1. 数据类型
2. byte 字节型，8bit，一个字节，和C不一样，不是char
3. char 字符型，8bit，也可进行数值运算
4. int 整型，32bit，4个字节
5. long 长整型，64bit，8个字节
6. short 短整型，16bit，2个字节
7. String 字符串，注意这个是对象，不是基本数据类型，防止搞混所以写在这里
8. float 单精度浮点，在给单精度浮点型赋值的时候最后要带上f，因为java默认双精度
9. double 双精度浮点
10. boolean 布尔型true/false
11. 基本数据类型的包装类
12. 基本数据类型的包装类对应关系如下图：



1. 包装类的好处：
   1. 封装了一些实用的方法和常量，如Byte.MIN\_VALUE是Byte类中存放byte数据类型最小值的常量
   2. 包装类在集合中用来定义集合元素的类型（不是很懂）
2. 通过Scanner从控制台获取信息

C可以通过scanf获取console的信息，java也可以，只不过java是面向对象语言，该工作自然也由类来完成，而且功能更强大，类Scanner可接受字符串和基本数据类型的数据，而C的scanf只能接受字符串，它位于java.util.Scanner包中，以下是常用方法：

1. Scanner.next()：接收字符串，空格或回车结尾，因此无法接收空格
2. Scanner.nextLine()：接收字符串，只能回车结束，因此可以接收空格
3. Scanner.nextInt()：接收一个int型数据
4. Scanner.nextDouble()：接收一个double型数据
5. Scanner.nextBoolean()：接收一个布尔型数据

ps：Scanner类没有提供直接接收char型的方法，可用charAt(int index)从next() or nextLine()获取，如cSex = myScanner.next().charAt(0);

1. 逻辑运算

符号和C都一样，要注意的是区分长路和短路，例如：

&长路与，符号两边是数字时，进行位与，是关系式时，无论第一个条件成立与否都会把下一个条件也算出来，&&相反，无位运算

1. 条件和循环语句

无论是if语句还是switch语句和C的格式都一模一样，switch同样无法处理字符串

循环语句也是，全部都一样

1. 数组

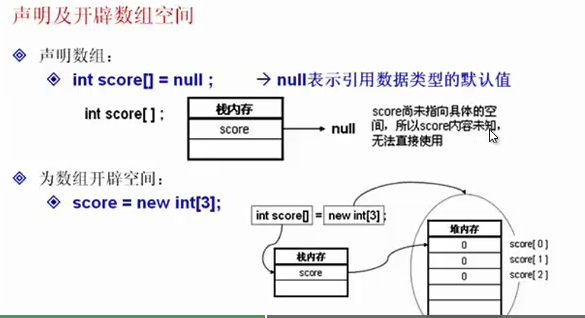
数组的差别和C比较大，具体如下：

1. java中数组的成员都是存在堆中的，而C都是存在栈中
2. 数组定义方式不同，C直接int a[10]即可，而java由于1中所说的原因，需要两个步骤：

声明：int a[] or int [] a

分配内存：a = new int[10]，也可称为实例化

下图可以很好的帮助理解



此外，java中若用循环给数组赋值，循环次数可由属性a.length指定，防止数组越界

ps：java数组的静态初始化和C一样都是int a[]={1,2,3}，但不能指定长度，二维数组也一样

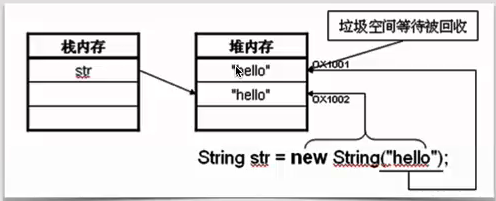
1. 字符串
2. 实例化String对象的两种方法
   1. 直接赋值

String str = “hello”

* 1. 使用关键字new

String str = new String(“hello”)

两种方法一般用第一种，原因如下图：



第二种方法会多存放一个hello

1. String内容比较

利用equals方法，不可用==，因为==比较的是字符串的地址，例：

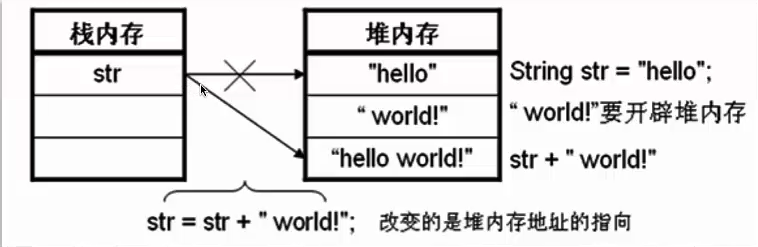
String str = “hh”;

String str1 = “hh”;

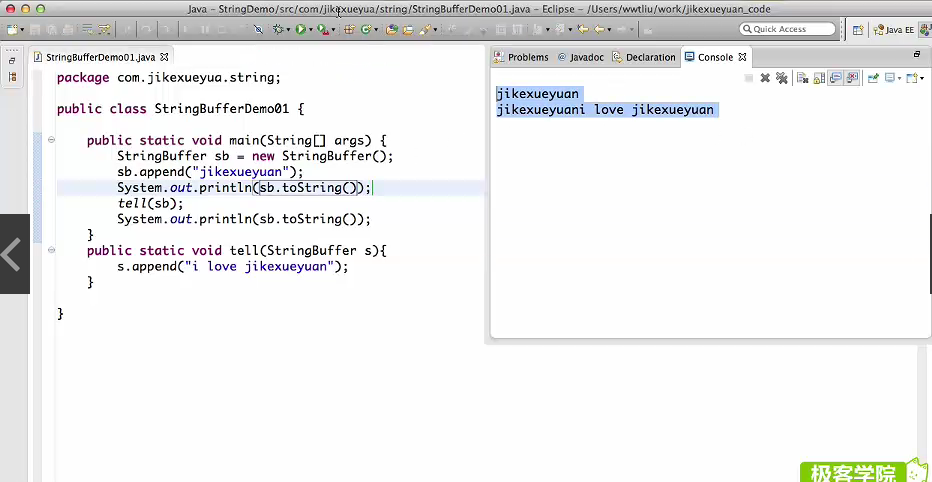
str.equals(str1);

1. String内容不可更改

String相加的时候改的不是内容而是地址指向：



1. 字符串常用方法
   1. 字符串长度：str.length()
   2. 字符串转字符数组：str.toCharArray()
   3. 字符串中取出制定位置的字符：Scanner中有讲到，str.charAt()
   4. 字符串转成byte数组:str.getBytes()
   5. 某字符在字符串中的位置:str.indexOf
   6. 去掉字符串的前后空格: str.trim()
   7. 字符串中取出子字符串：subString()
   8. 大小写转换：toLowerCase() toUpperCase()
   9. 判断字符串的开头结尾字符：endsWith() starWitch()
   10. 替换String字符串中的一个字符：repalce()
2. StringBuffer

它是一个操作类，但是它的内容可以更改，且不能直接赋值，要通过append()方法给它添加内容，后续再调用append就会加到之前字符串的末尾

常用方法：

append(字符串)

insert(位置，字符串)

indexOf()

repalce(开始位置，终止位置，字符串)

1. StringBuider

一个可变的字符序列，用来简易替换StringBuffer，单线程调用时优先考虑，多线程别用，方法和StringBuffer一样

1. 类与对象
2. 方法的创建和重载

创建：

访问修饰符 返回值类型 方法名（参数）{

方法主体

}

重载：

方法名称相同，但是参数类型和个数不同，通过参数类型和个数的不同来完成不同的功能。注意，返回值必须相同

1. 类定义及声明

定义：

class 类名称{

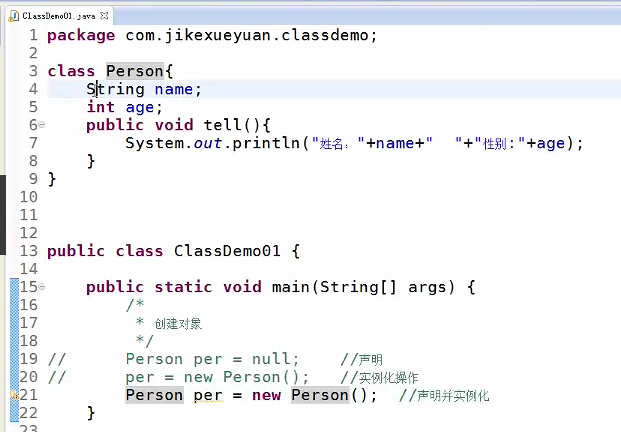
属性

方法

}

一般类名称开头字母大写

声明：声明一个类需要通过一个关键字class

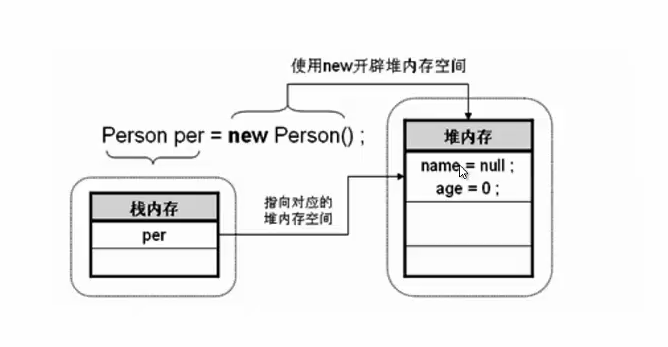


1. 类与对象的关系

类是对某一事物的描述，是抽象的、概念意义上的，对象是实际存在的该类事物的一个个体也被称为对象或实例

1. 类的内存划分

和数组差不多，下图一目了然：

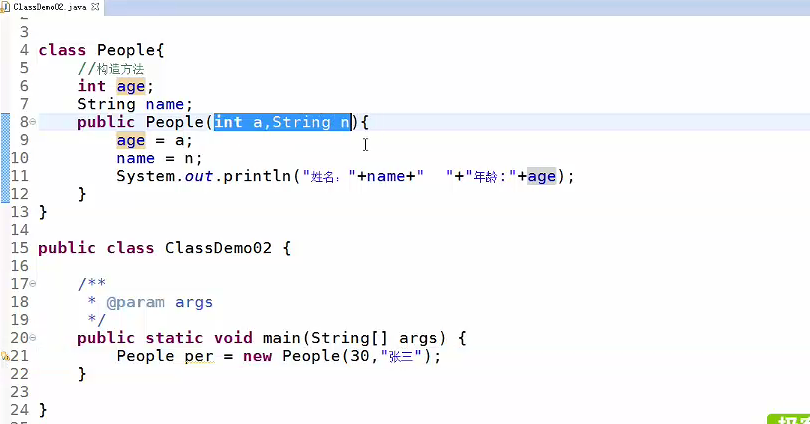


1. 面向对象三大特征
   1. 封装性  
      目的：保护某些属性和方法，对外部不可见  
      实现：使用关键子private声明，并实现该属性的get、set方法，让外部访问
   2. 继承  
      扩展类的功能
   3. 多态性  
      方法的重载

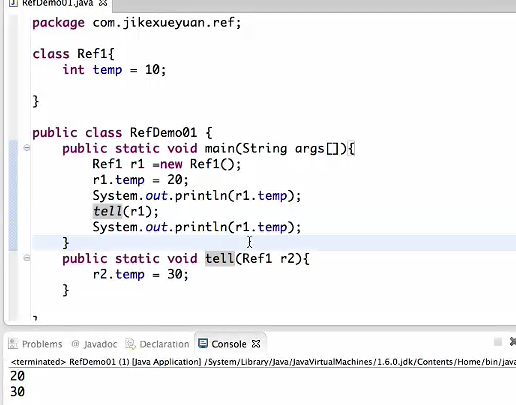
对象的多态性

1. 构造方法

和C++一样，java也有类的构造方法，不需要程序员写代码调用，创建类的实例时自动调用，方法名称和类名称相同，无需返回值，主要目的是为类中的属性做初始化，也可以重载



1. 引用传递

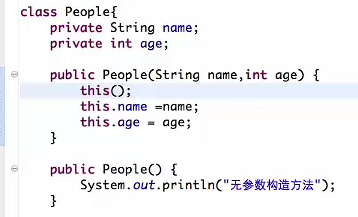


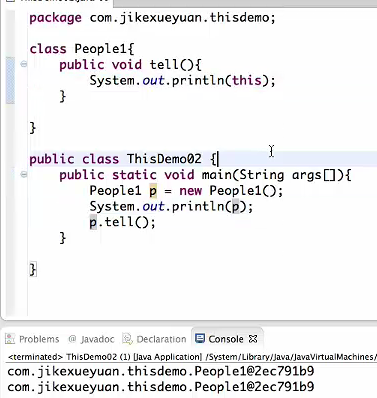
上图中，temp的值被改了，乍一看和C语言不一样，形参的值被改了，实际上并不是这样，这并不是一个值传递。可以回顾一下对象实例化的过程：

Ref1 r1=null;

r1 = new Ref1();

r1初始值位null，也就是说r1是个指针，那对于tell方法也一样，r2实际上是个指针，所以temp的值可以被改变。当然如果r1是个String类那就改不了了，这个是特殊情况，但是如果class中的temp是个String类，也是可以被改的，实际上就是让temp指向了另一块堆内存

1. this关键字
   1. 表示类中的属性和调用方法
   2. 调用本类中的构造方法  
        
      注意，图中的this（）必须放在构造方法的首行
   3. 表示当前对象



1. static关键字
   1. static声明属性属于全局属性  
      某个类的多个实例中的任意一个实例修改了static的属性，其他所有实例的该属性值都会改变，而且一般使用类名调用，不用实例名调用
   2. static声明的方法可以直接通过类名调用，同上
   3. 使用static方法的时候只能访问static声明的属性和方法，非static声明的属性和方法不能访问
2. 继承

基本概念：

扩展父类功能

实现：

class 子类名 extends 父类名{

}

限制：

只允许单继承，一个子类不能继承自多个父类，相当于一个儿子只能有一个爹；但 是可一多层继承，相当于孙子和爷爷

子类不能访问父类的私有成员，必须通过get和set方法访问

子类对象实例化：

在子类对象实例化前，先调用父类的构造方法再调用子类的构造方法，也就是现有爹再有儿子

方法重写：

子类定义了和父类同名的方法，且返回值类型相同，参数也相同，若同时想在重写方法中调用父类的方法，用super关键字调用即可，和Objective-C一样。被子类重写的方法不能拥有比父类方法更严格的访问权限

访问权限：

private(只能在类中访问) < default(整个包都能访问) < public(整个工程都能访问)。ps:一个文件中只能由一个public class

1. final
   1. final关键字在java中被称为完结器，表示最终的意思
   2. final能声明类、方法、属性
      1. 使用final声明的类不能被继承
      2. 使用final声明的方法不能被重写
      3. 使用final声明的变量变成常量，不能被修改
2. 多态性
   1. 向上转型

父类 父类对象 = 子类实例，这种形式称为父类的引用指向子类的对象

父类类型的引用可以调用父类中定义的所有属性和方法，而子类中定义而父类中未定义的方法无法访问

父类中的一个方法只有在子类中未被重写才能被父类类型调用

父类中定义的方法如果在子类中被重写了，那父类类型的引用将调用子类中的方法，这个就是动态链接

* 1. 向下转型

子类 子类对象 = (子类)父类实例

由于向上转型的父类类型引用无法调用子类中定义而父类中未定义的方法，此时可以用向下转型的子类对象引用子类的这些方法

小结：

方法的重载与重写就是方法的多态性表现

多个子类就是父类中的多种形态

父类引用可以指向子类对象，自动转换

子类对象指向父类引用需要强转

实际开发中尽量使用父类引用，利于扩展