# 一、Java概述

## 1.1 Java语言特点和发展历史

### 1.1.1 Java的历史发展

1991年当时SUN公司为了完成一个项目，在詹姆斯.高斯林的带领下开发一种新的语言，命名为ock，注册的时候发现被别人注册了。所以后来Java之父高斯林就把新的语言命名为他喜欢咖啡的名称Java，而且注册商标的团也是一杯冒着热气的咖啡。

Java从1.1版本发布以来经历了15个版本，然后在此期间SUN公司大概是在09年被oracle公司收购了，Java就变成了Oracle的产品。Java的产品可以在Oracle的官网上下载。

### 1.1.2 Java的特点

（1）面向对象特性

（2）安全性

（3）跨平台性

Java在近10几年都是程序语言应用排名第一，Java的后续开发学习应用主要有两个方向。第一、做web框架开发，信息化系统、网站建设，后续的学习课程包括Java Web开发、框架应用开发springboot、springCloud微服务开发；第二、做Android移动应用开发。

## 1.2 Java开发环境安装配置及编译原理

### 1.2.1 Java开发环境

1、Java运行环境 JDK

JDK和JRE的区别

JRE (Java runtime environment)Java运行环境

JDK(Java Development kit)Java开发工具包

Java开发工具包=Java运行环境+类库

综上所述，所以我们学习开发的同学们应该装JDK。

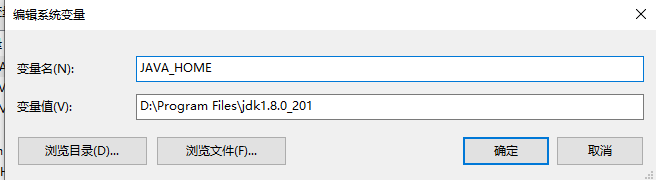
安装和配置的步骤：

第一、首先安装JDK在电脑上的某个位置。

第二、配置其实就是将JDK安装根目录下的bin目录的路径放入环境变量中就可以了。

（1）

新建一个环境变量叫JAVA\_HOME，然后把JDK的安装路径作为值

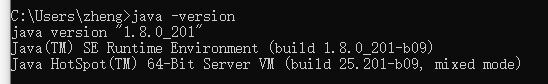


（2）找到path这个环境变量，添加一个内容

%JAVA\_HOME%\bin



第三、验证JDK的版本



2、Java语言编译工具

两个编译工具：eclipse和idea。

Eclipse下载地址：

https://www.eclipse.org/downloads/packages/

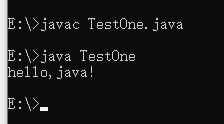
### 1.2.2 编译原理

Java语言的执行流程也是**编译+运行**流程。

Java的源代码文件 \*.java，其实也是一个文本文件；通过编译的过程生成一个字节码文件 \*.class；下一步就是通过Java虚拟机来执行class文件得到运行结果。

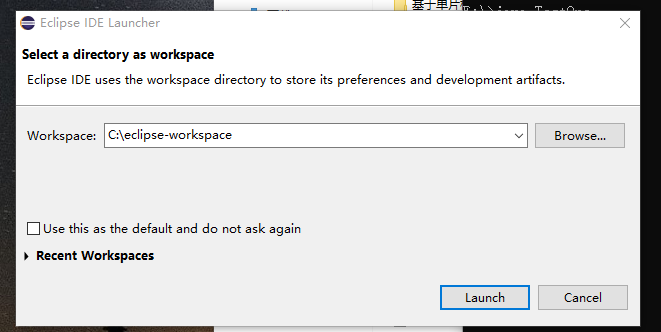
编译的命令是javac，执行的命令是java

如下:

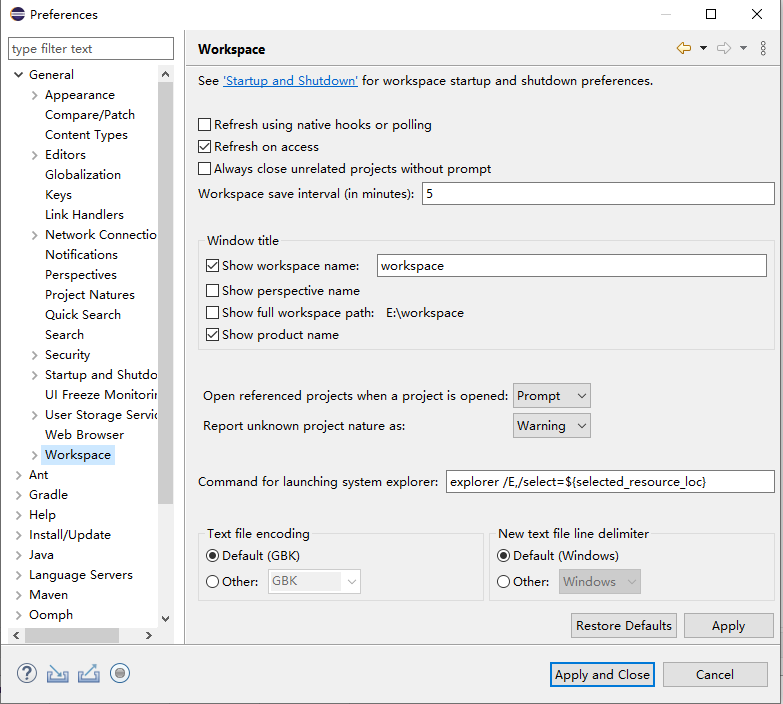


## 1.3 第一个Java程序

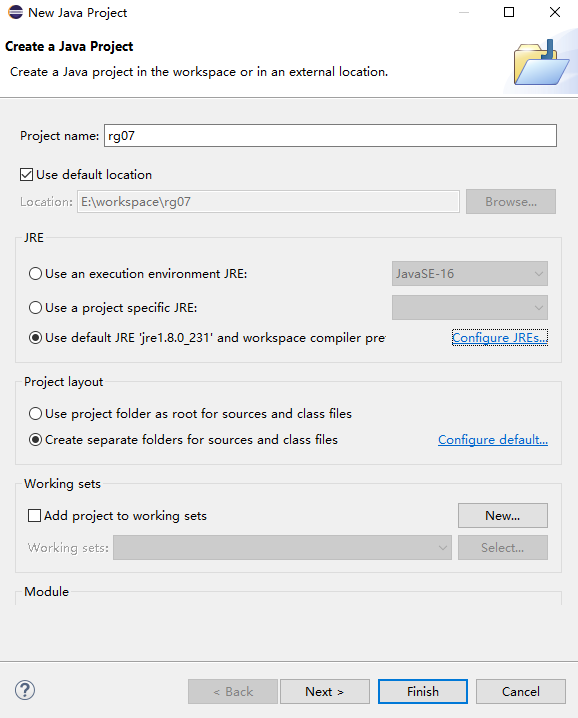
第一步、选择自己的工作空间【你的所有的开发源码以及项目放置的文件夹】



第二步、把工作空间的编码改成utf-8，因为eclipse默认设置编码是gbk。



第三步、新建一个项目



第四步、新建一个类【Java的所有代码都应该写在类里面】

**package** first;

**public** **class** FirstClass {

}

第五步、第一个Java程序

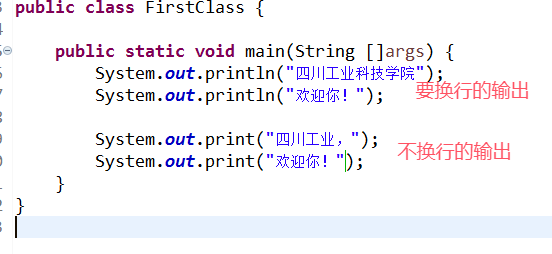
Java的程序执行入口都是main函数。



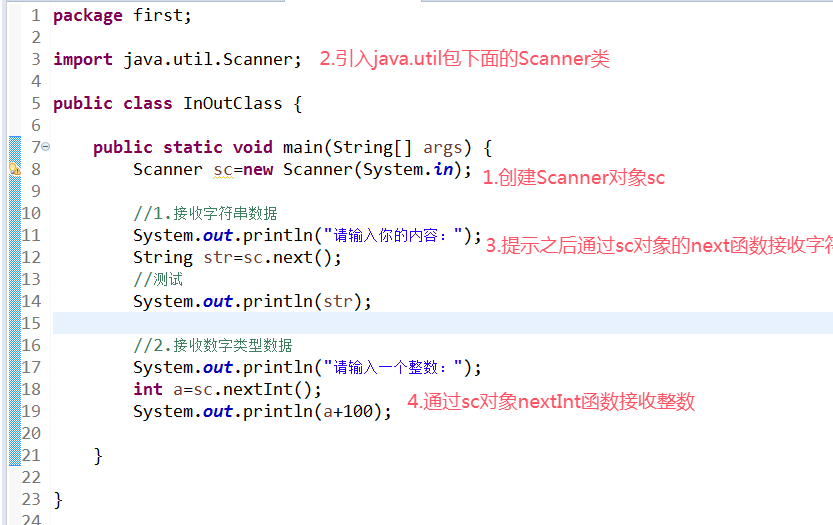
去看一下项目结构！

第六步、小案例

输出：



输入：



# 二、Java的变量常量以及数据类型

## 2.1 变量定义

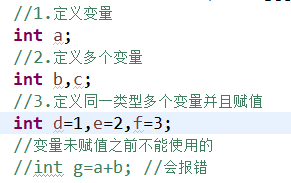
语法是这样的

类型 变量名;

提示：

可以在定义变量的时候赋值，也可以同时定义多个同一类型的变量。

例如：



注：变量的命名一定要注意以字母、数字和下划线组成，并且不能以数字开头。（语法规范）

经常使用英文并且采用驼峰命名法，特别强调不要使用中文拼音（开发规范）

例如：定义一个变量意思是学生名字stuName。

## 2.2 常量的定义

概念：

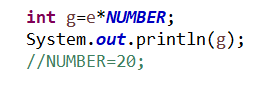
程序在编写过程中值不会被改变的量，和变量的区别是变量可以被改变。

使用场景：

比如圆周率；比如某个系统在运行过程中制定的某个系数；

语法：

public static final 类型 常量名=值;



注：常量名都以大写字母组成（开发规范）

## 2.3 基本数据类型

Bit Byte

位 字节 （一个字节=8位）

**8种基本数据：**

byte 字节 short int long 整型 (8 16 32 64) 单位是位

float double 浮点型 (32 64)

char 字符 8

boolean 布尔(true和false) 8

以byte为例，无符号数取值范围是

1 0-1

11 0-3

111 0-7

...

11111111 0-255 （256个数）

有符号数取值范围是 -128 到127 （256个数）

数据类型的自动转换遵守两个规则：

**整数类型可以自动转换为浮点型，反之不行；**

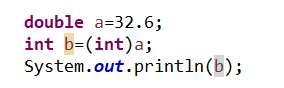
**位数小的类型可以转换为位数大的，反之不行。**

注意：Java中直接写小数，类型是double，是不能直接赋值给float类型的。

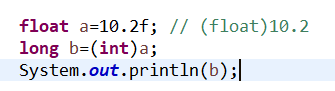
那么问题来了，如果我非要破坏以上的规则呢？强制转换的原则！

语法非常简单

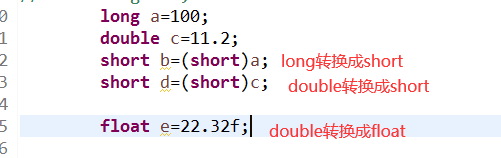
要转换为类型的变量=(类型)变量



结果为32，这样把浮点型强转为int会直接舍弃掉小数部分。



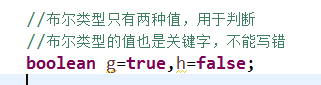
案例2：强制转换



案例3：字符类型



案例4：布尔类型



## 2.4 常用表达式

### 2.4.1 算术表达式

+

-

\*

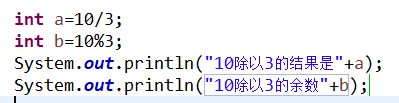
/

**如果两个运算的变量分别是浮点型和整型，一定会把整型变成浮点型再计算，得到的结果也一定是浮点型**。

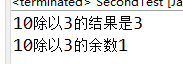
注意：

Java除法运算的时候，整数除以整数结果是整数，求**余数的运算符是 %**

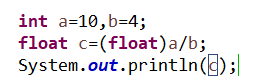
例子：



运行结果：



如果想除得的结果是小数,把其中某一个数强转为浮点型再计算！



结果是：



**重点：**

自增和自减

++和--

**++原来数值的基础上加1，**

**--原来数值的基础上减1，**

例如：

a=10;

a++;

a的结果就是11

变量++ 或者 ++变量

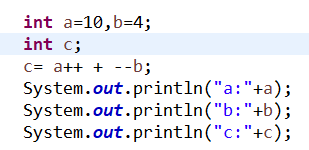
规则：

**++和-- 这个运算符号可以放在变量之前，也可以放在变量之后**

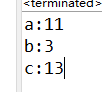
**符号在前，先自增（减）后运算**

**符号在后，先运算后自增（减）**

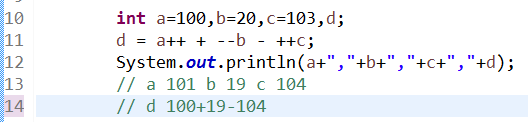
例子：



结果：



案例2：



注意：如果某个变量用了自增或者自减，那么他的数值一定会变，这个不用犹豫了，主要是用这个来运算的时候不好判断，用上面写的规则来判断。

### 2.4.2比较表达式

>

>=

<

<=

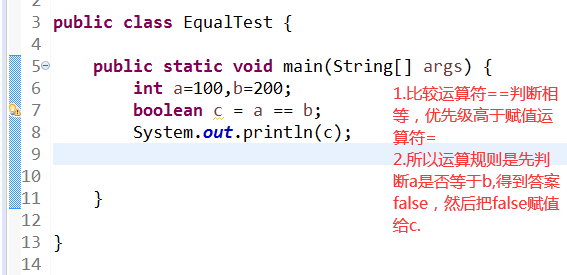
== 等于

!= 不等于

容易出错的地方是，把= 和== 混淆。=是赋值运算符，把某个值赋给变量或者常量；==比较运算符，判断两个变量或者值是否相等。

**比较运算符结果一定是boolean类型**

案例：



### 2.4.3 逻辑运算符

&& 与

|| 或

！ 非

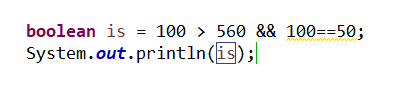
运算逻辑：

**&& 是 真真为真，见假为假**

**|| 是 假假为假，见真为真**

**! 是 取反**

例子：



**提示：逻辑运算符左右两边的值以及最后的结果都是Boolean类型**。

### 2.4.4 位运算符

自己去看位运算符和移位运算符

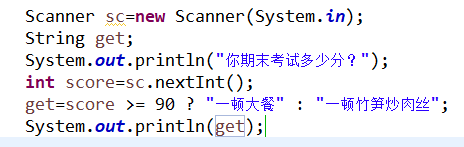
### 2.4.5 三元运算符

语法：

变量=逻辑或者判断表达式 ? 值1 : 值2;

前面的表达式返回为true，则值1赋值给变量，否则值2赋值给变量

案例：



### 2.4.6 赋值运算符

=就是赋值运算符

+=

\*=

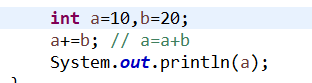
-=

/=

%=

这些都是赋值运算符

例如：



最终a=30

练习：

1、模拟买菜的，输入单价和购买的斤数，输出总价。要求输入和输出都用浮点型接收。

2、输入两个整数，计算他们的除得的结果和余数。

**3、输入两个整数，找出他们之前的素数。（只能被自己和1整除的数）**

# 三、Java的条件和循环结构

## 3.1 条件结构

条件结构有if 和 switch

例如：

系统登录的时候就有两个分支，一个是满足用户名和密码验证，登录成功；另一个是用户名和密码验证失败，登录失败。

### 3.1.1 if结构

语法：

（1）

if(判断的条件){

满足条件执行

}

只能判断一个条件

（2）

if(判断的条件){

满足条件执行

}else{

不满足条件执行

}

可以判断两个条件

（3）

if(判断的条件1){

满足条件1执行

}else if(判断的条件2){

满足条件2执行

}…else{

上述条件皆不满足

}

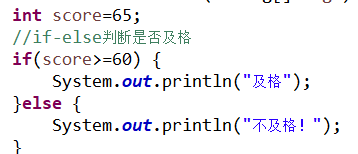
可以判断无数个条件

注：

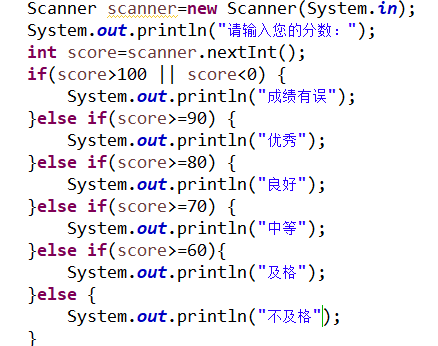
（1）理论上来说判断的条件要求是布尔型（如果是表达式的话，那么大多数情况应该比较表达式或者逻辑表达式）

（2）if-else if-else结构当满足某个条件的时候执行完毕，那么这个条件结构就结束；当进入后面的条件的时候，前面的条件就可以判断为不满足了。

案例1：

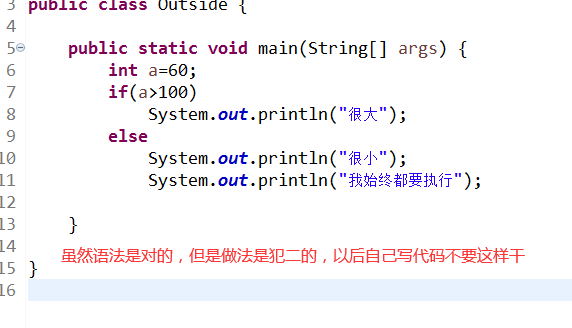


案例2：



关于if结构的简洁写法，在满足条件之后执行的代码块可以不用花括号，这个是没有语法错误的，但是记住一点就是如果不加花括号，这个条件结构只影响后面一句代码。

比如：



### 3.1.2 switch结构

语法：

switch(变量){

case 值1：

语句1;

**break;**

case 值2:

语句2;

**break;**

..

default:

语句n;

}

switch和if的区别：

switch只能判断变量等于某个值的情况，类似于选择题；但是if可以直接写表达式，所以if使用范围比switch广。

案例：



注意：

**break关键字一定要加；因为break表示结束这个结构或者跳出的意思；所以不加的话，当你进入某个条件的时候，执行完后面的语句，会把下面条件后的语句一起执行，直到结构完毕或者遇到下一个break。**

## 3.2 循环结构

while,do-while,for, for-each[集合章节讲]

### 3.2.1 while 和 do-while

**使用的时候一般用while**

语法：

**while(循环条件){**

**循环块语句;**

**}**

含义：判断循环条件是否为真，为真则执行循环快语句；执行完后继续判断，若条件还是为真，再次执行循环块语句；直到某一次执行完之后判断条件为假则结束循环。

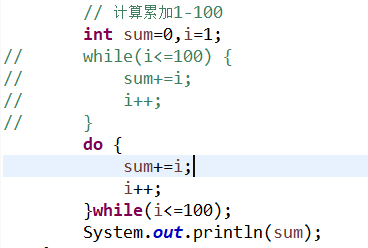
**do{**

**循环块语句；**

**}while(循环条件);**

**区别：执行结果只有在一种情况下有区别，就是循环条件一开始就不满足的情况。**

案例：



### 3.2.2 for

语法：

for(语句1;语句2;语句3){

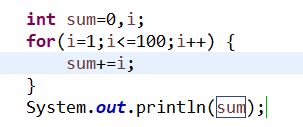
循环块语句;

}

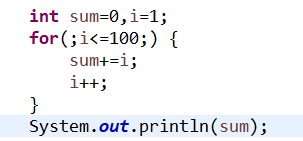
注：

**语句1是在for结构前执行，只执行一次；语句2就是类似于while的循环条件，每次循环块执行之后就去判断是否继续执行；语句3是循环块语句执行完之后，执行语句3，每次循环都要执行。**

例子：



也可以这样

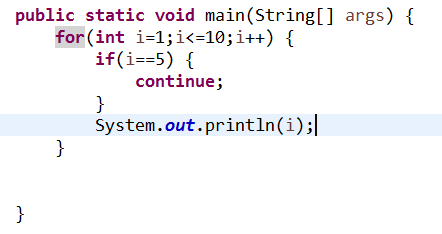


### 3.2.3 continue和break

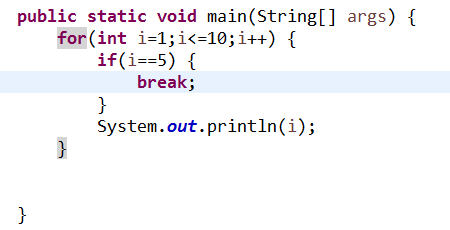
continue是退出**本次**循环

break是退出循环

案例：



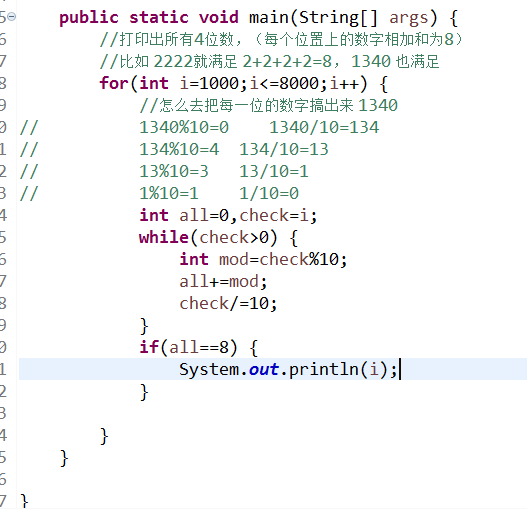
打印1-4,6-10,当i等于5的时候，结束这次循环，所以5不打印



只打印1-4，因为i等于5的时候，break关键会直接结束for循环

## 3.3 循环和条件结构嵌套及应用

案例：找出每位相加和为8的4位数



**练习：**

**1、输入行和列，打印出对应的\*号，比如我输入2行，3列，那么打印**

**\*\*\***

**\*\*\***

**2、输入两个整数，找出他们之间的素数。（只能被自己和1整除的数）**

**3、输入一个分数判断等级，要求当用户输入分数有误的情况重新输入，直到输入正确的分数为止**

**4、模拟用户输入密码的情况（3次机会），如果输入错误，提示还有几次机会，机会用完的话结束程序，提示卡被吞了；如果输入正确提示进入系统，结束循环了。**

**5、输出一个乘法口诀表**

# 四、Java的函数（方法）

## 4.1 自定义函数和调用

函数的作用：

把某一个在程序中会经常使用的功能代码放在一起封装，然后每次使用都进行调用。这样的话可以节省开发时间，提升代码的可读性以及后期的可维护性。

语法是：

**修饰符 static 返回类型 函数名(参数….){**

**函数内部代码**

**return 返回值;**

**}**

解释：

1、访问修饰符

四种修饰符

public 公共的 private私有的 protected受保护的 默认的 部分权限

**公共的整个项目所有其他类都可以使用**

**私有的只有本类的其他函数才可以使用**

**受保护的同一个包下的类或者子类可以使用**

**默认的也是同一个包下的类可以使用**

2、static 表示静态的

**有一个原则，就是静态的方法不能使用非静态的方法和公共变量**。所以在我们学习面向对象之前，所有的方法都要设置为静态的。

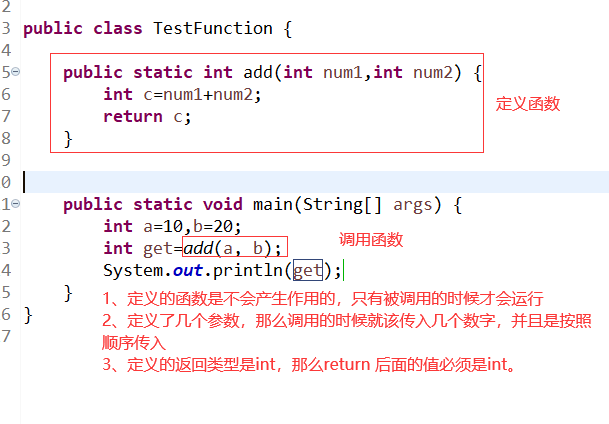
为什么静态不能使用非静态的内容。因为Java机制规定静态的内容是提前载入内存中的，内存中的内容无法直接使用没有进入内存的数据。

3、返回类型，Java规定如果函数有返回值的话，必须在方法中去设置返回的类型，这个类型要求和返回值一致，**返回值和返回类型是同时存在**。如果没有返回值则返回类型这一块使用void。

4、函数名没什么说的，遵守Java变量命名的规则和规范就可以了。

5、参数，是函数在执行的时候需要传入的数据，那么我在定义函数的时候给他规定好。参数包括 参数类型 参数名称，一个函数可以定义0到N个参数。然后你定义好的参数，执行的时候必须按照这个规则传入实际的数据，否则会报错。

## 4.2 案例



## 4.3 作业：要求改造一下

改造3.3 的练习

比如练习1：

**1、输入行和列，打印出对应的\*号，比如我输入2行，3列，那么打印**

把行和列左右参数传入函数，这个函数实现打印，在main里面调用一下。

**2、输入两个整数，找出他们之间的素数。（只能被自己和1整除的数）**

两个整数作为参数，找到素数之后放入数组或者集合，最后返回数组或者集合。

## 4.4 综合练习

模拟银行ATM取款的流程，完成功能如下：

1、验证密码，3次失败锁定，如果成功进入系统

2、系统功能为 查询、取款、退出

3、查询和取款分别可以选择两个币种（人民币、美元）进行操作，建议各有10000元

4、查询和取款完毕，用户可以选择继续操作还是返回上一级

5、在主菜单中，用户选择退出则谢谢使用，结束程序

# 五、面向对象

面向对象的编程思想，本质上就是和原有的面向过程编程产生了分歧。就是这个思想更加符合人类的管理思维，所以大多数高级管理软件或者系统都是采用面向对象的语言进行设计开发，那种底层与硬件设备打交道的程序还是采用C语言进行开发。

就是在设计或者开发一个程序或系统的时候，把其中的某些事物抽象成一个个的类，然后通过创建类的对象来进行管理。

实际开发过程：确定用户的需求、设计物理模型（数据表结构）、对应设计相应的实体类、分系统功能模块、按照模块进行功能开发、系统的分步测试、整个测试、测试版上线（用户测试环节）。

## 5.1 类和对象

### 5.1.1 概念

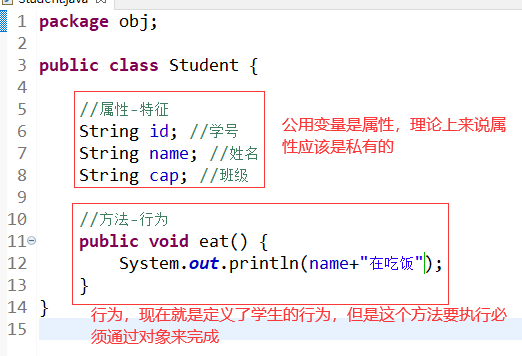
类是一系列具有相同特征和行为的事物的统称，是一个抽象的概念，不指向某一个具体的事物。

对象是类的实例化，可以从类中具像出某一个对象，是一个实实在在的事物。

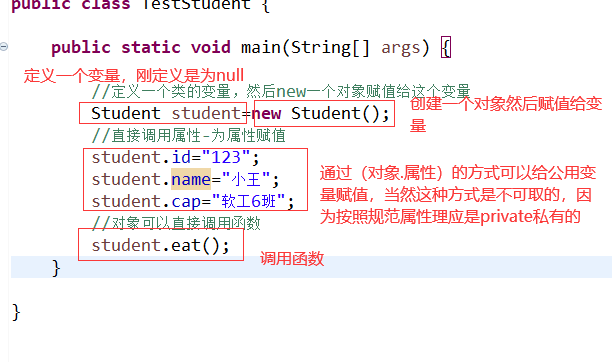
注意：类本身不具备执行能力，但是开发者可以实例化某个类得到对象，由对象来完成某个行为。

### 5.1.2 案例

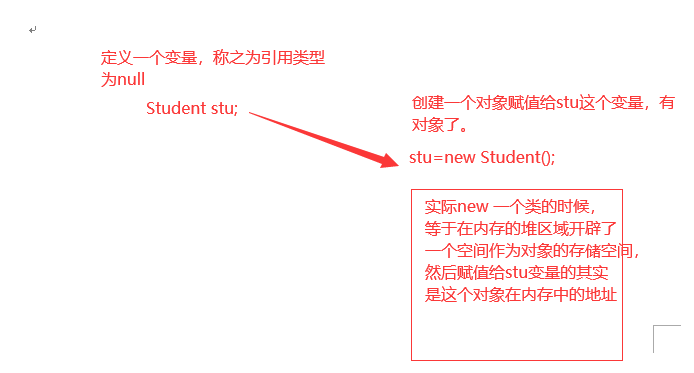
定义一个类：



实例化对象



注：

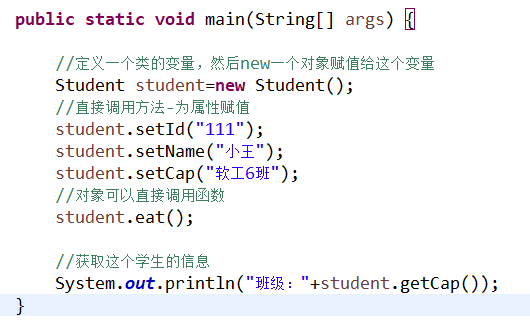


### 5.1.3 关于实体类的写法

类的set和get函数



对象使用



### 5.1.4 构造函数

构造函数就是创建对象的时候调用的函数，就叫做构造函数。每个类定义的时候系统都默认给一个不带任何参数的构造函数。但是如果开发者自定义了构造函数，那么原来提供的构造函数将会被取消。

构造函数的语法是

修饰符 类名(参数){

内容

}

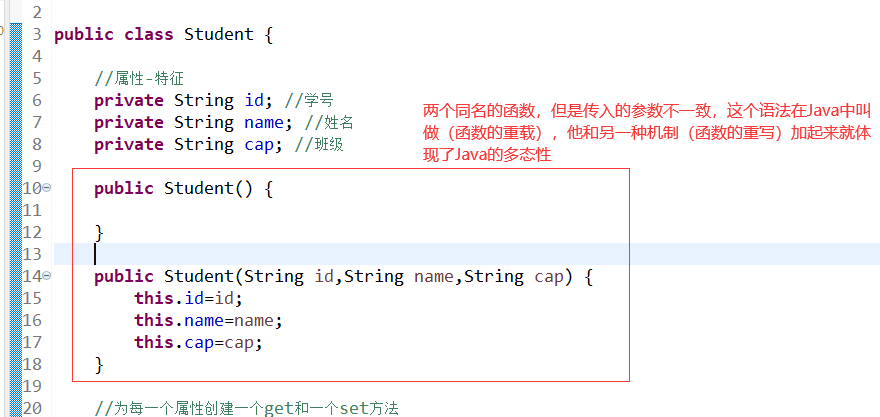
构造函数是没有返回类型（或者void）的，函数名称必须是类名（大小写都不能变）。

构造函数的作用是：

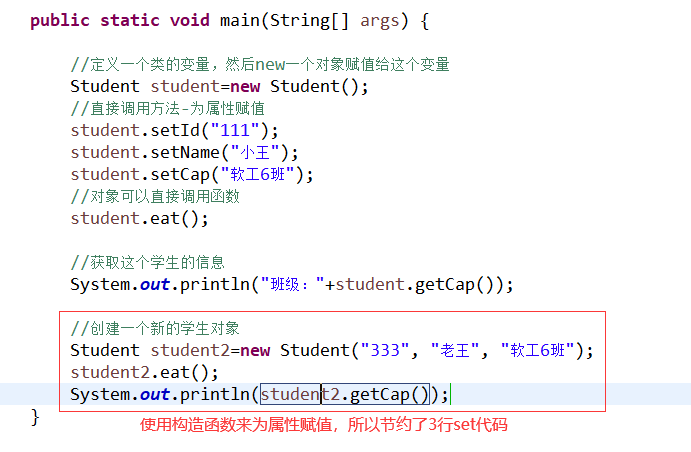
创建对象的时候就执行，**所以很多开发者喜欢把一些前期需要做的工作放在构造函数中来完成**。

案例：

一个类有多个属性，创建对象之后每个属性都用set函数赋值比较麻烦，那么我们可以使用构造函数在创建对象的同时给属性赋值。如下：



实例化



### 5.1.5 练习

定义玩家类（姓名、血量、攻击力），创建两个玩家互相殴打，直到某一方被干掉为止，这个属性值由控制台输入。

随机值 Random

# 六、继承和接口

### 6.1 继承

继承是一种Java语言的特性。设计继承这个特性的目的提高开发效率，增强代码的复用性。

例如：

四川大学做一个学生信息管理系统，学生使用功能如下

本科生：签到、考试、成绩、校外实习、体育测试

研究生：签到、考试、成绩、项目实践、论文发表

有重复的内容，写代码的时候就会重复。继承的机制

学生：签到、考试、成绩

本科生：校外实习、体育测试

研究生：项目实践、论文发表

本科生和研究生继承学生。

### 6.2 父类和子类

关系就是和现实生活一模一样

子类可以使用父类的所有属性和方法（私有的除外）

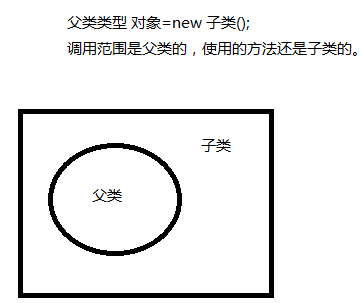
**继承的语法：**



（1）子类的对象可以使用父类的方法，同时子类本身也能够调用父类的方法。【非私有】

（2）子类如果不认同父类的方法，也可以重新改变这个执行内容。如果子类定义了同样的方法，那么调用就用子类的。方法的重写！

（3）子类的对象可以赋值给父类的类型，但是遇到这种情况，这个对象只能使用父类的方法，但是他调用的方法还是子类的方法。

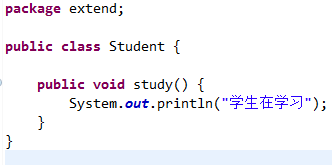


（4）一个父类可以有很多子类，但是一个子类只能继承一个父类。【单继承】单继承限制很多东西，可以多**接口**解决问题。

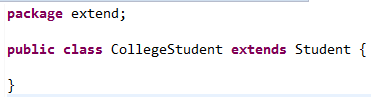
**重修内容：**

**（1）继承**

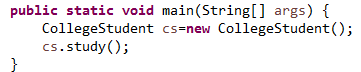
**比如有一个学生类**

****

**有一个大学生类**

****

**测试：**

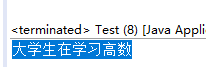
****

**子类继承了父类，就可以拥有父类的所有属性（公用变量）和行为（方法）【除了私有的】。**

**（2）方法的重写**

****

**如果子类也有一个同样的方法，那么后面调用的时候调用子类的方法，不会调用父类的。这种叫方法的重写。效果：**

****

### 6.3 抽象类

（1）不能够实例化，要使用抽象类的方法只能子类来继承，创建子类的对象来调用方法。

（2）抽象类的语法，在类的前面加abstract，抽象类和普通类的规范没有任何区别，只是多了一个规定，就是可以定义抽象方法。

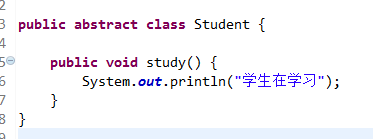
（3）什么是抽象方法，就是没有实现的方法。交给下面的继承这个类的子类来实现。父类有抽象方法，继承之后就一定要实现。

为什么要有抽象类，就因为上层结构在考虑问题的时候不会考虑那么细致，所以规定了一个大致的方向。必须由子类的具体实现。

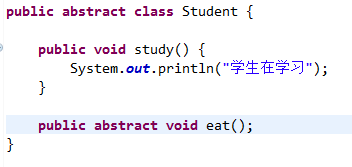
**重修内容：**

（1）抽象类

具体表现就是可以定义抽象的方法。定义抽象类就是在class前加abstract关键字。



**抽象方法就是没有完成的方法。定义抽象方法也是在方法定义的里面添加abstract关键字。**



比如eat就是抽象方法，就没有完成。但是谁要继承这个抽象类，就必须实现（重写）这个类的抽象方法。

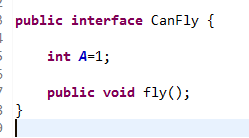


**（2）接口：**

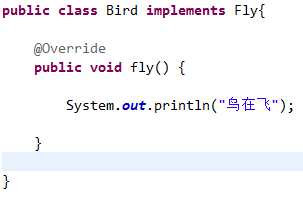
**定义一个接口，关键字是 interface**



**接口里面只能有常量和抽象方法。**

****

**一个实现类去实现一个接口，就必须重写接口的抽象方法**

****

**（3）抽象类和接口的区别**

**第一点、不同点抽象类可以定义变量和非抽象的方法，也可以定义常量和抽象方法，但是接口只能定义抽象方法和常量。**

**第二点、不同点因为抽象类是继承，所以子类只能继承一个父类；但是接口不一样，一个类可以去实现多个接口。**

**第三点、相同点是接口和抽象类都不能实例化，也就是创建对象，都只能够通过实现类来重写抽象方法之后，创建实现类的对象。**

**第四点、相同点是都可以定义抽象方法。**

### 6.4 接口是什么

接口是Java程序规定的一种语法结构，可以把他看成**一种规范规则**。有人来制定规范，有人参照执行。

比如：USB接口

让使用者看到的就是插槽和插口。几乎所有的电子设备和电脑厂商都生产USB接口的产品，他们是互相约定了？

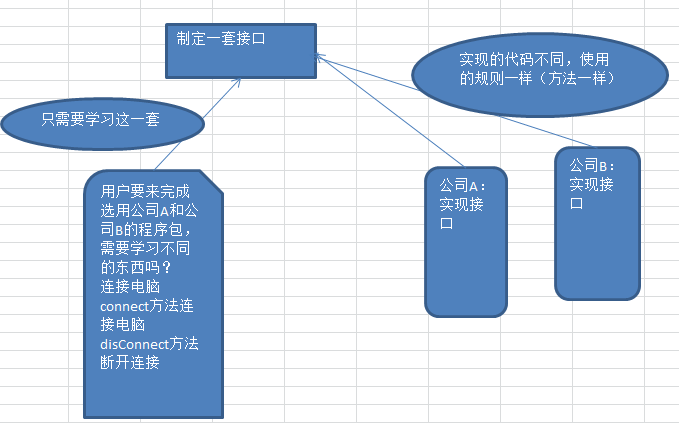
USB接口的规范和标准是由一个国家标准规范组织制定的，然后由厂商按照这个标准来生产。制定规则的组织不负责生产，但是下面的厂商不敢不按照这个规则做。

### 6.5 接口有什么用

制定规定，大家都来遵守这个规定，在很多的领域进行应用。

接口使用场景1

一般都是大公司推出一个标准，定义好了。如果其他的公司要想参与进来，就必须要按照这个标准来开发程序，其他的用户使用都可以学习这个标准就可以了。



使用场景2

公司开发一个项目，下面的程序员都是新手，写的代码和方法可能很不规范。

项目的技术负责人可以定接口，方法名、传入参数和返回值都给你定好，并且这个方法要实现什么功能也告诉开发者。开发者实现接口之后，就不用去管具体我要做什么工作，只需要完成一个个接口定义好的方法就行了。

所以得出了一个结论：接口都是很厉害的人用的，初学者或者独立开发者没什么机会用上。可能会遇到接口，自己写接口的机会不多。所以还是要学习并且掌握他的语法。

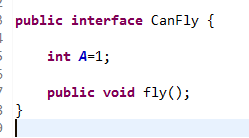
### 6.6 接口怎么用

1、定义一个接口



定义的关键字变成了interface

2、接口里面只能有常量和抽象方法

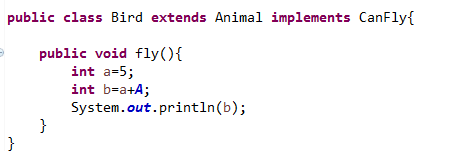


常量必须加final，但是接口里面只能有常量，所以final省略了也行；

抽象方法同理也是省略了abstract

3、实现一个接口

Implements关键字，并且一个类可以实现多个接口，接口之间用,隔开就可以了。



实现接口之后必须重写它的抽象方法，也可以使用接口定义的常量，但是不能修改值。

4、接口不能实例化，但是可以通过他的实现类来调用方法



# 七、常用类

## 7.1 String，StringBuffer类

案例1：

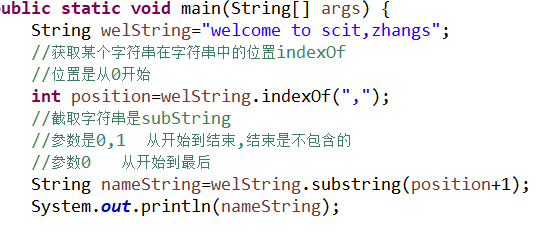
有一个欢迎词“welcome to scit,xxx”！xxx是人名，但是我不知道是谁，要求不管这个字符串传过来是什么，得到xxx的内容。

字符串的截取

用到的方法包括

indexOf

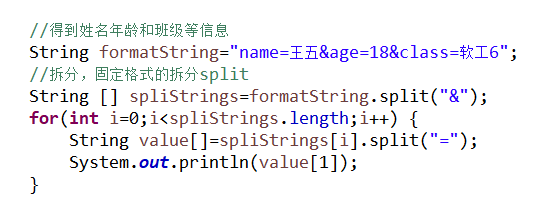
substring



案例2

有一个固定格式的字符串表示个人信息，比如“name=xxx&age=xxx&class=软工6”。

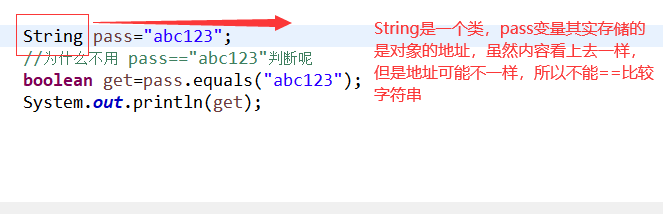
通过Split函数拆解



案例3

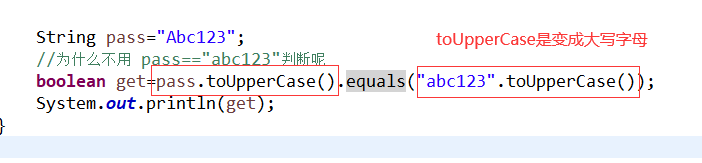
\*对比密码是否正确

Equals函数，通过字符串去比较另一个字符串，如果内容一致则返回true,否则false。



\*怎么样比较密码不区分大小写？

把比较的双方全部大写，然后再比较

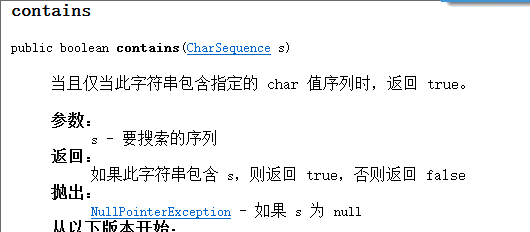


案例4：替换和查询

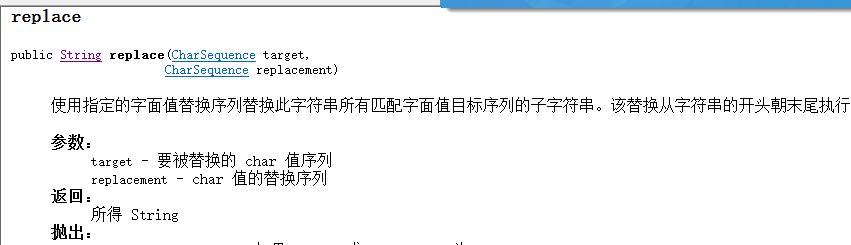
聊天过滤的需求，举例：你妈妈的病好点了吗？ 出现的可能是“ \*\*妈的病好点了吗？”

核心函数就是

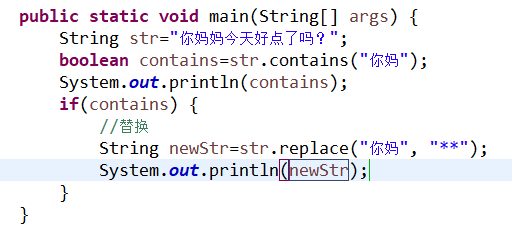
contains



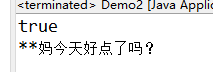
replace



例子代码：

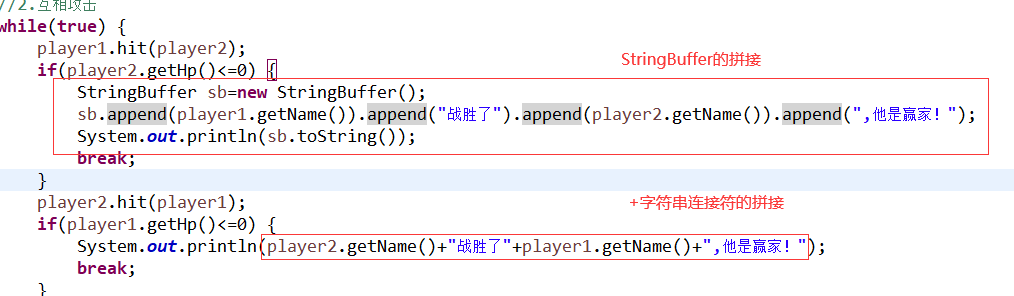


结果是：



案例5：字符串拼接

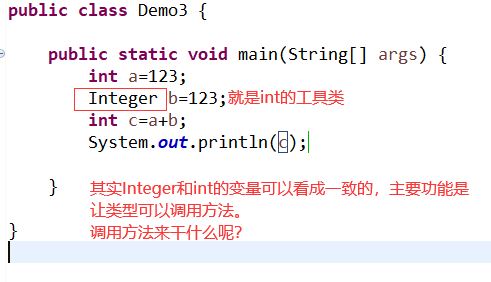
使用StringBuffer类可以实现字符串的拼接！效率是要优于字符串连接符的



## 7.2 基本数据类型的工具类

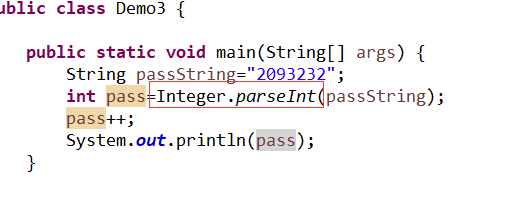
Java的所有的基本数据类型都有对应的工具类，因为基本数据类型不是引用类型，无法使用函数。所以给他对应一个工具类，这个类型的变量可以等同于一个对象，是可以调用函数的。

举例1：



举例2：

最常用的方式就是把字符串转换为相应的类型。



## 7.3 时间日期

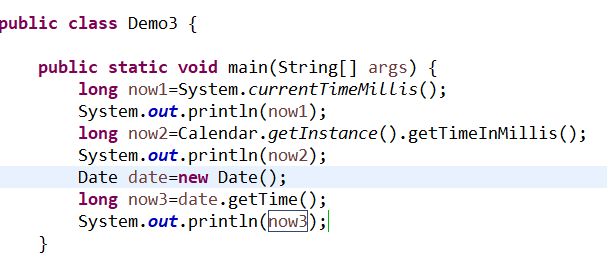
### 7.3.1 时间戳

时间戳的概念：

指从1970年1月1日零时到你指定的时间的间隔**毫秒数**，13位的整数。因为时间戳比较容易计算中间的间隔时间，所以很多数据库表设计，都是把时间字段存储使用时间戳的格式。

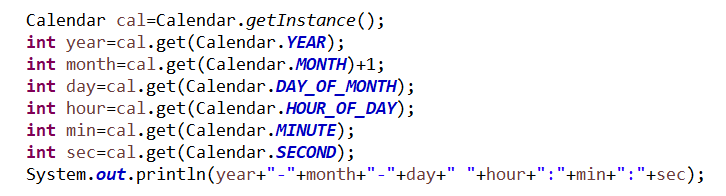
时间戳的获取

以下是时间戳获取的三种方式



### 7.3.2 获取时间日期

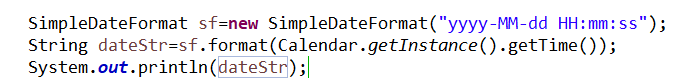
代码案例1：



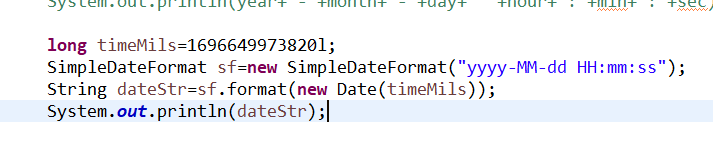
结果：



代码案例2：时间日期格式化类



代码案例3：把时间戳转化为格式化的类



# 八、数组

## 8.1 数组的功能

主要是因为有些同类型的数据比较多，不可能每个数据都设置一个变量去存储。比如说同学们的成绩，那么一个班少说30个人，需要30个变量；更夸张来说那一个学校的成绩，需要至少1万个变量；这个根本不现实。所以在此需求下，数组就产生了。

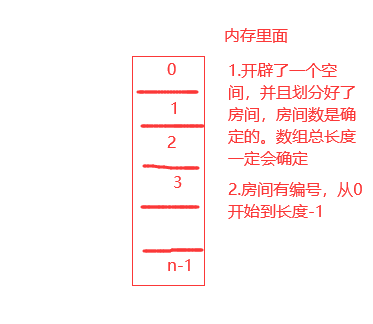
数组的根本作用，数组其实就是开发者定义的一个容器，去装数据的。其实本质就是在内存里面开辟一个空间【这个空间可以分成多个小空间，每个小空间是挨着的】，但是开始如果没有赋值，那么空间就是空着的，需要给值就把相应的值放入这个空间。内存机制就是这样的。

## 8.2 数组的定义和赋值

语法：

1、数据类型 [] 数组名=new 数据类型[整数];

注：整数是数组的长度，意思就是这个容器最多能装多少个数据。



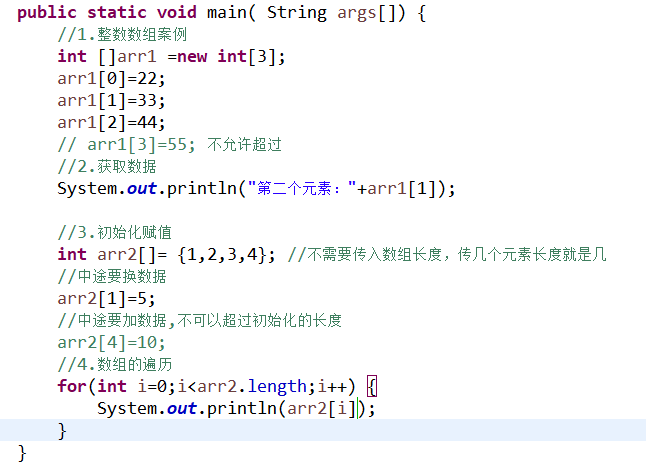
2、给数组添加数据

数组名[编号]=变量;

获取数据的方式也是如此： 数组名[编号]

3、可以在定义数组的同时给数组里面填入数据

例子1：



例子2：数组的冒泡排序法

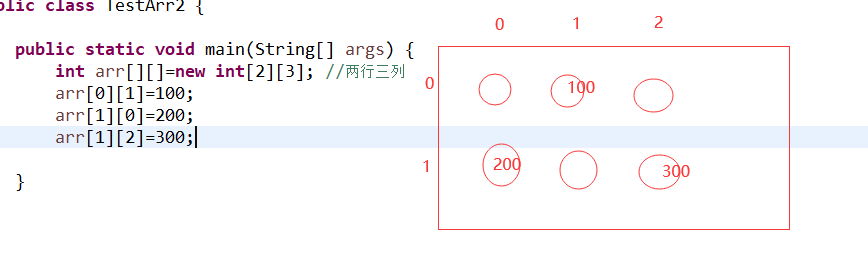
P77, 例4-2

## 8.3 二维数组

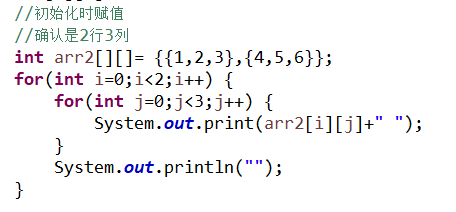
语法

类型 数组名[][]=new 类型[行][列];

举例1：



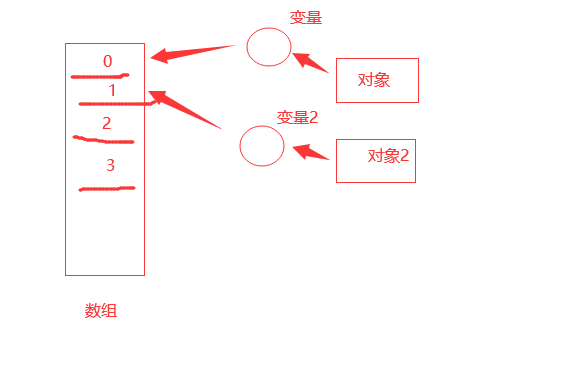
举例2：



书中的案例P80

## 8.4 对象数组

数组里面放的内容是对象【引用数据类型】

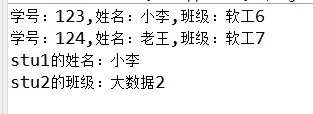


对象数组存储的是对象的内存地址，核心！

案例：



结果：



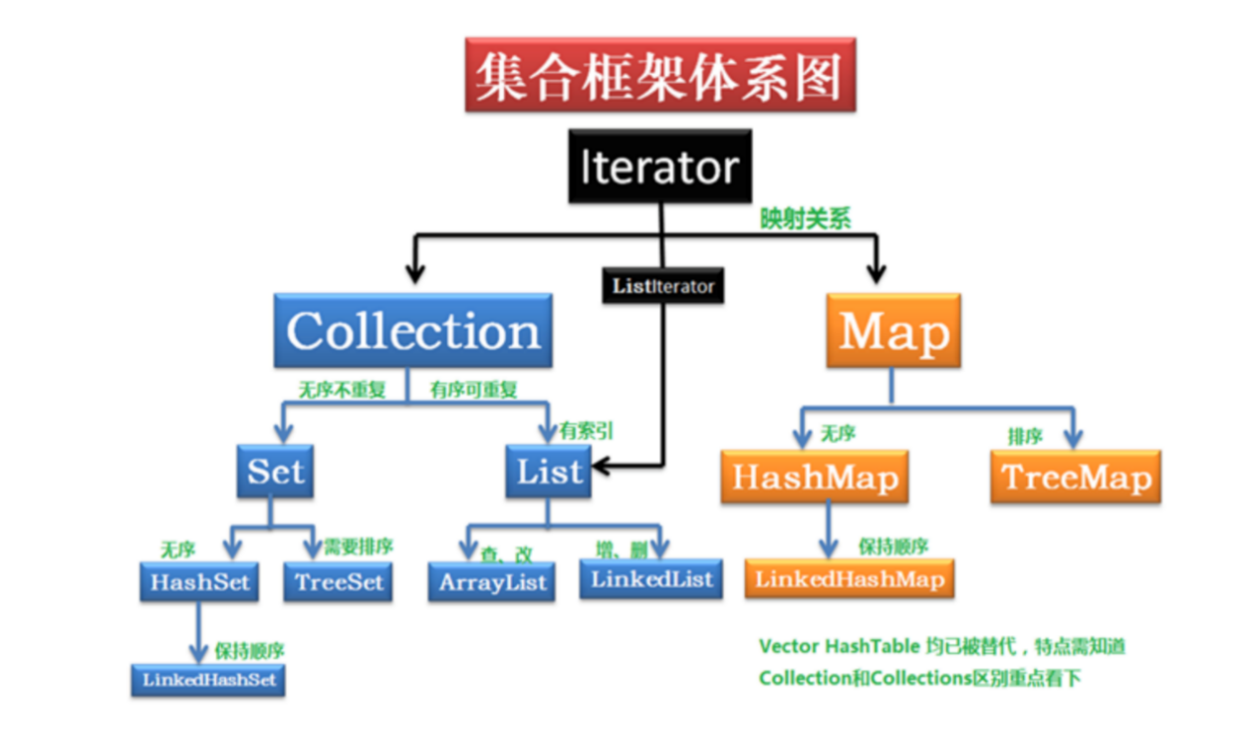
# 九、集合

## 9.1集合框架概述

（1）为什么要使用集合

关于数组的问题就是必须规定容器的最大容量，那么集合可以解决这个问题。集合的优势在于长度是可以伸缩的。

（2）集合的框图



## 9.2 列表集合的使用

\*集合有一个特点，就是集合能够装所有类型的数据。做到这一功能是因为集合有一个闭包和拆包的机制。每一个装入集合的数据，都会打包成Object类型【是所有类型的父类】装入，那么如果需要从集合中取出数据，得到的也是Object类型，可以进行**强制转换**得到你要的数据类型【不能转错了】。

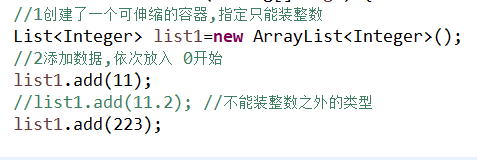
### 9.2.1 集合的创建和元素操作

举例：

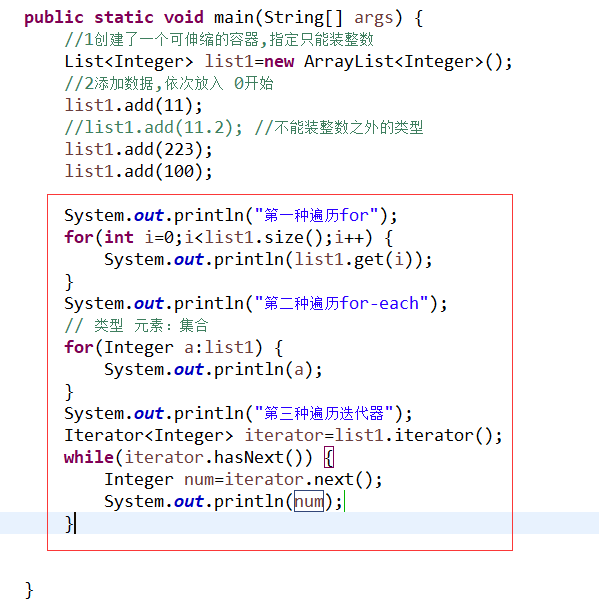


### 9.2.2 集合的泛型

因为Java规定集合可以装任何类型的数据，所以有一个闭包和解包的流程，代码相对繁琐并且容易出错。所以又有一个新的规定，就是如果开发者确定这个集合只装同一类型，那么可以指定类型，这个我们称之为泛型。



### 9.2.3 集合的遍历

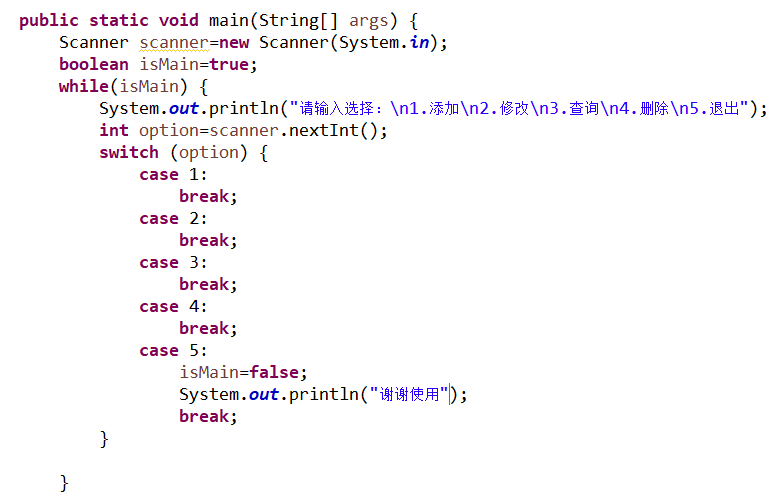


### 9.2.4 练习

要求完成一个学生信息管理案例，在控制台里面，根据用户数据输入选项去做学生信息的添加、修改、删除和查询，只要用户不选择退出的功能，那么就一直操作。学生的属性：学号、姓名、班级。

程序控制流程代码参考

如下：



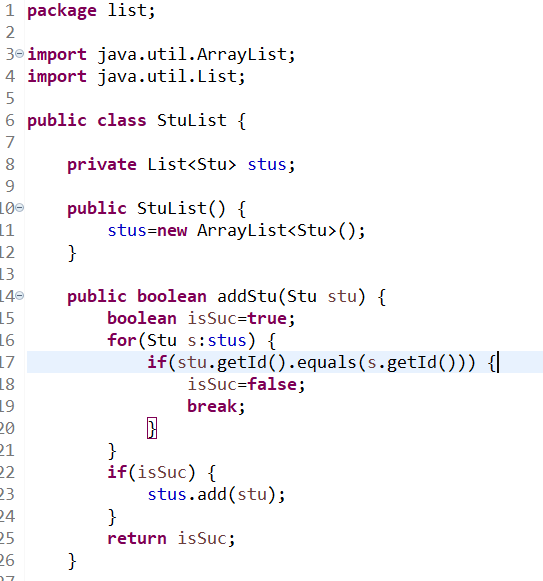
提示：再写一个类来实现底层的数据存储

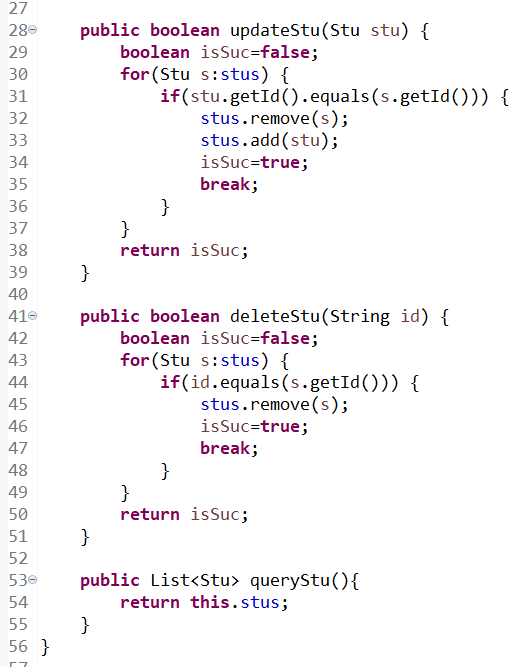
案例实现：

（1）实体类

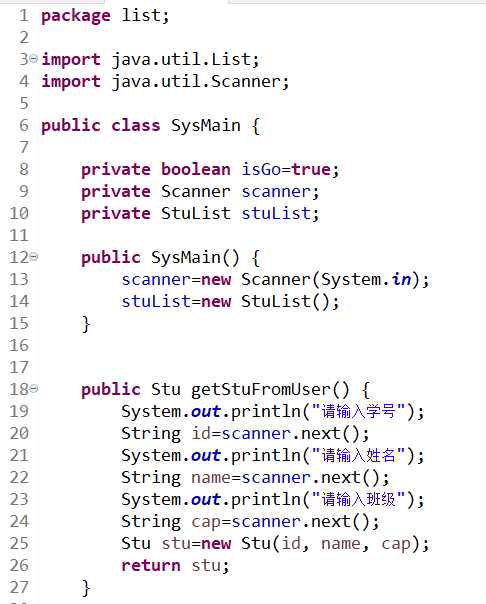


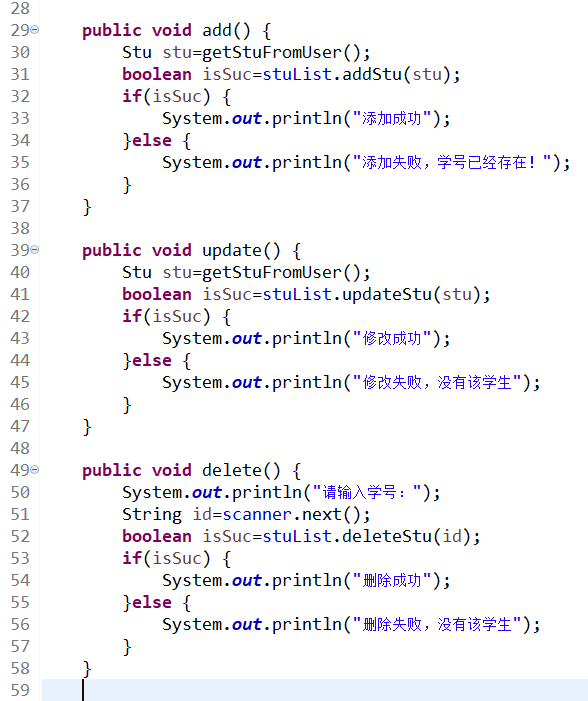
（2）底层业务类

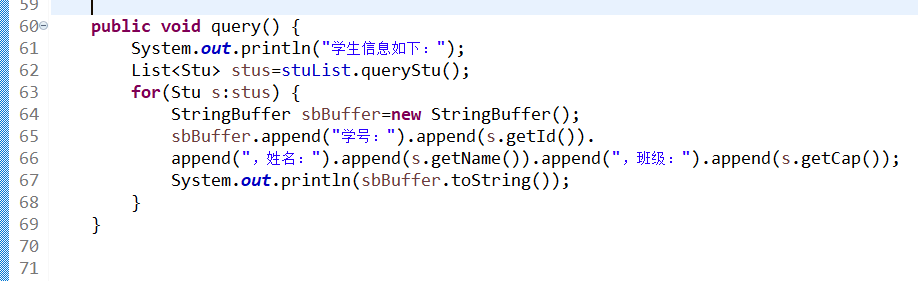




（3）界面显示类



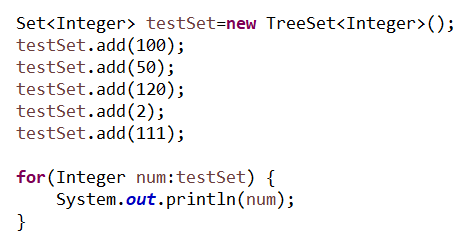




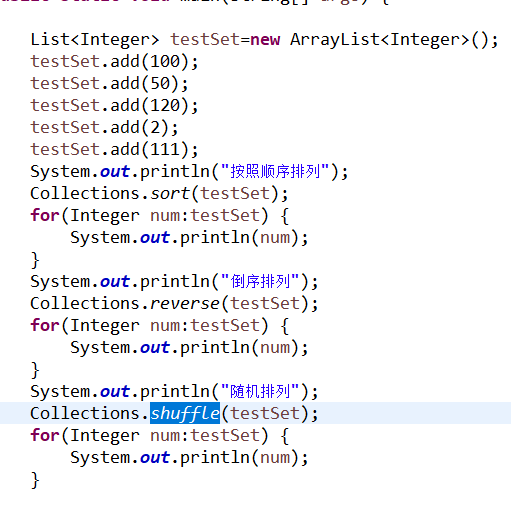
## 9.3 Set应用和Collections业务

Set和List的区别就是没有索引，有一些比较简单的算法逻辑可以在Set中实现。

案例1：



Collections不是集合的父接口，是一个集合的工具类，可以实现简单的业务逻辑的算法。



## 9.3 Map集合

（1）Map集合特点

就是一个键值对的存储形式key-value，好处就是方便检索。

特点：key是唯一的，如果重复存储，值会被覆盖。

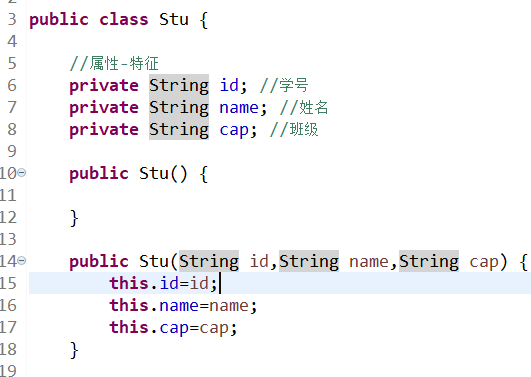
（2）使用的方式



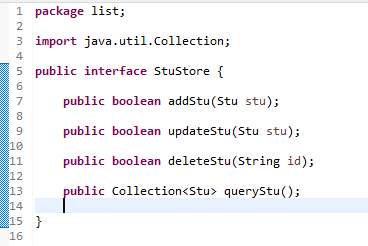
主要是前面的程序，可以使用Map来替换存储容器，可以节约开发时间。

9.2.4 练习代码

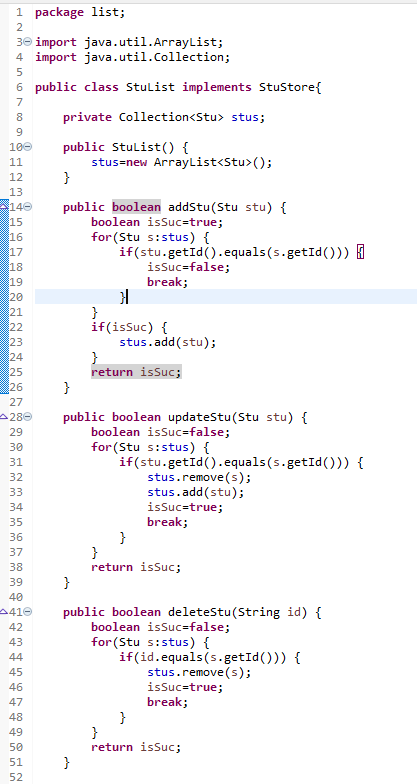
（1）实体类

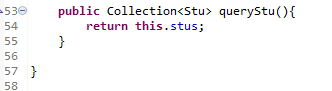


（2）存储内容接口

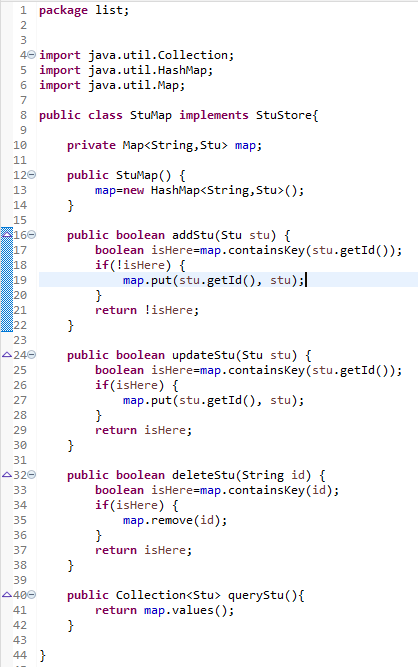


（3）使用Collection作为存储容器





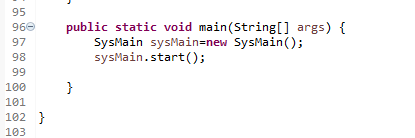
（4）使用Map作为存储容器



（5）界面类







# 十、异常

## 10.1 异常概念

Java针对程序在编译或者运行的环节中会出现或者可能出现的错误，有一种错误的处理机制，开发者称之为异常的处理。

实际上Java的错误分成了两类，错误和异常。出现了错误就会直接Java虚拟机就停止了，开发者是没有办法处理的；但是**异常是可以预见和处理的**。我们在开发的时候应该避免出现错误，因为错误会直接导致程序崩溃；但是可以处理异常，如果出现我们可以捕捉并且处理。

## 10.2 异常处理的方式

两种方式：第一种是try-catch结构，可以捕捉异常并且处理；第二种是在某个方法扔出异常，**谁来调用方法谁处理异常**。

\*特别一点就是，开发者可以在某段代码中直接扔出异常，交给别人去处理。

（1）捕获异常

try{

可能出现异常的代码

}catch(出现的异常类型 e){

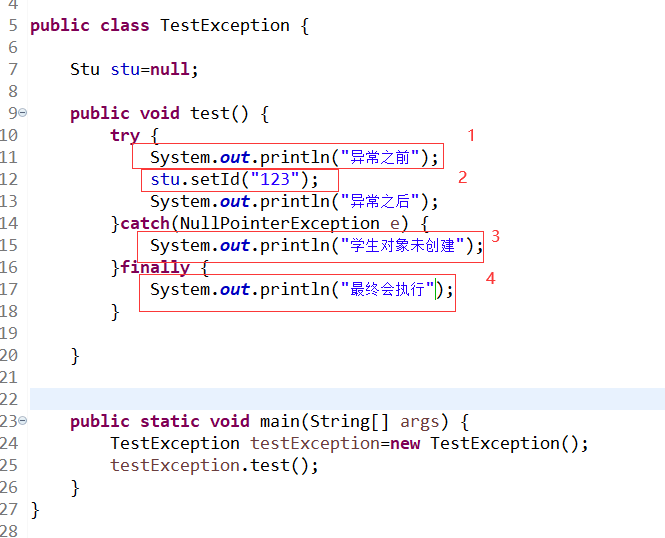
出现异常可以执行的代码

}finally{

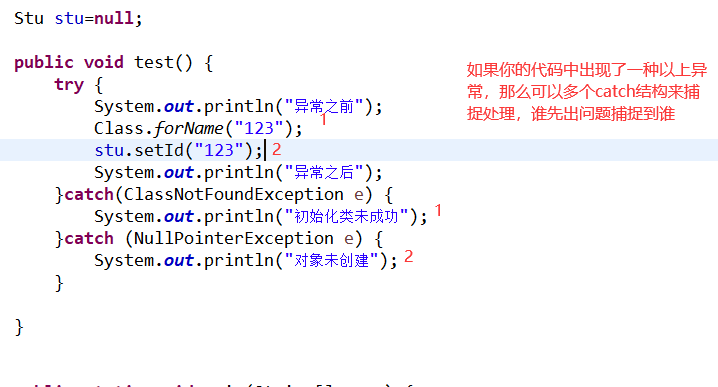
都会执行的代码

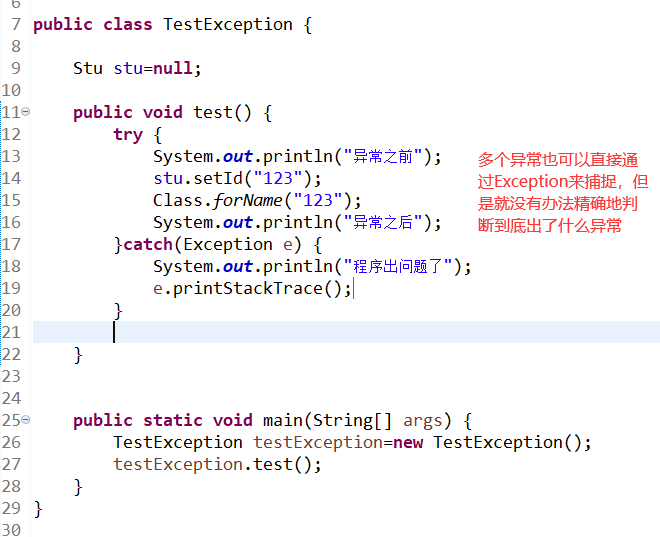
}

案例：



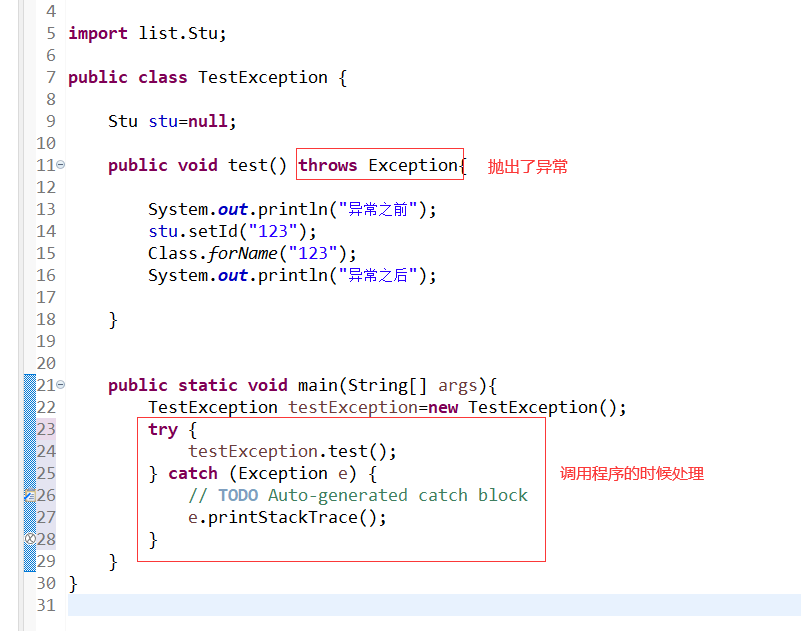
案例2：





（2）抛出异常

案例



（3）人为的抛出异常



# 十一、Java的IO流

## 11.1 流的分类

流是Java中数据传输的唯一方式，流的传输最基础的数据格式就是字节Byte，所有的流底层都是基于字节来传输的。

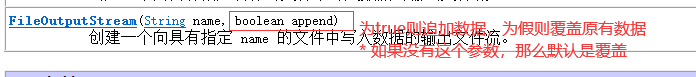
按照传输数据格式分：字节流、字符流和对象流

按照输入输出方式来分：输入流和输出流

\*输入和输出对应的对象都是内存，输入流是把其他的地方数据输入到内存，输出流是把内存中的数据输出到其他地方。

## 11.2 字节流

输出流

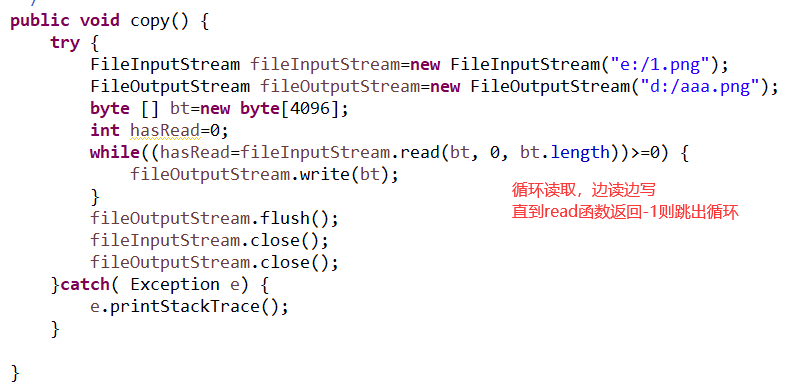


案例：

输入流，从文件中读取数据放入内存，可以用变量接收到，然后输出来看。



遇到大文件的时候，采用循环的方式来完成文件读写，案例！



## 11.3 字符流

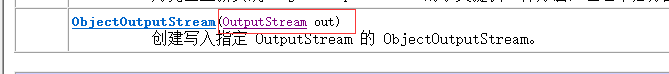
底层封装的还是字节的形式，如果是文本格式的数据，使用字符流会更加的方便操作，唯一的缺点就是字符流的初始化太麻烦了。

案例如下：

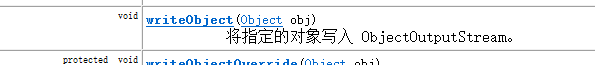


## 11.4 对象流

可以把内存的数据（对象）写入到文件，或者把文件中的数据读取到内存，赋值给对象。



输出方法，可以把具体的对象写入到文件



注意：输出的对象一定要实现一个序列化的接口

**练习：可以实现前面的集合的案例，数据的持久化，下节课要交！**

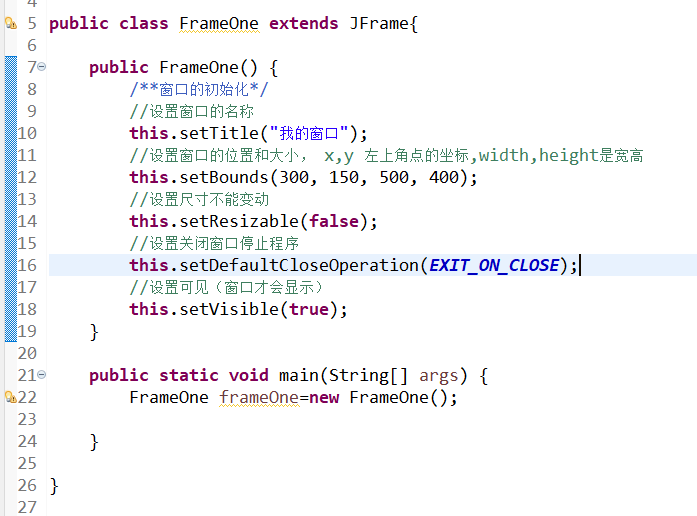
# 十二、GUI

Java的图形用户接口，最开始使用的awt包下的控件和类，但是后来在推广的过程中发现这个awt包有几个缺点（对系统底层的库依赖很强、重量级）；大部分开发者换成了swing包下的控件，这个包下的控件纯Java编写，对系统底层的库依赖不强的。

功能其实就是实现Java的图形用户界面。

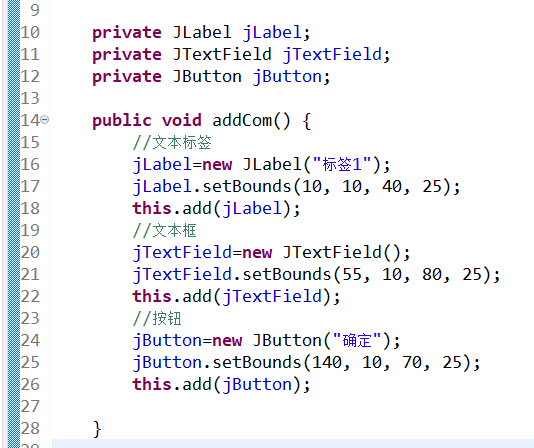
## 12.1 界面

通过一个类去继承JFrame，那么这个类本身就变成了一个可视化的界面，并且可以使用父类的所有非私有方法，可以调用这些方法来设置界面的各种属性，并且设置界面的显示。



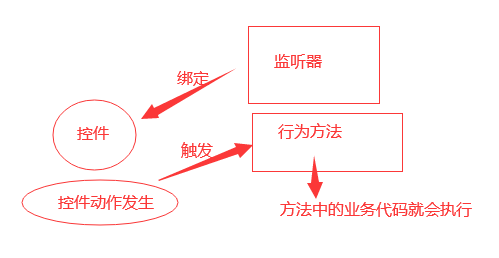
## 12.2 控件

使用绝对坐标的方式来插入控件。需要注意的是addCom函数一定要在setVisiable函数之前调用，因为添加完控件再显示，控件才能显示出来。



## 12.3 事件监听

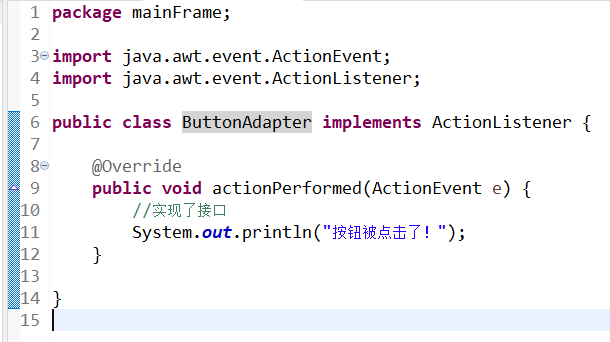
所谓的事件监听，就是监测界面上用户针对控件的某一个行为，然后实现相应的业务功能的要求。原理：



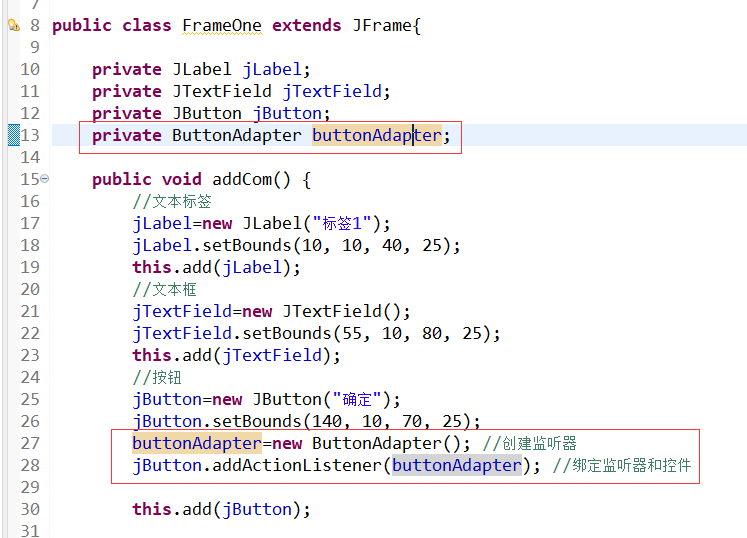
实现监听有三种方式

第一种：通过一个类去实现接口（因为监听器全是接口）

（1）在外面新建监听器



（2）把控件和监听器绑定



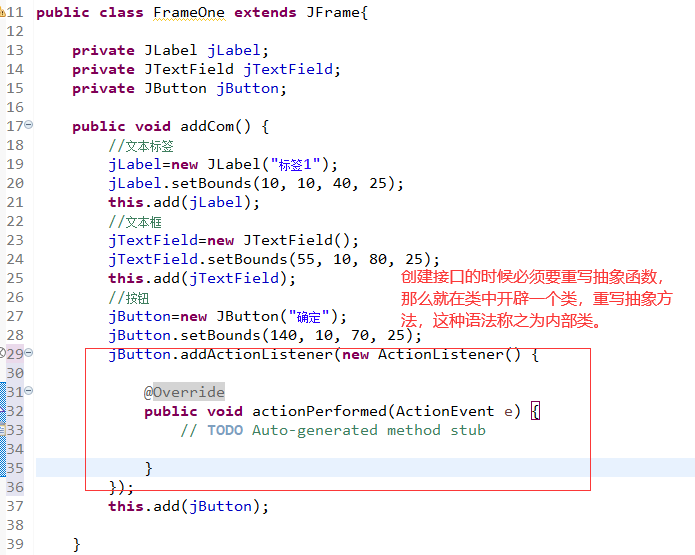
缺点是监听器和界面不在一个类，数据的传递很麻烦。

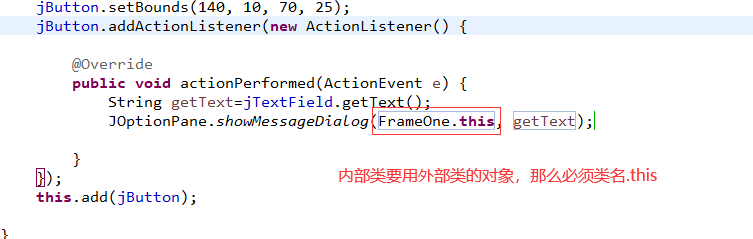
第二种方式、把这个界面本身设置为监听器

好处是界面这些控件和事件监听方法都在同一个类下面，完成功能比较方便。但是缺点是类设计方案功能界限模糊，代码结构相对混乱。



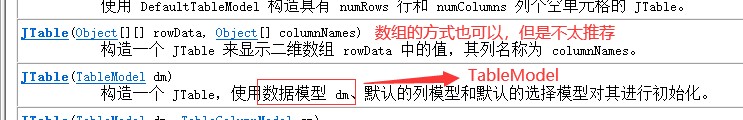
第三种方式：采用内部类的方式





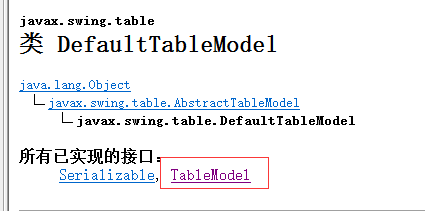
## 12.4 JTable表格控件

（1）在构造的时候传入Jtable需要显示的数据

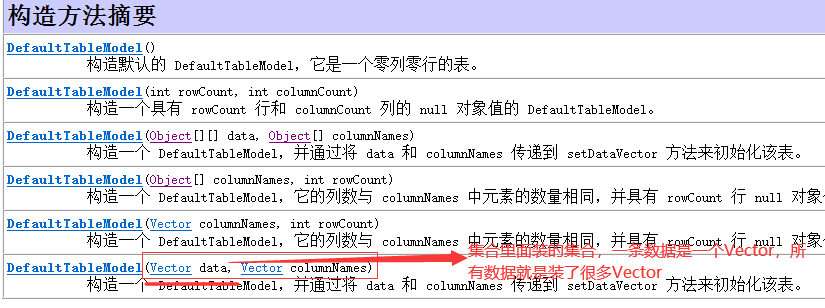


使用TableModel作为JTable数据的封装，查看一下TableModel的API。

TableModl是一个接口，所以需要传入它的实现类的对象。



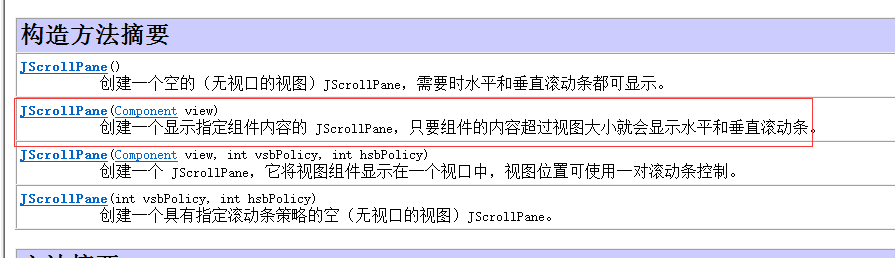
使用的构造函数如下：



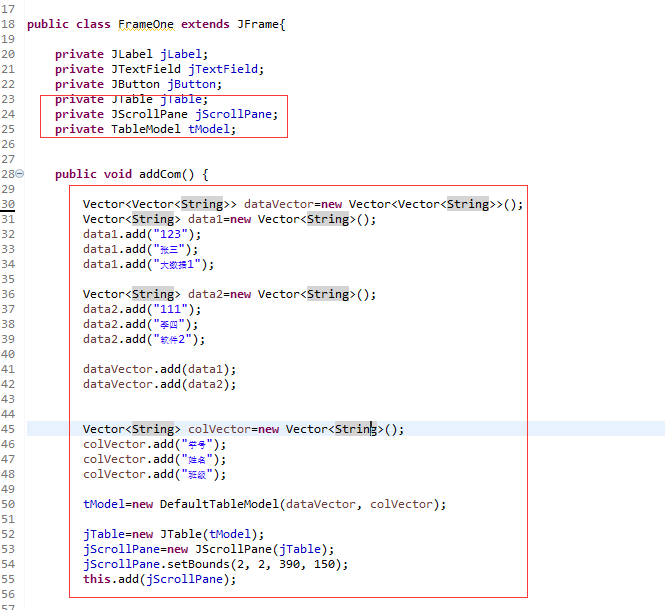
（2）解决JTable数据显示不全的情况

需要给JTable加入滚动条，以免数据量过大导致后面的数据看不到。

JScrollPane控件可以解决，相当于把JTable加入到JScrollPane中，然后再添加到界面。



根据这两个代码提示，完成JTable数据显示的代码



## \*作业：

**（1）要求把IO流存储到文件的数据读出来，然后用JTable显示**

**（2）通过界面输入数据放入集合对象中**

# 十三、多线程

## 13.1 多线程的概念

概念：

了解进程和线程的概念。进程是独立占用计算机硬件资源的，一个软件可能包含一个或者几个进程，当然一般来说都是一个，并且进程不能太多了。

为了解决进程占用资源的弊端，Java和其他高级语言就提出了一个线程的概念，线程的优点是不占用多余资源（**共享资源**：要用的时候才占用资源，不用的时候释放资源可以给其他的线程）。

使用场景：

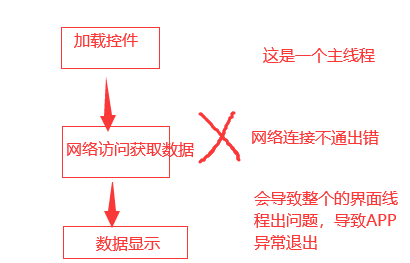
（1）游戏开发，控制多个角色和NPC同时动作，包括背景的变化。

（2）服务器的运行

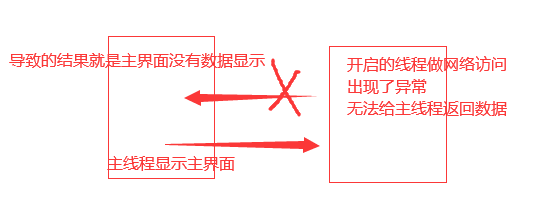
多线程的优势：

例子：Android的网络编程规定访问服务端接口的并且获取数据的代码必须放在非主线程里面。这样做的好处是，在网络不通的情况下，不会影响主线程的工作，最多就是你访问网络的线程出现了问题，数据没有返回。

主线程中使用代码访问网络的情况：



使用线程的情况如下：



## 13.2 多线程的实现

通过继承Thread类或者实现Runnable接口来实现。

第一步、新建一个类来继承Thread；

第二步、重写run函数

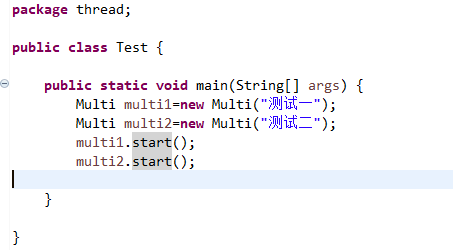
第三步、新建这个类的对象

第四步、调用start函数

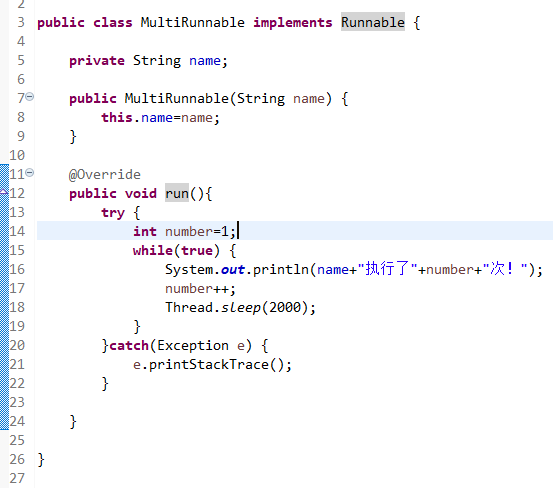
这样就可以开启一个线程，并且线程会执行run函数的代码

案例1：

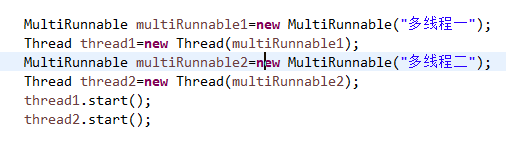




案例2：



启动的方式有区别，需要先把实现类的对象作为参数传入Thread类的构造函数，创建Thread对象，然后再通过调用start函数启动。



# 十四、JDBC

## 14.1 概念

Java的数据持久化的概念，用的最多的一种方式就是通过关系型数据库进行存储，常见的关系型数据库SQLServer、Oracle以及MySQL等等。

如果Java要去做数据库的操作（使用Java语言去操作数据库），必须要有底层的工具包。但是问题来了，就是数据库版本非常多，如果一个数据库厂商提供一套开发的工具包，那可能就会出现10多种不同的版本，如果这种情况出现，学习成本就太高了，那么可能Java就没人用了。

为了解决这个问题，当时SUN公司提出了一套标准的接口，去规定了各种操作数据库的方法名称、参数以及返回值。如果数据库厂商想要自己的产品对接Java，那么厂商来实现这一套接口，并且提供工具包给开发者。这样处理的话，那么开发者去操作所有的数据库只需要学习一套规则，就是SUN公司提出的这一套标准，这个标准就叫做JDBC（Java Database Connective）Java数据库连接。

## 14.2 JDBC如何使用

第一步、准备工作

开启一个MySQL服务，并且通过客户端去新建一个数据库，然后新建一张数据表。

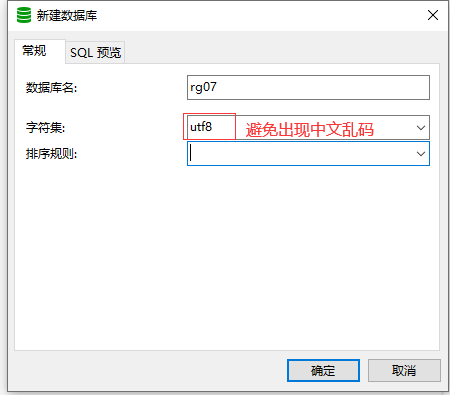
（1）打开MYSQL服务



（2）通过Navicat软件去连接MySQL服务



（3）新建自己的数据库

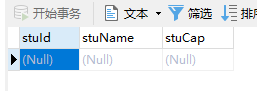


（4）新建一张表

create table stu(stuId varchar(32) primary key,

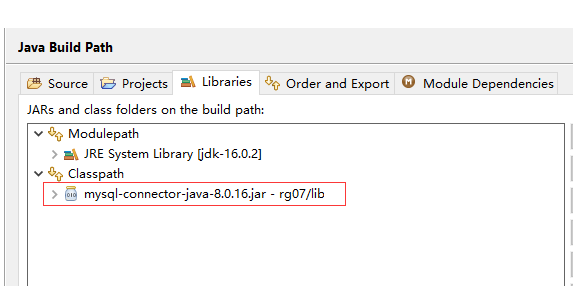
stuName varchar(32),stuCap varchar(32))

得到的结果如下：有一张stu的空表，里面有三个字段。

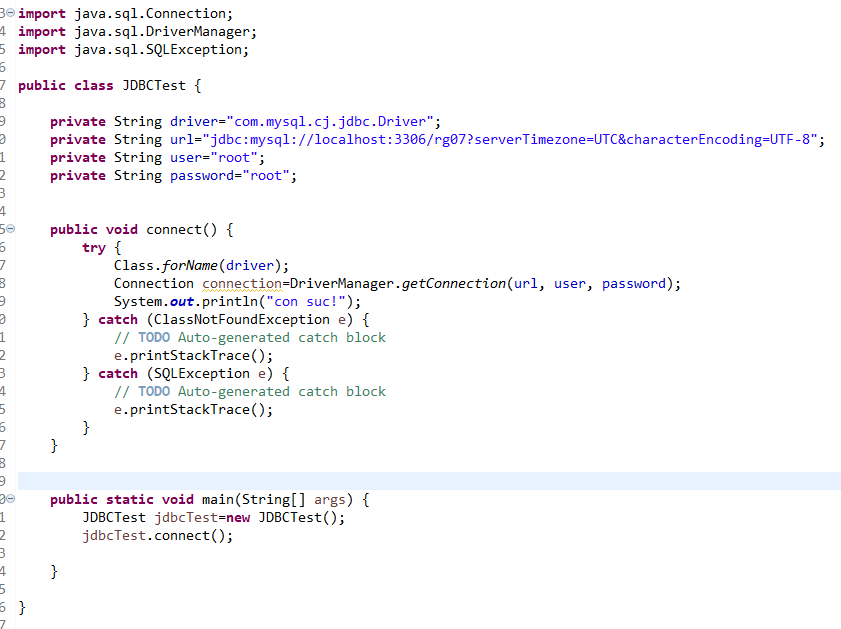


第二步、引入数据库的工具包

把开发包粘贴到你的项目里，然后添加到构建路径，结果如下：



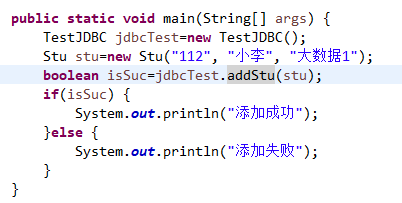
第三步、连接数据库



## 14.3 添加数据

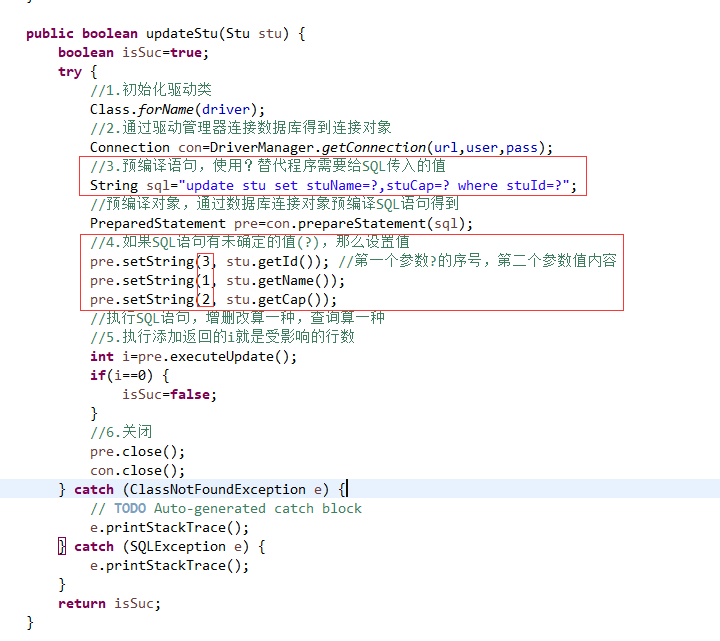
代码如下：

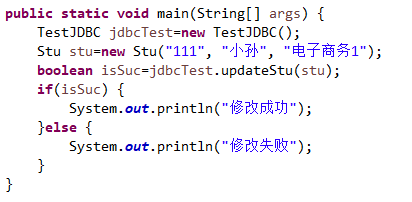




## 14.4 修改数据

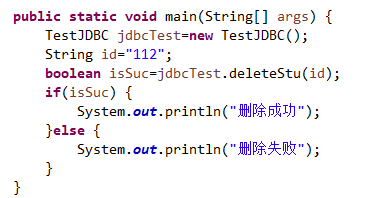
代码如下：





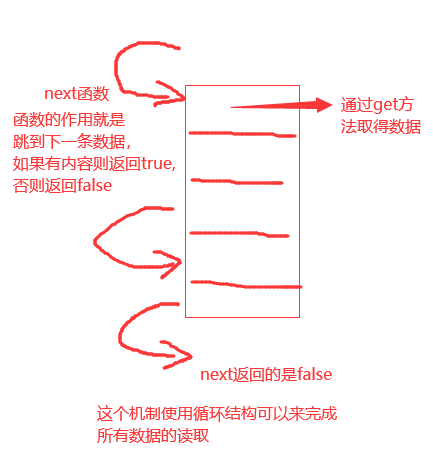
## 14.5 删除数据





## 14.6 查询数据

\*结果集数据的读取机制如下图：



案例代码：



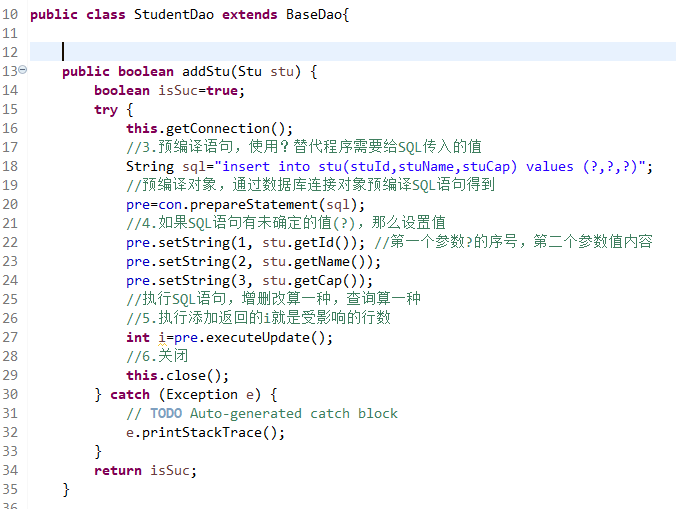
## 14.7 封装

把数据库配置、数据库连接、数据库连接资源的释放都封装到一个BaseDao基础类里面，其他的每张表的操作都放在不同的dao中，来继承BaseDao，然后可以实现数据库的连接和资源的释放方法统一使用。

BaseDao代码



StudentDao代码









## 14.8 应用-界面完成数据的增删改

把这个数据库的封装类在界面中引入，完成增删改查的功能。代码如下：



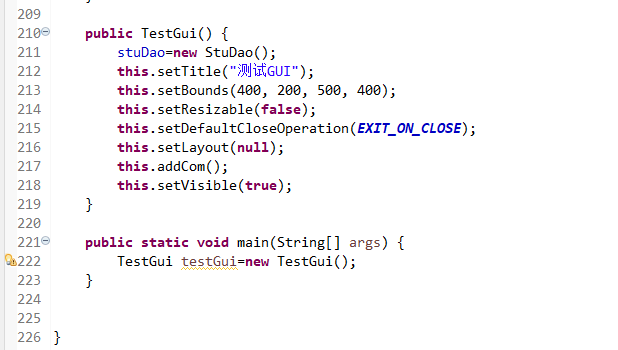










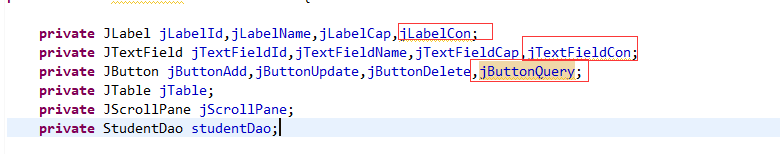


## 14.9 带条件的查询

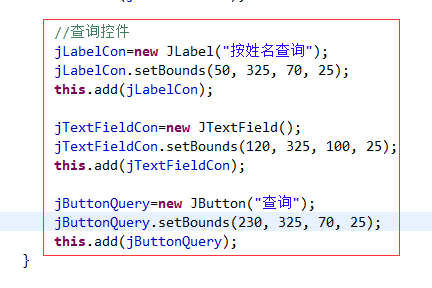
实现通过姓名做模糊查询

**第一步、完成界面的改造**

定义控件



addCom函数中加入控件

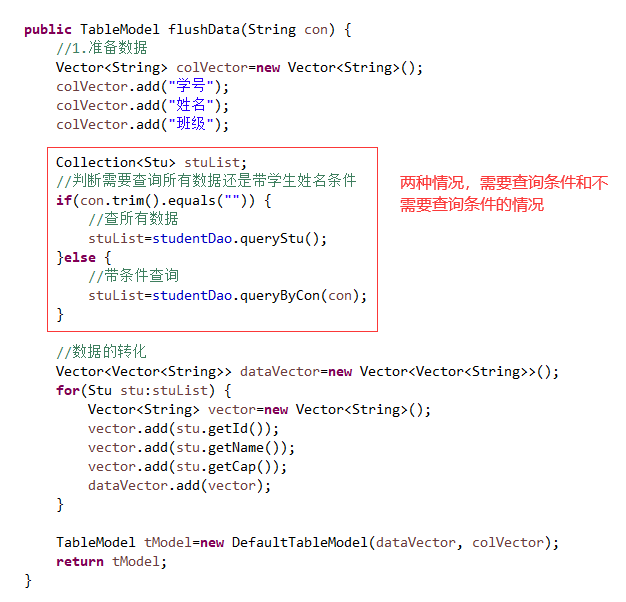


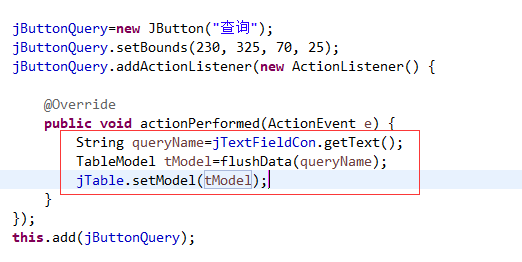
第二步、数据库代码

完成数据库表的根据姓名字段做模糊查询操作



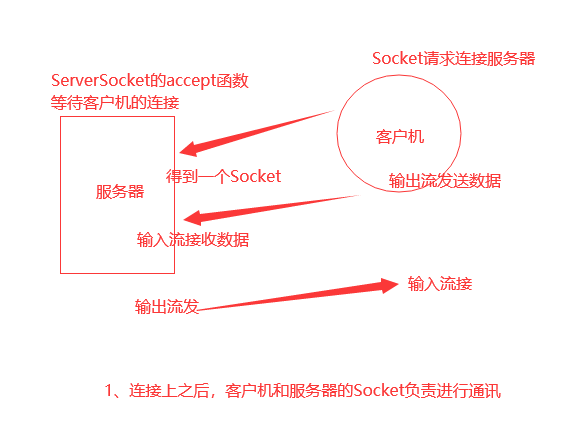
第三步、查询命令





# 十五、Socket编程

## 15.1 原理



## 15.2 案例代码



