Java学习我想好好学，时间不多了。

Java变量类型

Java局部变量

局部变量声明在方法，构造方法或者语句块中；

局部变量在方法、构造方法、或者语句块被执行的时候创建，当他们执行完成后，变量将会被销毁；

访问修饰符不能用于局部变量；

局部变量只在声明它的方法、构造方法或者语句块中可见；

局部变量是在栈上分配的。（此处的栈很疑惑）

局部变量没有默认值，所以局部变量被声明后，必须经过初始化，才可以使用。此处初始化我理解为赋值

在以下实例中age是一个局部变量。定义在pupAge（）方法中，它的作用域就限制在这个方法中。

Public class Test{

Public void pupAge(){

Int age = 0;

age = age + 7;

System.out.println(“Puppy age is : ” + age);

}

Public static void main(String args[]){

Test test = new Test();

Test.pupAge();

}

}

实例变量

* 实例变量声明在一个类中，但在方法、构造方法和语句块之外；
* 当一个对象被实例化，每个实例变量的值就跟着确定；
* 实例变量在对象创建的时候创建，在对象被销毁的时候销毁；
* 实例变量的值应该至少被一个方法、构造方法或者语句块引用，使得外部能够通过这些方式获取实例变量信息；
* 实例变量可以声明在使用前或者使用后；
* 访问修饰符可以修饰实例变量；
* 实例变量对于类中的方法、构造方法或者语句块是可见的。一般情况下应该把实例变量设为私有。通过使用访问修饰符可以使实例变量对子类可见；
* 实例变量具有默认值。数值型变量的默认值是0，布尔型变量的默认值是false，引用类型变量的默认值是null。变量的值可以再声明时指定，也可以在构造方法中指定；
* 实例变量可以通过变量名访问。但在静态方法以及其他类中买就应该使用完全限定名：

ObjectReference.VariableName。

实例：

Import java.io.\*;

public class Employee{

//这个成员变量对子类可见

public String name;

//私有变量，仅在该类可见

Private double salary;

//在构造器中对name赋值

//构造函数的作用，得好好想想

public Employee(String empName){

name = empName;

}

//设定salary的值

public void setSalary(double empSal){

salary = empSal;

}

//打印信息

public void printEmp(){

System.out.println(“name : ” + name );

System.out.println(“salary : ” + salary);

}

Public static void main(String args){

Employee empOne = new Employee(“Ransika”);

empONe.setSalary(1000);

empOne.printEmp();

}

}

类变量（静态变量）

* 类变量也称为静态变量，在类中以static关键字声明，但必须在方法、构造方法和语句块之外。
* 无论一个类创建了多少个对象，类只拥有类变量的一份拷贝。
* 静态变量除了被声明为常量外很少使用。常量是指声明为public/private,final和static类型的变量。常量初始化后不可改变。
* 静态变量存储在静态存储区，经常被声明为常量，很少单独使用static声明变量。
* 静态变量在程序开始时创建，在程序结束时销毁。
* 与实例变量具有相似的可见性，但为了对类的使用者可见，大多数静态变量声明为public类型。
* 默认值和实例变量相似数值型变量默认值为0，布尔型默认值是false，引用类型默认值是null。变量的值可以在声明的时候指定，也可以在构造方法中指定。此外，静态变量还可以在静态语句快中初始化。
* 静态变量可以通过：ClassName.variableName的方式访问。
* 类变量被声明为public static final类型时，类变量名称必须使用大写字母。如果静态变量不是public和final类型，其命名方式与实例变量以及局部变量的命名方式一致。

实例：

Import java.io.\*;

Public class Employee{

//salary是静态的私有变量

private static double salary;

//DEPARTMENT是一个常量

public static final String DEPARTMENT = “Development”;

public static void main(){

salary = 1000;

System.out.println(DEPArtMENT + “average salary : “ + salary);

}

}

**注意：**如果其他类想要访问该变量，可以这样访问：Employee.DEPARTMENT。

本章节中我们学习了Java的变量类型，下一章节中我们将介绍Java修饰符的使用。

Java语言支持的变量类型有：

局部变量、类变量、成员变量

Java修饰符

访问修饰符

非访问修饰符

修饰符用来定义类，方法或者变量，通常放在语句的最前端。

Public class className{

//…..

}

Private boolean myFlag;

Static final double weeks = 9.5;

Protected static final int BOXWIDTH = 42;

Public static void main(String[] arguments){

//方法体

}

Java中，可以使用访问控制符来保护对类、变量、方法和构造方法的访问。Java支持4种不同的访问权限。

默认的，也称为default，在同一个包内可见，不使用任何修饰符。缺省）

私有的，以private修饰符指定，在同一类内可见。私有化，封装相关知识

共有的，以public修饰符指定，对所有类可见。公共的）

受保护的，以protectd修饰符指定，对同一包内的类和所有子类可见。

以监管严到松排名 private ，缺省，protected，public

默认访问修饰符-不使用任何关键字

使用默认访问修饰符声明的变量和方法，对同一包内的类是可见的。接口里的变量都隐式声明为public static final,而接口里的方法默认情况下访问权限为public。

实例：

如下例所示，变量和方法的声明可以不使用任何修饰符。

String version = “1.5.1”;

boolean processOrder(){

return true;

}

私有访问修饰符—private

私有访问能修饰符是最严格的访问级别，所以被声明为private的方法、变量和构造方法只能被所属类访问，并且类和接口不能声明为private。

声明为私有访问类型的变量只能通过类中公共的getter方法被外部类访问。

Private访问修饰符的使用主要用来隐藏类的实现细节和保护类的数据。

下面的类使用了私有访问修饰符：

Public class Logger{

private String format;

public String getFormat(){

return this.format;

}

public void setFormat(String format){

this.format = format;

}

}

实例中，logger类中的format变量为私有变量，所以其他类不能直接的到和设置该变量的值。为了是其他类能够操作该变量，定义了两个公共public 方法：getFormat()（返回format的值）和setFormat(String)（设置format的值）

公有访问修饰符—public

被声明为public 的类、方法、构造方法和接口能够被任何其他类访问。

如果几个相互访问的public类分布在不同的包中，则需要导入public类所在的包。由于类的继承性，类所有的公有方法和变量都能被其子类继承。

以下函数使用了公有访问控制：

Public static void main(String[] arguments){

//…

}

Java程序的main()必须设置成公有的，否则，java解释器将不能运行该类。

受保护的访问修饰符—protected

被声明为protected的变量、方法、和构造器能被同一个包中的任何其他类访问，也能够被不同包中的子类访问。

Protected访问修饰符不能修饰类和接口，方法和成员变量能够声明为protected，但是接口的成员变量和成员方法不能为protected。

子类能访问Protected修饰符声明的方法和变量，这样就能保护不相关的类使用这些方法和变量。下面的父类使用，子类重载了父类的openSpeaker()方法。

class AudioPlayer{

protected Boolean openSpeaker(Speaker sp){

//实现细节

}

}

class StreamingAudioPlayer{

Boolean openSpeaker(Speaker sp){

//实现细节

}

}

如果把openSpeaker()方法声明为private，那么除了 如果把openSpeaker()声明public，那么所有的类都能够访问该方法。如果我们只想让该方法对其所在类的子类可见，则将该方法声明为protected。

访问控制和继承

请注意一下方法继承的规则：

* 父类中声明为public的方法在子类中也必须为public。
* 父类中声明为protected的方法在子类中要么声明为protected，要么声明为private。
* 父类中默认修饰符声明的方法，能够在子类中声明为private。
* 父类中声明为private的方法不能被继承。继承使用封装中的参数，构造函数赋值给参数赋值赋予的值是父类中变量的任意值包含初始值）-

非访问修饰符

为了实现一些其他的功能，java也提供了许多非访问修饰符。

Static修饰符，用来创建类方法和类变量。

Final修饰符，用来修饰类，方法和变量，final修饰的类不能够被继承，修饰的方法不能按继承类重新定义，修饰的变量为常量，是不可以修改的。如果在编辑器中输入3.14159.。。。。“派”不使用final修饰会是什么效果）

Abstract修饰符，用来创建抽象类和抽象方法。

Synchronized和volatile修饰符，主要用于线程的编程。总结一下访问修饰符哦）

Static修饰符

* 静态变量：

Static关键字用来声明独立于对象的静态变量，无论一个类实例化多少多少对象，他的静态变量只有一份拷贝。

静态变量也被称为类变量。局部变量不能被声明为static变量。

* 静态方法：

Static关键字用来声明独立于对象的静态方法。静态方法不能使用类的非静态变量。静态方法从参数列表得到数据然后计算这些数据。

对类变量和方法的访问可以直接使用classname.variablename和classname.methodname的方式访问。

如下例所示，static修饰符用来创建类方法和类变量。

Public class instancecounter{

private static int numInstances = 0 ;

protected static int getCount(){

return numinstances;

}

private static void addInstance(){

numInstances ++ ;

}

InstanceCounter(){

InstanceCounter.addInstance();

}

public static void main(String[] arguments){

System.out.println(“String with” + InstanceCounter.getCount() + “instance”);

for(int I = 0; I < 500 ; ++i){

new Instancecounter();

}

System.out.println(“Created” + instanceCounter.getCount() + “instances”);

}

}

Final 修饰符

Final变量：

Final变量能被显式地初始化并且只能初始化一次。被声明为final的对象的引用不能指向不同的对象。但是final对象的数据可以被改变。也就是说final对象的引用不能改变，但里面的值是可以改变的。

Final修饰符通常和static修饰符一起使用来创建类常量。

实例：

public class Test{

final int value = 10;

//下面是声明常量的实例

public static final int BOXWIDTH = 6;

static final String TITLE = “Manager”;

public void changeValue(){

value = 12;//将输出一个错误

}

}

Final 方法

类中的final方法可以被子类继承，但不能被子类修改。

声明final方法的主要目的是防止该方法的内容被修改

如下所示，使用final修饰符声明方法。

public class Test{

public final void changeName(){

}

}

Final类

Final类不能被继承，没有类能够继承final类的任何特性。

实例：

public final class Test{

//类体

}

Abstact修饰符

抽象类：

抽象类不能被用来实例化对象，声明抽象类的唯一目的是为了将来对该类进行扩充。

一个类不能同时被abstract和final修饰，如果一个类中包含抽象方法，那么该类一定要声明为抽象类，否则将出翔一个编译错误。

抽象类可以包含抽象方法和非抽象方法。

实例：

abstract class Caravan{

private double price;

private String mode1;

private String year;

public abstract void goFast();//抽象方法

public abstract void changeColor();

}

抽象类中不一定有抽象方法，但含有抽象方法的类一定是抽象类。

抽象方法

抽象方法是一种没有任何实现的方法，该方法的具体实现有子类提供。抽象方法不能被声明成final和static。任何继承抽象类的子类必须实现父类的所有抽象方法，除非该子类也是抽象类。如果一个类包含若干个抽象方法，那么该类必须声明为抽象类。抽象类可以不包含抽象方法。

抽象方法的声明以分号结尾，例如：public abstract sample();

实例：

public abstract class SuperClass{

abstract void m();//抽象方法

}

class SubClass extend SuperClass{

//实现抽象方法

void m(){

……………

}

}

Synchronized修饰符

Synchronized关键字声明的方法同一时间只能被一个线程访问。Synchronized修饰符可以应用于四个访问修饰符。

实例：

public synchronized void showDetails(){

……………….

}

Transient修饰符

序列化的对象包含被transient吸收的实例变量时，java虚拟机跳过该特定的变量。

该修饰符包含在定义变量的语句中，用来预处理类和变量的数据类型，

实例：

public transient int limit = 55; //will not persist

public int b; //will persist

volatitle修饰符

volatile修饰的成员变量在每次被线程访问时，都强迫从共享内存中重读该成员变量的值。而且，当成员变量发生变化时，强迫线程将变化值回写到共享内存。这样在任何时刻，两个不同的线程总是看到某个成员变量的同一个值。一个volatile对象引用可能是null。

实例：

public class MyRunnable implements Runnable{

private volatile Boolean active;

public void run(){

active = true;

while(active) {//line 1

//代码

}

}

public void stop(){

active = false;//line2

}

}

一般地，在一个线程中调用run()方法，在另外一个线程中调用stop()方法.如果线程（line1）中的active位于缓冲区的值被使用，那么当line2中的active设置成false时，循环也不会停止。

Java运算符

计算机的最基本用途之一就是执行数学运算，作为一门计算机语言，java也提供了一套丰富的运算符来操作变量。我们可以吧运算符分为以下几种：

* 算数运算符
* 关系运算符
* 位运算符
* 逻辑运算符
* 赋值运算符
* 其他运算符

算数运算符

算数运算符用在数学表达式中，它们的作用和在数学中的作用一样，下表列出来所有的算数运算符。

|  |  |  |
| --- | --- | --- |
| 操作符 | 描述 | 例子 |
| + | 加法-相加运算符两侧的值 | A + B等于30 |
| - | 减法-左操作数减去右操作数 | A-B等于-10 |
| \* | 乘法-相乘操作符两侧的值 | A\*B等于200 |
| / | 除法-左操作数除以右操作数 | B/A等于2 |
| % | 取模-左操作数除以右操作数的余数 | B%A等于0 |
| ++ | 自增-操作数的值增加1 | B++或++B等于21 |
| -- | 自减-操作数的值减少1 | B—或—B等于19 |

实例

下面的简单示例程序演示了算术运算符。赋值并粘贴下面的java程序并保存为Test.java文件，然后编译并运行这个程序:

Public class Test{

Public static void main(String args[]){

int a = 10;

int b = 20;

int c = 25;

int d = 25;

System.out.println(“a+b= ”+(a+b));

System.out.println(“a-b=”+(a-b))

.//没有写完，不想写了，后续看情况补

}

}

关系运算符

|  |  |  |  |
| --- | --- | --- | --- |
| == |  | 检查如果两个操作数的值是否相等，如果相等则条件为真。 | （A==B）为假（非真） |
| ！= |  | 检查如果两个操作数的值是否相等，如果值不相等则条件为真。 | （A!=B）为真 |
| > |  | 检查左操作数的值是否大于右操作数的值，如果是那么条件为真。 | （A>B）非真 |
| < |  | 检查左操作数的值是否小于右操作数的值，如果是那么条件为真。 | （A <B）为真。 |
| >= |  | 检查左操作数的值是否大于或等于右操作数的值，如果是那么条件为真。 | （A> = B）为假。 |
| <= |  | 检查左操作数的值是否小于或等于右操作数的值，如果是那么条件为真。 | （A <= B）为真。 |

位运算符

Java定义了位运算符，应用于整数类型（int），长整型（long），短整型（short），字符型（char），和字节型（byte）等类型。

位运算符作用在所有的位上，并且按位运算。假设 a = 60，和b=13；他们的二进制格式表示如下：

A = 0011 1100

B = 0000 1101

-----------------

A&b = 0000 1100

A | B = 0011 1101

A ^ B = 0011 0001

~A= 1100 0011

下表列出了位运算符的基本运算，假设整数变量A的值为60和变量B的值为13；

|  |  |  |
| --- | --- | --- |
| & | 按位与操作符，当且仅当两个操作数的某一位都非0时候结果的该位才为1. | （A&B），得到12，即  0000 1100 |
| | | 按位或操作符，只要两个操作数中的某一位有一个非0时候结果的该位就为1。 | （A|B）得到61，即  0011 1101 |
| ^ | 按位异或操作符，两个操作数的某一位不相同时候结果的该位就为1。 | （A^B）得到49，即  0011 0001 |
| ~ | 按位补运算符翻转操作数的每一位。 | （~A）得到-60，即  1100 0011 |
| << | 按位左移运算符。左操作数按位左移右操作数指定的位数（） | A<<2得到240，即 |
| >> | 按位右运算符。左操作数按位右移操作数指定的位数。 | A>>2得到15即1111 |
| >>> | 按位右移补零操作符。左操作数的值按右操作数指定的位数右移，移动得到的空位以零填充。 | A>>>2得到15即0000 1111 |

实例

下面的简单示例程序演示了位运算符。复制并粘贴下面的java程序并保存为Test.java文件，然后编译并运行这个程序：

public class Test{

public static void main(String args[]){

int a = 60; // 60 = 0011 1100

int b = 13;// 13 = 0000 1101

int c = 0;

c = a & b; //12 = 0000 1100

System.out.println(“a & b =” + c);

c = a | b; //61 = 0011 1101

System.out.println(“a | b =” + c);

c = a ^ b; //49 = 0011 0001

System.out.println("a ^ b = " + c );

c = ~a; /\*-61 = 1100 0011 \*/

System.out.println("~a = " + c );

c = a << 2; /\* 240 = 1111 0000 \*/

System.out.println("a << 2 = " + c );

c = a >> 2; /\* 215 = 1111 \*/

System.out.println("a >> 2 = " + c );

c = a >>> 2; /\* 215 = 0000 1111 \*/

System.out.println("a >>> 2 = " + c );

}

}

public class Test {

public static void main(String args[]) {

int a = 60; /\* 60 = 0011 1100 \*/

int b = 13; /\* 13 = 0000 1101 \*/

int c = 0;

c = a & b; /\* 12 = 0000 1100 \*/

System.out.println("a & b = " + c );

c = a | b; /\* 61 = 0011 1101 \*/

System.out.println("a | b = " + c );

c = a ^ b; /\* 49 = 0011 0001 \*/

System.out.println("a ^ b = " + c );

c = ~a; /\*-61 = 1100 0011 \*/

System.out.println("~a = " + c );

c = a << 2; /\* 240 = 1111 0000 \*/

System.out.println("a << 2 = " + c );

c = a >> 2; /\* 215 = 1111 \*/

System.out.println("a >> 2 = " + c );

c = a >>> 2; /\* 215 = 0000 1111 \*/

System.out.println("a >>> 2 = " + c );

}

}

a & b = 12

a | b = 61

a ^ b = 49

~a = -61

a << 2 = 240

a >> 15

a >>> 15

按位与操作符，当且仅当两个操作数的某一位都非0时候结果的该位才为1。 真真为真，1为真。真假为假，0为假。& 与

int a = 60; /\* 60 = 0011 1100 \*/

int b = 13; /\* 13 = 0000 1101 \*/

a & b

0000 1100 换算成16进制 等于 12

按位或操作符，只要两个操作数的某一位有一个非0时候结果的该位就为1。 真真为真，真假为真，假假为假。1为真，0为假。| 或

a | b

0011 1101

按位异或操作符，两个操作数的某一位不相同时候结果的该位就为1。 真假为真，真真为假，假假为假。1为真，0为假。 ^异或

a ^ b

0011 0001

按位补运算符翻转操作数的每一位。 真为假，假为真。 ~位补

~a

1100 0011

按位左移运算符。左操作数按位左移右操作数指定的位数。

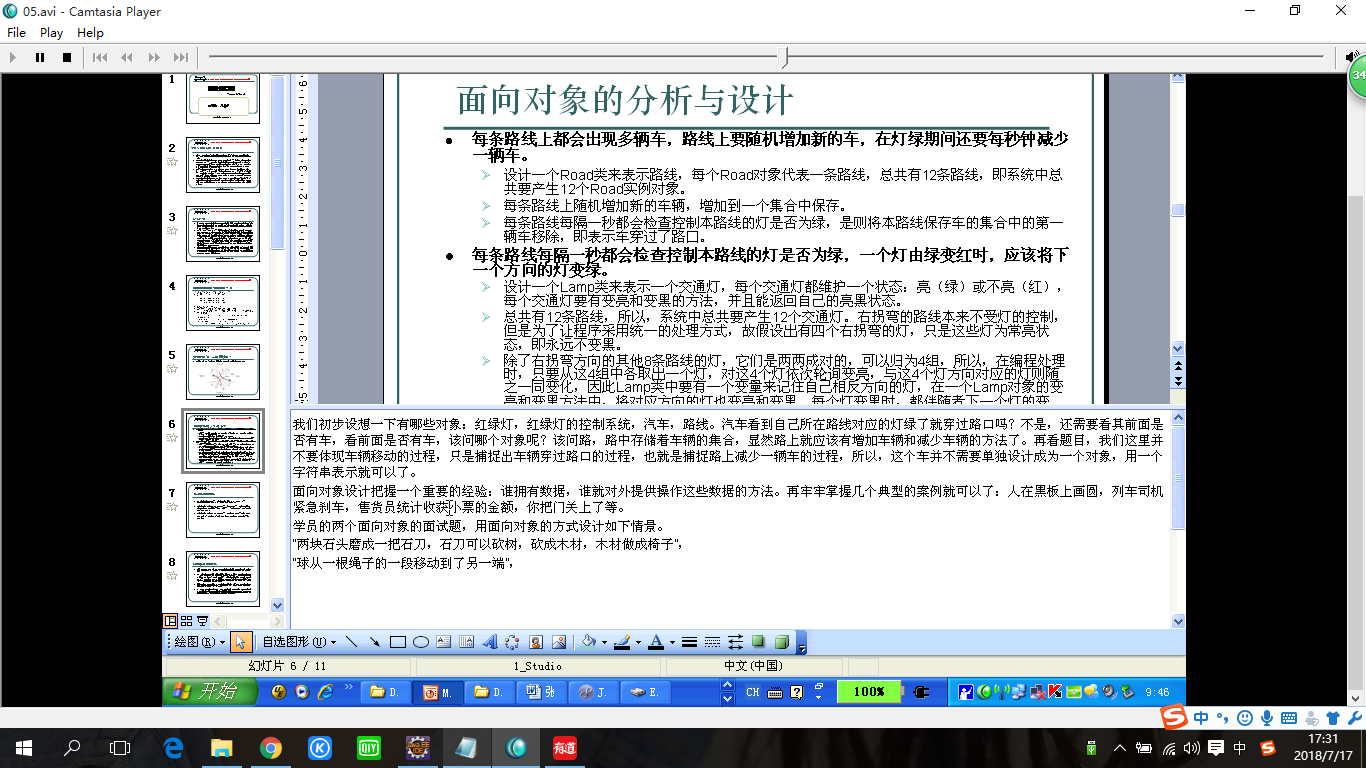
a << 2

0011 1100 << 2

1111 0000

>>

a >>> 2



面向对象的分析与设计。

题目：

球从一根绳子的一端移动到了另一端，

代码：

/\*

Rope 绳子抽象成一条横线，而按照题意，抽象出绳子的两个属性即移动的开始start和end

\*\*/

Class Rope{

private Point start;

private point end；

public Rope（Point start，Point end）{

this.start = start;

this.end = end;

}

public Point nextPoint(Point currentPoint){

/\*通过两点一线的数学公式可以计算出当前点的下一个点，这个细节不属于设计阶段要考虑的问题，如故宫当前点事终止点，则返回null，如果当前点不是线上的点，则抛出异常。

\*/

}

}

class Ball{

//引用绳子这个类，在同一个类中只允许存在一个主方法，同一个类中类与类可以相互引用

private Rope rope;

private Point currentPoint;

public Ball(Rope rope,startPoint){

this.rope = rope;

this currentPoint = startPoint;

}

public void move(){

currentPoint = rope.nextPoint(currentPoint);

System.out.println(“小球移动到了”+ currentPoint);

}

}

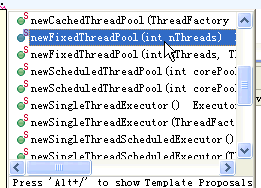


Figure https://www.cnblogs.com/ljp-sun/p/6580147.html

# [Eclipse的查找功能及其他技巧](https://blog.csdn.net/yn49782026/article/details/8608455)