JAVA SE基础部分：

JAVA 面向对象：

JAVA 应用部分：

# 基础部分：

**模块一：面向对象**

**模块二：java集合框架**

**模块三：java IO**

**模块四：JDBC 数据库操作**

【重点】Java SE基础知识：

【重点】面向对象：

基础部分;

高级部分：

异常处理：

包及访问权限：

【理解】JDK1.5 新特性：能看懂就行

【重点】Eclipse开发工具：

【了解】多线程：概念有用，实际效果不明显，为android开发

【重点】java类库：

【重点】JDBC：

【重点】java IO操作

【了解】网络编程：进行socket开发，也是为了android准备

【核心重点】DAO设计模式:

## 3.1 java 简介

一旦oracle收购了SUN之后，会出现哪些影响？实际上市场上就形成了三足鼎立：微软、oracle、adobe

|  |  |  |  |
| --- | --- | --- | --- |
| No | 对比点 | 微软 | Oracle |
| 1 | 操作系统 | Windows | UNIX |
| 2 | 数据库 | SQL server,Access | Oracle数据库，MySQL |
| 3 | 开发语言 | VS.net | PLSQL，SUN java |
| 4 | 中间件 | IIS | 收购了BEA的weblogic |

## 3.2 java 的主要特点：（了解）

·java足够简单

·java是一门面向对象的编程语言

·java避免了复杂的指针操作，使用引用完成操作

·具备自动的垃圾收集机制

·java语言具备很强的移植性

用户将做所有程序保存在\*.java之中，经过编译生成\*.class文件，而后在电脑上解释执行，但这只是有软件和硬件模拟出来的电脑，成为JVM（java 虚拟机）。

面试题：解释java实现可移植性原理。

Java程序最终通过字节码文件运行，运行的时候字节码需要JVM的支持，但是不同的操作系统上有不同的JVM，程序不关心操作系统，只关心JVM，只要JVM不变，程序可以在操作系统间任意的移植。

## 3.3 java开发环境的搭建

JDK（java 开发工具包）

建议显示英文，修改语言环境：【控制面板】->【管理工具】->【区域和语言选项】这样就可以显示正常的英文信息。之后会提示安装JRE，这个环境是执行java程序是使用的。Java、javac这两个命令并不属于windows环境。所以需要配置环境变量;

·所有可执行命令目录：C:\Program Files\Java\jdk1.6.0\_10\bin

·将目录配置到windows中：编辑path路径，添加上述路径到path中。

## 3.4 第一个java程序（重点）

进入D盘：d:

cd 目录

编译程序：javac 名称.java

执行程序：java 名称

查看当前版本信息：java-version

Oracle默认里有配置JDK，可能导致版本不合。

方法一：删掉与jdk有关配置

方法二：把新配置卸载最前面。

## 3.5第一个程序的相关解释

1、类的定义

Public 用于定义一个**类**，**命名要求：每个单词的首字母大写。**例如：TestHelloDemo。

public class:文件名必须和类名保持一致， \*.java文件中只能定义一个public class;

class**:** 在\*.java文件中可以定义多个class，但是只能有一个public class。

2、主方法：表示程序的起点

3、系统输出：直接在屏幕上输出system.out.println();

## 3.6、classpath属性（重点）

Classpath：此属性的作用是用于进行类文件路径的指定。例如在D盘中有一个类，如果想执行这个类则肯定要进入其所在的文件夹之中，如果说现在在不同的目录中呢？

这个时候就要设置classpath：SET CLASSPATH=类所在路径。

可是这样有一个问题，就是classpath乱指，所以默认情况下最好直接从当前所在路径中找到所需要的\*.class最合适。在开发之中一般将其设置为“.”：SET CLASSPATH = .

可是以上所设置的classpath只对一个命令窗口起作用，如果想对所有命令行方式起作用，则可以直接在环境属性中增加一个classpath选项。

·使用java命令执行程序的时候，会使用classpath指定的路径加载\*.class文件，并最终在JVM上执行。

面试题：解释path和classpath区别：

·path是windows的环境属性，用于指定可执行命令的路径；

·classpath在java程序执行的时候用于指定类的加载路径。

## 3.7、关键字和标示符（重点）

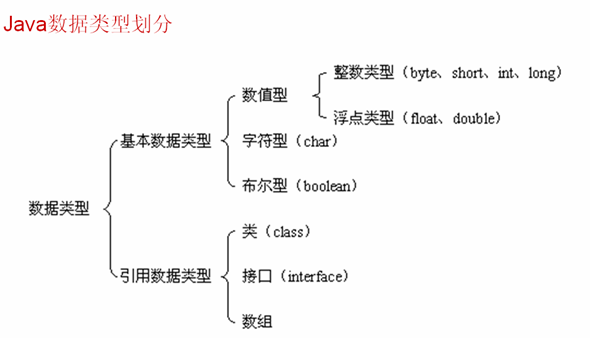
在定义变量名称，或类名称的时候都需要用到标示符：字母、数字、下划线，其中不能以数字开头。

·未使用到的关键字：goto、const

·有特殊含义的关键字：null、true、false

## 3.8、数据类型（重点）

### 3.8.1、数据类型化分（背）

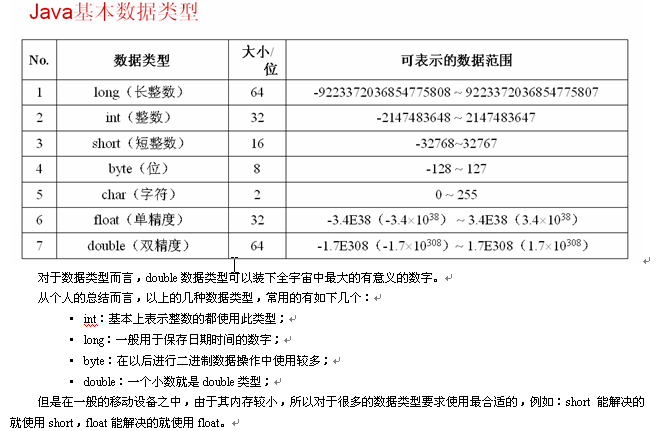


所谓引用数据类型就是指针，但是比指针简单。

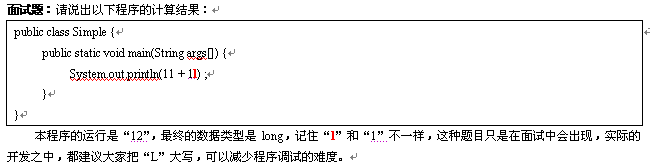
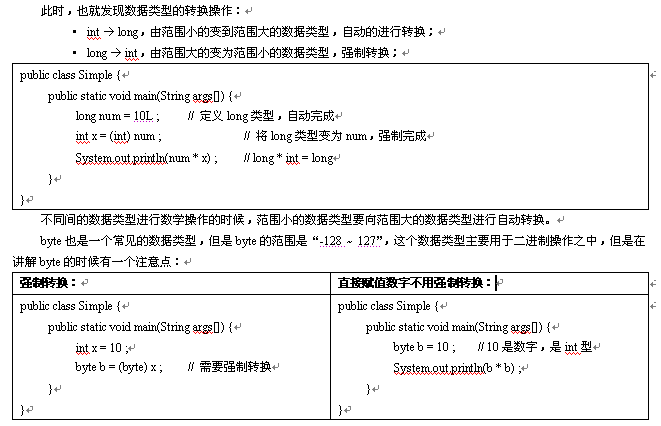
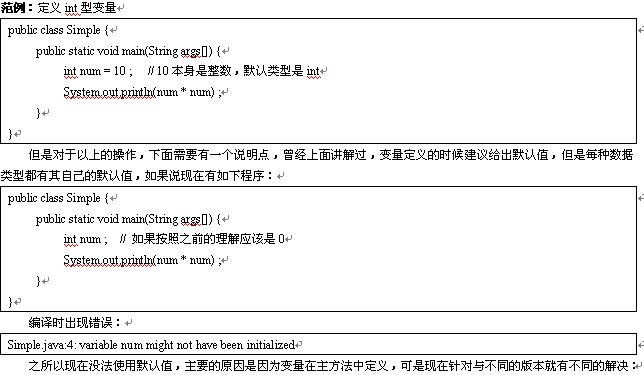


建议大家定义变量时对变量进行初始化。

对于double来说，可以装下全宇宙中最大的最有意义的数字。



### 3.8.2、 整型数据



### 3.8.3 字符型数据

C语言中使用的是ASCII码，可是java中的字符使用的UNICODE编码，而UNICODE编码本身包含了ASCII编码。

UNICODE是使用十六进制表示的编码格式，可以表示世界上任何的文字。

**要想用char类型保存汉字，前提是语言环境必须是中文。**

### 3.8.4、浮点型数据

默认小数double数据。但是想用float数据，需要在数字后加F。

一个bug：



Int型数据是没有小数点的，要想出现小数点的话，则必须将其中一个数据变为double或float。

### 3.8.5、布尔型

一般都会将0当false，非0为true。可是java中没有这样复杂，boolean型数据只有true和false两种取值。

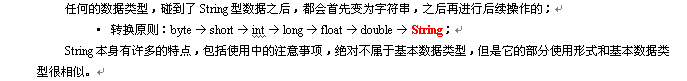
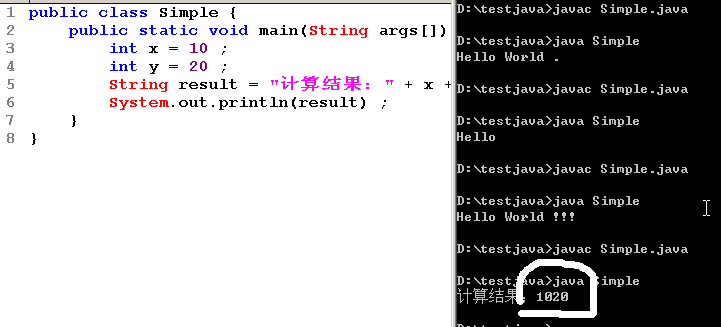
主要是控制程序的流程操作上使用较多。

### 3.8.6、字符串：String

String型数据保存字符串，但是由于String首字母大写，所以肯定是一个类，但是这个类的使用比较特殊而已。

在字符串中也可以用“+”进行字符串的连接操作。

注意：**只要是“”定义的内容都是字符串**。



## 3.9、运算符

3.9.1、逻辑运算

与、或、非

·与：&、&&

·或：|、||

1、与操作：所有条件都满足才行

·普通与操作：&

·短路与操作：&&，前面的条件不满足，不管后面的怎么样都不进行判断。

面试题：请解释&和&&、|和||的区别。

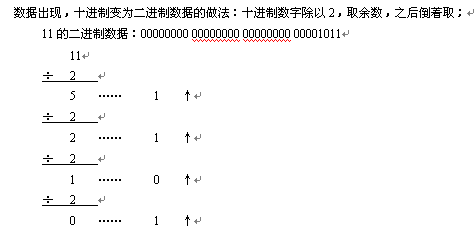
·&（普通与）和 |（普通或）指的是所有条件都进行判断；

·&&（短路与）和||（短路或）

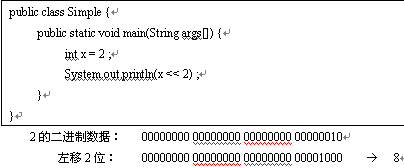
·在开发之中为了性能的提高，主要使用短路与和短路或操作。

·&和|除了用于逻辑运算之外，也可以进行位运算操作。

### 3.9.2、位运算（了解）

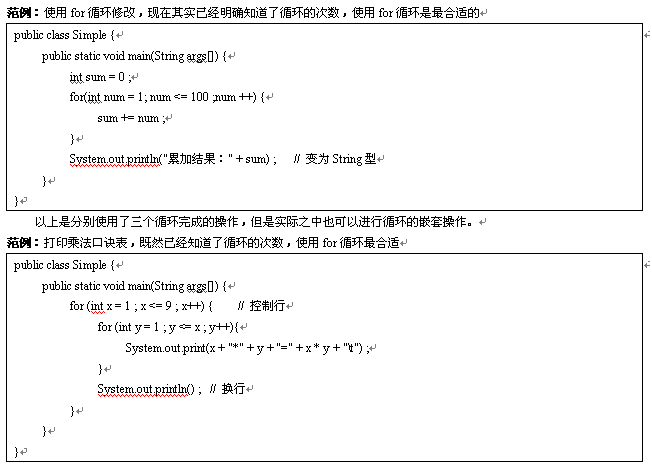
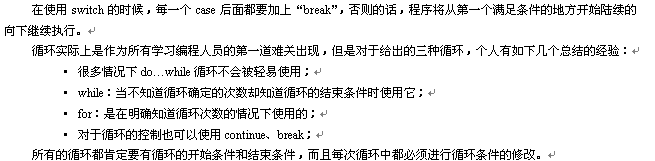
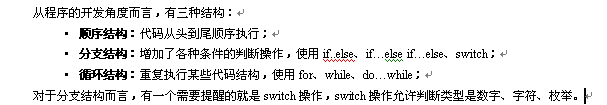


面试题：请问，如何以最快的速度计算2的3次方？



### 3.9.3、三目运算符

## 3.10、程序结构（重点）



//输出三角形

public class Test

{

public static void main(String args[]){

int x=10;

int y=0;

for (int i=0;i<=10 ;i++ )

{

for(int j=0;j<=10-i;j++)

{

System.out.print(" ");

}

for(int k=0;k<=i;k++)

{

System.out.print("\* ");

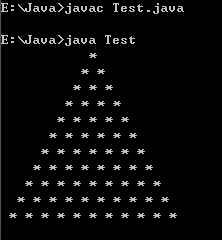
}

System.out.println("");

}

}

}



【了解】额外提醒：在C语言中有goto语句表示无条件跳转，但是这个关键字在java中，但是不能使用，可是在java中却可以使用continue实现与goto一同样的功能。

## 3.11、方法（重点）

### 3.11.1、方法的基本概念

可以重复调用的代码段。

由**主方法**直接调用的函数

Public static 返回值类型 方法名称（参数列表）{

[return [返回值]；]

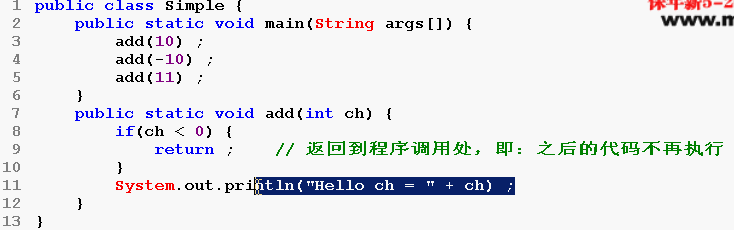
}

命名规则：第一个单词首字母小写，之后每隔单词首字母大写，例如studentName.

两个基本的定义方法的原则：

·第一个：代码重复

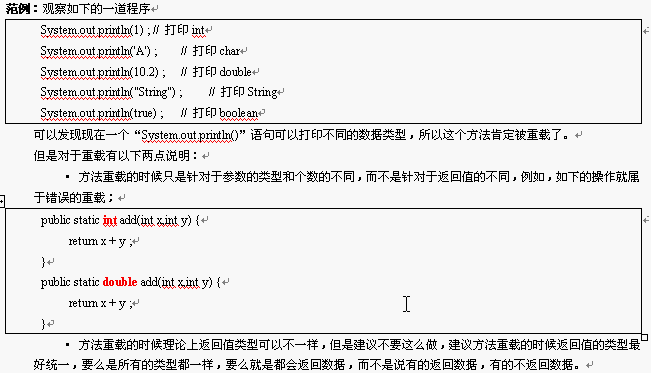
·第二个：我们希望主方法中的代码越少越好



这个功能主要是返回调用处，而且前提是方法返回值是void。

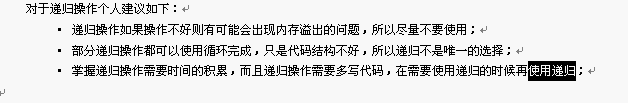
### 3.11.2、方法的重载

方法名称相同，但是参数的类型或者个数不同的情况。



### 3.11.3、方法的递归调用（理解）

两个条件：



public class Test

{

public static void main(String args[]){

System.out.println(add(100));

}

public static int add (int num){

if(num==1){

return 1;

}

return num+add(num-1);

}

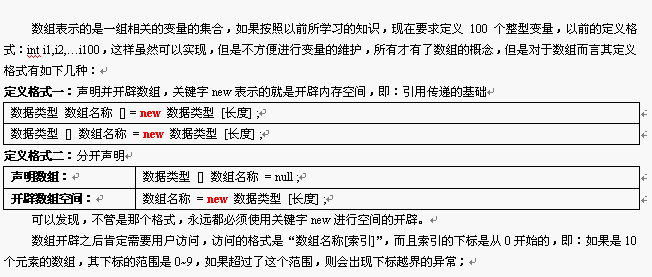
}

例子：递归求1+2+……+100.

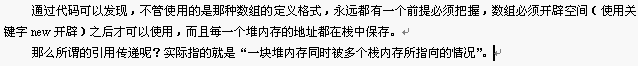
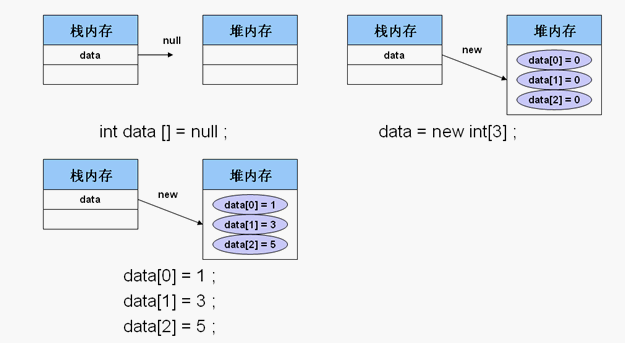
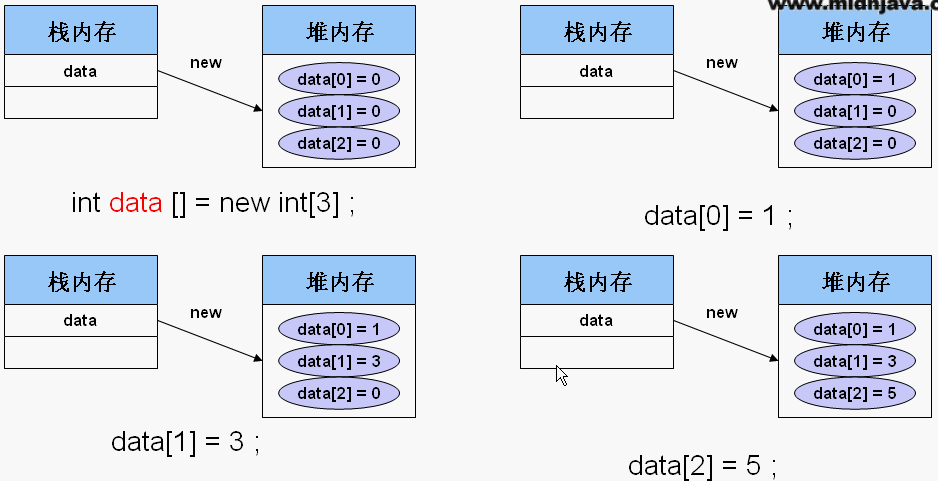
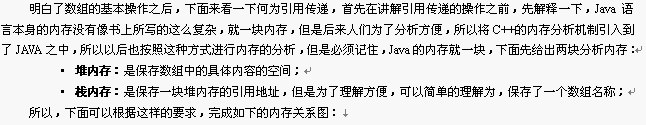
## 3.12、数组（重点）



### 3.12.1、数组的基本概念

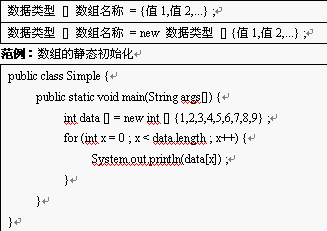


一般用数组.length来计算数组长度。

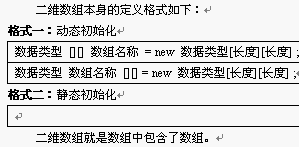


**引用传递的核心就在于：一块堆内存被不同的栈内存所指向，一个栈修改了内容，其他的所有对应栈内存的数据都要进行修改。**

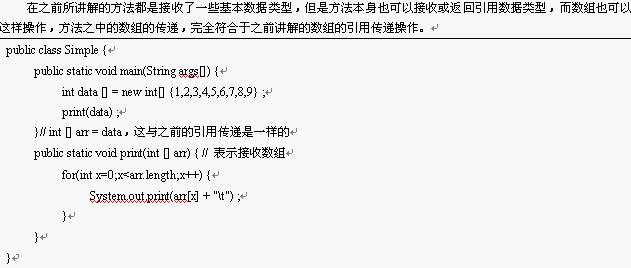
### 3.12.2、数组的静态初始化



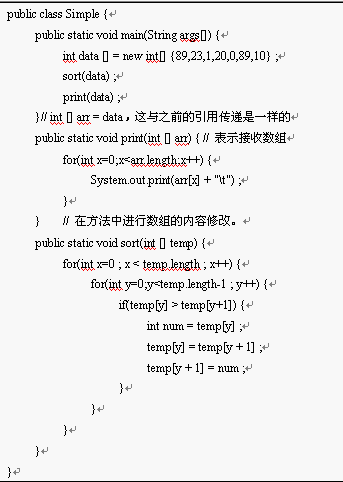
### 3.12.3、二维数组（了解）



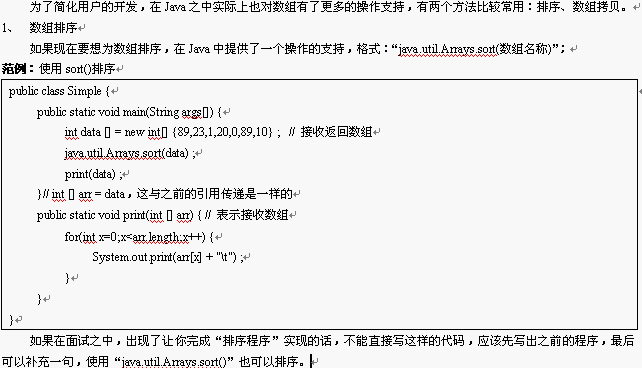
### 3.12.4数组和方法（核心）



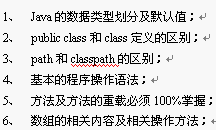
完成数据排序：



### 3.12.5、与数组有关的操作方法（重点）



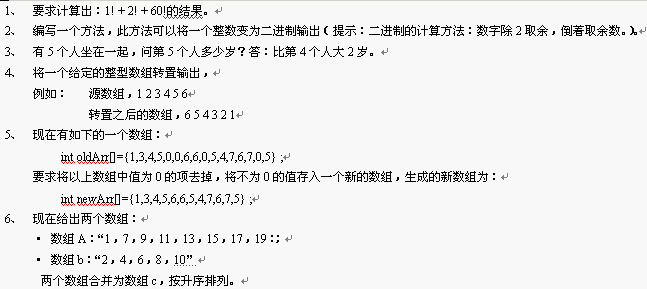
## 4、总结



## 5、预习任务



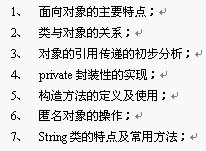
## 6、课后习题



# 正式上课

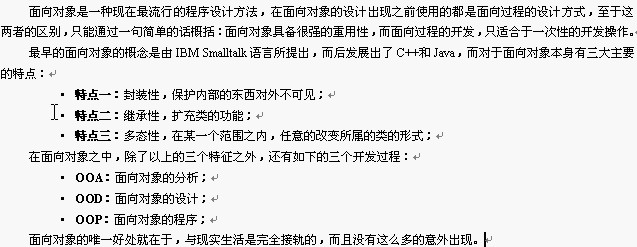
## 1、课程名称：面向对象

## 2、知识点



## 3、具体内容

### 3.1认识面向对象（理解）

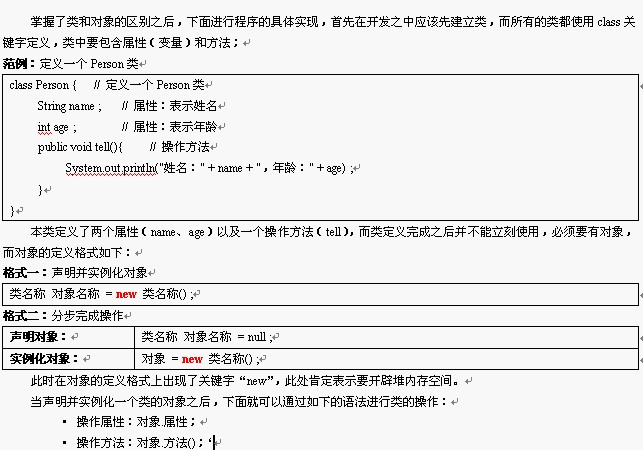


### 3.2类与对象（重点）

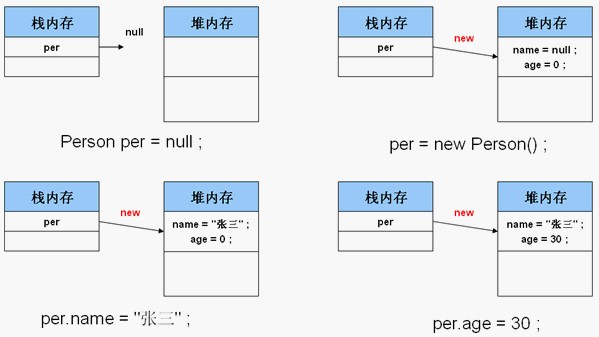
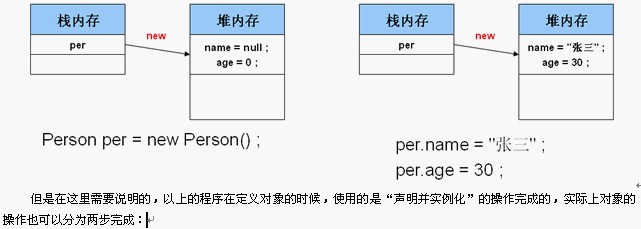
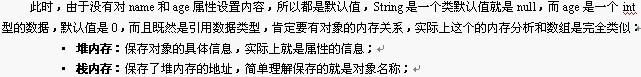
#### 3.2.1、类与对象的基本定义



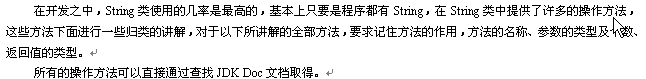
#### 3.2.2、类与对象的基本操作



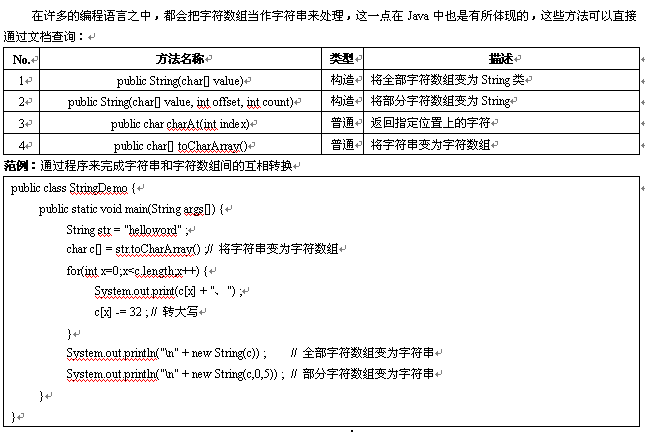
有“（）”的永远是方法。



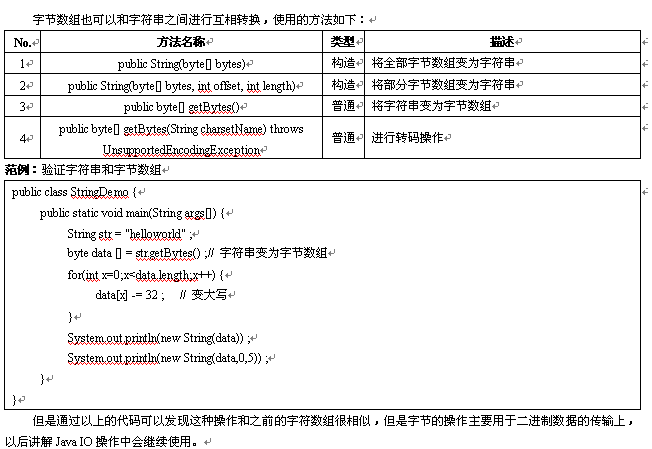
## 3.8字符串的操作方法(重点)



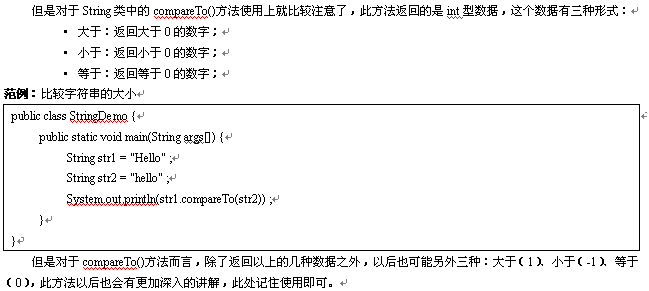
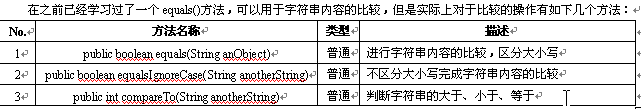
### 3.8.1字符与字符串



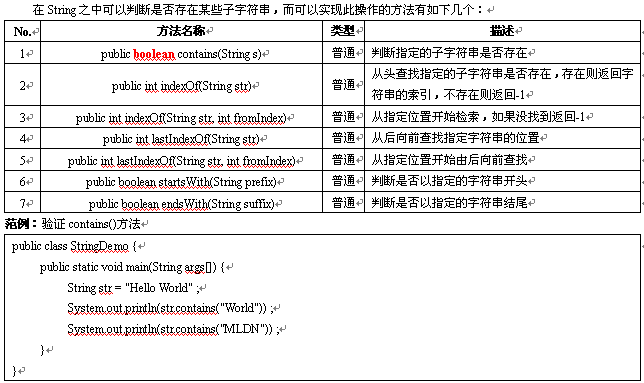
### 3.8.2字节数组与字符串



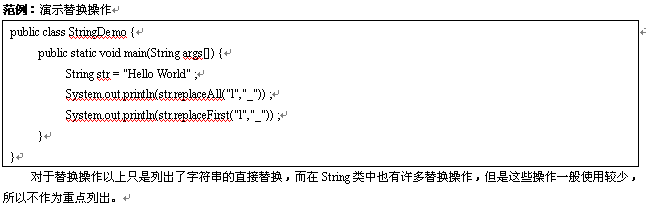
### 3.8.3字符串的比较



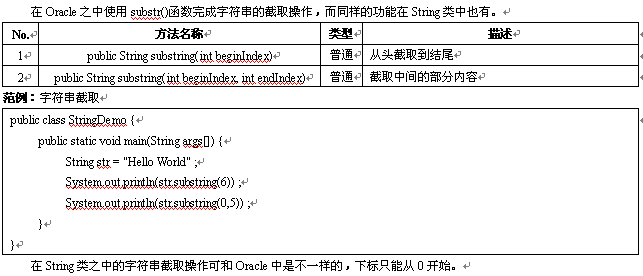
### 3.8.4字符串检索



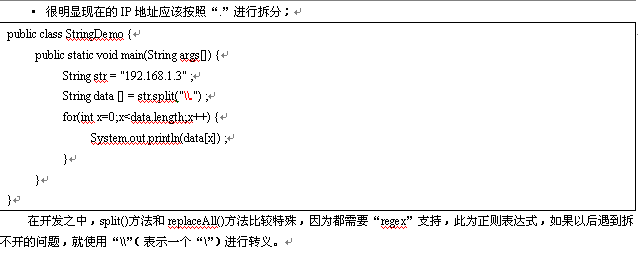
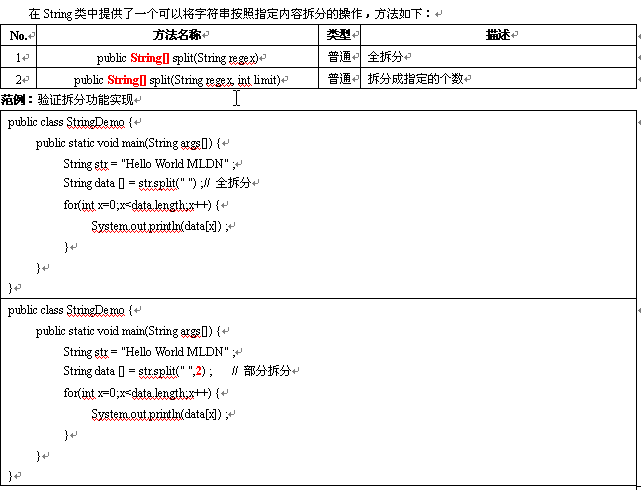
### 3.8.5字符串替换



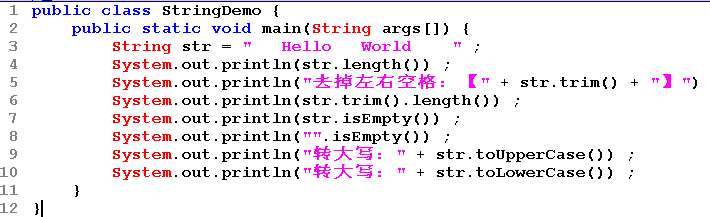
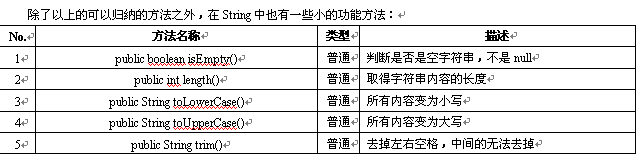
### 3.8.6字符串的截取



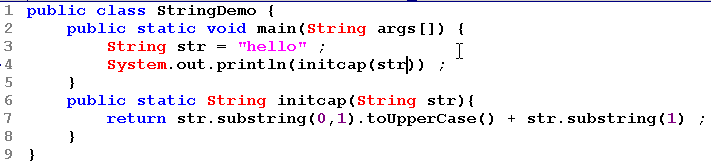
### 3.8.7字符串拆分(第一次讲解)



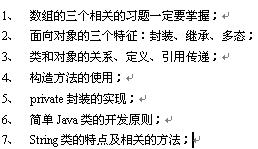
### 3.8.8其他方法



首字母转大写



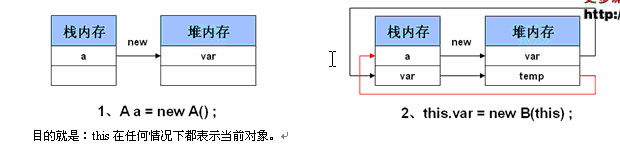
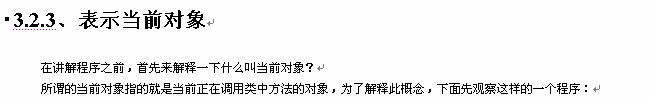
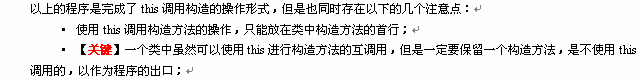
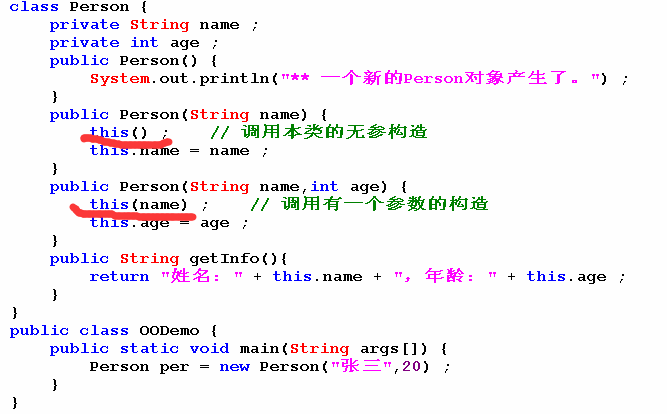
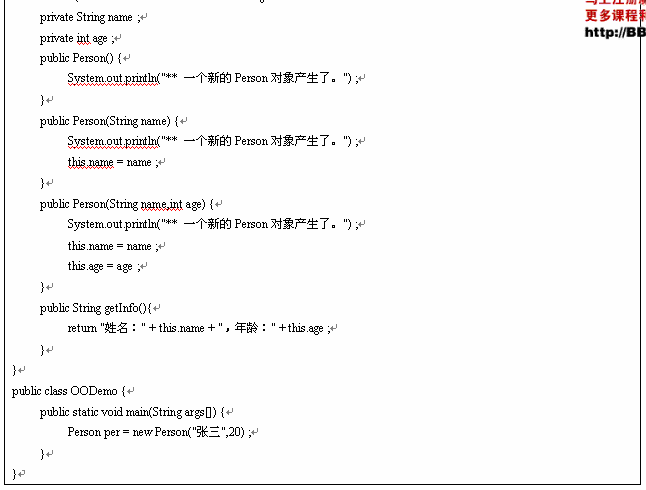
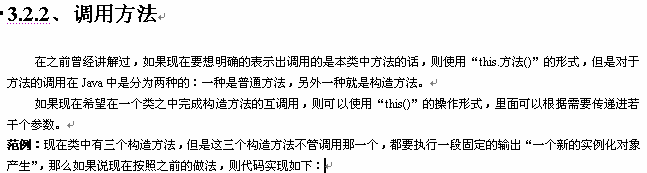
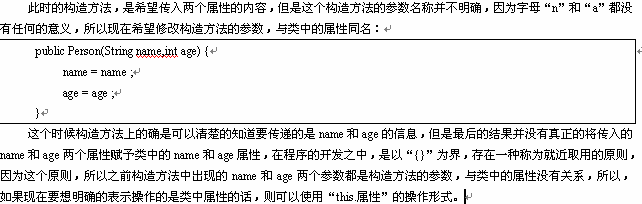
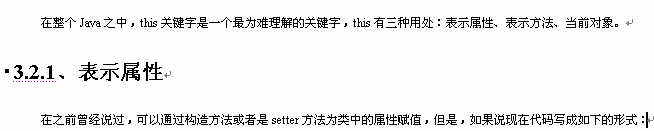
# 4、总结



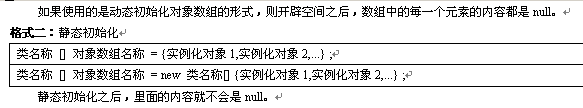
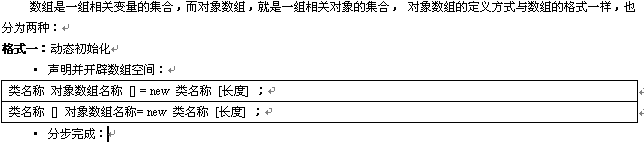
# 5、预习任务



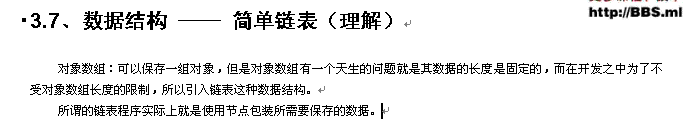
### 3.2、this 关键字（重点）



### 3.3、对象数组（重点）



## 数据结构——链表



**class** Node{

**private** String data;

**private** Node Next;

**public** Node(String data){

**this**.data = data;

}

**public** **void** setNext(Node next){

**this**.Next = next;

}

**public** Node getNext(){

**return** **this**.Next;

}

**public** String getData(){

**return** **this**.data;

}

}

**public** **class** Test {

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

Node root = **new** Node("根节点");

Node n1 = **new** Node("保存节点A");

Node n2 = **new** Node("保存节点B");

Node n3 = **new** Node("保存节点C");

Node n4 = **new** Node("保存节点D");

Node n5 = **new** Node("保存节点E");

root.setNext(n1);

n1.setNext(n2);

n2.setNext(n3);

n3.setNext(n4);

n4.setNext(n5);

//不知道循环多少次，使用while

Node currNode = root;

**while** (currNode != **null**){

System.*out*.println("数据:"+currNode.getData());

currNode = currNode.getNext();

}

}

}

结果如下：

数据:根节点

数据:保存节点A

数据:保存节点B

数据:保存节点C

数据:保存节点D

数据:保存节点E

当然也可以递归完成：

**public** **class** Test {

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

……

*printNode*(root);

}

**public** **static** **void** printNode(Node node){

**if**(node != **null**){

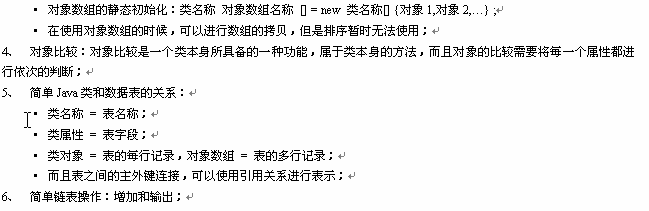
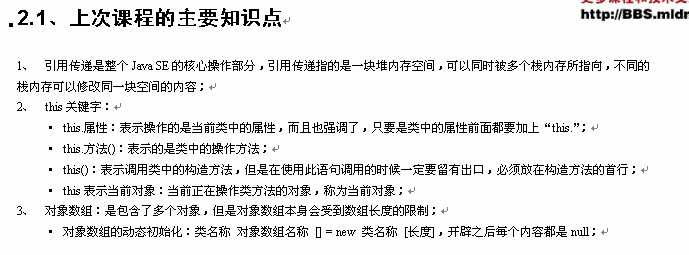
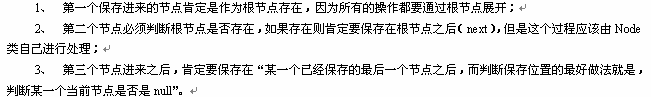
System.*out*.println(node.getData());

*printNode*(node.getNext());

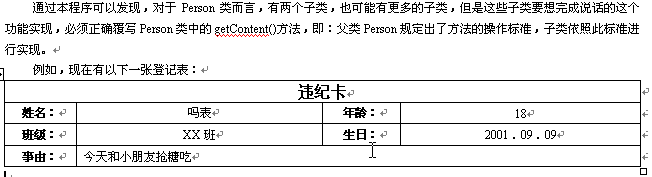
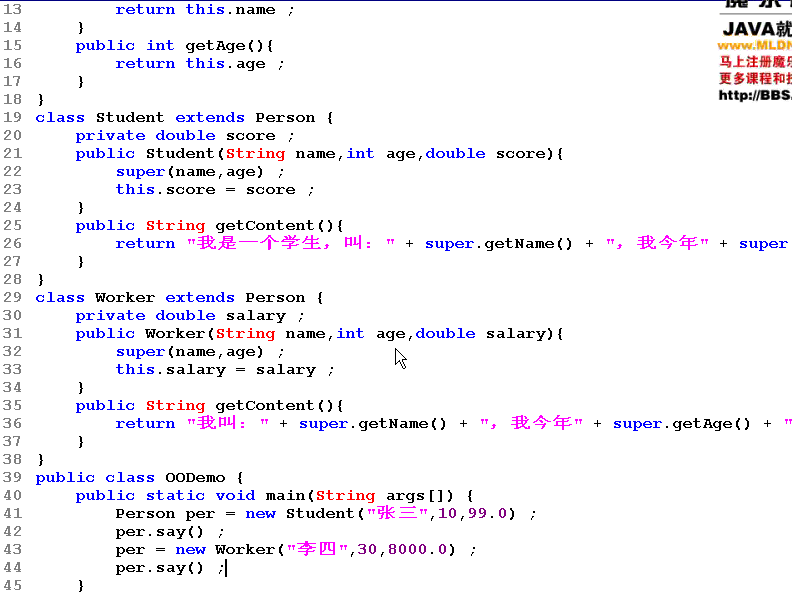
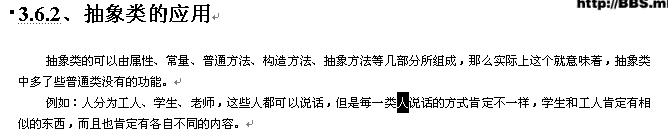
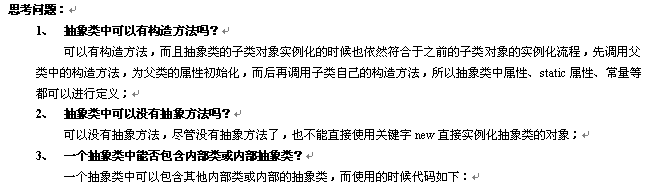
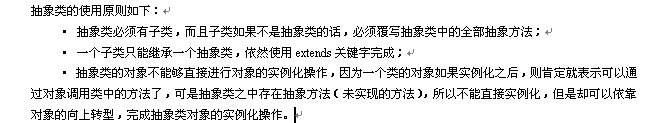
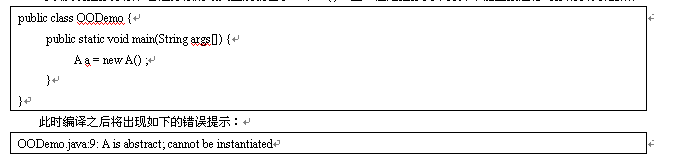
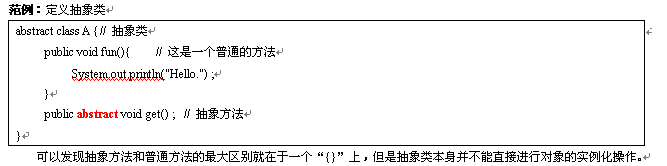
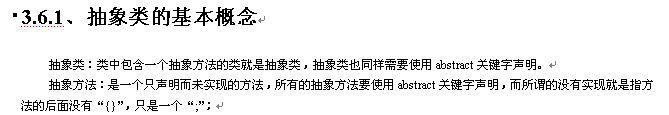
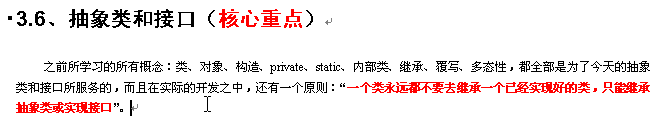
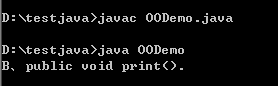
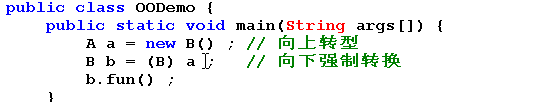
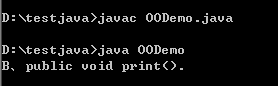
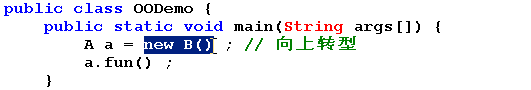
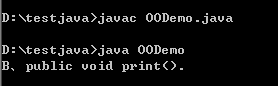
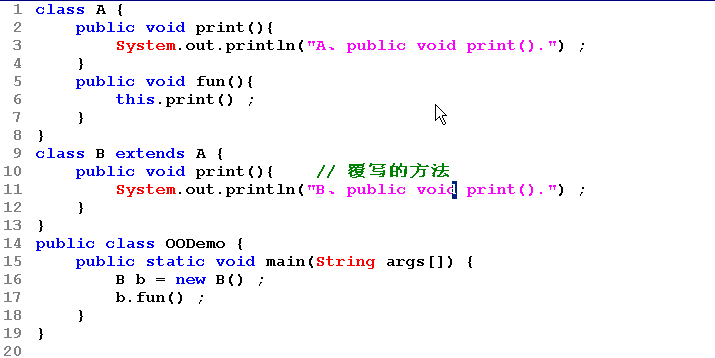
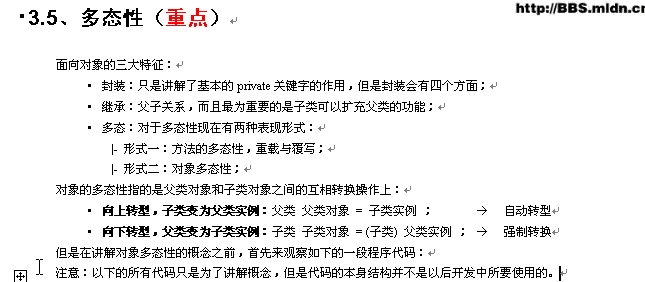
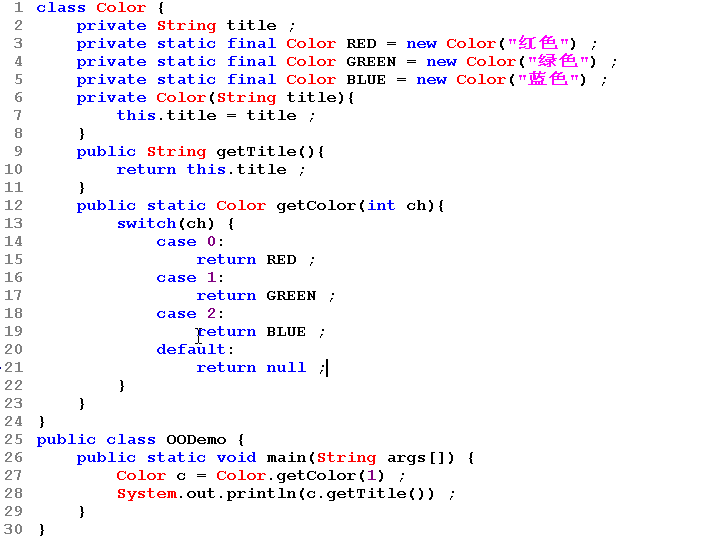
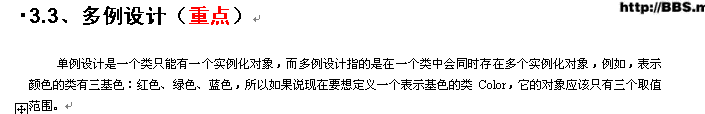
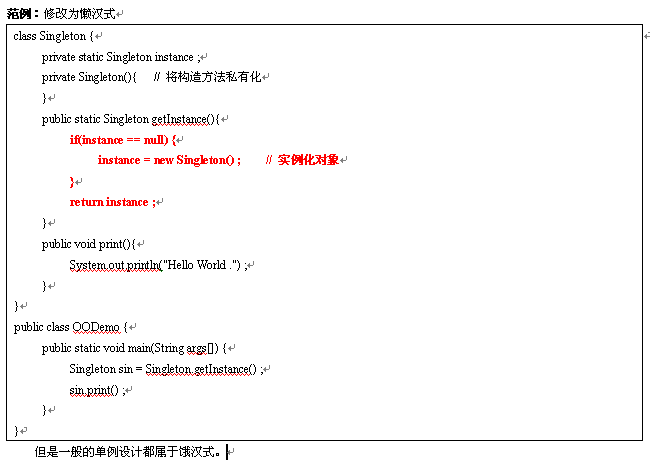
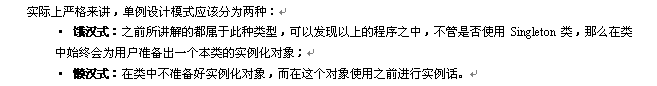
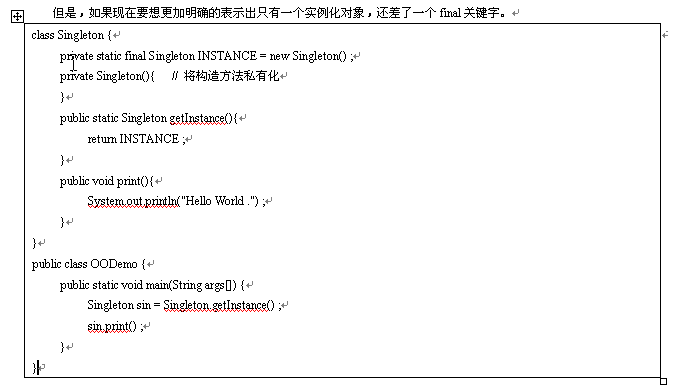
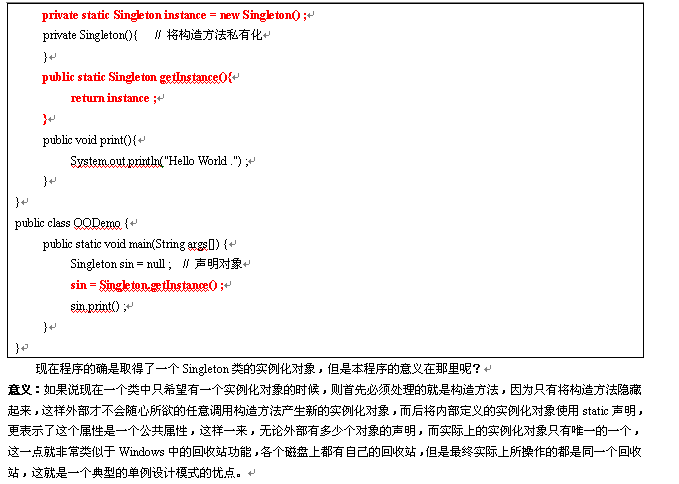
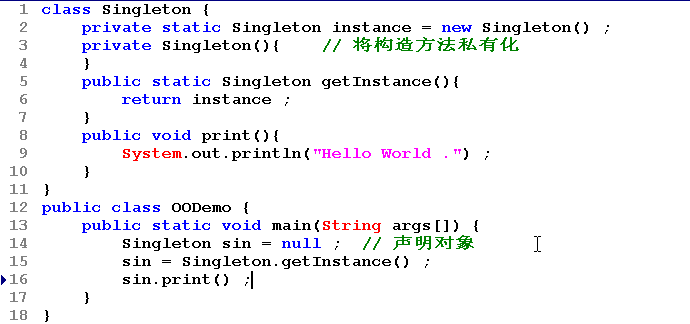
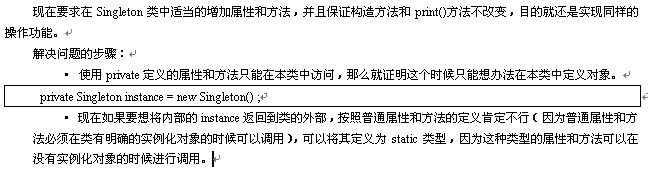
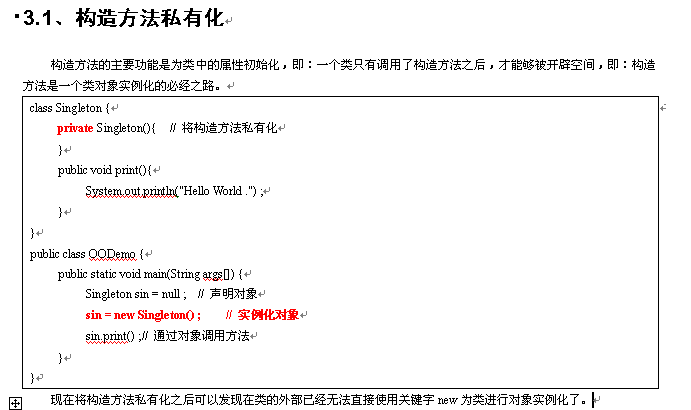
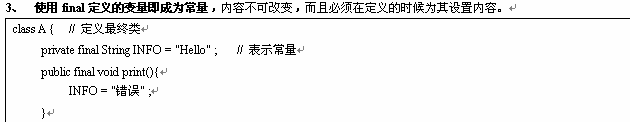
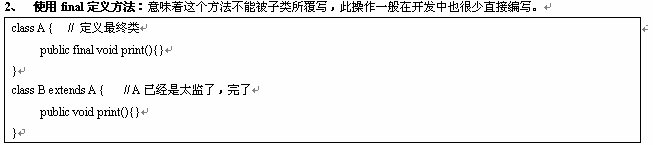
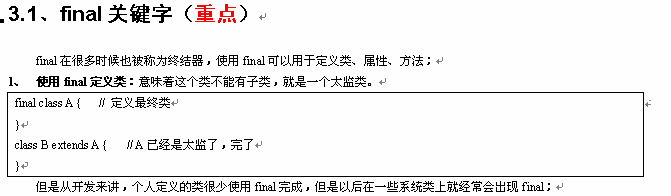
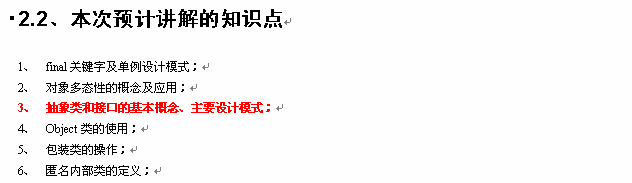
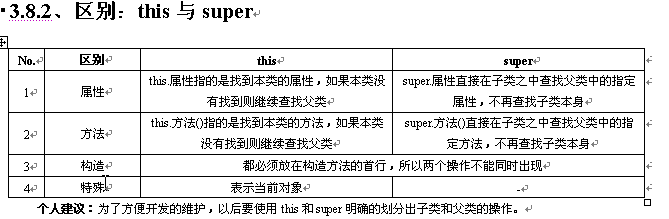
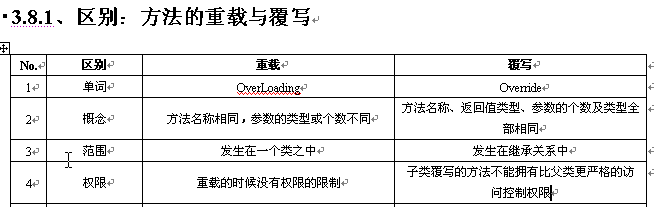
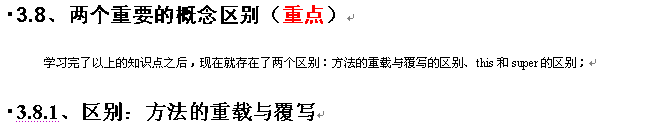
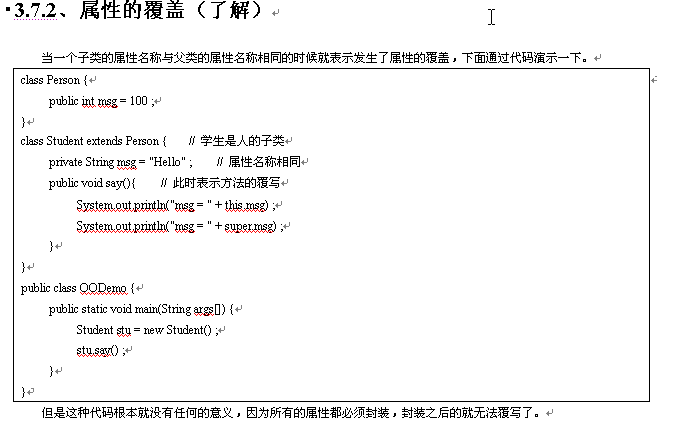
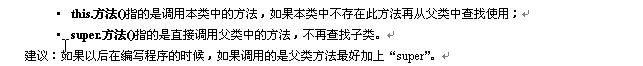
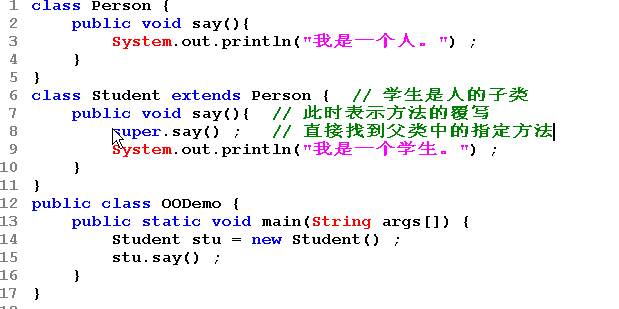
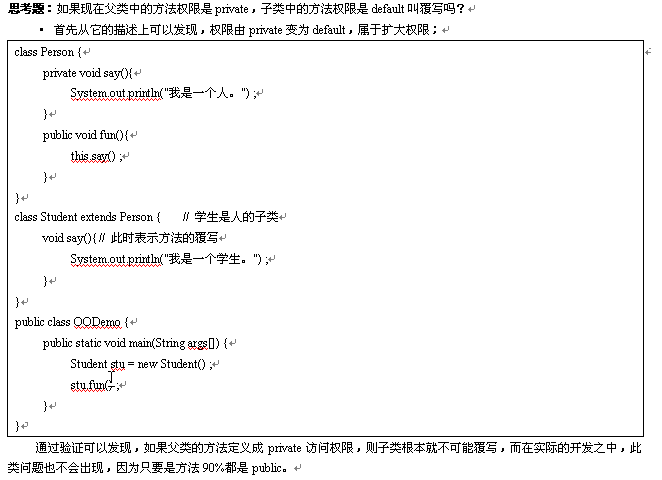
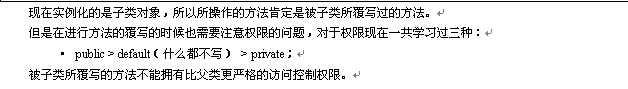
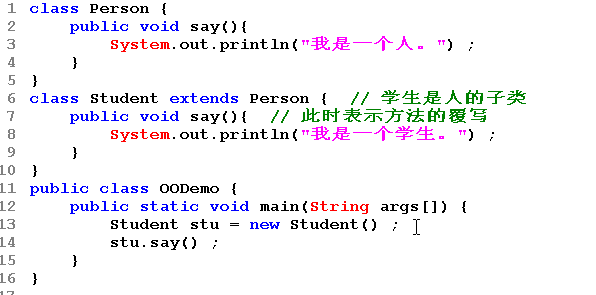
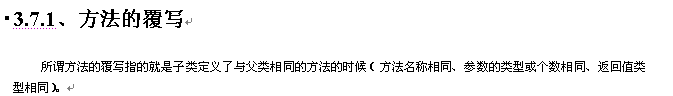
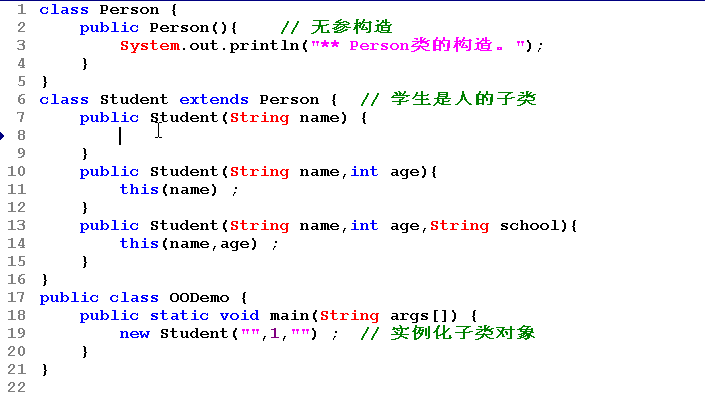
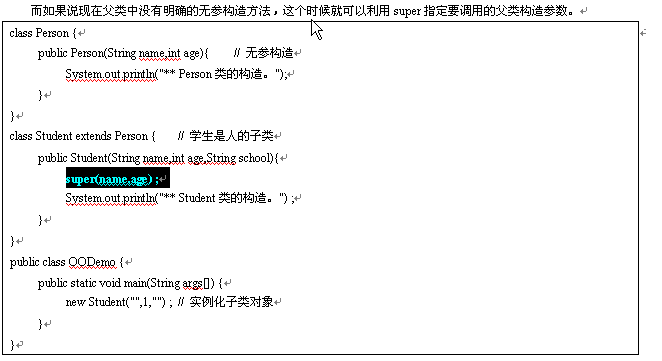
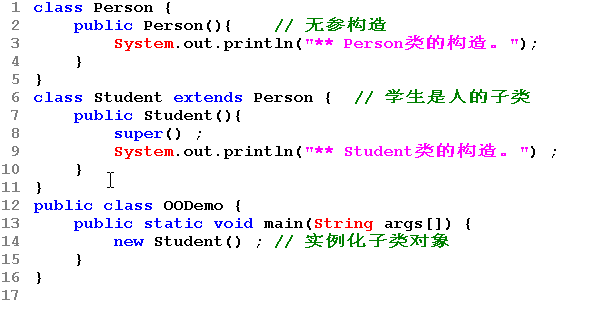
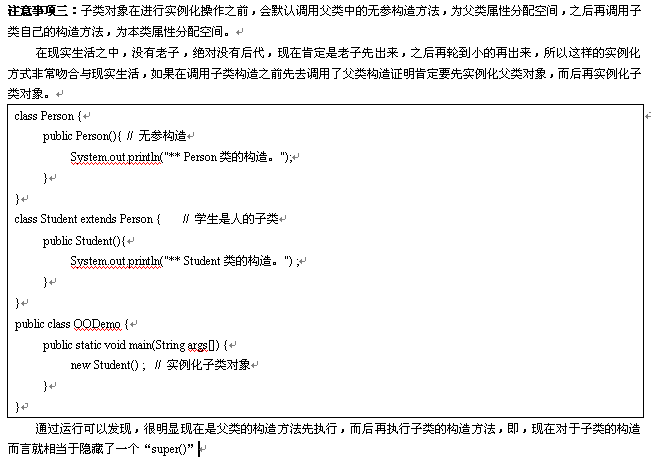
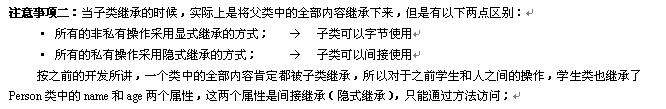
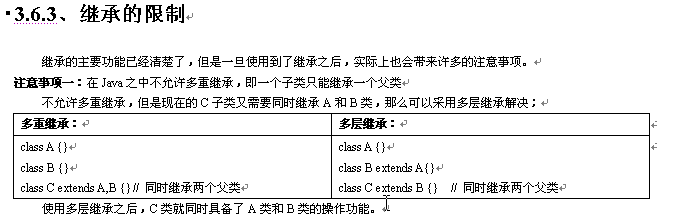
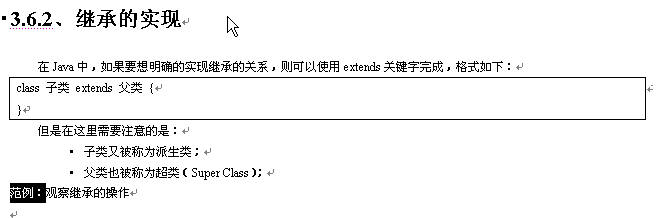
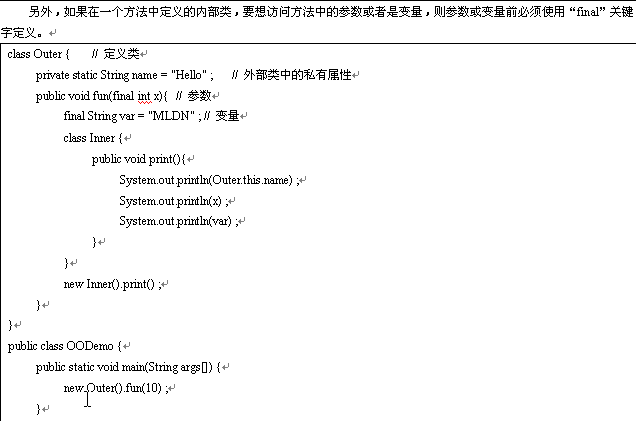
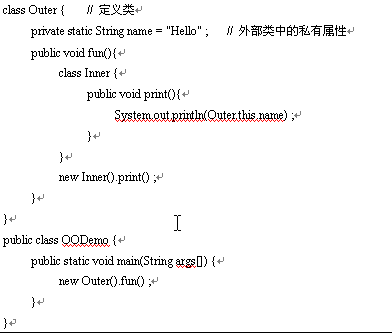
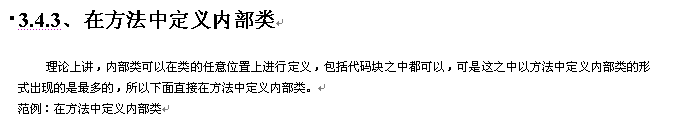
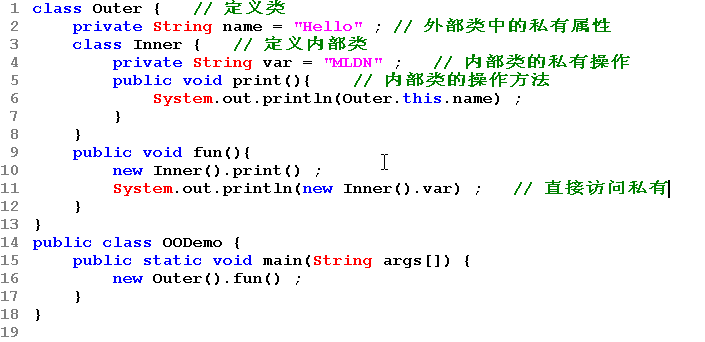
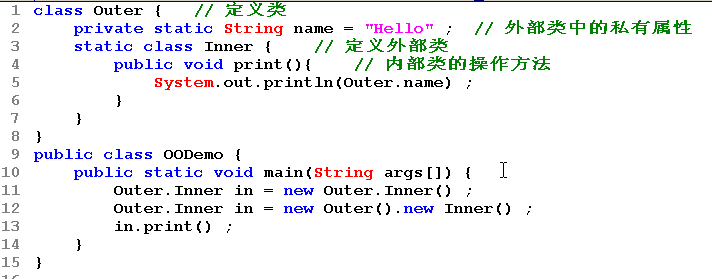
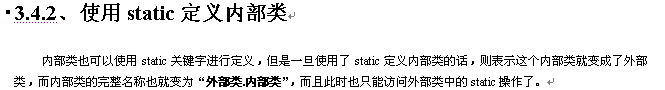
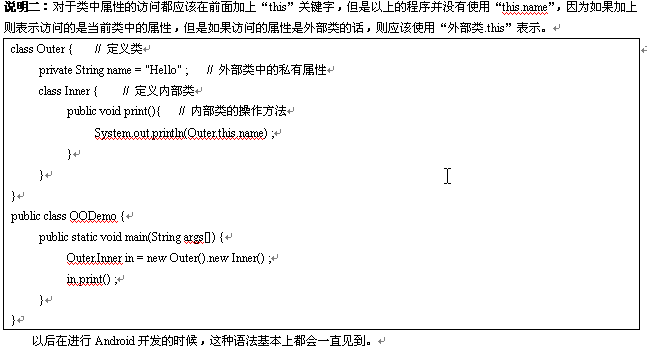
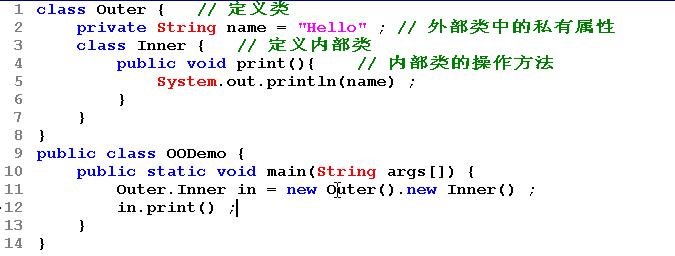
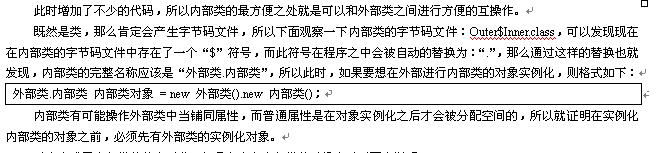
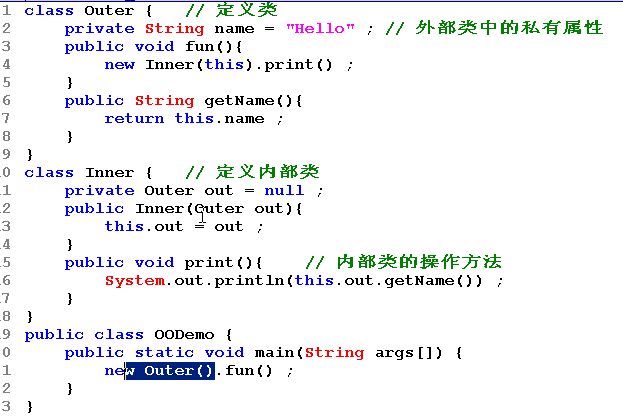
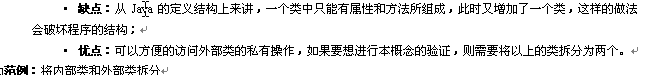
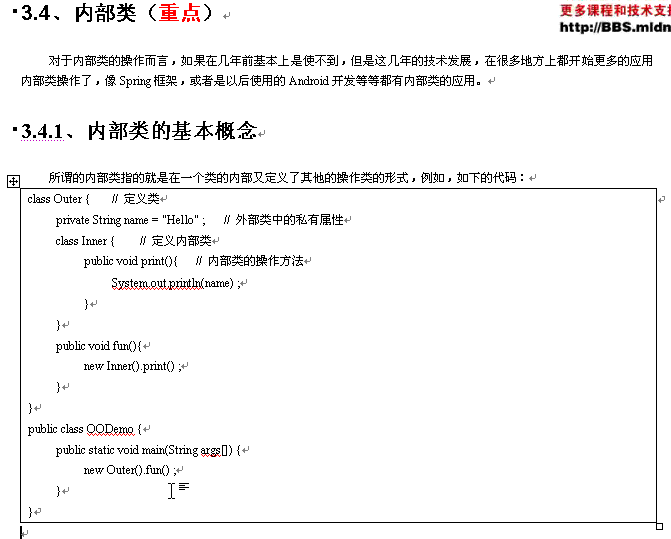
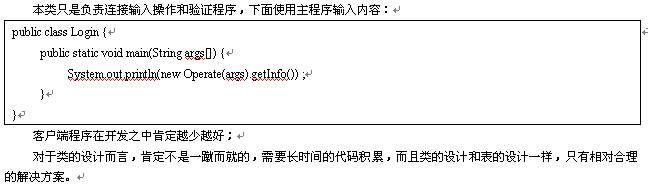
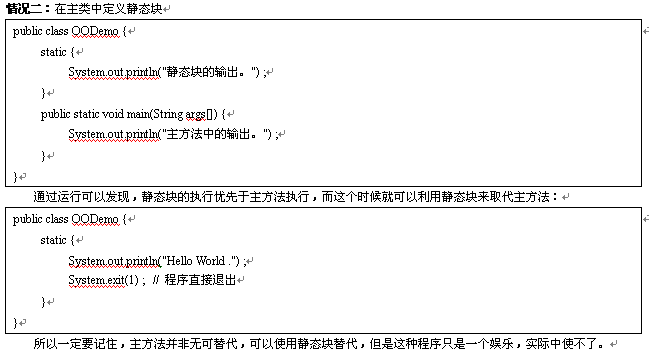
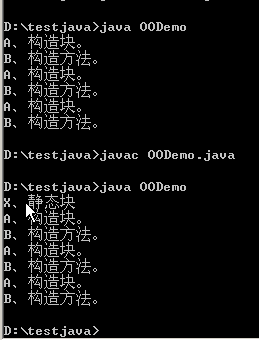
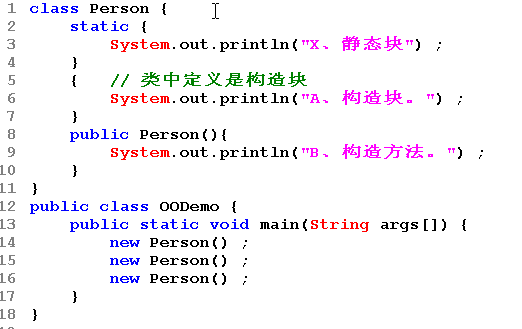
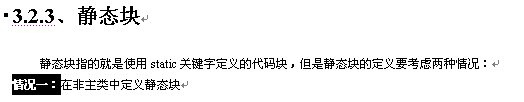
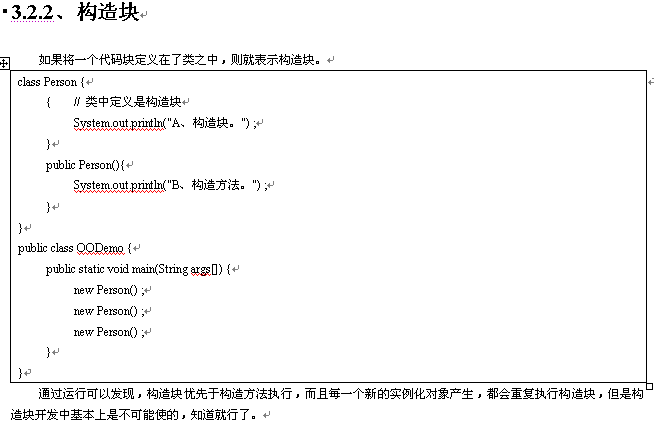
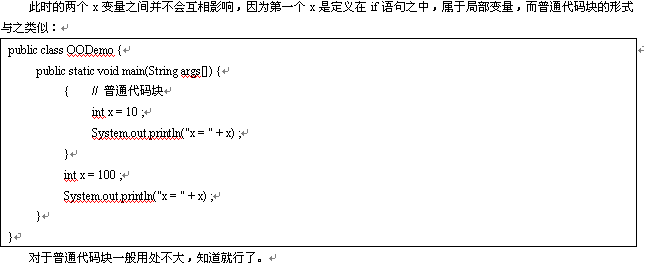
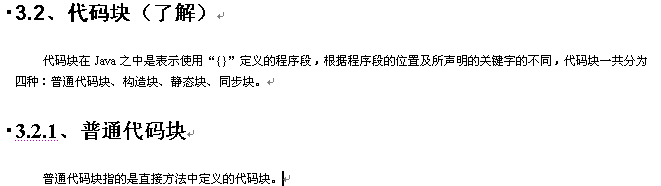
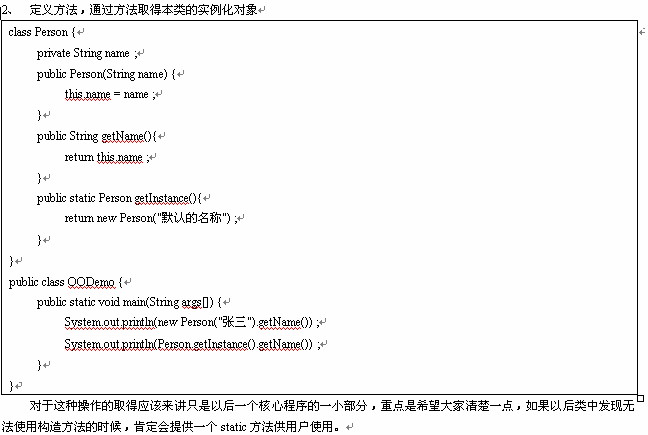
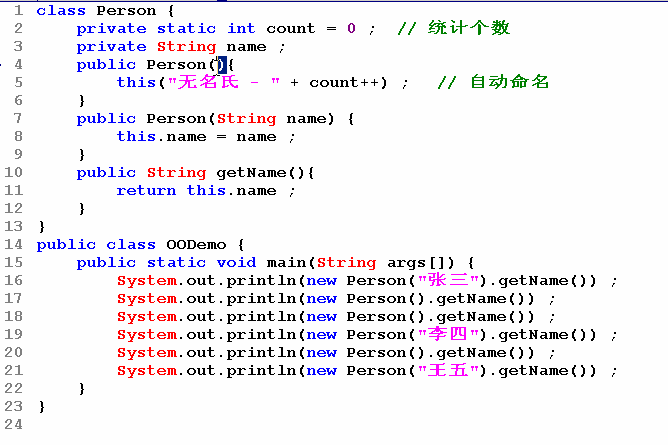
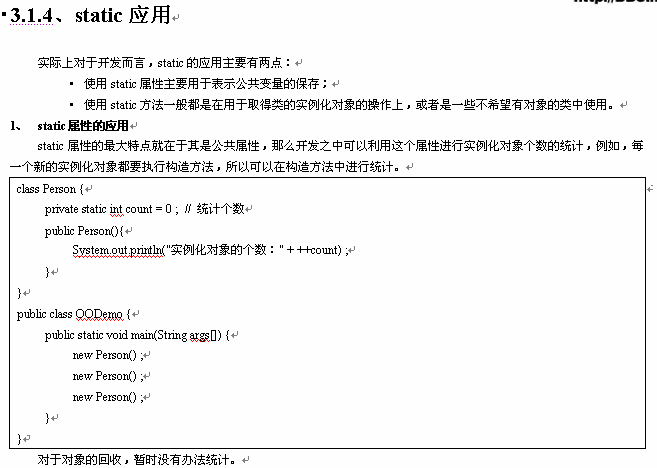
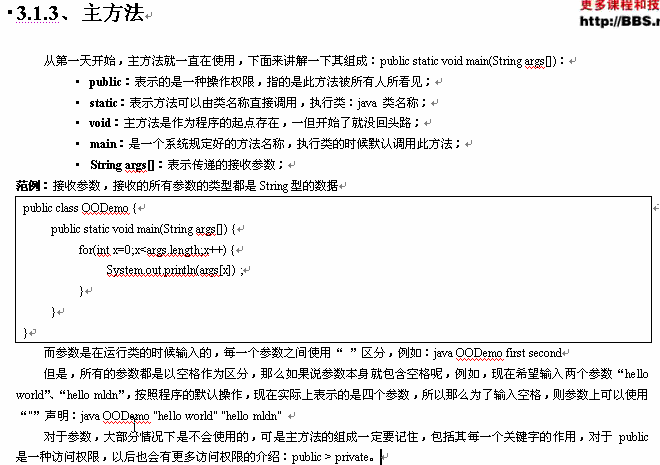
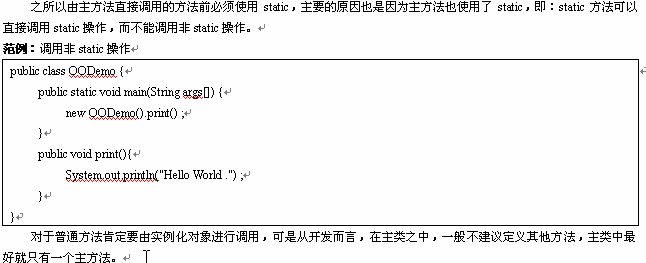
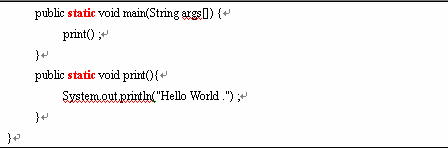
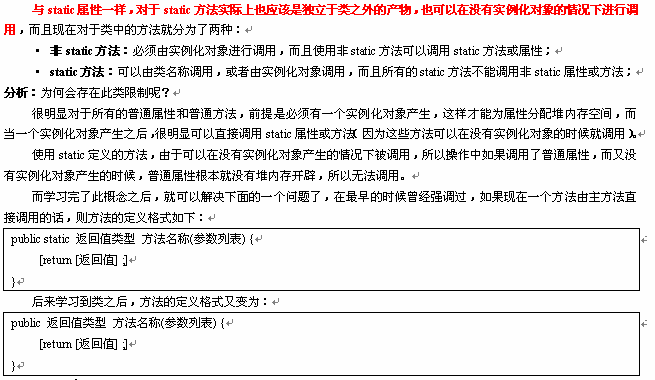
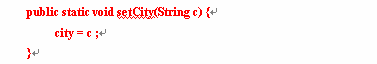
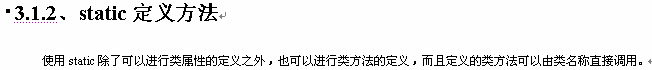
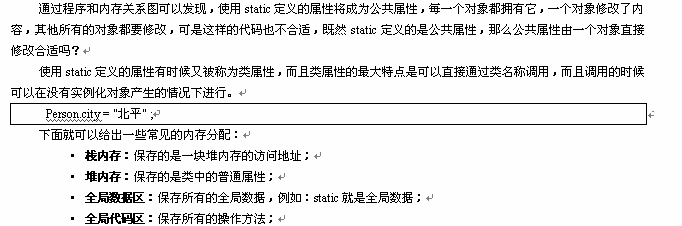
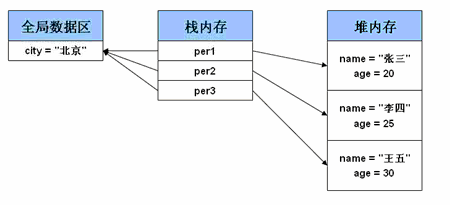
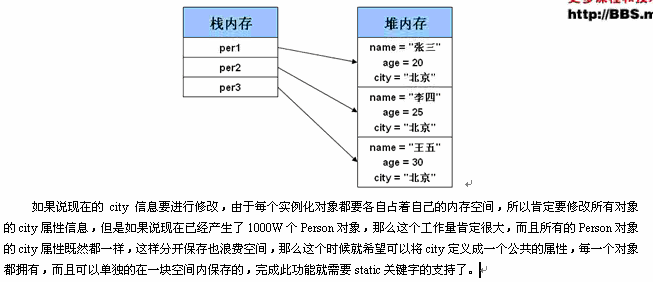
}

}

}



## Static 关键字



**abstract** **class** Person{

**public** **static** **final** **int** *RUN* = 1;

**public** **static** **final** **int** *DANCE* = 2;

**public** **static** **final** **int** *SING* = 3;

**public** **void** action(**int** code){

**switch**(code){

**case** *RUN*:

**this**.run();

**break**;

**case** *DANCE*:

**this**.dance();

**break**;

**case** *SING*:

**this**.sing();

}

}

**public** **abstract** **void** run();

**public** **abstract** **void** dance();

**public** **abstract** **void** sing();

}

**class** Baby **extends** Person{

**public** **void** run(){

System.*out*.println("I am a baby,I am running!");

}

**public** **void** dance(){

System.*out*.println("I am a baby,I am dancing!");

}

**public** **void** sing(){

System.*out*.println("I am a baby,I am singing!");

}

}

**public** **class** Test {

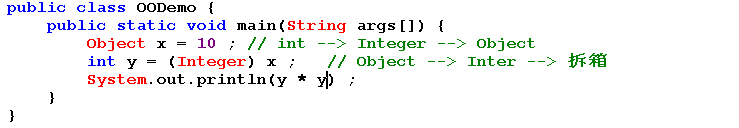
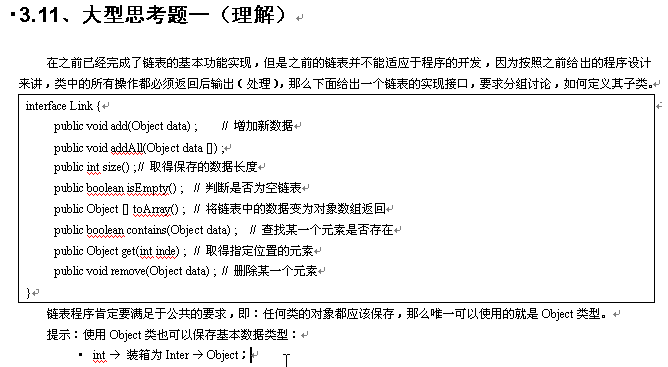
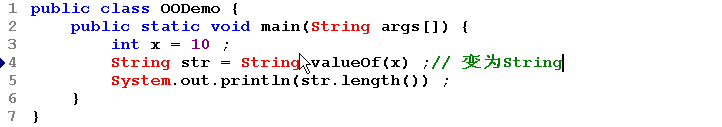
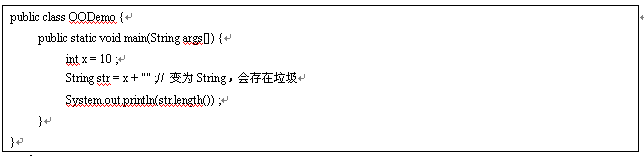
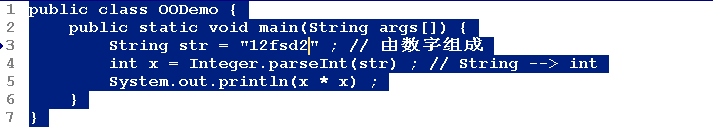
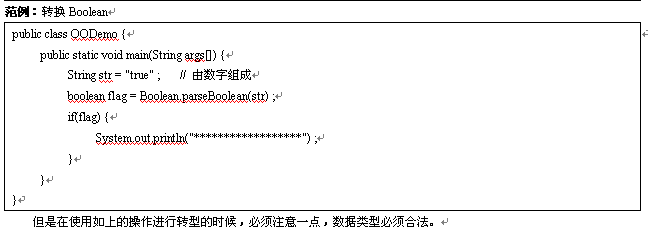
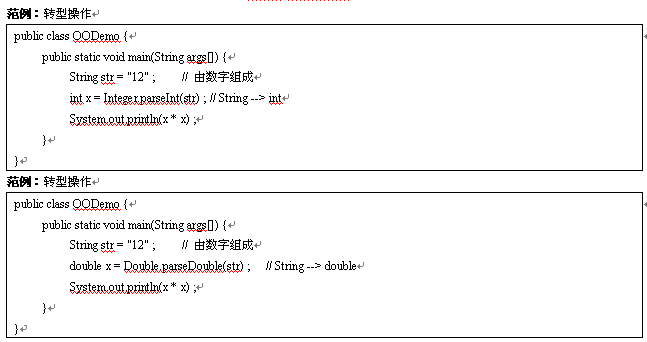
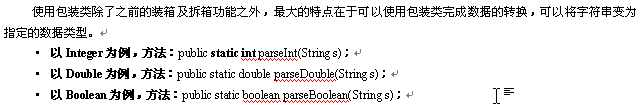
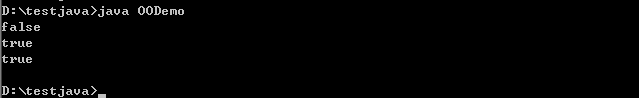
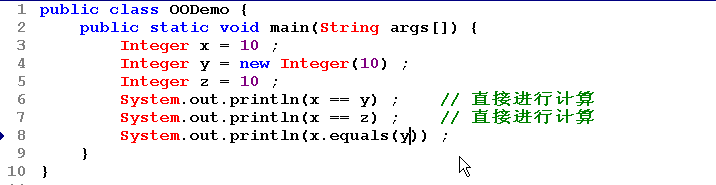
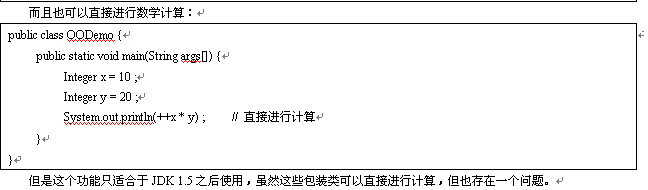
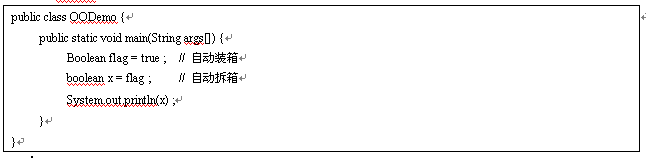
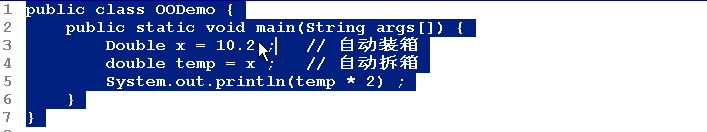
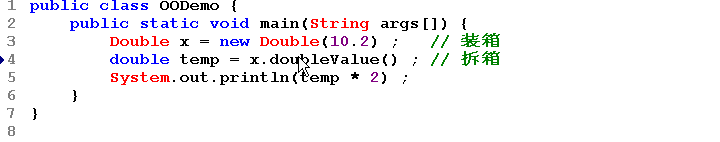
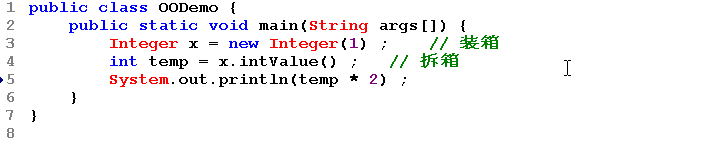
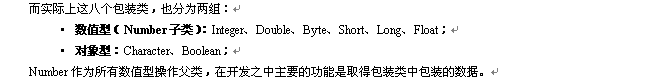
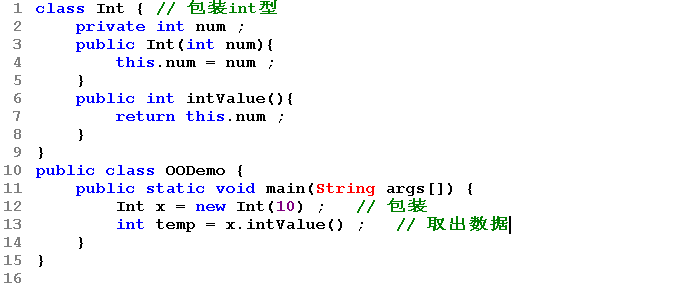
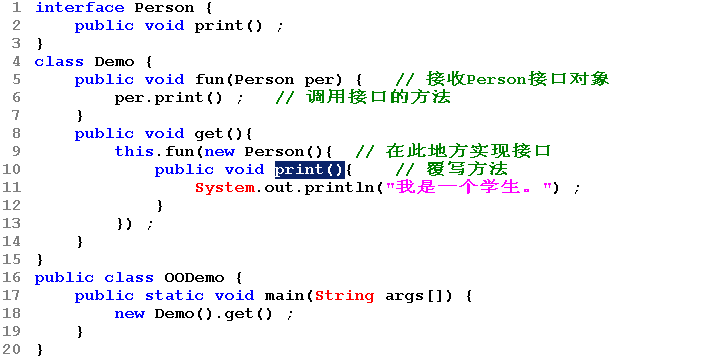
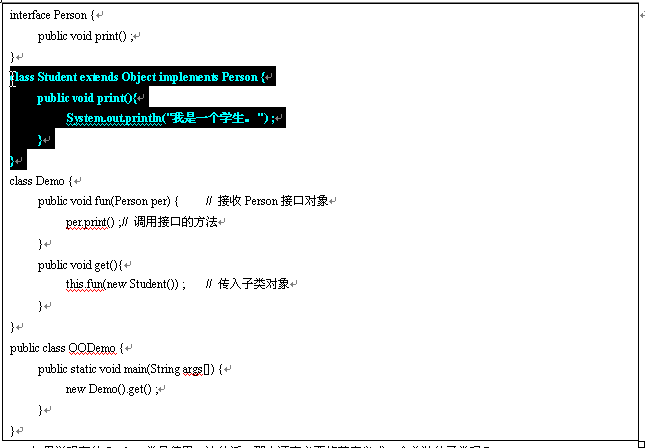
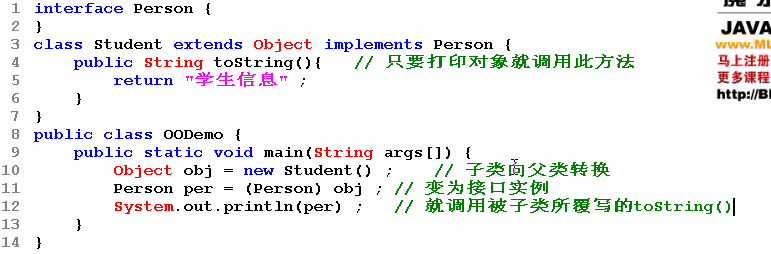
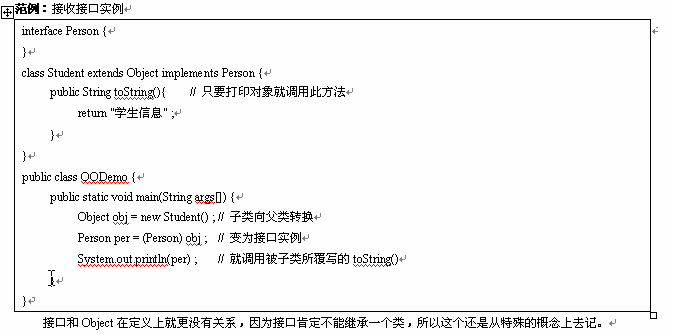
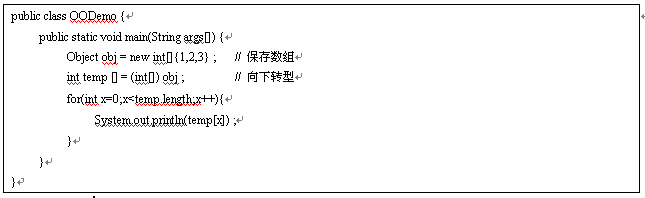
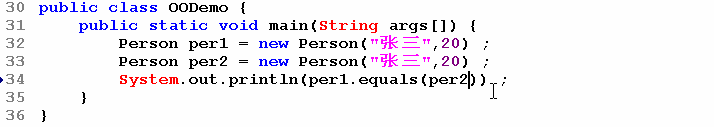
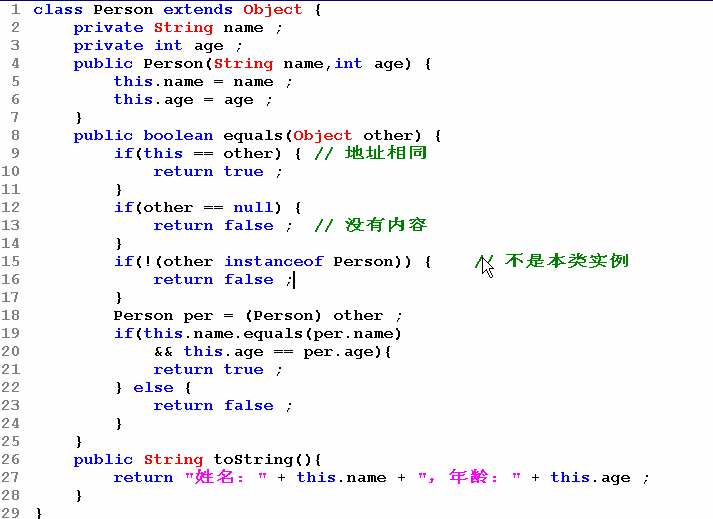
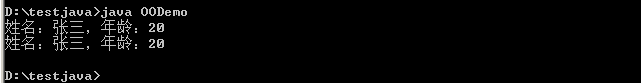
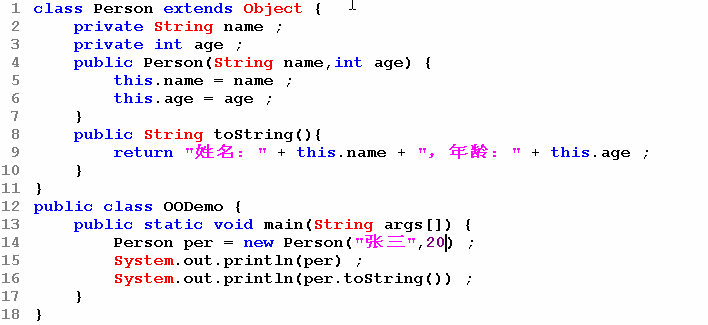
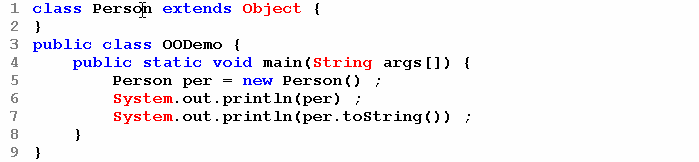
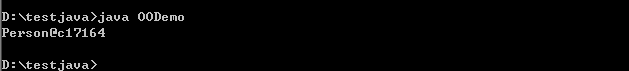
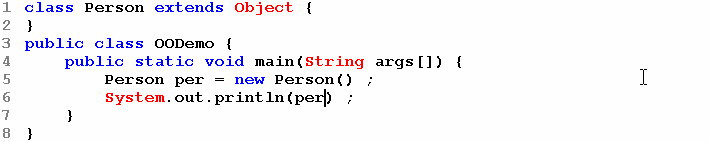
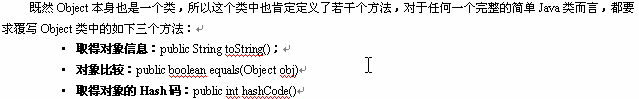
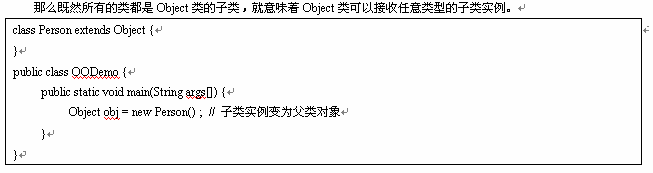
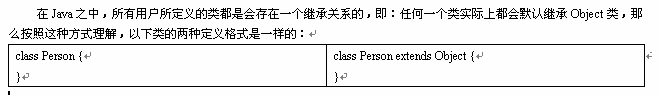
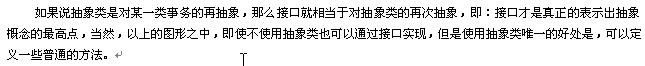
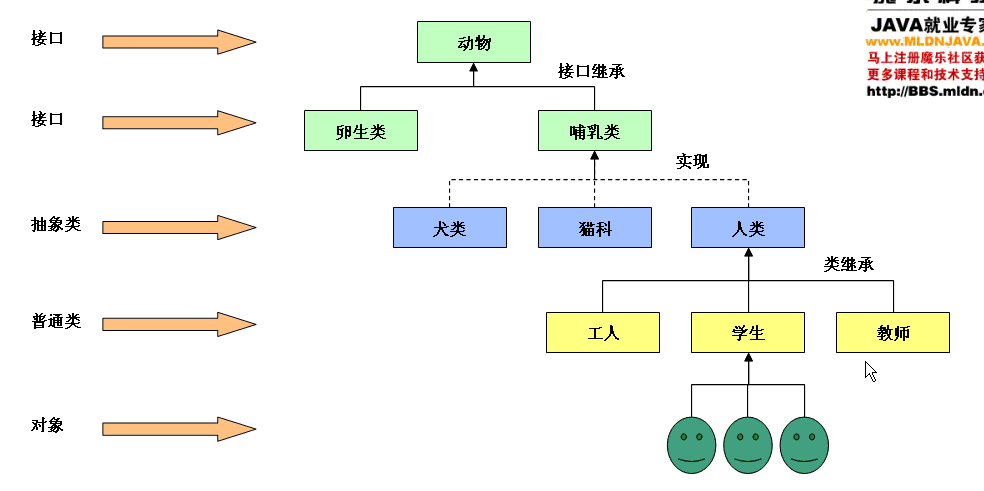
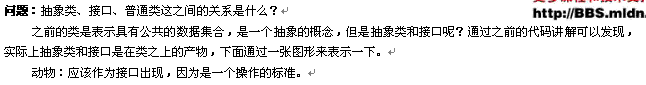
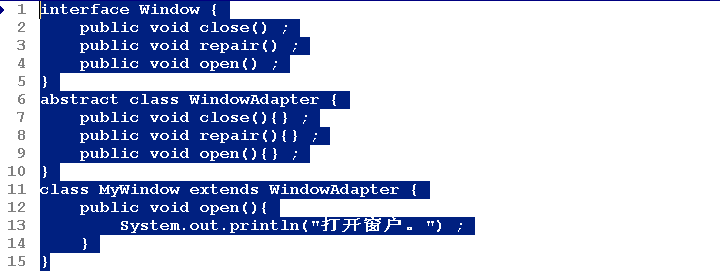
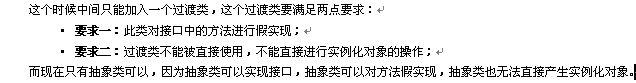
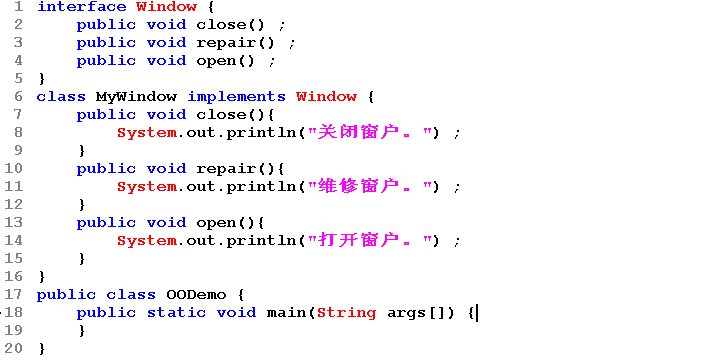
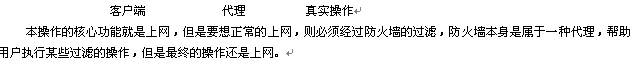
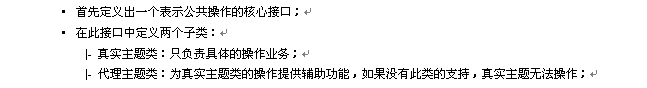
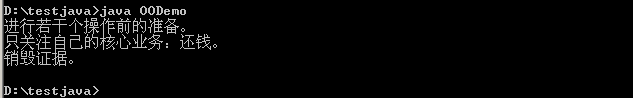
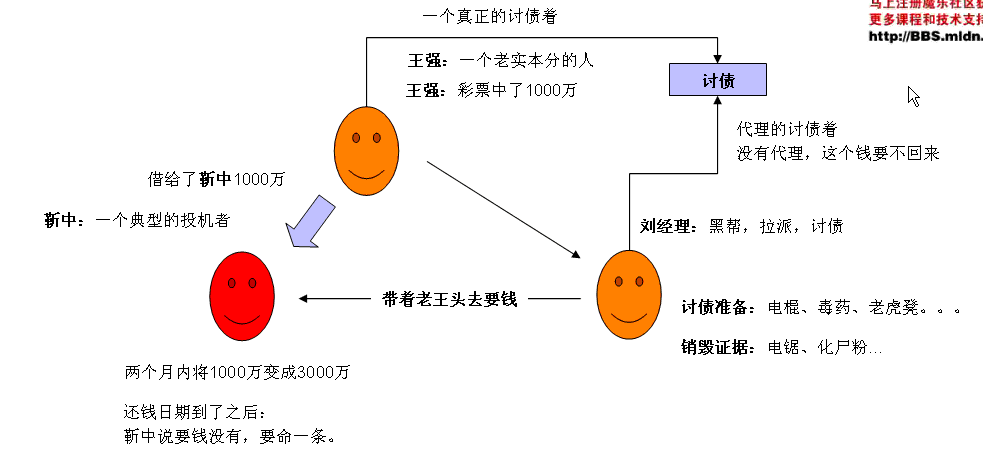
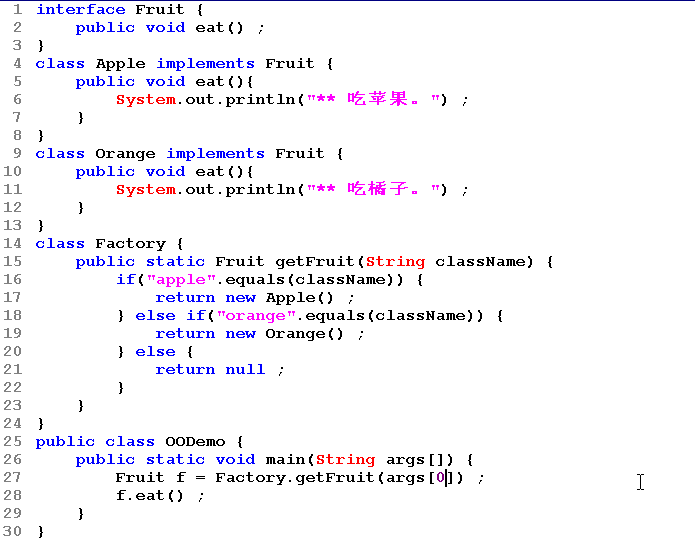
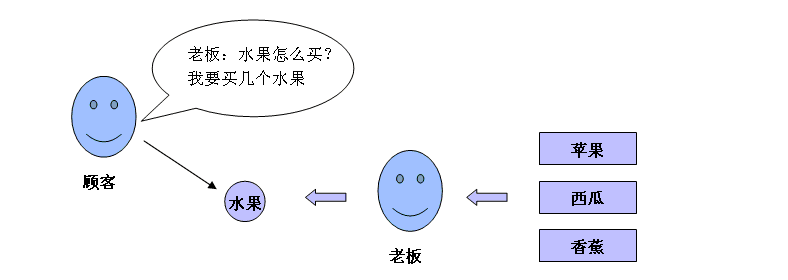
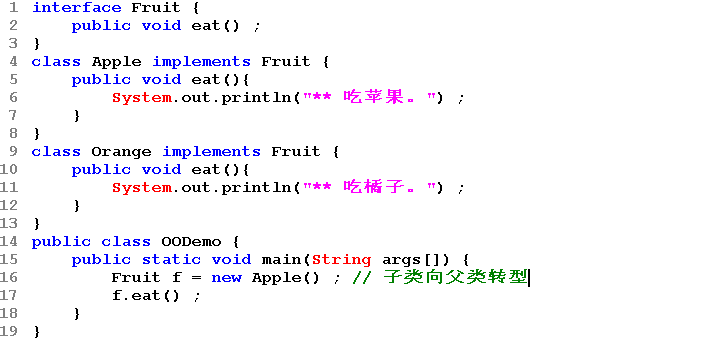
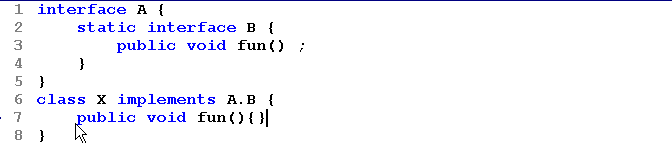
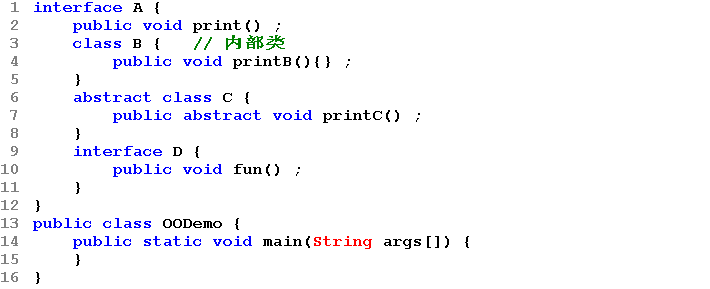
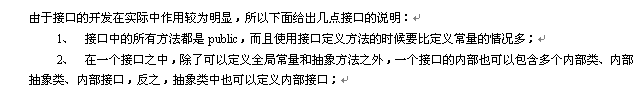
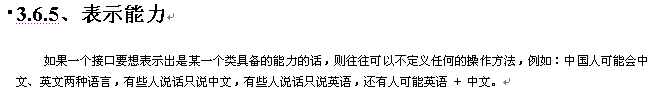
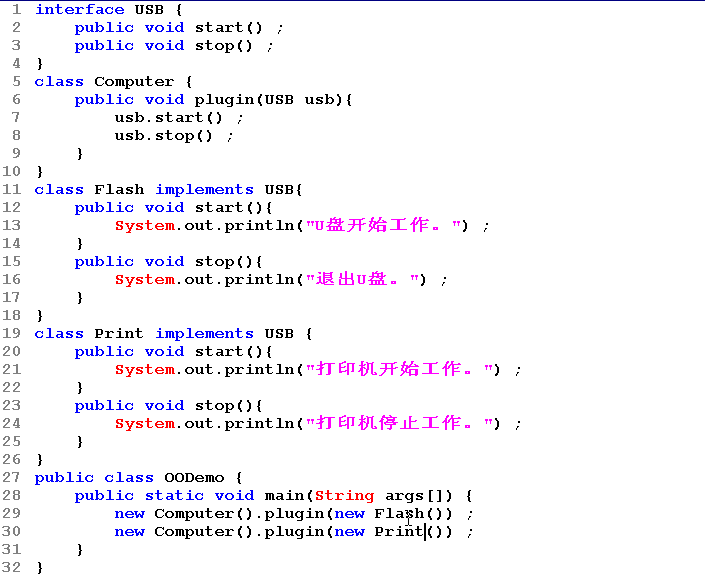
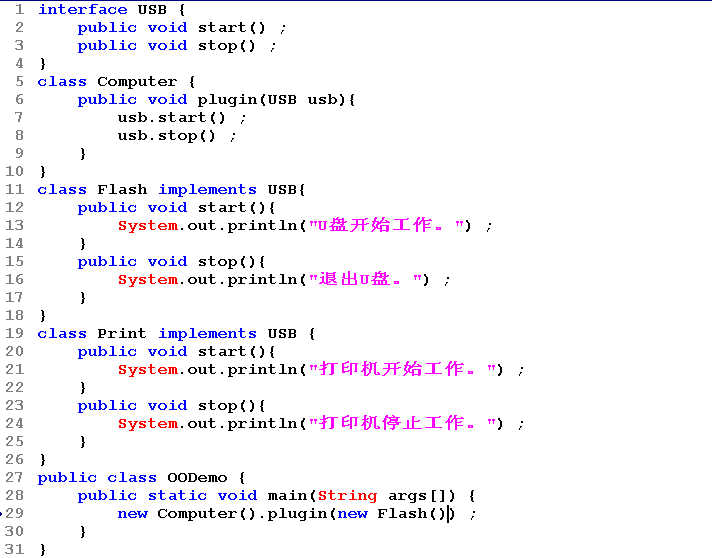
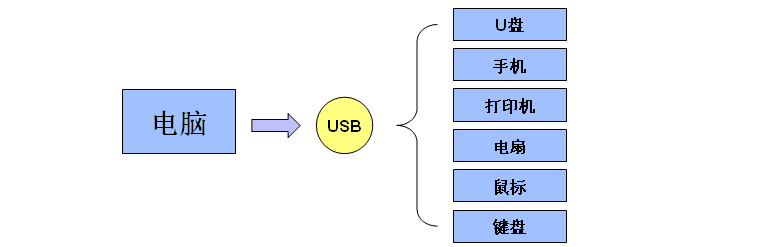
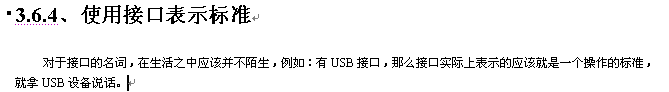
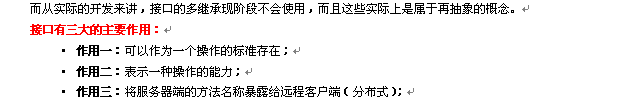
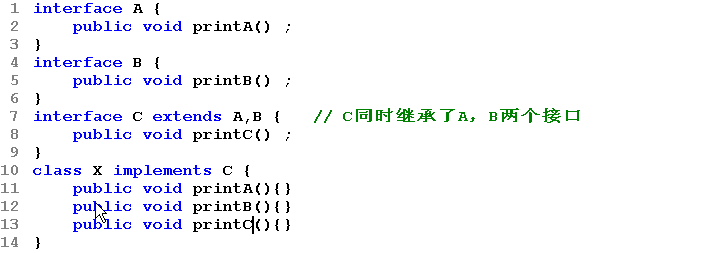
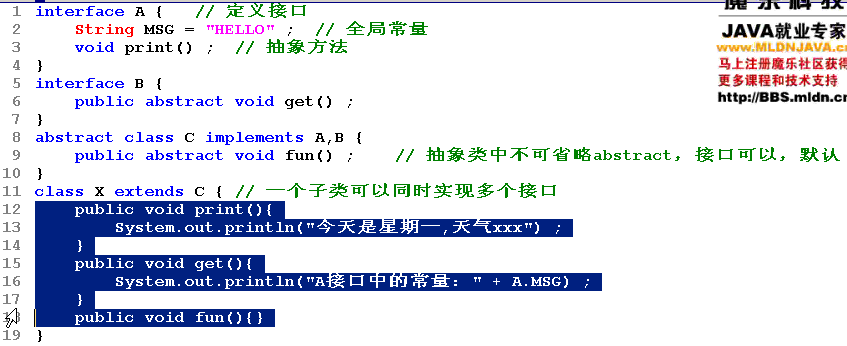
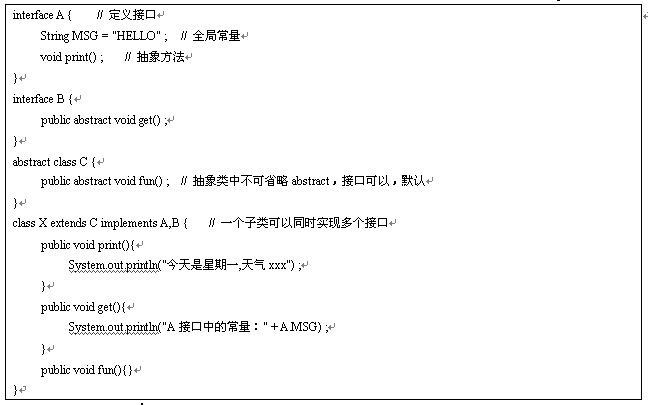
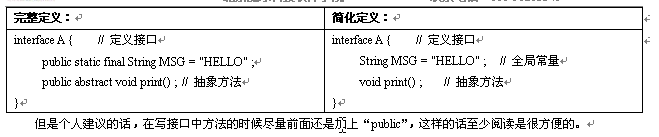
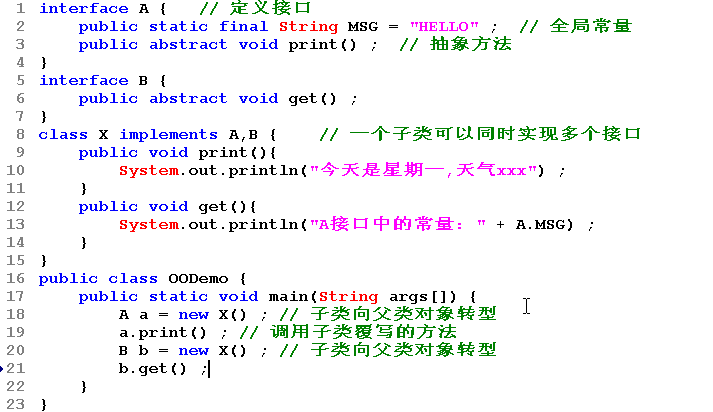
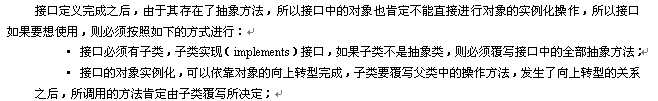
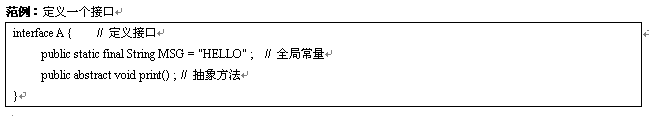
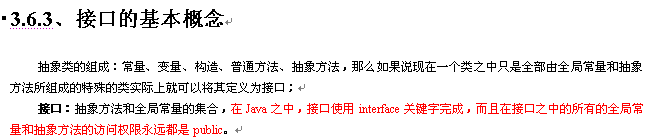
**public** **static** **void** main(String args[]){

Person baby = **new** Baby();

baby.action(Person.*RUN*);

}

}



**interface** Link{

**public** **void** add(Object data); //添加新数据

**public** **void** addAll(Object data []);

**public** **int** size(); //取得保存的数据长度

**public** **boolean** isEmpty(); //判断是否为空链表

**public** Object [] toArray(); //将链表中的数据变为对象数组返回

**public** **boolean** contains(Object data); //查找一个元素是否存在

**public** Object get(**int** inde); //取得制定位置的元素

**public** **void** remove(Object data); //删除某一个元素

}

**class** LinkImpl **implements** Link{

**private** **class** Node { //定义节点类

**private** Object data ; //数据使用Object保存

**private** Node next; //下一个节点

**public** Node(Object data){

**this**.data = data;

}

**private** **void** addNode(Node newNode){

**if**(**this**.next == **null**){

**this**.next = newNode;

}**else**{

**this**.next.addNode(newNode);

}

}

**private** **void** toArrayNode(){

LinkImpl.**this**.retData[LinkImpl.**this**.foot++] = **this**.data; //取当前数据

**if**(**this**.next != **null**) {//还有下一个元素

**this**.next.toArrayNode();

}

}

**private** **void** removeNode(Node previous,Object data){

**if**(**this**.data.equals(data)){

previous.next = **this**.next;

}**else**{

**this**.next.removeNode(**this**, data);

}

}

}

**private** Node root; //定义根节点

**private** Object retData []; //此数组用于保存返回的数据

**private** **int** foot = 0; //脚标

**private** **int** count = 0;

**public** **void** add(Object data){

**if**(data == **null**){

**return**; //不能增加null数据

}

Node newNode = **new** Node(data); //封装数据

**if**(**this**.root == **null**){

**this**.root = newNode;

}**else**{

**this**.root.addNode(newNode);

}

**this**.count++;

}

**public** **void** addAll(Object data []){

**if**(data == **null**){

**return**;

}

**for**(**int** x = 0;x<data.length;x++){

**this**.add(data[x]);

}

}

**public** **int** size(){

**return** **this**.count;

}

**public** **boolean** isEmpty(){

**return** **this**.count == 0;

}

**public** Object [] toArray(){

**if**(**this**.root == **null**){

**return** **null**;

}

**this**.retData = **new** Object[**this**.count];//根据已有的数据开辟空间

**this**.foot = 0;

**this**.root.toArrayNode();

**return** **this**.retData;

}

**public** **boolean** contains(Object data){

Object result [] = **this**.toArray();

**for**(**int** x = 0;x<result.length;x++){

**if**(result[x].equals(data)){

**return** **true**;//满足当前条件

}

}

**return** **false**;

}

**public** Object get(**int** index){

**if**(index>**this**.count){

**return** **null**;

}

**return** **this**.toArray()[index];

}

**public** **void** remove(Object data){

**if**(**this**.contains(data)){

**if**(**this**.root.data.equals(data)){

**this**.root = **this**.root.next;

}**else**{

**this**.root.next.removeNode(**this**.root,data);

}

**this**.count--;

}

}

}

**public** **class** Test {

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

Link all = **new** LinkImpl();

all.add("A");

all.add("B");

all.add("C");

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

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

Object temp [] = all.toArray();

**for**(**int** x = 0;x<temp.length;x++){

System.*out*.println(temp[x]);

}

System.*out*.println(all.contains("A"));

System.*out*.println(all.contains("c"));

System.*out*.println(all.get(1));

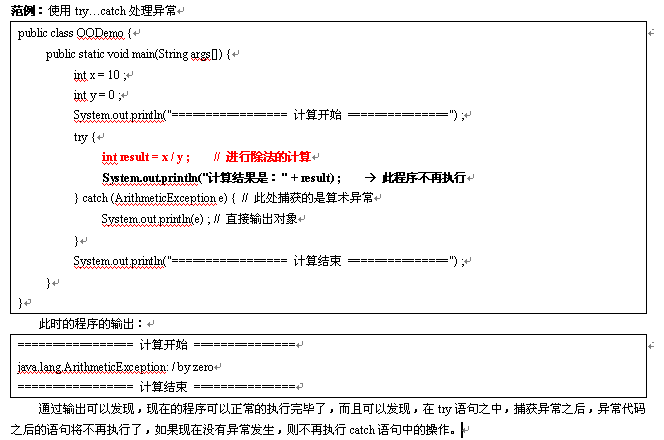
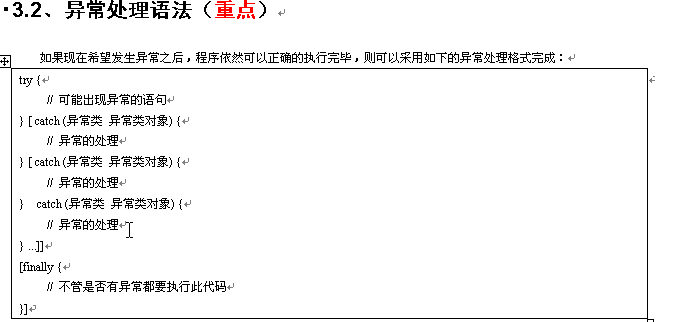
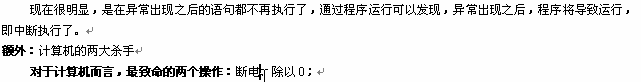
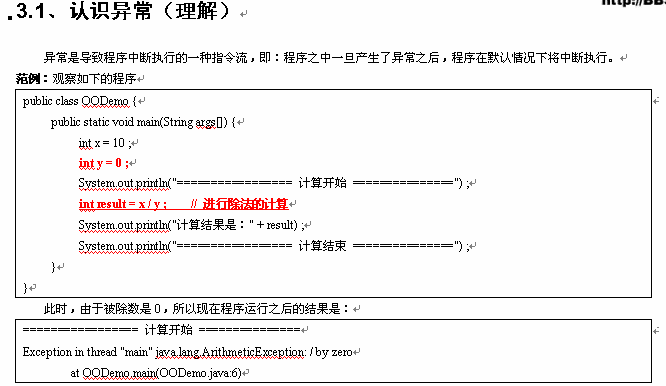
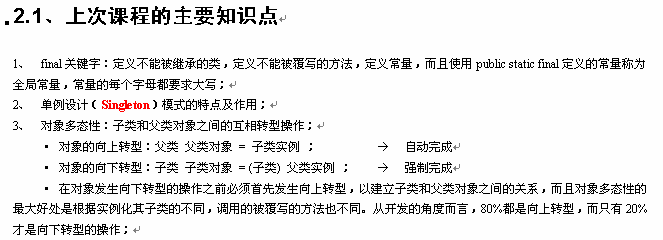
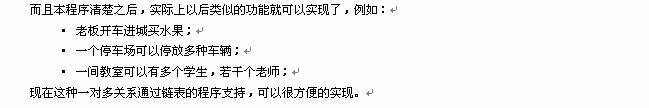
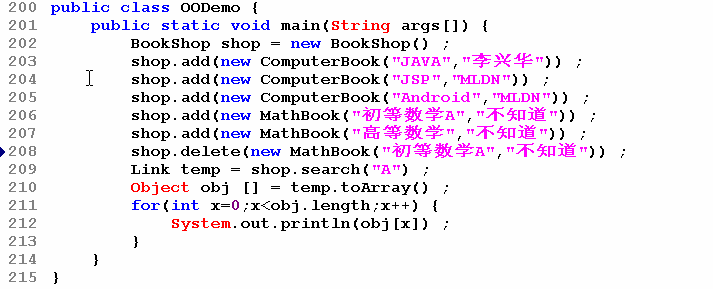
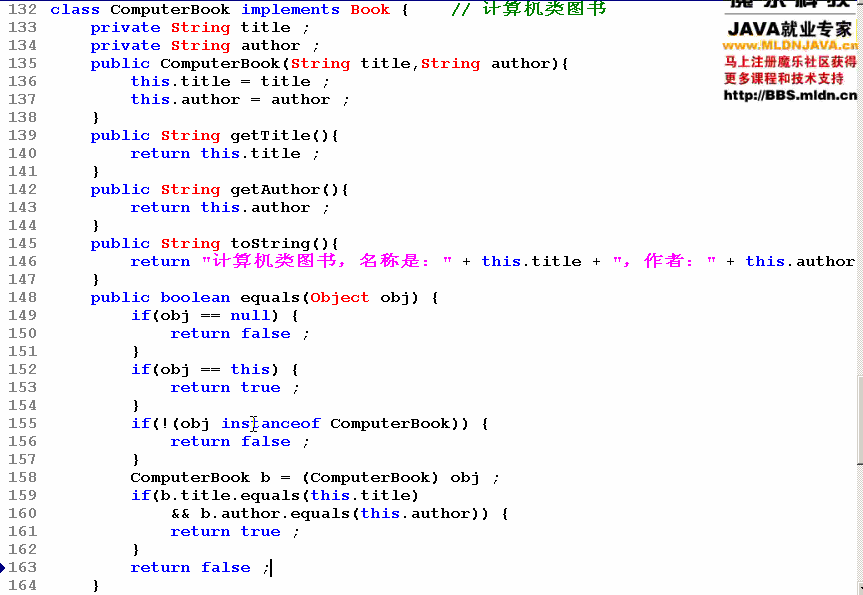
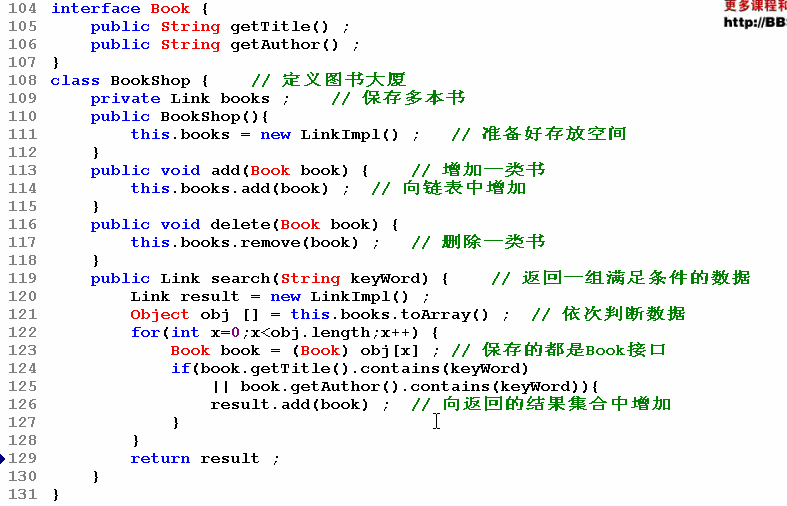
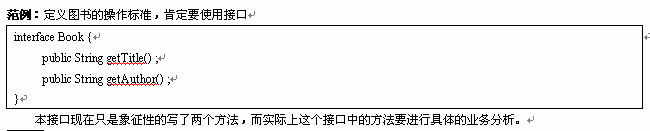
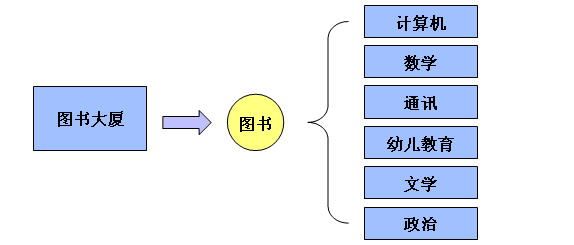
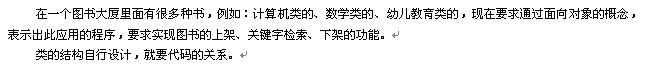
System.*out*.println(all.get(10));

all.remove("A");

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

}

}



**public** **class** Test {

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

**int** x = 10;

**int** y = 0;

**try**{

**int** result = x/y;

System.*out*.println("计算结果是"+result);

}**catch**(ArithmeticException e){

e.printStackTrace(); //更详细

//System.out.println(e);

}**finally** {

System.*out*.println("不管是否错误");

}

}

}

