❤技巧：

Ctrl+/ 加注释

Alt+/ 根据当前的名称来找到有可能的类或类方法

再不确定运算符优先级的情况下，使用括号进行优先级排序运算。

**任何一个类在使用时都需要先创建类对象！！！！！**

**类名 类对象名 = new 类名( );**

❤ 输出：

System.out.print(“.....”)/System.out.printIn(“.....”)

print为一个类方法/ printIn同样也为一个类方法

根据输出的参数类型的不同，print/printIn类中提供了许多重载函数，以此来面对输出不同类型的值的要求

Tip：二者区别 print输出不会自动换行，printIn会自动换行

❤输入： Tip: 参数要求不清楚，底层不清楚

借助Scanner类建立一个新的类对象，在输入时Scanner类的析构函数要求参数为System.in

Scanner A=new Scanner(System.in);

A为一个类对象，因此可以使用Scanner的类方法

1. nextLine(字符串类型)/nextInt(数值)/......

不同的类方法代表A所需要存储的数据类型，A的值为相对应的取到的值。

此外，由于是next，所以其本质是取一个对应类型的数据，光标顺势移动到已经取到的数据后面。假设输入 1 2，使用两次nextInt，则第一次A的值为1，第二次A的值为2.

❤字符串输出连接：

使用+进行字符串之间的连接

Tip：System.***out***.print(2+3+"=2+3="+2+3+"\n");//结果为 5=2+3=23

System.***out***.print(2+3+"=2+3="+(2+3));//结果为 5=2+3=5

（？规则未知）由于优先级计算规则，因此会出现上述的不同，通常在输出中如果要进行计算，那么计算需要使用括号将其括起来表示一次运算。

❤变量：

定义与C语言相同： int a=10;/int a; //定义一个变量a

变量可以放在输出中使用

例如：System.***out***.print("a="+a);或System.***out***.print(10+a);

❤常量：

final关键字

final int amount=100;//定义amount为一个常量，其值为100

❤浮点数：

Java中的整数除法 / 的含义与C相同，均为计算后去掉余数

整数除法指除法双方均为int型变量

例如 10/3; 结果为3

当除法式的某一边为浮点数时

例如: 10/3.0 此时3.0为浮点数 因此结果也为浮点数

乘法同理

Double A=7; 此时A的值转变为7.0

Tip：当一个变量的类型为double型时，A=in.nextInt()同样可用，系统会读取一个int，并转化为对应的浮点数。同理，A=in.nextDouble()会读取一个int/double类型并转换（系统自动类型转换）

但是，如果A类型为int的话，是无法读取一个double类型值并转换。

❤算术运算符：



优先级越高，在没有括号的情况下就会被越早的执行

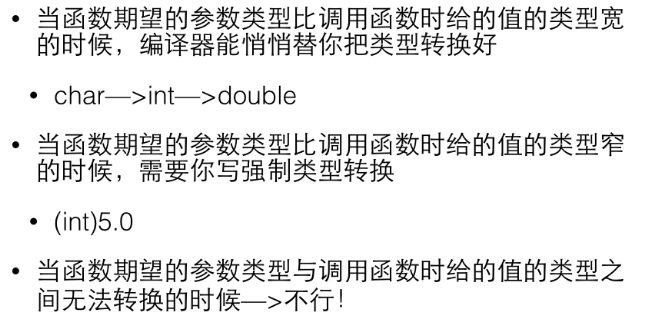
括号中的式子优先级是最高的

❤强制类型转换：

int a;

double b=10.2;

a=(int) b; //此时a的结果为10

a=(int)(30/3.0); //对右边运算式的结果进行强制类型转换

❤关系运算：（大小判断）



关系运算的返回值为true/false，例如：int a=100;

System.out.print(a>=90); //打印true

Tip：所有关系运算符的优先级均高于算术运算符（赋值除外），同级的关系运算符从左向右进行计算。此外，关系运算符在没有括号的情况下，相等与不相等这两个的优先级是低于其他的关系运算符的。例如：

10>4==5>3;// 结果为true

==优先级低，因此先计算10>4与5>3，结果均为true，因此结果为true

❤if语句：格式与C相同 无需分号结尾

If(条件语句){ //if后无需分号

.......//当执行语句仅有一行时，可以去掉大括号

} //不需要分号结束

与c相同，同样具有else，else if语句

Tip：在没有大括号进行层次分级的时候，else总会去与离他最近的if去进行一个匹配，例如：

If(条件1)

If(条件2) ......

else ..... //此时标红的两个会进行匹配

If(条件1){

If(条件2) ...... }

else ..... //此时标红的两个会进行匹配

因此建议只要出现if，else，else if的语句，就要为其执行语句加上一个大括号，即使其仅仅只有一条语句。

❤switch语句： 无需分号结尾

switch(表达式){ //表达式例如 int型常量 a

case 表达式值1: 语句1; break; //若表达式值可以对应上，则执行相应语句

case 表达式值2: 语句2; break;

case 表达式值3: 语句3; break;

...

case 表达式值N: 语句N; break;

default: 语句N+1; break; //上述所有的表达式值均对不上，则执行默认语句

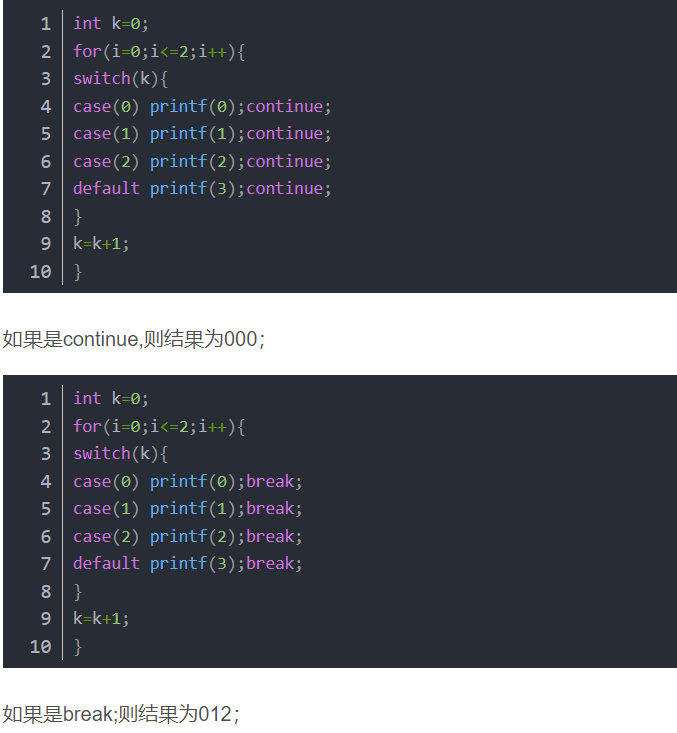
}

break的作用是在于当执行了一次case/default之后，便会直接终止此次的分支语句。如果没有break，可能会执行多次case中的执行语句。

Tip：1. 在switch语句中，“case 常量表达式”只相当于一个语句标号， 表达式的值和某标号相等则转向该标号执行，但不能在执行完该标号的语句后自动跳出整个switch 语句，所以会出现继续执行所有后面case语句的情况。

2. switch中continue生效是对于外部while/for循环，指忽略本次单次循环剩下的代码，从而执行下一次循环。

1. switch中break生效是对于当前所在switch分支语句。



❤while/for循环：格式与c相同

while(循环条件){

.......

} //无需分号

do{

.....

} while (循环条件); //此处需要分号

for(...;....;....;){

......

} //无需分号

❤随机数：

Math.random(); Math类中的random函数

可以随机生成一个属于[0,1)的浮点数

Math.random()\*100 可以获得一个[0,100)的浮点数

(int)(Math.random()\*100+1) 可以获得一个[1,100]的正整数

❤i++，++i含义与C相同

❤单层循环控制/循环跳出:

在C中可以使用return/break/continue来结束当前函数/结束当前一层循环/结束当次循环

在Java中有return type(返回值void时无type)，使用break/continue

❤多重循环控制： 注意continue tag：

在C中可以使用多层break/goto(不建议)语句

goto语句:需要一个标签 格式 标签名加 : 例如 A:

goto A: //表示转向标签A的地方并执行接下来的代码

同理，Java可以使用多层break/标签机制

Java标签机制格式与C相同 tag: 表示一个名叫tag的标签

与C不同的是，tag：后所接的下一行代码必须为一个循环体代码

C可以接任意语句

当在多层循环中使用break tag：时，系统会得知要跳出的是标签后所执行的循环，从而跳出多层循环。

当在多层循环中使用continue tag：时，系统会认为标签后所执行的循环体重的单次循环已经结束，需要进行下一次的循环。

参考代码：①break tag：

public class Jump {

public static void main(String[] args) {

outer:

for(int i = 0; i < 3; i++) {

System.out.print("loop "+i);

System.out.print();

for(int j = 0; j < 10; j++)

{

System.out.print("looper "+j);

if(j == 5)

break outer;

}

System.out.print("Jump");

}

System.out.print("跳出循环");

}

}

/\*Output： loop 0 looper 0 looper 1 looper 2 looper 3 looper 4 looper 5 跳出循环\*/

break tag：直接跳出标签后的循环体

②continue tag：//修改上述代码break outer变为continue outer

结果为loop 0 looper 0 looper 1 looper 2 looper 3 looper 4 looper 5 loop 1 looper 0 looper 1 looper 2 looper 3 looper 4 looper 5 loop 2 looper 0 looper 1 looper 2 looper 3 looper 4 looper 5 跳出循环

在执行到j==5时，执行continue outer；从而转向最外层for循环，但是由于continue跳出的是本次循环，如果i<3的话就会进行i++并继续执行。

❤逻辑运算：boolean类型

与c相同

运算有 &&（与运算） ||（或运算） ！（非运算）

非运算优先级高于其他两种运算

❤数组：

C/C++: type name[ (number) ]; //c可不输入number

Java: type [ ] name = new type [ number ];

/type [ ] name ={..........各元素值..................};

type为数组元素类型 name为数组名 number为数组大小

number可以为一个变量

name.length 为一个成员变量，表明当前数组的数组总长度

Java数组在创建时会自动进行初始化，int型会初始化全为0

Tip：int [ ] a={1,2,3,4,5};

Int [ ] b=a;

此时a，b两数组指向的为同一片内存区域，可以理解为数组名与c相同，都代表着内存地址，此时对b数组所做的操作，等价于对a数组所做的操作。创建两个相同元素的数组，应该分别创建然后进行一次遍历并赋值。

❤for-each循环①：仅适用于数组遍历

for( 变量类型 变量名 ：变量对应数组/容器 ){

..........

}

不需要规定次数，每一次从数组中按照遍历顺序取出数，并放入之前的变量中。

for(int k : num1){

.....

}

遍历访问num1数组，并且将每次遍历的值赋予局部变量k。

❤二维数组：

Type [ ] [ ] name = new type [number] [number];

❤逃逸字符/转义字符：在单双引号中使用



❤包裹类型：



对应的包裹类型可以看做是一个类，例如 Integer.MaxValue 调用成员变量，返回int的最大值，包裹类型中含有一些成员变量和一些类方法可以使用。

❤字符串：底层同样为一个字符数组

String s = new String (“.....”);

String s1 = “......”;

两种创建并初始化的方法，字符串之间的拼接使用 +

Tip:当某一边为字符串而另一边并非字符串时，会进行类型转换，转化为字符串并进行拼接，因此当print(“s11”+12+24)时，会打印出s111224

因此如果需要提前计算，则需要使用括号提升优先级

❤in.next()用于获取下一个字符，in.nextLine()用于获取一整行的字符串。

❤字符串类的类函数：

①假设有两个字符串s1,s2，由于底层本质为字符数组，因此

s1==s2; 比较的是两者是否指向同一片内存区域，即s1,s2本质为一个字符数组的存储地址。

如果要比较两个字符串的内容是否相等，s1.equals(s2)，即字符串类具有一个类方法为equals()，用于比较两个字符串的内容是否相等。

②s1,s2之间的大小比较：

s1.compareTo(s2); 返回值一个正负值或者为0

③ s1.length 返回值为当前字符串的大小/字符数量

④s1.charAt(index); 返回对应字符串数组下标的字符

⑤s1.substring(index1); 从下标为index1处开始截取并返回

s1.substring(index1,index2); 从下标为index1处开始截取，截取到index2之前并返回 （不包括index2下标所在的字符）

⑥s1.indexOf( 字符/字符串 ) 找到对应字符/字符串在当前字符串中的位置，并返回首个下标，返回-1表示不存在

s1.indexOf( 字符 , index ) 从下标index处开始寻找对应字符

如果需要从右向左进行查找,使用 lastindexOf

⑦s1.toLowerCase() 小写

s1.toUpperCase() 大写

此时的大小写并没有对原有的字符串进行修改，可以看做依据要求创建一个新的字符串。

⑧前后缀判断：

s1.startsWith(....) s1.endsWith(...)

⑨字符串中某字符整体替换/字符串修改:

s1.replace(char oldChar,char newChar)

使用newchar替换原字符串的所有oldchar

⑩字符串去空格：

s1.trim() 去掉开头和结尾的空格

1. 字符串分解：String [] test = s.split([separator，[limit]])

创建一个复制体赋值字符串s，separator为分隔符，limit为可选表示返回的字符数量。将字符串s根据分隔符separator进行拆分，拆分后放入一个字符串数组test中。

❤函数定义：

(public static) int Sum(int head,int tail) {

.....

return 0;

} //public static为修饰符 可根据情况选择

格式与C基本相同，多了函数头（public static）当返回值为void时，可以使用单独一个return来进行函数模块的终结。

❤类：

Java本质是一个个类项目，一个大的项目中包含着许许多多的类，当一个项目开头有 public class 时，证明其是一个公开的类。与C++不同的是，java类中必定会包含一个名称为main的类函数。

除了main函数外，可以在main函数之外，当前类之间的地方定义当前类的成员变量与类函数。

ClassName Name = new ClassName();//创建类对象语句

❤this指针：表示当前类对象中的成员变量与类函数

与C++相同，仅在类函数中可以使用，例如某个类中有成员变量price，有类函数为changeprice，Func：

void changeprice(int price){

this.price=price;

this.Func();/Func(); //类函数调用可加可不加

}

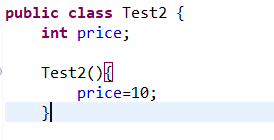
由于函数参数与成员变量名字相同，因此需要使用this指针去表明是当前类对象中的成员变量price，从而与函数参数的含义分开。

同理，调用类函数时也可以加上一个this。

❤析构函数/构造函数：函数创建时自动调用的函数

在类对象被创建的时候，会自动进行一次构造函数的执行。

例如：



类名为Test2 因此析构函数为Test2()

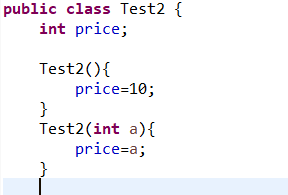
析构函数可以传参数，如果带有参数则在创建类对象的时候需要进行参数传递。

析构函数的主要目的是对于成员变量的初始化。

❤函数重载：

在同一个作用域中，包含多个同名但是参数不同（参数类型，参数顺序，参数个数）的函数，称这几个函数为重载函数。

在使用重载函数的时候，系统会根据参数的不同来判断出需要去使用的函数。



此时两个析构函数为重载函数，系统会根据创建时是否带有参数来判断使用哪一个析构函数来进行类对象的创建。

❤Java操作注意事项与静态static：

Java中，操作需要在一个函数体内执行。

在函数体外定义的static变量为全局变量，在函数体内定义的static变量为静态局部变量。

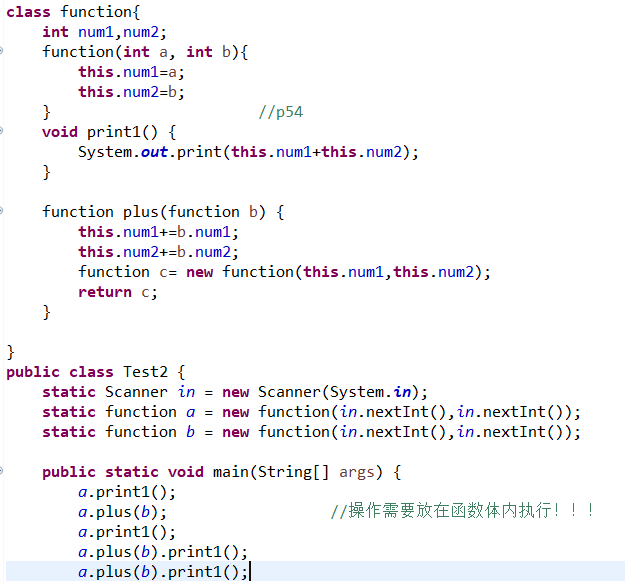
static的特点是生存期延长，并且只会被初始化一次。当函数再次启动时，静态局部变量的值不会被改变，维持上次函数执行完毕后的值。

❤Java全局变量：

Java全局变量与C不同，C可以直接在main函数之外进行定义，即所有在main函数之外定义的均为全局变量。

而java除了需要在main函数之外进行定义，还需要加上static关键词。

❤Java返回类对象：



①function plus作为一个类函数，其返回的为一个类对象，因此

1. plus(b).print1() 可以看做是 c.print1()

②类函数plus中所返回的对象c，其本质可以看做是一个地址的传递，

a.plus(b).print1() 看做在通过a.plus返回一个对象地址，并通过这个对象地址来进行print1()类函数的操作。

❤类定义的区域划分：



在一个类定义中，函数体main之外的为类的类函数或者成员变量，而函数体之内的为操作。

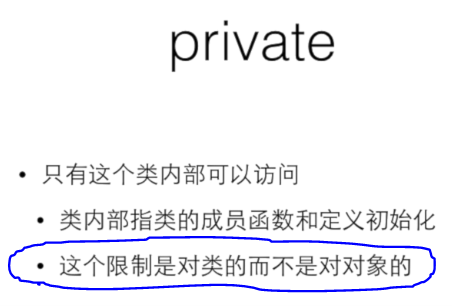
！！！注意如果以一个类对象作为成员变量的话，需要去建立一个相应的类对象才可以。

Tip：%02d的含义为以0作为填充，固定占位为2格。

❤私有变量 private

C++与Java原理相同（语法有些不同），被定义为私有变量的成员变量仅可以在当前类中进行访问，比如成员变量的初始化或者类函数中进行使用。

如果某个类函数的参数为当前类的类对象，并且在当前类函数中访问了作为参数的类对象的私有变量，仍然可以访问。



比如有个function类，含有一个私有变量a，通过构造函数为a初始化，

有一个类函数为： int f(function r){

return r.a;

}

此时如果有两个类对象，并且某个类对象执行类函数f，另一个类对象作为参数传递进去，此时可以访问另一个类对象的私有变量。

总结：在类函数中，那么可以访问其所在类的类对象的私有变量。

Tip：main函数也算是当前类的一个类函数，因此可以在main函数中去访问类对象的私有变量。

❤public，编译单元，包，friendly：

编译单元即一个.java文件/一个类文件，一个包中可以存放多个编译单元，一个文件可以存放多个包，一个编译单元中可以有多个类，但是只能有一个public类。在一个编译单元中如果需要使用到其他包里面的类，那么需要使用import进行类的导入。

import 包名.类名; 此时的.看做...中的意思

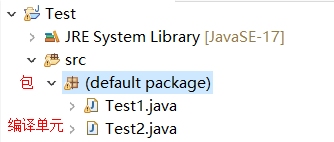
对于public修饰的变量或函数或类来说，可以被当前文件下的所有包中的编译单元使用。

对于一个没有前缀修饰符的成员变量或函数或类来说，默认为是friendly，此时可以被当前包中的其他编译单元所使用（无需import）。

修饰类仅可以使用public与friendly

private 仅可以在当前类、类对象中去访问私有变量（两个同类的类对象可以访问对方的private）

protected 对于所有的子类（可以不在同一包）以及包内其他编译单元可见。



❤成员变量与类变量：

成员变量是指每个成员对象所拥有的可以相同可以不相同的变量，而类变量通常是指类定义中所拥有的的变量，其在每个类对象身上的值均相同。

类变量严格来说是属于类的而不是属于类对象的，但是类对象可以去访问类变量与类函数。

❤static关键字：

static关键字表示一个静态变量，在java中用于设置一个类变量。

class Box(){

static int i=0;

......

}

此时Box类中含有一个类变量i，初始值为0.

静态变量存储在一片特殊区域中，且一个静态变量所在的位置是不会被改变的，静态变量的生存周期不随着函数结束而结束，也就意味着静态变量的值不会因为函数的再次使用而被再次初始化，静态变量在一个程序的运行中只会初始化一次。

类变量在初始化后，存储在一个特殊区域，类对象可以去访问，修改这个区域，每个类对象所访问的类变量都是同一片区域。

既可以通过 Box.i 去访问修改这个类变量，也可以通过 类对象.i 去访问修改这个类变量。

此外，对于静态函数来说，静态函数与当前类并没有关系，因此无法通过当前类去调用私有变量。静态函数更像是常规的代码，放在任何地方都可以去使用。Main函数就是一个静态函数，虽然main函数包含在一个类当中，但是无法直接去访问类中的私有变量，相比较之下类函数就可以直接访问某个类对象的私有变量。

❤容器类①：ArrayList容器类：是一个类，因此使用new新建类对象

ArrayList类似于一个可以自动延长的数组

ArrayList< type > name = new ArrayList< type > ( );

type为容器中的数据类型，name为容器名，ArrayList为其中一种容器名。

ArrayList容器与数组类似，按照顺序存储，下标从0开始，可以按照下标进行操作。

ArrayList类函数：

ArrayList< type > A = new ArrayList< type > ( );

①数据存入：

1. add(....); 按照顺序存入
2. add(....,index); 存入对应index下标，部分数据往后推

②数据量查询：

1. size(); 返回当前数据量值

③按下标寻找数据：

1. get(index); 按照下标index返回对应数据

④删除下标元素：

1. remove(index); 删除对应下标index数据，并返回删除数据

⑤将ArrayList容器中的数据转到数组中

type [ ] num = new type [ A.size( ) ] ;

1. toArray( num ) ;

按照容器中数据顺序复制一份并放入到对应的数组num中。

⑥整体输出容器内容：

System.out.print(A);



按顺序输出在一个方括号中（自带toString函数）

❤String数组：

数组元素类型为string，由于string本质为一个字符数组，因此可以看

做string数组中保存着多个字符数组，s[index]表示一个字符串，表示一个字符数组，同样也可以看做表示一个字符数组的起始地址。

创建时，系统会自动初始化每一个元素为null。

❤常规数据类型与数组、字符串、类对象、容器类/java传值：

常规数据类型（除string外），在函数中使用的时候，会产生一个复制体，复制变量的值，函数中所使用的为复制体，因此在函数中对其进行修改并不会影响原有的变量。

对于数组，string，类对象，容器来说，同样在函数中进行操作时也是产生一个复制体，但是复制体中所保存的可以看做是对应类型的地址，因此在函数中是通过地址去操作其本身。！！！

❤for-each循环②：针对字符串、类对象、容器类（与数组类似）

for-each循环同样作为一个函数，当遍历的数组/容器元素为字符串

(string)，类对象时，传进去的值其实是为对应的地址，此时可以在for-each循环中通过这个地址去访问对应的字符串与类对象本身。

❤容器：集合类：

HashSet< type > name = new HashSet< type > ( );

集合类与ArrayList类容器的最大区别在于ArrayList为有序排放，不会检测是否有重复值，与数组类似；而集合类无序存放，有重复值仅会保留一个值，与数组不同。

1. add(......) 集合放入数据
2. size( ) 返回集合数据量值

❤Hash表： key-value保存，类似数组

HashMap< key-type,value-type > name = new HashMap < key-type,value-type > ( );

其中key-type与value-type的数据类型如果为基础类型（boolean，char，int，double），则需要变为包裹类型。 原因未知

哈希表最大的特点是key-value存储，一个key对应一个value，Java中的哈希表不考虑碰撞机制，在发生冲突时，会清除掉旧数据，并将新数据放入其位置中。

哈希表在创建时自动进行初始化，类似于string数组，未定义的位置上的值均为null。

1. put(key,value); 将value放入相应的key位置
2. get(key); 根据对应的key获取其位置上的value
3. containsKey(key1); 检查key1位置上是否有value，返回true/false
4. keySet( ); 哈希表集合化/以集合的方式来表示当前的哈希表的key值

A.keySet( ).size( ); keySet( )返回集合类对象，size()为集合数据量

Tip：Java中null值等价于false，与c相同

❤！！！继承：

class 子类name extends 父类name

子类继承父类所有的非private变量与函数。

子类与父类的同名变量，本质是两个不同的变量。！！！

子类对象中如果没有某个变量或函数，那么会去其父类对象中进行寻找。

子类对象中如果有和父类重名的变量，那么会使用子类的变量。（即使子类变量没有被初始化）

Java就近原则使如果父类与子类出现同名变量，那么会根据所引用的变量前面的类来决定使用子类或父类中的变量。

例如： animal类为父类，dog类为子类，均有age变量

animal test1;

dog test2;

此时test1.age引用的为父类的age变量，test2.age引用的为子类的age变量。

Tip：①this指针的改变（就近原则）：this指针永远指向当前所在类的类对象，对于父类来说，永远指向父类对象，对于子类来说，永远指向子类对象。this指针所在类函数所在的类决定了this的指向。

②子类创建过程：子类创建类对象的同时，会创建一个父类对象。如果父类对象中的析构函数具有参数，那么此时需要在子类析构函数中增加一个super(...)函数，super函数可以看做调用父类的析构函数，其参数与父类析构函数的参数相同。

③访问父类变量的方法：第一种是将其变为protected型，第二种是在父类中建立一个访问私有变量的类函数，子类调用父类相应的类函数去访问对应的私有变量。

④访问父类同名函数的方法：super.函数名( ) 调用父类函数。

⑤！！！子类可以看做是父类，但是父类不可以看做子类。

例如：animal和dog类，dog可以看做是animal，但是animal不可以看做dog。

假设此时有一个ArrayList容器

ArrayList< animal > P = new ArrayList< animal >( );

此时既可以放入animal类对象，也可以放入dog类对象。

for( animal k , P )

此时的k既可以是父类animal，也可以是子类dog。

public void function(animal f);

此时的参数既可以是animal类，也可以是dog类。

❤！！！Java与C++在对象赋值上的不同点：

在C++中，可以进行一个对象对另一个对象的赋值，比如此时有两个在相同类中的类对象A与B，在C++中进行A=B，等价于把B类对象中的数据复制一份并赋予A类对象。

但是在Java中，无法进行对象与对象之间的赋值，原因是Java中全部都是传值操作，对于类对象，容器来说，当进行A=B时，传递的为一个地址，也就是把类对象B的地址传给了A，此时A对于这个类对象的操作就是直接操作其本身。

Java与C++在数组方面机制不同，数组名二者都代表着一个地址，假设为A，B，C++无法实现A=B的操作，但是java可以实现A=B的地址赋值操作，从而让A,B指向同一个数组。

Java与C++字符串的机制相同，字符串名表示一个地址，假设为A，B，两者都可以实现A=B，从而让A，B指向同一个位置。并且二者在字符串赋值的原理也相同，都是在某个地方建立一个新的字符串，返回一个地址给字符串A。 A=B，A=“.....”，此时A地址改变。

❤造型与类型：

类型转换： int a = (int)(10.2);

类型转换是把从一个类型转变到另一个类型，数据类型发生改变

造型是属于类的，animal类（父类）与dog类（子类），造型指的是编译器该如何去看待一个类，比如dog类，既可以看做dog也可以看做animal，因此可以在参数要求为animal类的地方放入一个dog类。但是反之，animal类并不一定可以看做是一个dog类，编译器会报错。

向上造型指的就是在需要父类的地方，可以放入一个子类（子类可以看做是父类）。但是在需要子类的地方，放入一个父类就可能会报错。

造型之间可以进行转换，既可以向上造型，也可以向下造型。仅仅是改变编译器对于类的看法，而并不会对类的内容做出任何改变。

❤override（重写）与overload（重载）：

override为重写父类中的函数，子类存在一个和父类完全相同的函数（限制符，函数名，函数参数），但是所执行的动作不同。可以在子类重写父类函数的时候在函数前加上@override，仅是个重写标志。

重载是指在同一个作用域中，存在着多个函数名相同但是参数不同的函数，当系统需要调用函数时，会根据调用时所传入的参数决定去使用哪一个函数。

❤多态：子类父类相同函数不同动作

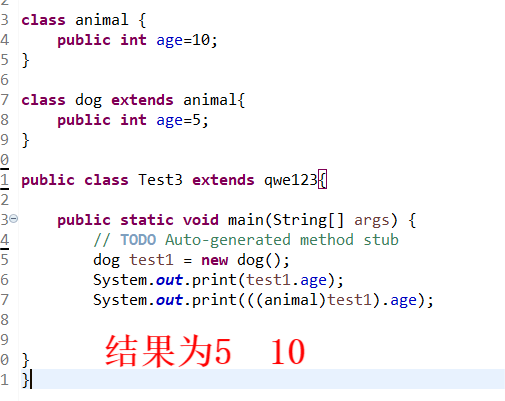
animal父类与dog子类假设均有一个bark类函数，且参数相同。

由于向上造型的原因，在一个需要animal类对象参数的函数中可以去放入一个dog类对象，此时系统会将其看做是dog类。在函数中如果使用bark函数，那么系统会根据当前传进来的类对象是animal还是dog，来决定到底去使用哪一个类中的bark函数。

❤！继承，多态总结：

子类继承父类，此时子类可以使用父类中的非私有变量与类函数（子类中使用父类的类函数需要 super.父类函数名（） ）

如果出现相同的变量名，系统会根据其造型来决定它所使用的到底是父类还是子类中的变量或者函数。例如：



第一个print中test1造型为dog，因此使用的是dog中的变量，第二次print中造型转变为了animal，因此使用的是animal中的变量。

函数同理，当子类父类中含有一个名称参数相同的函数时，系统会根据

所传进来的类对象的造型，来决定此时使用的函数到底是父类的还是子类的。

！！！向上造型决定了在需要一个父类对象的地方可以放入一个子类，而多态决定了系统会根据放入的造型（即子类或父类）来决定去使用哪一个类中的变量或者函数。

❤Object的类函数：返回值函数

Object是所有非子类的默认父类

public String toString ( ){

return “....”;

}

为一个特殊的类函数，表示为类对象或子类对象赋予一个字符串类型的返回值。

❤ StringBuffer类：

Java中字符串String在赋予一个新的字符串值时，会在某个地方开辟一块新的区域并赋值，返回String对象一个新的地址。

StringBuffer类也是创建字符串，但是它在赋予一个新值的时候，不会重新开辟一块新区域，而是会在原本的区域进行修改，StringBuffer有自动扩展区域大小的功能。

StringBuffer s= new StringBuffer(“... ...”)；

StringBuffer类方法不会重新开辟区域，仅修改原有区域。

StringBuffer类方法：

a.append("b"); //字符串a拼接"b"

a.insert(1,"b"); //字符串本质为字符数组 在a下标为1处插入"b" 原本字符后移

a.deleteCharAt(1); //字符串a删除下标1处的字符 原本字符前移

a.delete(index,index1); //字符串a删除 [index,index1) 处的字符

a.replace(1,9,"13908862319"); //从下标为1处开始替换 9为结束位置，如果替换字符串过长，自动延长

a.reverse(); //字符串反转

a.substring(int start[,int end] ); //截取字符串a，从下标start开始，如果有end到end结束,没有截取到队尾

a.length( ); //返回当前字符串的字符数

❤封装、耦合：

耦合指的是类与类之间的关系，通常指类使用另一个类的成员变量的多少，使用的成员变量越多耦合越大。

封装是一种降低类与类之间耦合度的一种方法，封装要求的是在类中将所有的变量定义为私有变量，如果外部有需要使用私有变量的情况，那么在类中定义类函数（接口），通过接口去传递给外部私有变量的值，或者外部通过接口去修改私有变量的值。

封装可以看做是将一系列的操作做成一个接口，当外部使用这个接口的时候，不需要知道接口内部是如何处理的，只需要去使用即可。

❤向上造型的作用：代码简化

向上造型指的是把一个子类类对象看做一个父类类对象，例如

animal test1 = new dog( );

其含义为创建一个新的dog类类对象，但是把它当做是一个animal类对象来看待。（详细用法见Hash表高阶）

好处在于，此时test1的本质上仍然为dog类，当使用test1中的类函数时，仍然是优先去子类中寻找，子类如果没有再去父类中寻找。

Tip：向上造型虽然使用的方法是从子类开始寻找，但是如果试着直接去取得test1的public变量，那么变量仅会在父类中寻找，而不会去子类中寻找相关的变量值。

❤Hash表高阶用法：实现操作不同对象

假设此时父类为animal 子类为dog，cat

HashMap< String, animal > List = new HashMap < String, animal > ( );

此时有了一个名为List的哈希表，key为String类型，value为animal类。List哈希表放入（“mao”，new cat( ) ）,（“gou”，new dog( ) ）。

Scanner in = new Scanner( System.in) ;

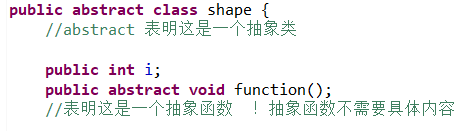
String s1=in.nextLine( );

animal test1 = List.get（s1）; //此处体现了向上造型思想

假设dog与cat均有eat（）函数，由于向上造型的关系，此处并不需要因为子类的不同，而去选择创建对应的子类对象。

向上造型方便了多态的实现。

❤抽象：对子类的规范/要求



当一个类中含有至少一个抽象函数的时候，就表明这是一个抽象类。抽象类可以看做是一个规范类，因为其无法产生类对象，因此其只可以作为父类使用。当子类继承了一个抽象类的时候，抽象类中非抽象函数仍可以看做正常父类中的函数使用，抽象函数必须进行重写。

抽象类无法产生类对象，但是可以产生一个抽象变量，例如图中shape为一个抽象类，其有一个子类为Circle类。

shape s = new Circle( );

此时可以看做是向上造型。

❤