1. @ 方法；是什么：完成特定功能的代码块（函数在java中叫方法）

怎么做：

方法:完成特定功能的代码块，不调用不执行

注意:在很多语言里面有函数的定义，而在Java中函数被称为方法。

方法格式:

修饰符 返回值类型 方法名(参数类型参数名1,参数类型参数名2...) {

方法体语句;

return返回值;

}

输出调用：有明确返回值的调用：输出调用（不推荐）

赋值调用：

Void类型：只有单独调用pinjie（a，b）

详细解释i

修饰符:其前就用public static 后面我们更详細的讲解其他的修饰符

返回值类型:就是功能结果的数据类型

方法名符合命名规则即可方便我们的调用。

参数:

实际参数:就是实际参与运算的

形式参数就是方法定义上的，用于接收实际参数的

参数类型就是参数的数据类型

参数名就是变量名

方法体语包:就是完成功能的代码

@ return：结束方法的

返回值:就是功能的结果，由return带给调用者

Java中关键字continue、break和return的区别：

continue：跳出本次循环继续下一次循环

break： 跳出循环体，继续执行循环外的函数体（跳出单层循环）

跳出多层循环：

格式：标签名：语句

return: 跳出整个函数体，函数体后面的部分不再执行

return关键字不是为了跳转出循环体，更常用的功能是结束一个方法，也就是退出一个方法，跳转到上层调用的方法

Java中关键字continue、break和return的区别：

continue：跳出本次循环继续下一次循环

break： 跳出循环体，继续执行循环外的函数体（跳出单层循环）

跳出多层循环：

格式：标签名：语句

return: 跳出整个函数体，函数体后面的部分不再执行

return关键字不是为了跳转出循环体，更常用的功能是结束一个方法，也就是退出一个方法，跳转到上层调用的方法

@

for(初始化语句：判断条件语句；控制条件语句){

循环体语句}

do while循环至少执行一次循环题

而for， while循环必须先判断条件是否成立，然后决定是否执行循环体语句

/\*if语句和switch语句的区别：

if语句：a：针对结果是布尔类型的判断

b;针对一个范围的判断

c：针对几个常量值的判断

switch语句：针对几个常量值的判断

\*/

/\*

switch语句和if语句的区别有：if语句，表达式的结果是boolean类型常用于区间判断。

break 中断 continue 继续 return 返回

switch：表达式类型不能是boolean类型，可byte，int，char，String，枚举。

常用于等值判断选择语句的选择。能用switch语句实现的就一定可以使用if实现，但是反之不一定。如果是区间范围就采用if，如果是等值判断使用switch。

@ 方法重载：方法重载概述：在同一个类中，方法名相同，参数列表不同，与返回值类型无关，参数列表不同：参数个数和参数类型不同

在同-一个类中，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。

方法重载特点：

与返回值类型无关，只看方法名和参数列表

在调用时，虚拟机通过参数列表的不同来区分同名方法

1. @数组

●数组初始化概述:

Java中的数组必须先初始化然后才能使用。

所谓初始化:就是为数组中的数组元素分配内存空间

并为每个数组心素赋值。

●数组的初始化方式

动态初始化:初始化时只指定数组长度，由系统为数

组分配初始值。Int[] a=new int[3]

静态初始化:初始化时指定每个数组元素的初始值，

由系统决定数组长度。Int arr={1,2,3}

数组操作常见的两个小问题

●数组索引越界

Array IndexOutofBounds Exception

访问到了数组中的不存在的索引时发生。

●空指针异常

NullPointer Exception

数组引用没有指向实体，却在操作实体中的元素

●格式1（动态初始化）

数据类型[][] 变量名= new数据类型[m][n];

●m表示这个二维数组有多少个- -维数组

●n表示每-一个一维数组的元素个数

举例:

int] arr = new int[3][2];

定义了一一个二维数组arr

●这个二维数组有3个- -维数组， 名称是arr[0] ar[1]arr[2]

●每个一维数组有2个元素，可以通过arr[m][n]来获取

●格式3（动态初始化）

数据类型[][]变量名= new数据类型[][<{元素..{元

素.元素..}};

简化版格式:

数据类型叨变量名= {元素..}.{元素..6..}.

举例:

●int[][] arr= {{1,2,3}{4,6},{6}};

java中的参数传递;

基本类型：形式参数的改变对实际参数没有影响

引用类型：形式参数的改变直接影响实际参数（它们拿的是同一个内存地址）

@ 我们就考虑使用类封装来这多个万法，将来再做数组的操作时，不用去找具体的方法，先找到这个类，然后使用这个类中的方法。这就是面向对象思想的编程方式

面向对象是基于面向过程的编程思想：

面向过程：强调的是每一个过程的步骤

面向对象：强调的是对象，然后由对象去调用功能（更关注于结果）

（有哪些类、每个类有哪些东西呢、类与类之间的关系）

（封装、继承、多态）

举例：洗衣服:

面向过程:把衣服脱下--找个盆放点洗衣粉--加点水--把衣服扔进去--搓-一搓--清洗衣服--扭于--晾起来

面向对象:把衣服脱下--打开全自动洗衣机--扔进去--一键打开即可--晾起来

@ 面向对象

我们怎么才能更符合面向对象思想呢?

A:有哪些类呢?

B:每个类有哪些东西呢?

C:类与类之间的关系是什么呢?

把太象装进冰箱的分析?(如何分析有哪些类呢?名词提取法。)

A:有哪些类呢?

太象

冰箱

Demo

B:每个类有哪些东西呢?

太象:

进去

冰箱

开门

关门

Demo :

main方法

1. 类与类之间的关系是什么呢》

Demo中使用大象和冰箱类的功能。

面向对象开发：就是不断的创建对象使用对象，指挥对象做事情

面向对象设计：其实就是在管理积维护对象之间的关系

面向对象特征：

封装(encapsulation)

继承(inheritance)

多态(polymorphism)

1. @类与对象的关系

类： 变量（属性） 方法（行为）

是一组相关属性和行为的集合，是抽象的概念

对象：是该类事物的具体表现形式，具体存在的个体

事物： 类：

属性（该事物的描述信息） 成员变量

行为 （事物的功能） 成员方法

对象的创建： 类名 对象名=new 类名（）；

Student s=new Student（）；

变量（variable） ：成员变量：在堆内存--》有默认值

局部变量：在栈内存--》必须定义，赋值，才可调用

生命周期不同：成员边量：随着对象的创建而存在，随着对象消失而消失

局部变量：随着方法调用而存在，随着方法调用完毕而消失

注意:局部变量名称可以和成员变量名称一样，在方法中使用的时候采用的是就近原则

如果看到了一个方法的形式参数是一个类类型（引用类型）这里其实需要的是一个该类的对象

@匿名对象（没有名字的对象）：new Student（）；

调用方法只调用一次的时候，调用多次不合适

new Student（）.show（）；//这里其实是重新创建了新的对象

好处：调用完毕就被回收

匿名对象可作为实际参数传递

//StudentDemo sd=new StudentDemo()；

sd.method（new Student（））;--》new StudentDemo().method(new Student());

@封装：隐藏对象的属性和实现细节，仅对外提供公共访问方式（笔记本）

定义--个学生类：

成员变量name，age

（私有修饰成员边量，提供公共方式访问）

成员方法show()方法

我们在使用这个案例的过程中发现了一个问题:

通过对象去给成员变量赋值可以赋值一些非法的数据

这是不合理的

应该是这个样子的在赋值之前.先对数据进行判断。判断到底在哪里做比较合适呢?

studentDemo类是一个测试类，测试类一般只创建对象，调用方法。

所以，这个判断应该定义在Student类中。

而我们在成员变量的位置可不可以进行数据判断呢?

是不可以的因为做数据校验必须要依靠一些逻辑语旬

逻辑语句是应该定义在方法中的，所以，我们最终决定在student类中提供一个方法

来对数据进行校验。

按照我们前面的分析我们给出了个方法进行校验

但是呢。它偏偏不调用方法来赋值还是直接赋值了

这样我们的方法就没有起到作用

我就应该要求你必须使用我的方法而不能直接调用成员变量赋值

怎么去强制要求不能直接使用成员变量呢?

@ 针对这种情况，Java就提供了一个关键字private

private:私有的：可以修饰成员变量和成员方法。

注意：被private修饰的成员只能在本类中使用

Private常见应用：

把成员变量用private修饰

提供对应的setxxx（）或getxxx（）方法

一个标准案例（学生类）的使用

@this（代表当前类的对象引用）它就代表当前类的一个对象

注意：谁调用这个方法，在该方法内部的this就代表谁

（this的使用场景：解决局部变量隐藏成员变量）

this：哪个对象调用那个方法，this就代表那个对象

name = name; //变量的使用规则就近原则

//这里是类名，目前还没有说过类似的用法

//这里的调用只能通过对象名

//这个对象如果存在，它应该代表的是Student的一一个对象。

. //那么谁能够代表当前类的对象呢? java就提供了个关键字this

this.name = name ;

@构造方法

●构造方法作用概述

给对象的数据进行初始化

●构造方法格式

方法名与类名相同

没有返回值类型，连void都没有

没有具体的返回值

●构造方法注意事项

如果你不提供构造方法，系统会给出默认构造方法

如果你提供了构造方法，系统将不再提供

构造方法也是可以重载的

给成员变量赋值有两种方法：

Setxxx（）和 构造方法

类的组成：成员变量：

构造方法：

成员方法：

方法具体划分: .

根据返回值

有明确返回值方法

返回void类型的方法

根据形式参数

无参方法

带参方法

@方法：

根据返回值类型：void类型和非void类型

根据形式参数：有参和无参方法

Student s = new Student () ;//创建对象

s. show() ;//调用无参无返回值方法

/ /调用无参有返回值方法

String result = s.getstring() ;

System. out . println (result) ;

s. method ("林青霞") ;//调用带参无返回值的方法

//调用帶参带返回值的方法

String result2 = s . function ("hello", "world") ;

System. out. println (result2) ;

注意：变量定义为成员变量--》变量是用来描述这个类的信息的

如果这个变量是用来描述这个类的信息的，那么这个变量就该定义为成员变量

@static

定义一个人类

姓名和年龄都是变化的这个我能接收因为每个人的姓名和年龄是不同的a

但是我们现在选取的几个人都是中国人他们的国籍一样的。

一样的国籍，我每次创建对象。在推内存都要开辟这样的空间

我就觉得有点浪费了怎么办呢?

针对多个对象有共同的这样的成员变量值的时候.java就提供啦关键字static

Static针对多个对象有共同的这样的成员变量值，static修饰的数据是被所有对象共享的（一改全改）

static的特点(它可以修饰成员变量还可以修饰成员方法)

A:随着类的加载而加载

回想main方法

B:优先于对象存在

c:被类的所有对象共享

举例咱们班级的学生应该共用同一个班級编号，其实这个特点也是在告诉我们什么时候使用静态?

如果某个成员变量 是被所有对象共享的。那么它就应该定义为静态的。

举例

饮水机(用静态修饰)

水杯(不能用静态條饰)

D:可以通过类名调用

其实它本身也可以通过对象名调用。

非静态的可称其为对象成员

静态修饰的内容我们一般称其为：与类相关的类成员

static关键字注意事项

A:在静态方法中是没有this关键字的，如何理解呢?

静态是随着类的加载而加载，this是随着对象的创建而存在，静态比对象先存在

B:静态方法只能访问静态的成员变量和静态的成员方法，

静态方法:

成员变量:..只能访问静态变量

成员方法:只能访问静态成员方法

非静态方法:

成员变量:可以是静态的a也可以是非静态的

成员方法可是是静态的成员方法也可以是非静态的成员方法。

简单记:静态只能访问静态

@ 静态变量和成员变量的区别：

所属不同

静态变量属于类，所以也称为为类变量

成员变量属于对象，所以也称为实例变量(对象变量)

内存中位置不同

静态变量存储于方法区的静态区

成员变量存储于堆内存

内存出现时间不同

静态变量随着类的加载而加载，随着类的消失而消失

成员变量随着对象的创建而存在，随着对象的消失而消失

调用不同

静态变量可以通过类名调用，也可以通过对象调用

成员变量只能通过对象名调用

@main方法的格式讲解

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

public:公共的。访问权限是最太的。由于main方法是被调用所以权限要够太

static:静态的。不需要创建对象通过类名就可以，方便jvm的调用

void:因为我们曾经说过.方法的返回值是返回给调用者，而main方法是被jvm调用。你返回内容给jvm没有意义。

main:一个常见的方法入我见过的语言都是以main作为入口

String[] args：这是个字符患数组。值去哪里了?这个东西到底有什么用啊?怎么给值啊?

这个东西早期是为了接收键盘录入的数据的。

格式是: java MainDemo hello world java

把构造方法私有，外界就不能再创建对象啦

Math类没有构造方法，因为它的成员都是静态

@ 代码块

在Java中，使用{}括起来的代码被称为代码块，根据

其位置和声明的不同，可以分为局部代码块，构造代

码块，静态代码块，同步代码块(后面讲解)。

●局部代码块（局部位置，用于限定变量的生命周期）

在方法中出现;限定变量生命周期，及早释放，提高内存利用率

●构造代码块

（在类中的成员位置，用{}括起来的代码，每次调用构造方法执行前，都会先执行构造代码块）

（作用：可以把多个构造方法中的共同方法放到一起，对对象进行初始化）

在类中方法外出现;多个构造方法方法中相同的代码存放到

一起， （每次调用构造都执行），并且在构造方法前执行

●静态代码块在类中方法外出现，加了static修饰

（在类中的成员位置，用{}括起来的代码，只不过它用static修饰啦）

（作用：一般是对类进行初始化）

在类中方法外出现，并加上static修饰: 用于给类进行初始化

在加载的时候就执行，（并且值执行-次）。（作用：一般是对类进行初始化）

执行顺序：静态代码块--》构造代码快--》构造方法

静态代码块：只执行一次

构造代码块;每次调用构造方法执行

1. @ 继承

@ 我们观察上面两个代码:

发现name，age成员变量，以及getXxx () /setXxx (),还有eat ()等都是相同的。

如果我们后来继续定义类。举例工人类，军人类。他们是不是也具备这些内容

那么我们每一次定义这样的类的时候都要把这些重复的内容都重新定义一遍。

麻烦不?麻烦所以我们要考虑改进?如何改进呢?

我这想的我能不能把这些相同的内容给定义到一个独立的类虫

然后，让这多个类和这个独立的类产生一个关系有了这个关系后is a 关系

这多个类就可以具备这个独立的类的功能。

为了实现这个效果，java就提供了一个技术 继承。

继承怎么表示呢?继承的格式是什么梯子的呢?

class Fu { }

class Zi extends Fu {

}

好处:

A:提高了代码的复用性

B:提高了代码的維护性

c:让类与类之间产生了关系是多态的前提

类与类产生了关系其实也是继承的一个鱉端:

类的耦合性增强了

开发的原则低耦合，高内聚

耦合：类与类的关系

内聚:就是自己完成基件事情的能力

继承的注意事项：

A:子类只能继承父类所有非私有的成员(成员方法和成员变量)

B:子类不能继承父类的构造方法但是可以通过super关键字去访问父类构造方法

C:不要为了部分功能而去继承

@ 那么我们什么时候考虑使用继承呢?

继承其实体现的是一种关系"is a"

Person：（is a）

Student

Teacher

采用假设法：

如果有两个类A,B。只有他们符合A是B的-种，或者B是A的一种，就可以考虑使用继承。

@ 问题是:（this和super）

我不仅仅要输出局部范围的num还要输出本类成员范围的num怎众办呢?

我还想要输出父类成员范围的num怎么办呢?

如果有一个东西和this相似但是可以直接访问父类的数据就好了

恭喜你这个关键字是存在的：super

this和super的区别?

分别是什众呢?

this代表本类对应的引用。

super代表父类存储空间的标识(可以理解为父类引用，可以操作父类的成员)

怎众用呢?

A:调用成员变量

this.成员变量调用本类的成员变量

super.成员变量调用父类的成员变量

B:调用构造方法

this(...) 调用本类的构造方法

super(. . .) 调用父类的构造方法

C:调用成员方法

this.成员方法调用本类的成员方法

super.成员方法调用父类的成员方法

@ 继承中构造方法的关系

A:子类中所有的构造方法默认都会访问父类中空参数的构造方法

B:为什么呢?

因为子类会继承父类中的数据，可能还会使用父类的数据

所以子类初始化之前.-定要先完成父类数据的初始化

注意:子类每一个构造方法的第一条语句默认都是super();

@ 如果父类没有无参构造方法那么子类的构造方法会出现什么现象呢? 报错.

如何解决呢?

A:在父类中加一个无参构造方法

B:通过使用super关键字去显示的调用父类的带参构造方法

c:子类通过this去调用本类的其他构造方法

子类中一定要有一个去访问了父类的构造方法否则父类数据就没有初始化

注意事项，

this(...)或者super(...)必须出现在第一条语旬上

如果不是放在第条语包上就可能对父类的数据进行了多次初始化所以必须放在第一条语句上

C:子父类的初始化(分层初始保)

先进行父类初始化然后进行子类初始化em.

问题:

虽然子类中构造方法默认有-个super ()

初始化的时候不是按照那个顺序进行的

而是按照分层初始化进行的

它仅仅表示要先初始化父类数据，再初始化子类数据。

@ 子类中的方法和父类中的方法声明一样，这个该怎么玩呢? .

通过子类调用方法：

a:先找子类中看有没有这个方法有就使用

b:再看父类中有没有这个方法有就使用

c:如果没有就报错。

@ ●方法重写概述

子类中出现了和父类中一模一样的方法声明，也被称为方法覆盖，方法复写。

使用特点:

●如果方法名不同，就调用对应的方法

●如果方法名相同，最终使用的是子类自己的

●方法重写的应用:

当子类需要父类的功能，而功能主体子类有自己特有

内容时，可以重写父类中的方法，这样，即沿袭了父

类的功能，又定义了子类特有的内容。

方法重写的注意事项：

A:父类中私有方法不能被重写，因为父类私有方法子类根本就无法继承

B:子类重写父类方法时访问权限不能更低，最好就-致

c:父类静态方法子类也必须通过静态方法进行重写

其实这个算不上方法重写，但是现象确实如此至于为什么算不上方法重写多态中我会进解

子类重写父类方法的时候最好声明模一样。

@ 1:方法重写和方法重载的区别?方法重载能改变返回值类型吗?

方法重写:（Override）

在子类中出现和父类中模一样的方法声明的现象

方法重载:（Overload）

同一个类中出现的方法名相同，参数烈表不同的现象

方法重载能改变返回值类型，因为它和返回值类型无关。

@ 2: this关键字和super关键字分别代表什么?以及他们各自的使用场景和作用？

this:代表当前类的对象引用

super:代表父类存储空间的标识

(可以理解为父类的引用通过这个东西可以访间父类的成员)

场景:

成员变量:

this.成员变量

super.成员变量

构造方法:

this(...)

super(. . .)

成员方法:

this.成员方法

super.成员方法

@@final

由于继承中方法有个现象：方法重写，所以父类的功能，就会被子类给覆盖调

有些时候，我们不想让子类去覆盖掉父类的功能，只能让他使用

这个时候，针对这种情况，Java就提供 了一个关键字: final

final:最終的意思常见的是它可以修饰类，方法。变量

特点:

fina1可以修饰类，该类不能被继承。

final可以絛饰方法.该方法不能被重写. (覆盖，复写)

final可以修饰变量，该变量不能被重新赋值因为这个变量其实常量

常量:

A:字面值常量 "hello", 10, true

B:自定义常量 final int X = 10;

@面试题:. final絛饰局部变量的问题

基本类型：基本类型的值不能发生改变。

引用类型：引用类型的地址值不能发生改变，但是该对象的堆内存的值是可以改变的

final修饰变量的初始化时机

A:被final絛饰的变量只能赋值一次。

B:在构造方法完毕前(非静态的常量)

5，多态 @

●多态概述

某一个事物，在不同时刻表现出来的不同状态。

举例:

猫可以是猫的类型。猫m= new猫();

同时猫也是动物的一-种，也可以把猫称为动物。

动物d= new猫（）;

在举一个例子:水（液态，气态，固态）在不同时刻的状态

多态的前提:

A:要有继承关系

B:要有方法重写

C:要有父类引用指向子类对象。

父 f=new 子();

多态中的成员访问特点:

A:成员变量：编译看左边运行看左边

B:构造方法：创建子类对象的时候访问父类的构造方法。对父类的数据进行初始化 C:成员方法：编译看左边运行看右边

D:静态方法：编译看左边运行看左边

(静态和类相关，算不上重写所以，访问还是左边的)

由干成员方法存在方法重写所以它运行看右边

多态的好处：A:提高了代码的维护性（继承保证）

B:提高了代码的扩展性（有多态保证）

多态的鱉端:

不能使用子类的特有功能

我就想使用子类的特有功能?行不行?

行，怎么用呢?

A:创建子类对象调用方法即可(可以但是很多时候不合理。而且太占内存了)

B:把父类的引用，强制转换为子类的引用(向下转型)

对象间的转型问题:

向上转型: Fu f= new Zi();

向下转型:Zi z=(Zi)f;//要求该f必须是能够转换为Zi的。

ClassCastException：类型转换异常：一般在多态的向下转型中容易出现

6, @抽象类

@只有看到了具体的动物，你才知道，这是什么动物。所以说， 动物本身并不是一个具体的事物，

而是一个抽象的事物。只有真正的猫，狗才是具体的动物。

同理，我们也可以推想，不同的动物吃的东西应该是不一样的，所以，我们不应该在动物类中给出

具体体现，而是应该给出一个声明即可。在Java中一个没有方法体的方法应该定义为抽象方法，而类中

如果有抽象方法，该类必须定义为抽象类。

@抽象类的概述:

动物不应该定义为具体的东西而且动物中的吃睡等也不应该是具体的,

我们把一个不是具体的功能称为抽象的功能，而一个类中如果有抽象的功能，该类必须是抽象类，

@抽象类的特点：

A:抽象类和抽象方法必须用abstract关键字修饰

B:抽象类中不一定有抽象方法,但是有抽象方法的类必须定义为抽象类

C:抽象类不能实例化

因为它不是具体的。

抽象类有构造方法但是不能实例化?构造方法的作用是什么呢?

用千子类访问父类数据的初始化

？（可通过具体的子类进行实例化）多态主要用于抽象类

//Animal是抽象的;无法实例化//Animal a = new Animal () ;

//通过多态的方式Animal a=new Cat() ;a.eat() ;

抽象类的实例化其实是靠具体的子类实现的，是多态的方式。

D:抽象的子类

a:如果不想重写抽象方法，该子类是个抽象类。

b:重写所有的抽象方法，这个时候子类是一个具体的类

@抽象类的成员特点：

成员变量：既可以是变量也可以是常量

构造方法:有，用于子类访问父类 数据的初始化

成员方法：既可以是抽象的也可以是非抽象的。

抽象类的成员方法特性：

A:抽象方法强制要求子类做的事情

B:非抽象方法子类继承的事情提高代码复用性

@分析；从具体到抽象

实现：从抽象到具体

@一一个类如果没有抽象方法。可不可以定义为抽象类?如果可以，有什么意义

A:可以

B:不让创建对象

abstract不能和哪些关键字共在

private冲突（私有不能被继承，抽象要求被重写）

final冲突

static无意义

1. @接口

狗看门，猫一般就是宠物--》现在有很多驯养员可以训练出，猫钻火圈，狗跳高

--》这些额外的动作，并不是所有的猫或者狗一开始就具备的，是经过特殊的培训出来的，

所以将这些额外的功能定义到动物类中不合适，动物类中定义的是共性的东西，是一开始就有的，而这是额外的东西，按照继承的思想，把这定义在父亲里是有问题的，

它也不适合直接定义在猫或狗中，因为不是所有的猫都这样，不是所有的狗都这样，因为只有部分猫狗才具有这些功能，

--》如何把这些额外的东西定义出来？

--》为了体现事物的扩展性，java提供了接口来定义这些额外功能，并不给出具体实现（抽象方法），将来哪些猫、狗需要被培训，只需要这部分猫狗把这些额外功能实现即可

（接口的功能--》额外扩展的功能）

接口的特点：

A:接口用关键字interface表示

interface 接口名{}

B:类实现接口用implements表示

class类名implements接口名{}

C:接口不能实例化

那么接口如何实例化呢?

按照多态的方式来实例化

D:接口的子类

a:可以是抽象类。但是意义不太

b:可以是具体类，要重写接口中的所有抽象方法。(推荐方案)

由此可见:

A:具体类多态(几乎没有)

B:抽象类多态(常用)

C:接只多态(最常用)

接口名+Impl 这种格式是接口的实现类格式

//接口方法不能带有主体？：

//接口最大的作用是作为一种协议，实现接口的一方提供功能，而使用接口的一方只要去调用接口中定义的方法就行了，

//至于具体的实现则由实现方去实现。起到了声明和实现分离，如果没有接口，所有都是直接的类与类之间的访问与调用，

//则会导致耦合性过高，不利于扩展。因为接口是用来标明方法调用的。在程序里调用接口，只调用方法就可以，不用关心方法是如何实现的

//，将来实现接口的类如何调整都不会影响到调用者。如果上来直接就调用具体方法，

//将来这个具体方法换掉的时候你不是就木了吗，也许就要改好几个类了。

//用术语说叫提高了耦合性（耦合性就是可以理解为粘连性，依赖性，因为往往一个项目的代码编程量是很大的，

//所以这个就要注意每段程序之间的连接不要太过依赖，那样就不方便后期的维护）

//接口是为一批类定义的一个标准,各自有各自的实现方法

//举例

//动物接口,动物都会叫

//但是,狗是狗叫,猫是猫叫...

//所以接口只能定义它会叫（声明）,不能具体到怎么叫（实现）

接只成员特点：

成员变量:只能是常量并且是静态的

默认絛饰符public static final

建议:自己手动给出

构造方法:接口没有构造方法，因为接口主要是扩展功能，而没有具体存在

成员方法：只能是抽象方法。

默认修饰符public abstract

建议自己手动给出

所有的类都默认继承自一个类: Object.

类Object是类层次结构的根类。每个类都使用Object 作为超类

类与类，

继承关系, 只能单继承，可以多层继承

类与接口

实现关系,可以单实现,也可以多实现

并且还可以在继承个类的同时实现多个接口

接口与接口

继承关系,可以单继承，也可以多继承

@抽象类和接口的区别:

A:成员区别

抽象类:成员变量可以变量，也可以常量

构造方法：有

成员方法:可以抽象也可以非抽象

接口：成员变量.只可以常量

成员方法:只可以抽象

B:关系区别

类与类:继承单继承

类与接口：可以实现单实现也可以多实现

接口与接口：继承.单继承多继承

C:设计理念区别.

1，抽象类被继承体现的是，"is a”的关系， 抽象类中定义的是该继承体系的共性功能。

不同的动物吃的东西应该是不一样的，所以，我们不应该在动物类中给出具体体现，而是应该给出一个声明即可。

只有看到了具体的动物，你才知道，这是什么动物。所以说， 动物本身并不是一个具体的事物，而是一个抽象的事物。只有真正的猫，狗才是具体的动物。

2，接口被实现体现的是: "like a”的关系， 接口中定义的是该继承体系的扩展功能。（它本身并不具备，随着。。。发展，它里面添加了新东西）（USB接口）

猫能跳高，一般来说，跳高这个动作，运动员可以，猫就像一个运动员了，但它不是一个运动员