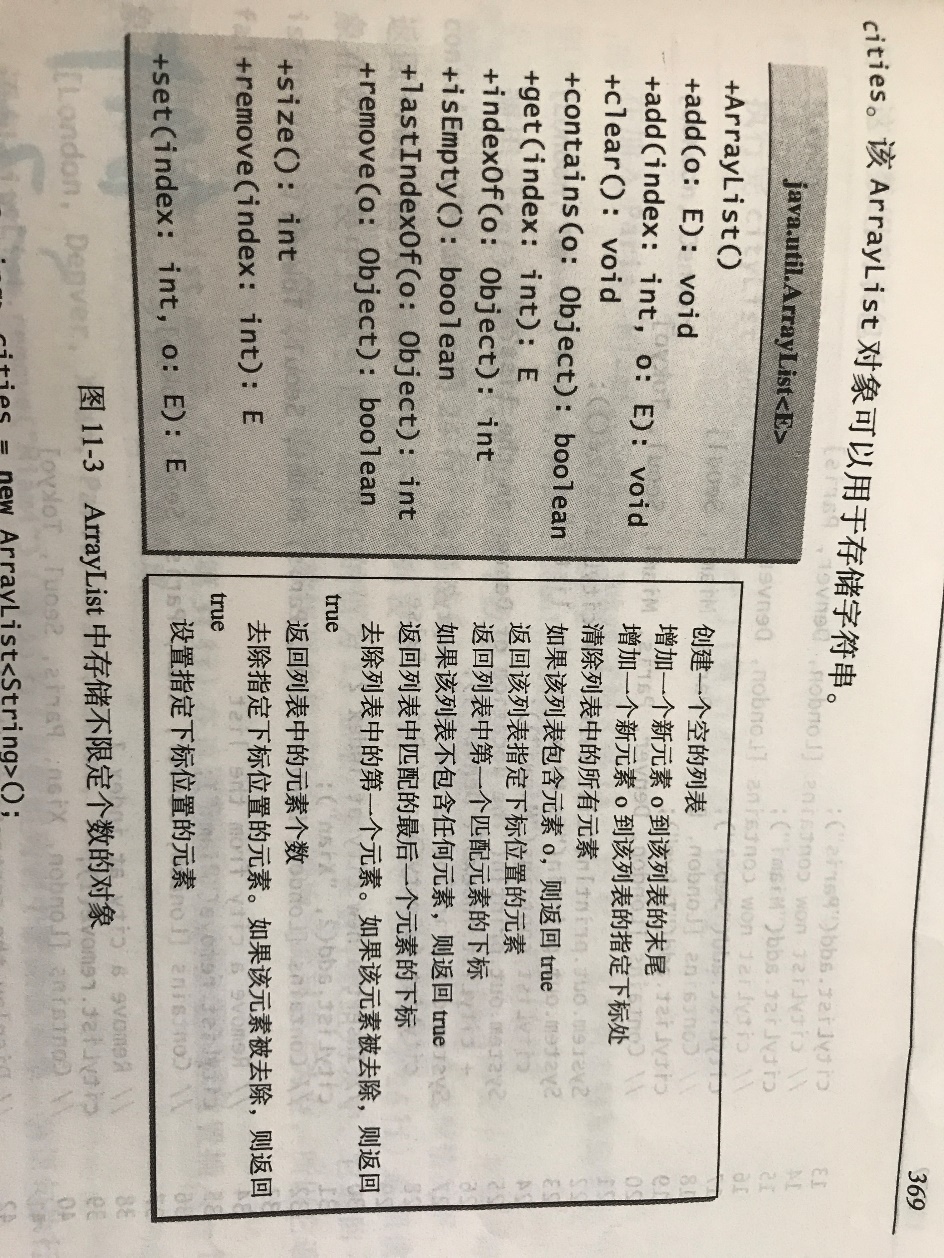
4.注意：

比较运算符==用来比较两个基本数据类型的值是否相等，或者判断两个对象是否具有相同的引用。如果想让equals方法能够判断两个对象是否具有相同的内容，可以在定义这些对象类时，重写Circle类中的equals方法。运算符==要比equals方法的功能强大些，因为==运算符可以检测到两个引用变量是否指向同一个对象。

## 11.11 ArrayList类

1.要点提示：ArrayList对象可以用于存储一个对象列表。

我们可以创建一个数组存储对象，但是这个数组一旦创建，它的大小就固定了。Java提供ArrayList类来存储不限定个数的对象。下图给出了ArrayList中的一些方法：



2.下面语句创建一个ArrayList并且将其引用赋值给变量dates。该ArrayList对象可以用于存储日期：

ArrayList<java.util.Date> dates=new ArrayList<java.util.Date>( );

从JDK1.7开始，该语句可以简化为：

ArrayList<java.util.Date> dates=new ArrayList< >( );

3.以下给出了一个使用ArrayList来存储对象的一个示例。**（这个例子很好的）**



4.数组和ArrayList的区别？

数组：一旦创建了一个数组，它的大小就确定下来了，我们可以访问数组中的元素，向数组中添加、插入和删除元素比较复杂。

数组列表：当创建ArrayList后，它的大小为0，而且向数组中添加、插入和删除元素是比较容易的（自带相关方法）。

5.数组和数组列表的排序问题？

数组排序方法：Java.util.Arrays.sort(array)

数组列表排序方法：Java.Util.Colllections.sort(arrayList)

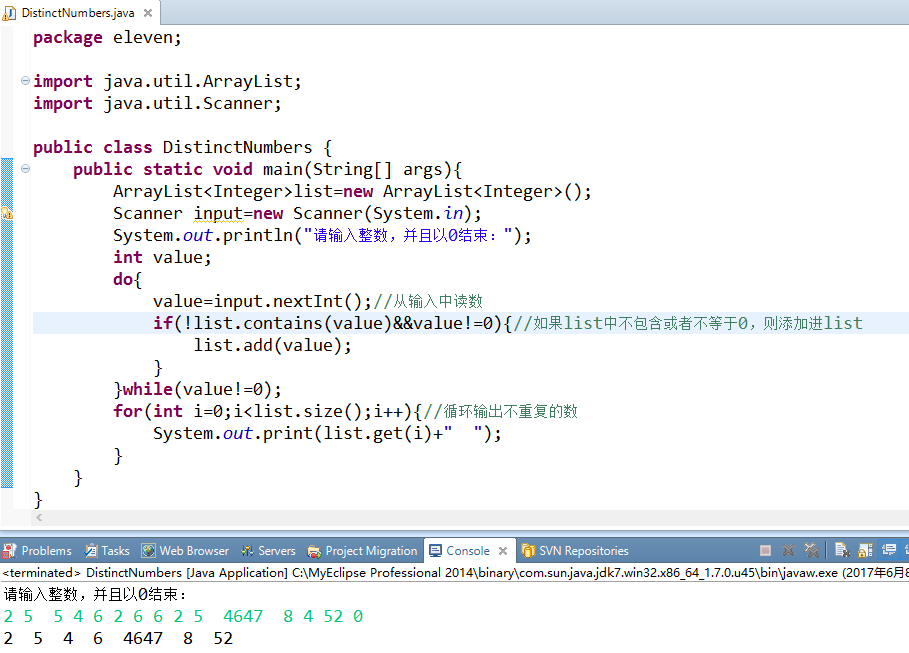
6.假设想创建一个用于存储整数的ArrayList，可以使用下面代码来创建一个列表吗？

ArrayList<int>list=new ArrayList<>( );

答案是不行的。这是因为存储在ArrayList中的元素必须是一中对象。不能使用int这种数据类型，但是你可以创建一个存储Integer对象的ArrayList，如图所示：

ArrayList<Integer> list=new ArrayList<>( );

下面是一个程序示例（提示用户输入一个数字序列，然后显示该序列中的不同数字）：



7.ArrayList的几个优点？

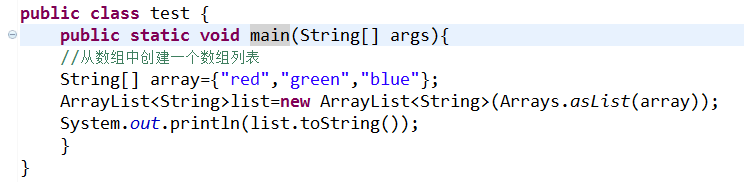
第一：ArrayList的大小是灵活的，所以无需提前定义它的大小。而当我们创建一个数组时，必须把大小给定它。

第二：ArrayList中包含许多有用的方法。比如可以使用contains方法来测试某个元素是否在列表中。如果使用数组，则需要编写额外的代码来实现该方法。

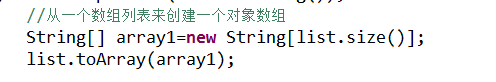
## 11.12对于列表有用的方法

1.要点提示：Java提供了方法，用于从数组创建列表、对列表排序、找到列表中最大和最小元素，以及打乱一个列表。

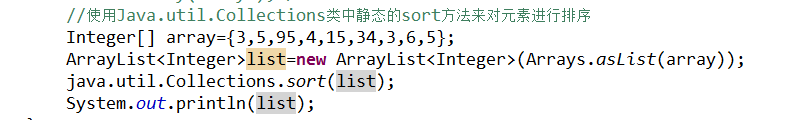
第一：从数组中创建一个数组列表



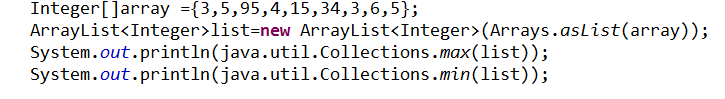
第二：从一个数组列表来创建一个对象数组



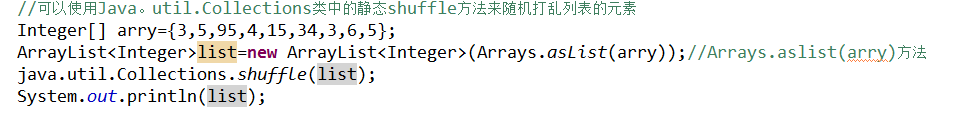
第三：如果列表中的元素是可比较的，比如整数、双精度浮点数或者字符串，则可以使用Java.util.Collections类中的静态的sort方法来对元素进行排序。



第四：可以使用Java.util.Collections类中静态的max和min方法来返回列表中的最大和最小元素。

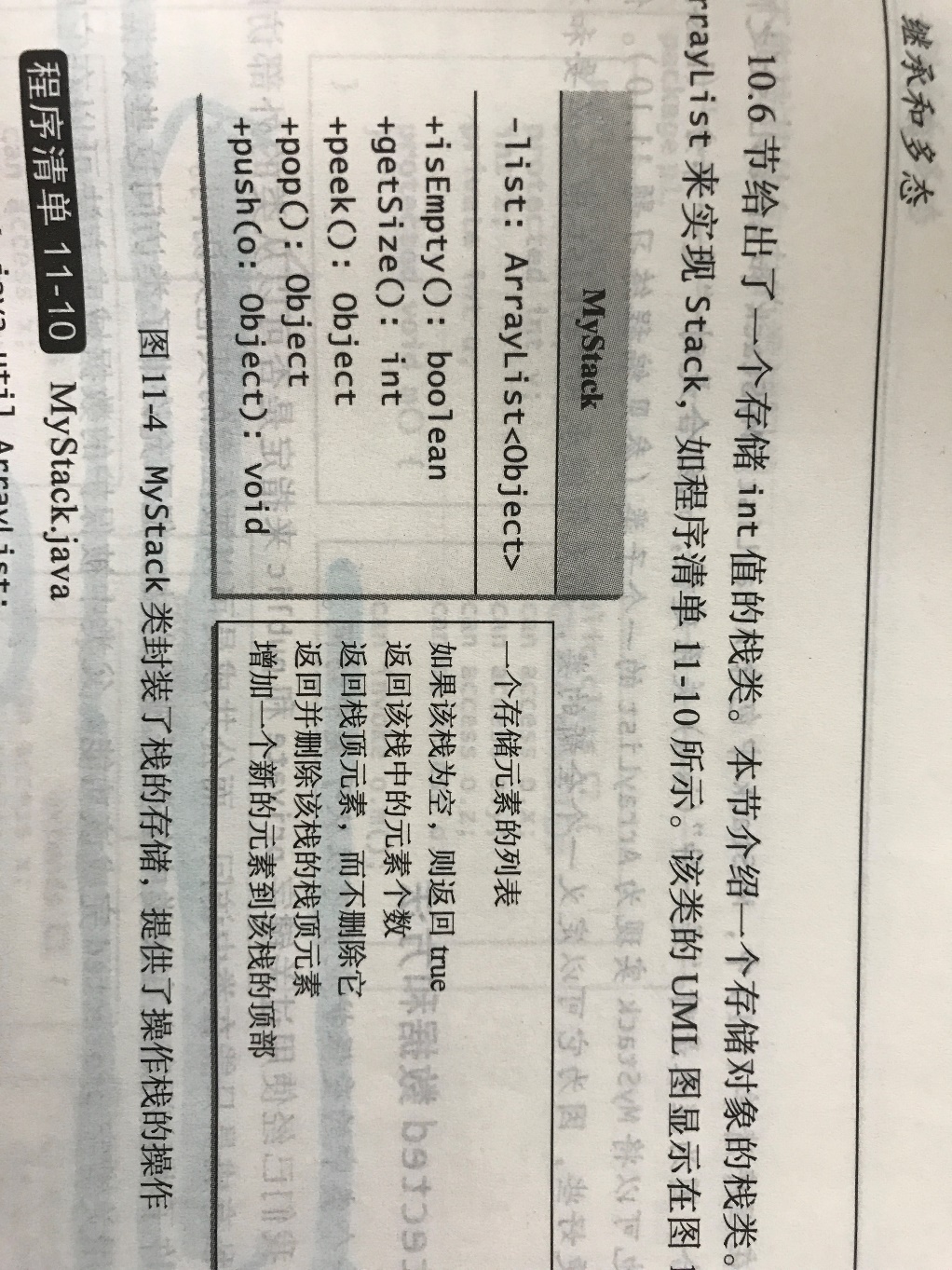


第五：可以使用Java.util.Collections类中的静态的shuffle方法来随机打乱列表的元素。



## 11.13示例学习：自定义栈类

1.要点提示：设计一个栈类，用于存放对象。如下是该类的UML图：





## 11.14 protected数据和方法

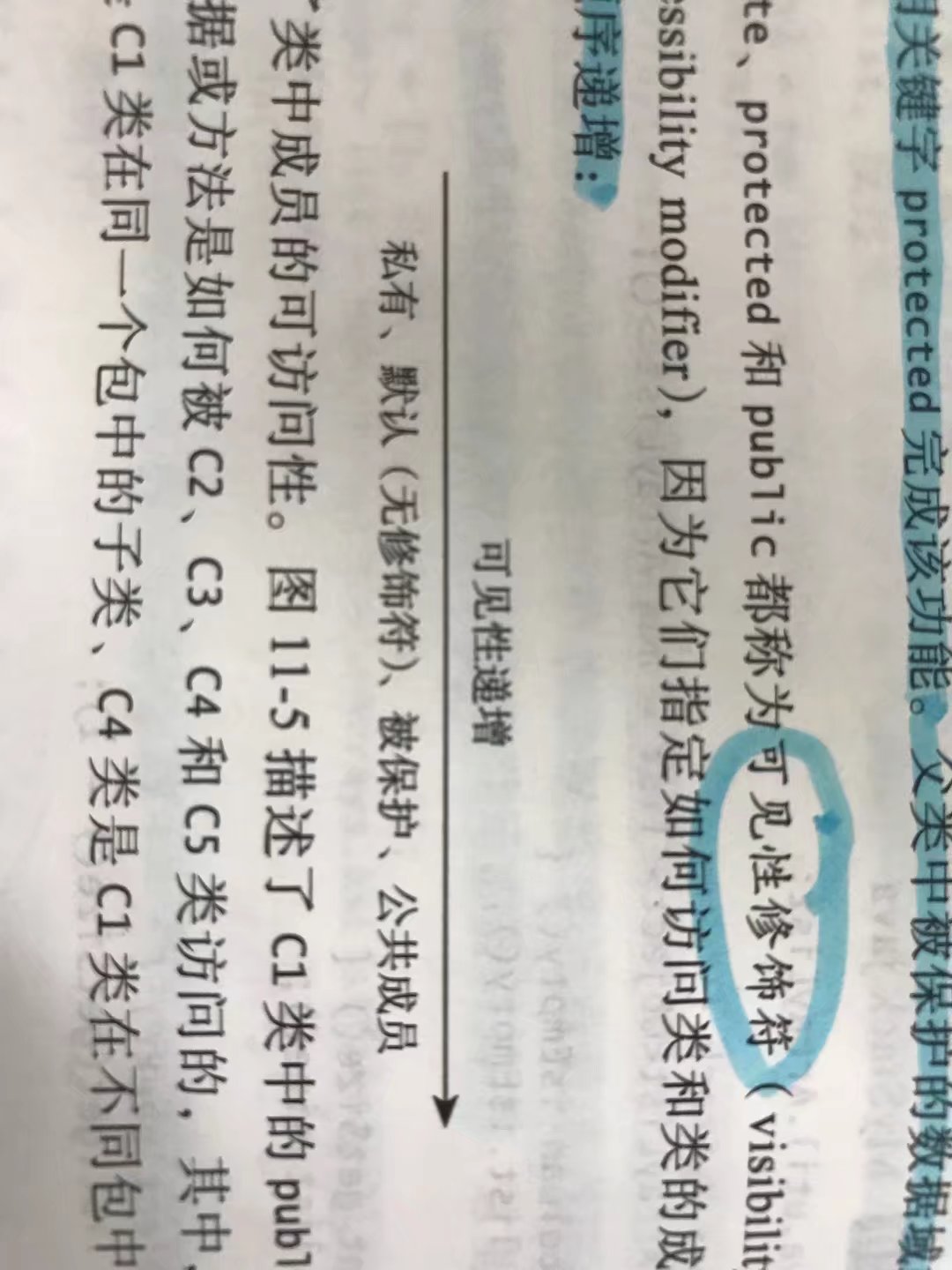
1.要点提示：一个类中的受保护成员可以从子类中访问。私有成员只能在类内访问，而公共成员可以被任意的其它类访问。

2.protected修饰符的适用范围：

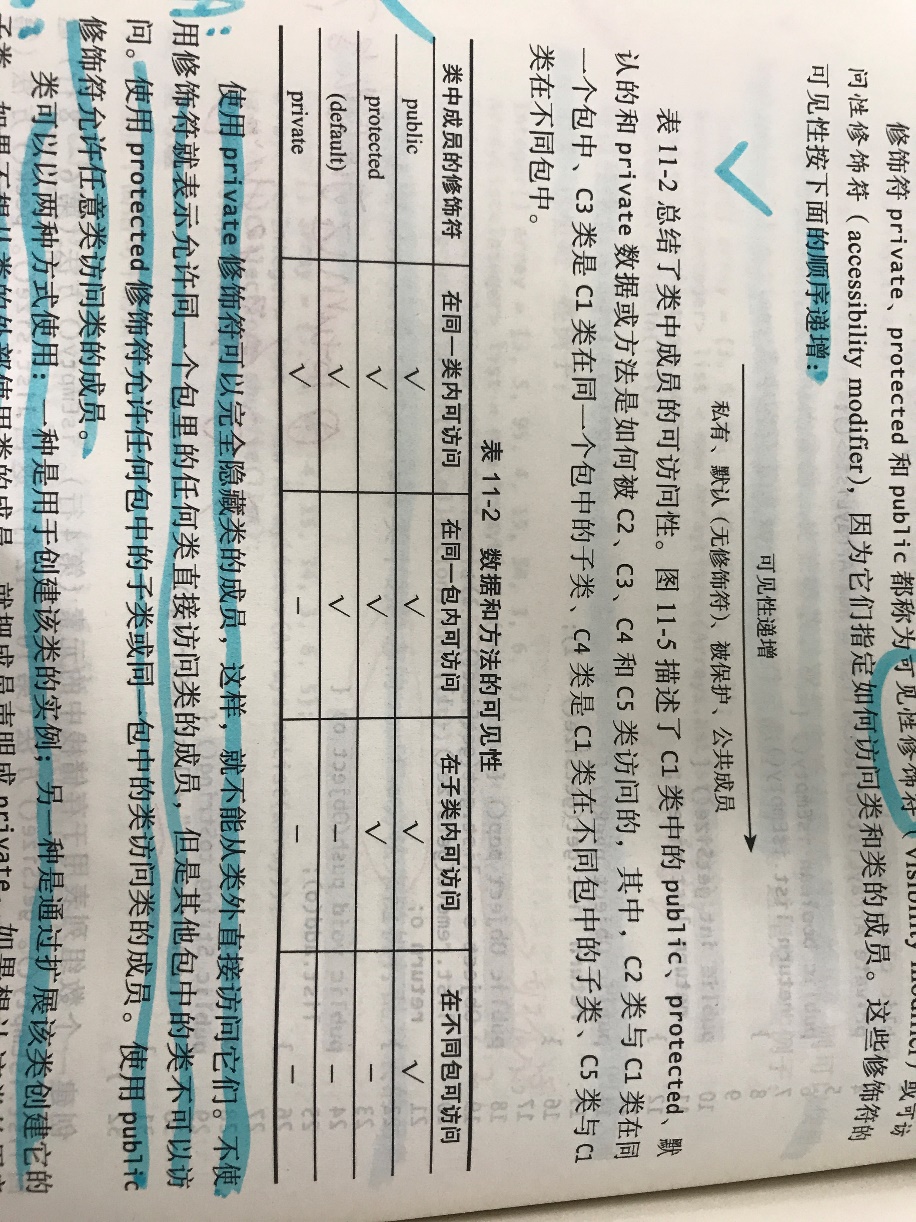
经常需要允许子类访问定义在父类中的数据域或方法，但不允许非子类访问这些数据域和方法。可以使用关键字protected来完成该功能。父类中被保护的数据域和方法可以在它的子类中访问。

3.可见修饰符的可见性顺序：

这些修饰符的可见性按一下顺序递增



4.数据和方法的可见性



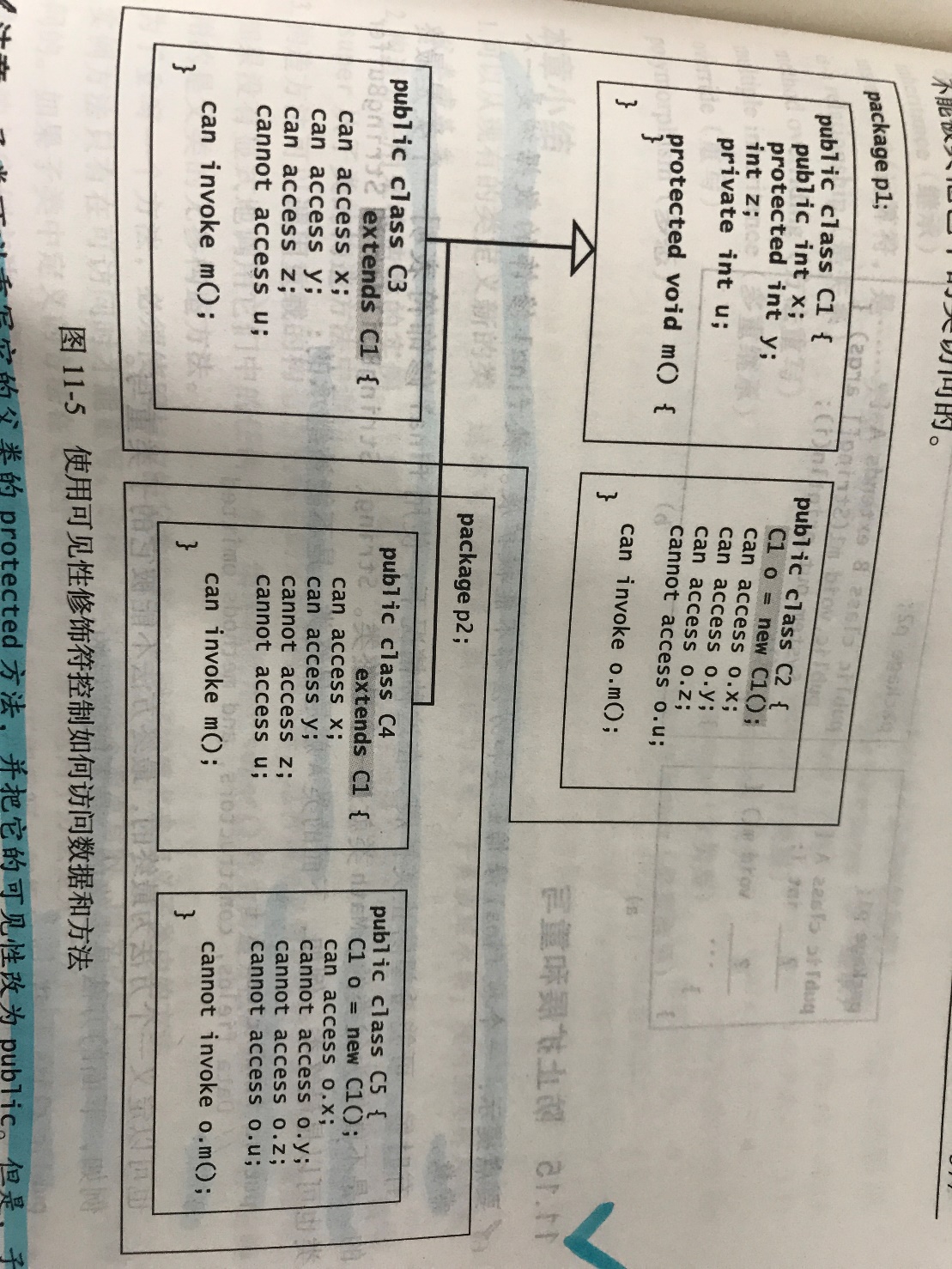
说明：使用private修饰符可以完全隐藏类成员，这样，就不能从类外直接访问它们；不使用修饰符（default）表示允许同一个包里的任何类直接访问类成员，但是其他包中的类不可以访问；使用protected修饰符允许任何包中的子类或同一包中的类访问类成员；使用public修饰符允许任意类访问类成员。

5.小结：

类可以以两种方式使用：一种是用于创建该类的实例；另一种是通过扩展该类创建它的子类。

如果不想从类的外部使用类的成员，就把成员声明为private。如果想让该类的用户都能使用类的成员，就成员声明为public。如果想让该类的扩展者使用数据和方法，而不想让该类的用户使用，则把成员声明成protected。

6.不同package、不同修饰符和继承关系下的访问权限



7.注意：子类可以重写父类的protected方法，并把它的可见性改为public。但是，子类不能削弱父类中定义的方法的可访问性。例如：如果一个方法在父类中定义的public，那么在子类中也必须定义为public（在子类中可以增强，不可以削弱。因为private不能被继承，所以只能说protected）。

## 11.15 防止扩展和重写

1.要点提示：一个被final修饰的类和方法都不能被扩展。被final修饰的数据域是一个常数。

2.有时候，可能希望防止类扩展。在这种情况下，使用final修饰符声明一个类是最终的，是不能作为父类的（不能被继承、不能扩展）。

例如，下面的类A是最终类，不能被继承

Public final class A { }

例如，下面的方法method1是最终版的，不能被重写

Public class Test{

Public final void method1(){}

}

3.注意：修饰符public、protected、private、static、abstarct以及final可以用在类以及类成员（数据和方法）上，只有final修饰符还可以用在方法中的局部变量上。方法内的最终局部变量就是常量。