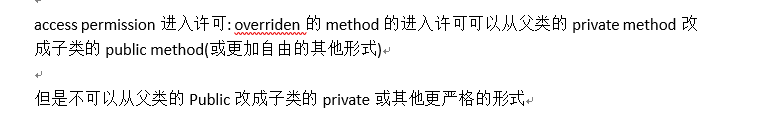
Ch7

Override





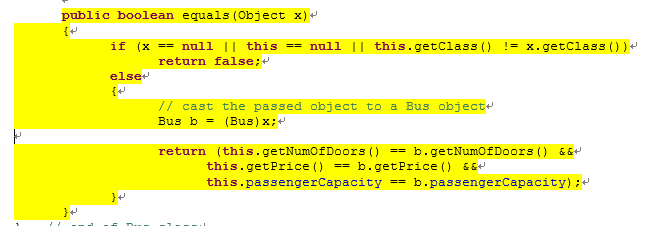


父类method private可以改成Public，不可以从父类Public改成子类private

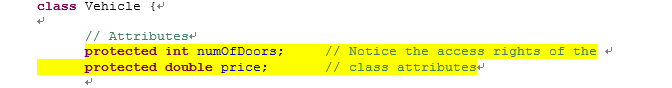
Overload是重载，是有相同的方法名，但参数类型或个数彼此不同  
Override是重写，是在子类与父类中，子类中的方法的方法名，参数个数、类型都与父类中的完全一样，在子类中覆盖掉了父类的改方法。

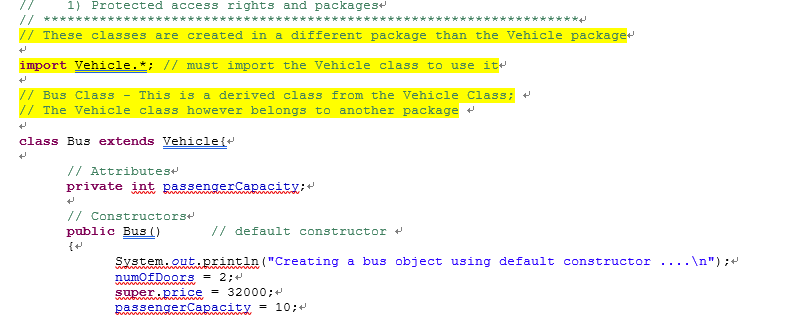
如果父类method之前有个final，那么子类就不能将他重写

如果父类class之前有个final，那么这个class就不能作为父类存在，不能有继承

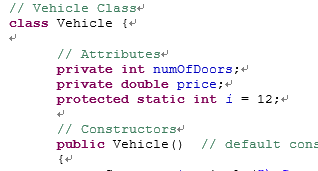


直接用的关系





如果父类是private

.

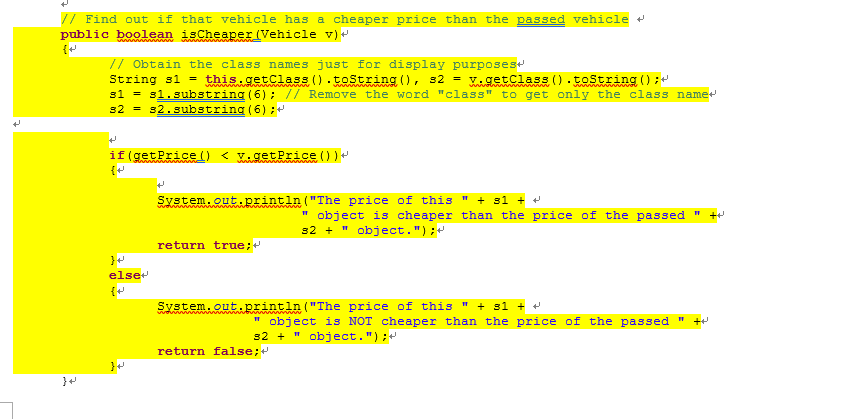
子类就必须要用get



因为get set方法毕竟是public的·

Java对所有method都是用late binding（除了private final static三种前缀·）

多态



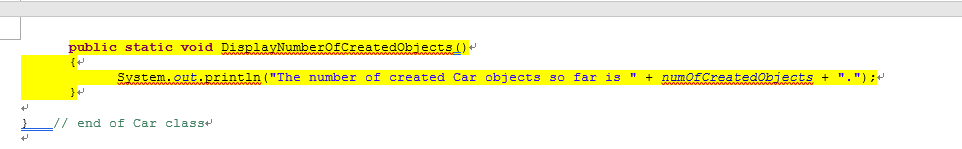
比如说这个getPrice，按说两个都是Vehicle，会用vehicle的getprice

但实际上他们用的是他们所属class的getprice

由于后期绑定，可以在base class中编写一个方法来执行任务，即使该任务的某些部分还没有定义。

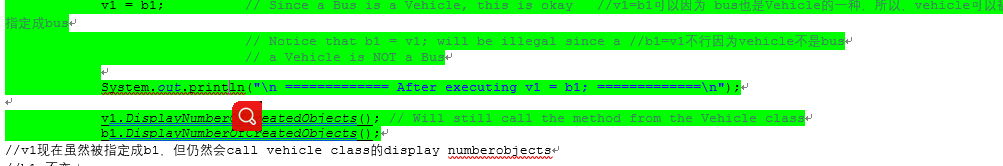
换句话说，这个isCheaper使用的getprice取决于参数的情况而不是制定好的

静态联编，比如static



是编译好的，你只能使用你这个class的displaynumberofcreatedobjecta，是制定好的

而后期绑定原来并没有指定是谁的·getprice，取决于你运行时是谁



V1=B1所以现在V1指向B1

当然不能b1=v1，因为v1不见得就是b1

Display，V1仍用的是自己的display

Vehicle数量不变，维度区别是BUS数量+1;

所以对于静态方法，取决于call他的variable name，而不是里面的object，v1天生就指定好了vehicle

在描述一个非static method里，可以直接引用static method而不用加object.xxmethod 或xxclass.xxmethod

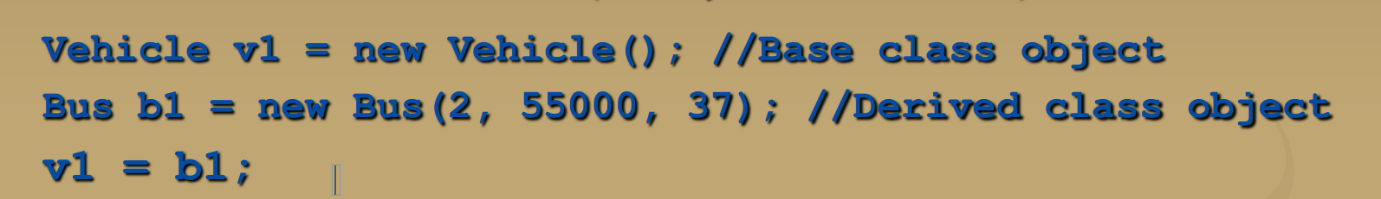
实际上他有一个隐藏的this,是this.xxstaticmethod

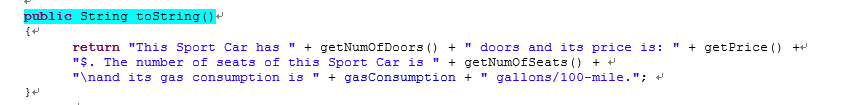
Final:final method不可以被子类的method overridden

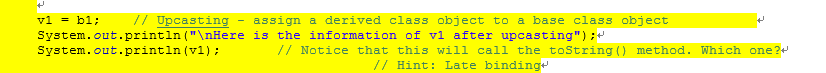
如果class加了final，那么这个class就不能作为base class(没有人继承他)

Upcasting 向上转型

把子类的Object赋值给一个父类或祖类的variable





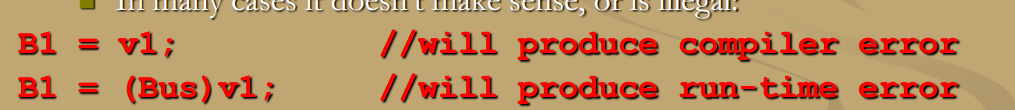


这里实际上会type b1的数据，因为没加static



非static取决于实际Object

因为V1不见得是B1

.

Downcasting

用type-cast （比如说（int）5.55这种），让子类object变成父类类型，再赋值

**class** RaceCar **extends** SportCar{

sc1 = (SportCar)rc2; // Will this explicit casting make a difference? Why? Why not?

System.*out*.println("\nHere is the information of sc1 after explicit casting");

System.*out*.println(sc1); // This will also call the toString() method. Which one?

运行结果是

Here is the information of sc1 after explicit casting

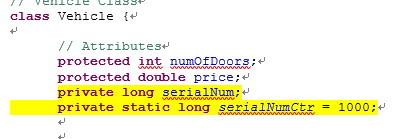
Executing getPrice() from the RaceCar class. The price is 85000.0$.

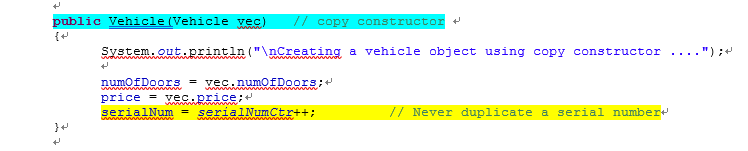
This Race Car has 3 doors and its price is: 85000.0$. The number of seats of this Race Car is 4

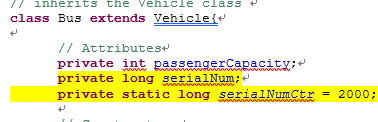
and its gas consumption is 4.0 gallons/100-mile.The horse power of this Race Car is: 450.

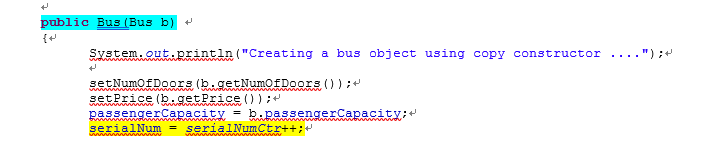
毕竟这些都不是static，所以根据实际object，rc2

当使用了polymorphism的时候，copy constructor可能会有些限制。









不同的Class,有不一样的serialnumctr

Deep copy object

所以我们用clone method其实更好，他继承自Object class

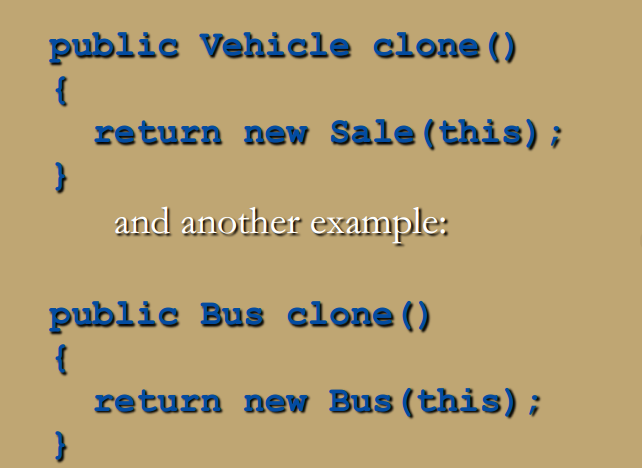
Clone没有参数，会return一个calling object的deep copy，callingobject.clone();

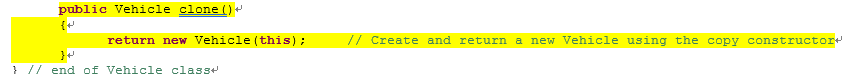
但是我们不希望直接用它，而是override一遍使其更好使用

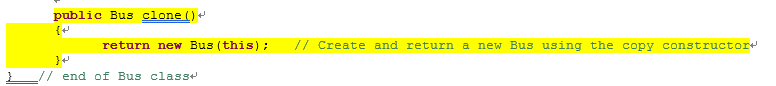
原来的写法



我们override以后的写法



.



一种好的copy array写法

// A method that takes an array of Vehicles inventory and return a copy of that array

**public** **static** Vehicle[] copyInventory\_3(Vehicle[] va)

{

// This is the correct version of this method, which uses the clone() method instead of

// the copy constructors

Vehicle[] vecarr = **new** Vehicle[va.length]; // create a new array with the same length

// as the passed array;

**for** (**int** i = 0; i < vecarr.length; i++) // then copy it

{

vecarr[i] = va[i].clone();

}

**return** vecarr;

}

Vehicle[] vecInv3 = *copyInventory\_3*(vecInv1);

完美copy

**public** Vehicle(Vehicle vec) // copy constructor

{

System.*out*.println("\nCreating a vehicle object using copy constructor ....");

numOfDoors = vec.numOfDoors;

price = vec.price;

serialNum = *serialNumCtr*++; // Never duplicate a serial number

}

一点不变好吧

Abstract class:不创造任何object 的class，只是用来derive别的class(作为别的class的父类)

一个abstract class必须有一个abstract method

Abstract method是不可以具体描述的（没有method body），在括号之后直接分号而不是花括号



Abstract method前缀不可以是private，不然别人压根没法继承

总结的来说就和原来一样，但是前缀改了，要加个abstract，然后抽象的method要加上，method至少一条

**abstract** **class** Vehicle {

// Attributes

**protected** **int** numOfDoors;

**protected** **double** price;

**private** **long** serialNum;

**private** **static** **long** *serialNumCtr* = 1000;

// Constructors

**public** Vehicle() // default constructor

{

System.*out*.println("\nCreating a vehicle object using default constructor ....");

numOfDoors = 4;

price = 10000;

serialNum = *serialNumCtr*++;

}

**public** Vehicle(**int** nd, **double** pr)

{

System.*out*.println("\nCreating a vehicle object using parameterized constructor ....");

numOfDoors = nd;

price = pr;

serialNum = *serialNumCtr*++;

}

**public** Vehicle(Vehicle vec) // copy constructor

{

System.*out*.println("\nCreating a vehicle object using copy constructor ....");

numOfDoors = vec.numOfDoors;

price = vec.price;

serialNum = *serialNumCtr*++; // Never duplicate a serial number

}

**public** **int** getNumOfDoors()

{

**return** numOfDoors;

}

**public** **void** setNumOfDoors(**int** nd)

{

numOfDoors = nd;

}

**public** **double** getPrice()

{

**return** price;

}

**public** **void** setPrice(**double** pr)

{

price = pr;

}

// Find out if that vehicle has a cheaper price than the passed vehicle

**public** **boolean** isCheaper(Vehicle v)

{

// Obtain the class names just for display purposes

String s1 = **this**.getClass().toString(), s2 = v.getClass().toString();

s1 = s1.substring(6); // Remove the word "class" to get only the class name

s2 = s2.substring(6);

**if**(getPrice() < v.getPrice())

{

System.*out*.println("The price of this " + s1 +

" object is cheaper than the price of the passed " +

s2 + " object.");

**return** **true**;

}

**else**

{

System.*out*.println("The price of this " + s1 +

" object is NOT cheaper than the price of the passed " +

s2 + " object.");

**return** **false**;

}

}

// Some abstract methods that MUST be defined by derived classes

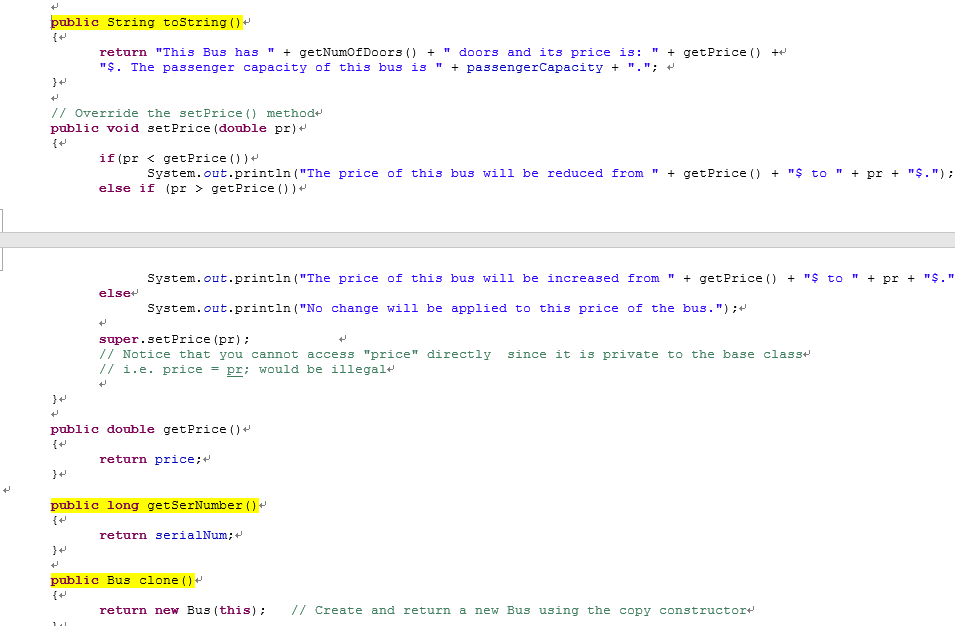
**abstract** **public** String toString();

// abstract can also be specified as follows

**public** **abstract** **long** getSerNumber();

**abstract** **public** Vehicle clone();

子类中



Abstract了几个就要用几个，用的时候无abstract前缀

没abstract method 的class叫做 concrete class具体类

Abstract不能创造object，他们的constructor只是为了子类super用

虽然abstract不能创造object，但是可以创造一个abstract class type的参数

**class** xxException **extends** Exception{

**public** xxException() {

**super**("xxException");

}

**public** xxException(String e) {

**super**(e);

}

**public** String getMessage\*({)

**return** **super**.getMessage()

}

}

Try{

Throw new xxexception();  
}

Catch(Exception e){

String s=e.getmessage();

System.out.println s

}

**public** **static** **double** divide(**double** x, **double** y) **throws** Exception //throws exception说明这个Method 所throw的Exception 是基础的exception类的

{

**double** result;

System.*out*.println("\nExecuting divide() ...");

**if** (y == 0)

**throw** **new** Exception("Error: Division by Zero Detected; Cannot Perform Division.");

**else**

{

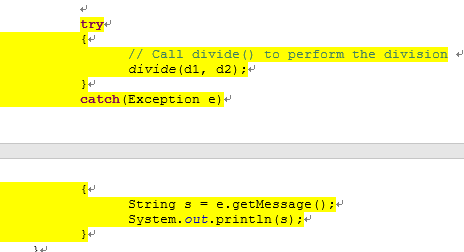
result = x/y;

System.*out*.println("The result of dividing " + x + " by " + y + " is: " + result);

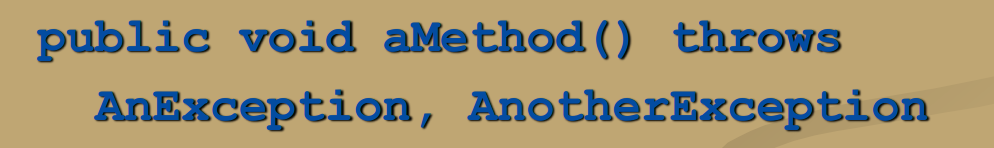
**return** result;

}

}



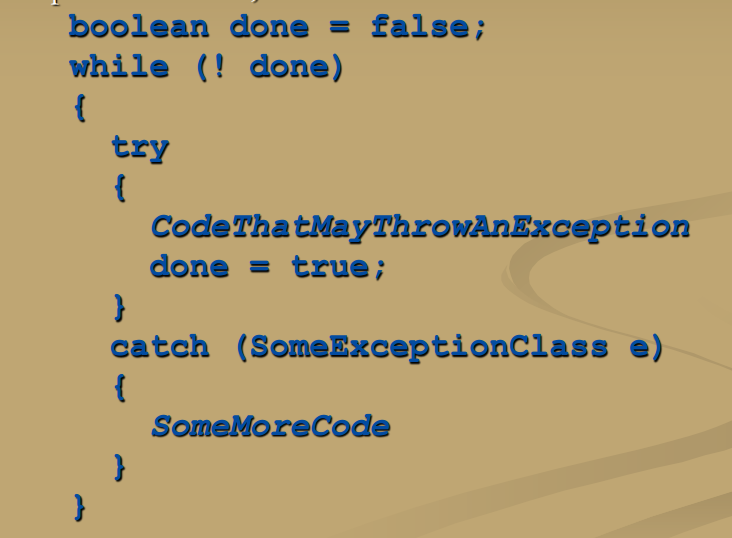
Throw clause还可以throw超过一种type的exception，用逗号隔开



如果一个method里即throw了一个exception然后又包括了一个catch语句，那么throwing exception不会中断程序（会进行catch与之后的程序）

如果没包含catch语句，exception被throw的一瞬间，method被中断

有时候可以通过loop一个action直到没有exception被产生



因为产生exception的瞬间就会中断，然后catch，只有不产生exception,才会继续

Scanner的nextInt:如果输入的不是int，会throw一个

除非这个exception被caught，不然程序会立即结束with 一个error message

如果被caught，catch block给出代码来进行一些其他操作，比如让用户再输入一遍int

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

Scanner kb = **new** Scanner(System.*in*);

**double** d1 = 0, d2 = 0; // To keep compiler happy!

**boolean** done = **false**, valuesAreNumbers = **false**;

**while**(!done)

{

// First make sure that the user enters two numbers

**while**(!valuesAreNumbers)

{

**try**

{

System.*out*.print("Please enter two values to perform a division: ");

d1 = kb.nextDouble();

d2 = kb.nextDouble();//不是double就回到catch

valuesAreNumbers = **true**;

}

**catch**(InputMismatchException e)

{

kb.nextLine(); // Clear that input line

System.*out*.println("You did not even enter two numbers! May you please enter numbers and nothing  
 else: ");

}

}//无限重复直到输入两个数字

// If you reach this point, we do have two numbers

**try**

{

// Call divide() to perform the division

*divide*(d1, d2);

// If you reach this line, then all is okay (no exception was thrown)

done = **true**;//对的话直接结束循环

}

**catch**(DivisionByZeroException e)

{分母为0，把valuesArenumbers重置为错，然后重新输入

String s = e.getMessage();

System.*out*.println(s);

valuesAreNumbers = **false**; // Reset that variable, so user can re-enter two values

}

**catch**(ResultIsAlwaysZeroException e)

{

String s = e.getMessage();

System.*out*.println(s);

done = **true**; // Since this is not an error, take it as a valid input and stop

}done=true但要说出啥信息

} // go back to the loop

System.*out*.print("Program reached last statement.");

}

Printwriter

PrintWriter pw = **null**;

String s;

Scanner kb = **new** Scanner(System.*in*);

**try**

{

pw = **new** PrintWriter(**new** FileOutputStream("Info.txt", **true**)); // Notice that second parameter

}

**catch**(FileNotFoundException e) // Since we are attempting to write to the file

{ // exception is automatically thrown if file cannot be created.

System.*out*.println("Could not open/create the file to write to. "

+ " Please check for problems such as directory permission"

+ " or no available memory.");

System.*out*.print("Program will terminate.");

System.*exit*(0);

}

写入信息： s = kb.nextLine();

pw.println("Name: " + s); // Write this info to the file

pw完以后要及时close();

如果一个程序试图用Scanner读取超过file的结尾，那么就会throw 一个exception

为了避免Exception作为信号来end这个file，Scanner这个类会有 这样的method

**private** **static** String[] getInfo(String articleName,String info,**int** articleNumber) {

Scanner sc = **null**;

String currentLine;

**int** count = 0;

String outInfo[] = **new** String[articleNumber];

**try** {

sc = **new** Scanner(**new** FileInputStream(articleName));

**while** (sc.hasNextLine()) {

currentLine = sc.nextLine();

**if** (currentLine.contains(info)) {

String needinfo=currentLine.replace(info, "");

String needinfo1=needinfo.replace("},","");

**if**(needinfo1.charAt(needinfo1.length()-1)==' ')

outInfo[count]=needinfo1.substring(0,needinfo1.length()-1);

**else**

outInfo[count]=needinfo1;

count++;

}

}

} **catch** (FileNotFoundException e) {

System.***out***.println();

System.*exit*(0);

}

sc.close();

**return** outInfo;

}

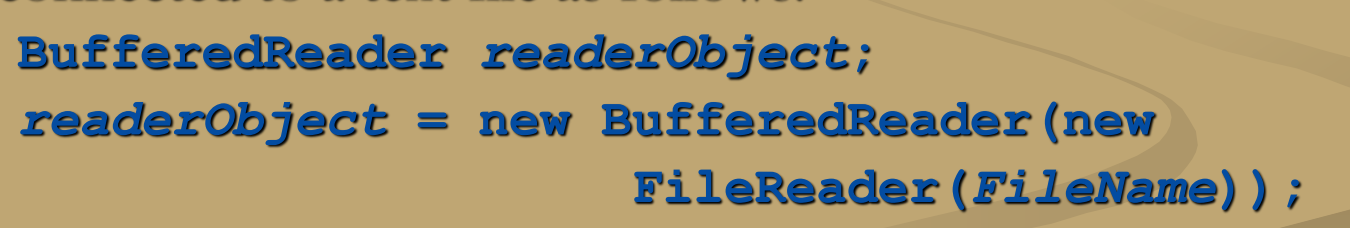
File:

File f = **new** File(路径);

f.exists();检查f这个文件存在不存在

f.delete();删除此文件。

BufferedReadear



String s1, s2;

PrintWriter pw = **null**;

Scanner kb = **new** Scanner(System.*in*);

BufferedReader br = **null**; // Create a BufferedReader object to read from input files

System.*out*.println("Please enter the name of the file you need to copy" +

" as well as the name of the file to be created: ");

s1 = kb.next();

s2 = kb.next();

// See if we can establish the two streams

**try**

{

br = **new** BufferedReader(**new** FileReader(s1));

pw = **new** PrintWriter(**new** FileOutputStream(s2));

// At this moment, both streams exist, so call the method to copy the file

// However, possible IOException may still occur while reading the input file

*fileCopy*(br, pw);

}

**catch**(FileNotFoundException e)

{

System.*out*.println("Problem opening files. Cannot proceed to copy.");

System.*out*.println("Program will terminate.");

System.*exit*(0);

}

**catch**(IOException e)

{

System.*out*.println("Error: An error has occurred while reading from the " + s1 + " file. ");

System.*out*.println("Program will terminate.");

System.*exit*(0);

}

+ **public** **static** **void** fileCopy(BufferedReader inFileStreamName, PrintWriter outFileStreamName)

**throws** IOException // Declare the IOException exception

{

// Read line by line from input file and copy it to output file

String s;

s = inFileStreamName.readLine();

**while**(s != **null**) // The readLine() method returns null when it is EOF

{

outFileStreamName.println(s);

s = inFileStreamName.readLine();

}

// Must close the files to flush the buffers

inFileStreamName.close();

outFileStreamName.close();

}

**public** **static** **void** displayFileContents(BufferedReader inFileStreamName) **throws** IOException

{

// Read line by line from input file and display it (on the monitor)

// The method actually uses the read() method (instead of readLine(), which we

// could have surely used here without a problem). The read() method reads

// character by character, but each character is read as type int, so casting is needed

// so that we can display character and not their integer values

**int** x;

x = inFileStreamName.read();

**while**(x != -1)

{

System.*out*.print((**char**)x); // MUST CAST; otherwise all what is read will be shown as integers

x = inFileStreamName.read();

}

// Must close the file

inFileStreamName.close();

Object

**public** **class** FileIO12{

**public** **static** **void** main(String[] args)

{

// This program will store student registration information of some course.

// The registration information will be entered by the user and stored in

// binary file.

**long** id;

String fileName, courseName, firstName, lastName, jk;

Scanner kb = **new** Scanner(System.*in*);

System.*out*.print("Please enter the course name for which you wish to add student registration: ");

courseName = kb.next();

// Construct file name based on the course name

fileName = courseName + "\_registeration.dat";

System.*out*.println("The registration information will be stored in a \"binary\" file called: " +

fileName);

ObjectOutputStream oos = **null**;//写在try外面

**try**

{

// Create an ObjectOutputStream to write into the binary file

oos = **new** ObjectOutputStream(**new** FileOutputStream(fileName));如果这步成功了，就会自动进行下面内容，如果有exception,那么就跳过

// If this point is reached, then the stream was created successfully.

// Place a string at the beginning of the file indicating the course number, then

// ask the user to enter the registration information

oos.writeUTF("This file includes the registration of " + courseName + ".");

System.*out*.print("Please enter the student ID you wish to register in " +

"the course or -1 to terminate://结束 ");

id = kb.nextLong();

**while**(id != -1)

{

jk = kb.nextLine(); // Clear the rest of the line 清除下一行内容

System.*out*.print("Please enter first name and last name of the student: ");

firstName = kb.next();

lastName = kb.next();

oos.writeLong(id);

oos.writeChar('\t'); // Add two tabs

oos.writeChar('\t');

oos.writeUTF(firstName);

oos.writeChar(' '); // Add one space to separate the names

oos.writeUTF(lastName);

System.*out*.println("One student info was added to the file.");

System.*out*.print("\nPlease enter the following student ID you wish " +

"to register in the course, or -1 to terminate: ");

id = kb.nextLong();

}

System.*out*.println("Registration completed.");

oos.close(); // Close the file//别忘了close

}

**catch**(IOException e)要catch IOexception

{

System.*out*.println("Problem writing to output file " + fileName);

System.*out*.println("Program will terminate.");

System.*exit*(0);

}

}

{

// This program reads a course student registration stored in a binary file,

// of a known format, then displays this information.

**long** id;

String fileName, courseName, firstName, lastName, jk, s;

**char** ch1, ch2, ch3;

Scanner kb = **new** Scanner(System.*in*);

System.*out*.print("Please enter the \"binary\" registration file name you wish to read: ");

fileName = kb.next();

ObjectInputStream ois = **null**;

**try**

{

// Create an ObjectInputStream to read from the binary file

ois = **new** ObjectInputStream(**new** FileInputStream(fileName));

// If this point is reached, then the stream was created successfully.

// So, start reading the file from its start until no more data can be read.

// When an attempt to read beyond the end of the file is reached, and exception

// is thrown. We will use this as an indication of the end of the file.

System.*out*.println("Here is the information stored in this file: ");

System.*out*.println("============================================\n ");

**try**

{

s = ois.readUTF(); // Reads the first line of the file and display it

System.*out*.println(s);

**while**(**true**)

{

// read the id, the two tabs, the first name, the space, then the last name

id = ois.readLong();

ch1 = ois.readChar(); // Read the two tabs

ch2 = ois.readChar();

firstName = ois.readUTF();

ch3 = ois.readChar(); // Read the space between the names

lastName = ois.readUTF();

System.*out*.println("" + id + ch1 + ch2 + firstName + ch3 + lastName);

}

}

**catch**(EOFException e)

{

System.*out*.println("Reading " + fileName + " has been completed.");

}

ois.close(); // Close the file

}

**catch**(FileNotFoundException e)

{

System.*out*.println("File: " + fileName + " could not been found.");

System.*out*.println("Program will terminate.");

System.*exit*(0);

}

**catch**(IOException e)

{

System.*out*.println("Error: Problem Reading from file: " + fileName + ".");

System.*out*.println("Program will terminate.");

System.*exit*(0);

}

}

}

Ch6

一个标准的recursive method的definition

1. 一个或多个情况——这个情况对这个正在被描述的method拥有一个或多个recursive calls

这些recursive calls应该解决这个method正在解决的任务的更小版本

1. 一个或多个情况，不包括recursive calls：这个叫做base cases or stopping cases

RECURSION VS ITERATIVE

Recursion并不是绝对需要的

任何可以被recursion解决的任务都可以用nonrecursive 方式解决

一个nonrecursive 版本的method叫做iterative version

一个iteratively written的method将会使用loop来取代recursion

一个recursively written的method更简单，但是跑得慢，用的存储更多，与iterative version相比较

**public** **class** Recursion6{

// A recursive method that uses binary search to find

// a value in a sorted array. If the value is found,

// its index is returned; otherwise, the method returns

// -1 as an indication that the value was not found in the array.

// The method expects an array of integers, a start index and

// an end index for the search, and finally the value that the search is for

**public** **static** **int** search(**int**[] A, **int** startIndex, **int** endIndex, **int** v)

{

**int** foundIndex = -1, mid;

**if**(startIndex > endIndex ) // Stopping condition; search is exhausted

**return** -1; // without finding the value

//起始搜索点都大于结尾搜索点了，矛盾，-1

// Now find if the value is in the middle of the array

// and if it is not there, then on which part of the array

// it could be there, then search only that part

mid = (startIndex + endIndex)/2; // Find the middle index//中间=起始加结尾/2

// If the array has an odd number of elements

// that index is in the exact middle; otherwise

// it is one of the middle two, which is okay

**if**(A[mid] == v )

{ //中间正好找到

System.*out*.print("\nThe value was found ......");

**return** mid; // Return the index where the value was found

}

**else** // Recursive Steps

{

**if**(v < A[mid]) // Value could only be in the left part of the array

{

System.*out*.print("\nWill search the array between index # " + startIndex +

" and index # " + (mid-1));

foundIndex = *search*(A, startIndex, mid-1, v);

}比他小，recursive，循环了一遍，小于的时候startindex=startindex,中间-1作为endindex

**else** // Value could only be in the right part of the array

{

System.*out*.print("\nWill search the array between index # " + (mid+1) +

" and index # " + endIndex);

foundIndex = *search*(A, mid+1, endIndex, v); 大雨的时候startindex=mid+1,endindex=endindex

}

}

**return** foundIndex;

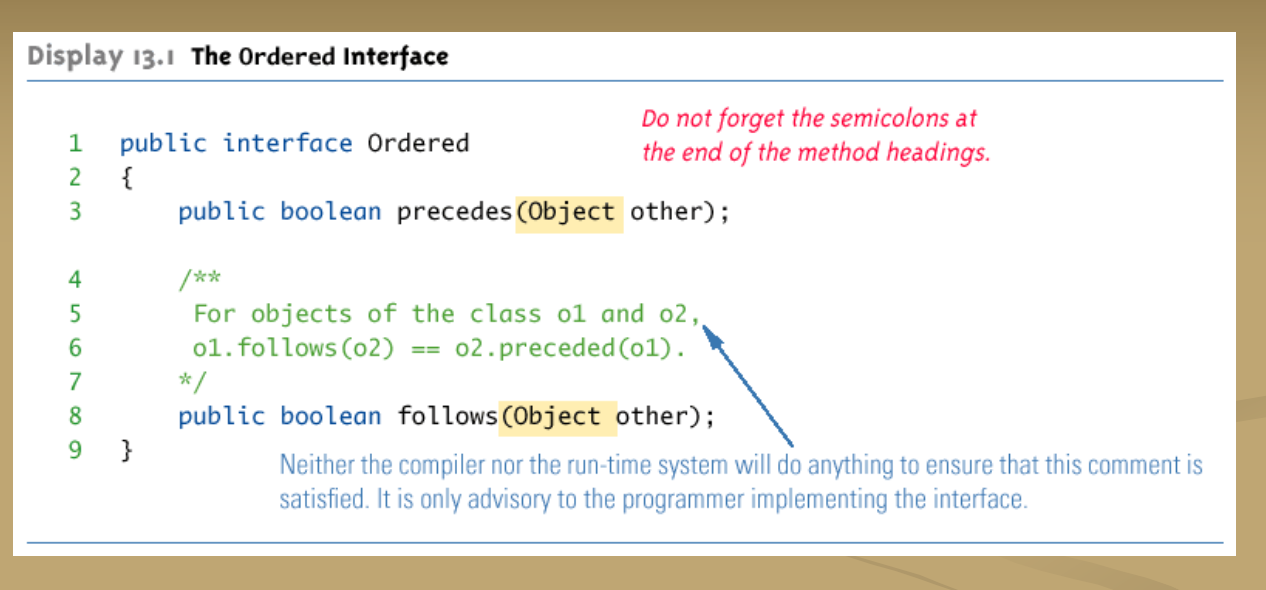
}

Ch7

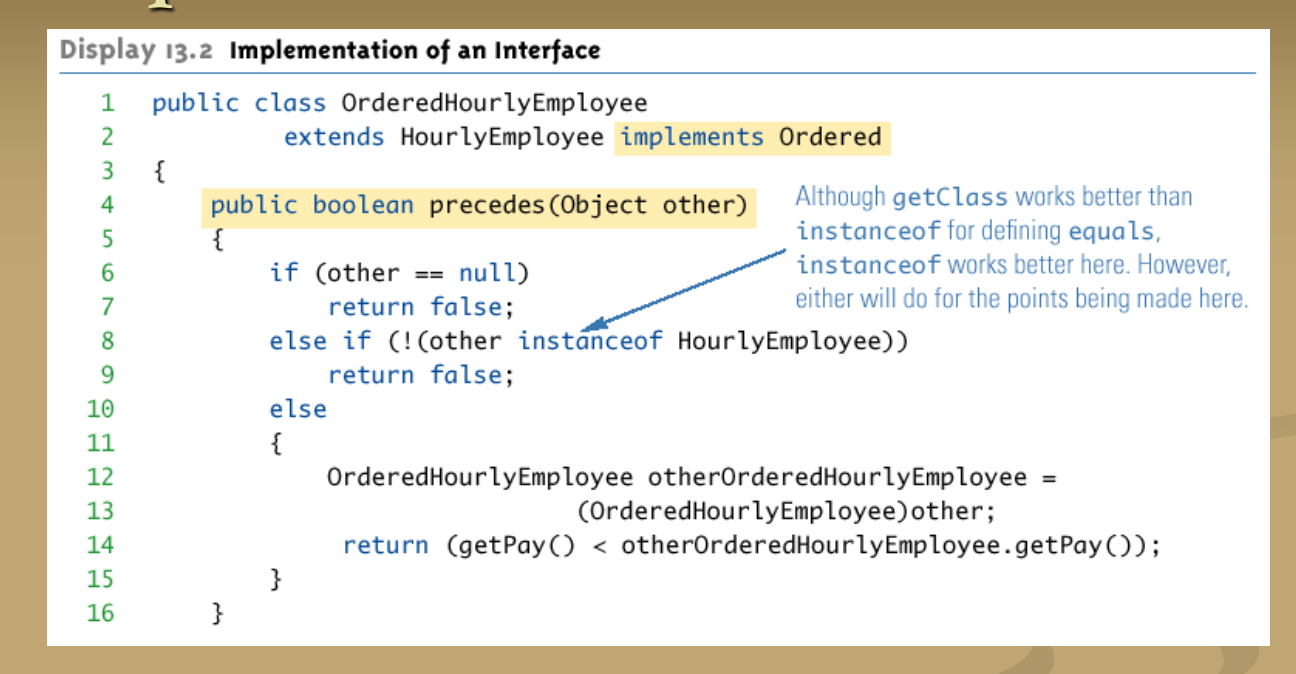
一个interface和他的所有method heading的前缀应该是Public

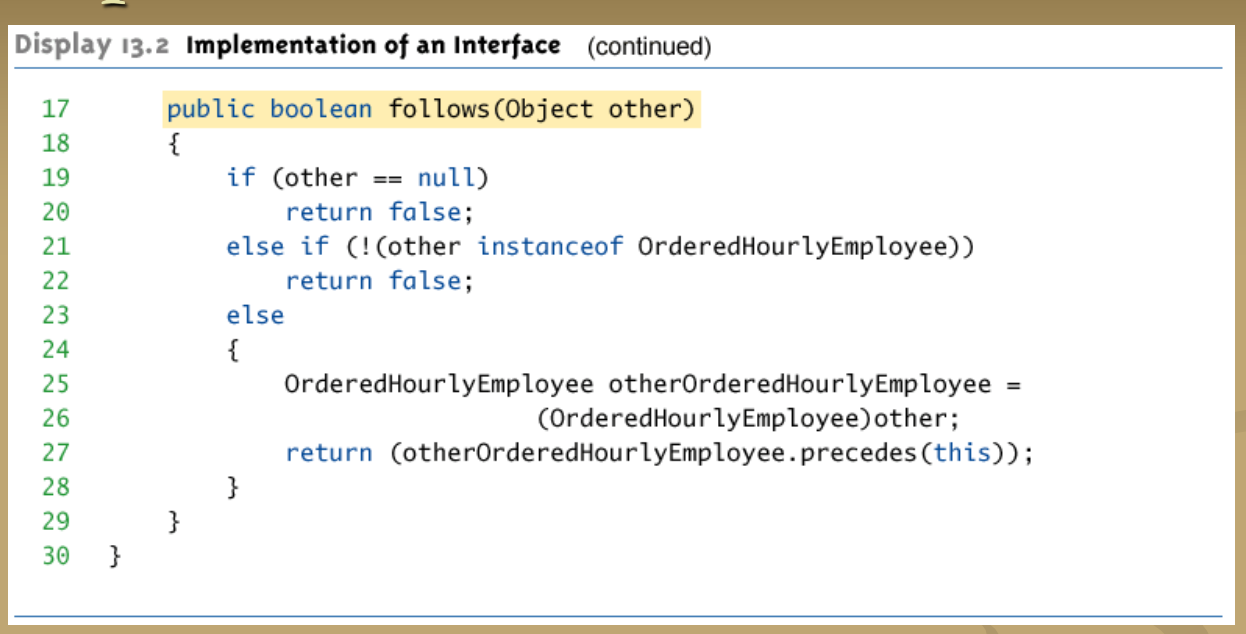
因为interface是一种type，所以一个method的参数可以是interface type。这个参数可以接受任意implement这个interface的class

注意了，不加花括号，直接分号结束

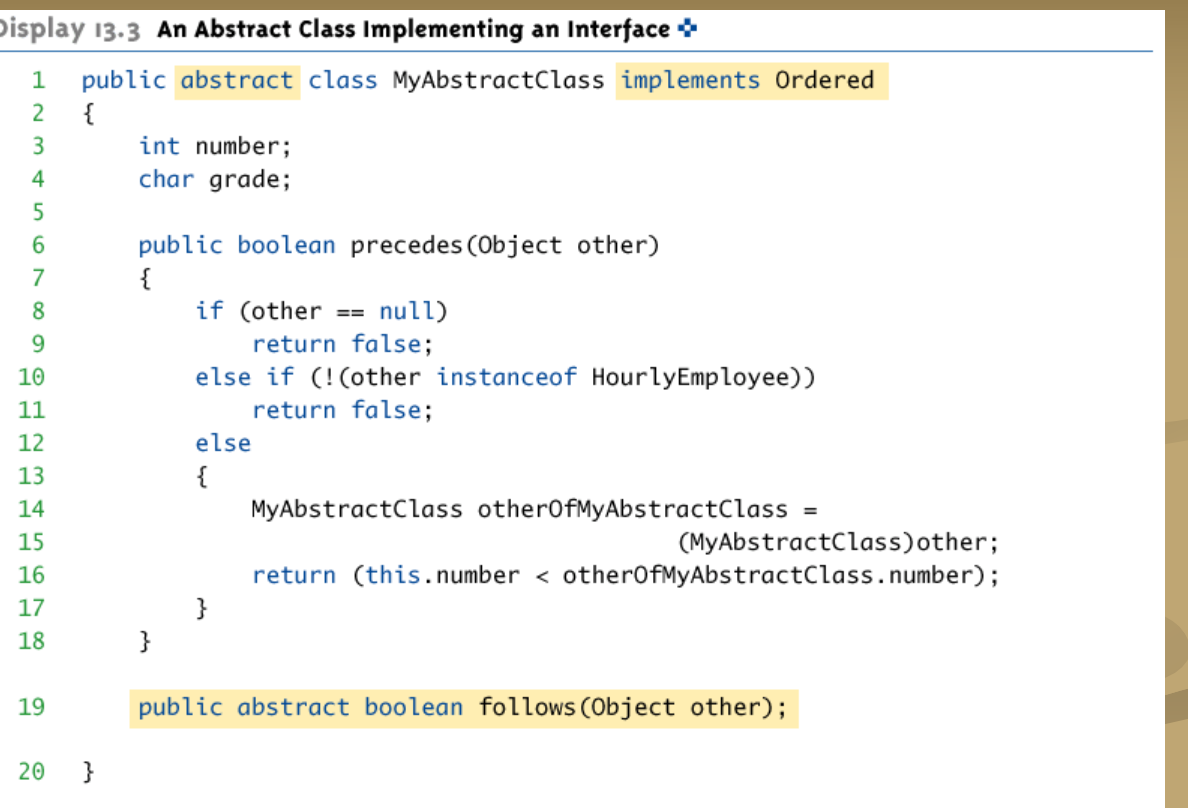








**Abstract class可以Implement interface**



**描述过后的就不用加abstract，没描述的要加abstract**

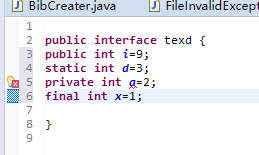
**常量前缀为**前缀必须为public/static/final

Defined Constant in interface

Constant:常量，一种特殊的变量，变量都是可以改动的，常量无法改动

前缀必须为public/static/final

Interface中可以有常量·，任意implement这个interface的class都可以使用这些常量



**Interface可以Implement interface**

如果一个标准class implement了一个derived interface，那么它不仅必须有derived interface的method，还要有base interface的method

可以implement 多个Interface:没有实际的method definition，不会出现此问题

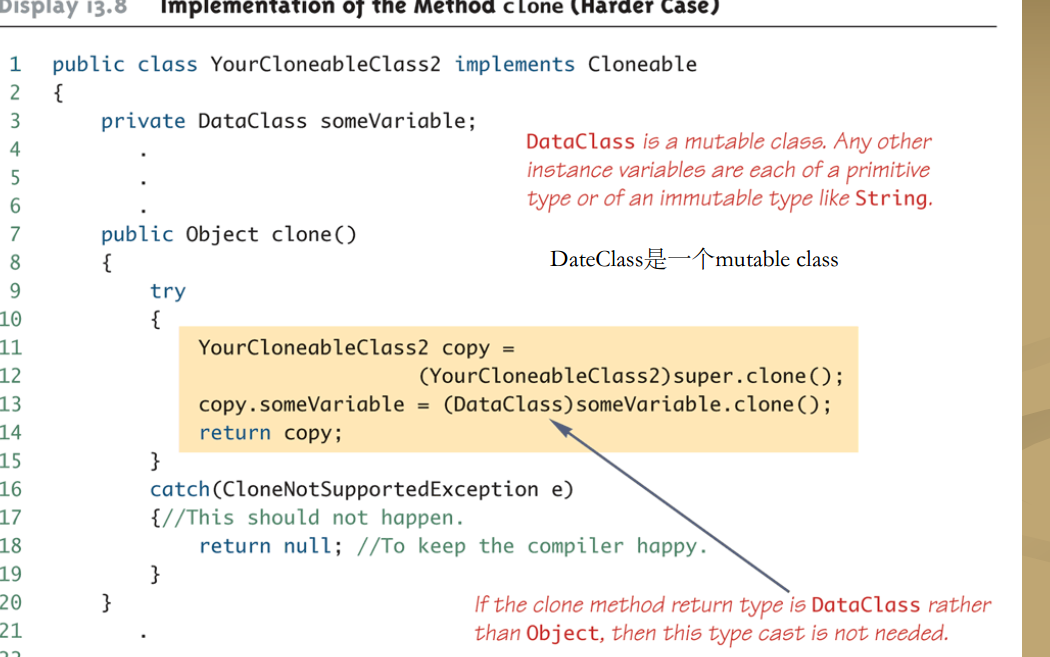
但会出现其他的问题

1. 继承常量的时候常量名字相等，值不等
2. Method 有相同的名字，但有不同的return type

如果class definition同时implement了这样两个Interface,就会有一个error, class definition也会Illegal

对于clone如果data都是primitive type data 或者immutable class type 的data（比如String）,那么就是adequate（胜任的），如果有class type的参数，那么就会导致privacy leak

为了防止这点



在inner class内的method definition:

可以使用Outer class的实例变量作为参数

可以使用outer class的private method

在outer class内的method definition：

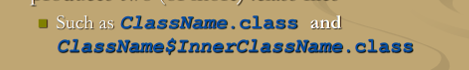
可以使用inner class的object中的实例变量作为参数

可以使用inner class的object的method

所以Inner class与outer class 的Public与Private是等价的

在JAVA中编译任何class都会产生一个.class file名字是

编译一个inner class会导致一个两个class都被编译，产生两个.classfile

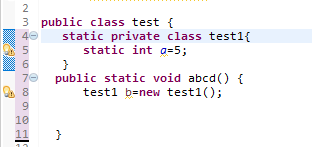


Static Inner Class

一个普通的inner class通过他的object与outer class链接

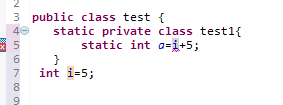
但以下情况，inner class必须是static的

1. 如果inner class的object是在outer class里的static method里创造出来的
2. 如果inner class必须有static member(variable或method)



因为static inner class与outer class的object没有联系，所以Inner class method不能使用outer class的instance variable

不能激活outer class的nonstatic method



为了使用static inner class的static method或者static variable，就直接用inner class的name.xx就行

如果inner class里面试图使用一个method

如果Inner class无此method，那就用outer class的此name的method

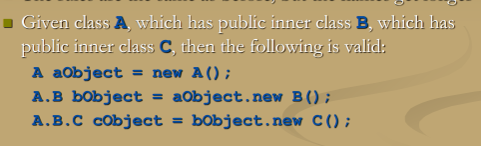
有，就用inner class的

如果有，还要用outer，必须使用以下invocation方法



Nesting inner classes

可以在Inner class里面在加Inner class，所有逻辑如上



对于一个有innerclass的outerclass来说

Outerclass的子类会自动继承innerclass作为一个innerclass，子类不能override这个innerclass

Outerclass可以作为子类

Innerclass也可以作为子类

Anonymous class匿名class

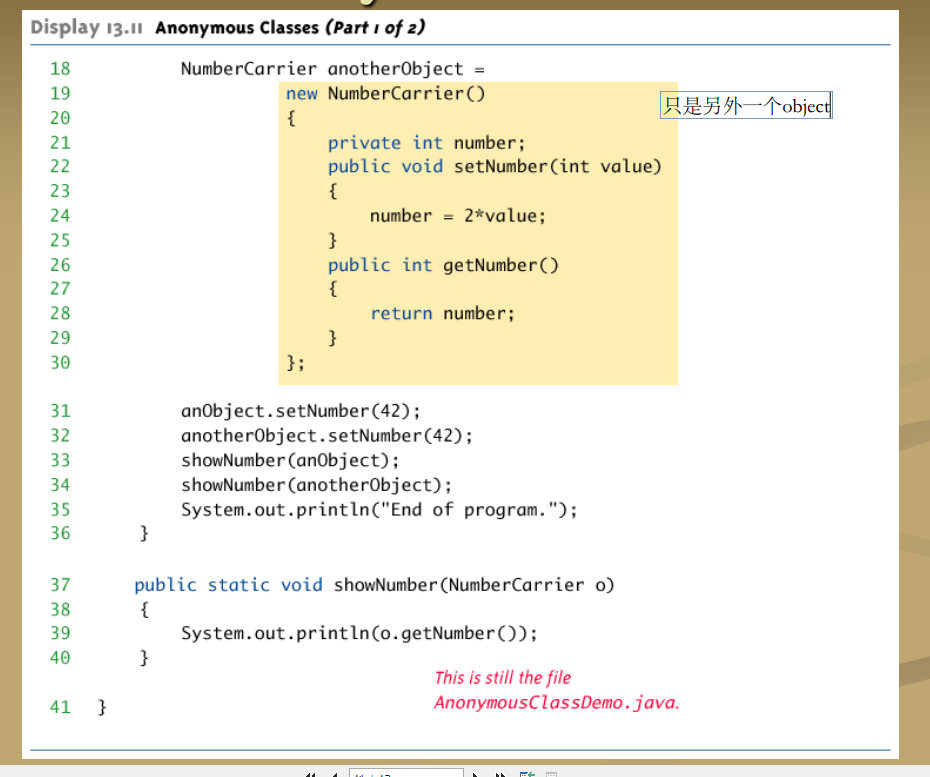
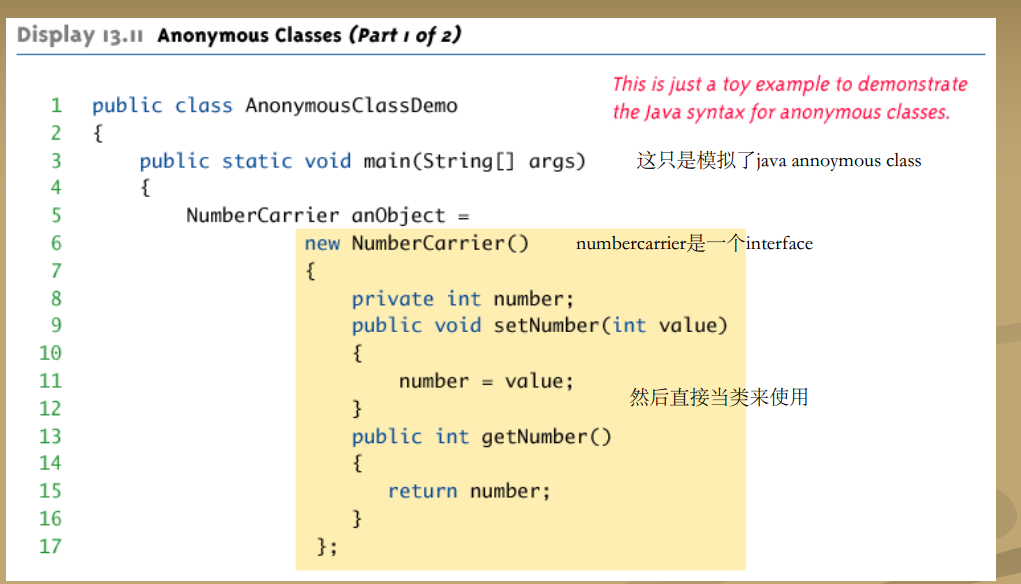
如果一个Object被创造了，但没必要给object的class命名，这时候就用了anonoymous class definition

这个class definition被嵌入在中

Anonymous classes有时被使用当他们被另外一个type的variable赋值的时候

另外的type通常是java interface

这个other type必须要做到，匿名类的object也是other type的object



**Arraylist**

add可以在arraylist中加入元素，加在空白的后面，

第一个参数是index，第二个是要add的数，这个操作完以后index1位置是c6，index2是原Index1

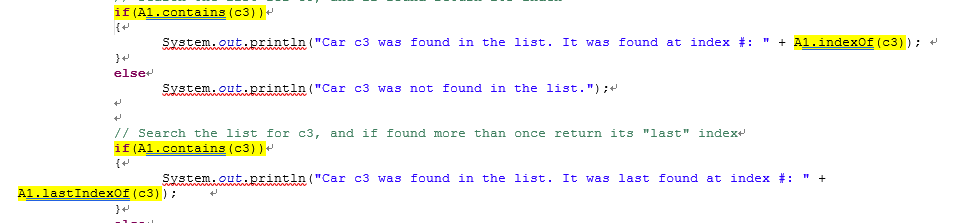
size，得到现在arraylist中总共有几个元素，哪怕length是5，而5个都是Null，那么就是size=0

set



Get

Contains 会返回一个boolean值



注意如果我们对同一个元素连续contains两次，第二个contain代表着这个array有两个以上的这个元素

indexOf返回一个Int值

会return最后一个c3所在

removeremove index2

remove以后size变小，如果remove了index2，原来Index2消除，index3变为index2

clear: 消除所有元素,size变为0

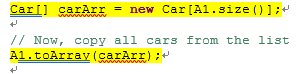
isEmpty: Empty，true

方法1这里创建了一个A1 arraylist

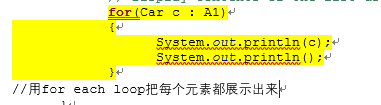


但这里1是Object类型，要使用typecast

方法2







**class** ExitListener **implements** ActionListener

{

**public** **void** actionPerformed(ActionEvent e)

{

System.*exit*(0);

}

}