类和对象

类和对象的关系

- **类：类是对现实生活中一类具有共同属性和行为的事物的抽象**
- **对象：是能够看得到摸的着的真实存在的实体**
- **简单理解：类是对事物的一种描述，对象则为具体存在的事物**

类的定义步骤：

① 定义类

② 编写类的成员变量

③ 编写类的成员方法

创建对象的格式： **类名 对象名 = new 类名();**

new：在堆内存开辟空间

在函数中定义的一些基本类型的变量和对象的引用变量都是在函数的栈内存中分配，

当在一段代码块定义一个变量时，Java 就在栈中为这个变量分配内存空间，当超过变量的作用域后(比如，在函数A中调用函数B，在函数B中定义变量a，变量a的作用域只是函数B，在函数B运行完以后，变量a会自动被销毁。分配给它的内存会被回收)，

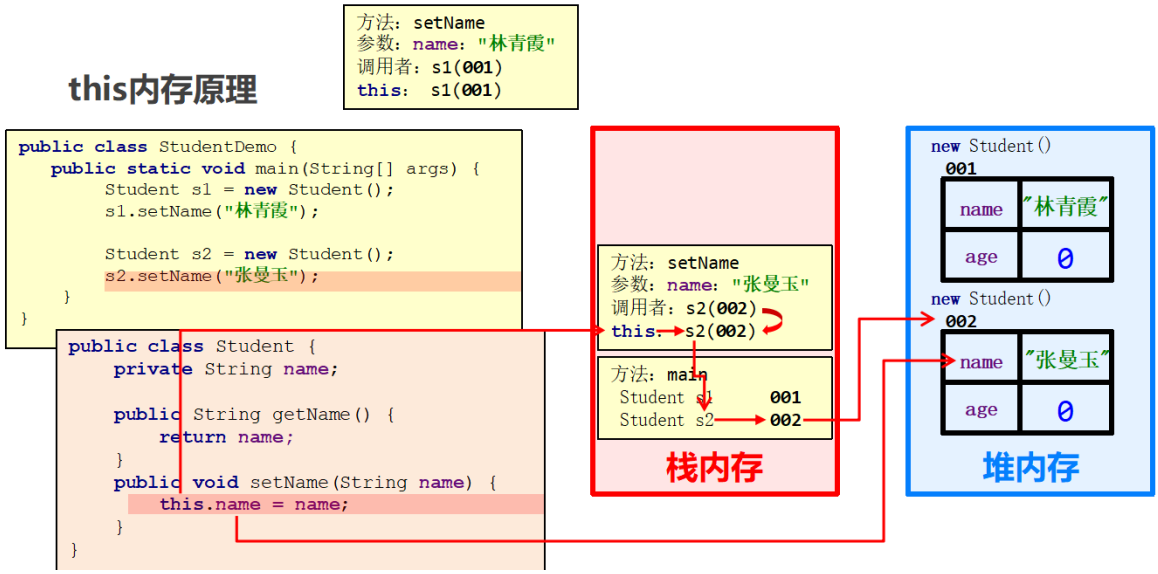
Java 会自动释放掉为该变量分配的内存空间，该内存空间可以立即被另作它用。

堆内存用来存放由 new 创建的对象和数组，在堆中分配的内存，由 Java 虚拟机的自动垃圾回收器来管理。

在堆中产生了一个数组或者对象之后，还可以在栈中定义一个特殊的变量，让栈中的这个变量的取值等于数组或对象在堆内存中的首地址，栈中的这个变量就成了数组或对象的引用变量，以后就可以在程序中使用栈中的引用变量来访问堆中的数组或者对象，引用变量就相当于是为数组或者对象起的一个名称。

引用变量是普通的变量，定义时在栈中分配，引用变量在程序运行到其作用域之外后被释放。而数组和对象本身在堆中分配，即使程序运行到使用 new 产生数组或者对象的语句所在的代码块之外，数组和对象本身占据的内存不会被释放，数组和对象在没有引用变量指向它的时候，才变为垃圾，不能在被使用，但仍然占据内存空间不放，在随后的一个不确定的时间被垃圾回收器收走(释放掉)。这也是 Java 比较占内存的原因，

实际上，栈中的变量指向堆内存中的变量，这就是 Java 中的指针！



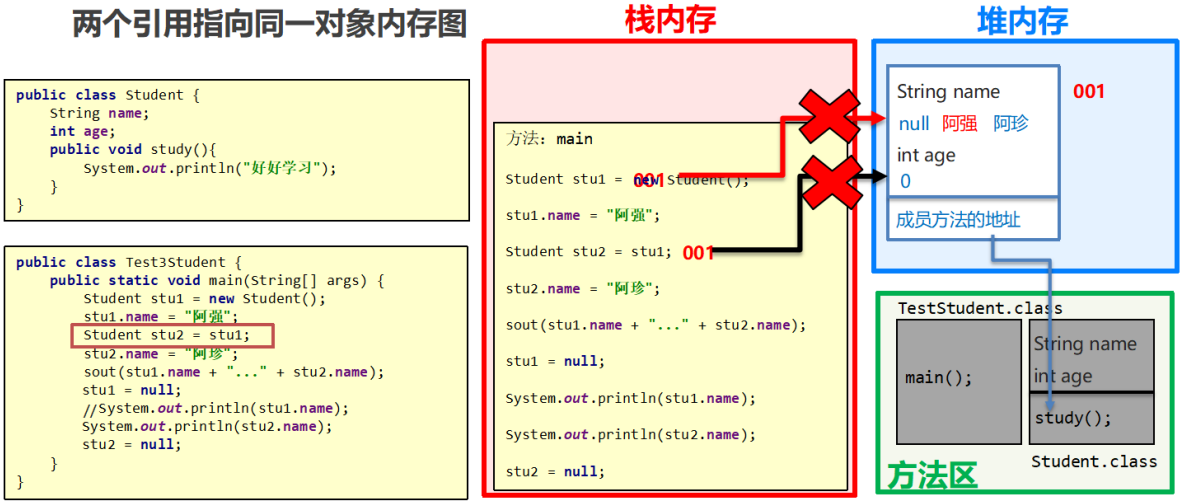

```
package com.itheima.object1;
public class TestStudent {
/*
创建对象的格式:
类名 对象名 = new 类名();
调用成员变量的格式:
对象名.变量名
调用成员方法的格式:
对象名.方法名();
*/


public class Student {
// 属性 : 姓名, 年龄
// 成员变量: 跟之前定义变量的格式一样, 只不过位置发生了改变, 类中方法外
String name;
int age;
// 行为 : 学习
// 成员方法: 跟之前定义方法的格式一样, 只不过去掉了static关键字.
public void study(){
System.out.println("学习");
}
}
```

```
public static void main(String[] args) {
// 类名 对象名 = new 类名();
Student stu = new Student();
// 对象名.变量名
// 默认初始化值
System.out.println(stu.name); // null
System.out.println(stu.age); // 0
stu.name = "张三";
stu.age = 23;
System.out.println(stu.name); // 张三
System.out.println(stu.age); // 23
// 对象名.方法名();
stu.study();
// com.itheima.object1.Student@b4c966a
// 全类名(包名 + 类名)
System.out.println(stu);
}
}
```

多个对象指向相同内存图

控制台：阿珍...阿珍
控制台：NullPointerException
控制台：阿珍



当多个对象的引用指向同一个内存空间（变量所记录的地址值是一样的）

只要有任何一个对象修改了内存中的数据，随后，无论使用哪一个对象进行数据获取，都是修改后的数据。

成员变量和局部变量的区别

类中位置不同：成员变量（类中方法外）局部变量（方法内部或方法声明上）

内存中位置不同：成员变量（堆内存）局部变量（栈内存）

生命周期不同：成员变量（随着对象的存在而存在，随着对象的消失而消失）局部变量（随着方法的调用而 存在，醉着方法的调用完毕而消失）

初始化值不同：成员变量（有默认初始化值）局部变量（没有默认初始化值，必须先定义，赋值才能使用）

封装

private是一个修饰符，可以用来修饰成员（成员变量，成员方法）

特点：被private修饰的成员，只能在本类进行访问，针对private修饰的成员变量，如果需要被其他类使用， 提供 相应的操作

提供“ *get变量名()* ”方法，用于获取成员变量的值，方法用public修饰

提供“ *set变量名(参数)* ”方法，用于设置成员变量的值，方法用public修饰

```
/*
学生类
*/
class Student {
//成员变量
String name;
private int age;
//提供get/set方法
public void setAge(int a) { // 设置成员变量的值
if(a<0 || a>120) {
System.out.println("你给的年龄有误");
} else {
age = a;
}
}
public int getAge() {  // 获取成员变量的值
return age;
}
//成员方法
public void show() {
System.out.println(name + "," + age);
}
}
/*
学生测试类
*/
public class StudentDemo {
public static void main(String[] args) {
//创建对象
Student s = new Student();
//给成员变量赋值
s.name = "林青霞";
s.setAge(30);
//调用show方法
s.show();
}
}
```

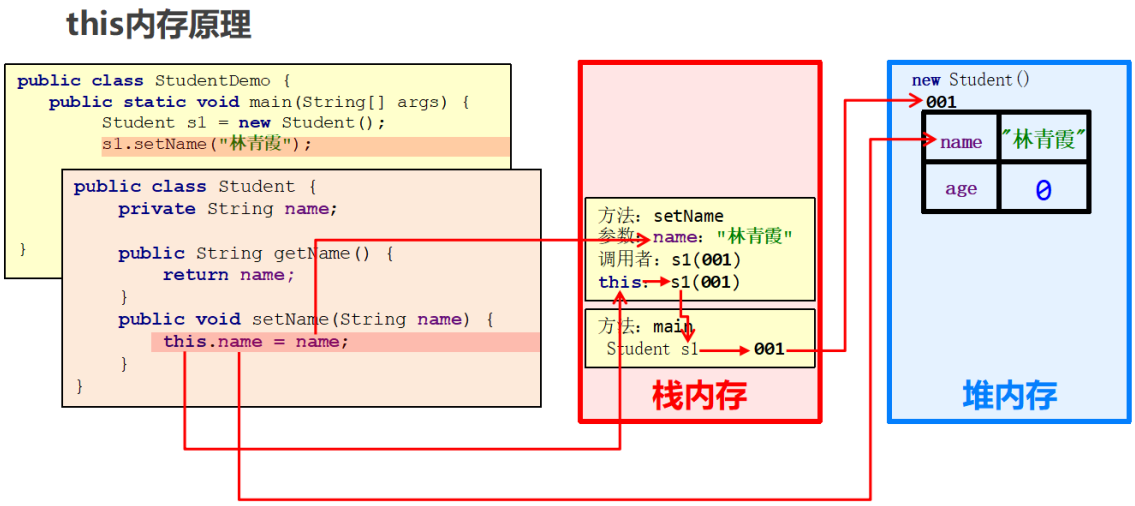
this关键字

跟python的self 差不多，用法也一样

方法的形参如果与成员变量同名，不带this修饰的变量指的是形参，而不是成员变量

方法的形参没有与成员变量同名，不带this修饰的变量指的是成员变量

this内存原理



封装思想

1. 封装概述 是面向对象三大特征之一（封装，继承，多态） 是面向对象编程语言对客观世界的模拟，客观世界里成员变量都是隐藏在对象内部的，外界是无法直接操作 的
2. 封装原则 将类的某些信息隐藏在类内部，不允许外部程序直接访问，而是通过该类提供的方法来实现对隐藏信息的操 作和访问 成员变量private，提供对应的getXxx()/setXxx()方法
3. 封装好处 通过方法来控制成员变量的操作，提高了代码的安全性 把代码用方法进行封装，提高了代码的复用性

构造方法

格式注意：

方法名与类名相同，大小写也要一致

没有返回值类型，连void都没有

没有具体的返回值（不能由retrun带回结果数据）

执行时机：

创建对象的时候调用，每创建一次对象，就会执行一次构造方法

不能手动调用构造方法

```
class Student {
    private String name;
    private int age;
    //构造方法
public Student() {
System.out.println("无参构造方法");
}
public void show() {
System.out.println(name + "," + age);
    }
}
/*
```

```
测试类
*/
public class StudentDemo {
public static void main(String[] args) {
//创建对象
Student s = new Student();
s.show();
    }
}
```

构造方法的作用

用于给对象的数据（属性）进行初始化

\1. 如果一个类中没有编写任何构造方法, 系统将会提供一个默认的无参数构造方法

\2. 如果手动编写了构造方法, 系统就不会再提供默认的无参数构造方法了

构造方法的注意事项

构造方法的创建：

如果没有定义构造方法，系统将给出一个默认的无参数构造方法

如果定义了构造方法，系统将不再提供默认的构造方法

构造方法的创建：

如果没有定义构造方法，系统将给出一个默认的无参数构造方法如果定义了构造方法，系统将不再提供默认的构造 方法

推荐的使用方式：

无论是否使用，都手动书写无参数构造方法，和带参数构造方法