# JAVA基础学习日记

# --19级学生沈章 记

Day1:

课1：

（SUN斯坦福网络公司推出的）

软件：一系列按照特定顺序组织排列的数据和指令（处理数据的方式）的集合。

常见软件：

系统软件：DOS（disk operation system）,windows,linux,mac等。

应用软件：

开发（制作软件）

人机交互：实现人与计算机之间的交互。

方式：图形化界面交互GUI（graphical users interface）；命令行方式CLI（command line interface）

计算机语音：实现人与计算机交互（C,C++,JAVA等）。

JAVA语音三种架构：J2EE(enterprise),J2SE(standard),J2ME(micro)

JAVA5.0后：JAVAEE,JAVASE,JAVAME

**Oracle公司**

课2：

JAVA语音特点1，实现跨平台管理JVM（JAVA virtual machine）。虚拟机不跨平台，每个版本都有每个版本指定的虚拟机，JAVA是管理各个版本的虚拟机。

课3：

环境搭建：

1. JRE（JAVA runtime environment JAVA运行环境）:JVM+JAVA核心类库
2. JDK（JAVA development kit JAVA开发包）：JAVA的开发工具（编译工具javac（java compiler）.exe+打包工具jar.exe）和运行环境

按需求使用上述两种环境搭建。

课4：

JDK下载，www.oracle.com

图形化界面方式，命令行方式DOS

DOS命令行操作H: → cd java → cd myeclipes → myeclipse\*（\*通配符） → dir+回车

课6：

图形化界面事实上就是命令行操作升级后的复杂操作界面。

早期操作常用DOS命令，但是dos命令需要记住许多命令不利于普及，后期发展为图形化界面操作。

常见DOS命令行：

dir:列出当前目录下的文件及文件夹；

Md（make directory）:创建目录；

Rd（remove directory）:删除目录，需要保证文件夹内部空，才可以删除；

Cd:进入指定目录；

Cd..：退回到上一级目录；

Cd\:退回到根目录；

Del\:删除文件；（del \*.class命令可以删除该目录下的.class文件）

Exit:退出dos命令行；

Cls（clear screen）:清屏；

Echo %path%：回显命令，显示path中所有的命令；

同时复制粘贴命令都存在。

Help帮助

课7：

疑问：为什么在任意目录下都能打开Notepad?

环境变量配置：

在dos命令行中执行\*\*\*命令程序，当\*\*\*命令程序不存在时，首先不报错，会在系统根目录的path环境变量（%SystemRoot%\system32）中寻找是否存在该执行程序。

怎样实现呢？

搭建环境变量步骤：

首先明白Path是代表的什么？

Path是指系统中应用程序所在的路径。

1. 下载并安装；2，配置环境变量；3，打开DOS命令行，输入\*\*\*（javac）

课8：

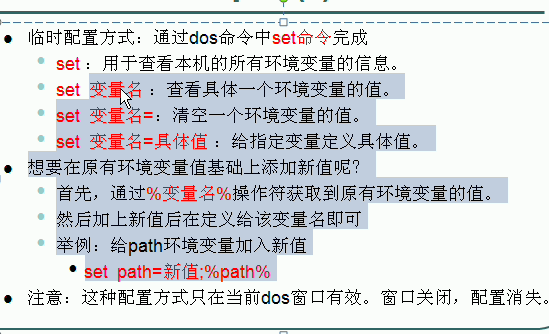
环境变量配置技巧：（环境：windows；变量：能记录数据的名称（%获取已有环境变量的值））

javac版本：javac -version（java compiler）

课9：

临时环境变量配置：

1. 查看环境变量（set path）;
2. 修改环境变量（set path=\*\*\*）;
3. 删除环境变量（set path= ）；
4. 添加新的环境变量set path=新值；%path%

仅在当前dos命令中有效，关闭后，命令无效。  


课10：

Hello world（第一步）

1. 编辑，最开始使用记事本，存储\*\*\*.java（源代码）文件（该文件关联记事本程序，故可以打开）
2. 将\*\*\*.JAVA文件翻译成虚拟机可识别文件（JAVA编译器Javac），即class文件（类文件）。
3. 执行类文件 java Demo

（遇到 Nosuchmethoderror: main 无main这类方法错误

public static void main(String[] args) （args:arguments参数）

{

System.out.println(“Hello World”);（println：输出完毕后，空行）

}

）

步骤： A，定义类；

1. 为了让该类独立运行，写入main；
2. 写入输出函数println与虚拟机互动。

课11：

Hello World细节：  
main：1，是程序的入口；2，虚拟机可调用；3，保证一个类的独立运行；4，有且仅有一个。

Javac（java compiler）：编译工具，dos中启动，用于编译.java文件，检查文件本身是否有错误。

Java：运行工具，运行虚拟机。找到Demo类文件，加载进内存（计算机临时存放空间）。

课12：

Hello World问题：

1. javac编译文件错误，或编译地址错误；
2. Javac编译文件类型错误，不是java文件；
3. mian没有语法错误，出现自定义函数。编译器无法判断出现错误，但虚拟机无法找到入口。若其他编译出现错误，如大小写问题，Javac编译会检查出错误。

课13：环境变量配置

创建名为“classpath”的环境变量，在变量值中输入“ %JAVA\_HOME%\lib;%JAVA\_HOME%\lib\tools.jar;”，然后点击确定，classpath环境变量就设置好了。

classpath环境变量：set classpath=\*\*\*地址；（后不要加分号）

Noclassdeffounderror:

A:Demo没有找到Demo错误；B:classpath环境变量写错（临时变量写入%classpath%）

classpath环境变量将java运行文件的路径告诉系统（即告知虚拟机在哪个地址执行java程序）。

如果没有配置classpath，虚拟机会在当前位置寻找java文件。

set classpath=.（点是指当前路径）;\*\*\*地址 该环境变量含有两个类路径设置。

public可以保证类名一致（在class前加入public），即：如果class类名的前面如果修饰的public则这个class类的文件命名必须和类名相同。

set path 可以查找path中的可执行程序，如.com;.exe;.bat...

Day2:

课1：

JAVA语言基础构成：

关键字；标志符；注释；常量和变量；运算符；语句；函数；数组

关键字：class public static void protected private 等等等。

特点：都是小写

类名首字母大写，若是多元单词组成，每一单词首字母必须大写。

函数名或方法，首字母小写，若是多元单词组成，后面的单词首字母大写。

main不是关键字，而是函数（主函数）。

命名时，需要认真理解命名规则与规范。

参考以下网址：<https://wenku.baidu.com/view/70976362bceb19e8b9f6ba23.html>。

课2：

标识符：在程序中自定义的一些名称：

由26英文字母大小写，数字0-9，符号\_$组成。

定义规则：1，数字不可以开头；2，不可以使用关键字。

注意：为提高阅读性，需自定义名称时有意义。

1，对于符号\_$可以用于连接某些名称，如Demo\_1$Test。

2，带下划线是同一类名称，如\_\*\*\*$\*\*\*；\_\*\*\*\*$\*\*\*\*。

课3：

注释：注解说明程序。

1. 单行注释，//\*\*\*\*\*\*\*\*\*\*；
2. /\* （） \*/，可以用于检查Java文件内部错误；
3. /\*\* \*/文档注释。

Javadoc，用于生成说明书，将文档注释和源代码放入一网页文件中，其中记录了文档注释和源代码。

单行注释中可以有多行和单行；多行注释中不可以有多行注释。

注意：注释完成后，文件大小并不会变大，反而会不变或变小。

课4：

注释的应用：对源代码说明注解，形成规范。

代码仅仅是思想的一种体现。

/\*思想：

需求：练习一个hello world程序。

思路：

1，定义一个类，因为Java程序都定义类中，java程序都是以类的形式存在的，类的形式其实就是一个字节码文件最终体现。

2，定义一个主函数，为了让该类可以独立运行。

3，因为演示hello world，在控制台上看到该字样，所以需要使用输出语句完成。

步骤：

1，用class关键字来完成类的定义，并起一个阅读性强的类名。

2，主函数：public static void main(String[] args)这是固定格式，jvm认识。

3，使用输出语句：System.out.println("hello world");

代码仅仅是思想的一种体现形式。

\*/

class Demo //这是一个用于演示/\*Hello World\*/的小程序。

{

//定义一个主函数，为了保证程序的独立运行。

public static void main(String[] args)

{

System.out.println("Hello World");//这是输出语句，用于将括号中的数据打印到控制台上，ln可以在数据结尾处换号。

}

}

课5：

常量与变量：

常量：表示不能改变的量

1. 整数常量；2，小数常量；3，布尔（boolean）型常量（true+false）；4，字符常量（单个数字或字母或字符，用’ ’标识）；5，字符串常量（一个多多个字符，用” ”标识）；6，null（仅有null）。

课6

进制的由来（用于计数），整数形式：

二进制，八进制（0开头，如023），十进制，十六进制（0x开头，如0x23）。

Byte字节=8个二进制位（bit位）

课7：

进制转换：

IP地址：IPV4（Internet protocol version 4，32位地址，4个字节）,IPV6（...）。

0000-0000 0000-0000 0000-0000 0000-0000共32位4字节，每字节为16\*16，0~255。

课8：

负数的进制：

课9：

变量：内存（临时存储区域）中的一个存储区域；（区域：作用范围）

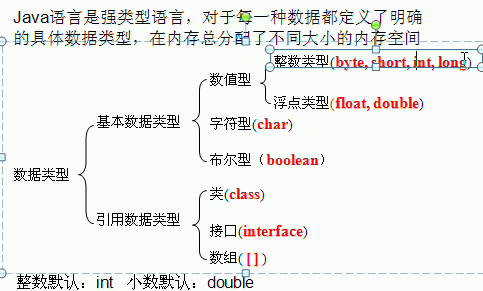
该区域有自己的名称（变量名）和类型（数据类型）；

该区域的数据可以在同一类型范围类不断变化。

（类型，变量名，初始值）

为什么要定义变量：用来不断存放同一类型的常量，并可以重复使用。

使用变量注意：



数据类型注意：

1，Byte:-128~127（-2^7~2^7-1 DVD，MP3的最小字节）；short:-2^15~2^15-1（2字节）；int:4字节（默认）；long:8字节长整型

2，根据精确程度分为：单精度float（4字节），双精度double（8字节）默认

3，char：0~65535，2字节，如’12’（×），’你’（✔）

4，boolean:true，false

课10：

变量的基本演示：

问题：1，其中 float， 由于浮点型是默认double，所以要在单精度float后加f，如1.3f。

2，遇见非法字符，说明出现中英文混淆。

3，作用域：上下括号问题。

课11：

类型提升&类型转换

注意：运算两端必须是同类型，如x = x + 3；但是，当两者都是数值型，两者同样可以运算。问题1，（会出现类型提升或转换，以占用内存较大的为主。）

问题2，byte x = 5; x = x + 3;会出现精度损失；

（分析：<https://blog.csdn.net/liangsir_l/article/details/70239198）>

课12：

字符类型运算过程：

二进制与数字，字母，符号的转换。（ASCII，GD2312，gb18030，GBK，Unicode）

A:65；a:97

课13：

类型运算细节：

byte b = 4; b = 3 + 7; System.out.println(b);

byte b = 4;（强制转换，int型4转化为byte型4）

▷类型转换精度丢失问题：

byte b1 = 4; byte b2 = 3; b = b1+b2; System.out.println(b);（b1，b2是变量，所以两值是不确定的，无法检查出b1+b2在byte型范围内。所以b值会丢失精度）运算右式是变化的，故左式无法确定。

对于默认类型就没有这方面的问题，如int b = 4; int b1 = 4; int b2 = 3; b = b1+b2; System.out.println(b);

课14：

算术运算符：+ - \* / %（取余，模运算）+(连接符)（如，println(3+”2”))）其中，模2运算，可以用于开关运算。（确定的用字符串，接+）

课15：算术运算符&赋值运算符：

++（自增，就在原有数据基础上），b = a++，a先将本身值存在一临时存储区域，作为临时值，在参与运算，自增，a = a+1，其次，a值再赋值给b。

int i = 3; i = i++; i=?

内部运算过程是：temp = i; i = i+1; i = temp;

赋值运算符：=， +=，\*=，/=，%=（存在编译自动类型转换）。

赋值运算符特点：在赋值运算过程，会在底层，进行类型转化或提升。

short s = 3;

s += 4;

s = s + 4;两者不等。

Day3:

课1：

比较运算符&逻辑运算符

比较运算符：运算结果仅有两种结果：true or false。

==， !=， <=， >=， <， >，instanceof（instanceof通过返回一个布尔值来指出，这个对象是否是这个特定类或者是它的子类的一个实例）。

逻辑运算符：用于连接两个boolean类型的表达式。

&：与and； |：或or；

课2：

逻辑运算符：^：异或xor；！：非not；&&；||。

&&与&的区别：首先两者结果相一致，int x = 1; System.out.println(x>2 && x<5);（双与计算，如果第一个计算为假，后面的计算不进行；单与不仅可以进行逻辑运算，还可以进行其他运算）

&&和&结果一致，但是有如下区别：

&：无论左边结果是什么，右边都参与计算；

&&：当左边结果为false，右边不计算。

||与|区别：

|：无论左边结果是什么，右边偶读需要计算；

||：当左边结果为true，右边不参与计算。

课3：

位运算符：（用于二进制或bit位计算）

^：一个数异或同一个数两次，结果不变。如，6^3^3 = 6。

可以用于数据加密。

~：按位取反，如~6 = -7，（0000 0000 0000 0110 → 1111 1111 1111 1001 → 因为是负数，计算机中存储是补码，故前值1000 0000 0000 0110 + 1 = 1000 0000 0000 0111，即-7 ）

课4：

移位运算符：

1. <<左移（2的次幂运算）；2，>>右移（补足位：如果首位是1，就补1，如果是0，则补0）2的次幂整除；3，>>>无符号右移（用0补）

课5&课6：

运算符运算

对两个整数变量的值进行变换（不能有第三方变量）

/\*这种方法不要用，如果两个整数的数值过大，会超过int 型变量范围，会强制转化，数据会发生变化。

a = a + b;b = a - b;a = a - b;\*/

运用异或的特点： a = a ^ b;

b = a ^ b;

a = b ^ a;

但是，真正在开发时，针对阅读性，选用第三方变量的方法。（a = a + b;b = a - b;a = a - b;）

课7：三元运算符：

格式：（条件表达式）？表达式1：表达式2；

如果条件为true，运算后的结果为表达式1；

如果条件为false，运算后的结果为表达式2；

如：int x = 3,y; y = (x > 1)?100:200;

// 运算符和if等语句的区别：

// 运算符必须运算完，if不需要。 ?

课9：

语句if\_1：

int x = 3;

if (x > 1) //当If语句中仅有单条条件语句时，{}可以省略不写；如果不写{}，仅运行后接的单条语句。

{

System.out.println("yes");

}

else //否则  
 {

}

System.out.println("over");

课10：

语句if\_2：（if else）

int a = 3,b;

/\*

if (a >1)

b = 100;

else

b = 200;

\*/

b = a>1?100:200;

//简写格式什么时候用？

//当if else运算后，有一个具体的结果时，可以简化为三元运算符。

（三元运算符必须经过运算，而If else不一定运算）

课11：

语句If\_3:

if(条件表达式)

{

执行语句；

}

else if(条件表达式)

{

执行语句；

}

...

else

{

}

语句嵌套。if (x == 1)//语句嵌套。

{

if (y == 1)

System.out.println("a");

else

System.out.println("b");

}

else

{

if (y == 1)

System.out.println("c");

else

System.out.println("d");

}

上语句嵌套语句可以省略{}。

课12：局部代码块

if (false);//局部代码块：可以定义局部变量的生命周期

{

System.out.println(“hello world”);

}

课13：

if else 更加有效率。

课14：

提高代码的复用性，当多个条件相同时，合并。

课15：

switch语句。

switch(表达式) //byte,short,int,char

{

case 取值1：

执行语句；

break;

case 取值2：

执行语句；

break;

...

default://默认，其它选项。 default可以任意放置位置。但是，执行顺序是：default在其它case执行完后，最后执行。

执行语句；

break; // break可以省略不写，因为，执行到大括号}处，不用break，可以直接出去。break是跳出循环，退出switch命令。

}

会从第一个case开始执行，直至执行到所需结果。

课16：switch练习。

//提高代码的复用性

SwitchTest.java:80: 错误: 非法字符: '\uff5d'

｝

^

SwitchTest.java:81: 错误: 解析时已到达文件结尾（大括号用了中文，显示结尾错误）

课17：If语句和switch语句区别：

If和switch的应用：

if:

1，对具体的值进行判断。

2，对区间判断。

3，对运算结果是boolean类型的表达式进行判断。

switch:（使用将对较少，后添加了枚举类型，字符串类型）

1，对具体的值进行判断。

2，值的格数通常是固定的。

3，对于几个固定的值判断，建议使用switch语句，因为switch语句会将具体的答案都加载进内存

效率相对高一点。

课18：

while语句（循环结构）：

课19：

Do while:错误: 找不到或无法加载主类 DoWhileDemo（类名不一致）

Do while语句，无论结果时否循环，至少执行一次。

错误: 已在方法 main(String[])中定义了变量 x（说明重复定义同一变量）

课20：

While练习-累加思想。

int x = 1,y = 0;//记录参与加法的数据；记录住每一次的和。

while(x <= 10)

{

y += x;

x++;

}

System.out.println(y+"是1到10的数字的和");

课21：

While练习-计数器思想。（记录一个变量的次数）

分析其中，哪些数值是在不断变化的。

int x = 6,y = 0;//记录参与加法的数据；记录住每一次的和。

while(x <= 100)

{

if(x % 6 == 0)

{

System.out.println(x);

y++;

}

x++;

}

System.out.println(y);

课22：

for循环：

For(初始化表达式；循环条件表达式；循环后的操作表达式)

｛

执行语句；（循环体）

｝

forDemo.java:10: 错误: 不兼容的类型: void无法转换为boolean

for(System.out.println("a");System.out.println("b");System.out.println("c"))

其中，println，ln的作用，仅在最后会空出一行！

// for(int a = 0,b = 0;a < 3;a++,b--)基本使用。

课23：

for练习&for和while的区别：

for(int x = 1,sum = 0;x <=10; x++)

{

sum = sum + x;

}

System.out.println("sum = "+sum);

错误: 找不到符号,符号:变量 sum

System.out.println("sum = "+sum);

（sum仅在for循环语句中存在，所以在println中无法输出）

int sum = 0；

for(int x = 1;x <=10; x++)

{

sum = sum + x;

}

System.out.println("sum = "+sum);

for和while的特点：

1. for和while可以互换。
2. 格式上的不同，在使用上有点小区别。

如果需要通过变量来循环进行控制，该变量只作为循环增量存在时，区别就体现出来了。

控制条件语句它所控制的变量，在for循环结束以后就不能被再次访问到了，但是while循环结束以后还是可以继续使用的，如果我们想要它继续使用，那么就要用while，否则推荐使用for。

如果对内存空间要求，循环结束后，该变量会从内存中结束，则优先考虑for。

//无线循环最简单的形式。

While(true)

{}

For(;;){}

课24：循环结构的使用场景。

什么时候使用循环结构呢？

当对某些代码执行很多次时，使用循环结构。

当对一个条件进行一次判断时，可以使用if语句。

当对一个条件进行多次判断时，可以使用while语句。

注意：

在会用循环时，一定要明确哪些语句需要参与循环，哪些不需要。

循环通常情况下，需要定义条件，需要控制次数。

课1：

for语句嵌套：

//大全套小圈思想。

for(int x = 0; x <4; x++) //循环控制的是行数

{

for(int y = 0; y < 5; y++) //内循环控制的是每一行的个数。

{

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

}

System.out.println();

}

课2：

For循环嵌套练习：

\*\*\*\*\* \*\*\*\*\* \*

\*\*\*\*\* \*\*\*\* \*\*

\*\*\*\*\* \*\*\* \*\*\*

\*\*\*\*\* \*\* \*\*\*\*

\* \*\*\*\*\*

54321 1

5432 22

543 333

54 4444

5 55555

错误: 需要class, interface或enum （原因是：括号出现多了或者少了）

课4：

错误: 找不到符号 符号: 变量 y（没有写变量。如int x = 12;）

//实现9\*9乘法表

/\*

1\*1 =1

1\*2 =2 2\*2 = 4

1\*3 =3 2\*3 = 6 3\*3 = 9

\*/

for(int x = 1;x <= 9;x++)

{

for(int y=1;y <= x;y++)

{

System.out.print(y+" \* "+x+" = "+x\*y+"\t");

}

System.out.println();

}

\n 可以在任意处换行，用于字符串中。

\t 制表符的使用。！！！ 在不使用表格的情况下在垂直方向按列对齐文本，第一位至第九位对齐。

\b 退格。

\r 按下回车键。

Windows系统中回车符其实是有两个符号组成\r\n。

linux中回车符是 \n。

\ 转义字符，如” \”hello world \”” ==”hello world”（\ 仅作用于其后的数据）

课5：

For循环嵌套练习：

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

**for**(**int** x = 5 ; x>0;x--)

{

**for**(**int** z=x;z<5;z++){

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

}

**for**(**int** y = 0;y<x; y++)

{

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

}

System.*out*.println();

}

课6：

break（跳出，终止循环） & continue（结束本次循环，继续下一次循环）  
/\*

Break:跳出

break作用范围：要么是switch语句，要么是循环语句。

记住：当break语句单独存在时，下面不要定义其他语句。

break跳出所在的当前循环。

如果出现了循环嵌套，break想要跳出指定的循环，可以通过标号来完成。

\*/

/\*

continue：继续。

作用的范围：循环结构。

continue:结束本次循环，继续下次循环。

如果continue单独存在时，下面不要有任何语句，因为执行不到。

\*/  
错误: 无法访问的语句 System.out.println("x="+x);（不执行的语句，多余）

课1：

函数定义:

函数就是定义在类中的具有特定功能的一段独立小程序。

函数也称方法。

FunctionDemo.java:11: 错误: 无法从静态上下文中引用非静态 方法 add(int,int)

int c = add(3,4);

// public 权限问题

public static int add(int a,int b)

{

return a+b;

}

课2：java函数格式

/\*

定义函数的格式：

修饰符 返回值类型 函数名（参数类型 形式参数1，参数类型 形式参数2，...）

｛

执行语句；

return 返回值；

｝

\*/

修饰符（让其具备更多的含义）

函数名（提高阅读性规范：第一个需要小写，之后的得大写）

参数（参与运算的数据）类型

return功能：返回值；结束语句，之后的语句无效。

课3：函数细节

特殊情况：功能没有具体的返回值。

这时return后面直接用分号结束。

返回值类型怎么体现呢？因为没有具体值。所以不可以写具体的数据类型。

在Java中只能用一个关键字来标识这种情况，关键字是：void。

总结：没有具体返回值时，返回值类型用void来表示。

注意：如果返回值类型是void，那么函数中的return语句可以省略不写。

课4：

函数特点：

1)定义函数可以将功能代码进行封装

2)便于对该功能进行复用

3)函数只有被调用才会被执行

4)函数的出现提高了代码的复用性

5)没有具体返回值时，用关键字void，那么该函数最后一行可以不写return。

注意：函数中只能调用函数，不可以在函数内部定义函数。

定义函数时，函数的结果应该返回给调用者，交由调用者处理。（不能在定义的函数中直接输出）。

课5：

输出语句中，必须保证是具体数值或结果。而不能使用System.out.println(add(3,4));

产生错误: 此处不允许使用 '空' 类型 System.out.println(add(3,4));

课6：函数两个明确

明确一：这个功能的结果是什么？

是和。是功能的结果，所以该功能的返回值类型是int。

其实就是在明确函数的返回值类型。

明确二：这个功能实现过程中是否需要未知内容参与运算？

有，加数和被加数。这就是函数的参数列表（参数的个数，参数的类型）

其实就是在明确参数列表。

课7：

比较运算符的结果就是boolean类型。故

return (a==b)?true:false 就是return a==b;

课8：

两个明确练习2：

定义函数时，函数的结果应该返回给调用者，交由调用者处理！分工明确

课9：内存加载过程

class FunctionDemo2

{

public static int add (int a,int b)

{

return a+b;

}

public static void main (String[] args)

{

Int x = add(3,5);

System.out.println(“x=”+x);

}

}

Javac FunctionDemo3.java（编译检查过程，是否出现语法错误，生成.java指定格式的文件） → FunctionDemo3.class（虚拟机访问，通过java FunctionDemo3 ,启动虚拟机，寻找main函数，保证该类独立运行）

栈：先进后出

课10：函数-重载

概念：在同一个类中，允许存在一个以上的同名函数，只要它们的参数个数或者参数类型不同即可。

函数体名是根据函数功能而定。

调用的不确定性。

/\*

函数的重载。

1，同一个类，

2，同名。

3，参数个数不同，or 参数类型不同。

4，函数重载和返回值类型无关。

5，Java是严谨性语言，如果函数出现的调用的不确定性，会编译失败。

\*/

//加法运算，两个整数的和。

public static int add(int a,int b) //函数体名是根据函数功能而定。

{

return a+b;

}

//加法运算，两个小数的和。

public static double add(double a,double b) //函数体名是根据函数功能而定。

{

return a+b;

}

//加法运算，三个整数的和。

public static int add(int a,int b,int c) //函数体名是根据函数功能而定。

{

return a+b+c;

}

课11：

重载练习：

提高代码的阅读性，通过参数类型来区分这些重载函数。

课1：

数组概述：

同一种类型数据的集合，其实数组就是一个容器。

数组优势：

为数组中的元素编号从0开始，方便操作这些元素。

格式1：

元素类型[] 数组名 = new 元素类型[元素个数或数组长度]

如：int [] arr = new int [5];

格式2：

元素类型[] 数组名 = new 元素类型[] {元素，元素，...};

Int [] arr = new int [] {1,3,5,7};

Int [] arr = {1,3,5,7};

课2：

数组定义

arr[0]，0为索引

ArrayDemo.java:9: 错误: 需要 '.class'

System.out.println(arr[]);（arr[]中没有索引，如arr[1]）

课3：

数组内存空间划分：

1. 寄存器。CPU处理
2. 本地方法区。和所在系统（windows,linux,mac...）相关。
3. 方法区。（数据区，共享区，共享数据区）
4. 栈内存。
5. 堆内存。

课4：栈内存：

存储的都是局部变量。

而且变量所属的作用域一旦结束，该变量就自动释放。

{

/局部代码块。限定局部变量的生命周期。

}

//栈中如果挂了一个，会直接挂掉；而堆中，挂掉一个，其他还会继续。

课5：堆内存：

存储是数组和对象（其实数组就是对象），凡是new建立在堆中。

特点：

int arr[] = new int [3];

第一步：方法（主函数main， arr）进栈；

第二步：分配堆内存地址，数组中的实体（int[3]）入堆，进行初始化；

堆内存中初始化（每一个实体都会初始化分配数值，不同类型初始化值都不同，

如boolean：false，double:0.0，float:0.0f，字符串初始化null，char:’\u0000’）。

解释char:’\u0000’：

’\u0000是char类型的最小值，它转换成整数时为0，’\u0000’与’ ’是不相等的。

第三步：将实体int[3]的地址给栈中的arr，称为arr引用（即c++中的指针）或指向堆内存的一个实体，arr = NULL;（不指向任何实体）；

第四步：那么堆中的实体将不会像栈中直接释放，而会被视作垃圾，自动回收（不定时时间内，java在内存管理方面优于c++）。

注意：

1. 每一个实体都有首地址值。
2. 堆内存中的每一个变量都有默认初始值，根据类型的不同而不同。boolean：false，double:0.0，float:0.0f，char:’\u0000’，字符串初始化null（\u代表unicode表，0000代表两个字节，即空格空位）
3. 垃圾回收机制。（java 与 c++不同）

课6：数组内存图解。

课7：数组常见问题

int [] arr = new int [3];//声明一个arr名的数组，创建一个int型数组实体，大小为3

arr[0] = 89;

System.out.println(arr[3]);

//Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3

// at ArrayDemo2.main(ArrayDemo2.java:10)

//数组索引超出范围（数组角标异常），即越界。

//当访问到数组中不存在的角标时，就会发生该异常

//Exception in thread "main" java.lang.NullPointerException

at ArrayDemo2.main(ArrayDemo2.java:11)

//（空指针异常）

//当引用型变量没有任何实体指向时，还在用其操作实体，就会发生异常。

//[I@15db9742 哈希方式？？？

[:数组，I:int类型。

Day5:

课1：数组第二种定义格式

//静态初始化方式

//元素类型 [] 数组名 = new 元素类型[]{元素，元素，...}

int [] arr = new int []{89,34,270,17};

课2：数组常见操作-遍历

/\*

对于数组操作最基本的动作就是存和取。

核心思想：就是对角标的操作。

\*/

length:数组长度。（关键字）

课3：

数组常见操作-最值

最大值，最小值

排序，冒泡排序

/\*

错误: 不兼容的类型: int无法转换为int[]

int max = getMax2(arr[maxIndex]);

^

注: 某些消息已经过简化; 请使用 -Xdiags:verbose 重新编译以获得完整输出

\*/

（送入的是arr，不是arr[maxIndex]）

课4：数组-选择排序

数组类型不需要返回。

课5：数组-冒泡排序

课6：数组-排序位置置换代码提取

课7：数组-排序的性能问题

课8：数组-查找

课9：数组-折半查找（二分查找）

课10：折半查找练习

Arrays.binarySearch函数。（Java自带二分查找）//Arrays.binarySearch仅在java中存在。如java thread 就没有。

//如果存在返回的具体的角标位置，不存在返回的是 -插入角标-1

可是为什么我无法找到该函数。~~！！！

课11：进制转换\_1

课12：进制转换\_2-查表法

// 0，1，2，3，4，5，6，7，8，9，A， B, C， D， E， F

// 0，1，2，3，4，5，6，7，8，9，10，11，12，13，14，15

/\*

什么时候使用数组呢？

如果数据出现了对应关系，而且对应关系的一方是有序的数字编号，并作为角标使用。

这时就必须想到数组的使用。

就可以将这些数据存储到数组中。

根据运算的结果作为角标直接去查数组中对应的元素即可。

这种方式：成为查表法。

\*/

课13：进制转换\_整合

public static void trans(int num,int base,int offset) { //base进制；offset:偏移量

if(num == 0)

{

System.out.println("0");

return; //如果不加return，会继续执行下面程序，故可以

//通过return，直接结束下面程序。

}

char [] chs = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};

int pos = arr.length;

While(num != 0)

{

int temp = num &15;

arr[--pos] = temp;

num = num >>>offset;

}

System.out.println(“pos = ”+pos);

for(int x = pos;x<arr.length;x++)

{

System.out,print(arr[x]);

}

System.out.println();

}

课14查表法练习：

课1：

数组中的数组：（二维数组）

int [][] arr = new int [3][2]; //创建一个二维数组，该数组中有3个一位数组，每一个一位数组中都有2个元素。

（int [] arr 和 int arr []实际上是一样的，早期c++是arr[]）

课2：

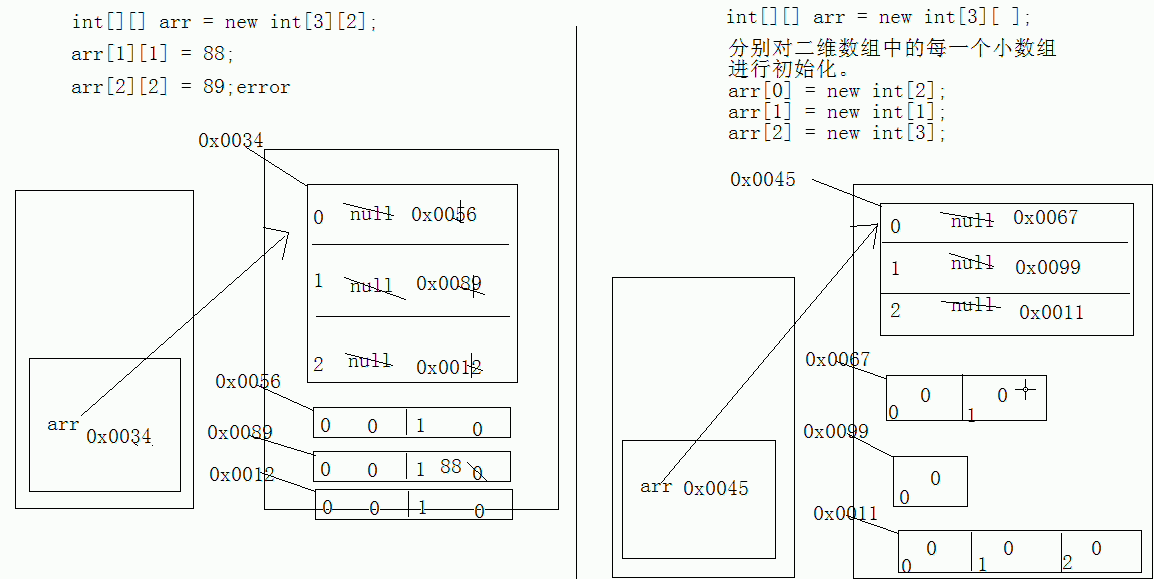
Int[][] arr = new int [3][];

分别对二维数组中的每一个小数组进行初始化。

Arr[0] = new int [2];

arr[1] = new int [1];

arr[2] = new int [3];



int [] arr = new int [3]; // [I@15db9742 ：@左边是实体的类型。@右边是实体的哈希值。

int [][] arr = new int [3][2]; //创建一个二维数组，该数组中有3个一位数组，每一个一位数组中都有2个元素。

System.out.println(arr); //直接打印二维数组。 // [[I@15db9742 （[[代表二维）

System.out.println(arr[0]); //直接打印二维数组中的角标0的一维数组。 // [I@6d06d69c （[代表一维）

System.out.println(arr[0][0]); //直接打印二维数组中的角标0的一维数组的角标为0的元素。 // 0

int [][] arr = new int [3][]; //创建一个二维数组，该数组中有3个一位数组，每一个一位数组中都有0个元素。

System.out.println(arr); //直接打印二维数组。 // [[I@15db9742

System.out.println(arr[0]); //直接打印二维数组中的角标0的一维数组。 // null

System.out.println(arr[0][0]); //直接打印二维数组中的角标0的一维数组的角标为0的元素。 // NullPointerException

课3：二维数组另一种定义方式：

int [][] arr = {{3,1,7},{5,8,2,9},{4,1}};

课4：二维数组应用场景

for(int x = 0 ; x<arr.length;x++)

{

//System.out.println(arr[x]); // [I@15db9742

// [I@6d06d69c

} // [I@7852e922

（上述命令输出，不能确定输出的是哪一具体元素）

课1： 面向对象

面向过程。一种思想

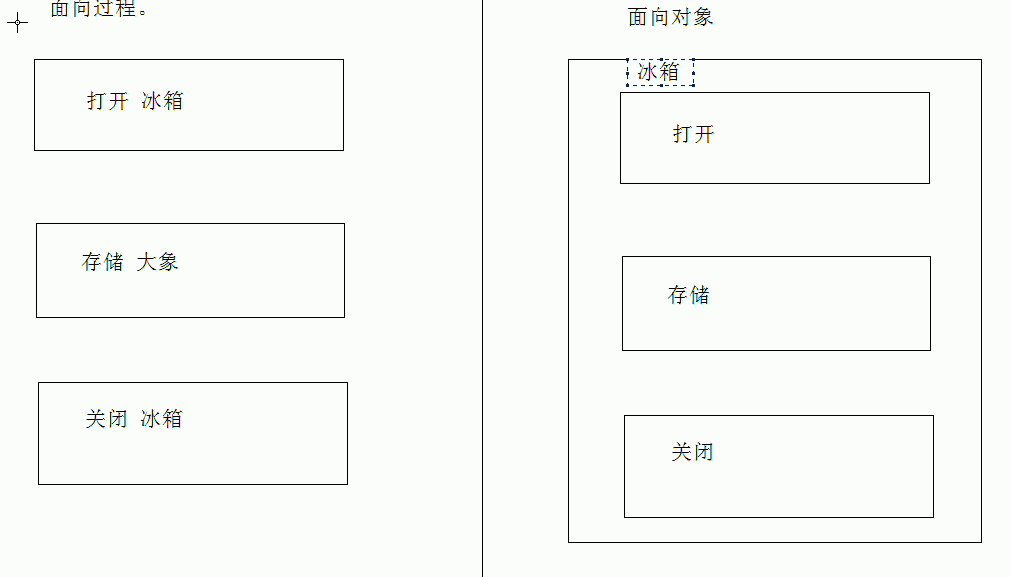
举例：大象装进冰箱。

面向过程：强调的是过程（一系列动作），如：打开冰箱。存储大象。关上冰箱。

C语言。

面向对象。思想：

强调的是对象（实体）。



如：C++ java C#（.com（dot））

特点：

1. 面向对象就是一种常见的思想。符合人们的思考习惯。
2. 面向对象的出现，将复杂的问题简单化。
3. 面向对象的出现，让曾经在过程中的执行者，变成了对象中的指挥者。

课2—3：面向对象\_举例

万物皆可成对象

数据皆有属

有对象，找对象。没有对象，造对象。

名词提炼法。

课4：类与对象之间的关系。

类：用java语音对现实生活中进行描述，通过类的形式体现。

对于事物的描述只关注两个方面。

一个是属性（数据），一个是行为（功能）。

对象：其实就是该类事物实实在在存在的个体。

关系：类，事物的描述；对象：该类事物的实例。在java中通过new来创建。

课5：类与对象的体现。

课6：对象的具体细节。

描述小汽车

分析：

1. 属性。

轮胎数。

颜色。

2， 行为。

运行。

定义类其实就是在定义类中的成员。

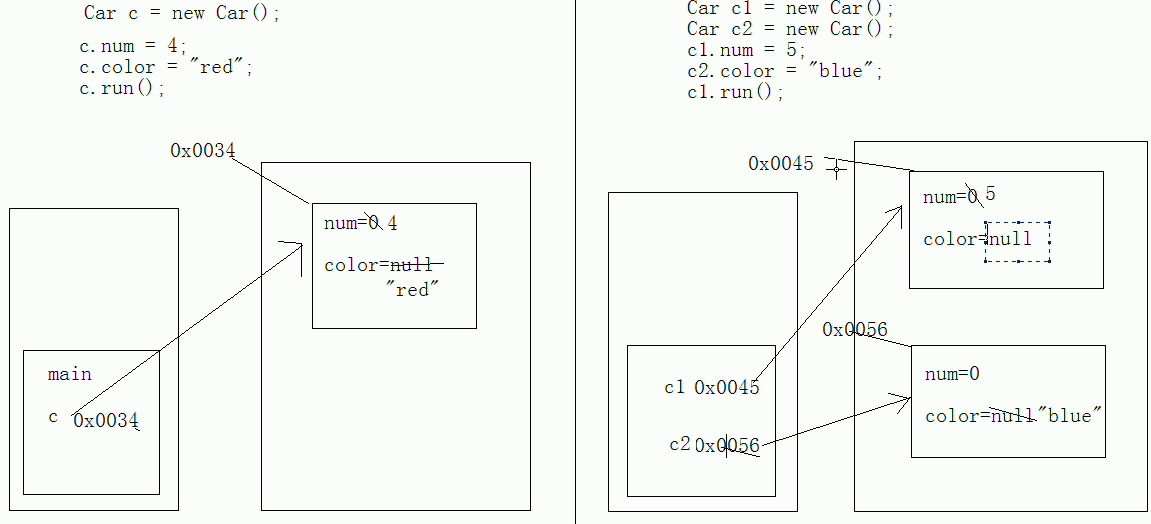
成员： 成员变量<-->属性，成员函数<-->行为。

课7：对象的内存体现。

（int 型初始化0，字符串在堆中，初始化为null）

方法 = 函数

数组 与 对象 ：数组封装同类数据；对象封装某一实体的数据



一旦创建一个对象，那么必具备该类中所描述的内容。

课8：

成员变量和局部变量的区别：

1，

成员变量定义在类中，整个类中都可以访问。

局部变量定义在函数，语句，局部代码块中，只在所属区域有效。

2，

成员变量存在于堆内存的对象中。

局部变量存在于栈内存的方法中。

3，

成员变量随着对象的创建而存在，随着对象的消失而消失。

局部变量随着所属区域而存在，随着所属区域的结束而释放。

4，

成员变量都有默认初始化值。

局部变量没有默认初始化值。

课9：成员变量和局部变量同名&显示初始化（加载）

默认初始化和显示初始化过程（为默认值重新赋值）。

课10：

类类型参数

public static void show(Car c)

//类类型的变量一定指向对象，要不就是null。

课11：匿名对象

/\*

匿名对象，没有名字的对象。

Car c = new Car();

c.run();

new Car().run();

1，当对象对方法仅进行一次调用的时候，就可以简化成匿名对象。

new Car().num = 5;

new Car().color = “green”;

new Car().run;

2，匿名对象可以作为实际参数进行传递。

Car c1 = new Car():

show(c1);

show(new Car());

\*/

Car c = new Car();

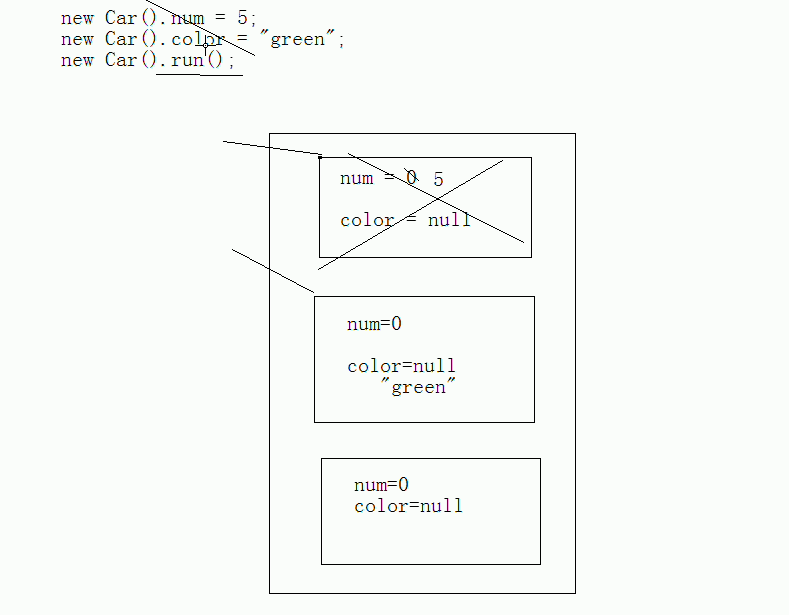
c.run();

c.run();

new Car().run();

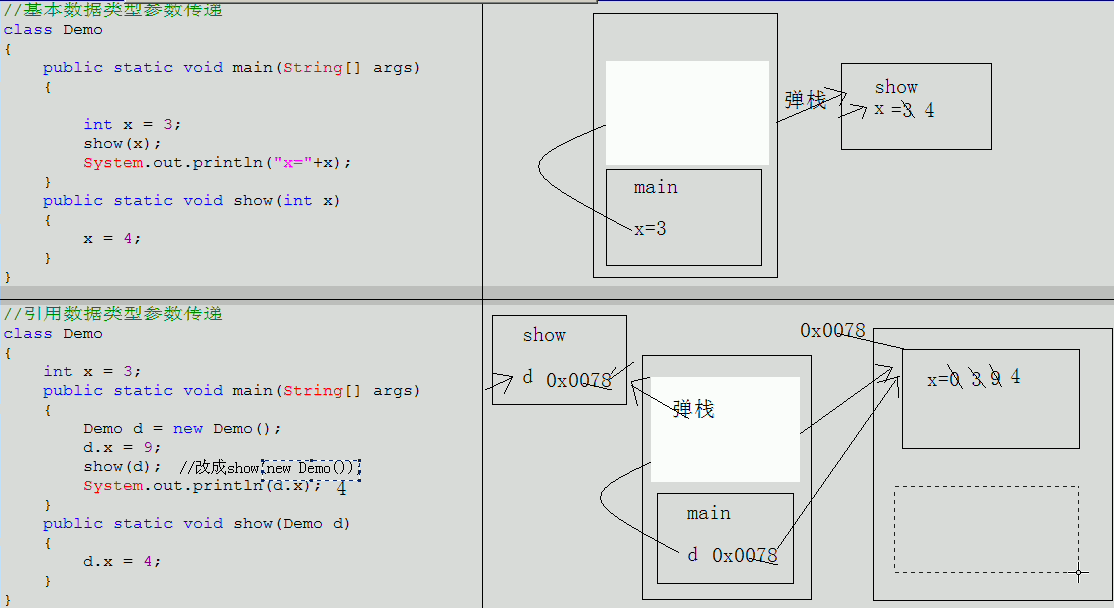
new Car().run();

上述调用是不同性质的，c.run是对同一个对象调用两次；而new Car().run();是对两个对象分别调用一次。



课12：面向对象\_基本数据类型参数传递图解。

1. 压栈（入栈） & 弹栈（出栈） （随所属区域结束后，释放该栈内存）



课14：封装（Encapsulation）

class Person

{

private /\*私有\*/ int age; //私有仅在本类中有效，外界无法访问。

void speak()

{

System.out.println("age = "+age);

}

}

// P.age = -20; //直接访问，造成安全问题，不确定性。

课15：封装-思想

是指隐藏对象的属性和实现细节，仅对外提供公共访问方式。

好处：

将变化隔离。

便于使用。

提高重用性。

提高安全性。

封装原则：

将不需要对外提供的内容都隐藏起来。

把属性（数据）都隐藏，提供公共方法对其访问。

//private:私有

1是一个权限修饰符，

2用于修饰成员。

3私有的成员只在本类中有效。

注意：私有仅仅是封装（隐藏细节）的一种体现而已。（封装如，笔记本，手机；函数；类；框架）便于使用，不需要了解细节。

常用方法之一：将成员变量私有化，对外提供对应的set，get方法对其进行访问。提高对数据访问的安全性。

public void setAge(int a) //setXxx getXxx

{

age = a;

}

public int getAge() //通常，get返回的数据都和int 类型相同。

{

return age;

}

Day07:

课1：构造函数

特点：

1 函数名与类名相同

2 不用定义返回值类型

3 没有具体的返回值

作用：

给对象进行初始化。

注意：

1 默认构造函数的特点。

2 多个构造函数是以重载的形式。

Person()

//构造（构建创造对象时，调用的函数）函数，而且是空参数。

{ //创建对象都必须要通过构造函数初始化。

System.out.println("person run");

}

class \*\*\*

{

public static void main(String [] args)

{

Person p = new Person();

}

}

构造函数：构建创造对象时，调用的函数。

作用：可以给对象初始化。

课2：默认构造函数

一个类中如果没有定义过构造函数，那么该类中会有一个默认的空参数构造函数。

如果在类中定义类指定的构造函数，那么类中的默认构造函数就没有了。

如：class Demo{}，会在编译后，默认自动生成：class Demo{ Demo(){} }

class Demo

{

Demo(){}

Demo()

{

sop{"haha"};

}

}

内部两个Demo 函数是冲突的，一旦Demo手动定义，Demo默认生成的函数会消失。

课3：构造函数和一般函数的区别

构造函数：对象创建时，就会调用与之对应的构造函数，对对象进行初始化。

一般函数：对象创建后，需要函数功能才能调用。

构造函数：对象创建时，会调用只调用一次。

一般函数：对象创建后，可以调用多次。

如： public static void main(String [] args)

{

Person p = new Person(); //构建创造对象时，调用的函数。作用：可以给对象初始化。

//对象创建时，会调用只调用一次。

p.Person(); //已经初始化了，不需要再次调用。

new Person(); 新对象，是可以的。

p.speak();

p.speak(); //对象创建后，可以调用多次。

}

课4：构造函数-重载。

什么时候定义构造函数呢？

在描述事物时，该事物已存在就具备的一些内容，这些内容都定义在构造函数中。

Person() //构造（//构建创造对象时，调用的函数）函数，而且是空参数。

{ //创建对象都必须要通过构造函数初始化。

name = "baby"; //初始化动作

age = 1; //初始化动作

System.out.println("person run");

}

Person(String n)

{

name = n;

}

Person(String n,int m)

{

name = n;

age = m;

}

Person(int m，String n)

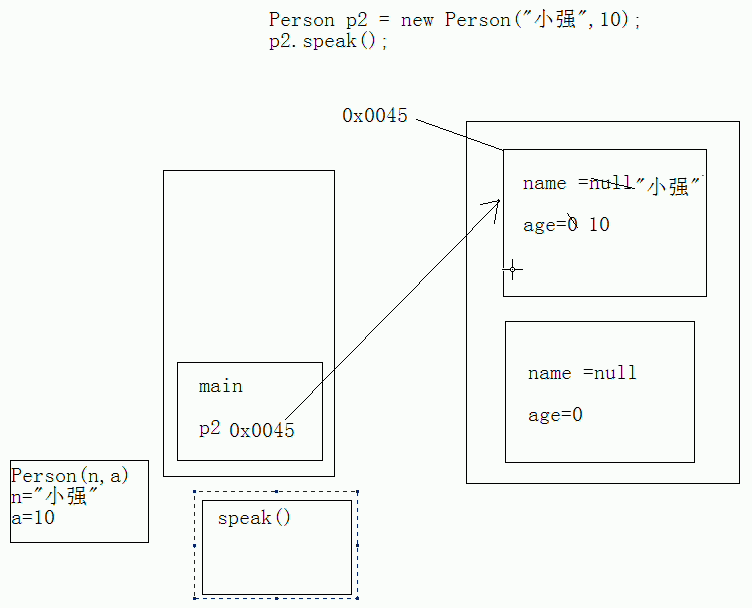
{

name = n;

age = m;

}

课5：构造函数-内存图解。



课6：构造函数-细节。

1 构造函数与一般函数存在的必要性。（构造函数，函数创建时初始化使用；一般函数（一般方法），函数构造后使用。一次与多次的关系）

2 构造函数是给对象初始化的，故不能直接使用该方法。

如，public void speak() {Person(); ... }

3 void person ｛｝：一般函数，可以没有返回值。（一般函数的首字母小写，构造函数首字母大写）

上述一般函数出现了，不能构造函数。如出现void Person(){},,, Person p = new Person();不能构造函数（无法在类Person中找到符号。即不能new对象）

4 return（执行到此结束该函数运行）可以存在。

课7：关键字this，使用场景，this的原理图解。

/\*

当成员变量和局部变量重名，可以用关键字this来区分。

this：代表对象（本类对象）。代表哪个对象呢？当前对象。

this就是所在函数所属对象的引用。

简单说：哪个对象调用了this所在的函数，this就代表哪个对象。

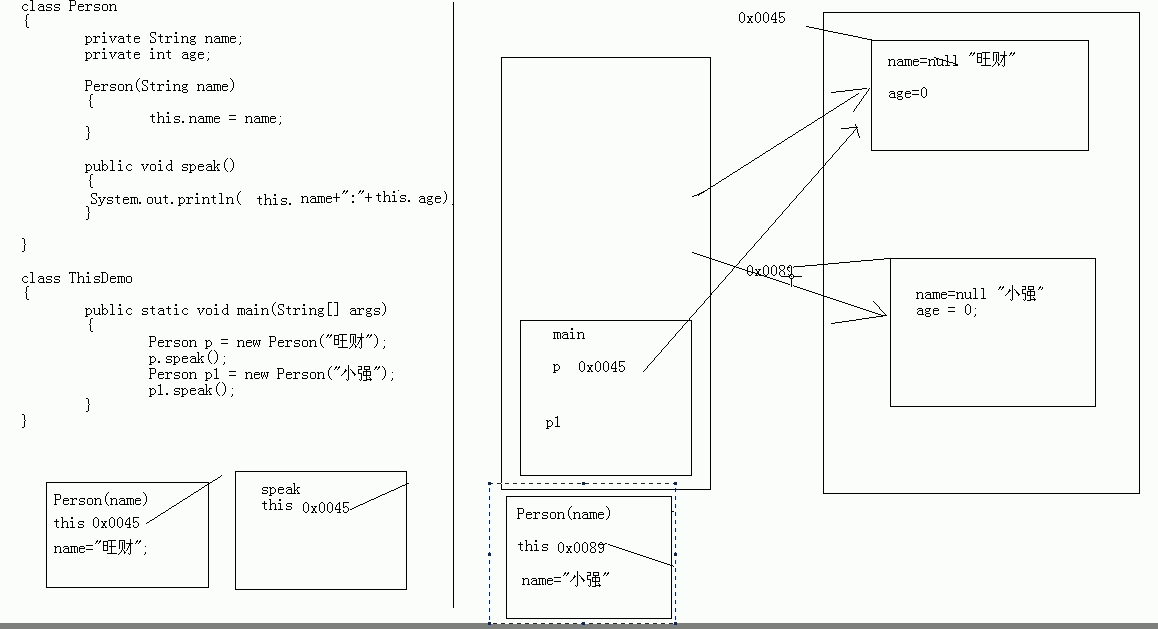
\*/

Person(String name)

{

this.name = name;

}



课8：this-使用场景二以及细节。

一个类中的成员想被执行，必须要有对象调用。

this也可以用于在构造函数中调用其他构造函数。

注意：只能定义在构造函数的第一行。因为初始化工作要先执行。

课9：this-应用

只要在本类中用到了本类的对象，通常都用this（本类对象）。

public boolean compare(Person p)

{

/\*

if(this.age == p.age) //其中，this.age = age

return true;

else

return false;

\*/

return this.age == p.age; //判断语句！

}

p2.compare(p1);

p2是调用compare方法，所以this对象是p2，传入的p1是 Person p类中的p变量。

课10：static关键字-数据共享

课11：static特点

（一）优点：

1 对象所共享的数据。（多个对象共享同一个数据，如，饮水机，公共电话）

2 可以直接受类名调用。

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

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

（二）特点：

1. static 是一个修饰符，用于修饰成员。
2. 被static修饰的成员，被所有的对象所共享。
3. static优先于对象存在，因为static成员随着类的加载就已经存在（而对象不一定存在）。
4. static修饰的成员多了一种调用方式，可以直接被类名所调用。类名.静态成员。
5. static修饰的数据是共享数据，对象中的存储是特有数据。

课12：static关键字-成员变量和静态变量的区别

String name; //成员变量，实例变量

**static** String *country* = "CN"; //静态变量，类变量。

（三）有何区别？

1，两个变量的生命周期不同。

成员变量随着对象的创建而存在，随着对象的被回收而释放。

静态变量随着类的加载而存在，随着类的消失而消失（JVM的消失，类消失）。

1. 调用方式不同。

成员变量只能被对象调用。

静态变量可以被对象调用，还可以被类名调用（最好使用类名调用，专用方式）。

1. 别名不同。

成员变量也称为实例变量。

静态变量称为类变量。

1. 数据存储位置不同。

成员变量数据存储在堆内存的对象中，所以也叫对象的特有数据。

静态变量数据存储在方法区（共享数据区）的静态区，所以也叫对象的共享数据。

课13：static注意事项。

**class** Person

{

String name; //成员变量。实例变量

**static** String *country* = "CN"; //静态变量，类变量。

**public** **static** **void** show()

{

System.*out*.println(*country*+":"+name);

}

}

错误: 无法从静态上下文中引用非静态 变量 name

（四）静态使用的注意事项：

1. 静态方法只能访问静态成员。（非静态既可以访问静态，又可以访问非静态）
2. 静态方法中不可以使用this或者super关键字。
3. 主函数是静态的。

故上面程序改为一下：

**public** **void** show()

{

System.*out*.println(Person.*country*+":"+**this**.name); //静态方法只能访问静态成员。非静态变量省略用this，静态变量省略用类名。

// System.out.println(country+":");

}

将每个代码封装到函数当中，将函数封装到每个类当中，主函数创建对象，调用方法。

课14：static-main函数解析

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

主函数特殊之处：

1. 格式是固定的。
2. 被JVM所识别和调用。

public：因为权限必须是最大的。

static:不需要对象的，直接用主函数所属类名调用即可。

void:主函数没有具体的返回值。

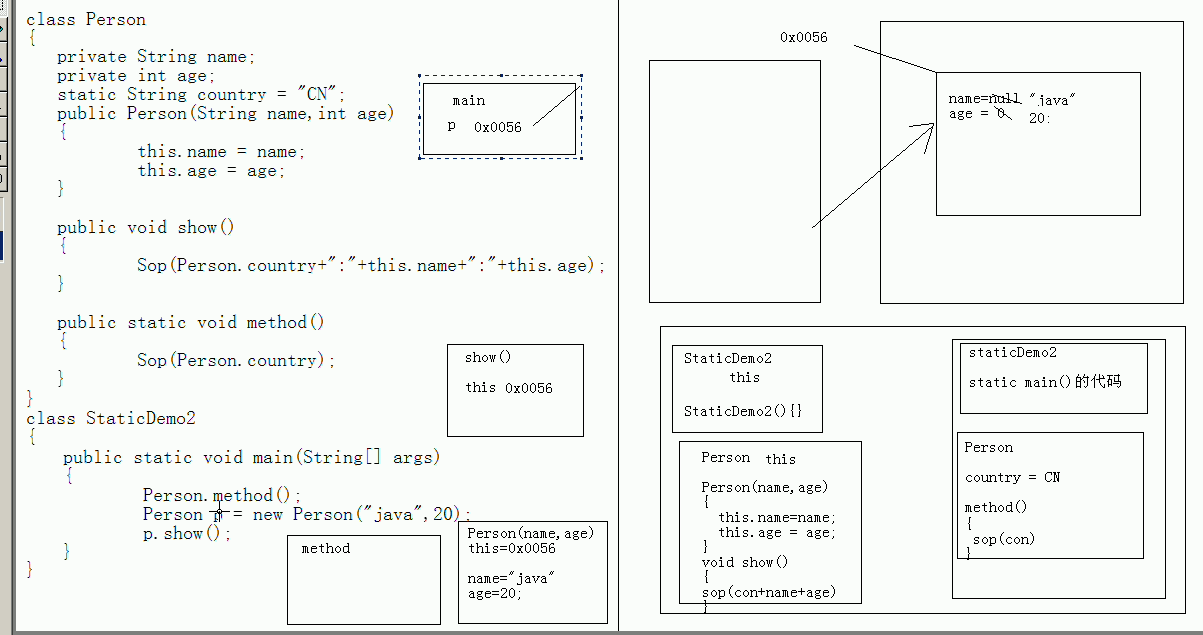
main:函数名，不是关键字，只是一个JVM识别的固定的名字。

String [] args：这是主函数的参数列表，是一个数组类型的参数，而且元素都是字符串类型。

在DOS中运行时，java mainDemo 后 +空格+内容+内容+xxx+xxx（长度等于空格数） 会实现封装，将数据封装到args []数组中。

课15：static-内存图解

//return; 每一个函数都有return语句，只不过有的无返回值，就可以省略不写。



课16：static-什么时候用

静态什么时候用？

1. 静态变量。

（加静态和不加静态的区别：对象中，方法区中；共享和特有）

当分析对象中所具备的成员变量的值都是相同的。

这时，这个成员可以被静态修饰。

只要数据在对象中都是不同的，就是对象的特有数据，不许存储在对象中，是非静态的。如果是相同的数据，对象不需要做修改，只需要使用即可，不需要存储在对象中，定义成静态变量。

1. 静态函数。

函数是否用静态修饰，就参考一点，就是该函数功能是否访问到对象中的特有数据。

简单点说，从源代码看，该功能是否需要访问非静态的成员变量，如果需要，该功能就是非静态。

如果不需要，就可以将该功能定义成静态的。当然，也可以定义成非静态，但是非静态需要被对象调用，而仅创建对象调用非静态的，没有访问特有数据的方法，该对象的创建是没有意义的。

对象是用于封装特有数据的，没有使用这些特有数据，那为什么要创建对象？

课17-static-静态代码块

代码块：if代码块，for代码块，while代码块，static代码块...

随着类的加载而执行。进行初始化，而且只执行一次。

（该类中全是静态变量）

作用：

用于给类进行初始化。

static int num;

static

{

num = 10;

num \*= 3;

System.out.println("haha");

}

主函数被调用才执行。故下面程序，会先输出a。

class staticCodeDemo

{

static

{

System.out.println("a");

}

public static void main(String [] args)

{

//new staticCode().show();

//new staticCode().show();

staticCode().show();

System.out.println("b");

}

}

课18：构造代码块。

构造函数的代码块：对所有调用该类对象时，进行初始化。

共性代码块：给所有对象进行初始化。

特性代码块：给对应的对象进行针对性的初始化。

局部代码块：在一般函数内部的代码块。

{ //构造代码块。可以给所有对象进行初始化。

System.out.println("person run");

cry(); //共性代码块

}

Person() //构造函数是给对应的对象进行针对性的初始化。

{

name = "baby";

//cry(); //特性代码块

}

Person(String name)

{

this.name = name;

//cry();

}

public void cry()

{

System.out.println("哇哇 ");

}

public void speak()

{

System.out.println("name = "+name);

{

//局部代码块

}

}

执行顺序：

静态代码块先执行-->（有对象的前提下）构造代码块-->构造函数

课1：

Dos小技巧：

用Del命令删除\*.class文件，那么后面可以直接通过该类（使用了另外的其他类）编译可以重新获取其他类。

事实上，可以直接在dos中，通过javac \*.java，会编译所有可以编译的.java文件。

课2：数组工具中静态的使用。

//ArrayTool tool = new ArrayTool();

//该对象的创建，浪费堆内存空间，因为没有访问到特有数据。

//所以，ArrayTool中的getMax,需要静态法。

//对ArrayTool中的方法加静态。

//所以要强制，不能创建对象。

故在ArrayTool类中，

private ArrayTool(){}

//该类中的方法都是静态的，所以该类是不需要创建对象的，为了保证不让其他成员创建该类对象。

//可以将构造函数私有化。

课3：数组工具类-文档注释

public作用：提高类的权限；保证了类名和文件名必须一致。

Javadoc（文档注释）：仅提取公有的方法，和注释。

例如：DOS命令中输入 javadoc -d . -author -version xxx.java

设置类路径：Set classpath =.;%classpath%。

课4：单例设计模式-概述&体现

设计模式：对问题行之有效的解决方式，其实它是一种思想。

1，单例设计模式。

解决的问题：可以保证一个类在内存中的对象唯一性。

必须对于多个程序使用同一个配置信息对象时，就需要保证该对象的唯一性。

如何保证对象的唯一性？

1. 不允许其他程序用new创建该类对象。
2. 在该类创建一个本类实例。
3. 对外提供一个方法让其他程序可以获取该对象。

步骤：

1. 私有化该类构造函数。
2. 通过new在本类中创建一个私有类，静态的本类对象。
3. 定义一个公有的静态方法，将创建的对象返回。

**private** **static** Test t = **new** Test();

**private** Test(){}

**public** **static** Test getInstance()

{

**return** t;

}

首先，要使唯一性。必须构造私有的构造函数，故 private Test(){}。

然后，私有的构造函数，不能通过对象访问，故需要用static对getInstance，访问类名.成员，返回t。

由于，静态访问的内容t必须是静态的，故t是静态的，前加static。

课5：单例模式-内存图解

main进栈-（变量S1）->类加载进方法区（S加载入静态区，getInstance()入静态区）-->S初始化为Null-->入堆，在堆中创建对象Single 类对象-->将该对象实例地址赋给静态区的S，S指向该对象地址。

----->getInstance方法进栈-->返回S-->将S1指向返回静态区的S，S指向的是Single创建的对象地址。

课6：单例模式-懒汉式&恶汉式

//恶汉式

**class** Single//类一加载，对象就已经存在了。

{

**private** **static** Single *s* = **new** Single();

**private** Single(){}

**public** **static** Single getInstance()

{

**return** *s*;

}

}

//懒汉式

**class** Single2{

//类加载进来，没有对象，只有调用了getInstance方法时，才会创建对象。

//延迟加载形式。但有可能存在安全问题。不一定是完全单例设置。

**private** **static** Single2 *s* = **null**;

**private** Single2(){}

**public** **static** Single2 getInstance()

{

**if**(*s*==**null**)

*s* = **new** Single2();

**return** *s*;

}

}

课7：

继承（Extends）-概述

**class** Person

{

String name;

**int** age;

}

**class** Student/\*子类（抄类）\*/ **extends** /\*继承\*/ Person /\*父类（基类）\*/

{

**void** study()

{

System.*out*.println(name+"...student study..."+age);

}

}

/\*

继承的好处：

提高了代码的复用性。

让类与类之间产生了关系，给第三个特征多态提供了前提。

\*/

课8：继承-单继承和多重继承

Java中支持单继承。不直接支持多继承，但对C++中的多继承机制进行改良。

单继承：一个子类只能有一个直接父类。

多继承：一个子类可以有多个直接父类（java中不允许，进行改良）。

java认为多继承会带来很多问题，故进行改良。

不直接支持，是因为多个父类中有相同成员，会产生调用不确定性。

不支持多继承，如下：

class A

{

void show()

{

System.out.println("a");

}

}

class B

{

void show()

{

System.out.println("b");

}

}

class C extends B //extends A,B

{

}

java支持多层（多重）继承。

C继承B，B继承A。

就会出现继承体系。

当要使用一个集成体系时，（集合和IO中）

1. 查看该体系中的顶层类，了解该体系的基本功能。
2. 创建体系中的最子类对象，完成该功能的使用。

课9：定义继承

什么时候定义继承呢？

当类与类直接存在着所属关系的时候，就定义继承。

xxx是yyy中的一种：xxx extends yyy

所属关系： is a 关系

课10：子父类中成员变量的特点

/\*

\* 在子父类中，成员的特点体现。

\* 1，成员变量。（当子类中有该变量，就优先选择子类）

\* 2，成员函数。

\* 3，构造函数。

\*/

当本类的成员和局部变量同名用this区分。

当子父类中的成员变量同名用super 区分父类。

this和super的用法很相似。

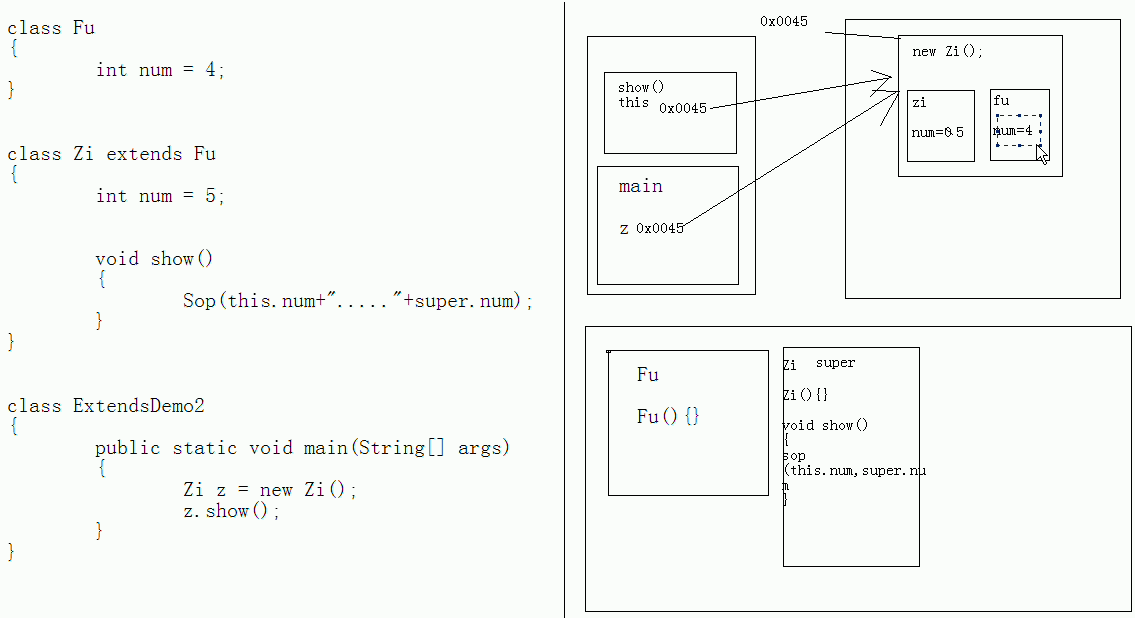
this：代表一个本类对象的引用。

super：代表一个父类空间（不代表父类对象）。

为什么子类能调用父类空间？

（因为子类有super.yyy。）

课11：继承-子父类中成员变量的内存图解



private int num = 4; //错误: num可以在Fu中访问private ..

// 子类对父类中的num是继承过来的

子类不能直接访问父类中私有内容。

间接访问方法！

课12：继承-子父类中成员函数特点-覆盖

当子父类中出现成员函数一模一样的情况，会运行子类的函数。

这种现象，称为覆盖操作。这时，函数在子父类中的特性。

函数两个特性：

1. 重载。同一个类中。 overload
2. 覆盖。子类中。覆盖也成为重写，覆写。overwrite

覆盖注意事项：

1. 子类方法覆盖父类方法时，子类权限必须大于等于父类权限，才可以覆盖。

/\*public\*/ void show() //权限大于子类，不能覆盖 （父类）

void show() （子类）

/\*private\*/ void show()

//private不能用，因为子类无法知道该方法

2，静态只能覆盖静态，或反被静态覆盖。（取决于类，而不是对象）

/\*static\*/void show() （父类）不能覆盖，因为static静态没有对象

static void show() （子类）当子类也是静态，是可以覆盖的。

课13：覆盖的应用

使用继承来提高程序的扩展性。

保留父类原有功能，扩展子类特有内容。

/\*

\* 什么时候使用覆盖操作？

\*

\* 当对一个类进行子类扩展时，子类需要保留父类的功能声明。

\* 但是要定义子类中该功能的特有内容时，就使用覆盖操作完成。

\*/

Day09

课1：子父类中的构造函数-子类的实例化过程。

/\*

子父类中的构造函数的特点。

在子类构造对象，发现，访问子类构造函数时，父类也运行了。

为什么？

原因是：中资料的构造函数中第一行有一个默认的隐式语句。super();

\*/

class Zi extends Fu

{

Zi()

{

//super();//调用的就是父类中的空参数的构造函数。隐式语句。

System.out.println("Zi run");

return; //隐式语句。

}

}

同样，也可以调用非空参数的构造函数。

如：class Fu

{

Fu(int x) //非空参数

{

System.out.println("Fu run");

}

}

class Zi extends Fu

{

Zi()

{

//super();//调用的就是父类中的空参数的构造函数。隐式语句。

super(4);//调用的就是父类中的非空参数的构造函数。

System.out.println("Zi run");

return; //隐式语句。

}

}

问题：

构造函数不能覆盖。

子类的实例化过程：子类中所有的构造函数默认都会访问父类中的空参数的构造函数。

super();功能：访问的是哪个，根据是否是空参数。选择是Fu();,还是Fu(int x);

课2：子类的实例化过程-细节。

为什么子类实例化的时候要访问父类中的构造函数呢？

那是因为子类继承了父类，获取到了父类中内容（属性），所以在使用父类内容之前。

要先看父类是如何对自己的内容进行初始化的。

所以子类在构造对象时，必须访问父类中的构造函数。

为什么完成这个必须的动作，就在子类的构造函数中加入了super();语句。

如果父类中没有定义空参数构造函数，那么子类的构造函数必须用super明确要用父类中那个构造函数。同时，子类构造函数中如果使用this调用了本类构造函数时，那么super就没有了，

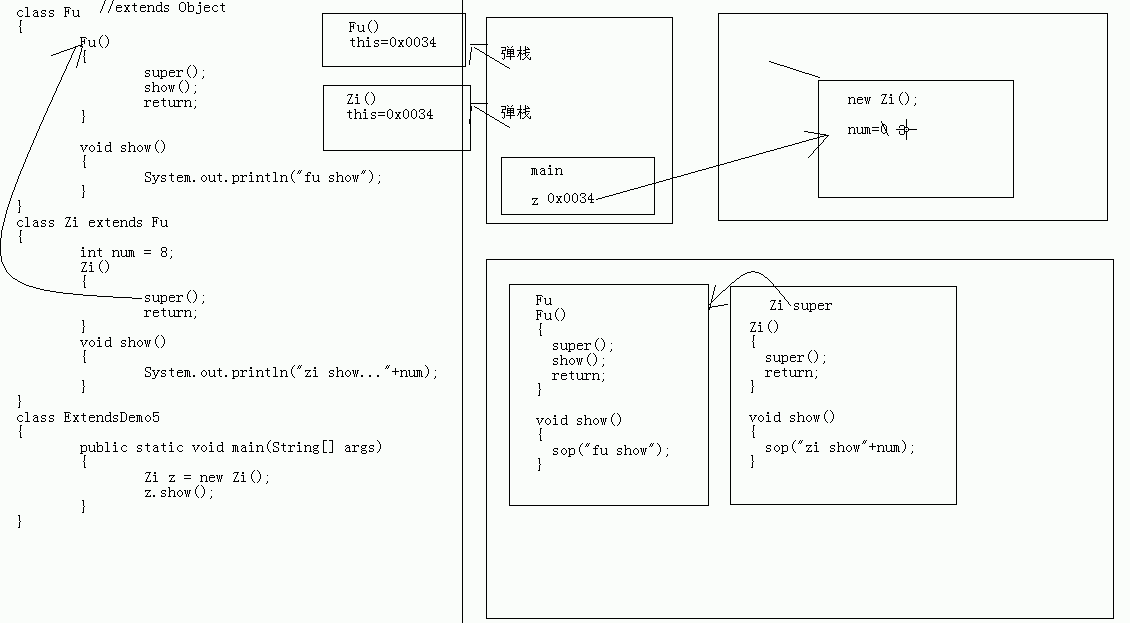
但是可以保证的是，子类中肯定会有其他的构造函数访问父类的构造函数。

注意：super语句必须要定义在子类构造函数的第一行。因为，父类的初始化动作要完成。

课3：子类的实例化过程-图解

**class** Fu **extends** Object

//父类在加载时，并不会先加载Object。因为Object早已经存在，加载过了。



**super**();

//-->通过super初始化父类内容时，子类的成员变量并未显示初始化。等super父类初始化完毕后，

//才进行子类的成员变量显示初始化。

Class Person

{

Person p = new Person();

}

/\*

\* 一个对象实例化过程：

\* Person p = new Person();

\* 1，JVM会读取指定的路径下的Person.class文件，并加载内存。

\* 并会先加载Person的父类（如果有直接的父类的情况下）

\* 2，在堆内存中开辟空间，分配地址。

\* 3，并在对象空间中，对对象中的属性进行默认初始化。

\* 4，调用对应的构造函数进行初始化。

\* 5，在构造函数中，第一行会先调用父类中的构造函数进行初始化。

\* 6，父类初始化完毕后，在对子类的属性进行显示初始化。

\* 7，在进行子类构造函数的特定初始化。

\* 8，初始化完毕后，将地址值赋值给引用变量。

\* \*/

课4：final关键字（最终化修饰符）

final可以修饰类，方法，变量。

//继承弊端：打破了封装性。

子类的方法覆盖了父类，那么父类封装的内容就被破坏。

/\*

\* final关键字：

\* 1，final是一个修饰符，可以修饰类，方法，变量

\* 2，final修饰的类不可以被继承。

\* 3，final修饰的方法不可以被覆盖。

\* 4，final修饰的变量是一个常量，只能赋值一次。

\* 为什么用final修饰变量，其实，在程序中，如果一个数据是固定的，

\* 那么直接使用这个数据就可以了，但是这样阅读性差，所以它该为数据起个名称。

\* 而且这个变量名称的值不能变化，所以加上final固定。

\*

\* 写法规范：常量所有字母都大写，多个单词，中间用\_连接。

\* \*/

static final int x = 7; //固定的是显示初始化。 通常final前都会静态。

/\*final\*/class Fu

{

/\*final\*/void method() //错误: Zi中的method()无法覆盖Fu中的method()

{

//调用了底层系统的资源。

}

}

课5：抽象类

Abstract class Demo{

abstract void show();

}

抽象函数-->抽象类

课6：抽象类特点

/\*

\* 抽象类：

\* 抽象：模糊，笼统，不具体。

\*

\* 特点：

\* 1，方法只有声明没有实现时，该方法就是抽象方法，需要被关键字abstract修饰。

\* 抽象方法必须定义在抽象类中，该类必须也被abstract修饰。

\* 2，抽象类不可以被实例化。为什么？

\* 因为调用抽象方法没有意义。

\* 3，抽象类必须由其子类覆盖了所有的抽象方法后，该子类才可以实例化。

\* 否则，这个子类还是抽象类。

\*

\* \*/

课7：抽象类-细节。

/\*

\*问题：

\*1，抽象类中有构造函数吗？

\* 有，用于给子类对象进行初始化。

\*2，抽象类可以不定义抽象方法吗？

\* 可以的。但是很少见。目的就是不让该类创建对象。AWT的适配器对象就是这种类。

\* 通常这个类中的方法有方法体，但是却没有内容。

\*class Demo

\*{

\* void show1();

\* //void show(){} //有方法体，没有内容。

\* void show2();

\*}

\*3，抽象关键字不可以和哪些关键字共存？

\* private 不行。（抽象方法需要被覆盖，但是private私有后，无法访问）

\* static 不行。

\* final 不行。

\*4，抽象类和一般类的异同点。

\* 相同点：抽象类和一般类都是用来描述事物的，都在内部定义了成员。

\* 不同点：

\* 1，一般类有足够的信息描述事物。

\* 抽象类描述事物的信息有可能不足。

\* 2，一般类中不能定义抽象方法，只能定义非抽象方法。

\* 而抽象类中可以定义抽象方法，同时，也可以定义非抽象方法。

\* 3，一般类可以被实例化。

\* 抽象类不可以被实例化。

\*5，抽象类一定是个父类吗？

\* 是的。因为需要子类覆盖其方法后才可以对子类实例化。

abstract class Demo

{

abstract void show();

}

abstract class SubDemo extends Demo

{

}

class D extends SubDemo

{

}

\* \*/

课8：抽象类-练习

课9：接口-定义

/\*

abstract class AbsDemo

{

abstract void show1();

abstract void show2();

}

\* 当一个抽象类中的方法都是抽象的时候，这时可以将该抽象类用

\* 另一个形式定义和标识，就是接口 interface。

\*

//定义接口使用的关键字不是class，是interface。

\*对于接口当中常见的成员：而且这些成员都有固定的修饰符。

\*1，全局常量：public static final

\*2，抽象方法:public abstract

\*

\*由此得出结论，接口中的成员都是公共的权限。

\*/

interface Demo

{

public static final int Num = 4;

public abstract void show1();

public abstract void show2();

}

课10：接口-实现implements

//类与类之间是继承关系，类与接口之间是实现关系。

/\* 接口不可以实例化。

\* 只能由实现了接口的子类并覆盖了接口中所有的抽象方法后，该子类才可以实例化。

\* 否则，这个子类就是一个抽象类。

\* \*/

class DemoImpl implements Demo

{

public void show1(){}

public void show2(){}

}

课11：接口-多实现

/\*

\* 在java中不直接支持多继承，因为会出现调用的不确定性。

\* 所以java将多继承机制进行改良，在java中变成了多实现。

\*

\* 一个类可以实现多个接口。

\* \*/

interface A

{

public void show();

}

interface Z

{

//public void show();

//public int show(); //不可以实现。

//public int add(int a,int b);

}

class Test implements A,Z //多实现

{

/\*

public int add(int a,int b)

{

return a+b+3;

}

\*/

}

课12：接口-细节

/\*

\* 一个类在继承另一个类的同时，还可以实现多个接口。

\* \*/

class Q

{

public void method()

{}

}

abstract class Test2 extends Q implements A,Z

{

}

/\*

\* 接口的出现避免了单继承的局限性。

\* \*/

interface CC

{

void show();

}

interface MM

{

void method();

}

interface QQ extends CC,MM //接口与接口之间是继承关系，而且接口可以多继承。

{

void function();

}

class WW implements QQ

{

//覆盖3个方法。

public void show(){}

public void method(){}

public void function(){}

}

课13：接口-特点

接口是对外暴露的规则。

接口是程序的功能扩展。

接口的出现降低耦合性。（网卡与主板之间，鼠标与电脑之间）

接口可以用来多实现。

类与接口之间是实现关系，而且类可以继承一个类的同时实现多个接口。

接口与接口之间可以有继承关系。

课14：接口与抽象类的区别

/\*

\* 抽象类和接口的异同：

\*

\* 相同点：

\* 都是不断向上抽取而来的。

\*

\* 不同点：

\* 1，抽象类需要被继承，而且只能单继承。

\* 接口需要被实现，而且可以多实现。

\* 2，抽象类中可以定义抽象方法和非抽象方法，子类继承后，可以直接使用非抽象方法。

\* 接口中只能定义抽象方法，必须由子类去实现。

\* 3，抽象类的继承，是 is a 关系，在定义该体系的基本共性内容。

\* 接口的实现是 like a 关系，在定义体系额外功能，

\*

\* 犬按功能分：有导盲犬，搜爆犬。

\*

\* class 犬

\* {

\* abstract void 吼叫（）；

\* }

\* //abstract class 导盲

\* interface 导盲

\* {

\* abstract void 导盲（）；

\* }

\* class 导盲犬 extends 犬 implements 导盲

\* {

\* public void 吼叫（）{}

\* public void 导盲（）{}

\* }

\* //在不同的问题领域中，有不同的分析方式。

\* \*/

day10:

课1：接口的应用：

课2：

多态-概述

定义：某一类事物的多种存在形态。

/\*

对象的多态性。

Class 动物

{}

Class 猫 extends 动物

{}

Class 狗 extends 动物

{}

猫 x = new 猫();

动物 x = new 猫(); //一个对象，两种形态。

猫这类事物既具备猫的形态，又具备动物的形态。

这就是多态性。

简单说：就是一个对象对应着不同类型。

多态在代码中的体现：

父类或者接口的引用指向其子类的对象。

多态的好处：

提高了代码的扩展性，前期定义的代码可以使用后期的内容。

\*/

Class Animal

{

Abstract void eat();

}

Class Dog extends Animal

{

void eat()

{

System.out.println(“啃骨头”);

}

void lookHome()

{

System.out.println(“看家”);

}

}

Class Cat extends Animal

{

void eat()

{

System.out.println(“吃鱼”);

}

void catchMouse()

{

System.out.println(“捉老鼠”);

}

}

课4：多态-弊端&前提

多态的弊端：

前期定义的内容不能使用(调用)后期子类的特有内容。

多态的前提：

1，必须有关系，继承，实现。

2，要有覆盖。

课5：多态-转型

Animal a = **new** Cat();

//自动类型提升，猫对象提升了动物类型。但是特有功能无法访问。

// 作用就是限制对特有功能的访问。

//专业讲：向上转型。将子类隐藏，就不用使用子类的特有方法。

a.eat();

//如果还想用具体动物猫的特有功能。

//你可以将该对象进行向下转型。

//向下转型的目的是为了使用子类中的特有方法。

Cat c = (Cat)a;

c.eat();

c.catchMouse();

//注意：对于转型，自始自终都是子类对象在做着类型的变换。

Animal a1 = **new** Dog();

Cat c1 = (Cat)a1; //java.lang.ClassCastException

c1.eat();

注意：（类型转换异常）：狗不能被转换成猫。

Exception in thread "main" java.lang.ClassCastException: Dog cannot be cast to Cat at DuoTaiDemo.main(DuoTaiDemo.java:96)

课6：多态-转型2

课7：多态-类型判断-instanceof

public static void method(Animal a)// Animal a = new Dog();

{

//通常在向下转型前用于健壮性的判断。

健壮性判断延伸！

a.eat();

//a.catchMouse();

//弊端：前期定义的内容不能使用(调用)后期子类的特有内容。

if(a instanceof Cat)

//instanceof:用于判断对象的具体类型。只能用于引用数据类型。

{

Cat c = (Cat)a;

c.catchMouse();

}

else if(a instanceof Dog) //instanceof:用于判断对象的具体类型。只能用于引用数据类型。

{

Dog d = (Dog)a;

d.lookHome();

}

健壮性判断网上查找内容解释：

查找条件有员工工号（字符串），人员姓名（汉字），手机号码（字符串），岗位，工作职责，固定电话。

六个查询条件拼接成一个对象QueryCondition传到后台。

后台有一个方法 search（QueryCondition q）

不健壮的程序就是search 不管q为什么直接调用相应方法到数据库中查询。这样可能就有一大堆的空指针异常和一些奇奇怪怪的错误。那么要怎么做呢，处理你能想到的所有异常情况。

1.没有判断对象是否为空。----空指针异常

什么条件都没有，是不是就意味着查询所有数据。这样数据量小还勉强可以，要是数据量百万级，这样，你要么请求超时，要么内存溢出。

处理：程序中首先对q进行判断是否为空。接着对q中所有属性进行判断。如果全部为空。要么不让查询，要么调用一个方法分页查询。

2.字符串长度没有限制。------测试故意找茬

遇到过一个测试在姓名字段输入5000+汉字。具体多少不记得了，结果：这个查询调用的是一个存储过程，参数定义是varchar2（10）. 不产生异常就怪了。请不要说这是测试变态，这就是程序健壮性。其实我们在前台输入的时候对长度进行限制或者后台判断有没有超过某些程序中限制长度，建议前台做，比较简单。

3.数字限制-----不符合实际情况的输入

岗位是一个数字1-6级，根据岗位去匹配相应基本工资，岗位为除数。理论上岗位不会有负数和0.所以很多时候就不加处理。于是，当你输入0的时候就产生了一个ArithmaticException。好吧，我什么都不说了，怪自己。

4.字符串与数字之间的转换。----大坑

如果你输入手机号码，正常情况应该是全部数字。假设数据库中该列类型就是number类型，为了提高查询速度，所以在查询前会将字符串转换为数字。这就给自己埋了一个大坑。如果用户输入013123568978，或者-4654654654这些都是无法转换的。有时候我们不加判断就直接转换，程序不抛异常就怪了。

贴一个不那么常见的异常

1.邮件发送----内存溢出

for（//获取收件人信息

for（）//生成附件并发送

）

这是一个两重循环。生成附件方法生成的附件先放在内存中，邮件发送出去就回收。

刚刚开始发送几十封，附件10m以内一点问题没有。以为该功能已经实现。不过不得不说测试牛，造了一个好几g的附件。这下够服务器喝一壶的了，内存溢出了。修复吧，把附件大小限制为10m以内。以为雨过天晴。没想到，最多一次发送4000+封邮件。为了加快发邮件速度，我还自以为聪明的使用了多线程。这次又溢出。

这此事故让我对内存产生了敬畏，不要以为内存永远够用。这个怎么避免我还没有完美的解决方案。最后我是牺牲了性能

2.接口限制--调用次数，资源阻塞

这种异常是我遇到的比较坑爹的。不是自己写的代码原因，但是还是会导致我的程序挂了。

有一次做一个地图类应用，调用的是xx的免费接口。程序一切正常，有一天我在吃着泡面，运维告诉你，屌丝，你的应用挂了。我的第一反应是不可能，都用了这么久了。速度吃掉老坛酸菜，看日志，xxx接口访问限制。我去，神马情况，一直好好的，咔咔开始调试。搞了半天搞不出来，实在没辙了，咨询了一下xx技术支持。才知道对于请求量2000+的ip是会被封掉的。不吐槽，免费的就是坑爹的。不过这种异常只能遇到一次解决一次。没有规律可循

总结一下：程序健壮性就是你的程序在遇到异常的情况下还能运行或者给予用户友好的提示。所以写代码的时候尽量考虑周全。该做异常处理的就做异常处理，性能和正确性比起来一文不值。

最后链接下面的健壮性博文，进一步理解健壮性：

https://www.cnblogs.com/zhuispeed/archive/2010/07/08/1773893.html

课8：多态-成员变量

课9：多态-成员函数

课10：多态-静态函数

/\*

\* 多态时，

\* 成员的特点：

\* 1，成员变量。

\* 编译时，参考引用型变量所属的类中的是否有调用的成员变量。有，编译通过；没有，编译失败。

\* 运行时，参考引用型变量所属的类中的是否有调用的成员变量，并运行该所属类中的成员变量。

\* 简单说，编译和运行都参考左边，哦了。

\* 作为了解。

\*

\* 2，成员函数（非静态）。

\* 编译时，参考引用型本类所属的类中，是否有调用函数。有，编译通过；没有，编译失败。

\* 运行时，参考的是对象所属的类中是否有调用的函数。

\* 简单说，编译看左边，运行看右边。

\*

\* 3，静态函数。

\* 编译时，参考引用型本类所属的类中，是否有调用的静态方法。

\* 运行时，参考引用型本类所属的类中，是否有调用的静态方法。

\* 简单说，编译和运行都看左边。

\*

\* 其实对于静态方法，是不需要对象的。直接用类名调用即可。

\* \*/

public static void useUSB(USB u) //接口类型的引用，用于接收（指向）接口的子类对象。// USB u = new UPan();

{ //事实上，就是多态的使用。

if(u != null)

{

u.open();

u.close();

}

}

}

课11：内部类-概述

将一个类定义在另一个类的里面，对里面那个类就称为内部类（内置类，嵌套类）。

class InnerClassDemo

{

public static void main(String[] args)

{

// Outer out = new Outer();

// out.method();

//直接访问外部类中的内部类中的成员。

// Outer.Inner in = new Outer().new Inner();

// in.show();

//如果一个内部类是静态的，那么相当于一个外部类。

// Outer.Inner in = new Outer.Inner();

// in.show();

//如果内部类是静态的，成员是静态的。

Outer.Inner.function();

}

}

课12：内部类-修饰符

private、static、final都可以用来修饰内部类。而外部类不可以用这些修饰符修饰。

class Outer{

/\*private\*/ /\*static\*/ class Inner

{

static void function()

//如果内部类中定义了静态成员，该内部类也必须是静态的。

{

System.out.println("fuction run..."+num);

}

}

}

//直接访问外部类中的内部类中的成员。

记住该调用形式：// Outer.Inner in = new Outer().new Inner();

//如果一个内部类是静态的，那么相当于一个外部类。

// Outer.Inner in = new Outer.Inner();

//如果内部类是静态的，成员是静态的。

Outer.Inner.function();

课13：内部类-细节

/\*

\* 为什么内部类能直接访问外部类中的成员呢？

\* 那是因为内部类持有了外部类的引用。 外部类名.this

\*

\* \*/

class Outer

{

int num = 3;

class Inner

{

int num = 4;

void show()

{

int num = 5;

System.out.println(Outer.this.num);

//Inner.this.num = this.num =3;

//Outer.this.num = 3;

}

}

课14：内部类-局部内部类

/\*

\* 内部类可以存放在局部位置上。

\*

\* 内部类在局部位置上只能访问局部中被final修饰的局部变量。

\* \*/

class Outer

{

int num = 3;

Object method()

// Object method(final int y)

{

final int x = 9;

class Inner

{

void show()

{

//System.out.println("show..."+num);

System.out.println("show..."+y);

}

}

Object in = new Inner();

return in;

// in.show();

}

}

class{

Xxx(){

Object obj = out.method();

}

}

课15：匿名内部类-概述

/\*

\* 匿名内部类，就是内部类的简写格式。

\*

\* 必须有前提：

\* 内部类必须继承或者实现一个外部类或者接口。

\*

\* 匿名内部类：其实就是一个匿名子类对象。

\*

\* 格式：new 父类or接口(){子类内容}

\* \*/

将下列内部类转成匿名内部类

/\*

class Inner extends Demo

{

void show()

{

System.out.println("show..."+num);

}

}

\*/

new Demo() //匿名内部类。

{

void show()

{

System.out.println("show........."+num);

}

}.show();// new Inner().show();

课16：匿名内部类-应用

// Inner in = new Inner();

// in.show1();

// in.show2();

Inter in = new Inter()

{

public void show1()

{

}

public void show2()

{

}

};

in.show1();

in.show2();

\* 通常的使用场景之一：

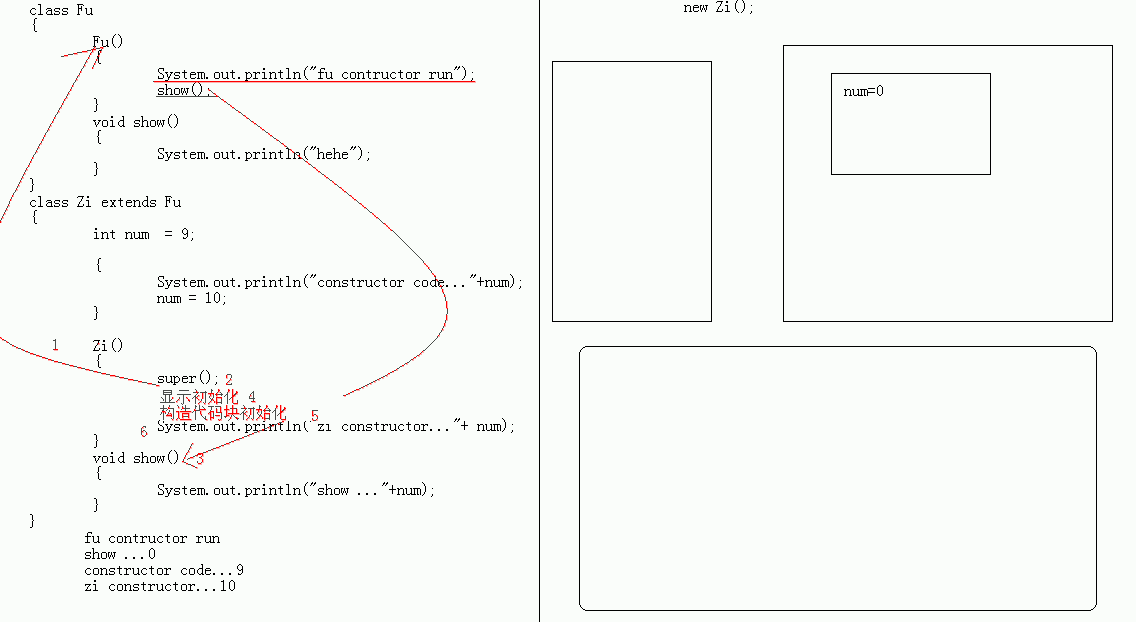
\* 当函数参数是接口类型时，而且接口中的方法不超过三个。

\* 可以用匿名内部类作为实际参数进行传递。

\* \*/

课17：匿名内部类-细节

课18：对象的初始化过程



Day11:

课1：Object类-equals方法

/\*

Object：所有类的根类。

Object是不断抽取而来，具备着所有对象都具备的共性内容。

常用的共性功能：

\*/

**public** **class** Object {

**private** **static** **native** **void** registerNatives();

**static** {

*registerNatives*();

}

调用本地方法。（native）

Person p1 = new Person(20);

Person p2 = new Person(20);

Person p3 = p1;

System.out.println(p1 == p2); //比较地址

System.out.println(p1.equals(p2)); //比较地址

System.out.println(p1.equals(p3)); //比较地址

==与equals的区别：

首先，eqauls()方法是超类Object中的方法，而Java中所有的对象都是继承自Object类的，所以子类是可以重写eqauls()方法而实现不同的功能。

对于任何非空的参考值x和y ，当且仅当x和y引用相同的对象（ x == y具有值true ）时，该方法返回true 。

课2：Object类-equals方法覆盖

/\*

\* 一般都会覆盖此方法，根据对象的特有内容，建立判断对象是否相同的依据。

\* \*/

public boolean equals(Object obj) // Object obj = p2;

{

if(!(obj instanceof Person))

{

// return false;

throw new ClassCastException("类型错误");

}

Person p = (Person)obj;

return this.age == p.age;

}

课3：Object类-hashCode方法

System.out.println(p1.hashCode()); // 778966024

System.out.println(Integer.toHexString(p1.hashCode()));// 2e6e1408

课4：Object类-getClass方法

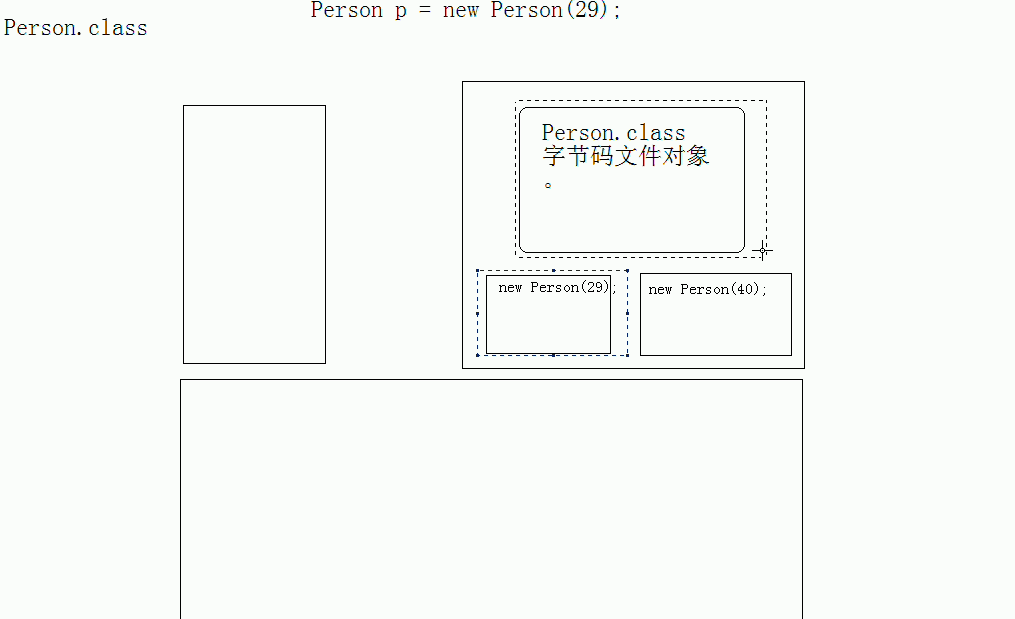
Class classz1 = p1.getClass();

Class classz2 = p2.getClass();

System.out.println(classz1 == classz2);

System.out.println(classz1.getName());

返回此Object的运行时类。表示 类对象的运行时类的Class对象。



课5：Object类-toString方法

//Object类-toString方法

public String toString()

{

return "Person :"+age;

}

课1：异常（exception）-概述

/\*

\* 异常：是在运行时期发生的不正常情况。

\*

\* 在Java中用类的形式对不正常情况进行了描述和封装对象。

\*

\* 描述不正常的情况的类，就称为异常类。

\*

\* 以前称正常流程代码和问题处理代码相结合，

\* 现在将正常流程代码和问题处理代码分离。提高阅读性。

\*

\* 其实异常就是java通过面向对象的思想将问题封装成了对象。

\*

\* 用异常类对其进行描述。

\* 不同的问题用不同的类进行具体的描述。比如角标越界。空指针等。

\* \*/

课2：异常-体系

问题很多，意味着描述的类也很多，将其共性进行向上抽取，形成了异常体系。

\* 最终问题（不正常）就分成了两大类。

\* Throwable:无论是error，还是异常，问题，问题发生就应该可以抛出，让调用者知道并处理。

\* //该体系的特点就在于throwable及其所有的子类都具有可抛性。

\* 可抛性到底指的是什么呢？怎么体现可抛性呢？

\* 其实是通过两个关键字来体现的。

\* throws thorw ，凡是可以被这两个关键字所操作的类和对象都具备可抛性。

\* |--1，一般不可处理的。Error

\* 特点：是由JVM抛出的严重性的问题。

\* 这种问题发生一般不针对性处理，直接修改程序。

\*

\* |--2，可以处理的。Exception

\*

\* 该体系的特点：

\* 子类的后缀名都是用其父类名作为后缀，阅读性很强。

\* \*/

int [] arr = new int[1024\*1024\*800]; //java.lang.OutOfMemoryError:

// arr = null; //java.lang.NullPointerException

// System.out.println(arr[3]); //java.lang.ArrayIndexOutOfBoundsException:

课3：异常-原理&异常对象的抛出throw

正常我们在控制台上受到的异常信息，我们都会收集作为日记（时间，具体信息）。

如：

H:\java\myeclipes\Workspaces\day11\src>java ExceptionDemo2

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3

at Demo.method(ExceptionDemo2.java:6)

at ExceptionDemo2.main(ExceptionDemo2.java:17)

if(arr == null){

throw new NullPointerException("数组的引用不能为空！");

}

if(index >= arr.length){

throw new ArrayIndexOutOfBoundsException("数组的角标越界了，哥们，你是不是疯了"+index);

}

\* \*/

课4：异常-原理&异常对象的抛出throws

错误: 未报告的异常错误FuShuIndexException; 必须对其进行捕获或声明以便抛出

编译问题: 1 如果函数发生异常，需要在函数上声明，否则编译失败。

2 调用了声明异常的函数，必须处理。

处理之一：抛出

例如，发生以下非子类的异常，可以进行声明抛出：

if(index < 0)｛

throw new FuShuIndexException("角标为负数啦！"); // 错误: 未报告的异常错误FuShuIndexException; 必须对其进行捕获或声明以便抛出

}

处理：public 类型 xxx()throws FuShuIndexException //对该方法声明抛出

课5：异常-编译时检测异常和运行时异常的区别&throw和throws的区别

/\*

\* 对于角标是整数不存在，可以用角标越界表示，

\* 对于负数为角标的情况，准备用负数角标异常来表示。

\*

\* 负数角标这种异常在java中并没有定义过。

\* 那就按照java异常的创建思想，面向对象，将负数角标进行自定义描述，并封装成对象。

\*

\* 这种自定义的问题描述成为自定义异常。

\*

\* 注意：如果让一个类成为异常类，必须要继承异常体系，因为只有成为异常体系的子类才有资格具备可抛性。

\* 才可以被两个关键字所操作， throws throw

异常的分类：

1，编译时被检测异常：只要是Exception和其子类都是，除了特殊子类RuntimeException体系。

这种问题一旦出现，希望在编译时就进行检测，让这种问题有对应的处理方式。

这样的问题都可以针对性的处理。

1. 编译时不检测异常（运行时异常）：就是Exception和RuntimeException和其子类。

这种问题的发生，无法让功能继续，运算无法进行，更多是因为调用者原因导致的或者引发了内部状态的改变导致的。（多线程会出现）

那么这种问题一般不处理，直接编译通过，在运行时，让调用者调用时的程序强制停止，让调用者对代码进行修正。

所以自定义异常时没要么继承Exception，要么继承RuntimeException。

throw 和 throws 的区别。

\* 1，throws 使用在函数上，

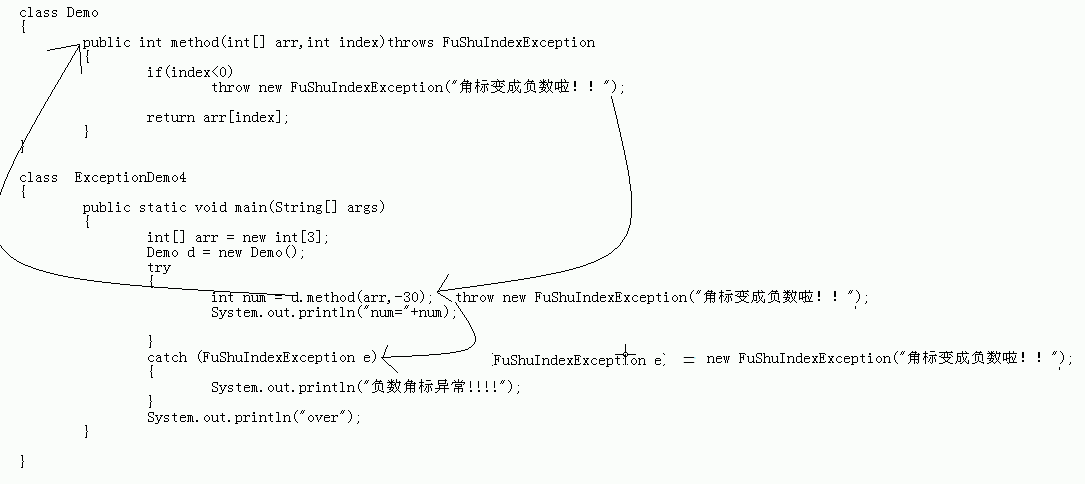
\* throw 使用在函数内。

\* 2，throws抛出的是异常类，可以抛出多个，用逗号隔开。

\* throw抛出的是异常现象。

\* \*/

课6：异常-异常捕捉trycatch



课7：异常-多catch情况

class FuShuIndexException extends Exception

{

FuShuIndexException()

{

}

FuShuIndexException(String msg)

{

super(msg); //爹都给办完了，直接找爹

}

}

class Demo

{

public int method(int [] arr, int index)throws FuShuIndexException,NullPointerException

{

if(arr == null)

throw new NullPointerException("没有任何数据实体");

if(index < 0)

{

throw new FuShuIndexException("角标为负数啦！"); // 错误: 未报告的异常错误FuShuIndexException; 必须对其进行捕获或声明以便抛出

}

return arr[index];

}

}

class ExceptionDemo4

{

public static void main(String[] args)

{

int [] arr = new int [3];

Demo d = new Demo();

try

{

// int num = d.method(arr,-30);

int num = d.method(null,-30);

System.out.println("num = "+num);

}

catch(NullPointerException e)

{

System.out.println("String : "+e.toString());

}

catch(FuShuIndexException e)

{

System.out.println("message : "+e.getMessage());

System.out.println("String : "+e.toString());

e.printStackTrace(); //jvm默认的异常处理机制。

System.out.println("角标为负数啦！异常");

}

/\*

catch(Exception e) //多catch父类的catch放在最下面。否则，编译失败。

{

}

\*/

System.out.println("over");

}

}

课8：异常-异常处理规则

\* 异常处理的原则：

\* 1，函数内容如果抛出需要检测的异常，那么函数上必须要声明。

\* 否则必须在函数内用trycatch捕捉，否则编译失败。

\*

\* 2，如果调用到了声明异常的函数，要么trycatch要么throws，否则编译失败。

\*

\* 3，什么时候catch，什么时候throws呢？

\* 功能内容可以解决，用catch。

\* 解决不了，用throws告诉调用者，由调用者解决。

\*

\* 4，一个功能如果抛出多个异常，那么调用时，必须有对应多个catch进行针对性的处理。

\* 内部有几个需要检测的异常，就抛几个异常，抛出几个，就catch几个。

课9：异常-finally代码块

// System.exit(0);

//退出JVM。只有exit可以直接退出，而不直接finally。

class ExceptionDemo5

{

public static void main(String[] args)

{

Demo d = new Demo();

try

{

int num = d.show(-3);

System.out.println("num = "+num);

}

catch(ArrayIndexOutOfBoundsException e)

{

System.out.println(e.toString());

// return;

// System.exit(0); //退出JVM。

}

finally //通常用于关闭（释放）资源。

{

System.out.println("finally");

}

System.out.println("over");

}

}

//连接数据库

//查询。Exception

//关闭连接。

/\*

\* try catch finally 代码块组合特点：

\* 1，

\* try catch finally

\*

\* 2,

\* try catch(多个) 当没有必要资源需要释放时，可以不用定义finally。

\*

\* 3，

\* try finally 异常无法直接catch处理，但是资源需要关闭。

\* 4,

\* void show()throws new Exception

\* {

\* try

\* {

\* //开启资源。

\* throw new Exception();

\*

\* }

\* finally

\* {

\* //关闭资源。

\* }

\* }

\* \*/

课10：异常-异常应用

课11：异常-注意事项

\* 异常的注意事项：

\* 1，子类在覆盖父类方法时，父类的方法如果抛出了异常，

\* 那么子类的方法只能抛出父类的异常或者该异常的子类。

\*

\* 2,如果父类抛出多个异常，那么子类只能抛出父类异常的子集。

\*

\* 简单说，子类覆盖父类只能抛出父类的异常或者子类或者子集。

\*

\* 注意：如果父类的方法没有抛出异常，那么子集覆盖时绝对不能抛,就只能try。

\*

\*

\* \*/

interface Inter

{

void function();

}

class D implements Inter

{

public void function()// throws Exception

}

Day12:

课1：包（package）-概述

对类文件进行分类管理。（文件夹 folder）

给类提供多层命名（名称）空间。（包名全为小写字母）

卸载程序文件的第一行。

类名的全程是 包名.类名。

包也是一种封装形式。

**package** mypack.haha.hehe.xixi;

编译 javac -d . Package.java ; javac -d c:\myclass DemoA.java

运行 java mypack.haha.hehe.xixi.PackageDemo

课2：包-包之间访问-Protected（保护）

四种权限大小使用：

public protected default private

同一类中 OK OK OK OK

同一类中 OK OK OK

子类中 OK OK

不同包中 OK

课3：包-导入import

import packa.DemoA; //导入了packa包中的DemoA类。

//import packa.\*; //导入了packa包中所有的类。 \*通配符

//import packa.abc.\*;

//pack\DemoA.class

//pack\DemoA\DemoAbc.class

//导包的原则：用到哪个类，就导入哪个类。

//import 干嘛用的啊？ 为了简化类名书写。

课4：jar包

jar -cf haha.jar pack （创建一个jar文件包）

jar -cvf haha.jar pack

jar -tvf haha.jar

jar -xvf haha.jar创建解压文件。（MANIFEST：压缩包的清单文件。）

Set classpath = ./haha.jar

Java pack.jarDemo

用法: jar {ctxui}[vfmn0PMe] [jar-file] [manifest-file] [entry-point] [-C dir] files ...

选项:

-c 创建新档案

-t 列出档案目录

-x 从档案中提取指定的 (或所有) 文件

-u 更新现有档案

-v 在标准输出中生成详细输出

-f 指定档案文件名

-m 包含指定清单文件中的清单信息

-n 创建新档案后执行 Pack200 规范化

-e 为捆绑到可执行 jar 文件的独立应用程序

指定应用程序入口点

-0 仅存储; 不使用任何 ZIP 压缩

-P 保留文件名中的前导 '/' (绝对路径) 和 ".." (父目录) 组件

-M 不创建条目的清单文件

-i 为指定的 jar 文件生成索引信息

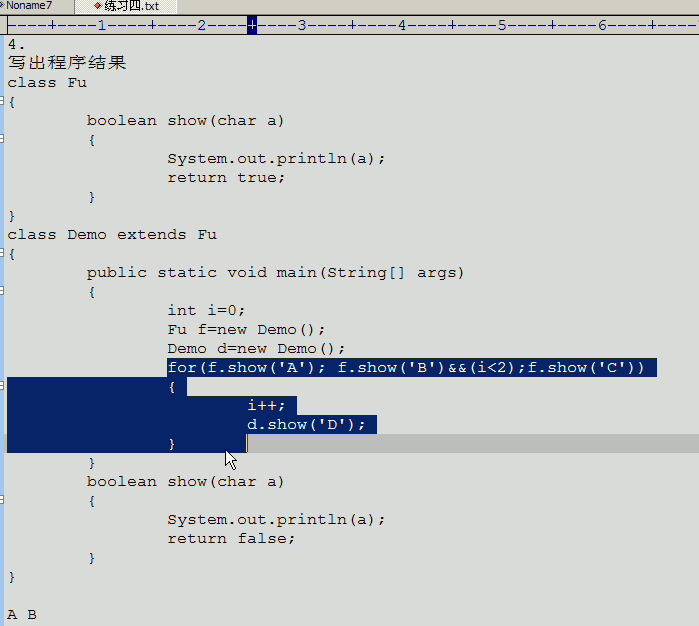
-C 更改为指定的目录并包含以下文件

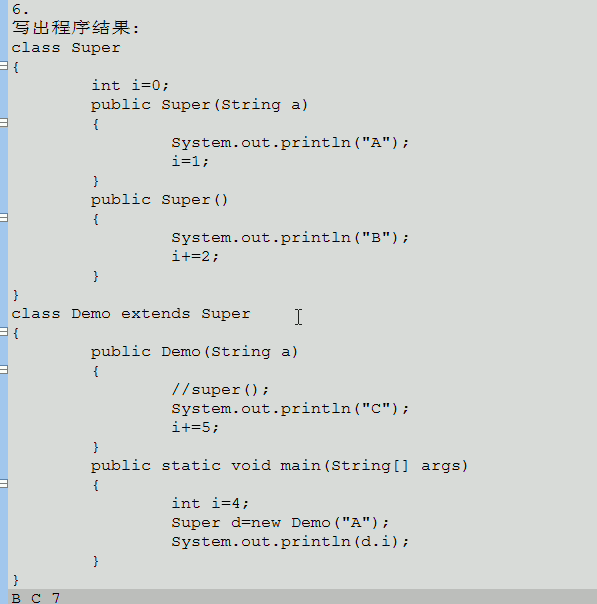
如果任何文件为目录, 则对其进行递归处理。

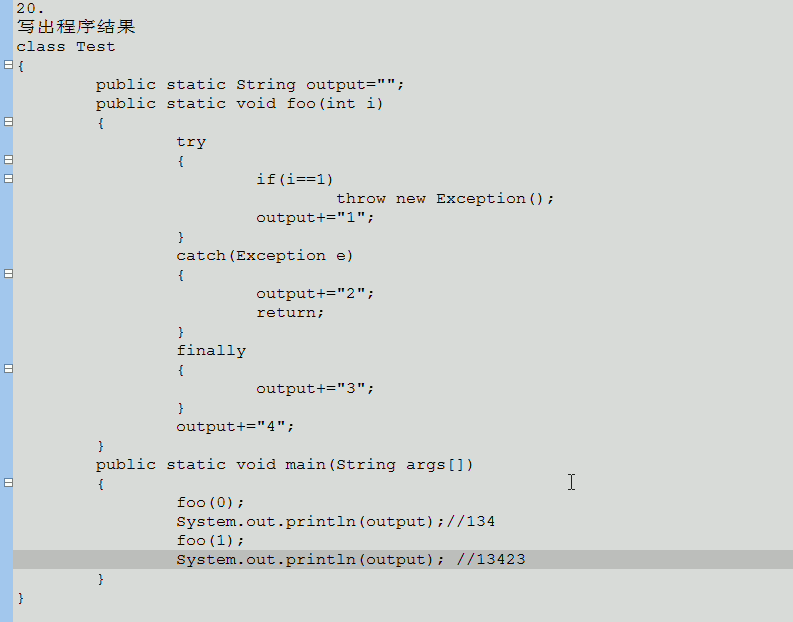
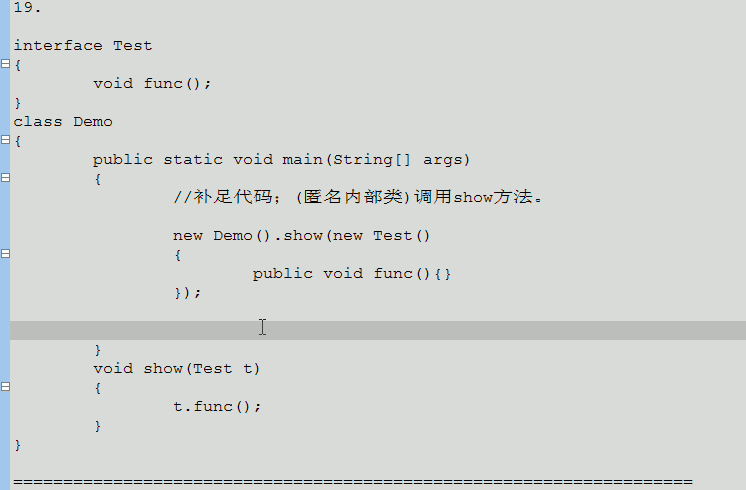
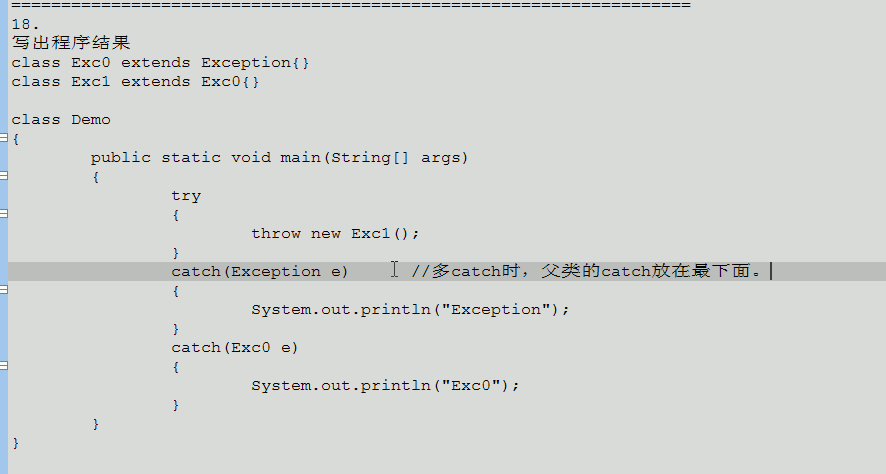
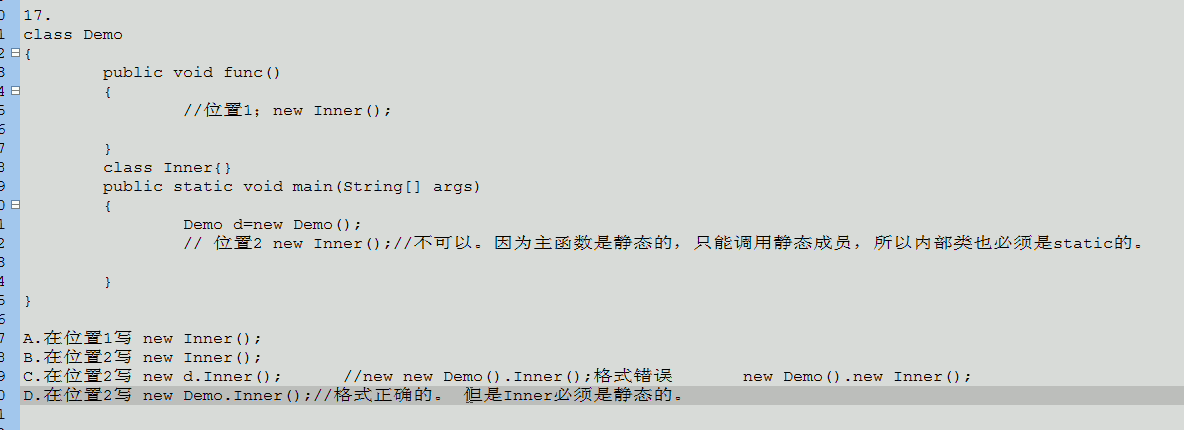
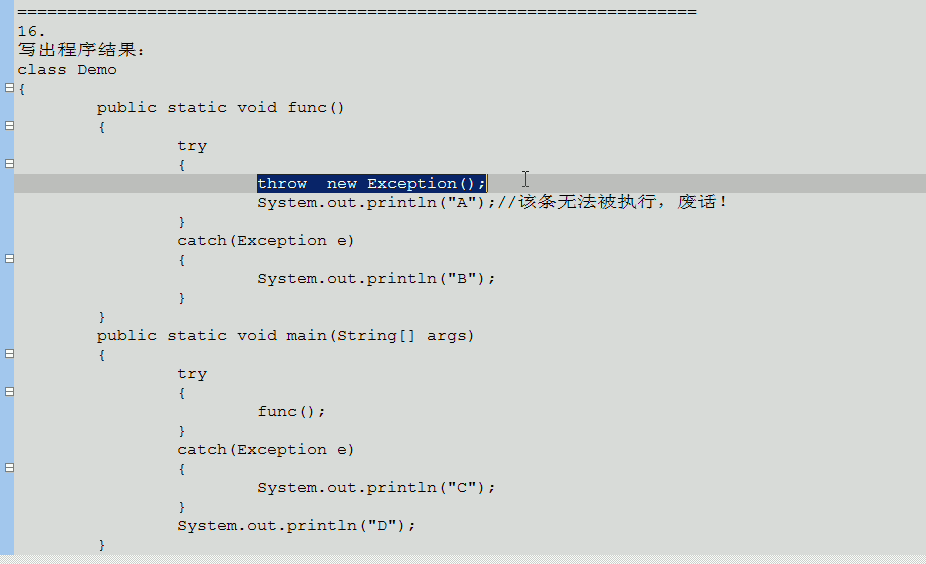
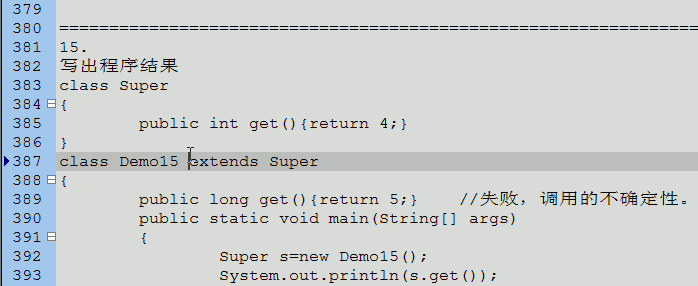
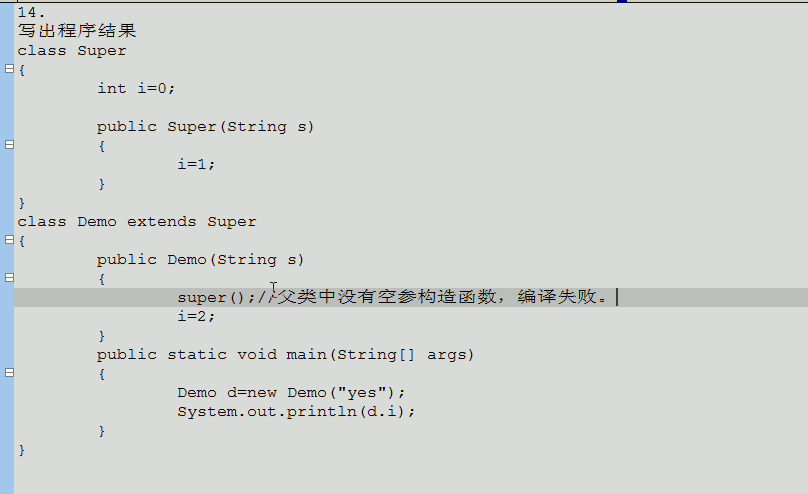
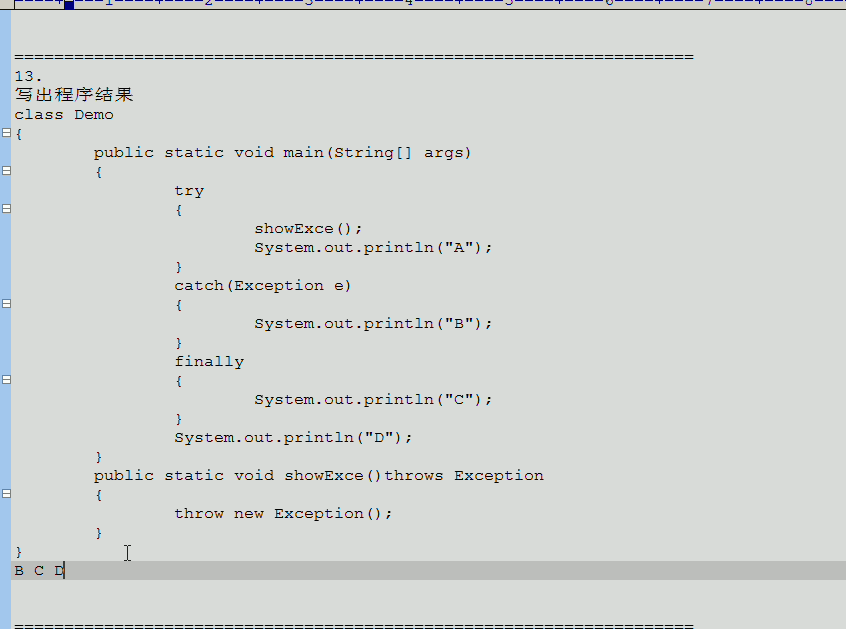
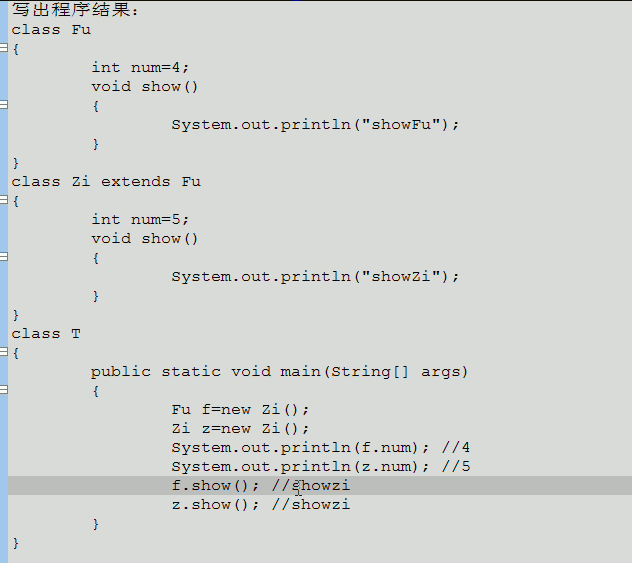
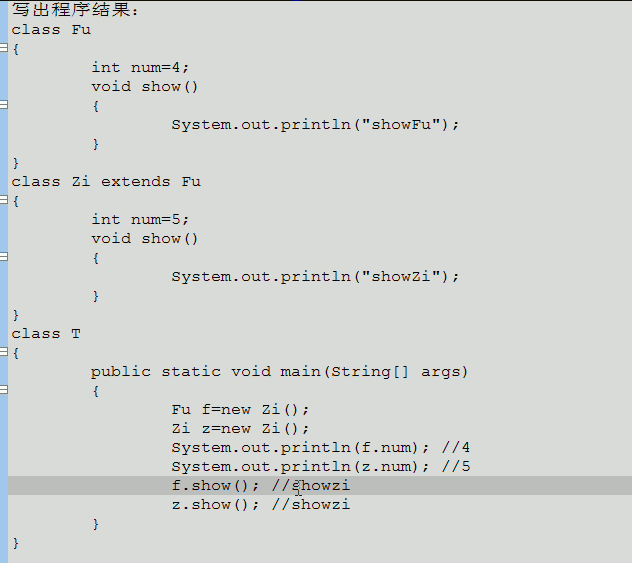
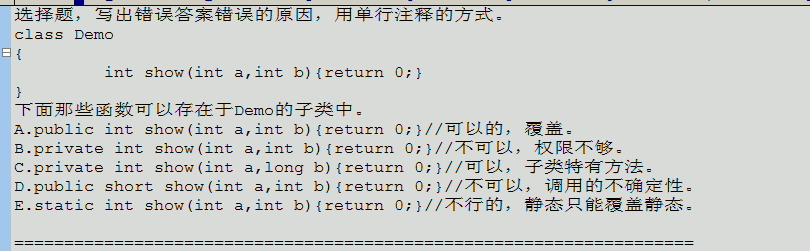
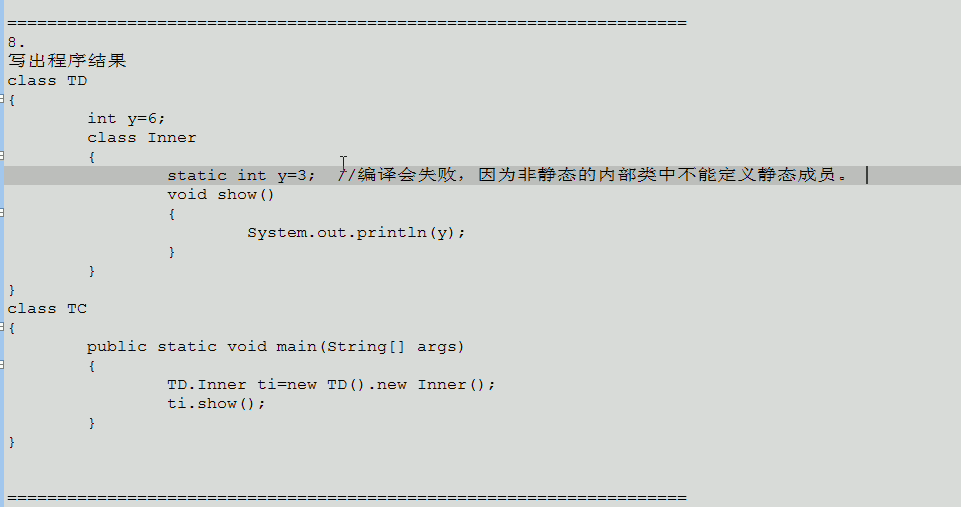
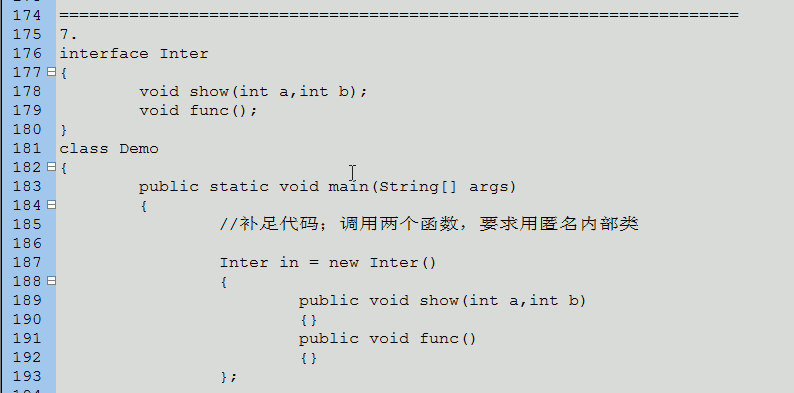
清单文件名, 档案文件名和入口点名称的指定顺序

与 'm', 'f' 和 'e' 标记的指定顺序相同。

练习4解析1：







21：建立一个图形接口，声明一个面积函数。图形和矩形都实现这个接口，并得出两个图形的面积。

注：体现面向对象的特征，对数值进行判断。用异常处理。不合法的数值要出现“这个数值是非法的”提示，不再进行运算。

interface Areable

{

public double getArea();

}

class Rec implements Areable

{

private int length;

private int width;

Rec(int length, int width)

{

if(length<=0 || width <=0)

throw new NoValueException("数值非法");

this.length = length;

this.width = width;

}

public double getArea()

{

return length \* width;

}

}

class Cir implements Areable

{

private int radius;

public static final double PI = 3.14;

Cir(int radius)

{

if(radius<=0)

throw new NoValueException("数值非法");

this.radius = radius;

}

public double getArea()

{

return PI \* radius \* radius;

}

}

class NoValueException extends RuntimeException //不catch时用Runtime

{

NoValueException()

{

super();

}

NoValueException(String msg)

{

super(msg);

}

}

class Test21

{

public static void main(String[] args)

{

// Rec r1 = new Rec(-3,5);

// double a = r1.getArea();

// System.out.println("area = "+a);

Cir r2 = new Cir(-3);

double b = r2.getArea();

System.out.println("area = "+b);

}

}

课1：多线程（Thread）-概述

进程：正在进行中的程序（直译）

进程：就是进程中一个负责程序执行的控制单元（执行路径）

一个进程中可以多执行路径，称之为多线程。

一个进程中至少有一个线程。

开启多个线程是为了同时运行多部分代码。

每一个线程都有自己运行的内容。这个内容可以成为线程要执行的任务。

课2：多线程-好处与弊端

好处：多个线程同时进行，解决了多部分同时运行的问题；

弊端：线程太多回到效率的降低。

其实应用程序的执行都是cpu在做着快速切换完成的。这个切换是随机的。

课3：多线程-JVM中的多线程

JVM启动时就启动了多个线程，至少有两个线程可以分析的出来。

1. 执行main函数的线程。

该线程的任务代码都定义在main函数中。

1. 负责垃圾回收的线程。

finalize

protected void finalize()

throws Throwable

当垃圾收集确定不再有对该对象的引用时，垃圾收集器在对象上调用该对象。 一个子类覆盖了处理系统资源或执行其他清理的finalize方法。

public static void gc()运行垃圾回收器。

主线程结束后，JVM不一定结束，其他线程如果还在运行，那么JVM就没有结束。

课4：多线程-主线程运行示例

课5：多线程创建的方式一-继承Thread类

如何创建一个线程呢？

创建线程方法一：继承Thread类。

步骤：

1. 定义一个类继承Thread类。
2. 覆盖Thread类中的run方法。
3. 直接创建Thread的子类对象创建线程。
4. 调用start方法开启线程，并调用线程的任务run方法执行。

/\*

\* 创建线程的目的是为了开启一条执行路径，去运行指定代码和其他代码，实现同时运行。

\*

\* 而运行的指定代码就是这个执行路径的任务。

\*

\* JVM创建主线程的任务都定义在了主函数当中。

\*

\* 而自定义的线程，它的任务都在哪儿呢？

\* Thread 类用于描述线程，线程是需要任务的。所以Thread类也对任务的描述。

\* 这个任务就通过THread类中的run方法来体现。也就是说，run方法就是封装自定义线程运行任务的函数。

\*

\* run方法中定义就是线程要运行的任务代码。

\*

\* 开启线程是为了运行指定代码，所以只有继承Thread类，并复写run方法。

\* 将运行的代码定义在run方法中即可。

\* \*/

课6：Thread类中的方法&线程名称

可以通过Thread的getName获取线程（运行对象）的名称 Thread-编号（从0开始）

Thread.currentThread().getName()：获取当前线程对象的名称。

Demo(String name)

{

super(name);

// this.name = name;

}

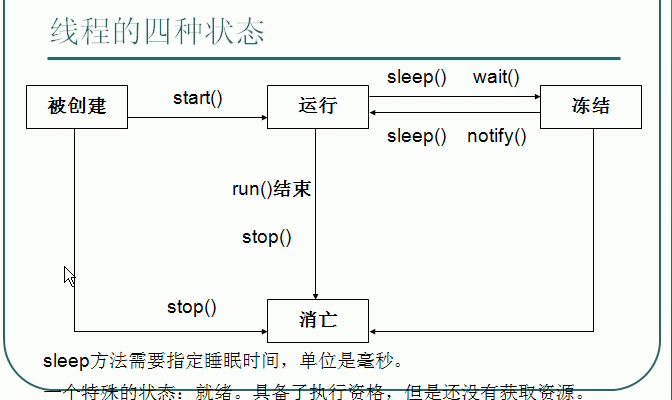
两个结果的名称，null....x=......小强；null....x=3......main

课7：多线程-运行图解

Day13:

课8：线程的状态

线程的四种状态



Notify：唤醒<--> sleep 休眠

CPU的执行资格：可以被CPU的处理，在处理队列中排队。

CPU的执行权：正在被CPU的处理。

冻结：释放执行权的同时，释放执行资格。

具备执行资格，没有执行权：临时阻塞状态。（正在等待）

冻结状态可以到达临时阻塞状态。

课9：创建线程的第二种方式-实现Runnnable接口

课10：多线程-第二种方法细节

课11：多线程-第二种方法好处

Runnable：它的出现仅仅是将线程的任务进行了对象的封装。

1. 将线程的任务从线程的子类中分离出来，进行了单独的封装。

按照面向对象的思想将任务的封装成对象。

1. 避免了java单继承的局限性。

所以，创建线程的第二种方式较为常用。

课12：多线程-卖票实例

IllegalThreadStateException 无效线程状态异常

课13：多线程-线程安全问题的现象。

/\*

线程安全问题产生的原因：

1. 多个线程在操作共享的数据。
2. 操作共享数据的线程代码有多条。

当一个线程在执行操作共享数据的多条代码过程中，其他线程参与了运算。

就会导致线程安全问题的产生。

\*/

课15：同步代码块

线程安全问题：

解决思路：

就是将多条操作共享数据的线程代码封装起来，当有线程在执行这些代码的时候，

其他线程操作不可以参与运算的。

必须要当前线程把这些代码都执行完毕后，其他线程菜可以参与运算。

在java中，用同步代码块就可以解决这个问题。

同步代码块的格式：

Synchronized（对象）

{

需要被同步的代码；

}

课16同步的好处和弊端

好处：解决了线程的安全问题。

弊端：相对降低了效率，因为同步外的线程都会判断同步锁。

课17：同步的前提

前提：同步中必须有多个线程并使用同一个锁。

class Ticket implements Runnable //extends Thread

{

private int num = 100;

Object obj = new Object(); //前提

课18：同步函数

// private Object obj = new Object();

private int sum;

public synchronized void add(int num)

{

// synchronized(obj)

{

sum = sum +num;

try

{

Thread.sleep(10);

}

catch(InterruptedException e)

{

}

System.out.println("sum = "+sum);

}

}

课19：验证同步函数的锁

同步函数与同步代码块

两者区别：

同步函数的使用的锁是this（**synchronized**(**this**)）

同步代码块的锁是任意的对象。

所以同步代码块用的较多。

课20：验证静态同步函数的锁

静态的同步函数使用的锁是 该函数所属字节码文件

可以用getClass方法获取， 也可以用当前 类名.class表示。

课21：单例模式涉及的多线程问题。

课22：死锁示例

/\*

死锁：常见情景之一：同步的嵌套。

\*/

Day14:

课23：线程间通信-示例

思考1：wait(),notifyAll()，用来操作线程为什么定义在了Object类中？

1. 这些方法存在与同步中。
2. 使用这些方法时必须要标识所属的同步的锁。
3. 锁可以是任意对象，所以任意对象调用的方法一定定义Object类中。

思考2：wait(),sleep()有什么区别？

Wait()释放CPU执行权，释放锁。

Sleep()：释放cpu执行权，不释放锁。

/\*

线程间通讯：

多个线程在处理同一资源，但是任务却不同。

\*/

课24：多线程-等待/唤醒机制

涉及的方法：

1，wait();让线程处于冻结状态，被wait的线程会被存储到线程池中。

2，Notify();唤醒线程池中一个线程（任意）。

3，notifyAll();唤醒线程池中的所有线程。

这些方法都必须定义在同步中。

因为这些方法是用于操作线程状态的方法。

必须要明确到底操作的是哪个锁上的线程。

为什么操作线程的方法wait,notify定义在Object类中。

因为这些方法是监视器的方法。监视器其实就是锁。

锁可以是任意的对象，任意的对象调用的方式一定定义在Object类中。

错误: 未报告的异常错误InterruptedException; 必须对其进行捕 获或声明以便抛出

r.wait();

对其进行trycatch捕捉可以，修正错误。

课26：线程间通信-多生产者多消费者问题

1 **if**(flag) 仅判断一次的问题。

2 //while(flag)死锁问题。

课27：线程间通信-多生产者多消费者问题解决

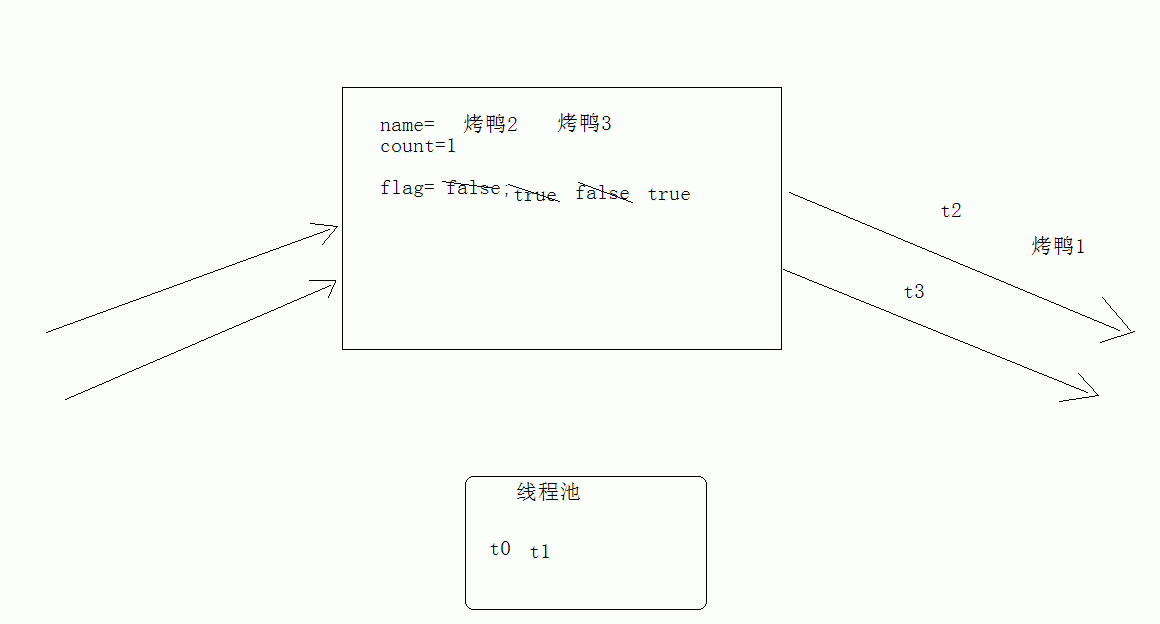
notifyAll命令，可以唤醒全部休眠状态。

If判断标记，只有一次，会导致不该运行的线程运行了，出现了数据错误的情况。

While判断标记，解决了线程获取执行权后，是否要运行！

Notify：只能唤醒一个线程，如果本方唤醒了本方，没有意义。

NotifyAll解决了，本方线程一定会唤醒对方线程的问题。



课28：线程间通信-多生产者多消费者问题-jdk1.5新特性

同步代码块对于锁的操作是隐式的。

Lock lock = new ReentrantLock();

void show()

{

Lock.lock();//获取锁

Code...

Lock.unlock();//释放锁

}

/\* Jdk1.5以后将同步和锁封装成了对象，

\* 并将操作锁的隐式方式定义到了该对象中，

\* 将隐式动作变成了显示动作。

\*

\* Lock接口：出现替代了同步代码块或者同步函数。将同步的隐式锁操作变成现实锁操作。

\* 同时更为灵活。可以一个锁上加上多组监视器。

\* lock()：获取锁。

\* unlock():释放锁，通常需要定义finally代码块。

\*

\* Condition接口：出现替代了Object中的wait notify notifyAll方法。

\* 将这些监视器方法单独进行了封装，变成Condition监视器对象。

\* 可以任意锁进行组合。

\* await();

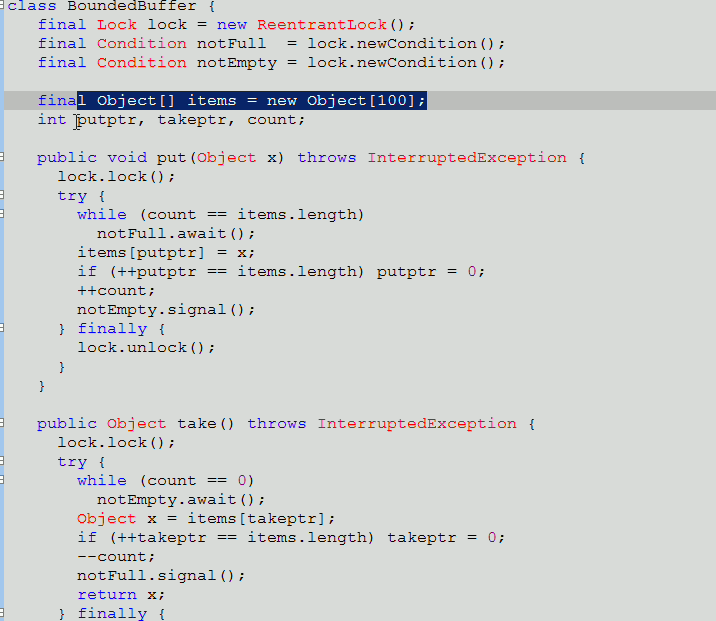
\* signal();

\* signalAll();

课29：线程间通信-多生产者多消费者问题-jdk1.5新特性-Condition

课30：线程间通信-多生产者多消费者问题-jdk1.5解决办法

课31：线程间通信-多生产者多消费者问题-jdk1.5解决办法-范例



lock.unlock();}

课32：多线程（wait和sleep的区别）

\*wait和sleep的区别：

\*1，wait可以指定时间也可以不指定。

\* sleep必须指定时间。

\*2，在同步中时，对cpu的执行权和锁的处理不同。

\* wait:释放执行权，释放锁。

\* sleep:释放执行权，不释放锁。

课33：多线程（停止线程方式-定义标记）

/\*

\*停止线程：

\*1，stop方法。不安全

\*2，run方法。

\*怎么控制线程的任务结束呢？

\*任务中都会有循环结构明知要控制住循环就可以结束任务。

\*

\* 控制循环通常就用定义标记flag来完成。

\*

\* 但是如果线程处于了冻结状态，无法读取标记，如何结束呢？

\*

\* 可以使用interrupt()方法将线程从冻结状态强制恢复到运行状态中来，让线程具备CPU的执行资格。

\*

\* 当时强制动作会放生了InterruptedException，记得要处理

\* \*/

课34：守护线程-setDaemon（后台线程）

课35：多线程（其他方法—join等）

public static void main(String[] args) throws Exception

{

Demo d = new Demo();

Thread t1 = new Thread(d);

Thread t2 = new Thread(d);

t1.start();

t1.join(); //t1线程要申请加入进来，运行。临时加入一个线程运算时可以使用 //join方法。

t2.start();

t2.setPriority(Thread.MAX\_PRIORITY);设置优先级（MAX\_PRIORITY，MIN\_PRIORITY，NORM\_PRIORITY）

for(int x = 0;x<50; x++)

{

System.out.println(Thread.currentThread().getName()+"......"+x);

}

}

Day15

课1：常用对象API（String特点）

加入包cn.itcast.p1.string.demo

String类：

字符串是一个特殊的对象。

字符串一旦初始化就不可以被改变。

String str = “abc”;

String str1 = new String(“abc”);

有什么区别？

String s = "abc"; //创建一个字符串对象在常量池

String s1 = **new** String("abc");//创建两个对象一个new一个字符串对象在堆内存中

课2：String类-构造函数

[String](mk:@MSITStore:C:\\Users\\SZHJ\\Desktop\\jdk%20api%201.8_google.CHM::/java/lang/../../java/lang/String.html" \l "String-byte:A-)(byte[] bytes)

通过使用平台的默认字符集解码指定的字节数组来构造新的 String 。

[String](mk:@MSITStore:C:\\Users\\SZHJ\\Desktop\\jdk%20api%201.8_google.CHM::/java/lang/../../java/lang/String.html" \l "String-byte:A-int-int-java.nio.charset.Charset-)(byte[] bytes, int offset, int length, [Charset](mk:@MSITStore:C:\\Users\\SZHJ\\Desktop\\jdk%20api%201.8_google.CHM::/java/lang/../../java/nio/charset/Charset.html" \o "class in java.nio.charset) charset)

构造一个新的String通过使用指定的指定字节子阵列解码[charset](mk:@MSITStore:C:\\Users\\SZHJ\\Desktop\\jdk%20api%201.8_google.CHM::/java/lang/../../java/nio/charset/Charset.html" \o "java.nio.charset中的类) 。

**char** [] arr = {'w','a','p','q','x'};

// String s = new String(arr);

String s = **new** String(arr,1,3);

System.*out*.println("s="+s);

课3：String类-常见功能-获取1

课4：String类-常见功能-获取2

\* 按照面向对象的思想对字符串进行功能分类。

\* “abcd”

\*

\* 1，获取：

\* 1.1 获取字符串中字符的个数（长度）。

\* int length();

\* 1.2根据位置获取字符。

\* char charAt(int index);

\* 1.3根据字符获取在字符串中第一次出现的位置。

\* int indexOf(int ch):根据字符获取在字符串中第一次出现的位置。

\* int indexOf(int ch,int fromIndex):从指定位置进行ch的查找第一次出现的位置。

\* int indexOf(String str):根据子字符串获取在字符串中第一次出现的位置。

\* int indexOf(String str,int fromIndex):从指定位置进行子字符串str的查找第一次出现的位置

\* 根据字符获取在字符串中最后一次出现的位置。

\* int lastIndexOf(int ch):从指定位置进行ch的查找最后一次出现的位置。

\* int lastIndexOf(int ch,int fromIndex):从指定位置进行ch的查找最后一次出现的位置。

\* int lastIndexOf(String str):根据子字符串获取在字符串中最后一次出现的位置。

\* int lastIndexOf(String str,int fromIndex):从指定位置进行子字符串str的查找最后一次出现的位置 。

\*

\* 1.4获取字符串中的一部分字符串，也叫子串。

\* String subString(int beginIndex, int endIndex)//从beginIndex字符到endIndex-1处的字符

\* String subString(int beginIndex)

\* \*/

// StringConstructorDemo2();

StringMethodDemo\_1();

}

public static void StringMethodDemo\_1() {

String s = "abcdae";

System.out.println("length : "+s.length());//6

System.out.println("char : "+s.charAt(2));//c

System.out.println("index :"+s.indexOf('a'));//0

System.out.println("lastIndex : "+s.lastIndexOf('a'));//4

// System.out.println("char : "+s.charAt(20));//字符串角标越界：java.lang.StringIndexOutOfBoundsException

System.out.println("index :"+s.indexOf('k'));// -1 我们可以根据-1，来判断该字符或者字符串是否存在。

System.out.println("beginIndex : "+s.substring(2,4));

}

课5：String类-常见功能-转换

\* 2.1 将字符串变成字符串数组（字符串的切割）。

\* String [] split(String regex):涉及到正则表达式

\* 2.2 将字符串变成字符数组。

\* char [] toCharArray();

\* 2.3 将字符串变成字节数组。

\* byte [] getBytes();

\* 2.4 将字符串中的字母转成大小写。

\* String toUpperCase();大写

\* String toLowerCase();小写

\* 2.5 将字符串中的内容进行替换。

\* String replace(char oldChar,char newChar);

\* String replace(String s1,String s2);

\* 2.6 将字符串两端的空格去除。

\* String trim();

\* 2.7 将字符串进行连接。

\* String concat(string);

\* 2.8 将基本数据类型变成字符串类型。

\* String valueOf(xxx);

String s = "张三，李四，王五";

// String s = "张三.李四.王五";

// String [] arr = s.split(",");

String [] arr = s.split("\\.");//正则

for(int i=0;i<arr.length;i++)

{

System.out.println(arr[i]);

}

char [] chs = s.toCharArray();

for(int i=0;i<chs.length;i++)

{

System.out.println(chs[i]);

}

s = "ab你";

byte [] bytes = s.getBytes();

for(int i=0;i<bytes.length;i++)

{

System.out.println(bytes[i]);

}

System.out.println("Abc".toUpperCase());

System.out.println("Abc".toLowerCase());

s = "java";

String s2 = s.replace('a', 'b');

System.out.println(s==s2);

System.out.println("-"+"abc".trim()+"-");

System.out.println("-"+" ab c "+"-");

System.out.println("-"+" ab c ".trim()+"-");

System.out.println("abc".concat("kk"));

System.out.println("abc"+"kk");

System.out.println('a'+1);

System.out.println(String.valueOf('a')+1);

System.out.println(""+'a'+1);

课6：String类-常见功能-判断

3,判断

\* 3.1两个字符串内容是否相同？

\* boolean equals(Object obj);

\* boolean equalsIngoreCase(String str);忽略大小写比较字符串内容。

\* 3.2字符串中是否包含指定字符串？

\* boolean contains(String str);

\* 3.3字符串是否以指定字符串开头，是否以指定字符串结尾。

\* boolean startsWith(string);

\* boolean endsWith(string);

课7：String类-常见功能-比较

4,比较

\* int compareTo(String anotherString);从开头开始比较，返回两字符串的大小比较（大小等于）。

课8：String类-intern方法

//intern():对字符串池进行操作。

String s1 = **new** String("abc");

// String s2 = new String("abc");

String s2 = s1.intern();

System.*out*.println(s1==s2);

课9：String类-练习1-字符串数组排序

课10：String类-练习2-子串的次数

文档注释所用，不要删除！

/\*\*

\* **@param** args

\*/

课11：String类-练习3-最大相同子串

课12：String类-练习4-模拟一个trim功能一致的方法。

课9：StringBuffer类-特点&添加功能

字符串的组成原理就是通过该类实现的。

StringBuffer可以对字符串内容进行增删。

StringBuffer是一个容器。

很多方法都和String相同。

StringBuffer是可变长度的。

课10：StringBuffer类-插入&删除&修改&查找

/\*

\* StringBuffer就是字符串缓冲区。（初始容量为16字符）

\* 用于存储数据的容器。

\* 特点：

\* 1，长度是可变的。

\* 2，可以存储不同类型的数据。

\* 3，最终要转成字符串使用。（append）

\* 4,可以对字符串进行修改。（insert）

\*

\* 既然是一个容器对象，应该具备什么功能？

\* 1，添加：

\* StringBuffer append(data);

\* StringBuffer insert(offset,data);

\* 2，删除：

\* StringBuffer delete(start,end);包含头不含尾。

\* StringBuffer deleteCharAt(int index);删除指定位置元素。

\* 3,查找：

\* char charAt(index);

\* int indexOf(string);

\* int lastIndex0f(string);

\* 4,修改：

\* StringBuffer replace(start,end,string);

\* void setCharAt(index,ch);

\*

\* 增删改查：C(create)U(update)R(read)D(delete)

\*

\*

\*

\* \*/

StringBufferDemo\_1();

}

private static void StringBufferDemo\_1() {

StringBuffer sb = new StringBuffer("abce");

// sb.append("xixi");

// sb.insert(2, "qq");

// sb.delete(1, 3);

// 清空缓冲区。

// sb.delete(0, sb.length());

// sb = new StringBuffer();

// sb.replace(1, 3, "nba");

// sb.setCharAt(2, 'c');

// 可以作为清空缓冲区使用。

sb.setLength(10);

// System.out.println("sb="+sb);

// System.out.println("Len="+sb.length());

System.out.println(sb.reverse());

课11：StringBuffer类-可变长度数组

StringBuffer(int capacity);

课12：StringBuilder类

一个可变的字符序列，兼容StringBuffer（即拥有所有的该类的功能。）但不保证同步。

而StringBuffer是同步的，线程安全的可变的字符序列。

/\*

\* jdk1.5以后出现了功能和StringBuffer功能一模一样的对象。单个线程使用的等价类，就是StringBuilder。

\* 不同的是：

\* StringBuffer是线程同步的。通常用于多线程。

\* StringBuilder是线程不同步的。通常用于单线程。它的出现提高了效率。

\*

\* jdk的升级：

\* 1，简化书写。

\* 2，提高效率。

\* 3，增加安全性。

\*

\* \*/

课13：StringBuilder类-练习

public static String ArrayToString\_2(int[] arr) {

StringBuilder sb = new StringBuilder();

sb.append("{");

for (int i = 0; i < arr.length; i++) {

if(i!=arr.length-1)

{

sb = sb.append(arr[i]+",");

}

else

sb = sb.append(arr[i]+"}");

}

return sb.toString();

public static String ArrayToString(int [] arr) {

String str = "{";

for(int x=0;x<arr.length;x++)

{

if(x!=arr.length-1)

str =str+ arr[x]+","; //多次在字符串常量池中操作。

else

str =str+ arr[x]+"}";

}

return str;

}

上述两种将数组转换成字符串。很明显用StringBuilder更好。效率高。

Day16:

课1：基本数据类型对象包装类-概述

将基本数据类型封装成对象的好处在于可以在对象中定义更多的功能方法操作该数据。

常用的操作之一：用于基本数据类型与字符串之间的转换。

例如：Integer 的 parseInt方法，intValue方法。

课2：基本数据类型对象包装类-字符串转成基本数值

Integer:

类型为Integer的对象包含一个单一字段，其类型为int。

该类还提供了一些将int转换为String和String转换为int。

int ParseInt(String s)throws [NumberFormatException](mk:@MSITStore:C:\\Users\\SZHJ\\Desktop\\jdk%20api%201.8_google.CHM::/java/lang/../../java/lang/NumberFormatException.html" \o "class in java.lang)

将字符串参数解析为带符号的十进制整数。

如果出现异常可以进行处理。（java.lang.NumberFormatException）

如果不想发生异常，必须传输数字类型的字符串。

\* 基本数据类型对象包装类。

\* 为了方便操作基本数据类型，将其封装成了对象，在对象中定义了属性和行为丰富了该操作。

\* 用于描述该对象的类就称为基本数据类型对象包装类。

\*

\* byte Byte

\* short Short

\* int Integer

\* long Long

\* double Double

\* float Float

\* char Character

\* boolean Boolean

\*

\* 该保证对象主要用基本类型和字符串之间的转换。

\*

\* 基本类型--->字符串

\* 1，基本类型数值+""

\* 2,用String类中的静态方法valueOf(基本类型数值);

\* 3,用Integer的静态方法valueOf(基本类型数值);

\* 字符串--->基本类型

\* 1，使用包装类中的静态方法：xxx parsexxx(xxx类型的字符串);

\* int parseInt("intString");

\* long parseLong("longString");

\* boolean parseBoolean("booleanString"); ...

\* 只有Character没有parse方法。

\*

\* forDigit();toUpperCase(char ch);toLowerCase(char ch);等等的使用。

课3：基本数据类型对象包装类-进制转换

/\*

\* 整数具备不同的进制体现。

\* 十进制-->其他进制。

\* toBinaryString(int i);

\* toOctalString(int i);

\* toHexString(int i);

\* toString(int i,int radix); //radix:基数

\* 其他进制-->十进制。

\* parseInt(String str,int radix);

课4：基本数据类型对象包装类-JDK1.5自动装箱拆箱

**int** num = 4;

num = num +5;

// Integer i = new Integer(4);

Integer i = 4; //自动装箱，简化书写。

i = i+6; // i = new Integer(i.intValue()+6) //自动拆箱操作

Integer a = **new** Integer(128);

Integer b = **new** Integer(128);

System.*out*.println(a==b);

System.*out*.println(a.equals(b));

Integer x = 128; //jdk1.5后，自动装箱，如果装箱的是一个字节，那么该地数据会被共享，不会重新开辟新的空间。

Integer y = 128;

System.*out*.println(x==y);

System.*out*.println(x.equals(y));

课5：基本数据类型对象包装类-JDK1.5自动装箱拆箱-练习

课1：常用对象API-集合框架-概述

集合类的由来：

对象用于封装特有数据，对象多了需要存储，如果对象的个数不确定。

就使用集合容器进行存储。

集合特点：

1，用于存储对象的容器。

2，集合的长度是可变的。

3，集合中不可以存储基本数据类型值。

集合容器因为内部的数据结构不同，有多种具体容器。

不断的向上抽取，就形成了集合框架。

框架的顶层Collection接口。

课2：常用对象API-集合框架-体系&共性功能

Collection的常见方法：

1，添加。

boolean add(Object obj);

boolean addAll(Collection coll);

2,删除。

boolean remove(object obj);

boolean removeAll(Collection coll);

void clear();

3,判断。

boolean contains(object obj);

boolean containsAll(Collection coll);

boolean isEmpty();判断集合中是否有元素。

4,获取。

int size();

Iterator iterator();取出集合中元素的方式：迭代器。

5，其他。

boolean retainAll(Collection coll);取交集。

Object [] toArray();将集合转成数组。

课3：常用对象API-集合框架-方法演示\_1

课4：常用对象API-集合框架-方法演示\_2

课5：常用对象API-集合框架-迭代器的使用

Collection coll = new ArrayList();

coll.add("abc1");

coll.add("abc2");

coll.add("abc3");

//使用了Collection中的iterator()方法。

// Iterator it = coll.iterator();

// //以下两种方法。

// 使用next()方法后，可以对集合中的每个元素进行操作，判断。

// while(it.hasNext())

// {

// System.out.println(it.next());

// }

//而while循环，会在内存中一直有it。浪费内存。

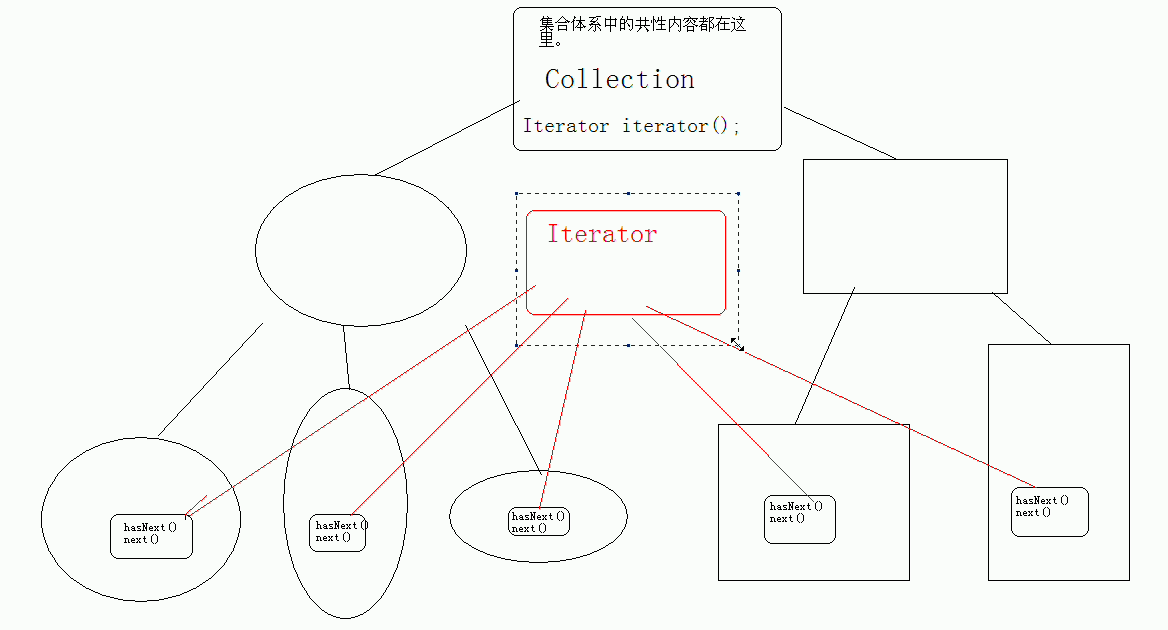
//开发选择下面的for循环，循环结束后会释放it的内存空间。

for (Iterator it = coll.iterator(); it.hasNext();) {

System.out.println(it.next());

}

课6：常用对象API-集合框架-迭代器的原理



Iterator iterator();取出集合中元素的方式：迭代器。（内部类的原理）

该对象必须依赖于具体容器，因为每一个容器的数据结构不同。

所以该迭代器对象是在容器中进行内部实现的。

对于使用容器者而言，具体的实现不重要，只要通过容器获取到该实现的迭代器的对象即可。

也就是iterator方法。

课7：常用对象API-集合框架-list和set的特点

Collection

|--List:有序（存入和取出的顺序一致，）元素都有索引（角标），列表元素可以重复。

|--Set:元素不能重复，无序。

课8：常用对象API-集合框架-list常用方法

插入：

1.关于导入包，中.util含义：

util是utiliy的缩写，意为多用途的，工具性质的包   
这个包中主要存放了：集合类（如ArrayList，HashMap等），随机数产生类，属性文件读取类，定时器类等类。

1. E:Object类

课9：常用对象API-集合框架-listIterator接口

ListIterator it = list.listIterator();//获取列表迭代器对象。

//它可以实现在迭代过程中的增删改查。

//注意：只有list集合具备该迭代功能。

while(it.hasNext()){

Object obj = it.next();

if(obj.equals("abc2"))

// it.add("abc9");

it.set("abc9");

}

// System.out.println("hasnext:"+it.hasNext());

// System.out.println("hasprevious:"+it.hasPrevious());

// 迭代器list同样可以实现正序逆序遍历功能

while(it.hasPrevious()){

System.out.print(it.previous());

}

System.out.println(list);

课10：常用对象API-集合框架-list常用子类的特点

List：特有的常见方法：有一个共性特点就是都可以操作角标。

（增删改查）

1，添加

void add(index ,element);

void add(index,collection);

2，删除

Object remove(index);

3,修改：

Object set(index,element);

4,获取：

Object get(index);

int indexOf(object);

int lastIndexOf(object);

List subList(fromIndex,toIndex);

list集合可以完成对元素的增删改查。

List:

|--Vector:内部是数组数据结构，是同步的（安全的）。增删，查询都很慢。

|--ArrayList:内部是数组数据结构，是不同步的。替代了Vector。查询速度好很快。

|--LinkedList:内部是链表数据结构的，是不同步的。增删元素的速度很快。

课11：常用对象API-集合框架-数组和链表

数组（索引或角标）和链表都是带标号的。

Day17:

课12：常用对象API-集合框架-Vector

事实上，接口的Enumeration功能与Iterator接口的功能是重复的。

此外，Iterator还添加了一个可选的删除操作，并具有较短的方法名称。

课13：常用对象API-集合框架-LinkedList集合

课14：常用对象API-集合框架-LinkedList集合-练习（堆栈和队列）

Jdk1.6的升级：提供的新方法

课15：常用对象API-集合框架-ArrayList集合存储自定对象

什么是JavaBean?

一般程序中，称它为数据层，就是用来设置数据的属性和一些行为，然后我会提供获取属性和设置属性的get/set方法。在MVC（Model View Controller）设计模型中是model，又称模型层。

（简单说：就是通过JAVA语言写成的可重用组件，或者说可复用的类）

JavaBean的特点：

1，这个Java类必须具有一个无参的构造函数

2，属性必须私有化。

3，私有化的属性必须通过public类型的方法暴露给其它程序，并且方法的命名也必须遵守一定的命名规范。

JavaBean的优势：

使用 javabean 之后，优势也就是 java 的优势：组件技术，代码重用，易于维护。

generate constructor from surperclass:生成空参构造函数。

generate constructor using fields：生成带参的构造函数。

Generate getters and setters:将私有属性暴露，生成对外提供的用于访问私有属性的public方法。

强制转换Person类快捷键：ctrl+1-->add cast to “it.next()”。

课16：常用对象API-集合框架-HashSet集合

Set:元素不可以重复（唯一），是无序的。

Set接口在的方法和Collection一致。

|--HashSet:内部数据结构是哈希表，是不同步的。

课17：常用对象API-集合框架-哈希表\_1

哈希表确定元素是否相同。

1. 判断的是两个元素的哈希值是否相同。

如果相同，在判断两个对象的内容是否相同。

1. 判断哈希值相同，其实判断的是对象的hashCode的方法。

判断内容相同，用的是equals方法。

注意：如果哈希值不同，是不需要判断equals。

课18：常用对象API-集合框架-哈希表\_2

如何保证该集合的元素唯一性呢？

是通过对象的hashCode和equals方法来完成对象唯一性的。

如果对象的hashCode值不同，那么不用判断equals方法，就直接存储到哈希表中。

如果对象的hashCode值相同，那么要再次判断对象的equals方法是否为true。

如果为true,视为相同元素，不存。如果为false，那么视为不同元素，就进行存储。

记住，如果元素要存储到HashSet集合中，必须覆盖hashCode和equals方法。

一般情况下没如果定义的类会产生很多对象，比如人，学生，书，通常需要覆盖hashCode和equals方法。

建立对象判断是否相同的依据。

课19：常用对象API-集合框架-HashSet存储自定义对象

Person对象的应用。

Override：自动生成复写。

知识点：java 中的instanceof 运算符是用来在运行时指出对象是否是特定类的一个实例。instanceof通过返回一个布尔值来指出，这个对象是否是这个特定类或者是它的子类的一个实例。

课20：常用对象API-集合框架-练习

alt+上下键  上下移动：实现某行的上下移动。

课21：常用对象API-集合框架-LinkedHashSet集合

课22：常用对象API-集合框架-TreeSet集合

具有自然排序的能力。

课23：常用对象API-集合框架-TreeSet集合-Comparator比较器

课24：常用对象API-集合框架-TreeSet集合-二叉树

|--TreeSet:可以对Set集合中的元素进行排序，是不同步的。

判断元素唯一性的方法：就是根据比较方法的返回结果是否为0，就是相同元素，不存。

TreeSet 对元素进行排序的方式一：

让元素自身具备比较功能，就需要实现Comparable接口。覆盖compareTo方法。

如果不按照对象中具备的自然顺序进行排序，那么对象中不具备自然顺序。

可以使用TreeSet集合的第二种排序方式二：

让集合自身具备比较功能，定义一个类实现Comparator接口，覆盖compare方法。

将该类对象作为参数传递给TreeSet集合的构造函数。

（该方法常用）

课25：常用对象API-集合框架-TreeSet集合-练习-字符串长度排序

Day18:

课1：常用对象API-集合框架-泛型 （Generic）-概述

a1.add("abc"); //public boolean add(Object obj),add是object类型。

a1.add("haha");

a1.add(4)； //==a1.add(new Integer(4); 但是int类型不能被强转为String类型。

Iterator it = a1.iterator();

while(it.hasNext()){

String str = (String)it.next();

System.out.println(str);}

例如： int [] arr = new int [3];

Arr[0] = 9.0;

上例是错误的。

课2：常用对象API-集合框架-泛型-擦除&补偿

课3：常用对象API-集合框架-泛型-在集合中的应用

1. 快捷生成访问属性的调用方法：alt+shift+s-->+alt+r

类的部分或者全部属性不希望让外部世界直接访问，而不用public字段修饰。这样，方法调用成了访问这些属性的唯一途径public的getXxx()方法来获取属性和public的setXxx()方法来修改属性。JavaBean就是一个很好的例子，其严格遵守面向对象的设计逻辑，所有属性都是private。对于任何属性xxx，都有。

1. 自动生成（空参和含参）构造函数：右击-->source-->Generate Constructor using field。
2. 鼠标放在某类上，为其add type。
3. Comparable<T>:T含义是Type。

Map<K,V>:K是key，V是value的意思。

5，快速达到：覆盖(override)&实现功能：source-->Override/Implement Methods

课4：常用对象API-集合框架-泛型-泛型类

其中对.java文件重构refactor ，可以修改该文件的文件名。

Superclass：继承（extends）类。

课5：常用对象API-集合框架-泛型-泛型方法

泛型定义在方法上：定义位置在修饰符后 <xxx> 返回值之前。

课6：常用对象API-集合框架-泛型-泛型接口

课7：常用对象API-集合框架-泛型-泛型限定-上限

通配符的基本使用。

自定义泛型（T）类型和通配符（?）类型的区别？

自定义T可以用来操作（T t = it.next();）

而？：只能用作不区分类型情况下为Collection添加类型（Collection<?> a），不能上述操作。

泛型限定：(Collection<? **extends** Person> a1)

课8：常用对象API-集合框架-泛型-下限

课9：常用对象API-集合框架-泛型-上限的体现

课10：常用对象API-集合框架-泛型-下限的体现

课11：常用对象API-集合框架-泛型-泛型限定（通配符的体现）

泛型的通配符。

泛型的通配符：？

\* 迭代并打印集合中的元素。

\*

\* 可以对类型进行限定：

\* ?extends E:接收E类型或者E类型的子对象。上限！

\*

\* ?super E:接收E类型或者E的父类型。下限！

一般存储元素时都是用上限，因为这样取出都是按照上限类型来运算的。不会出现类型安全隐患。

什么时候用下限呢？通常对集合中的元素进行取出时，可以用下限。如：比较器。

课12：常用对象API-集合框架-泛型-集合查阅的技巧

集合的一些技巧：

需要唯一吗？

需要：Set

需要制定顺序：

需要：TreeSet

不需要：HashSet

但是想要一个和存储已知的顺序（有序）：LinkedHashSet。

不需要:List

需要频繁增删吗？

需要：LinkedList

不需要：ArrayList

如何记录每一个容器的结构和所属体系呢?

看名字！

List

|--ArrayList

|--LinkedList

Set

|--HashSet

|--TreeSet

后缀名就是该集合所属的体系。

前缀名就是该集合的数据结构。

看到array：就要想到数组，就要想到查询快，有角标。

看到link：就要想到链表，就要想到增删快，就要想到add get remove + first last的方法。

看到hash：就要想到哈希表，就要想到唯一性，就要想到元素需要覆盖hashCode和equals方法。

看到tree：就要想到二叉树，就要想到排序，就要想到两个接口Comparable和Comparator。

而且通常这些常用的集合容器都是不同步的。

课1：常用对象API-集合框架-Map集合特点&常用方法

课2：常用对象API-集合框架-常用方法演示

课3：常用对象API-集合框架-重点方法KeySet演示图解

课4：常用对象API-集合框架-重点方法entrySet演示图解

嵌套类。如：AbstractMap.SimpleEntry<K,V>。

课5：常用对象API-集合框架-方法value演示

课6：常用对象API-集合框架-Map集合-常见子类对象

课7：常用对象API-集合框架-HashMap存储自定义对象

课8：常用对象API-集合框架-TreeMap存储自定义对象

课9：常用对象API-集合框架-LinkedHashMap&关联源码

课10：常用对象API-集合框架-Map集合练习-记录字母次数思路

存映射关系的是：数组。

容器中，我们用的是Map。

课11：常用对象API-集合框架-记录字母次数代码

课12：常用对象API-集合框架-TreeMap存储自定义对象

课13：常用对象API-集合框架-Collections-排序

//collections自带交换功能。

//比较list中字符串大小

for (int i = 0; i < list.size()-1; i++) {

for (int j = i+1; j < list.size(); j++) {

if(list.get(i).compareTo(list.get(j))>0){

// T temp = list.get(i);

// list.set(i, list.get(j)); //?：Object中没有set调用。

// list.set(j, temp);

//collections自带交换功能。

Collections.swap(list, i, j);

}

}

}

课14：常用对象API-集合框架-Collections-折半&最值

BinaraySearch();

Max();

课15：常用对象API-集合框架-Collections-逆序&替换

课16：常用对象API-集合框架-Collections-其他方法&将非同步集合转换为同步集合的方法

课17：常用对象API-集合框架-Arrays-方法介绍

课18：常用对象API-集合框架-Arrays-asList方法

课19：常用对象API-集合框架-Collections-toArray方法

课20：常用对象API-集合框架-JDK5.0 特性-ForEach

课21：常用对象API-集合框架-JDK5.0 特性-函数可变特性

课22：常用对象API-集合框架-JDK5.0 特性-静态导入

Day20:

课1：其他对象API-System类

系统中换行：

\r\n：windows操作

\n:Linux

不同系统的默认属性信息有很多不同，怎样获取跨平台信息呢？

private static final String LINE\_SEPARATOR = System.getProperty("line.separator");

//通过.getProperty获取当前系统的分隔符：

System.out.println("Hello-"+LINE\_SEPARATOR+"World");

//为系统设置一些属性，信息。这些属性和信息是全局的，其他程序都可以使用。

System.setProperties("myclasspath","c:\myclass");

课2：其他对象API-Runtime类

Runtime没有构造方法-->不能创造对象-->都是私有化的-->该类中的方法都是静态的。

所以，Runtime中必然存在一返回对象的静态方法。getRuntime();

添加异常：ctrl+1-->add thorws declaration。

/\*

\* Runtime：没有构造方法摘要，说明该类不能创建对象。

\* 又发现还有非静态的方法，说明该类应该提供静态的返回

\* 而且只有一个，说明Runtime类使用了单例设计模式。

\* \*/

课3：其他对象API-Math类

/\*

\* Math:提供了操作数字运算的方法，都是静态的。

\*

\* 常用的方法：

\* cell():返回大于参数的最小整数。

\* floor():返回小于参数的最大整数。

\* round():返回四舍五入的整数。

\* pow(a,b):a的b次方。

\* random:返回一个伪随机数。

\* \*/

课4：其他对象API-Date类

课5：其他对象API-Date类-日期对象转换为字符串

课6：其他对象API-Date类-字符串转换为日期对象

课7：其他对象API-Date类-练习

课8：其他对象API-Calendar类-基本演示

课9：其他对象API-Calendar类-练习

课1：IO流-输入流&输出流

IO流用来处理设备之间的数据传输

Java对数据的操作是通过流的方式

Java用于操作流的对象都在IO包中

流按操作数据分为两种：字节流与字符流

流按流向分为：输入流，输出流

课2：IO流-字节流&字符流

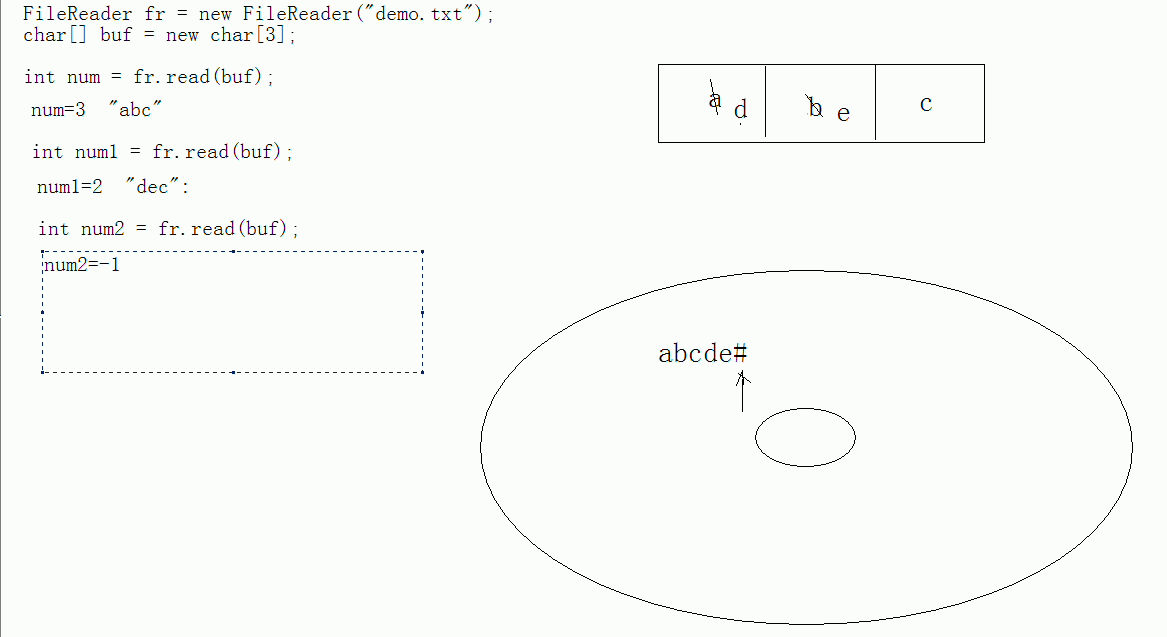
课3：IO流-字符流-FileWriter

课4：IO流-字符流-FileWriter-细节

课5：IO流-字符流-FileWriter-IO异常处理

课6：IO流-字符流-FileReader-读取方式一

课7：IO流-字符流-FileReader-读取方式二



课8：IO流-字符流-练习-复制文本文件\_1

//1,读取一个已有的文本文件，使用字符读取流和文件相关联。

FileReader fr = new FileReader("IO流\_2.txt");

//2，创建一个目的，用于存储读到的数据。

FileWriter fw = new FileWriter("copytext\_1.txt");

//3，频繁的读写操作。

int ch = 0;

while((ch = fr.read())!=-1){

fw.write(ch);

}

//4，关闭流资源。

fw.close();

fr.close();

}

课9：IO流-字符流-练习-复制文本文件\_2

FileReader fr = null;

FileWriter fw = null;

try{

fr = new FileReader("IO流\_2.txt");

fw = new FileWriter("CopyTextTest\_2.txt");

//创建一个临时容器，用于缓存读到的字符。

char [] buf = new char[BUFFER\_SIZE];

//定义一个变量记录读取到的字符数（其实就是往数组里装的字符个数）

int len = 0;

while((len=fr.read(buf))!=-1){

fw.write(buf,0,len);

}

}catch(Exception e){

// System.out.println("读写失败");

throw new RuntimeException("读写失败");

}finally{

if(fw!=null)

try {

fw.close();

} catch (IOException e) {

e.printStackTrace();

}

if(fr!=null)

try {

fr.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

课10：IO流-字符流-练习-复制文本文件\_图解

课11：IO流-缓冲区-解释

缓冲区的出现提高了对数据的读写效率。

对应类

BufferedWriter

BufferedReader

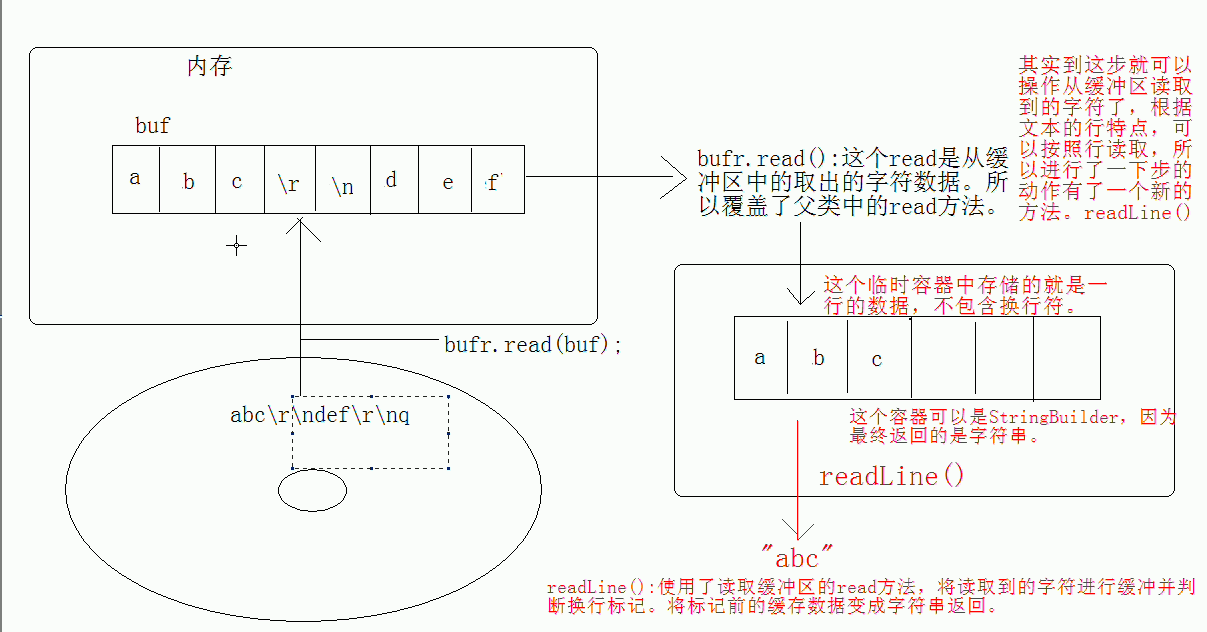
缓冲区要结合流才可以使用。

在流的基础上对流的功能进行了增强。

课12：IO流-字符流-缓冲区-BufferWriter

课13：IO流-字符流-缓冲区-BufferReader

课14：IO流-字符流-缓冲区-BufferReader-readLine方法原理



课15：IO流-字符流-缓冲区-复制文本文件

课16：IO流-字符流-缓冲区-自定义MyBufferedReader-read方法

课17：IO流-字符流-缓冲区-自定义MyBufferedReader-readLine方法

课18：IO流-字符流-缓冲区-装饰设计模式

课19：IO流-字符流-缓冲区-装饰设计模式和继承的区别

课20：IO流-字符流-缓冲区-LineNumberReader

课21：IO流-操作文件基本演示

课22：IO流-练习-复制MP3

课23：IO流-演示键盘录入

课24：IO流-读取键盘录入

课25：IO流-转换流

课26：IO流-转换流-需求演示

课27：IO流-流的操作基本规律

课28：IO流-流的操作基本规律-需求体现1

课29：IO流-流的操作基本规律-需求体现2

转换流：

InputStreamReader:字节到字符的桥梁。解码。

OutputStreamWriter:字符到字节的桥梁。编码。

流的操作规律：

之所以要弄清楚这个规律，是因为流对象太多，开发时不知道用哪个对象合适。

想要知道开发时用哪些对象。只要通过四个明确即可。

1，明确源和目的（汇）。

源：InputStream Reader

目的： OutputStream Writer

2，明确数据是否是纯文本数据。

源：是纯文本： Reader

否：InputStream

目的：是纯文本：Writer

否：OutputStream

到这里，就可以明确需求中具体要使用哪个体系。

3，明确具体的设备。

源设备：

硬盘：File

键盘：system.in

内存：数组

网络：Socket流

目的设备：

硬盘：File

控制台：system.out

内存：数组

网络：Socket流

4，是否需要其他额外功能。

1，是否需要高效（缓冲区）；

是，就加上buffer。

2，转换。

需求1：复制一个文本文件。

1，明确源和目的。

源：InputStream Reader

目的：OutputStream Writer

2，是否是纯文本？

是！

源：Reader

目的：Writer

3，明确具体设备。

源：

硬盘：File

目的：

硬盘：File

FileReader fr = new FileReader("a.txt");

FileWriter fw = new FileWriter("b.txt");

4，需要额外功能吗？

需要，需要高效。

BufferedReader bufr = new BufferedReader(new FileReader("a.txt"));

BufferedWriter bufw = new BufferedWriter(new FileWriter("b.txt"));

==================================================================

需求2：读取键盘录入信息，并写入到一个文件中。

1，明确源和目的。

源：InputStream Reader

目的：OutputStream Writer

2，是否是纯文本？

是！

源：Reader

目的：Writer

3，明确具体设备。

源：

键盘：system.in

目的：

硬盘：File

InputStream in = system.in;

FileWriter fr = new FileWriter("b.txt");

这样做可以完成，但是麻烦。

4，需要额外功能吗？

需要。转换。将字节流转换成字符流，因为明确的源是Reader，这样操作文本数据便捷。

所以要将已有的字节流转成字符流。使用字节-->字符：InputStreamReader。

InputStreamReader isr = new InputStreamReader(System.in);

FileWriter fw = new FileWriter("b.txt");

还需要功能吗？

需要：想高效。

BufferedReader bufr = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufw = new BufferedWriter(new FileWriter("b.txt"));

=====================================

需求3：将一个文本文件数据显示在控制台上。

1，明确源和目的。

源：InputStream Reader

目的：OutputStream Writer

2，是否是纯文本？

是！

源：Reader

目的：Writer

3，明确具体设备。

源：硬盘 File

目的：控制台 system.in

FileReader fr = new FileReader("a.txt");

OutputStream out = system.out;//PrintStream

4，需要额外功能吗？

需要。转换。

FileReader fr = new FileReader("a.txt");

OutputStreamWriter osw = new OutputStreamWriter(system.out);

还需要功能吗？

需要：想高效。

BufferedReader bufr = new BufferedReader(new FileReader("a.txt"));

BufferedWriter bufw = new BufferedWriter(new OutputStreamWriter(system.out));

===================================

需求4：读取键盘录入数据，显示在控制台上。

1，明确源和目的。

源：InputStream Reader

目的：OutputStream Writer

2，是否是纯文本？

是！

源：Reader

目的：Writer

3，明确具体设备

源：键盘system.in

目的：控制台system.out

InputStream in = system.in;

OutputStream out = system.out;

4，明确额外功能？

需要。转换。因为都是字节流，但是操作的却都是文本数据。

InputStreamReader isr = new InputStreamReader(system.in);

OutputStreamReader osw = new OutputStreamReader(system.out);

为了将其高效。

BufferedReader bufr = new BufferedReader(new InputStreamReader(system.in));

BufferedWriter bufw = new BufferedWriter(new OutputStreamReader(system.out));

课30：IO流-转换流的编码解码

需求：将一个中文字符串数据按照指定的编码表写入到一个文本文件中。

1，目的。OutputStream Writer

2，纯文本，Writer

3，设备：硬盘File

FileWriter fw = new FileWriter("a.txt");

fw.write("你好");

注意：既然需求中已经明确了指定编码表的动作。

那就不可以使用FileWriter，因为F内部使用默认的本地码表。

只能使用其父类OutputStreamWriter。

OutputStreamWriter接收一个字节输出流对象，既然是操作文件，那么该对象是FileOutputStream。

OutputStreamWriter osw = new OutputStreamWriter(new FileOutputStream("b.txt"),charsetName);

需要高效吗？

BufferedWriter bufw = new BufferedWriter(new OutputStreamWriter(new FileOutputStream("b.txt"),charsetName));

什么时候使用转换流呢？

1，源或者目的对应的设备是字节流，但是操作的是文本数据，可以使用转换流作为桥梁。

提高对文本操作的便捷。

2，一旦操作文本涉及到具体的指定编码表时，必须使用转换流。

课31：IO流-File对象-构造函数&字段

流只能操作数据！

File类：

用来将文件或者文件夹封装成对象

方便对文件与文件夹的属性信息进行操作。

File对象可以作为参数传递给流的构造函数。

了解File类中的常用方法。

知识点（关于文件路径）：

前缀概念用于处理UNIX平台上的根目录，并在Microsoft Windows平台上驱动说明符，根目录和UNC路径名，如下所示：

对于UNIX平台，绝对路径名的前缀始终为"/" 。 相对路径名没有前缀。 表示根目录的抽象路径名具有前缀"/"和空名称序列。

对于Microsoft Windows平台，包含驱动器说明符的路径名的前缀由后面跟着":"的驱动器号组成，如果路径名是绝对的，则可能后跟"\\" 。 UNC路径名的前缀为"\\\\" ; 主机名和共享名称是名称序列中的前两个名称。 没有指定驱动器的相对路径名没有前缀。

课32：IO流-File对象-常见功能-获取

/\*

\* File对象的常见方法。

\*

\* 1，获取。

\* 1.1获取文件名称。

\* 1.2获取文件路径（相对路径\绝对路径）。

\* 1.3获取文件大小。

\* 1.4获取文件修改时间。

\* \*/

课33：IO流-File对象-常见功能-创建和删除

课34：IO流-File对象-常见功能-判断

课35：IO流File对象--常见功能-重命名

课36：IO流-File对象-常见功能-系统根目录和容量获取

课37：IO流-File对象-常见功能-获取目录内容

课38：IO流-File对象-常见功能-过滤器

课39：IO流-File对象-练习-深度遍历文件夹

课40：IO流-File对象-练习-递归

课41：IO流-File对象-练习-删除目录

课42：IO流-Properties集合的-基本功能

课43：IO流-Properties集合的-list方法

课44：IO流-Properties集合的-store方法

#：#标注的信息都是注释信息。

课45：IO流-Properties集合的-修改配置信息

课46：IO流-Properties集合的-练习

课47：IO流-综合练习-文件清单列表

课48：IO流-打印流-PrintStream

课49：IO流-打印流-PrintWriter

课50：IO流-序列流-SequenceInputStream

打印流：

PrintWriter与PrintStream

可以直接操作输入流和文件。

序列流：

SequenceInputStream

对多个流进行合并。

操作对象：

ObjectInputStream与ObjectOutputStream

被操作的对象需要实现Serializable（标记接口）。

课51：IO流-序列流-SequenceInputStream-枚举和迭代

课52：IO流-文件切割

课53：IO流-文件合并

课54：IO流-文件切割合并+配置文件

课55：IO流-ObjectOutputStream-对象的序列化

课56：IO流-ObjectOutputStream-对象的反序列化

课57：IO流-序列化接口-Serializable

课58：IO流-关键字-transient

课59：IO流-RandomAccessFile-写入

（Access：访问）

课60：IO流-RandomAccessFile-读取&随机读取

课61：IO流-RandomAccessFile-随机写入&细节

课62：IO流-管道流-PipeStream

课63：IO流-操作基本类型数据的流对象-DataStream

IO包中的其他类：

操作基本数据类型：

DataInputStream与DataOutputStream。

操作字节类型：

ByteArrayInputStream与ByteArrayOutputStream。

操作字符数组类型：

CharArrayReader与CharArrayWrite。

操作字符串类型：

StringReader与StringWriter。

课64：IO流-操作数组的流

关闭ByteArrayInputStream，ByteArrayOutputStream没有任何效果，因为该对象不调用底层资源，仅在内存中运行。

课65：IO流-编码表

ASCII:美国标准信息交换码、

用一个字节的7位可以表示。

ISO8859-1：拉丁码表。欧洲码表

用一个字节的8位表示。

GB2312：中国的中文编码表。

GBK：中国的中文编码表升级，融合了更多的中文文字符号。

Unicode：国际标准码，融合了多种文字。

所有文字都用两个字节来表示，Java语言使用的就是Unicode。

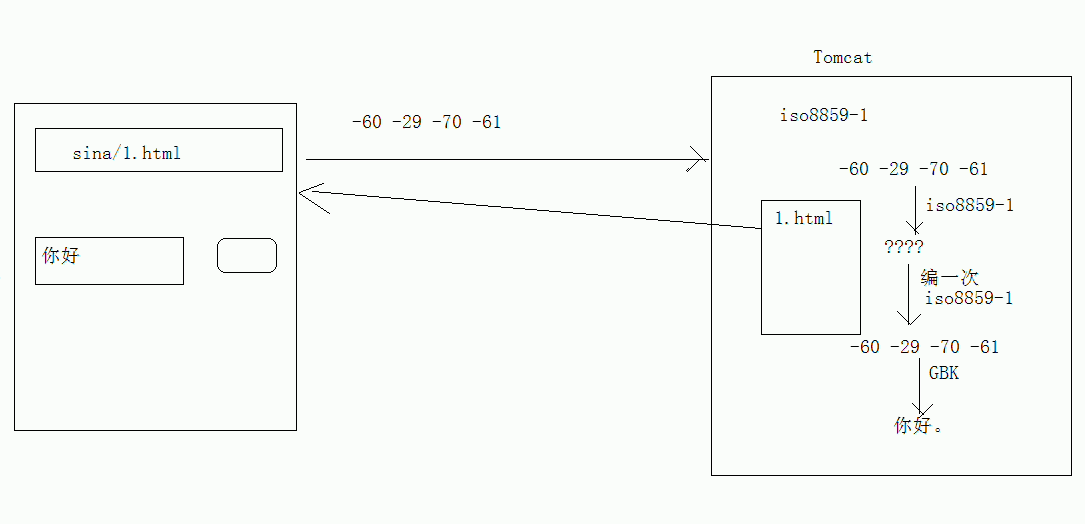
UTF-8：（Unicode Transform Format）最多用三个字节来表示一个字符。

...

？：位置字符。

课66：IO流-简单编码解码

课67：IO流-编码解码问题-1



课68：IO流-编码解码问题-2

课69：IO流-联通问题

课70：IO流-练习-按字节截取字符串

Day25:

课1：GUI-概述

1.（Graphical User interface图形化用户接口）

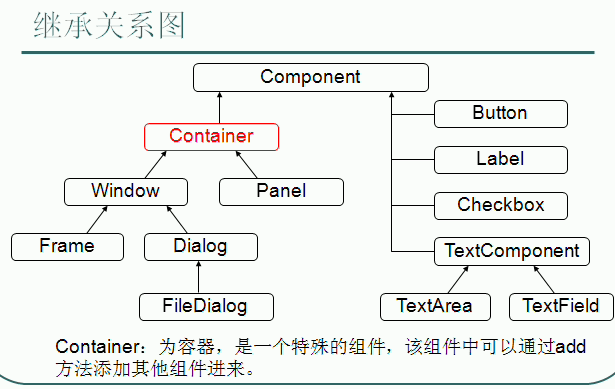
2. CLI：（Command line interface），即常见的DOS命令行操作。

Java为GUI提供的对象都存在Java.awt和Javax.swing两个包中。

Java.awt（abstract window toolkit）抽象窗口工具包。

需要调用本地系统方法实现功能，属重量级控件。

Javax.swing：在awt的基础上。建立的一套图形界面系统，其中提供了更多的组件，而且完全有Java实现。增强了移植性，属轻量级控件。





布局管理器

容器中的组件的排放方式，就是布局。

常见的布局管理器：

FlowLayout（流式布局管理器）

从左到右的顺序排列。

Panelmorning的布局管理器。

BorderLayout（边界布局管理器）

东南西北中。

Frame默认的布局管理器。

GridLayout（网络布局管理器）

选项卡

GraiBageLayout（网络包布局管理器）

非规则的矩阵。

课2：GUI-FRAME演示

多后台开启后，可以点击右下方（Display Selected Console），终结（terminated）各个后台。

课3：GUI-事件监听机制

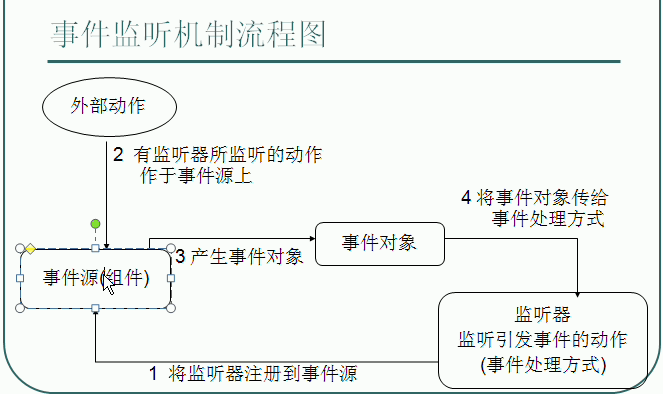
事件监听机制组成：

事件源（组件），

事件（Event），

监听器（Listener），

事件处理（引发事件后处理方式）。



WindowAdapter和WindowListener的区别：

WindowAdapter是个抽象类，里面对接口WindowListener的方法都进行了空实现。  
这样编程时不需要自己再去实现全部接口WindowListener里的方法。只需覆写自己需要的方法，其他的方法，WindowAdapter都实现了。

// 事件监听机制。

f.addWindowListener(new WindowAdapter() {

@Override

public void windowClosing(WindowEvent e) {

// super.windowClosing(e);//不需要自己操作，内部会自动运行。

// System.out.println("closing......"+e);

System.exit(0);

}

});

课4：GUI-ActionListener演示

课5：GUI-鼠标事件

课6：GUI-键盘事件

课7：GUI-Swing演示&装插件

课8：GUI-练习-列出目录内容

课9：GUI-菜单

课10：GUI-练习

Day28:

课1：反射机制-概述&应用场景

课2：反射机制-细节&Class对象

/\*

\* JAVA反射机制是在运行状态中，对于任意一个类（class文件）。都能够知道这个类的所有属性和方法，

\* 对于任意一个对象，都能够调用它的任意一个方法和属性；

\* 这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。

\*

\* 动态获取类中信息，就是java反射。

\* 可以理解为对类的解剖。

\*

\* 要想对字节码文件进行解剖，必须要有字节码文件对象。

\* 如何获取字节码文件对象呢？

\* \*/

课3：反射机制-获取Class对象的三种方法

/\*

\* 获取字节码对象的方式：

\* 方式1，Object类中的getclass方法。

\* 想要用这种方式，必须明确具体的类，并创建对象。

\*

\* 方式2，任何数据类型都具备一个静态的属性.class来获取其对应的class对象。

\* 相对简单，但是还是要明确用到类中的静态成员。

\*

\* 方式3，只要通过给定的类的字符串名称就可以获取该类，更为扩展。

\* 可以用Class类中的方法完成。

\* 该方法就是forName。

\* 这种方式只要有名称即可，更为方便，扩展性能更强。

课4：反射机制-获取Class中的构造函数

Class是把所有类封装为一个Class 类，一个个.class文件就是一个Class对象。  
搞清楚万物皆对象，类也可以是对象。

对象在类中产生，那么类（字节码，在虚拟机中就叫字节码）在哪产生呢。答案是java中的Class（java.lang.Class）。那么在装进虚拟机的时候 这时候的类就叫字节码了 这些个字节码\*\*\*归属于这个Class，是他的对象，所以我们就叫这个时候字节码是字节码对象。

field:字段；method：方法；access：访问。

/\*

\* 当要获取指定名称对应类中所体现的对象时，

\* 而该对象初始化不使用空参数构造，该怎么办？

\*

\* 既然是通过指定的构造函数进行对象的初始化。

\* 所以应该先获取到该构造函数。通过字节码文件对象即可完成。

\* 该方法是：getConstructor（paramterTypes）;（任何类型都可以被class所描述）

\* \*/

// 早期：new时候。先根据被new的类的名称找寻该类的字节码文件，并加载进内存，并创建该字节码文件对象，

// 再接着创建该字节文件对应的Person对象。

// cn.itcast.bean.Person p = new cn.itcast.bean.Person();

// 现在：

String name = "cn.itcast.bean.Person";

// 找寻该名称类文件，并加载进内存，再产生Class对象。

Class clazz = Class.forName(name);

// 如何产生该类的对象呢？（创建新实例）

Object obj = clazz.newInstance();

// 获取到了指定的构造函数的对象。

Constructor constructor = clazz.getConstructor(String.class,int.class);

// 通过该构造器对象的newInstance方法进行对象的初始化。

Object obj = constructor.newInstance("小强",33);

课5：反射机制-获取Class中的字段

// 只能获取公有的。

Field field = null;//clazz.getField("age");//java.lang.NoSuchFieldException:

// 只获取本类，但包含私有。

field = clazz.getDeclaredField("age");

// 对私有字段的访问取消权限检查。暴力访问。

field.setAccessible(true);

Object obj = clazz.newInstance();

field.set(obj, 89);

Object o = field.get(obj);

// java.lang.IllegalAccessException

课6：反射机制-获取Class中的方法

// 获取的都是公有的方式。

Method [] method = clazz.getMethods();

// 获取本类中的所有方法，包含私有。

method = clazz.getDeclaredMethods();

for(Method med : method){

System.out.println(med);

}

课7：反射机制-反射练习

课1：正则表达式（Regex:regular expression）-概述

/\*

\* 正则表达式

\*

\* 正则表达式用于操作字符串数据。

\* 通过一些特定符号来体现的。

\* 所以我们为了掌握正则表达式，必须要学习一些符号。

\*

\* 虽然简化了，但是阅读性差。

\*

\* \*/

课2：正则表达式-常见的规则

/\*

\* 正则表达式对字符串的常见操作：

\* 1，匹配。

\* 其实使用的就是String类中的matches方法。

\*

\* 2，切割。

\* 其实使用的就是String类中的split方法。

\* 组：((A)(B(C)))

\*

\* 3，替换。

\* 其实使用的就是String类中的replaceAll()方法。

\*

\* 4，获取。

\* 将正则规则进行对象的封装。

\* Pattern p = Pattern.compile(a\*b);

\* 通过正则对象的matcher方法字符串相关联，获取要对字符串操作的匹配器对象Matcher。

\* Matcher m = p.matcher("aaaaab");

\* 通过Matcher匹配器对象的方法对字符串进行操作。

\* boolean b = m.matches();

\*

\*/

注意：

特殊符号"."。本身，"."代表特殊符号含义。

// 组的概念。封装成某组（.）\\x ：第x组。

String [] names3 = str3.split("(.)\\1+");

课3：正则表达式-常见功能-匹配

见上

课4：正则表达式-常见功能-切割

见上

课5：正则表达式-常见功能-替换

见上

课6：正则表达式-常见功能-获取

见上

课7：正则表达式-练习1

治疗口吃：我我我...我我...我我我要...要要要要...要要要...学学学学学...学学编编...编编编编..编.程程..程程.程程...程

String str = "我我我...我我...我我我要...要要要要...要要要...学学学学学...学学编编...编编编编..编.程程..程程.程程...程";

// 1，将字符串中.去掉，用替换。

str = str.replaceAll("\\.+", "");

// System.out.println(str);

// 2，替换叠词。

str = str.replaceAll("(.)\\1+","$1");

System.out.println(str);

}

课8：正则表达式-练习2

对IP地址排序。192.168.10.34 127.0.0.1 3.3.3.3 195.70.11.55

/\*

\* 1，为了让ip可以按照字符串顺序比较，只要让ip的每一段的位数相同。

\* 所以，补零，按照每一位所需，作多0进行补充。每一段都加两个0。

\* \*/

String ip\_str = "192.168.10.34 127.0.0.1 3.3.3.3 195.70.11.55";

ip\_str = ip\_str.replaceAll("(\\d+)", "00$1");

// 然后每一段保留数字3位。

ip\_str= ip\_str.replaceAll("0\*(\\d{3})", "$1");

System.out.println(ip\_str);

// 将IP地址切出。

String [] ips = ip\_str.split(" +");

TreeSet<String> ts = new TreeSet<String>();

for(String ip : ips){

ts.add(ip);

}

for(String ip : ts){

System.out.println(ip.replaceAll("0\*(\\d+)", "$1"));

}

课9：正则表达式-练习3

对邮件地址校验。

String mail = "abc1@sina.com.cn";

String regex = "[a-zA-Z0-9\_]+@[a-zA-Z0-9]+(\\.[a-zA-Z]{1,3})+";

// regex = "\\w+@\\w+(\\.\\w+)+";

boolean b = mail.matches(regex);

System.out.println(mail+":"+b);

课10：正则表达式-练习4-爬虫

## Day1:

课1：网络模型

OSI（Open System Interconnection）开放系统互联参考模型

TCP/IP参考模型

网络通讯要素

IP地址

端口号

传输协议



1. 主要定义物理设备标准，如网络的接口类型，光纤的接口类型，各种传输介质的传输速率等。主要作用是：传输比特流，（通过0、1进行传输，数模转换和模数转换）
2. 将从物理层接收的数据进行MAC地址（网卡的的地址）封装与解封装，数据（帧）通过交换机进行交换。
3. 将从数据链路层获得的帧进行IP封装与解封装，通过路由器进行工作。该数据是数据包。
4. 定义了数据的协议和端口号，如TCP、UDP（QQ聊天数据） 。从网络层获取的数据进行分段和传输。到达后再进行重组，该数据是段。
5. 通过传输层建立数据传输的通路。在系统之间发起会话和接收会话请求。
6. 主要是对接收的数据进行解释、加密与解加密、压缩与解压缩。
7. 应用层，主要是一些终端的应用。如FTP的文件下载，WEB的IE浏览，QQ之类的。

课2：

课3：

IP地址：InetAddress

1. 网络中设备的标识
2. 不易记忆，可用主机名
3. 本地回环地址：127.0.0.1 主机名：localhost

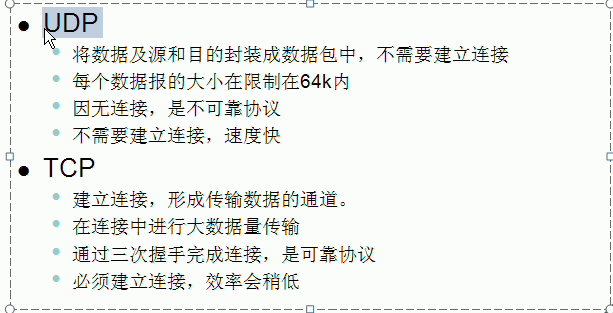
端口号

1. 用于标识进程的逻辑地址，不同进程的标识。
2. 有效端口：0~65535，其中0~1024系统使用或保留端口。

传输协议

1. 通讯的规则
2. 常见协议：TCP，UDP

课4：



课5：IP对象\_InetAddress

该对象没有构造函数，故不能直接调用。

课6：

.org：非盈利组织

.com：盈利

DNS（Domain Name System，域名系统） 域名解析器

C:\Windows\System32\drivers\etc：hosts文件可以查看本机的域名

域名解析：首先，会先访问本机的域名，然后再对互联网的域名访问解析。

课7：

Socket：插座，套接字

Socket就是为网络服务提供的一种机制。

通信的两端都有Socket。

网络通信其实就是Socket间的通信。

数据在两个Socket间通过IO传输。

UDP传输

* DatagramSocket
* （此类表示用于发送和接收数据报数据包的套接字。 ）
* DatagramPacket
* （该类表示数据报包。

数据报包用于实现无连接分组传送服务。 仅基于该数据包中包含的信息，每个消息从一台机器路由到另一台机器。 从一台机器发送到另一台机器的多个分组可能会有不同的路由，并且可能以任何顺序到达。 包传送不能保证。 ）

建立发送端，接收端

建立数据包。

调用Socket的发送接收方法。

关闭Socket。

发送端 与接收端是两个独立的运行程序。

课10：TCP传输

Socket

此类实现客户端套接字

和

ServerSocket

此类实现服务器套接字

建立客户端和服务器端

建立连接后，通过Socket中的IO流进行数据的传输

关闭socket

同样，客户端与服务器端是两个独立的应用程序。