# 软件：

一系列按照特定顺序组织的计算机数据和指令的集合。

软件的出现实现了更好的人际交互，图形界面更是提供了友好的交互方式。

# 命令提示符（常见Dos命令）

cd/ 退回到根目录

cd user\koterwong 进入到指定目录

cd user\koter\* \*为通配符

cd.. 退回上一目录

dir 列出该目录所有文件

md koterwong 创建指定目录 //保证该文件夹没有子目录

rd koterwong 删除指定目录

echo kobe>koterwong.txt 创建文件

dir koterwong.txt 删除文件

# java 的跨平台性

Java程序运行(JVM)虚拟机中。不同的系统对应不同版本的虚拟机，因为不同系统的内核不同，虚拟机运行过程中可能会调用到系统底层的东西(例如调用到系统的时间)。但是JAVA程序只依赖虚拟机。

Android虚拟机

# java

|---JDK Java Development Kit(JDK)

|---Java Runtime Environment(JRE)

|--- Java Virtual Machine(JVM)

# java 环境变量配置

JAVA\_HOME //动态的获取某一环境变量的值

设置JDK 安装目录，后期运行tomcat,eclipse等都需要此变量。设置 PATH 和 CLASSPATH 的时候，也可以方便设置。

Path //在命令行下执行(.exe可执行文件)，优先在该目录下找，找不到再去系统配置的path找

% JAVA\_HOME%\bin; //javac -version 查看环境变量是够配置成功

配置到这里就可以在Dos命令行中运行Java程序。

CLASSPATH环境变量。

它是javac编译器的一个环境变量。它的作用与import、package关键字有关。当我们通过improt java.util.\*时引入该包下的类的时候，编译器搜索的路径。直接设置为

CLASSPATH = .;%JAVA\_HOME%\lib “.”是先在该工程下寻找

环境变量临时配置方式：

Set 查看所有环境变量的配置

Set path 查看path环境变量的值

Start 打开新的Dos命令行窗口

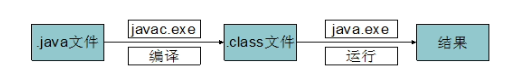
# 在命令行下运行Java程序

HelloWorld.java

* Javac HelloWorld.java 编译成.class 文件()

启动编译器 编译成虚拟机识别的二进制文件.class

* Java HelilWorld //启动java虚拟机运行java程序。



JAVA语言组成

1. 标识符
2. 程序中自定义的一些名称
3. 由26个英文字母大小写，数字，\_$组成
4. 规则： 不能数字开头 、 不可可使用关键字
5. 区分大小写。
6. 为了提高阅读性，见名知意。
7. Java命名规范
8. 包名 多个单词组成，所有字母小写。
9. 类名，接口名 多单词组成 所有单词的首字母大写
10. 变量名，方法名 多单词组成，第一个字母小写，第二个字母开始每个首字母大写
11. 常量名 所有字母都大写，多单词用下划线连接。XXX\_YYY\_ZZZ。
12. 常量 表示不能改变的数值final修饰的
13. 整数常量。所有整数
14. 小数销量。所有小数
15. 布尔型常量。 true , false
16. 字符型常量。 将一个数字字母或者符号用单引号(' ')表示
17. 字符串常量。 讲一个或者多个字符用双引号标示 “ ”
18. Null常量。 只有一个数值null

* Java整数的标示方式

1. 十进制。 0-9，满十进一
2. 八进制。 0-7，满八进一
3. 十六进制。 0-9、A-F，满十六进一。用0x开头表示，例如oxffff。

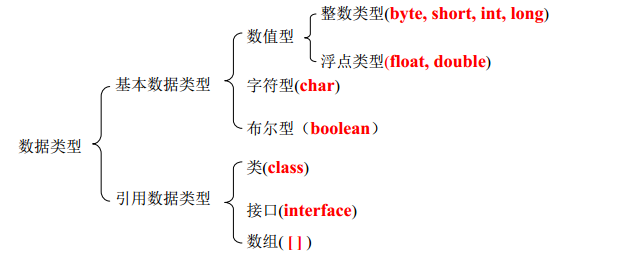
负数的二进制 计算机存储的是反码。

正数的原、反、补码相同  
负数的原码最高位是1，即代表负；  
反码就是后七位按位取反；  
负数补码就是反码加1。

1. 变量 将不确定的数据进行存储，也就是需要在内存中开辟一个空间。

开辟方式 ----> 数据类型 变量名称 变量名

Java对每一种数据又明确定义数据类型，在内存中分配了不同的大小空间。



1. 类型转换

自动类型转换(也叫隐示类型转换)

数值的自动类型转换

低--------------------------------------------->高

byte,short,char-> int -> long -> float -> double

|  |  |  |
| --- | --- | --- |
| 操作数类型1 | 操作数类型2 | 转换后的类型 |
| byte、short、char、int | int | int |
| byte、short、char、int、long | long | long |
| byte、short、char、int、long、float | float | float |
| byte、short、char、int、long、float | long | long |

强制类型转换(也叫显示类型转换)

byte b = 3;

b = b + 4;//报错

b = (byte)b+4;//强制类型转换，强制将b+4的结果转换为byte类型，再赋值给b。

1. 运算符

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| + | 加法 | 1 + 2 |
| - | 减法 | 4 - 3.4 |
| \* | 乘法 | 7 \* 1.5 |
| / | 除法 | 3.5 / 7 |
| % | 取余 | 7 % 2 |
| ++ | 自增 | 3++ |
| -- | 自减 | 3-- |

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| & | 与 | 都是1才是1 |
| | | 或 | 有1就是1 |
| ^ | 异或 | 相同0不同1 |
| ~ | 非 | ~4 |
| << | 左移 | 5 << 1 5\*2 |
| >> | 右移 | 6 >> 1 6/2 |

算数运算符 比较运算符 位运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 说明 | 举例 |
| > | 大于 | a > 4.2 |
| >= | 大于等于 | 3.4 >= b |
| < | 小于 | 1.5 < 9 |
| <= |  | 6 <= 1 |
| == | 等于 | 2 == 2 |
| != | 不等 | 2 != 2 |
| && | 与 | true && false |
| || | 或 | (3 > 1) || (2 == 1) |
| ！ | 非 | ！true |

&和&&的特点：

&:无论左边是true是false。右边都运算。

&&:当左边为false时，右边不运算。

|：两边都参与运算。

||：当左边为true。右边不运算

应该注意的问题：

• 负数取模System.out.println(1%-5+"和"+-1%5); //输出结果为1和-1，结果只看被模数

• int a = 4343 ; a = a/1000\*1000; 结果为4000 int int计算还是int

• “+” 在输出语句中的作用，是连接字符串，将非字符串转化成字符串。

位移运算符：

<< 结果为乘以 2 的移动的位数的幂

>> 结果为除以 2 的移动次数的幂

左移动后最高位补什么又原有的最高位而定0补0 ，1补1。

右移则统统补0。

1. 转义字符通过反斜线"\"，来转变字母或符号的含义

\b 退格 \t 制表符(Tab) \n 换行 \f \r Enter 键

\" \'

\\ 将 \ 从转义字符转义成 字符 \

System.out.println("\\\\"); //打印结果为 \\

# Java语言结构

1. 顺序结构
2. 控制结构
3. 循环结构

if else 简写格式是三目运算符： 变量 = (条件表达式)?表达式1:表达式2

好处：简化代码。弊端：因为是一个运算符，所以运算必须要有一个结果。

int x = 3;

switch(x) { //可以接受的类型 byte short int char string enum

case 1: // 必须要有break; 否则有一个匹配，则下面都会得到执行。

break;

default : //没有匹配项执行。

break;

}

//if 和 Switch都用于判断，具体场景如何选择呢？

1. 如果是数值判断，判断的数值不多，并且都符合byte short int char这地种类型。

建议Switch语句，因为效率更高。

1. 其他情况，对区间判断，对Boolean类型判断，使用if，if使用范围更广泛

循环结构的思想

1. 累加思想，通果变量记录每次变化的结果。
2. 计数器思想，通过一个变量，记录数据状态发生的次数。

While 和 for 的区别：

while与for可以互换

1. for循环定义的变量，在for循环结束就是在内存中释放。

while循环使用的变量在循环结束后还可以继续使用。循环计数count++

1. 无限循环格式： while(true) , for(;;),无限循环存在的原因是并不知道循环多少次，而是根据某些条件，来控制循环。

# Java中的方法

* 方法只有被调用才会被执行
* 定义方法可以将功能代码进行封装，便于对该功能进行复用，提高了代码的复用性
* 对于方法没有具体返回值的情况，返回值类型用关键字void表示，那么该方法中的return语句如果在最后一行可以省略不写。return：用于结束函数。

 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。

方法的重载

* + 类中，允许存在一个以上的同名方法，只要它们的参数个数或者参数类型不同即可
  + 与返回值类型无关，只看参数列表。
  + 方便了阅读，优化了程序设计。

# Java数组

同一类型数据的集合，其实是一个容器，好处是对数据进行编号，方便操作这些元素。

* int [] a = new int [4];
* int a [] = new int [] {5,1 ,3};
* 简写 :元素类型[] 数组名 = {元素，元素， ……};
* 二维数组 int[] x,y[]; x是一维数组， y是二维数组。
  + 格式1： int[][] arr = new int[3][2]; arr[0] 表示第一个一维数组
  + 格式2： int[][] arr = new int[3][];

 二维数组中有3个一维数组，每个一维数组都是默认初始化值null

可以对这个三个一维数组分别进初始化

* 格式3： int[][] arr = {{3,8,2},{2,7},{9,0,1,6}};
  1.  定义一个名称为arr的二维数组
  2.  二维数组中的有三个一维数组
  3.  每一个一维数组中具体元素也都已初始化
  4.  第一个一维数组 arr[0] = {3,8,2};
  5.  第二个一维数组 arr[1] = {2,7};
  6.  第三个一维数组 arr[2] = {9,0,1,6};
  7.  第三个一维数组的长度表示方式： arr[2].length;

Java Arrays类常用方法：

定义了一系列的数组操作的静态方法。

Arrays.*sort*(array); //对数组进行排序

String string = Arrays.*toString*(array); //将数组转化成字符串

//判断数组中是否包含值，查找

String[] stringArray = { "a", "b", "c", "d", "e" };

boolean b = Arrays.asList(stringArray).contains("a");

编写一个程序，具有排序的功能(选择，冒泡。等等)

编写一个程序，具有查找的功能(遍历查找(效率低)，折半查找(方法不一)必须有序)。

## Java内存结构

为了提高效率，Java程序运行时，虚拟机会在内存中开辟不同的地址，不同的地址空间存放不同的数据，形成了Java 语言的逻辑关系。

* 栈内存 调用方法时，用于存储局部变量，当数据使用完，所占空间会自动释放。
* 堆内存 数组和对象，通过new建立的实例都存放在堆内存中。
  1. 每一个实体都有内存地址值
  2. 实体中的变量都有默认初始化值
  3. 实体不在被使用，会在不确定的时间内被垃圾回收器回收
* 方法区(共享区)(数据区)。 分类静态区，非静态区，存放方法体。常量池，字符串常量
* 本地方法区
* 寄存器

Java面向对象

将功能封装成对象，强调对象的功能。完成某件事情，我们要操纵对象去完成我们想要做的事情。 它是一种思想，在编码中，要寻找能完成功能的对象，创建这些功能的对象。

* 一个很好的例子就是，把大象放进冰箱。 面向对象只强调冰箱这个对象具有的，打开、存储、关闭这三个功能。

面向对象的特点：

*  是一种符合人们思考习惯的思想
*  可以将复杂的事情简单化
* 提高对象功能的复用性。
*  将程序员从执行者转换成了指挥者。指挥对象具有的功能。

# 类与面向对象：

 开发的过程：其实就是不断的创建对象，使用对象，指挥对象做事情。

 设计的过程：其实就是在管理和维护对象之间的关系。

面向对象的三大特征：

• 封装(encapsulation)

• 继承(inheritance)

• 多态(polymorphism)

成员变量和局部变量

* 成员变量在堆内存中 ，成员变量，系统默认赋值。 String 默认=null；
* 局部变量，在栈内存中。

匿名对象： new Class();

* 当对象的方法只调用一次时，可以使用匿名对象
* 要调用的方法，参数为类的引用(Person person)。匿名对象可作为实参传递。

匿名对象传递参数，节省栈内存中的空间。

# 面向对象特点之： 封装encapsulation

封装：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。
* 把属性都隐藏，提供公共方法对其访问。

Private 权限修饰符：

* 仅仅是封装的一种体现形式
* 把成员变量设置为私有，提高了代码的健壮性。
* 创建对象的时候，私有的成员变量不能访问，但提供的set，get方法。注意。set，get方法设置和获取的是该对象的变量，而不是类的成员变量。

构造函数

默认的构造函数权限和类的权限一致。

构造代码块：

创建对象的时候执行，并且优先级高于构造方法。

和构造方法都可用于初始化成员变量。但是构造代码块是对所有对象的共性进行初始化，而构造方法只初始化本对象。

# This关键字

this 代表其所在函数所属对象的引用。即调用者的对象。

* 在构造方法中

this.name = name ; //用于对该对象的成员变量初始化

* this的应用：当定义类中功能时，该函数内部要用到调用该函数的对象时，这时用this来表示这个对象。 但凡本类功能内部使用了了本类对象，都用this表示。

public boolean compare(Person p){

return this.age==p.age;

}

* this 用于构造函数间互相调用。

this(name) //调用到只有一个参数的构造方法

* this关键字的省略（非静态方法相互调用）

类中的成员调用都需要用到对象。省略了this关键字

void test(){

System.out.println("this关键字的省略");

(this.)test1();//调用test1方法，即省略了this。this指调用test的对象

}

void test1(){

}

* static方法相互调用，同样也省略了关键字，即方法所属的类名。

public static void main(String[] args) {

test(); //main方法是静态方法，只能调用静态方法。省略了当前类名

}

static void test(){

System.out.println("this关键字的省略");

}

# Static 静态关键字

* 随着类的加载而加载,。

用到该类的内容的时候就会加载，例如创建对象，调用静态方法

当类加载到内存中，静态变量就在内存中开辟空间。随着类的消失而消失，生命周期最长。而成员变量随着对象的消失而消失。

静态变量存放在方法区。

* 优先于对象存在 //先被加载到内存，并且存放空间不同。
* 被对象所共享 //你创不创建对象，我都在哪里。
* 通过类名直接调用 //优先于对象存在。

静态使用注意事项

* 静态方法(方法块)只能访问静态成员，（静态先于对象，成员变量属于对象，没有对象既不能访问成员变量） 反过来，非静态可以调用非静态，静态变量
* 静态方法中不可以定义this，super关键字this,super同样代表对象，没有对象

就不可以调用，（但是静态代码块中可以调用this关键字来访问对象的属性）

* Main方法是静态的。不能调用非静态成员和方法。

静态有利有弊

* 利：对共享数据进行单独空间存储，节省空间。可以直接通过类名调用。
* 弊：生命周期过长。访问出现局限性（静态虽好，只能访问静态）
* 使用static变量 ：存在所有对象共享的数据，定义成静态变量。
* 使用static代码块：随着类的加载而执行，只执行一次，并优先于主函数

用于给类进行初始化。主函数需要，先被JVM调用。

* 使用static方法 ： 该方法没有使用到成员变量，或者该类无需创建对象，只是对功能的封装。
* 例如，一些常用的工具类 ，如果创建对象，即在堆内存中开辟空间。但是该对象并没有要操作的数据，所以，为了严禁行，将该类封装的方法用static修饰，通过类名直接调用。并把构造方法私有化，禁止创建对象。
* 工具类只保留 .class文件也能正常运行。虚拟机运行的是.class文件。

Main方法

* 是一个特殊的函数，程序的入口，被JVM调用
* Public 代表主函数访问权限最大
* Static 随着类的加载就已经加载
* Void 主函数没有集体的返回值
* main 不是关键字，但是是一个特殊的单词，可以被JVM识别。
* 函数的参数(String []args) 函数的参数，是一个字符数组。

JVM调用main时，传入 new String[]; 长度0 ； 内容为null；

拓展：

* Java HelloWorld koter wong 向main方法中出入字符数组

# 生成Java api(应用程序接口) 帮助文档

在命令行下，定位到.java 文件目录

Javadoc -d apihelp -author -version Util.java

//apihelp 帮助文档要存放的文件夹，在该目录下。

//-author -version 注明 作者和版本。

//Util.java 要生成的文档源文件。

生成的文档只会显示public修饰的变量和方法.

# 创建对象的执行过程。

Person person = new person(“koterwong”,18);

1.new 用到了Person.class。 所以先找到Person.class 文件并加载到内存。

2.初始化static变量和static代码块，if exit。

3.在堆内存中，开辟空间，并默认初始化对象的成员了变量。

4.执行该对象的初始化代码块 ，if exit

5.执行对象的构造方法。

6.将对象的引用(地址)赋给person。Person在栈内存中开辟空间。

调用方法执行过程。

person.setName(name); //调用对象的方法，成员方法。

Person.setName(name); //调用类的静态方法

区别：方法的局部变量都存放栈内存中。 但是成员方法，会调用到对象在堆内存中的属性。

而静态方法，通过类名调用，只会调用到方法区的资源。

## 设计模式

解决问题最行之有效的方法，java中存在23种设计模式

其中之一：单例设置模式：解决一个类在内存中只存在一个对象。

* 构造方法私有化
* 提供getInstance()方法

1. 饿汉式

class Single{ //用final 和static修饰。

private static final Single s = new Single();

private Single(){}

public static Single getInstance(){

return s;

}

}

1. 懒汉式

class Single{

private static Single single=null;

private Single() {} //构造方法私有化

//对象single是共享数据，多个线程使用必须加锁。

public synchronized static Single getInstance() {

if (single==null) { //如果为null，说明方法区没有赋值

return single = new Single();

}else { //返回方法区中，已经有引用的对象。

return single;

}

}

}

懒汉式的优化

class Single{

private static Single s = null;

private Single(){}

public static Single getInstance(){

if(s==null){ //减少判断锁的次数

synchronized(Single.class){

if(s==null)

s = new Single();

}

}

return s;

}

}

# 面向对象特点之：继承(inheritance)

* Java只支持单继承，多继承容易带来安全隐患
* Java用多实现(接口)的方式，代替了多继承的方式。
* Java体系多层继承体系。

Super关键字

super关键字的存在。 方法区，子类中

this 关键字的存在。 方法区，子类中

子类的构造方法中必须显示或隐士的调用父类的一个构造方法。

this super 在构造函数中只能出现在程序的第一行，先进行初始化操作。

1.变量

当子类重新定义了变量，super.name 调用父类的变量。 //重新定义父类变量不常见。

在堆内存中。如果子类重新定义

new B

num super

num this

num变量，创建对象的时候，会在

堆内存中开辟两个空间，分别保存

父类和子类的变量。通过，super

this，分别调用。子类方法区存在

this，和super引用。调用父类

资源

//子类中重新定义了num

class A{

int num = 4;

out{System.out.println(this.num)} //子类,this值父类对象，输出4

}

class B extends A{

int num = 5; //子类重新定义了num。

void show(){

System.out.println(super.num);

System.out.println(this.num); //默认省略了this

}

}

//子类中没有重新定义 ，其中super和this同时指向了num。

class A{

int num = 4;

}

class B extends A{

//int num = 5;

void show(){

System.out.println(super.num);

System.out.println(this.num);

}

}

2.方法的重写，

只能重写private 以上的方法。并且，子类方法的权限，大于或等于父类的方法。

重写的方法，必须名称，参数，返回值类型全部相同。

当子类重写了父类的方法，super.show() 调用父类的变量。

3.构造方法

为什么子类构造方法要调用父类的的构造方法？

因为父类中的数据，子类可以直接获取。所以在子类对象创建时，需要先检查父类中构造函数的初始化情况。进而获取父类初始化完成后的变量。

4.子类调用父类构造方法的方式

子类的所有构造函数，默认都会访问父类中空参的构造函数。隐式的super();

当父类中没有无参构造器时，子类必须有构造方法，并且手动的通过super语句来调用父类的构造器。

另外，子类的构造器也可以使用this()来调用本类的构造器，通过其他构造器访问父类的构造器。但是，子类中必须至少有一个构造器去访问父类的构造器。

final关键字

继承的出现打破了封装性，于是就出现了final关键字。

* final修饰的变量，是一个常量，只能被赋值一次
* final修饰的方法，不能被重写
* final修饰的类不能被继承
* 内部类定义在类中的局部位置上时，只能访问该局部变量都被final修饰的局部变量。

描述事物是，一些数据是固定的不需要改变，为了增强阅读性，都给这些值起个名字，由于这些值不需要改变，就加上final修饰。作为常量，命名方式XXX\_YYY,方便阅读，区分。

# 抽象类

* 抽象就是从多个事物中将共性的，本质的内容抽取出来。抽象描述事情不具体，
* 多个对象都具备相同的功能，但是功能具体内容有所不同，那么在抽取过程中，只抽取了功能定义，并未抽取功能主体，那么只有功能声明，没有功能主体的方法称为抽象方法。
* 例如：狼和狗都有吼叫的方法，可是吼叫内容是不一样的。所以抽象出来的犬科虽然有吼叫功能，但是并不明确吼叫的细节。

抽象类的特点

1. 抽象类和抽象方法必须用abstract关键字来修饰。
2. 抽象类不可以用new创建对象。因为调用抽象方法没有意义。
3. 抽象是从子类对象中，抽象出来的概念，现实有可能并不存在。例如上面抽象的狗和狼
4. 抽象类和一般类并没有太大不同。只不过抽象类定义了不确定的部分，也就时该事物的功能，需要明确出现，但是有无法定义主体。
5. 特殊： 抽象类中可以不定义抽象方法，目的是不让该类创建对象。（Java存在这样的类）

# 抽象类的最佳实践-----🡪Java设计模式之 ：模板方法

什么是模板设计模式？

在定义功能时，功能的一部分是确定的，另一部分是不确定的，并且确定的部分调用到了不确定的部分，那么将不确定的部分暴露出去，由该类的子类去完成。这样就提高了程序的拓展性和复用性。

下面例子最好的实现了该设计模式

//需求：获取一段程序的运行时间。

**public** **class** Demo{

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

SubTime subTime = **new** SubTime();

subTime.getTime();

}

}

**abstract** **class** GetTime{

**public** **final** **void** getTime() { //定义成final不允许子类重写

**long** startTime = System.*currentTimeMillis*();

runCode(); //这里就调用了子类重写的方法 相当于this.tunCode

**long** endTime = System.*currentTimeMillis*();

System.***out***.println("运行时间："+(endTime-startTime));

}

**public** **abstract** **void** runCode();

}

**class** SubTime **extends** GetTime{

**public** **void** runCode() {

**for**(**int** i = 0;i<1000;i++){

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

}

}

}

# Java接口 (增强了拓展性，降低了耦合性)

接口中的成员修饰符是固定的。

* 成员变量：public static final
* 成员方法： public abstract
* 接口中的变量和方法必须全部暴露出去
* 接口中成员变量是静态常量，可以直接通过接口名直接调用。也可以通过子类的类名或者对象直接调用。

接口中的小知识点：

1. 一个实现了多个接口，但是不同接口中的(必须名字和返回值都相同)方法相同，实现类只需要重写一个方法即可。
2. 实现了子接口（该接口继承了另一个接口），需要重写子接口与父接口的所有方法
3. 接口支持多继承，接口由于没有方法体，不存在类继承的安全隐患，所有接口就支持多继承

接口的特点：

* 接口是对外暴露的规则 (CPU主板的针脚)
* 接口是程序的功能拓展 (提高了主板的功能拓展，升级CPU)
* 降低了继承的耦合性(继承锁死了子类和父类的关系) (降低了CPU和主板的耦合性)
* 接口与接口之间可以有继承关系

关于耦合性

从多个功能中抽象出共性，定义抽象方法。这个抽象方法可以放在抽象类中或者接口。

从抽象类中获取这个方法，就需要继承这个类。但是有继承了抽象类中的其他非private方法，而这些方法不是子类想要继承的。这样就增加了耦合性。把这个抽象方法定义在接口就降低了这种耦合性。下面看一个列子

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

**abstract** **void** eat(); //定义子类

**void** smoke(){

System.***out***.println("有的人吸烟"); //子类默认继承了该方法，但是不是每//个人都吸烟，所有酒吧该方法拓展出去。定义在接口中，谁有吸烟的需求谁去实现。

}

}

**class** Man **extends** Person{

**void** eat() {System.***out***.println("人都需要吃饭")}

}

//将somke定义在接口中。对吸烟这个方法，谁需要谁进行拓展。

interface Smokeing{public abstract void smoking();}

abstract class Person{abstract void eat();}

class Smoker extends Person implements Smokeing{

void eat() {

System.out.println("人都需要吃饭");

}

public void smoking() {

System.out.println("有的人会吸烟");

}

}

# 面向对象特点之：多态Polymorphism

/\*\*

\* 多态是指事物的多种形态

\* 1.多态的体现

\* 父类的引用指向子类对象。

\* 2.多态的前提

\* 类之间是继承或实现关系，但是必须子类重写了父类的方法。

\* 3.多态的好处

\* 提高了代码拓展性。找到了对象共性，指挥所有的对象去做相同事情。

\* 4.多态的应用

\* 多态的类型转换：

\* 1.向上类型转换，自动转换 Animal animal = new Dog();

\* 2.向下类型转换，强制转换 Dog dog = (Dog)animal //注意，前提看注意

\* 注意： 1.不要将父类对象转换成子类类型。因为父类中不存在子类特有的功能。

\* 2.多态自始至终都是子类的对象在做变化。

\*

\* 在多态情况下，函数和变量的调用方式。

\* 成员方法：(非静态方法)

\* 编译时：要查看引用变量所属的类中是否有所调用的成员。

\* 在运行时：要查看对象所属的类中是否有所调用的成员，因为方法运行参照的

\* 是对象，所有会调用对象的方法。

\* 成员变量：(实际开发中不常见

\* 只看引用变量所属的类。即只看左边

\* 在多态情况下的通过对象调用静态方法(不常见)

\* 静态方法静态绑定。

\* 成员方法是动态绑定，和对象进行绑定，只能通过对象调用。

\*/

//父类引用获取子类成员变量。

public class Demo{

public static void main(String[] args) {

A a = new B();

System.out.println(a.getNum());

}

}

class A{

private int num = 1;

public int getNum() {

return num;

}

}

class B extends A{

private int num = 3;

public int getNum() {

return num;

}

}

接口，和多态增加程序拓展性的很好的例子。

/\*\*

\* 接口和多态最佳实践

\* 需求：给主板添加拓展功能。(不是焊死在里面，而是提供一个接口。)

\* 要求：降低主板和添加新设备的耦合性。

\*/

public class Demo {

public static void main(String[] args) {

MainBroad mBroad = new MainBroad();

mBroad.addPCI(new PCI() { //通过匿名内部类添加。最好写一个类实现接口

public void open() {

System.err.println("添加的设备打开");

}

public void close() {

System.err.println("添加的设备打开");

}

});

}

}

//通过接口定义想主板中的添加规则

interface PCI{

public void open();

public void close();

}

class MainBroad{

public void run(){

System.out.println("主板在运行");

}

//添加拓展的方法。通过回调open()和close()方法。控制打开和关闭

public void addPCI(PCI pci){

pci.open();

pci.close();

}

}

//一个数据库的小例子，没有具体实现。降低主程序和数据库的耦合性，通过多态。

public class Demo4 {

public static void main(String[] args) {

//在主程序中操作数据库，当数据库连接

DataBaseInterface my\_db = new My\_Database();

my\_db.add();

}

}

//定义数据库连接接口，让添加的数据库实现这个借口。提高拓展性。

interface DataBaseInterface{

public abstract void add();

public abstract void query();

}

class My\_Database implements DataBaseInterface{

public void add() {}

public void query() {}

}

# Object类的常用方法。

我们定义的类自动继承了Object类，也就继承了Object中的所有public方法。

1. toString();返回对象的字符串表示。

我们一般从写这个方法。让其返回对象的属性。

**public** **class** Demo {

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

Test test = **new** Test();

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

}

}

**class** Test{

String name = "koterwong";

//重写toString方法，让其返回对象的属性。

**public** String toString() {

**return** "test" + name ;

}

}

1. equals(Object obj) 方法。

返回对象是否与参数中的Object对象相等。

当我们要比较两个对象的属性是否相同时，就需要从写该方法

重写equals有两种方法，第一是比较同一类的连个不同对象的属性是否相等，可以用eclipse自动生成。如下：

**public class Demo4 {**

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

**Demo demo = new Demo("koterwong");**

**Demo demo2= new Demo("koterwong");**

System.out.println(demo==demo2); //不重写比较的哈希值是否相等

System.out.println(demo.equals(demo2));

**}**

**}**

**class** Demo2{

String name = "";

public Demo(String name){

this.name = name;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Demo other = (Demo) obj;

**if** (name == **null**) {

**if** (other.name != **null**)

**return** **false**;

} **else** **if** (!name.equals(other.name))

**return** **false**;

**return** **true**;

}

}

第二种方法比较类的不同对象的属性是否相等，这种需要比较少见。

public class Demo4 {

public static void main(String[] args) {

Test test = new Test();

Demo demo = new Demo();

System.out.println(test.equals(demo)); //输出结果为true

}

}

class Test{

String name = "koterwong";

@Override

public boolean equals(Object obj) {

if (obj instanceof Demo) {

return this.name == ((Demo)obj).name;

}else {

return false;

}

}

}

class Demo{

String name = "koterwong";

}

1. hashCode() 返回该对象的哈希码值。

在集合框架中会有涉及。

public class Demo4 {

public static void main(String[] args) {

Demo demo = new Demo();

System.out.println(demo); //直接输出引用为类名加哈希值

System.out.println(Integer.toHexString(demo.hashCode()));

} //将哈希值转化成十六进制。

}

class Demo{

String name = "koterwong";

}

1. getClass() 返回此 Object 的运行时类。返回类型是[Class](mk:@MSITStore:C:\Users\Koterwong\Desktop\JDK_API_1_6_zh_CN.CHM::/java/lang/Class.html)<?>

Class 对象是在加载类时由 Java 虚拟机构造的！用于调用静态成员。也可以理解成加载类的 类名.class 文件

public class Demo {

public static void main(String[] args) {

Demo demo = new Demo();

System.out.println(demo.getClass().getName());;

System.out.println(Demo.class.getName());

System.out.println(Demo.class==demo.getClass());

}

}

输出结果为：

com.koterwong.javademo.Demo

com.koterwong.javademo.Demo

true

# Java中的内部类。

当描述事物时，事物的内部还有事物，该事物用内部类来描述。

(身体和心脏)

class Body{

private class XinZang{ //不需要对外暴露，封装内部类

}

public void show(){ //提供对外访问的规则

new XinZang().

}

}

访问方式：

内部类可以直接访问外部类中的成员，包括私有成员。

外部类要访问内部类中的成员必须要建立内部类的对象。

成员内部类：(看成类的成员变量)

可以被private，static成员修饰符修饰。

创建内部类对象

Outer.Inner inner = new Outer().new Inner();

创建静态内部类对象

Outer.Inner inner = new Outer.Inner();

被static修饰的内部类只能访问外部类中的静态成员。

注意：

当内部类中定义了静态成员，该内部类必须是static的。因为静态成员需要优先加载到内存，而内部类也必须加载到内存。

当外部类中的静态方法访问内部类时，内部类也必须是static的。

方法内部类：

1. 不可以被成员修饰符修饰
2. 可以直接访问外部类中的成员，因为持有外部类中的引用。
3. 只能访问方法中被final修饰的局部变量。

匿名内部类

定义匿名的内部类，必须继承一个类或实现接口。

# 包(package)

*  给类提供多层命名空间。 (类名相同时，通过包名进行区分)
*  写在程序文件的第一行。 **package** com.koterwong.javademo;
*  类名的全称的是 包名.类名。 com.koterwong.javademo.Demo;
*  包也是一种封装形式。

public 相当于没有权限。

protected 子类中可以直接调用不同包中父类protected方法或成员变量。

default 不能用于修饰类只能访问同一包中的内容。

private 不能修饰类，被private修饰只能本类使用，可以提供get,se

import关键字

* 简化类名的书写。创建对象的时候，不用写类的全名。
* 用来导包中的类，不导入包中的子包。通常写import mypack.Demo; 而不写import mypack.\*;

Jar包

 Java的压缩包

方便项目的携带。

方便于使用，只要在classpath设置jar路径即可。

数据库驱动， SSH框架等都是以jar包体现的。

Jar包的操作

通过jar.exe工具对jar的操作。

创建jar包 :jar -cvf mypack.jar packa packb

查看jar包 :jar -tvf mypack.jar [>定向文件]

解压缩 :jar -xvf mypack.jar

自定义jar包的清单文件 :jar –cvfm mypack.jar mf.txt packa packb