Java 第一天　　4 月 23 日

1、Java 之父 Golsling

1995 年 5 月 23 日　Java 诞生

1998 年 12 月　　1.2 版本　Java2　　J2SE J2EE J2ME

2004 年 12 月　　1.5 版本(5.0)　Java　　JavaSE JavaEE JavaME

2、Java SE --- Java 标准平台

Java EE ---　企业级平台

Java ME ---　微小平台，用在资源受限的平台上

3、（1）跨平台 --- 一次编译，到处运行

（2）简　单 --- 简化 C++，取消了指针，对指针进行了上层的封装，它保证能够通过这个指针（引用），来访问有效的内存单元。

不允许多继承，使继承关系成树装图，每个类都只能由一个父类。

java 语言的开发效率高，但执行效率低。（相当于 c++的 55%）

（3）纯面向对象的语言 --- 都要放在类中

（4）垃圾回收机制 --- 自动垃圾收集，永远不会出现内存泄露的问题

4、虚拟机

java 语言是先编译后解释，java 源码是通过编译生成一种特殊的.class 的中间字节码文件，然后再由 JVM 进行解释运行。

（编译）　（解释）

.java　----> .class　---> 可执行文件

所以效率低，是 C++的 20 倍

5、java 源代码中的 main 方法的定义写法。main 方法是程序的入口。

```
public class Hello{          //一个源文件中只能有一个公开类，而且源文件的文件名与公开类的类名完全一致
    public static void main(String[] args){ //程序入口　public static 可以调换顺序
            System.out.println("Hello world");    //打印语句
    }
}
```

编译命令　javac xxxx.java 源文件的名字，源文件中的一个类会对应编译生成一个.class 文件

运行命令　java xxxx 类的名字 --- 启动虚拟机

6、环境变量

JAVA_HOME = /opt/jdk1.5.06　　JDK 安装路径　--- JDK = JRE {JVM(硬件)+编译器（软件）} +编译器工具+类库

PATH = $Path:$Java_Home/bin:.

ClassPath = .　　类路径

7、包 --- 分类放置，减少命名空间

包名.类名　表示一个类的全限定名

java xxx.xxx.ClassA　--- 运行时要在包结构的上一层目录来运行。

javac -d . xxxx.java　--- 编译的时候，按照包结构存放字节码文件，此命令生成的.class 文件在当前目录

package xxx.xxx，包的定义在一个程序中只能由一个

import xxx.xxx.xxx.ClassA; --- 在程序中声明 ClassA 类，使用的时候不需加包名，使用同一包内的类时，不用引入

import p1.p2.p3.p4.p5.*; --- 声明 p5 包中所有的类，不能代表其子包

系统会在每个 java 程序中隐含导入了 java.lang 这个包

8、java 中的注释，

单行注释 //……

多行注释 /* …….*/

文档注释/** ……..<p>(换行标签)*/,用 javadoc 命令可以根据原码中的文档注释生成注释文档（html 格式）。文档注释中可以使用 html 标签。

javadoc -d 路径（指定注释文档的保存路径） xxx.java

文档注释一般写在类定义之前，方法之前，属性之前。

在文档注释中可以用 @author 表示程序的作者，@version 表示程序的版本，前两个注释符号要写在类定义之前，用于方法的注释@param 对参数进行注释,@return 对返回值进行注释 @throws 对抛出异常的注释。

10、标识符规则 --- 类，方法，变量，同时也是包名的规范

由字母（汉语中的一个汉字是一个字母），数字，下划线和货币符号组成，不能以数字开头。

大小写敏感

没有长度限制，不能有空格

不能使用 Java 的关键字和保留字

java 中的关键字

goto 和 const 在 java 中虽然不再使用但是还作为保留字存在

java 中没有 sizeof 这个关键字了，java 中的 boolean 类型的值只能用 true 和 false，且这两个也是关键字

enum 枚举 assert 断言

一个标识符尽量符合语义信息，提高程序可读性

类　　名 ：每个单词首字母大写，

变量和方法 ：第一个单词小写，后边的每个单词首字母大写

包　　名 ：全部小写

常　　量 ：全部大写

11、局部变量：定义在方法中的变量

（1）先赋值后使用

（2）从定义变量的代码块开始到代码块结束

（3）在同一范围内不允许两个局部变量发生命名冲突

Java 第二天　　4 月 24 日

1、Java 的数据类型

（1）简单类型 --- 8 种有符号的数据类型

a) 整型

byte　　1 字节　8 位　　-128 到 127

short　　2 字节　16 位　　-2^15 到 2^15-1　-32768 到 32767

int　　4 字节　32 位　　-2^31 到 2^31-1　-2147483648 到 2147483647　　整型值存放，正数存放原码（二进制码），负数则存放补码（原码按位取反末位加一）。

long　　8 字节　64 位　　-2^63 到 2^63-1　long a = 40000l;　//在数字的结尾加“l”，表示 long 类型的字面值

b) 浮点类型

float　　4 字节　32 位　　float a = 1.3f;

double　　8 字节　64 位　　double a = 1.3;

double 和 long 比较，double 表示的数值要大，因为有科学计数法

float a = 1.3;　//error 会造成精度丢失，强转 float a = (float)1.3;或者 float a = 1.3f;

c) 字符类型

char　　2 字节　16 位

d) 布尔型

boolean　　false/true　　//Java 中 false 与 0 不能转换

自动类型提升：

a 运算符 b

a 和 b 中有一个是 double，结果就是 double

a 和 b 中有一个是 float，结果就是 float

a 和 b 中有一个是 long，结果就是 long

除以上之外的情况，结果都是 int

char 是无符号的 16 位整数,字面值必须用单引号括起来；　‘a’

String 是类，一个 String 的字符串是一个对象，非原始数据类型；

在字符串中使用"+"，表示字符串的连接，系统重载了这个运算符

char 类型也可以用通用转译字符，但是不能用 ASCII 码。可以用“\u0000”(十六进制)这种格式，因为 char 型中使用的是 unicode 编码方式。

```
char c1 = 'a';
char c2 = 97;
char c3 ='\u0061';  //一个字节用 2 个十六进制的数字表示
```

整数除 0 有异常，double 除 0 没有异常

（2）对象类型

2、java 中的运算符（java 的运算符的优先级和结合性和 c++相同）

*** System.out.println(3/2)　按整型计算 得 1 ***

```
byte a = 1;
a=a+2;      //不合法，自动类型提升，赋值丢失精度
a += 2 ;   //不会发生自动类型提升的,合法
a++ ;       //不会发生自动类型提升的
```

1) >>=　前面是零补零，前面是一补一；　右移一位比除以 2 效率高

2) >>>=　无符号右移（强制右移都会移进零），

>>=和>>>=对于负数不一样

正数：右移 n 位等于除以 2 的 n 次方

负数：变成正数。

3) &&　短路与，前面为假，表达式为假，后面的操作不会进行，& 非短路运算符，会对所有条件进行判断。

例：　int a = 4;

if(a<3&(b=a)==0)　b 赋值

if(a<3&&(b=a)==0)　b 不赋值

4) ||　短路或，前面为真，表达式为真，后面的操作不会进行，| 会对所有条件进行判断。

5)instanceof，是用于判断一个对象是否属于某个类型

6)java 中的求余运算符“%”可以对两个实型变量求余 5%2=1

3、流程控制

（1）控制流

if()

if()….else

if()…..else if()….else

注意：else 只是和其上面的同层的最近的 if()来配对。

（2）switch(s){

case 'a':……..

case 1:……break；

default:

……

}

注意：switch()内数据类型为 byte short char int 类型，只有以上四种类型的才可以在 switch()中使用。

case 块中不加 break 时顺序执行下面的语句。

4、循环语句

for(int i=0;i<n;i++){} ---- 确定循环

while(){} ---- 循环 0 或多次

do{} while(); ---- 注意加分号 循环 1 或多次

5、语法

在 main 方法中要使用全局变量或调用其他方法时，要求全局变量和方法要有 static 修饰

命令行参数 java Class a b

args[0] = a

args[0] = b

字符串转成 int 类型 int i = Integer.parseInt(a);

6、数组

内存中的连续空间

数组是一次性定义多个相同类型的变量

Java 中一个数组就是一个对象,有属性，没有方法

int[] a; //定义一个数组，变量名就是指向数组首地址的指针

a=new int[10]; //为数组分配空间

a.length; //得到数组的长度

数组创建后有初始值：数字类型为 0 布尔类型为 false 引用类型为 null

String[] 默认值 null

初始化、创建、和声明在同一时间

int[] i = {0,1}; //显示初始化

int[] i = new int[]{0,1}; //注意：这样显示的初始化的时候，new int[] 中括号中必须是空的，不能填数字

Car[] c = {new Car(),new Car()};

数组拷贝

System.arrayCopy(Object src, int srcPos, Object dest, int destPos, int length);

src 源数组，srcPos 从第几位开始拷贝（数组下标，从 0 开始），dest 目标数组，destPos 目标数组放置的起始位置，length，表示要拷贝的长度。拷贝一个数组到另一个数组。

多维数组

1) 有效定义

```
int[][] i1 = new int[2][3]; （同时给定一维，二维的空间）
int[][] i2 = new int[2][]; （给定一维的空间，二维空间待定，不规则的二维数组）
i2[0] = new int[2],i2[1] = new int[3];
```

* C++中 int[][] =new int[][3];有效

2) 无效定义

```
int[][] i1 = new int[][3];
```

3) 数组长度 ----------数组的属性 length（在二维数组中这个属性只代表第一维的长度）

```
int[] i = new int[5];
int len = i.length;    //len = 5;

Student[][] st = new Student[4][6];
len = st.length;          //len = 4;
len = st[0].length;     //len = 6;
```

练习：（1）数组元素的偏移

在使用数组的时候，通常使用一个辅助变量纪录数组中的有效元素的个数，还表示下一个可插入位置的下标

（2）数组元素的轮换

偶数个球队足球比赛，每个队和每个队有且只有一场比赛

把数字按"U"字型摆放，左边最上边的数字不变，其他数字按逆时针旋转

（3）有一个奇阶的方阵，填如数字，使得

第一个数一定放在第一行的最中间，第二个放在右上方，行越界，放在此列的另一侧，列越界放在此行的另一侧

行列都越界的时候，退回来，放在原来的数字下边，当要放入的地方有数字的时候，退回来，放在原来的数字下边

（1）

```
public class ArrayInsertDelete{
    static int[] a={1,2,3,4,5,6};
      static int index=6;
      static void insert(int pos,int value){
            if (index==a.length) expand();
            for(int i=index;i>pos;i--){
                  a[i]=a[i-1];
            }
            a[pos]=value;
        index++;
      }
      static void delete(int pos){
            index--;
```

```
            for(int i=pos;i<index;i++){
                a[i]=a[i+1];
            }
        }
        static void expand(){
            int[] b=new int[a.length*2];
            System.arraycopy(a,0,b,0,a.length);
            a=b;
        }
        static void print(){
            for(int i=0;i<index;i++){
                System.out.print(a[i]+"\t");
            }
            System.out.println();
        }

        public static void main(String[] args){
            print();
            insert(2,7); //{      1,2,7,3,4,5,6}
           print();
            delete(4);     //{1,2,7,3,5,6}
            print();
       }

}
```

Java 第三天　　4 月 25 日

1、面向对象的思想

anything is an Object（万物皆对象）

符合人们客观看待世界的规律

2、抽象，从对具体的对象中抽取有用信息。

对象有其固有属性，对象的方法，即对象的行为（对象能做什么）

对象本身是简单的（功能简单），多个对象可以组成复杂的系统（对象之间彼此调用对方的方法）

3、面向对象的优点

1）对象应当是各司其职（功能简单），各尽所能（把自己的功能作到最好）。（弱耦合性实现了前面所述的对象的特点）

2）对象的耦合性，是对象之间的联系，对象和系统之间的联系。

对象的耦合性要尽量的弱，也就是对象之间的联系尽可能的弱，对象和系统之间的联系尽可能的弱。

弱耦合性要通过一个标准来实现

3）可重用性

对象的功能越简单，复用性就越好。（对象的耦合性弱，复用性就比较强）

4）可扩展性

系统的可插入性，是在系统中加入新的对象之后的系统稳定性。

对象的可替换性，是在系统中替换原有的对象之后的系统的稳定性。

4、面向过程和面向对象的比较

面向过程是先有算法，后又数据结构 --- （怎么解决问题）

面向对象是先有对象（数据结构），后有算法（对象间的方法调用） --- （用什么做）

5、类的定义

1）属性

类型 变量名； --> 实例变量，系统赋初始值

	默认值	使用范围	命名冲突
实例变量	系统赋初始值	在类的内部使用	允许实例变量和局部变量发生命名冲突，变量的值->局部优先
局部变量	先赋值后使用	定义他的代码块	同范围内不允许两个局部变量发生命名冲突

2）方法的定义：

不允许方法的声明和实现分开

方法声明的组成：

(a)方法的修饰符（0 或多个修饰符出现的顺序无关） |

(b)方法的返回值类型 |顺

(c)方法名 |序

(d)方法的参数表 |向

(e)方法中允许抛出的异常 |下

V

6、方法的重载

java 中方法的重载（overload）方法名相同，参数表不同（个数，类型，排列顺序），其他部分可以不同。

调用时要给出明确参数并确定调用某一方法。在编译时，编译器会根据参数选择适当的方法，所以重载也叫编译时多态。

** 方法的重载可以屏蔽一个对象同一类方法由于参数不同所造成的差异 **

向上就近匹配原则

如果方法的参数表中的数据类型和调用时给出的参数类型不尽相同时会根据向上匹配的就近原则。(类型就近向上转化匹配）

```
public void m(byte a,short b){
    System.out.println("m(byte,short)");
}

public void m(short a,byte b){
    System.out.println("m(short,byte)");
}
```

此时若用 2 个 byte 作为参数调用 m()方法，编译会出错，因为按照向上就近匹配原则，提升哪个都可以，出现语义不明确的错误

7、构造方法

1）方法名与类名相同，无返回值

2）系统提供默认无参的，空的构造方法

3）若自定义一个有参的构造方法，则系统将不提供无参的，空的构造方法

不能手工调用，在一个对象的生命周期中，只会被调用一次

8、对象的创建

Java 在堆中创建对象的过程：

1）分配空间

2）初始化属性，若声明实例变量的时候没有初始化，用默认值初始化属性

3）调用某个构造方法（前提：不考虑继承关系）

Student s；　//声明一个 Student 类的对象变量

s = new Student();　//调用 Student 类的无参的构造方法，创建一个对象,把这个对象的地址赋给一个 Student 类的引用

（引用指向了这个对象，引用中保存对象的首地址，以后都是通过引用访问这个对象的属性和方法）

s.age　　s 引用所指向的对象的 age 属性

9、变量

包括简单变量（原始数据类型），对象变量。

简单变量传值，对象变量传址！

```
static void method3(Student s){
   s=new Student();          //创建一个新对象，把对象的地址赋给形参的引用
   s.age=20;                  //改变新对象的状态
}

Student stu=new Student();
System.out.println(stu.age);     //  打印出 Student 类的 age 属性的初始值
method3(stu);         //method3 方法返回，形参 s 指向的被改变的对象已经成为了垃圾对象
System.out.println(stu.age);     //打印的仍是原来那个对象的 age 的值
```

10、this

是一个引用，指向的是自己，即当前对象（哪个对象调用了方法，哪个对象就是当前对象）

可以用来区分实例变量和局部变量。this.age = age ;

this(),他表示调用本类其他的构造方法，注，只能写在构造方法的第一行。

项目练习：

项目名称:Bank Account Management System 银行账户管理系统 简称 BAM

项目描述:这是一个基于 C/S 结构的银行账户在线管理系统,用户可以通过 ATM 终端界面来操作自己的银行账户.

项目实施方式:这是一个同步练习,随着达内 CoreJava 课程的深入,这个项目将趋于完整,学员的任务是随着知识点的深入,完成每一个进阶的项目要求.

练习 1:

(面向对象基础语法)

写一个账户类(Account)

属性:

id:账户号码 长整数

password:账户密码　String

name:真实姓名　String

personId:身份证号码 String

email:客户的电子邮箱　String

balance:账户余额　double

方法:
deposit: 存款方法,参数是 double 型的金额
withdraw:取款方法,参数是 double 型的金额
构造方法:
有参和无参,有参构造方法用于设置必要的属性

Java 第四天　　4 月 26 日

1、复习

面向对象思想

符合我们看待客观世界的规律

Everything is an object

对象：客观存在，有什么（属性），能做什么（方法）

每个对象都是简单的，有简单的对象拼装成复杂的系统

面向对象的优点：

1）各司其职，各尽所能

2）弱耦合性

3）可重用性，可扩展性

类：对象共性的抽象，客观对象在人脑中的主观反映

对象的模版

属性：在类里，但在任何方法之外定义的一个变量 --- 实例变量

有默认值，访问范围，至少是本类内部，可以和局部变量发生命名上的冲突，局部优先，this 区分

方法：方法定义（声明对象能做什么）/方法实现（怎么做）

定义：修饰符 返回值类型 方法名 （参数表） 抛出的异常

重载（Overload）：方法名相同，参数表不同。

编译时多态，编译器根据参数表选择一个方法

为什么要有方法的重载？

屏蔽一个对象同一类方法由于参数表不同所造成的差异

就近向上匹配

构造方法：没有返回值，方法名和类名是相同的，系统提供默认无参的空的构造方法，** 建议自己写无参构造

构造对象的过程：

1）分配空间

2）初始化属性

3）调用构造方法

类名 引用名=new 类名(构造参数);

方法参数传递规则：

简单类型参数：传值

对象类型参数：传引用，实参和形参指向同一个对象

2、面向对象的三大特征：封装、继承、多态。

java 中的封装

封装，一个对象和外界的联系应当通过一个统一的接口，应当公开的公开，应当隐藏的隐藏。

（对象的属性应当隐藏），一个对象的内部是透明的，就是把对象内部的可透明性和隐藏的特性区分开，该透明的透明，该隐藏的隐藏。

（封装的属性）java 中类的属性的访问权限的默认值不是 private，要想隐藏该属性或方法，就可以加 private（私有）修饰符，来限制只能够在类的内部进行访问。

对于类中的私有属性，要对其给出一对方法（getXxx(),setXxx()）访问私有属性，保证对私有属性的操作的安全性。

方法的封装，对于方法的封装，该公开的公开，该隐藏的隐藏。方法公开的是方法的声明（定义），即（只须知道参数和返回值就可以调用该方法），隐藏方法的实现会使实现的改变对架构的影响最小化。。

封装会使方法实现的改变对架构的影响最小化。

完全的封装，类的属性全部私有化，并且提供一对方法来访问属性。

Java Bean 一种组件规范 --> 所有属性私有，访问方法按照命名规范 setXxx(),getXxx()方法

3、java 中的继承

继承，是对有着共同特性的多类事物，进行再抽象成一个类。这个类就是多类事物的父类。父类的意义在于可以抽取多类事物的共性。

泛化：把共性从子类中抽取出来。

特化：现有父类，再有子类的过程

父类与子类 -> 从一般到特殊的关系

java 中的继承要使用 extends 关键字，并且 java 中只允许单继承，也就是一个类只能有一个直接的父类。

这样就是继承关系呈树状，体现了 java 的简单性。

子类只能继承在父类中可以访问的属性和方法（实际上父类中私有的属性和方法也会被继承但子类中无法访问罢了）。

实际上一个子类对象中包含一个父类对象

访问控制修饰符:（可以修饰属性和方法）

private 修饰符，表示只有本类内部可以访问，不能继承。

default 修饰符，方法不加修饰符，会默认为 default，表示在同一个包中可以访问，父子类在同一包中，子类可以继承父类的相应内容。（可以修饰类）

protected（保护）修饰符，表示同一包中可以访问，不同包的子类也可以访问继承。

public 修饰符，表示公开，在任何地方都可以访问，能继承。（可以修饰类）

修饰符的权限是由上而下逐渐变宽的。

继承的意义，就在于子类可以在父类的基础之上对父类的功能进行发展,继承可以使系统的耦合性降低，也就是使对象间的联系便的松散，使多类对象间的联系用其父类对象代替。

注意：构造方法不能被继承。

父类的属性及方法的确定：要从子类的角度来看子类间的共性，当所有子类都有这个属性时，就应当考虑是否该放在父类中，方法也是如此，方法可以被看作是对象的行为，而类的方法这时这一类对象所共有的行为，所以也应当在方法的确定时注意是不是所有的子类型中都需要有这种方法，并且会根据不同的类型的行为的方式也不同才可以覆盖着个方法。

4、java 中方法的覆盖

子类中有和父类中可访问（可继承到子类）的同名同返回类型同参数表的方法，就会覆盖从父类继承来的方法。

在父子类中，出现方法名相同，参数表不同的情况，叫方法的重载

方法覆盖父子类返回值类型也要相同

注意：在 jdk1.4 以前要求方法的覆盖时，需要方法的返回值，参数表，方法名必须严格相同，而在 jdk1.5 中方法覆盖，子类的中覆盖的方法的返回值可以是父类中被覆盖的方法的返回值类型的子类型。

注意：子类的方法覆盖父类的方法时，方法的修饰符要么相同，要么子类中的方法的修饰符表示的访问权限要宽于父类。父类中的私有方法，不能被继承到子类，就是说子类中即使将其覆盖了也不会有多态。

注意：父子类中有同名的属性不叫子类覆盖了父类的属性，这种情况较作属性的遮盖（shadow）。

5、super 关键字

(1)区分父子类间的属性遮盖和方法覆盖

(2)super()，表示在子类的构造方法中调用父类的构造方法（可以通过这种方法在子类的构造方法中初始化父类中的属性），super()也只能出现在构造方法的第一句上。super(),在子类的构造方中指明构造父类时调用哪一个父类的构造方法构造父类。

super,这里所表示的是一个父类的对象，可以通过 super 来使用父类中可以访问的方法（可以在父类中定义 setXxx()，getXxx()方法来访问父类中的私有属性），super 可以屏蔽父子类中同名属性的冲突。

注意：在写类的时候，一定要写默认无参的构造方法，如果一个构造方法的第一句既不是 this(),也不是 super()时，那么就会在这里隐含的调用他的父类的无参的构造方法，即隐含的有 super()。

6、创建对象的过程

1）递归的构造父类的对象，默认调用父类无参的构造方法 super()

2）分配本类空间

3）初始化本类实例变量（属性）

4）调用本类的构造方法

7、java 中的多态 --- 运行时多态

```
Animal a = new Dog();     //引用类型可以是对象类型的父类
```

对象类型　客观类型

引用类型　主观类型

以子类覆盖了父类的方法为前提

1）对象类型不变

2）只能对一个对象调用引用类型中定义的方法

3）运行时会根据对象类型找覆盖之后的方法

```
Animal a = new Dog();
Dog d = new Dog();
a = d;          //把子类引用赋值给父类引用，是合法的
d = (Dog)a; //把父类引用赋值给自类引用，需要强转
```

引用 instanceof 类名　　//判断前面的引用和后面的类是否兼容

可以翻译为“是不是”

```
a instanceof Dag;      //a 指向的对象是不是 Dog 类
```

一般用在强制类型转换之前，避免转换异常

多态可以使代码变得更通用，以适应需求的变化。也就是定义在父类中的方法，可以在子类中有不同的实现

将其覆盖，在为父类型的对象变量赋值相应需要功能的子类的对象实例。
可以屏蔽不同子类的差异

作业：
1.设计一个形状类,方法:求周长和求面积
形状类的子类:Rect(矩形),Circle(圆形)
Rect 类的子类:Square(正方形)
不同的子类会有不同的计算周长和面积的方法
创建三个不同的形状对象,放在 Shape 类型的数组里,分别打印出每个对象的周长和面积

2.某公司的雇员分为以下若干类：
Employee：这是所有员工总的父类，属性：员工的姓名,员工的生日月份。方法：getSalary(int month) 根据参数月份来确定工资，如果该月员工过生日，则公司会额外奖励 100 元。
SalariedEmployee：Employee 的子类，拿固定工资的员工。属性：月薪
HourlyEmployee：Employee 的子类，按小时拿工资的员工，每月工作超出 160 小时的部分按照 1.5 倍工资发放。属性：每小时的工资、每月工作的小时数
SalesEmployee：Employee 的子类，销售人员，工资由月销售额和提成率决定。属性：月销售额、提成率
BasePlusSalesEmployee：SalesEmployee 的子类，有固定底薪的销售人员，工资由底薪加上销售提成部分。属性：底薪。

写一个程序，把若干各种类型的员工放在一个 Employee 数组里，写一个函数，打印出某月每个员工的工资数额。
注意：要求把每个类都做成完全封装，不允许非私有化属性。

Java 第五天　　4 月 27 日

一、复习
1、封装
该隐藏隐藏，该公开的公开
属性隐藏，同时提供 get/set 方法
有些方法应该隐藏
方法声明公开，实现隐藏。实现的改变对架构的影响最小
2、继承
一般－>特殊
单继承：一个类最多只能有一个直接父类。类之间可以形成树状关系

根据访问权限，子类如果可以访问父类的属性和方法，就能继承
private 私有　不能继承
default 本类＋同包　同包子类可以继承，不同包子类不能继承
protected 本类＋同包＋不同包子类　可以继承
public 公开 可以继承

方法的覆盖（Override）：
方法名相同，参数表相同，返回值相同，访问修饰符比父类相同或更宽，抛出的异常不能比父类更宽

继承关系下对象的构造过程
1）递归的构造父类对象

2）分配本类空间
3）初始化本类属性
4）调用本类的构造方法

super：
super. ->父类对象,访问父类被覆盖的方法或者父类被遮盖的属性
super() ->用在构造方法时，用来指明调用父类的哪个构造方法，放在构造方法的第一行，默认调用父类无参构造方法

3、多态
编译时多态：方法的重载
运行时多态：

子类对象当作父类对象来用!!! 屏蔽同一父类下，不同子类差异
Animal a = new Dog();
允许引用类型和对象类型不同，但要求引用类型是对象类型的父类。
对象类型代表了对象自身客观的实际类型，引用类型代表了主观上把对象当作什么类型来看待。
引用类型：编译时类型，主观类型
对象类型：运行时类型，客观类型

1）对象运行时类型不变
2）只能对对象调用其编译时类型定义的方法
3）运行时根据对象类型去匹配对象类型中覆盖之后的方法

```
Super s1;
Sub s2;
s1=s2;
s2=s1; //error , s2=(Sub)s2
```

强制类型转换：当我们把父类的引用赋值给子类引用的时候，需要强制类型转换。强制类型转换失败：类型转换异常.
为了避免类型转换异常，使用 instanceof 判断
引用 instanceof 类名 引用指向的对象的类型与后面的类名是否兼容

多态的作用：通用编程
我们可以把不同子类的对象都当作父类对象看待，评比不同子类的差异。

二、CARP(组合/聚合复用原则)
实现代码重用最好的方法不是继承

两种复用
白盒复用，也就是继承复用，破坏封装，父类中的可以被子类访问到的就可以被继承，这样会有些不需要的内容被继承下来，所以这种方式不太好。
黑盒复用，也叫组合复用，也就是把要复用代码的类的对象作为本类中的一个属性，然后再通过方法的委托来实现由选择的复用。
方法的委托就是在本类的方法内部通过该类的对象调用要使用类的方法，不破坏封装。
注意：尽量用组合复用替代继承复用。

三、多态

1、多态用于参数，可以在方法的参数中传入其父类类型，在运行时会根据实际的运行时类型来在方法中进行相应的操作。

多态一般用在方法的参数上

```
void feed(Animal a){
   a.eat();
}
```

调用的时候 feed(new Dog()); //运行时，调用的是 Dog 类中覆盖的 eat()方法

2、多态用于返回值，可以在方法的返回值类型上是用其实际返回值的父类型，在使用其返回值时也不关心其实际类型。

```
public static Animal getAnimal(int type){
        if (type==0) return new Dog();
        else return new Cat();
    }
```

屏蔽子类差异，可扩展（只修改方法的实现，不必修改方法的声明）

3、Animal a = new Dog();

a.age; //访问属性是没有多态的，访问的是引用的 age 属性

a.eat(); //调用 Dog 类中覆盖 Animal 类中的 eat()方法，多态

4、Animal a = new Dog();

method(a);

运行结果调用参数是 Animal 类对象的那个 method()方法

方法的重载只看引用类型，跟引用指向的对象类型没有关系

四、对象的比较

```
String s1 = new String("abc");
String s2 = new String("abc");
```

s1 == s2; -> false 判断两个引用是否指向同一对象，即地址是否相同

s1.equls(s2); -> true 判断两个引用指向的对象的内容是否相同

练习：

(继承,多态)

银行的客户分为两类,储蓄账户(SavingAccount)和信用账户(CreditAccount),区别在于储蓄账户不允许透支而信用账户可以透支,并允许用户设置自己的透支额度.

注意:CreditAccount 需要多一个属性 ceiling 透支额度

为这两种用户编写相关的类

同时要求编写 Bank 类,属性:

1.当前所有的账户对象的集合,存放在数组中

2.当前账户数量

方法:

1.用户开户,需要的参数:id,密码,密码确认,姓名,身份证号码,邮箱,账户类型,返回新创建的 Account 对象

提示:用 s1.equals(s2) 可以比较 s1,s2 两个字符串的值是否相等.账户类型是一个整数,为 0 的时候表示储蓄账户,为 1 的时候表示信用账户
2.用户登录,参数:id,密码 返回 Account 对象
3.用户存款,参数:id,存款数额,返回修改过的 Account 对象
4.用户取款,参数:id,取款数额,返回修改过的 Account 对象
5.设置透支额度 参数:id,新的额度 ,返回修改过的 Account 对象.这个方法需要验证账户是否是信用账户

用户会通过调用 Bank 对象以上的方法来操作自己的账户,请分析各个方法需要的参数

另外,请为 Bank 类添加几个统计方法
1.统计银行所有账户余额总数
2.统计所有信用账户透支额度总数
写个主方法测试你写的类

Java 第六天 4 月 28 日

修饰符

一、static
修饰属性，方法，代码块

1、静态属性：全类公有，称为类变量
那么这个属性就可以用 类名.属性名 来访问
（共有的类变量与对象无关，只和类有关）
类加载：虚拟机通过 I/O 流把一个类的信息从字节码文件中读入虚拟机并保存起来
一个类只会加载一次
类变量，会在加载时自动初始化，初始化规则和实例变量相同。
注意：类中的实例变量是在创建对象时被初始化的，被 static 修饰的属性，也就是类变量，是在类加载时被创建并进行初始化，类加载的过程是进行一次。也就是类变量只会被创建一次。

2、静态方法：
使这个方法成为整个类所公有的方法，可以用 类名.方法名 直接访问

注意：static 修饰的方法，不能直接访问（可以通过组合方式访问）本类中的非静态(static)成员（包括方法和属性）
本类的非静态（static）方法可以访问本类的静态成员（包括方法和属性），可以调用静态方法。
静态方法要慎重使用。在静态方法中不能出现 this 关键字，因为这是针对对象而言的。

java 中的 main 方法必须写成 static 的，因为，在类加载时无法创建对象，静态方法可以不通过对象调用。
所以在类加载时就可以通过 main 方法入口来运行程序。

注意：父类中是静态方法，子类中不能覆盖为非静态方法。
在符合覆盖规则的前提下，在父子类中，父类中的静态方法可以被子类中的静态方法覆盖，但是没有多态。
使用引用调静态方法，相当于使用引用的类型去调用静态方法。（在使用对象调用静态方法是其实是调用编译时类型的静态方法）

3、初始代码块

在定义属性的位置上，在任何方法之外，定义一个代码块

动态初始代码块：在初始化属性之前调用初始化代码块 {……}

静态初始代码块：在类加载时运行 static{……} 只被运行一次，往往用作一个类的准备工作

二、一个类在什么时候被加载？时机 （延迟加载，能不加载就不加载）

（1）new 一个对象的时候，加载

（2）没有创建对象，访问类中静态成员（方法和属性），加载

（3）声明一个类的引用，不加载

（4）创建子类，先加载父类，再加载子类

（5）父类中的公开静态方法，子类继承，使用子类的类名调用此方法，加载父类

```
class Super{
        public static m(){}
}
class Sub extends Super{}
```

在主函数中运行以下代码：

```
Sub.m();    //加载了父类之后，虚拟机已经知道 m()方法的调用了，就不会再加载子类，延迟加载
```

（6）没有创建对象，访问类中静态常量（能计算出结果的常量，在编译的时候会用计算出来的结果替换表达式），不加载

没有创建对象，访问类中静态常量（不确定的值），加载

（7）CoreJava day16

三、设计模式（编程套路）

GOF（Group Of Four）四人帮模式 23 种

1、单例模式 Singleton：

```
class A{
        private static A a = new A();   //私有静态的实例变量指向自己，在类加载时创建唯一对象
        public static A newInstance(){ //提供公开静态的访问点，回返唯一实例
             return a;
        }
        private A(){}          //私有的构造方法，防止滥用
}
```

2、不变模式 ：

便于实例共享，减少对存储空间的消耗

String 类采用了不变模式

字符串中的内容是不变的

```
String a1 = "123";   //系统会先去串池中找"123",找到，就共享使用一个，没找到就在串池中创建一个
String a2 = new String("123");   //在堆空间中创建"123"
```

池化的思想，把需要共享的数据放在池中（节省空间，共享数据）

只有 String 类可以用“ ”中的字面值创建对象。

在 String 类中，以字面值创建时，会到串池空间中去查找，如果有就返回串池中字符串的地址，并把这个地址付给对象变量。

如果没有则会在串池里创建一个字符串对象，并返回其地址付购对象变量，当另一个以字面值创建对象时则

会重复上述过程。

如果是 new 在堆空间中创建 String 类的对象，则不会有上述的过程。

a2=a1.intern(); //返回字符串在串池中的引用

消极方面：字符串连接“+”，产生很多的中间对象

StringBuffer 类，字符串是可变的

s.append("A"); //连接字符串，不创建中间对象

大量字符串连接的时候用 StringBuffer 取代 String

四、final

修饰变量，方法，类

1、修饰变量

被 fianl 修饰的变量就是常量（常量名大写），一旦赋值不能改变

修饰局部变量：修饰基本数据类型 -> 变量的值不能改变

修饰引用 -> 引用只能指向固定的对象

修饰实例变量：默认值不生效，可以再赋值

有两次赋值机会：初始化变量的时候 final int a = 20; 对于直接在初始化时赋值，final 修饰符常和 static 修饰符一起使用，避免浪费空间

构造方法中设置 this.a = a;

但是不能同时使用这两种方法

在一个对象完成创建的时候，对象中的所有 final 属性必须都完成赋值

类变量可以是 final 的，也有两次赋值机会 ：定义变量的时候就赋值 ； 静态初始代码块中

2、修饰方法

不能被子类覆盖

从面向对象的角度理解，可以保持操作的稳定性

3、修饰类

不能被继承

在树状单继承关系中，final 类是树叶节点

在一个 final 类中的所有方法，默认都是 final 的

注意：final，不能用来修饰构造方法。

在父类中如果有常量属性，在子类中使用常量属性时是不会进行父类的类加载。

静态常量如果其值可以确定，就不会加载该类，如果不能确定则会加载该常量所在的类。

```
class Super{
    private final void m(){}    //用 final 可以证明出 private 的方法不继承给子类
}
class Sub extends Super{
    public void m(){}   //不是方法的覆盖
}
```

五、abstract 抽象的

修饰类和方法

1、修饰类 -> 抽象类

不能创建对象，可以声明引用，并通过引用调用类中的方法

主要用于被子类继承的，可以用父类引用指向子类对象

2、修饰方法

只有声明，没有实现，用“;”代替“{}”

需要子类继承实现（覆盖）。

如果一个类中有抽象方法，那么这个类必须是抽象类。

抽象类中不一定有抽象方法

注意：父类是抽象类，其中有抽象方法，子类继承父类，必须把父类中的所有抽象方法都实现（覆盖）了，子类才有创建对象的能力，

否则子类也必须是抽象类。

抽象类中可以有构造方法，是子类在构造子类对象时需要调用的父类（抽象类）的构造方法。

抽象类的合理性：

没有抽象类的实例，只有抽象类子类的实例

抽象方法，定义和实现分离

抽象（abstract）方法代表了某种标准，定义标准，定义功能，在子类中去实现功能（子类继承了父类并需要给出从父类继承的抽象方法的实现）。

方法一时间想不到怎么被实现，或有意要子类去实现而定义某种标准，这个方法可以被定义为抽象。(abstract)

六、三个修饰符都能修饰方法（不包含构造方法）

1、构造方法在创建对象的时候使用，如果是 static，那么只会在加载类的时候调用一次

构造方法不能被继承（final），谈不到覆盖，更不会由子类实现（abstract）

2、final 和 abstract，private 和 abstract，static 和 abstract，这些是不能放在一起的修饰符

因为 abstract 修饰的方法是必须在其子类中实现（覆盖），才能以多态方式调用，以上修饰符在修饰方法时子类都覆盖不了这个方法。

final 是不可以覆盖，private 是不能够继承到子类，所以也就不能覆盖。

static 是可以覆盖的，但是在调用时会调用编译时类型的方法（引用类型的方法），因为调用的是父类的方法，而父类的方法又是抽象的方法，不能调用。

所以上的修饰符不能放在一起。

作业:(语言高级特性,三个修饰符)

1.修改 Account 类,银行用户的账号(id)是自动生成的,初始值为 100000,第一个开户的用户 id 为 100001,第二个为 100002,依此类推.

提示:构造对象的时候采用 static 属性为 id 赋值

2.对于 Account 类,有两个方法,存款方法和取款方法,请修改这两个方法.

存款方法改为不允许子类修改

取款方法根据不同的子类而不同,因此,改为抽象方法,在两个子类中分别实现

3.将 Bank 类作成单例

Java 第七天　　4 月 29 日

一、复习

static

属性　类变量 全类共有 类加载时初始化，类名访问

方法　静态方法 类名调用，静态方法中不能访问类的非静态成员，可以覆盖，只能被静态方法覆盖，没有多态

初始代码块　类加载时运行

类加载：

一个类编译之后会形成.class 文件，储存了类的全部信息。

当 JVM 第一次使用一个类的时候，会根据 ClassPath 找到对应的.class 文件，用输入流把文件中的信息读入 JVM 并保存起来，这样，JVM 就“认识”了这个类

类加载时机：第一次用这个类的时候

1. 第一次创建类的对象，会加载
2. 访问类的静态成员，会加载
3. 声明类的引用，不会加载
4. 加载子类，必然先加载父类
5. 如果调用的是子类从父类中继承下来的静态方法，只会加载父类
6. 如果访问的是类的静态常量，如果在编译的时候能够确定这个常量的值，则运行时不会加载，
   否则，编译时无法确定常量值，那么运行时就会加载

设计模式：单例模式

用途：一个类只能有一个对象

实现：会有一个静态的属性，就是该类的一个对象。提供一个静态方法，来获得这个唯一的静态属性（单例），同时构造方法私有

final

变量：常量，一旦赋值，不能改变，为属性常量赋值的时机：对于实例常量，初始化或者构造方法中可以赋值；
对于类常量(static final)，初始化或者静态初始代码块中可以赋值

方法：不能被覆盖

类 ：不能被继承　所有方法默认都是 final 的

abstract

方法 ：只有定义，没有实现

类 ：不能构造对象，但可以用来声明一个引用（作为编译时类型）

如果一个类有抽象方法，这个类必须是抽象类，如果一个类是抽象类，不一定有抽象方法

子类继承一个抽象类，就必须覆盖（实现）父类（抽象类）中的所有抽象方法，否则，子类就也得是抽象类

抽象类有构造方法，给子类继承

通过 abstract，我们可以把一个方法的定义放在父类，留给子类实现。

二、接口

接口是一种程序结构，是特殊的抽象类。

接口中的方法必须都是公开的抽象方法（public abstract），接口中的属性都是公开静态常量（public static final）。

接口中没有构造方法

```
abstract class ClassA{
    public static final int A = 10;
    public static final double B = 3.14;
    public abstract void m1();
```

```
    public abstract void m2();
}

interface IA{
    int A = 10;
    double B = 3.14;
    void m1();
    void m2();
}   //与上面的抽象类逻辑上等价，编译之后，也会生成一个 IA.class 文件
```

所以一个源文件中可以写多个接口，但只能有一个公开接口，并且要与源文件的名字一致

接口可以继承，但是只能由接口继承。

在用类去继承时要换用 implements 关键字，这时类和接口也不叫做继承关系，而是实现关系，但其实质也是继承。

一个类可以继承也只能继承另外一个类，但是可以实现多个接口，其语法是在 implements 后面写接口名，多个接口以“,”分隔。

接口之间是可以多继承的，其语法和类的继承语法是相同的。

在接口多继承时，在 extends 后写接口名如果要继承多个接口，接口名以“,”分隔，接口的继承关系只是把其父接口中的抽象方法继承到子接口中。

要实现接口就必须实现接口中的所有方法。

一个类可以在继承一个类的同时，还可以实现一个或多个接口。采用接口就绕开了单继承限制。

class Impl extends ClassE implements IA，IB{……}

接口类型也可以做为编译时类型使用，但其实际的运行时类型必须是完全实现接口的类的对象实例，这样就使多态变得很灵活了

接口类型的引用指向接口某一个实现类的对象

接口的意义：

1，接口可以实现多继承。

2，用接口可以实现混合类型（主类型，副类型），java 中可以通过接口分出主次类型。主类型使用继承，副类型，使用接口实现。

3，接口进一步深化了标准的思想，接口本身就是一个标准，

他起到了降低耦合性的作用，接口可以使方法的定义和实现相分离，也就是将接口的定义者和实现者相分离，

接口也可以用于降低模块间或系统间的耦合性。

针对接口编程可以屏蔽不同实现间的差异，看到的只是实现好的功能。

练习：

在原有的雇员练习上修改代码：

公司会给 SalariedEmployee 每月另外发送 2000 元加班费

给 BasePlueSalesEmployee 发放 1000 元加班费

改写原有的代码，加入以上的逻辑

并写一个方法，打印出本月公司总共发放了多少加班费

练习：

验证歌德巴赫猜想,输入一个大于 6 的偶数,请输出这个偶数能被分解为哪两个质数的和

如 10=3+7　　12=5+7

质数:除了 1 和自身,不能被任何数整除的数

要求:两个人一组合作完成,一个人负责拆数,另一个人负责写方法,判断一个数是不是质数

接口：定义标准，
接口的实现：实现标准
接口的调用者：标准的使用
针对接口编程原则，也就是按照标准实现。

接口的定义者定义好了标准，接口的使用者就可以写使用代码，接口的实现者写好实现之后把实现对象传入接口的使用者中。
他调用接口中方法也就是掉用接口实现中的方法。
这种过程叫做接口的回调（先有标准使用者，后有标准实现者，把标准的实现者对象传给使用者，再由使用者通过接口调用标准实现者的方法）。

作业：

1、体验接口回调

根据文档和给定的 class 文件,实现接口,并且将实现类的对象作为参数调用 Match 类中的 match 方法,实现石头剪刀布的游戏

2、接口

为 SavingAccount 和 CreditAccount 各自添加一个子类
LoanSavingAccount 类:用户可以贷款,不可以透支
LoanCreditAccount 类:用户可以贷款,可以透支

说明:贷款和透支是不一样的,透支指的是账户余额小于 0,而贷款用户需要一个贷款额的属性.
在 ATM 机上,用户可以选择贷款,也可以选择还贷款,而还贷款就是要把账户余额上的资金转到贷款额上
例如:用户余额 10000 元,贷款额 100000 元,用户可以选择还款 5000 元,则用户余额变为 5000,贷款额变为 95000 元.

利用接口来抽象出 LoanSavingAccount 类和 LoanCreditAccount 类的共性
接口中的方法:
requestLoan:贷款
payLoan:还贷
getLoan:获取用户贷款总额

为 Bank 类添加三个方法,
贷款:参数 id,贷款额,返回修改过的 Account 对象
还贷款:参数 id,还款额,返回修改过的 Account 对象
统计所有账户贷款的总数

Java 第八天　　5 月 8 日

一、复习
接口

是个特殊的抽象类，属性：公开静态常量，方法：公开抽象方法
没有构造方法
接口之间可以多继承，一个类在继承另外一个类的同时，和可以实现多个接口

优点：
1、实现多继承，不会破坏类之间的单继承简单的树状关系。区分主类型和次要类型。
2、标准，解耦合工具
　　标准的使用者和标准的实现者通过借口隔离开，使得接口实现者的改变对使用者没有影响

接口的回调：
有了接口之后，先有接口使用者，后有接口的实现者，把接口实现者对象传给接口使用者，接口使用者通过接口，调用接口实现这中的方法

二、Object
java 中所有的类的父类或直接或间接的或隐含的都是 Object 类。
java 不允许循环继承，也就是互相继承是不可以的。

主要方法：
（１）finalize()：对象被垃圾收集的时候最后调用的方法
　　不能把释放资源的代码写在其中，程序员不能控制调用时机
（２）equals(): 对象内容的比较
　　Object 类中的 boolean equals(Object o)方法是用来比较对象的内容是否相等，其返回值是 boolean 类型的值，相同为真，不同则为假。
　　实际上还是比较对象地址是否相同。String 类覆盖了 equals()方法，他比较是对象中的内容是否相同。
　　子类中也推荐覆盖 Object 类中继承的 equals()方法，自己制定比较规则

　　自反性：s.equals(s)　true
　　对称性：s1.equals(s2) true
　　　　s2.equals(s1) true
　　传递性：s1.equals(s2) true
　　　　s2.equals(s3) true
　　　　则 s1.equals(s3) true
　　覆盖 equals()方法的步骤

```
boolean equals(Object o){
    if(this==o) return true;//1,看看是不是一个对象
    if(o==null) return true;//2,看看对象是不是空
    if(!(o instanceof 本类类名)) return false；//看看是不是本类对象
    ……//根据本类设计。
}
```

（３）toString()：返回对象的字符串表现形式
　　Object 类中的 toString()方法他返回的是类名加上他的地址的一个字符串。在子类中推荐覆盖 toString()方法。
　　System.out.println(person);实际上打印的是 person 对象 toString 方法的返回值.

练习：
Employe 类 属性：name,age,salary
把 equals().toString()方法覆盖

三、封装类

JAVA 为每一个简单数据类型提供了一个封装类，使每个简单数据类型可以被 Object 来装载。

除了 int（Integer）和 char（Character），其余类型首字母大写即成封装类类型名。

转换字符的方式：

```
int I=10;
String s=I+” ”;
String s1=String.valueOf(i);

Int I=10;
Interger I_class=new integer(I);
```

封装类.字符串.基本类型

```
int--------------------(Integer(x.toString))----------------->Integer
String -----------------(Integer.valueOf() )----------------->Integer
Integer-----------------(x.toString() )---------------------->String
int----------------------(100+””)------------------------->String
String------------------(Integer.parseInt() )--------------->int
Integer-----------------(Integer.intValue() )--------------->int
```

四、异常

帮错误发生的时候减少损失，提高容错性

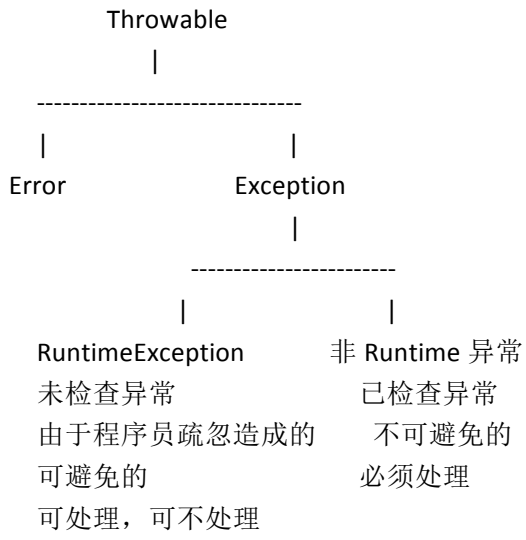
1、常的分类

Throwable 有两个子类：Error 和 Exception。

一个 Error 对象表示一个程序错误，指的是底层的、低级的、不可恢复的严重错误。此时程序一定会退出，因为已经失去了运行所必须的物理环境。

对于 Error 错误我们无法进行处理，因为我们是通过程序来应对错误，可是程序已经退出了。

我们可以处理的 Throwable 类中只有 Exception 类的对象（例外/异常）。



2、异常的产生和传递

throw new 一个异常对象; ---表示抛出一个异常
throw new NullPointerException();
相当于 return，函数返回上一级

传递：
沿着方法调用链反向传递！
当一个方法中出现异常，而没有作处理，则以异常对象为返回值返回调用处（逐级传递）
异常返回给虚拟机时，虚拟机终止退出，程序结束

3、异常的处理
（1）声明抛出
是方法声明的第五部分 throws+异常名字（多个异常用“,”分隔）
出现异常，不处理，抛给上一级处理
并且子类抛出异常的范围不能比父类抛出异常的范围更宽。
（2）捕获异常
try - catch
try - catch - finally
try - finally //不捕获异常，当异常发生，返回上一级之前，要运行 finally 中的代码
以上语句可以嵌套

```
返回类型 方法名（参数）{
 try{
      可能出错语句
      正常语句
 }catch(异常类 e) {  //某种异常的引用
      对异常的处理
 }
 正常语句
}
```

捕获多个异常：
程序任何时刻只发生一个异常
可对产生的每个异常分别捕捉，也可由同一异常进行处理，前提是这个共用的异常应该是所有这些该被捕获的异常的父类,但是，对于非受查异常不成立
当 try 后面有多个语句块时，注意 catch 异常的顺序，子类必须放在父类的前面

finally 关键字
无论异常是否发生，一定会执行的代码，可放在 finally 块内。
要点：没有异常产生时：正常执行 try{}catch(){} —> 进入 finally 语句块 —> 方法中剩余代码
有异常产生时（捕找到） —> 进入 catch 处理 —> 进入 finally 语句块 —> 方法中剩余代码
有异常产生时（没捕找到）—> 进入 finally 语句块 —> 离开方法
一般写一些释放资源的代码
在 try - catch 块中遇到 System.exit(0);则不会执行 finally 中的代码

Throwable 有一个 message 属性。在使用 catch 的时候可以调用：
Catch(IOException e){System.out.println(e.getMessage())}; //打印出来的是创建(throw new)异常对象的时候，给定的参数

Catch(IOException e){e.printStackTrace()};　　//打印堆栈追踪信息

以上两条语句都是可以打印出错的过程信息。告诉我们出错类型所历经的过程，在调试的中非常有用。

开发中的两个道理：

①如何控制 try 的范围：根据操作的连动性和相关性，如果前面的程序代码块抛出的错误影响了后面程序 代码的运行，那么这个我们就说这两个程序代码存在关联，应该放在同一个 try 中。

②对已经查出来的例外，有 throw(消极)和 try catch（积极）两种处理方法。

对于 throws 把异常抛到 try catch 能够很好地处理异常的位置（即放在具备对异常进行处理的能力的位置 ）。如果没有处理能力就继续上抛。

4、自定义异常

（1）继承 Exception 类

（2）构造方法：

不带参数的构造方法

带参数的构造方法：参数指出错误性质，super(message)；把参数传递给父类构造异常

作业

（1）为 Account 类及其子类添加 toString 方法和 equals 方法

（2）(Exception)

为 BAM 添加几个异常类

BalanceNotEnoughException :用于取钱的时候余额不足的情况(包括账户余额超过透支额的情况)

RegisterException:用于开户异常的情况,例如密码两次输入不一致等情况

LoginException:用户登录异常的情况,例如 id 错误,密码错误

LoanException:贷款额不能为负数,如果用户试图将贷款额置为负数,则会抛出这个异常

以上四个异常类有一个共同的父类 BusinessException

并妥善的处理这些异常

Java 第九天　　5 月 9 日

一、复习

1、Object 所有的类直接或间接的父类

finalize:垃圾收集的时候调用

toString:返回对象的字符串形式

equals:判断两个对象内容是否相同

2、包装类

8 种基本类型各自提供了对象形式

3、异常处理

提高容错性

异常的分类：Throwable

Error　　:错误，不可避免，不可处理

Exception:Runtime：未检查：可处理可不处理，首先应该努力避免异常（本可避免的）

非 Runtime：已检查：必须要处理

异常对象的产生：throw 抛出异常，沿着方法调用链反向传递

异常处理：throws 消极，声明抛出

try-catch 捕获异常

try-catch-finally

try-finally

方法覆盖 ：子类覆盖方法不能比父类被覆盖方法抛出更多的，范围更宽的异常

4、自定义异常

二、断言

在 JDK1.4 之后开始出现，是一个调试工具

其后跟的是布尔类型的表达式，如果表达式结果为真不影响程序运行。如果为假系统出现低级错误(Error)，在屏幕上出现 assert 信息。

assert a%2==0;

编译器的默认设置：把断言语句忽律

javac -source 1.4 源文件名 //表示用 1.4 新特性编译

java -enableassert(-ea) 类名 //打开断言功能

assert a%2==0:"a 必须是偶数"; //出现错误时的提示信息

assert 只是用于调试。在产品编译完成后上线 assert 代码就被删除了。

三、内部类

内部类也就是定义在类内部的类。是编译时语法。

内部类的分类：
成员内部类、
局部内部类、
静态内部类、
匿名内部类（图形监听时要用到，要掌握）。

1、成员内部类

四个访问权限修饰符都可以修饰成员内部类。
内部类和外部类在编译时是不同的两个类，内部类对外部类没有任何依赖。
内部类是一种编译时语法，在编译时生成的各自的字节码文件(Outer.class 和 Outer$Inner.class)，内部类和外部类没有关系。
内部类中可以访问外部类的私有成员。（与 C++的友员相比，不破坏封装）
作为外部类的一个成员存在，与外部类的属性、方法并列。
内部类和外部类的实例变量可以共存。
在内部类中访问实例变量：this.属性
在内部类访问外部类的实例变量：外部类名.this.属性。
在外部类的外部访问内部类，使用 out.inner.

成员内部类的特点：

（1）内部类作为外部类的成员，可以访问外部类的私有成员或属性。（即使将外部类声明为 private，但是对于处于其内部的内部类还是可见的。）
（2）用内部类定义在外部类中不可访问的属性。这样就在外部类中实现了比外部类的 private 还要小的访问权限。

注意：内部类是一个编译时的概念，一旦编译成功，就会成为完全不同的两类。

对于一个名为 outer 的外部类和其内部定义的名为 inner 的内部类。编译完成后出现 outer.class 和 outer$inner.class 两类。

（3）成员内部类不能含有静态成员。***

建立内部类对象时应注意：
在外部类的内部可以直接使用 inner s=new inner();（因为外部类知道 inner 是哪个类，所以可以生成对象。）
而在外部类的外部，要生成（new）一个内部类对象，需要首先建立一个外部类对象（外部类可用），然后在生成一个内部类对象。内部类的类名是外部类类名.内部类类名。
Outer o=new Outer();
Outer.Inner in=o.new.Inner()。

2、静态内部类

（注意：前三种内部类与变量类似，所以可以对照参考变量）
静态内部类定义在类中，任何方法外，用 static class 定义。
静态内部类只能访问外部类的静态成员。
生成（new）一个静态内部类对象不需要外部类对象：这是静态内部类和成员内部类的区别。
静态内部类的对象可以直接生成：
Outer.Inner in=new Outer.Inner()；
而不需要通过生成外部类对象来生成。这样实际上使静态内部类成为了一个顶级类。
静态内部类不可用 private 来进行定义。

3、局部内部类

在方法中定义的内部类称为局部内部类。
与局部变量类似，在局部内部类前不加修饰符 public 和 private，其范围为定义它的代码块。

注意：
局部内部类不仅可以访问外部类私有实例变量，还可以访问外部类的局部常量（也就是局部变量必须为 final 的）
在类外不可直接访问局部内部类（保证局部内部类对外是不可见的）。
在方法中才能调用其局部内部类。
通过内部类和接口达到一个强制的弱耦合，用局部内部类来实现接口，并在方法中返回接口类型，使局部内部类不可见，屏蔽实现类的可见性。

4、匿名内部类

匿名内部类是一种特殊的局部内部类，它是通过匿名类实现接口，并只创建一次。

匿名内部类的特点：
（1）一个类用于继承其他类或是实现接口，并不需要增加额外的方法，只是对继承方法的事先或是覆盖。
（2）只是为了获得一个对象实例，不许要知道其实际类型。
（3）类名没有意义，也就是不需要使用到。

注：一个匿名内部类一定是在 new 的后面，用其隐含实现一个接口或实现一个类，没有类名，根据多态，我们使用其父类名。
因其为局部内部类，那么局部内部类的所有限制都对其生效。
匿名内部类是唯一一种无构造方法类。
大部分匿名内部类是用于接口回调用的。
匿名内部类在编译的时候由系统自动起名 Out$1.class。

如果一个对象编译时的类型是接口，那么其运行的类型为实现这个接口的类。
因匿名内部类无构造方法，所以其使用范围非常的有限。
当需要多个对象时使用局部内部类，因此局部内部类的应用相对比较多。匿名内部类中不能定义构造方法。

匿名内部类的写法：

```
interface A{
    void ia();
}
class B{
    public A bc(){
        return new A(){          //匿名类实现了 A 接口
            void ia(){……}
        };
        }
}
```

使用匿名内部类：

```
B b=new B();
A a=b.bc();
a.ia();
```

注意：当类与接口（或者是接口与接口）发生方法命名冲突的时候，此时必须使用内部类来实现。
用接口不能完全地实现多继承，用接口配合内部类才能实现真正的多继承。

对于两个类，拥有相同的方法：

```
class People
{
   run();
}
interface Machine{
    run();
}
```

此时有一个 robot 类：

```
class Robot extends People implement Machine.
```

名为 run()的方法有 2 个，不可直接实现。

```
interface Machine{
    void run();
}
class Person{
    void run(){System.out.println("run");}
}
class Robot extends People{
    class Heart implements Machine{
        public void run(){
            System.out.println("发动机运行");
        }
    }
```

```
    public Machine getHeart(){
        return new Heart();
    }
    public void run(){
        System.out.println("机器人跑");
    }
}
public class Test{
    public static void main(String[] args){
                Robot r=new Robot();
        r.run();
        r.getHeart().run();
    }
}
```

练习：把前面石头剪刀布的游戏改写为:采用匿名内部类来实现接口,并获得对象去调用 match 方法

四、集合

集合（集合类的对象）是用来管理其他若干对象的。
集合中保存的是对象的引用，数组是最基本的集合

集合框架

1，接口
集合中用到的类，接口在 java.util 包中，在使用时注意将其引入 import。

Collection 用来管理多个对象，集合中的每个元素都是对象。
1)List 一个 List 的实现类的对象在管理多个对象时会按顺序组织对象（即按照将对象放入的顺序存储）
List 实现类的对象是有顺序的，List 实现类对象中的内容是可重复的。（注意，顺序和排序的区别）
2)Set 一个 Set 的实现类表示一个数学概念上的集合，Set 的实现类的对象中的元素是无顺序的，也就是不会按照输入顺序来存放，Set 的实现类对象中的元素是不重复的。
3)SortedSet,他是 Set 的子接口，他的实现类会对集合中的元素进行排序。但是要指定排序规则，他会按排序规则进行排序。

Map，Map 中没有对象，而是键值对，由 Key，value 组成的键值对
Key 是没有顺序，不可重复的。
value 是可以相同的,一个 Key 和一个 value 一一对应。
Map 接口（以下介绍其子接口）
SortedMap，这个接口的实现类同样可以实现，不过是对键值对中的 Key 进行排序，这个接口的实现类也是要指定排序规则的。

1> ArrayList 是接近于功能的集合类，ArryList 的实质就是一个会自动增长的数组，ArrayList 是用封装的数组来实现的 List 接口的。
Collection 的实现类对象的遍历方式是用迭代来实现的。
在使用迭代器时先要获得一个迭代器的对象，Iterator（迭代器接口）这是一个接口，迭代器是在集合类中实现的，也就是说，他是一个内部类（匿名内部类）实现的。

Iterator 接口中定义的常用方法方法 hasNext()，next()。

hasNext(),这个方法会使用一个游标，并通过判断游标指向的位置是否存放有对象。

next()方法也是 Iterator 接口中定义好的方法，这个方法会使游标指向下一个元素的位置，游标会跳过第一个元素，并返回其中的内容。

Collections 这是一个工具类，也是 java.util 包中的，这个类中的 sort(list 接口的实现类的对象)方法，其参数是一个集合类的对象，这个方法使用来对集合类的对象进行排序的。以后，我将以集合这个名字来称呼集合类的对象。，对于字符串对象内容的集合来说会按字典顺序排序（升序），对于数字内容的集合排序也会按照升序排序。

排序可分为两部分内容，一个是排序的规则，也就是按照什么来进行排序，并且排成什么样的顺序。
第二个就是排序的算法，他决定了排序的效率。

在对自定义的集合内容排序时，需要先定义那个类型的排序规则。
Comparable 接口，这个接口中只定义了一个 compareTo(Object o)，方法的返回类型是整型，如果当前对象大于参数对象就返回正数，当前对象等于参数对象是就返回 0，当前对象小于参数对象时就返回负值，这样写就是升序排列，反之则是进行降序排列，在实现这个接口中的方法时，返回值定义方式，只有这两种。

根据指定类型的排序规则实现了 Comparable 接口，那么就可以对存有这个类型的集合进行整体排序。Comparable 接口，也叫做可比较接口。这个接口在 java.lang 包下。只要实现了这个接口，就是可排序的。

接下来介绍另外一种对自定义类型对象的集合整体排序的方法，也就是实现比较器接口（Comparator），这个接口中定义了一个 compare(Object o1，Object o2)方法来比较两个对象，这个方法的返回值定义和上面介绍的那个方法是一样。

注意：在 API，帮助文档中以上两个方法的参数类型是 T，这代表的模板类型，也就是集合中存放的内容的类型，在 JDK1.4 中其参数就是 Object 类型，模板类型的详细内容会在最后的 JDK5.0 新特性中讲到。

Comparator 接口可以在匿名内部类中实现，Collections 中的 sort(集合了的对象，比较器)方法，可以对自定义类型内容的集合进行整体排序。

2> LinkedList，它是 List 接口的实现类，其底层是用双向循环链表来实现的。

注意：ArrayList 的查询效率比较高，增删动作的效率比较差，适用于查询比较频繁，增删动作较少的元素管理的集合。

LinkedList 的查询效率低，但是增删效率很高。适用于增删动作的比较频繁，查询次数较少的元素管理集合。

ArrayList，LinkedList 都是线程不安全的。

实现堆栈 1，数组（ArrayList，增删效率比较低，不适合）

2，LinkedList（实现堆栈的好方法）

3，java.util.Stack 类，Stack 是 Vector 的子类，Vector 类是一个线程安全的（是一个重量级的类），并继承了 Vector 的方法，Verctor 类（这个类也是 List 接口的实现类）和 ArrayList 的功能近乎相同。（不推荐使用 Stack 类来实现堆栈）。

Java 第十天　　5 月 10 日

一、复习内部类

1、成员内部类

可以访问外部类的私有成员，外部类类名.this.属性

构造成员内部类对象，必须先构造一个外部类对象，外部类对象.new 构造内部类对象

2、静态内部类

只能访问外部类的静态成员

构造静态内部类对象时，不再需要构造外部类对象

3、局部内部类

在外部类方法内定义的内部类

不仅能访问外部类的私有成员，而且还能访问外部类的局部变量，但是要求局部变量是 final 的

4、匿名内部类

局部内部类，用于实现一个借口或者继承一个类，只会构造一次

内部类的作用：

访问外部类的私有成员，不破坏封装。可以给编程带来一些方便

我们可以把接口公开，把接口的实现类以内部类的形式隐藏起来。强制用户通过接口来实现弱耦合

接口＋内部类实现多继承

二、List 接口的实现类

1、ArrayList

底层使用数组实现

2、Vector

ArrayList 轻量级 快 线程不安全

Vector 重量级 慢 线程安全的

3、LinkedList

底层使用双向循环链表实现

ArrayList 数组 查询快 增删操作慢

LinkedList 链表 查询慢 增删操作快 栈

使用组合复用实现栈

三、Set 接口

1、HashSet

Set 的实现类的集合对象中不能够有重复元素，HashSet 也一样是使用了一种标识来确定元素的不重复，是元素内容不重复

HashSet 用一种算法来保证集合中的元素是不重复的，HashSet 的底层实现还是数组。

Object 类中的 hashCode()的方法是所有子类都会继承这个方法，这个方法会用 Hash 算法算出一个 Hash（哈希）码值返回，HashSet 会用 Hash 码值去和数组长度取模，

模（这个模就是对象要存放在数组中的位置）相同时才会判断数组中的元素和要加入的对象的内容是否相同，如果不同才会添加进去。

Hash 算法是一种散列算法。

注意：所有要存入 HashSet 的集合对象中的自定义类必须覆盖 hashCode(),equals()两个方法，才能保证集合中元素容不重复。

在覆盖 hashCode()方法时，要使相同对象的 hashCode()方法返回相同值，覆盖 equals()方法再判断其内容。为了保证效率，所以在覆盖 hashCode()方法时，也要尽量使不同对象尽量返回不同的 Hash 码值。

如果数组中的元素和要加入的对象的 hashCode()返回了相同的 Hash 值（相同对象返回相同整数）,才会用 equals()方法来判断两个对象的内容是否相同（不同对象返回不同整数）。

练习：
把若干 Employee 对象放在 Set 中并遍历,要求没有重复元素

2、SortedSet 接口是 Set 的子接口。
TreeSet 是 SortedSet 接口的实现类，他可以对集合中的元素进行排序。
要存放在 TreeSet 中自定义类的对象，这个类要么是已经实现了 Comparable 接口，要么是能给出 Comparator 比较器，
TreeSet 可以自动过滤掉重复元素所以不用重载 hashCode()方法，TreeSet 会根据比较规则判断元素内容是否相同，TreeSet 会在元素存入时就进行了排序。
判断对象重复的依据：compareTo()方法的返回值为 0，就是重复元素
（在 TreeSet 给出排序规则时，一定要注意对象内容相等的条件，一定要注意在主观的认为两个对象内容相同时，才可以使用比较少的条件来进行判断）

在要排序时才使用 TreeSet 类（存储效率比较低），HashSet 的存储效率比较高，在需要为 HashSet 的对象排序时，就可以把 HashSet 中的元素放入 TreeSet。

四、Map
Map 中只可以存放键值对（Key，value），其中 Key 是不可以重复的。Key 和 value 是一一对应的。

HashMap，是 Map 接口的实现类，Key 时无序存放的，其中 Key 是不可以重复的，它也是通过 Hash 码值来保证 Key 不重复的，Key 和 value 是一一对应的。
如果要加入的键值对和 HashMap 中键值对的 Key 是相同的就会将这个集合中的 Key 所队应的 value 值进行覆盖，在使用自定义类型作为 Key 时，那就是要覆盖 hashCode(),equals()方法，也就是和 HashSet 中要放入自定义类型是的处理方法相同。
这个类的对象是线程不安全的。

遍历：（1）vlaues() 返回所有值（value）的集合，是一个 Collection
（2）keySet（） 返回所有键对象的集合，是一个 Set
过遍历这个 Set，用 get()方法来获得 Key 所对应的 value，也就遍历了 Map。

Hashtable，也是 Map 接口的实现类，他和 HashMap 比较相似，只不过这个类对象是重量级的，也是线程安全的。他不允许 Key 和 value 为 null。

Properties，这个类是 Hashtable 的子类，他的 Key 和 value 只能是字符串。

SortedMap 是 Map 的子接口
TreeMap，是 SortedMap 的实现类，他会按照 Key 进行排序。和 TreeSet 类一样，在使用自定义类作 Key 时，要用自定义类实现 Comparable 接口。

练习：
达内希望在学生毕业的时候统计出学生在校期间考试成绩的排名,写一个 Student 类,其中用集合来管理每个学生的各个科目的考试成绩,
将多个 Student 对象放在集合中,打印出学生的总分以及排名

(集合)
改写 Bank 类,采用集合的方式来管理多个 Account 对象
为 Bank 类添加一个方法
打印所有用户的总资产排名
说明:一个用户可能会有多个账号,以身份证号为准.总资产指多个账户余额的总和,不需要考虑贷款账户的贷款额

考试系统
Exam 类　考试类
属性：　若干学生　一张考卷
提示：学生采用 HashSet 存放

Paper 类　考卷类
属性：若干试题
提示：试题采用 HashMap 存放，key 为 String，表示题号，value 为试题对象

Student 类　　学生类
属性：姓名　一张答卷　一张考卷

Question 类　　试题类
属性：题号 题目描述　　若干选项　　正确答案(多选)
提示：若干选项用 ArrayList

AnswerSheet 类　　答卷类
属性：每道题的答案
提示：答卷中每道题的答案用 HashMap 存放，key 为 String，表示题号，value 为学生的答案

问题：为 Exam 类添加一个方法，用来为所有学生判卷，并打印成绩排名（名次、姓名）

Java 第十一天　　5 月 11 日

一、复习
集合：用一个对象储存管理多个对象

Collection：元素都是对象
遍历：迭代遍历

List：元素有顺序，可以重复
　　遍历：还可以用 for 循环（下标）
　　排序：Collections.sort（list）
实现类：
ArrayList：底层数组实现，查询快，而增删慢；轻量级，线程不安全
Vector：底层数组实现，重量级，线程安全
LinkedList：底层链表实现，查询慢，增删快
在 Java 中用 LinkedList 实现一个栈，不用数组，因为栈的主要功能就是增删，数组慢；不用 Vector，因为效率低

Set：元素无序，元素内容不重复
SortedSet：按照各种排序规则给 Set 排序

实现类：
HashSet：采用哈希算法保证元素不重复，覆盖 hashCode()保证哈希码相同，equals()保证 true
TreeSet：元素一定要实现了 Comparable 接口的
根据排序规则，compareTo()返回 0，说明是重复元素

Map：元素是键值对
key 无序，不重复 value 可以重复
SortedMap：按照 key 排序
遍历：values() 遍历所有的值对象
keySet() 遍历所有的键对象
实现类：
HashMap：线程不安全 ， 允许 null 作为 key 和 value
Hashtable：线程安全，不允许
TreeSet：SortedSet 的实现类

二、java 中的图形界面

GUI，图形化的用户接口，为了人机交互使用的。

BS 与 CS 的联系与区别。
C/S 是 Client/Server 的缩写。服务器通常采用高性能的 PC、工作站或小型机，并采用大型数据库系统，如 Oracle、Sybase、Informix 或 SQL Server。客户端需要安装专用的客户端软件。
B/S 是 Brower/Server 的缩写，客户机上只要安装一个浏览器（Browser），如 Netscape Navigator 或 Internet Explorer，服务器安装 Oracle、Sybase、Informix 或 SQL Server 等数据库。
在这种结构下，用户界面完全通过 WWW 浏览器实现，一部分事务逻辑在前端实现，但是主要事务逻辑在服务器端实现。浏览器通过 Web Server 同数据库进行数据交互。
C/S 与 B/S 区别：
１．硬件环境不同:
C/S 一般建立在专用的网络上，小范围里的网络环境，局域网之间再通过专门服务器提供连接和数据交换服务.
B/S 建立在广域网之上的，不必是专门的网络硬件环境,例与电话上网，租用设备．信息自己管理．有比 C/S 更强的适应范围，一般只要有操作系统和浏览器就行
２．对安全要求不同
C/S 一般面向相对固定的用户群，对信息安全的控制能力很强．一般高度机密的信息系统采用 C/S 结构适宜．可以通过 B/S 发布部分可公开信息.
B/S 建立在广域网之上，对安全的控制能力相对弱，可能面向不可知的用户。
３．对程序架构不同
C/S 程序可以更加注重流程，可以对权限多层次校验，对系统运行速度可以较少考虑.
B/S 对安全以及访问速度的多重的考虑，建立在需要更加优化的基础之上．比 C/S 有更高的要求 B/S 结构的程序架构是发展的趋势，从 MS 的.Net 系列的 BizTalk 2000 Exchange 2000 等,
全面支持网络的构件搭建的系统. SUN 和 IBM 推的 JavaBean 构件技术等,使 B/S 更加成熟.
４．软件重用不同
C/S 程序可以不可避免的整体性考虑，构件的重用性不如在 B/S 要求下的构件的重用性好.
B/S 对的多重结构,要求构件相对独立的功能．能够相对较好的重用.就入买来的餐桌可以再利用,而不是做在墙上的石头桌子
５．系统维护不同?

C/S 程序由于整体性，必须整体考察，处理出现的问题以及系统升级．升级难．可能是再做一个全新的系统

B/S 构件组成,方面构件个别的更换,实现系统的无缝升级．系统维护开销减到最小.用户从网上自己下载安装就可以实现升级.

６．处理问题不同

C/S 程序可以处理用户面固定，并且在相同区域，安全要求高需求，与操作系统相关．应该都是相同的系统

B/S 建立在广域网上，面向不同的用户群，分散地域，这是 C/S 无法作到的．与操作系统平台关系最小.

７．用户接口不同

C/S 多是建立的 Window 平台上,表现方法有限,对程序员普遍要求较高

B/S 建立在浏览器上，有更加丰富和生动的表现方式与用户交流．并且大部分难度减低,减低开发成本.

８．信息流不同

C/S 程序一般是典型的中央集权的机械式处理，交互性相对低

B/S 信息流向可变化, B-B B-C B-G 等信息、流向的变化，更像交易中心。

构造图形界面的步骤

1，选择一个容器

2，设置容器的布局管理器

3，向容器添加组件

4，事件的监听

容器（Container）用于管理其他的组件的对象。组件必须放到容器里。

JFrame，这是一个最顶层的窗体容器，所有其他的组件必须放在顶层容器里。

```
JFrame frame = new JFrame("Hello Swing");  //创建窗体，字符串为窗体的标题
frame.setSize(500,300);  //设置窗口大小，500 像素长，300 像素高
frame.setVisible(true);  //设置可见性
```

JPanel，他不是顶层容器，必须放在顶层容器中，任何一个容器都有 add()方法，Panel 面板是透明的（默认）。他也是一个组件。

JDialog 对话框容器，他要依附于其父组件，他不是一个顶层容器。

布局管理：对于任何一个容器类中都有 setLayout()方法，用容器对象调用这个方法，来设置容器的布局管理器（LayoutManager 这是一个接口，所有布局管理器都实现了这个接口）。

可用的布局管理器：

所有的布局管理器实现了一个接口 java.awt.LayoutManager

FlowLayout，流式布局管。尝试在一行中按增加顺序摆放组件，窗体大小改变时，组件位置会相应发生改变

Panel 的默认布局管理就是 FlowLayout。

```
FlowLayout flow = new FlowLayout();
 frame.setLayout(flow);
```

BorderLayout，按方位进行布局管理，（North，South，East，West，Middle）不明确指定，就会默认加载在中间（Middle），每个部分只能放一个组件

frame.add(Component comp,String place)；这个方法是在指定的位置添加组件。

JFrame 的默认布局管理器

GridLayout，网格布局，通过行列，间距，来用网格分割，把组件放入如网格中，先行后列摆放组件。可以保证每个组件的大小都是一样的

```
frame.setLayout(new GirdLayout(3,2));  //把容器平均的分为 3 行 2 列，先左后右，先上到下的顺序排列
```

CardLayout，卡片布局，组件重叠放置。

GridBagLayout，组件可以跨行跨列的网格布局。

*** 注意：一定要在图形界面都其他功能都设置好之后才能设置可见性。

JButton ：按钮
JTextField：单行文本域
JTextArea：多行文本区
JPasswordField：密码输入框
JScrollPane：滚动窗体　使用一个多行文本域作为参数创建滚动窗体
JComboBox：下拉选择框

JRadioButton:单选按钮
JCheckBox：多选按钮
JList:多行列表
JLabel：标签
JEditorPane：显示结构化文档
Border:边框

JMenuBar：菜单条
JMenu：菜单
JMenuItem：菜单项
JPopupMenu：弹出式菜单

JSlider：滑动条
JProgressBar：进度条
JTabbedPane：分层面板
JSplitPane：分隔面板
JToolBar：工具条

JFileChooser：文件选择器
JColorChooser：颜色选择器

显示对话框
JoptionPane 里面有很多静态方法可以弹出对话框

注意：具体的方法可以去参看 Java2 SE 的 API 文档。

三、awt 事件模型（观察者模式）（重点）

事件模型中，包括事件源对象，事件处理者（事件监听者对象），事件对象。

事件源和事件处理者之间建立了授权注册关系，也就是在事件源类中有一个事件处理者的对象作为属性，也可能是一个事件处理者的集合。

事件对象
事件源————————〉事件处理者

这就是事件模型的机制，也就是由事件源对象发送一个消息（事件对象），然后事件处理者调用相应的方法处理事件。

在事件监听器接口中定义的方法，都要以事件对象为参数。

*** 一个事件源可以注册多个同类型的监听器，也可以注册多种多个事件监听器，
一个事件监听器也可以为多个事件源服务。

事件对象继承自 EventObject 类，并可以通过 getSource()方法获得事件源对象，当然需要在构造事件对象时将事件源对象传入,来区分是哪个事件源发出的事件，所以要用事件对象作为参数。

事件源，事件对象，监听接口，在 java.awt 包中提供了很多已经定义好的，只需要实现监听接口就好了。

什么是发消息：
A，B，C 三个类，分别作为事件源，事件处理者，事件对象。
在 A 类中有一个 B 类的属性或者是一个内容为 B 类对象的集合，也就是事件源和事件处理者之间的建立了授权关系，
在 B 类需要实现一个自定义的接口，这个自定义的接口继承了 EventListener，EventListener 接口中没有定义任何方法，这只是一个标记接口。
实现在自定义接口中定义好的用于事件处理的方法，C 类要继承 EventObject 类。
这些方法是以事件对象为参数的 b(C c)，而后在 A 类 a(C c)方法中使用 B 类的对象调用 B 类中的 b(C c)方法，
并把事件对象作为参数，并在 main 方法中用 A 类的对象调用了 a(c)方法，这也就叫做 A 类对象给 B 类发送了消息。

也就是说事件源对象间接调用了事件监听器的方法，并以事件对象为实参传到事件监听器的方法中，要就叫事件源给事件监听器的方法发了一个消息（事件对象）。

例子如下：

```
import java.util.*;
//事件源类
class A{
	private String test;
	private List li=new ArrayList();
	public A(String test){
		this.test=test;
	}
	public String getTest(){return this.test;}
	public void addB(B b){
		this.li.add(b);
	}
	public void removeB(B b){
		this.li.remove(b);
	}
	   /*
	   * 所谓的事件源给事件监听器，发送事件对象。
	   * 其实就是事件源用事件为参数，调用时间监听器的相应方法
	   */
	public void fire(){
		C c=new C(this);
```

```
            Iterator it=li.iterator();
            while(it.hasNext()){
                B b=(B)it.next();
                b.b(c);
            }
        }
}
//事件监听器的接口，要继承 EventListener 标记接口
//监听接口中的每一个方法，都应该以对应的时间对象作为参数
interface Blistener extends EventListener{
        void b(C c);
}
//事件监听器，实现接口
class B implements Blistener{
        public void b(C c){
            A a=(A)c.getSource();
            System.out.println(a.getTest()+" "+c.getMessage());
        }
}
/*
* 事件对象类
* 事件对象类中要封装事件源对象
*/
class C extends EventObject{
        private String message;
        public C(Object src){
            super(src);
        }
        public void setMessage(String message){
            this.message=message;
        }
        public String getMessage(){return this.message;}
}
public class Test{
        public static void main(String[] args){
            A a1=new A("Event");
            B b1=new B();
            c1.setMessage("Test");
            a1.addB(b1);//注册监听器
            a1.fire();//发送事件
        }
}
```

以上代码只是事例,在引入包之后可以运行。

在 Java 的图形编程中，所有动作（事件）都已经提供了相应的事件对象和事件监听接口，例如：实现窗口的关闭按钮，点击关闭按钮会发出相应的事件对象，相应的调用监听器中实现好的方法。
相应的方法清参阅 Java2 SE API 帮助文档。

缺省适配模式，通过一个抽象类实现接口，抽象类中的接口方法实现，都是一个无意义的空实现，可以继承这个抽象类，只覆盖向覆盖的方法就可以了。

在 java.awt.event 包中，会有一些适配类，也就是把相应的 XXXListener，换成 XXXAdapter 就是适配类。
在 java.awt.event 包中的 ActionEvent 类，在以下操作中会发送这个事件，
1，JButton 组件，按钮被点击
2，JTextField 组件，在单行文本域中按 Enter 键。
3，JCheckBox 组件，选中了复选框。
4，JRadioButton 组件，选中了单选按钮。
5，JMenu 组件，选中菜单项。

作业：
写一个股市类作为事件源,事件源会随机产生波动,写两个监听类,一个会在股市跌的时候卖出,涨的时候买入,另一个投资逻辑刚好相反

Java 第十二天　　5 月 13 日

一、复习 AWT 事件模型（Observer 模式）
1、事件源
2、事件对象
3、事件监听器

事件源和监听器事先进行授权注册，当事件条件满足时，事件源会给注册的监听器发送一个事件对象，由事件监听器作出相应的处理。

一个事件源可以是多种事件的事件源
一个事件源就同一类事件可以注册多个监听器
一个监听器可以同时注册在多个事件源当中

事件源和监听器是独立，弱耦合的，是各司其职的

事件对象中会封装事件源对象
事件监听接口中的每一个方法都要以事件对象为参数
事件源中要保存和它有监听关系的监听器
事件源给事件监听器发送事件对象：事件源以事件对象作为参数，调用监听器接口的相应方法，通过回调，调用的是不同监听实现类的方法

二、
在 Java 的图形编程中，所有动作（事件）都已经提供了相应的事件对象和事件监听接口，
例如：实现窗口的关闭按钮，点击关闭按钮会发出相应的事件对象，相应的调用监听器中实现好的方法。
相应的方法清参阅 Java2 SE API 帮助文档。

缺省适配模式，通过一个抽象类实现接口，抽象类中的接口方法实现，都是一个无意义的空实现，可以继承这个抽象类，只覆盖向覆盖的方法就可以了。

在 java.awt.event 包中，会有一些适配类，也就是把相应的 XXXListener，换成 XXXAdapter 就是适配类。

适配类是抽象类，其中对接口 XXXListener 中的方法进行了空实现，实现这个类，覆盖对自己有用的方法

在 java.awt.event 包中的 ActionEvent 类，在以下操作中会发送这个事件，
1，JButton 组件，按钮被点击
2，JTextField 组件，在单行文本域中按 Enter 键。
3，JCheckBox 组件，选中了复选框。
4，JRadioButton 组件，选中了单选按钮。
5，JMenu 组件，选中菜单项。

添加事件监听：
1、实现监听接口
2、将监听器对象注册在组件(事件源)中

ActionEvent
事件源 --- 组件 JButton 按钮　　点击触发 ActionEvent
　　JTextField 单行文本域　输入内容以后回车触发 ActionEvent
jtf.getText();　//得到文本域中的内容

练习：
1、写一个图形界面,采用 BorderLayout 布局,中间的部分放置一个可以滚动不可编辑的 JTextArea,南面放置一个可以编辑的 JTextField,
但在 TextField 中输入文字并按下回车的时候,文字会添加到 TextArea 中

2、为 BAM 添加用户界面
需要以下几个类:

BAMClient 其中会包含一个 Frame,这是用户主界面
MainPanel:主界面,用户可以选择开户或者登录
RegisterPanel:用户开户具体用到的界面
LoginPanel:用户登录需要的界面
BusinessPanel:界面上会显示账户的功能 至少包括存款和取款,对于可透支的用户,还允许用户修改透支额度,对于贷款用户,还允许用户贷款和还贷款

注:本练习的界面布局不做要求,请阅读现有代码,添加事件处理代码
提示:在开户或者登录之后都会跳到 BusinessPanel,而用户点击了交易之后,界面停留在 BusinessPanel
要随时注意在 BusinessPanel 上根据数据的变化更新显示信息

三、多线程
C++的多进程是 OS 系统并发的一个任务
Java 中没有多进程，一个 JVM 就是一个进程

线程是在进程中并发的一个顺序的执行流程

多进程：划分时间片，宏观上并行，微观上串行
多线程：cpu 在进程内部再划分时间片

CPU ，代码 ，数据

进程：进程间数据独立
线程：数据空间共享，堆空间的共享（堆空间中存放的是对象），栈空间是独立的
所以线程间切换容易，称为轻量级进程

一个线程对象代表了一个线程，并非就是一个线程
线程是操作系统中负责维护的资源
java.lang.Thread 类的一个对象就代表一个线程
线程是底层 OS 维护的资源，JVM 跑在 OS 上，在 JVM 中创建一个 Thread 对象，调用其 start（）方法，底层 OS 会申请一个线程资源，线程对象可到底层管理一个线程
创建好线程之后，把要让线程执行的代码封装到线程对象中（覆盖 run()方法）

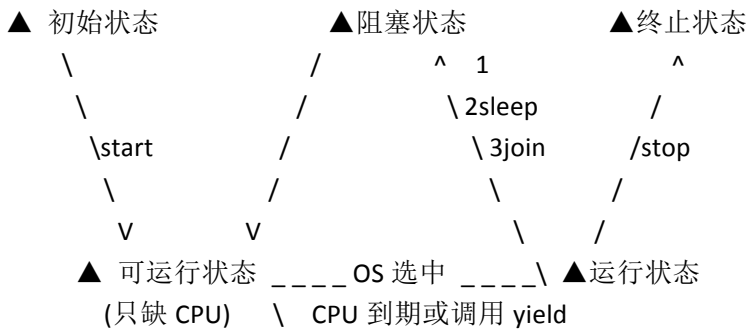
实现线程代码的方式：
1、继承 Thread 类，覆盖 run()方法
去底层申请线程并运行，对线程对象调 start()方法，main 方法是一个主线程
宏观并行，微观串行
2、实现 Runnable 接口
使用多态获得 Runnable 对象，成为目标对象
再利用目标对象构造线程对象 Thread t = new Thread(target);

四、多线程的状态转换图（7 状态图）
见另一文件，名为 Thread.pdf

作业：
用两种方式实现两个线程,一个线程负责打印 1-2600,另一个线程打印 A-Z,反复打印 100 遍

Java 第十三天 5 月 14 日

13.1
■ 阻塞状态



下面为线程中的 7 中非常重要的状态：（有的书上也只有认为前五种状态：而将“锁池”和“等待池”都看成是“阻塞”状态的特殊情况：这种认识也是正确的，但是将“锁池”和“等待池”单独分离出来有利于对程序的理解）
1，初始状态，线程创建，线程对象调用 start()方法。
2，可运行状态，也就是等待 Cpu 资源，等待运行的状态。
3，运行状态，获得了 cpu 资源，正在运行状态。
4，阻塞状态，也就是让出 cpu 资源，进入一种等待状态，而且不是可运行状态，有三种情况会进入阻塞状态。

1)如等待数据输入（输入设备进行处理，而 CPU 不处理），则放入阻塞，直到输入完毕，阻塞结束后会进入可运行状态。

2)线程休眠，线程对象调用 sleep()方法，阻塞结束后会进入可运行状态。

```
public static void sleep(long millis)
                    throws InterruptedException
```

括号中以毫秒为单位，使线程停止一段时间，间隔期满后，线程不一定立即恢复执行。

当 main()运行完毕，即使在结束时时间片还没有用完，CPU 也放弃此时间片，继续运行其他程序。

```
try{
    Thread.sleep(1000);
}catch(InterruptedException e){
    e.printStackTrace(e);
}
```

线程中有异常，只能 trycatch，子类中不能抛出比父类更多的异常，父类 run 方法没有抛出异常。

3)线程对象 2 调用线程对象 1 的 join()方法，那么线程对象 2 进入阻塞状态，直到线程对象 1 中止。

```
public final void join() throws InterruptedException
```

表示其他运行线程放弃执行权，进入阻塞状态，直到调用线程结束。

实际上是把并发的线程变为串行运行。

```
t1 num
t2 char    if(c=='m') -> t1.join()
//t2 对 t1 调用 join，t2 进入了阻塞状态
//当条件成立时，t1 加入打印数字，一直到打印完，此时 t2 继续运行
```

5，中止状态，也就是执行结束。

6，锁池状态

7，等待队列

13.2 共享数据的并发处理

13.2.1

■ 数据的错误发生

多线程并发访问同一个对象（临界资源）

破坏了原子操作，就会发生数据不一致的情况

13.2.2 共享数据的并发处理

■ 多线程同时并发访问的资源叫做临界资源。

多个线程同时访问对象并要求操作相同资源时分割了原子操作就会出现问题。（原子操作，不可再分的操作）会出现数据的不一致或数据不完整，为避免这种现象采用对访问的线程做限制的方法。

■ 互斥锁机制，利用每个对象都有一个 monitor(锁标记)，当线程拥有这个锁标记时才能访问这个资源，没有锁标记便进入锁池。任何一个对象系统都会为其创建一个互斥锁，这个琐是为了分配给线程的，防止打断原子操作。每个对象的锁只能分配给一个线程。

■ Synchronized 用法

1．Synchronized 修饰代码块（同步代码块），

```
public void push(char c){
    synchronized(this)//只有持有当前对象锁标记的线程才能访问这个代码块
```

```
        {
            ...
        }
    }
```

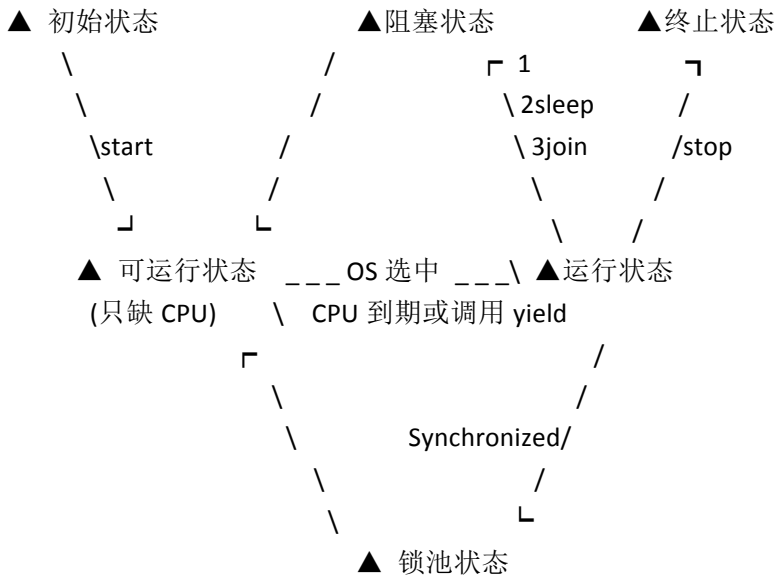
对括号内的对象加锁，只有拿到锁标记的对象才能执行该代码块

2．Synchronized 修饰方法

```
public synchronized void push(char c) {
        ...
}
```

在整个方法里，对当前对象的加锁，只有拿到锁标记的对象才能执行该方法。

■

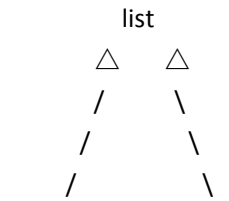


锁池：一个空间，每个对象都有一个，用来存放等待锁标记的线程

当一个对象中有了锁标记，不会释放其它对象的锁标记。

当 t1 线程正在访问对象 O 的同步方法时，别的线程 t2 不能访问 O 的任何同步方法，但还是可以访问其它的非同步方法

■ ArrayList Vector



ArrayList　　Vector(所有方法都做成了同步)

■ 构造方法　×对象没有完全构造好了，没有当前对象概念

抽象方法　×抽象方法没有代码块，没用

静态方法　√是对类对象的加锁

☆注意：构造方法不能 Synchronized 修饰

静态方法可以用 Synchronized 修饰（是对类对象加锁，类对象会在反射时讲到）

抽象方法不能用 Synchronized 修饰，不影响子类覆盖，子类在覆盖这个方法是可以加 Synchronized，也可以不加 Synchronized，所以根据 Java 不允许写废代码的特点是不能写在一起。

■ 练习：

1、用数组实现个栈；一个线程负责入栈一个线程负责出栈；长度为 6

char[6]，

T1, Push A~Z

T2, Pop

长度为 6，并且不允许扩充

注意：对当前对象加锁，一个代码块或者方法是同步的(Synchronized)，当前对象的锁标记没有分配出去时，有一个线程来访问这个代码块时，就会的到这个对象的锁标记，直到这个线程结束才会释放着个锁标记，其他想访问这个代码块或者是方法线程就会进入这个对象锁池，如果没有得到当前对象的锁标记，就不能访问这个代码块或者是方法。当一个线程想要获得某个对象锁标记而进入锁池，这个线程又持有其他对象的锁标记，那么这个线程也不会释放持有的锁标记。

注：方法的 Synchronized 特性本身不会被继承，只能覆盖。

线程因为未拿到锁标记而发生阻塞进入锁池（lock pool）。每个对象都有自己的一个锁池的空间，用于放置等待运行的线程。由系统决定哪个线程拿到锁标记并运行。

使用互斥锁的注意事项

举例：男孩和女孩例子，每个女孩是一个对象，每个男孩是个线程。每个女孩都有自己的锁池。每个男孩可能在锁池里等待。

```
Class Girl{
    Public void hand(){

    }
    Public syncronized void kiss(){

    }
}
Class Boy extends Thread{
    Public void run(){

    }
}
```

注意：只读不用加同步，只写也不用加同步，只有读写操作兼而有之时才加同步。

注意：在 java.io 包中 Vector 和 HashTable 之所以是线程安全的，是因为每个方法都有 synchronized 修饰。Static 方法可以加 synchronized , 锁的是类对象。但是 Vector 是 jdk 1.0 的 ArrayList 是 jdk1.2 所以实际应用还是使用 ArrayList。

注意：内同步，外同步，内同步，即，类内的方法加同步(synchronized)修饰,外同步即，在需要控制只能由一个线程进行访问时，把需要控制的方法写在同步代码块里。

Java 第十四天　　5 月 15 日

14.1 多线程的通信

■ 因为线程的死锁，从而引发要多线程的通信

死锁：每个线程不释放自己拥有的资源，却申请别的线程拥有的资源，会造成死锁问题

■ 线程间的通信：等待通知机制

■ 每一个对象都有一个等待队列。

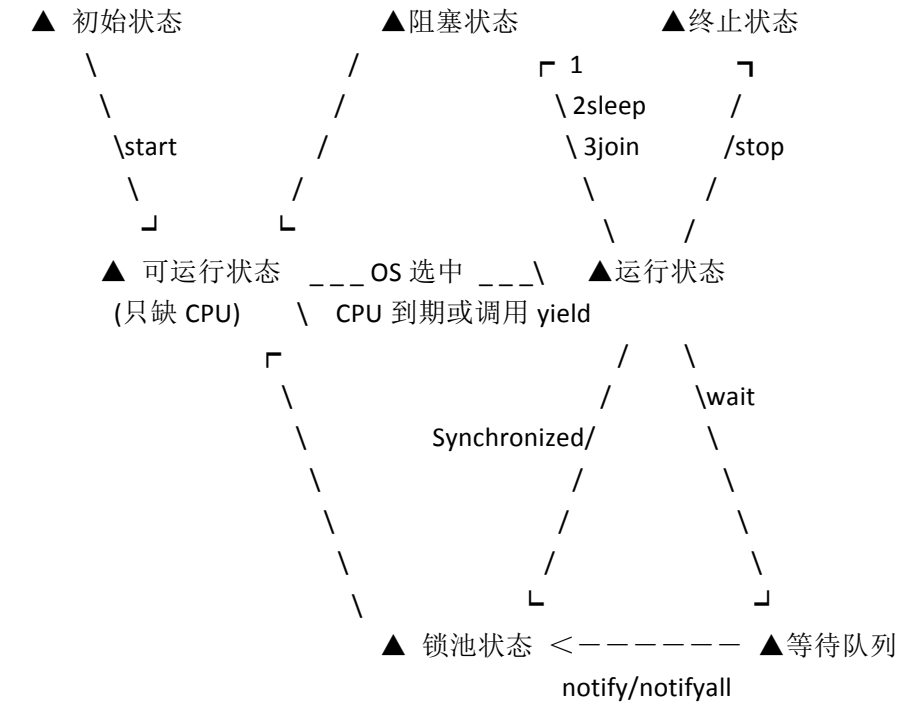
线程 t1 对 O 调用 wait 方法，△必须是在对 O 加锁的同步代码块中。

结果：1、线程 t1 会释放它拥有的所有的锁标记

2、会进入 O 的等待队列，开始阻塞

线程 t2 对 o 调用 notify/notifyAll 方法时，也必须是在对 o 加锁的同步代码块中。结果：会从 o 的等待队列中释放一个/全部线程

■



练习

1、利用线程的通信机制,用两个线程打印以下的结果:

1 2 A 3 4 B 5 6 C 7 8 D ... 49 50 Y 51 52 Z

2、思考为什么调用 wait（）方法之前用 while 判断，而不用 if 判断

currentThread

public static Thread currentThread()返回对当前正在执行的线程对象的引用。

14.2 IO 流

14.2.1 java.io Class File

■ 一个 File 对象代表了一个文件或目录

```
File f=new File("1.txt");//在堆里申请了个 File 对象的空间
f.createNewFile();//创建了个文件对象，不会产生文件，只有操作 File 对去创建文件
f.delete();
```

```
f.mkdir();
System.out.println(f.getName());//相对路径
System.out.println(f.getAbsolutePath());//绝对路径
```

■ File[] listFiles(FileFilter filter)

返回表示此抽象路径名所表示目录中的文件和目录的抽象路径名数组，这些路径名满足特定过滤器。

java.io

接口 FileFilter

accept

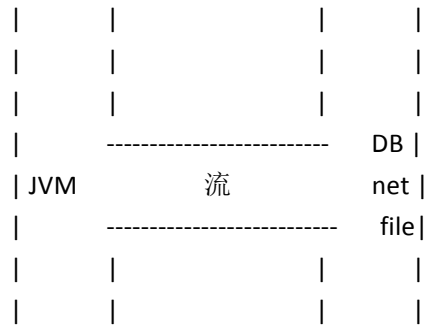
boolean accept(File pathname)

File pathname 是指被遍历目录中所有文件和目录

作业：打印目录下所有的.java 文件，包括子目录(提示：递归)

14.2.2

■ io 流是用于 JVM 和数据源之间交换数据



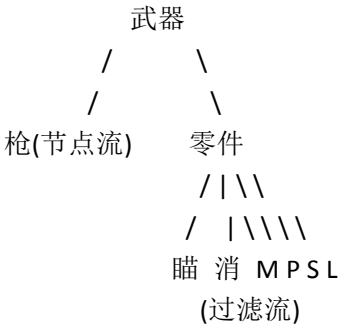
■ 一个流也是个对象

流的分类：输入流、输出流

字节流、字符流

节点流、过滤流 (功能，过滤流是给其它增加个功能，本身不传输数据)

■ 装饰模式



14.2.3.1 字节输入流

java.io

类 InputStream

字节输入流的所有类的超类

■ java.io

类 FileInputStream

public FileInputStream(String name)

throws FileNotFoundException//文件不存在会抛异常

FileInputStream(File file)

FileInputStream(FileDescriptor fdObj)

■ FileInputStream 中方法介绍

void close()

关闭此文件输入流并释放与此流有关的所有系统资源

int read()

从此输入流中读取一个数据字节。

返回下一个数据字节；如果已到达文件末尾，则返回 -1。

int read(byte[] b)

从此输入流中将最多 b.length 个字节的数据读入一个字节数组中。

返回：读入缓冲区的字节总数，如果因为已经到达文件末尾而没有更多的数据，则返回 -1。

int read(byte[] b, int off, int len)

从此输入流中将最多 len 个字节的数据读入一个字节数组中。

Java 第十五天　　5 月 16 日

15

15.1.1 DataInputStream/DataOutStream 增加读写

关流只关最外层的过滤流就行了\

15.1.2 BufferedInputStream/BufferedOutputStream 增加缓冲区

void flush()刷新此缓冲的输出流。

15.1.3 管道流（和 UnixC++中的 FIFO 相同）

■ PipedInputStream 和 PipedOutputStream （字节流）

这两个是节点流，注意，用来在线程间通信。

所有输入流读方法都是阻塞的

```
PipedOutputStream pos=new PipedOutputStream();
PipedInputStream pis=new PipedInputStream();
try
{
    pos.connect(pis);
    new Producer(pos).start();//线程类对象，在构造时，使用管道流通信
    new Consumer(pis).start();//线程类对象，在构造时，使用管道流通信
}
catch(Exception e)
{
    e.printStackTrace();
}
```

15.1.4 随机存取文件

RondomAccessFile 类允许随机访问文件，这个类也是支持直接输出输入各种数据类型。

GetFilepoint()可以知道文件中的指针位置，使用 seek()定位。

Mode(“r”:随机读；”w”：随机写；”rw”：随机读写)

1) 实现了二个接口：DataInput 和 DataOutput;

2) 只要文件能打开就能读写;

3) 通过文件指针能读写文件指定位置;
4) 可以访问在 DataInputStream 和 DataOutputStream 中所有的 read()和 write()操作;
5) 在文件中移动方法:
a. long getFilePointer(): 返回文件指针的当前位置。
b. void seek(long pos): 设置文件指针到给定的绝对位置。
c. long length(): 返回文件的长度。

15.2 字符流

■ 字符流可以解决编程中字符的编码问题。从字符到整数，对字符集和整数集建立一一对应的关系，就算叫做编码，从整数映射到字符，就叫做解码。

■ 编码问题：

字节流的字符编码：
字符编码把字符转换成数字存储到计算机中，按 ASCii 将字母映射为整数。
把数字从计算机转换成相应的字符的过程称为解码。

编码的方式：
每个字符对应一个整数。不同的国家有不同的编码，
当编码方式和解码方式不统一时，产生乱码。因为美国最早发展软件，所以每种的编码都向上兼容 ASCII 所以英文没有乱码。

ASCII（数字、英文）	1 个字符占一个字节（所有的编码集都兼容 ASCII）
ISO8859-1（欧洲）	1 个字符占一个字节
GB-2312/GBK	1 个字符占两个字节
Unicode	1 个字符占两个字节（网络传输速度慢）
UTF-8	变长字节，对于英文一个字节，对于汉字两个或三个字节。

■ InputStreamReader 和 OutputStreamWriter（字节流转化成字符流的桥转换器）
这两个类不是用于直接输入输出的，他是将字节流转换成字符流的桥转换器，并可以指定编解码方式。

Reader 和 Writer （字符流类，所有字符流的父类型）

1) Java 技术使用 Unicode 来表示字符串和字符，而且提供 16 位版本的流，以便用类似的方法处理字符。
2) InputStreamReader 和 OutputStreamWriter 作为字节流与字符流中的接口。
3) 如果构造了一个连接到流的 Reader 和 Writer，转换规则会在使用缺省平台所定义的字节编码和 Unicode 之间切换。

BufferedReader/（BufferedWriter，不常用）（这两个类需要桥转换）
PrintWriter（带缓存的字符输出流，不需要桥转换）

常用输入输出类型，不需要桥接，其中其它方法请参看 API 文档。
以上两个都是过滤流，需要用其他的节点流来作参数构造对象。
BufferedReader 的方法：readLine():String ，当他的返回值是 null 时，就表示读取完毕了。要注意，再写入时要注

意写换行符，否则会出现阻塞。
BufferedWriter 的方法：newLine() ，这个方法会写出一个换行符。
PrintWriter 的方法：println(…String,Object 等等)和 write()，println(...)这个方法就不必再写换行符了，在使用时会自动换行。

注意：在使用带有缓冲区的流时，在输入之后就要 flush()方法，把缓冲区数据发出去。

原则：保证编解码方式的统一，才能不至于出现错误。
java.io 包的 InputStreamread 输入流的从字节流到字符流的桥转换类。这个类可以设定字符转换方式。
OutputStreamred:输出流的字节流桥转换成字符流
Bufferread 有 readline()使得字符输入更加方便。
在 I/O 流中，所有输入方法都是阻塞方法。
Bufferwrite 给输出字符加缓冲，因为它的方法很少，所以使用父类 PrintWrite，它可以使用字节流对象，而且方法很多。

15.3 StringTokenizer
java.util
类 StringTokenizer

15.4 对象序列化
■ 把对象放在 IO 流上
ObjectInputStream 和 ObjectOutputStream（对象流）

对象流是过滤流，需要节点流作参数来构造对象。用于直接把对象写入文件和从文件读取对象。
只有实现了 Serializable 接口的类型的对象才可以被读写，Serializable 接口是个标记接口，其中没有定义方法。对象会序列化成一个二进制代码。

writeObject(o)，readObject()这两个是对象读写操作时用的方法。

```
Object o = new Object();
FileOutputStream fos=new FileOutputStream("Object.txt");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(o);
oos.close();

FileInputStream fis =new FileInputStream(“Object.txt”);
ObjectInputStream ois =new ObjectInputStream(fis);
Object o = (Object)Ois.readObject();
ois.close();
```

对象流读取结束返回 EOFException 异常对象。

一个类中有其他类型的对象，那么，这个类实现了 Serializable 接口，在对象序列化时，也同样要求这个类中属性都能够对象序列化（基本类型除外）。

注意：对于对象流的操作，在写对象时要一次写入完毕，如果使用追加模式写入，只会读取到上一次写入的对象，使用对象流写入时，会先写入一个头部，然后写入数据，

最后加上结束符号，如果使用追加方式写入的话，那就会在结束符号继续向下写入，但是在读取时只会读到结束符为止，以后再次写入的数据就会丢失。

注意：在使用对象流写入对象时要一次向文件写入，不能够采用追加方式。

serialver 命令判断是否一个属性或对象可序列化，
serialver TestObject（TestObject 必须为已经编译，也就是.class）
执行结果：如果不可序列化；则出现不可序列化的提示。如果可以序列化，那么就会出现序列化的 ID：UID。

Externalizable 这是 Serializable 的子接口，他可以让用户自定义如何序列化对象。
readExternal(ObjectInput in)，writeExternal(ObjectOutput out)这是这个接口中的两个方法，通过这两个方法可以定制序列化过程。这个方法不安全，可以调用以上两个方法改变对象的状态。

transient 只能用来修饰属性。表示这个属性在对象序列化时将被忽略。
transient int num;
表示当我们对属性序列化时忽略这个属性（即忽略不使之持久化）。所有属性必须都是可序列化的，特别是当有些属性本身也是对象的时候，要尤其注意这一点。

java.util.StringTokenizer 类，这个类是用于字符串截取的。
StringTokenizer（参数 1，参数 2）按某种符号隔开文件
StringTokenizer(s,”:”) 用“：”隔开字符，s 为对象。

补充：字节流结束返回-1，字符流结束返回 null，对象流结束返回 EOFException
引申---------〉异常经常被用在流程控制， 异常也是方法的一种返回形式。

作业：
用字符流把文件中的唐诗排列成古文的格式
用对象序列化把若干 Employee 对象写到文件中,再读取出来

项目练习：
修改 Bank 类,账户信息会采用对象序列化的方式存放在文件中.当 Bank 对象生成的时候会读取文件,设置账户集合.当账户信息改变的时候,会随时更新文件
设计一个 FileDAO 类(文件数据访问对象),负责对文件的访问,包括存放账户,提取账户等方法,在 Bank 类中,会通过 FileDAO 对象来访问文件

注意:如果已有的账户对象会存在文件中,那么为新的账户对象分配 id 的做法也应相应的改变,过去的用 static 属性的做法不再合适,应该改为,把下一个可用的 id 存放在一个文件中,
每创建一个新对象的时候都会读取这个文件,获得新对象的 id,并且修改文件中的 id,使其加 1.这个工作可以放在 Account 类的构造方法中

Java 第十六天　　5 月 17 日

网络编程

网络基础知识

网络编程的目的就是指直接或间接地通过网络协议与其他计算机进行通讯。

计算机网络形式多样，内容繁杂。网络上的计算机要互相通信，必须遵循一定的协议。目前使用最广泛的网络协议是 Internet 上所使用的 TCP/IP 协议。

IP 地址：计算机在网络中唯一标识，相对于 internet，IP 为逻辑地址。

IP 地址分类：

A 类地址

A 类地址第 1 字节为网络地址，其它 3 个字节为主机地址。另外第 1 个字节的最高位固定为 0。

A 类地址范围：1.0.0.1 到 126.155.255.254。

A 类地址中的私有地址和保留地址：

10.0.0.0 到 10.255.255.255 是私有地址（所谓的私有地址就是在互联网上不使用，而被用在局域网络中的地址）。

127.0.0.0 到 127.255.255.255 是保留地址，用做循环测试用的。

B 类地址

B 类地址第 1 字节和第 2 字节为网络地址，其它 2 个字节为主机地址。另外第 1 个字节的前两位固定为 10。

B 类地址范围：128.0.0.1 到 191.255.255.254。

B 类地址的私有地址和保留地址

172.16.0.0 到 172.31.255.255 是私有地址

169.254.0.0 到 169.254.255.255 是保留地址。如果你的 IP 地址是自动获取 IP 地址，而你在网络上又没有找到可用的 DHCP 服务器，这时你将会从 169.254.0.0 到 169.254.255.255 中临得获得一个 IP 地址。

C 类地址

C 类地址第 1 字节、第 2 字节和第 3 个字节为网络地址，第 4 个个字节为主机地址。另外第 1 个字节的前三位固定为 110。

C 类地址范围：192.0.0.1 到 223.255.255.254。

C 类地址中的私有地址：

192.168.0.0 到 192.168.255.255 是私有地址。

D 类地址

D 类地址不分网络地址和主机地址，它的第 1 个字节的前四位固定为 1110。

D 类地址范围：224.0.0.1 到 239.255.255.254

Mac 地址：每个网卡专用地址，也是唯一的。

端口(port)：应用程序（进程）的标识（网络通信程序）OS 中可以有 65536（2^16）个端口，进程通过端口交换数据。连线的时候需要输入 IP 也需要输入端口信息。

计算机通信实际上的主机之间的进程通信，进程的通信就需要在端口进行联系。

192.168.0.23:21

协议：为了进行网络中的数据交换（通信）而建立的规则、标准或约定，协议是为了保证通信的安全。

不同层的协议是完全不同的。

OSI 网络参考模型（理论性较强的模型）

七层，应用层、表示层、会话层、传输层、网络层、数据链路层、物理层：

网络层：寻址、路由（指如何到达地址的过程）

传输层：端口连接

TCP 模型：应用层/传输层/网络层/网络接口

层与层之间是单向依赖关系，上层依赖于下层，下层不依赖于上层，层与层之间的连接是虚连接。对等层之间建立协议。

端口是一种抽象的软件结构，与协议相关：TCP23 端口和 UDT23 端口为两个不同的概念。

端口应该用 1024 以上的端口，以下的端口都已经设定功能。

TCP/IP 模型

Application
(FTP,HTTP,TELNET,POP3,SMPT)
Transport
(TCP,UDP)
Network
(IP,ICMP,ARP,RARP)
Link
(Device driver,….)

注：
IP：寻址和路由
ARP（Address Resolution Protocol）地址解析协议：将 IP 地址转换成 Mac 地址
RARP（Reflect Address Resolution Protocol）反相地址解析协议：与上相反
ICMP（Internet Control Message Protocol）检测链路连接状况。利用此协议的工具：ping , traceroute

TCP Socket

TCP 是 Tranfer Control Protocol 的简称，是一种面向连接的保证可靠传输的协议。通过 TCP 协议传输，得到的是一个顺序的无差错的数据流。发送方和接收方的成对的两个 socket 之间必须建立连接，以便在 TCP 协议的基础上进行通信，当一个 socket（通常都是 server socket）等待建立连接时，另一个 socket 可以要求进行连接，一旦这两个 socket 连接起来，它们就可以进行双向数据传输，双方都可以进行发送或接收操作。

1) 服务器分配一个端口号，服务器使用 accept()方法等待客户端的信号，信号一到打开 socket 连接，从 socket 中取得 OutputStream 和 InputStream。

2) 客户端提供主机地址和端口号使用 socket 端口建立连接，得到 OutputStream 和 InputStream。

TCP/IP 的传输层协议

建立 TCP 服务器端

一般，我们把服务器端写成是分发 Socket 的，也就是总是在运行，

创建一个 TCP 服务器端程序的步骤：

1). 创建一个 ServerSocket
2). 从 ServerSocket 接受客户连接请求
3). 创建一个服务线程处理新的连接
4). 在服务线程中，从 socket 中获得 I/O 流
5). 对 I/O 流进行读写操作，完成与客户的交互
6). 关闭 I/O 流
7). 关闭 Socket

```
ServerSocket server = new ServerSocket(post)
Socket connection = server.accept();
ObjectInputStream put=new ObjectInputStream(connection.getInputStream());
ObjectOutputStreamo put=newObjectOutputStream(connection.getOutputStream());
```
处理输入和输出流;
关闭流和 socket。
典型的服务器端。
```
public class Server1 {
	public static void main(String[] args) throws Exception {
		ServerSocket ss=new ServerSocket(9000);
		while(true){
			Socket s=ss.accept();//获得一个 Socket 对象。
			Thread t=new Thread1(s);//分发 Socket。
			t.start();
		}
	}
}
class Thread1 extends Thread{
	Socket s;
	public Thread1(Socket s){
		this.s=s;
	}
	public void run(){
		try {
			OutputStream o=s.getOutputStream();
			PrintWriter out=new PrintWriter(o);
			out.println("Hello Client");
			out.flush();
			s.close();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
}
```
建立 TCP 客户端

创建一个 TCP 客户端程序的步骤：

1).创建 Socket
2). 获得 I/O 流
3). 对 I/O 流进行读写操作
4). 关闭 I/O 流
5). 关闭 Socket

```
Socket connection = new Socket(127.0.0.1, 7777);
ObjectInputStream input=new ObjectInputStream(connection.getInputStream());
ObjectOutputStream utput=new ObjectOutputStream(connection.getOutputStream());
```

处理输入和输出流;
关闭流和 socket。

练习：
实现一个网络应用，客户端会给服务器发送一个字符串，服务器会把这个字符串转换成大写形式发回给客户端
并有客户端显示，同时，服务器会告知客户端，他是第几个客户端

UDP socket
这种信息传输方式相当于传真，信息打包，在接受端准备纸。
先由客户端给服务器发消息。以告诉服务器客户端的地址。
特点：
1) 基于 UDP 无连接协议
2) 不保证消息的可靠传输
3) 它们由 Java 技术中的 DatagramSocket 和 DatagramPacket 类支持
DatagramSocket（邮递员）：对应数据报的 Socket 概念，不需要创建两个 socket，不可使用输入输出流。
DatagramPacket（信件）：数据包，是 UDP 下进行传输数据的单位，数据存放在字节数组中，其中包括了目标地址和端口以及传送的信息（所以不用建立点对点的连接）。
DatagramPacket 的分类：
用于接收：DatagramPacket(byte[] buf,int length)
DatagramPacket(byte[] buf,int offset,int length)
用于发送：DatagramPacket(byte[] buf,int length, InetAddress address,int port)
DatagramPacket(byte[] buf,int offset,int length,InetAddress address,int port)
注：InetAddress 类网址用于封装 IP 地址
没有构造方法，通过
InetAddress.getByAddress(byte[] addr):InetAddress
InetAddress.getByName(String host):InetAddress
等。

建立 UDP 发送端

创建一个 UDP 的客户端的程序的步骤：

1). 创建一个 DatagramPacket，其中包含发送的数据和接收方的 IP 地址和端口号。
2). 创建一个 DatagramSocket，其中包含了发送方的 IP 地址和端口号。
3). 发送数据
4). 关闭 DatagramSocket

```
byte[] buf = new byte[1024];
DatagramSocket datagramSocket = new DatagramSocket(13);// set port
DatagramPackage intputPackage = new DatagramPackage(buf,buf.length);
datagramSocket.receive(inputPackage);
DatagramPackage outputPackage = new DatagramPackage(buf,0， buf.length,inetAddress,port);
datagramSocket.send(outputPackage); //客户端向服务器发信的过程
```

没建立流所以不用断开。

建立 UDP 接受端

创建一个 UDP 的服务器端的程序的步骤：

1). 创建一个 DatagramPacket，用于存储发送方发送的数据及发送方的 IP 地址和端口号。
2). 创建一个 DatagramSocket，其中指定了接收方的 IP 地址和端口号。
3). 接收数据
4). 关闭 DatagramSocket

```
byte[] buf = new byte[1024];
DatagramSocket datagramSocket = new DatagramSocket();//不用设端口，因为发送的包中端口
DatagramPackage outputPackage=new DatagramPackage(Buf,buf.length,serverAddress,serverPort);
DatagramPackage inputPackage=new DatagramPackage(buf,0， buf.length);  //收信之前要准备一封空信
datagramSocket.receive(inputPackage);
```

UDP 处理多客户端，不必多线程，回信可以一封一封的回
在收发信设置为死循环

java.net.URL　　统一资源定位器
定位在 internet 上的某一资源
更高层的网络传输
常用格式
协议名：／／主机名：端口号／定位寻找主机内部的某一资源
http://www.tarena.com.cn:80/bin/index.html

作用：
１、必须完成
　　网络聊天室，以广播的形式传输数据
２、QQ 聊天室
　　实现点对点的数据传输
３、统计一个目录下的所有“.java”文件及子目录下的".java"文件总共多少行

Java 第十七天　　5 月 21 日

一、Java5.0 新特性
1、编译器的功能更加强大，JVM 变化不大
2、Java 在逐渐与 C++融合
3、五小点，四大点

二、五小点
1、自动封箱 AutoBoxing/自动解封
　自动封箱和自动拆箱，它实现了简单类型和封装类型的相互转化时，实现了自动转化。
　byte b　　-128~127
　Byte b　　在以上数量的基础上多一个 null

　简单类型和封装类型之间的差别

封装类可以等于 null ，避免数字得 0 时的二义性。

```
Integer i=null；
int ii=i；    //会抛出 NullException 异常。相当于 int ii=i.intValue();
Integer i=1；  //相当于 Integer i=new Integer(1);
i++;      // i = new Integer(i.intValue()+1);
```

在基本数据类型和封装类之间的自动转换

5.0 之前

```
Integer i=new Integer（4）；
int   ii= i.intValue();
```

5.0 之后

```
Integer i=4；
Long l=4.3;
```

```
public void m(int i){.....}
public void m(Integer i){.....}
```

以上两个函数也叫方法重载

自动封箱解箱只在必要的时候才进行。能不封箱找到匹配的就不封箱。

2、静态引入 StaticImport

使用类中静态方法时不用写类名

```
System.out.println(Math.round(PI));
```

可以用以下代码实现：

```
import static java.lang.System.*;  //注意，要写" 类名.* "
import static java.lang.Math.*;
out.println(round(PI));
```

注意：静态引入的方法不能重名

3、for-each

统一了遍历数组和遍历集合的方式

```
for(Object o:list){       //Object o 表示每个元素的类型 ，list 表示要遍历的数组或集合的名字
    System.out.println(o);  //打印集合或数组中的每个元素
}
```

4、可变长参数

处理方法重载中，参数类型相同，个数不同的情况

```
public void m(int... is){.....}
```

int... is 相当于一个 int[] is

编译器会把给定的参数封装到一个数组中，再传给方法

在一个方法中只能有一个可变长参数，而且，必须放在最后一个参数的位置

5、格式化输入/输出

java.util.Formatter 类 对格式的描述

System.out.printf("Hello %s",str); //打印字符串类型的变量，用一个占位符

格式化 I/O(Formatted I/O)

java.util.Sacner 类可以进行格式化的输入，可以使用控制台输入，结合了 BufferedReader 和 StringTokener 的功能。

三、四大点

1、枚举

枚举是一个类，并且这个类的对象是现成的，在定义类的时候，即定义好了对象

程序员要使用的时候，只能从中选择，无权创建

```
enum 枚举名{
   枚举值 1(..)，枚举值 2(..)，.....;
}
```

（1） 在 5.0 之前使用模式做出一个面向对象的枚举

```
final class Season{
public static final Season SPRING=new Season();
public static final Season WINTER=new Season();
public static final Season SUMMER=new Season();
      public static final Season AUTUMN=new Season();
      private Season(){}
}
```

完全等价于

```
enum Season2{
SPRING(..),//枚举值
SUMMER(..),
AUTUMN(..),
WINTER(..)
}
```

枚举本质上也是一个类，Enum 是枚举的父类。

这个类编译以后生成一个.class 类

这个类有构造方法，但是是私有的

枚举中的 values()方法会返回枚举中的所有枚举值

枚举中可以定义方法和属性，最后的一个枚举值要以分号和类定义分开，枚举中可以定义的构造方法。

枚举不能继承类（本身有父类），但可以实现接口，枚举不能有子类也就是 final 的，枚举的构造方法是 private（私有的）。

枚举中可以定义抽象方法，可以在枚举值的值中实现抽象方法。

枚举值就是枚举的对象，枚举默认是 final，枚举值可以隐含的匿名内部类来实现枚举中定义抽象方法。

（2）枚举类(Enumeration Classes)和类一样，具有类所有特性。Season2 的父类是 java.lang.Enum;

隐含方法： 每个枚举类型都有的方法。

```
Season2[] ss=Season2.values();        ----获得所有的枚举值
for(Season2 s:ss){
   System.out.println(s.name());     -----  打印枚举值
   System.out.println(s.ordinal()); -----  打印枚举值的编号
```

```
    }
```

（3） enum 可以 switch 中使用（不加类名）。

```
switch( s ){
    case SPRING:
            …….
    case SUMMER:
            …….
    …….
}
```

（4）枚举的有参构造

```
enum Season2{
    SPRING(“春”),--------------------------------逗号
    SUMMER(“夏”),--------------------------------逗号
    AUTUMN(“秋”),--------------------------------逗号
    WINTER(“冬”);--------------------------------分号
    private String name;
    Season2(String name){      //构造方法必须是私有的,可以不写 private,默认就是私有的
        this.name=name;
        }
        String getName(){
            return name;
        }

}
Season2.SPRING.getName()   ----------------------春
```

（5）枚举中定义的抽象方法，由枚举值实现：

```
enum Operation{
    ADD('+'){
        public double calculate(double s1,double s2){
            return s1+s2;
        }
    },
    SUBSTRACT('-'){
        public double calculate(double s1,double s2){
            return s1-s2;
        }
    },
    MULTIPLY('*'){
        public double calculate(double s1,double s2){
            return s1*s2;
        }
    },
    DIVIDE('/'){
```

```
            public double calculate(double s1,double s2){
                return s1/s2;
            }
        };
        char name;
        public char getName(){
            return this.name;
        }
        Operation(char name){
            this.name=name;
        }
        public abstract double calculate(double s1 ,double s2);
    }
```

有抽象方法枚举元素必须实现该方法。

```
    Operator[] os = Operator.values();
    for(Operator o:os){
        System.out.println("8 "+o.name()+" 2="+o.calculate(8,2));
    }
    for(Operator o:os){
        System.out.println("8 "+o.getName()+" 2="+o.calculate(8,2));
    }
```

运行结果：

```
8 ADD 2=10.0
8 SUBSTRACT 2=6.0
8 MULTIPLY 2=16.0
8 DIVIDE 2=4.0
8 + 2=10.0
8 - 2=6.0
8 * 2=16.0
8 / 2=4.0
```

2、泛型

(1)增强了 java 的类型安全，可以在编译期间对容器内的对象进行类型检查，在运行期不必进行类型的转换。

而在 java se5.0 之前必须在运行期动态进行容器内对象的检查及转换，泛型是编译时概念，运行时没有泛型

减少含糊的容器，可以定义什么类型的数据放入容器

(2)

```
List<Integer> aList = new ArrayList<Integer>();
aList.add(new Integer(1));
// ...
Integer myInteger = aList.get(0);   //从集合中得到的元素不必强制类型转换
```

支持泛型的集合，只能存放制定的类型，或者是指定类型的子类型。

```
HashMap<String,Float> hm = new HashMap<String,Float>();
```

不能使用原始类型

GenList<int> nList = new GenList<int>(); //编译错误

编译类型的泛型和运行时类型的泛型一定要一致。没有多态。

List<Dog> as = new ArrayList<Dog>();

List<Animal> l = as; //error Animal 与 Dog 的父子关系不能推导出 List<Animal> 与 List<Dog> 之间的父子类关系

（3）泛型的通配符"?"

? 是可以用任意类型替代。

<?> 泛型通配符表示任意类型

<? extends 类型> 表示这个类型是某个类型或接口的子类型。

<? super 类型> 表示这个类型是某个类型的父类型。

```
import java.util.*;
import static java.lang.System.*;
public class TestTemplate {
   public static void main(String[] args) {
        List<Object> l1=new ArrayList<Object>();
        List<String> l2=new ArrayList<String>();
        List<Number> l3=new ArrayList<Number>();   //Number --- Object 的子类，所有封装类的父类
        List<Integer> l4=new ArrayList<Integer>();
        List<Double> l5=new ArrayList<Double>();

        print(l1);
        print(l2);
        print(l3);
        print(l4);
        print(l5);
   }
       static void print(List<? extends Number> l){       //所有 Number 及其子类 l3,l4,l5 通过
        for(Number o:l){
              out.println(o);
        }
   }
       static void print(List<? extends Comparable> l){……}       //任何一个实现 Comparable 接口的类 l2,l4,l5 通过
       static void print(List<? super Number> l){……}                 //所有 Number 及其父类 l1,l3 通过
      // "?"可以用来代替任何类型，例如使用通配符来实现 print 方法。
       public static void print(GenList<?> list){……} //表示任何一种泛型

}
```

（4）泛型方法的定义 --- 相当于方法的模版

把数组拷贝到集合时，数组的类型一定要和集合的泛型相同。

<...>定义泛型，其中的"..."一般用大写字母来代替，也就是泛型的命名，其实，在运行时会根据实际类型

替换掉那个泛型。

在方法的修饰符和返回值之间定义泛型

<E> void copyArrayToList(E[] os,List<E> lst){……}

static <E extends Number & Comparable> void copyArrayToList(E[] os,List<E> lst){……} //定义泛型的范围类在前接口在后

static<E , V extends E> void copyArrayToList(E[] os,List<E> lst){……} //定义多个泛型

"super"只能用在泛型的通配符上，不能用在泛型的定义上

```
import java.util.*;
public class TestGenerics3 {
   public static void main(String[] args) {
        List<String> l1=new ArrayList<String>();
        List<Number> l2=new ArrayList<Number>();
        List<Integer> l3=new ArrayList<Integer>();
        List<Double> l4=new ArrayList<Double>();
        List<Object> l5=new ArrayList<Object>();
        String[] s1=new String[10];
        Number[] s2=new Number[10];
        Integer[] s3=new Integer[10];
        Double[] s4=new Double[10];
        Object[] s5=new Object[10];
        copyFromArray(l1,s1);
        copyFromArray(l2,s2);
        copyFromArray(l3,s3);
        copyFromArray(l4,s4);
        copyFromArray(l5,s5);

   }
   //把数组的数据导入到集合中
   public static <T extends Number&Comparable> void copyFromArray(List<T> l,T[] os){
        for(T o:os){
             l.add(o);
        }
   }
}
```

受限泛型是指类型参数的取值范围是受到限制的. extends 关键字不仅仅可以用来声明类的继承关系，也可以用来声明类型参数(type parameter)的受限关系.

泛型定义的时候，只能使用 extends 不能使用 super，只能向下，不能向上。

调用时用<?>定义时用 <E>

（5）泛型类的定义

类的静态方法不能使用泛型，因为泛型类是在创建对象的时候产生的。

```
class MyClass<E>{
   public void show(E a){
        System.out.println(a);
   }
```

```
    public E get(){
        return null;
    }

}
```

受限泛型

```
class MyClass <E extends Number>{
    public void show(E a){

    }
}
```

3、注释

4、并发

四、反射 reflect

反射，在运行时，动态分析或使用一个类进行工作。

反射是一套 API，是一种对底层的对象操作技术

1、类加载

类加载，生成.class 文件，保存类的信息

类对象，是一个描述这个类信息的对象，对虚拟机加载类的时候，就会创建这个类的类对象并加载该对象。

Class，是类对象的类。称为类类。只有对象才会被加载到虚拟机中。一个类只会被加载一次。

2、获得类对象的三种方式：（类对象不用 new 的方法得到的）

1)也可以用 类名.Class,获得这个类的类对象。

2)用一类的对象掉用 a.getClass()，得到这个对象的类型的类对象。

3)也可以使用 Class.forName(类名)（Class 类中的静态方法），也可以得到这个类的类对象，

（注意，这里写的类名必须是全限定名（全名），是包名加类名，XXX.XXX.XXXX）。强制类加载，这种方法是经常使用的。

一个类的类对象是唯一的。

在使用 Class.forName(类名)时，如果使用时写的类名的类，还没有被加载，则会加载这个类。

Class c;

c.getName(); 返回类名

c.getSuperclass(); 这个方法是获得这个类的父类的类对象。

c.getInterfaces(); 会获得这个类所实现的接口，这个方法返回是一个类对象的数组。

方法对象是类中的方法的信息的描述。java.lang.reflect.Method,方法类的对象可以通过类对象的 getMethods() 方法获得，

获得的是一个方法对象的数组，获得类中的定义的所有方法对象，除了构造方法。

构造方法对象，是用来描述构造方法的信息。java.lang.reflect.Constructor 构造方法类的对象可以通过类对象的 getConstructors()方法获得，

获得这个类的所有构造方法对象。

属性对象，使用来描述属性的信息。java.lang.reflect.Field 属性类的对象对象可以通过类对象 getFields() 这个方法是获得所有属性的属性对象。

作业：

1、通过运行时命令行参数输入一个类名，类出类中所有的方法

2、实现一个带泛型的堆栈，用来存放 Number

3、实现一个栈，用来存放任意类型

方法：遍历,pop,push,从数组拷贝

4、定义一个枚举，枚举值是课程，每个枚举值有个教师姓名的属性

Java 第十八天　　5 月 22 日

一、反射

1、获取方法和属性

反射可以获取这个类中定义的方法和属性的信息，简单数据类型在使用反射时要转换成封装类。

```
Class c = Teacher.class;
Method m = c.getMethod("mrthod",int.class);
```

2、通过类对象生成类的对象

```
Class c = Teacher.class;
Object o = c.newInstance();
```

3、通过类对象调用方法

```
//1.get class Object
Class c=Class.forName("Student");
//2.get Constructor object
Class[] cs={String.class};
Constructor con=c.getConstructor(cs);//按照参数表来调用制定构造方法。
//3.create object
Object[] os={"liucy"};
Object o=con.newInstance(os);
//4.get method object
String methodName="study";
Class[] pcs={String.class};
Method m=c.getMethod(methodName,pcs);//按照参数表来获得制定的方法对象。
//5.invoke the method
Object[] ocs={"EJB"};
m.invoke(o,ocs);
```

二、CoreJava 5.0 的注释

1、定义：Annotation 描述代码的代码（区：描述代码的文字）

给机器看　　给人看的

2、注释的分类：

（1）、标记注释：没有任何属性的注释。@注释名

（2）、单值注释：只有一个属性的注释。@注释名(value=___)

在单值注释中如果只有一个属性且属性名就是 value，则“value＝”可以省略。

（3）、多值注释：有多个属性的注释。多值注释又叫普通注释。

@注释名（多个属性附值，中间用逗号隔开）

3、内置注释：

（1）、Override（只能用来注释方法）

表示一个方法声明打算重写超类中的另一个方法声明。如果方法利用此注释类型进行注解但没有重写超类方法，则编译器会生成一条错误消息。

（2）、Deprecated

用 @Deprecated 注释的程序元素，不鼓励程序员使用这样的元素，通常是因为它很危险或存在更好的选择。在使用不被赞成的程序元素或在不被赞成的代码中执行重写时，编译器会发出警告。

（3）、SuppressWarnings（该注释无效）

指示应该在注释元素（以及包含在该注释元素中的所有程序元素）中取消显示指定的编译器警告。

4、自定义注释

自定义注释

```
public   @interface Test{


}
```

在自定义注释时，要用注释来注释（描述）注释。

@target()，用来描述（注释）注释所能够注释的程序员元素。

@Retention(),描述（注释）注释要保留多久。

注释的属性类型可以是

8 种基本类型

String

Enum

Annotation

以及它们的数组

5、注释的注释：java.lang. annotation 包中

（1）、Target：指示注释类型所适用的程序元素的种类。

例：@Target(value = {ElementType.METHOD});

说明该注释用来修饰方法。

（2）、Retention：指示注释类型的注释要保留多久。如果注释类型声明中不存在 Retention 注释，则保留策略默认为 RetentionPolicy.CLASS。

例：Retention(value = {RetentionPolicy.xxx}

当 x 为 CLASS 表示保留到类文件中，运行时抛弃。

当 x 为 RUNTIME 表示运行时仍保留

当 x 为 SOURCE 时表示编译后丢弃。

（3）、Documented：指示某一类型的注释将通过 javadoc 和类似的默认工具进行文档化。应使用此类型来注释这些类型的声明：其注释会影响由其客户端注释的元素的使用。

（4）、Inherited：指示注释类型被自动继承。如果在注释类型声

明中存在 Inherited 元注释，并且用户在某一类声明中查询该注释类型，同时该类声明中没有此类型的注释，则将在该类的超类中自动查询该注释类型。

注：在注释中，一个属性既是属性又是方法。

■ 标注：描述代码的文字

描述代码的代码。给编译器看的代码，作用是规范编译器的语法。

```
class Student{
   @Override
```

```
    public String toString(){
       return “student”;
    }

}
```

■ 注释类型 java.lang

1、标记注释（没有属性）

@Override

2、单值注释

@注释名(a="liucy")

@注释名（prameter=10）

int parameter

特例:

@注释名 (value=“134”)

等价于 @注释名 (“134”)

3.普通注释（多值注释）

(key1=value,……)

@__(a="liucy,b=10)

■ @Override 只能放在方法前面

@Deprecated 用于描述过期

@SuppressWarnings 抑制警告

@SuperessWarning({“ddd”,”aaa”,”ccc”}) //JVM 还没有实现这个注释

三、Java5.0 中的并发

1、所在的包：Java.util.concurrent

2、重写线程的原因：

（1）何一个进程的创建（连接）和销毁（释放资源）的过程都 是一个不可忽视的开销。

（2）run 方法的缺陷：没有返回值，没有抛例外。

3、对比 1.4 和 5.0 的线程

5.0	1.4
ExecutorService	取代 Thread
Callable Future	取代 Runnable
Lock	取代 Synchronized
SignalAll	取代 notifyAll()
await()	取代 wait()

三个新加的多线程包

Java 5.0 里新加入了三个多线程包：java.util.concurrent, java.util.concurrent.atomic,

java.util.concurrent.locks.

java.util.concurrent 包含了常用的多线程工具，是新的多线程工具的主体。

java.util.concurrent.atomic 包含了不用加锁情况下就能改变值的原子变量，比如说 AtomicInteger 提供了 addAndGet()方法。Add 和 Get 是两个不同的操作，为了保证别的线程不干扰，以往的做法是先锁定共享的变量，然后在锁定的范围内进行两步操作。但用 AtomicInteger.addAndGet()就不用担心锁定的事了，其内部实现保证了

这两步操作是在原子量级发生的，不会被别的线程干扰。

java.util.concurrent.locks 包包含锁定的工具。

Callable 和 Future 接口

Executor 接口替代了 Thread 类，他可以创建定量的和动态以及周期性的线程池。

ExecutorService 接口，线程池，用来存放线程来节省创建和销毁资源的消耗。

Callable 是类似于 Runnable 的接口，实现 Callable 接口的类和实现 Runnable 的类都是可被其它线程执行的任务。Callable 和 Runnable 有几点不同：

Callable 规定的方法是 call()，而 Runnable 规定的方法是 run().

Callable 的任务执行后可返回值，而 Runnable 的任务是不能返回值的。

call（）方法可抛出异常，而 run（）方法是不能抛出异常的。

Future 对象可以获得线程运行的返回值

运行 Callable 任务可拿到一个 Future 对象，通过 Future 对象可了解任务执行情况，可取消任务的执行，还可获取任务执行的结果。

以下是 Callable 的一个例子：

```
public class DoCallStuff implements Callable{ // *1
        private int aInt;
        public DoCallStuff(int aInt) {
                this.aInt = aInt;
        }
        public String call() throws Exception { //*2
                boolean resultOk = false;
                if(aInt == 0){
                        resultOk = true;
                }   else if(aInt == 1){
                        while(true){ //infinite loop
                                System.out.println("looping....");
                                Thread.sleep(3000);
                        }
                } else {
                        throw new Exception("Callable terminated with Exception!"); //*3
                }
                if(resultOk){
                        return "Task done.";
                } else {
                        return "Task failed";
                }
        }
}
```

*1: 名为 DoCallStuff 类实现了 Callable，String 将是 call 方法的返回值类型。例子中用了 String，但可以是任何 Java 类。

*2: call 方法的返回值类型为 String，这是和类的定义相对应的。并且可以抛出异常。

*3: call 方法可以抛出异常，如加重的斜体字所示。

以下是调用 DoCallStuff 的主程序。

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
public class Executor {
        public static void main(String[] args){
                //*1
                DoCallStuff call1 = new DoCallStuff(0);
                DoCallStuff call2 = new DoCallStuff(1);
                DoCallStuff call3 = new DoCallStuff(2);
                //*2
                ExecutorService es = Executors.newFixedThreadPool(3);
                //*3
                Future future1 = es.submit(call1);
                Future future2 = es.submit(call2);
                Future future3 = es.submit(call3);
                try {
                        //*4
                        System.out.println(future1.get());
                         //*5
                        Thread.sleep(3000);
                        System.out.println("Thread 2 terminated? :" + future2.cancel(true));
                        //*6
                        System.out.println(future3.get());
                } catch (ExecutionException ex) {
                        ex.printStackTrace();
                } catch (InterruptedException ex) {
                        ex.printStackTrace();
                }
        }
}
```

*1: 定义了几个任务

*2: 初始了任务执行工具。任务的执行框架将会在后面解释。

*3: 执行任务，任务启动时返回了一个 Future 对象，如果想得到任务执行的结果或者是异常可对这个 Future 对象进行操作。Future 所含的值必须跟 Callable 所含的值对映，比如说例子中 Future 对印 Callable

*4: 任务 1 正常执行完毕，future1.get()会返回线程的值

*5: 任务 2 在进行一个死循环，调用 future2.cancel(true)来中止此线程。传入的参数标明是否可打断线程，true 表明可以打断。

*6: 任务 3 抛出异常，调用 future3.get()时会引起异常的抛出。

运行 Executor 会有以下运行结果：

```
looping....
Task done. //*1
looping....
looping....//*2
looping....
looping....
```

```
looping....
looping....
Thread 2 terminated? :true //*3
//*4
java.util.concurrent.ExecutionException: java.lang.Exception: Callable terminated with Exception!
        at java.util.concurrent.FutureTask$Sync.innerGet(FutureTask.java:205)
        at java.util.concurrent.FutureTask.get(FutureTask.java:80)
        at concurrent.Executor.main(Executor.java:43)
        …….
```

*1: 任务 1 正常结束
*2: 任务 2 是个死循环，这是它的打印结果
*3: 指示任务 2 被取消
*4: 在执行 future3.get()时得到任务 3 抛出的异常

lock 接口

实现类 ReentrantLock

我们可以用 lock 对象，来对临界资源加锁，只有获得 lock 对象才能访问临界资源，如果没有获得 lock 对象，就会进入 lock 对象的锁池。trylock()方法会返回布尔值，这个方法是用来判断这个锁对象是不是已经被线程获取，如果返回值为 true，则会直接获得这个锁对象，如果返回 false，线程不会阻塞还会继续运行。

```
Lock lock=new ReentrantLock();
publci void test(){
    try{
  lock.lock();//加锁
  .....//需要加锁的临界资源。
  }finally{
     lock.unlock();//解锁。
  }
}
```

ReadWriteLock 读写锁接口

ReentrantReadWriteLock 是 ReadWriteLock 的实现类。

readLock()分配读锁，读锁可以分配多个线程，但是在分配读锁后所有读锁释放前，写锁时不能分配的。
Lock writeLock() 写锁只能分配给一个线程，在分配写锁后写锁是放前，读锁不能被分配。

Condition 接口和实现类

await()替代了 wait()方法。
notify()，notifyAll() 在 JDK5.0 中已经用 signal() ，signalAll()方法替换掉了，在 JDK5.0 中，可以使用多个等待队来存放等待的线程，并对线程进行分类。

Queue 接口（Collection 的子接口，对列接口）

LinkedList 也实现了这个在 JDK5.0 中的新接口 Queue，并且这个类自动的实现了生产者和消费者的同步。

JDK5.0 的高级同步

Semaphore 类（信号量）也就是可以向线程分配许可证，指定许可证数量可以实现多线程的同步。

Semaphore s=new Semaphore(4);//可以分配 4 个许可证，许可证都被分配出去时，得不到许可证的线程就会阻塞。

acquire()方法，获得许可证。release() 方法，释放一个许可证，也有相应的方法指定释放和获得许可证的数量的方法。

CountDownLatch 类，

CountDownLatch 中有个计数器，访问这个类的对象就会从计数器中减一，countDown()方法会将原有的设置的计数器值减一，当 countdown 计数器为零时会使放所有 await()的线程。

CyclicBarrier 类

CyclicBarrier 和 CountDownLatch 比较相似

CyclicBarrier 在构造时给出线程数，只有等待的线程数到了构造方法中指定的数量，当最后一个线程等待后，所有的线程都会被释放，这个类是一个多线程汇合的工具。

Exchanger 类，用 exchange()方法可以使两个线程间相互交换对象，在两线程的同步点，等待第二个线程。在同步点时，交换对象，并同时被释放。