第一章 面向对象思想

1.1 面向对象思想概述

Java语言是一种面向对象的程序设计语言，而面向对象思想是一种程序设计思想，我们在面向对象思想的指引下，使用Java语言去设计、开发计算机程序。这里的**对象**泛指现实中的一切事物，每种事物都具有自己的**属性**和**行为**。面向对象思想就是在计算机程序设计过程中，参照现实中事物，将事物的属性特征，行为特征抽象出来，描述成计算机事件的设计思想。它区别于**面向过程**思想，**强调的是通过调用对象的行为来实现功能，而不是自己一步一步地去操作实现。**

- **面向过程**：强调步骤
- **面向对象**：强调对象

特点：面向对象思想是一种更符合我们思考习惯的思想，他可以将复杂的事情简单化，并将我们从执行者变成了指挥者。面向对象的语言中，包含了三大基本特征，即**封装、继承和多态**

1.2 类和对象

1.2.1 什么是类

类：是一组相关**属性**和**行为**的集合。可以看成是一类事物的模板，使用事物的属性特征和行为特征来描述该类事物

- 属性：该事物的状态信息
- 行为：该事物能够做什么

例：猫

属性：名字、体重、年龄、颜色

行为：走、跑、叫

1.2.2 什么是对象

对象：是一类事物的具体体现。对象是类的一个**实例**，必然具备该类事物的属性和行为

例：一只猫（一类事物的一个实例）

属性：Tom、5kg、2years、Yellow

行为：贴墙角走、蹦跶着跑、喵喵叫

1.2.3 类与对象的关系

- 类是对一类事物的描述，是**抽象的**
- 对象是一类事物的实例，是**具体的**
- **类是对象的模板，对象是类的实体**

1.3 类的定义

1.3.1 事物与类的对比

现实世界的一类事物：

- **属性**：事物的状态信息
- **行为**：事物能够做什么

Java中用class描述事物：

- **成员变量**：对应事物的**属性**
- **成员方法**：对应事物的**行为**

1.3.2 类的定义格式

```
public class ClassName{
    //成员变量
    //成员方法
}
```

- **定义类**：就是定义类的成员，包括**成员变量**和**成员方法**
- **成员变量**：和以前定义变量几乎是一样的。只不过位置发生了改变。在**类中，方法外。**
- **成员方法**：和以前定义方法几乎是一样的。只不过把`static`去掉，`static`的作用在后面将详细讲解

例：

```
public class Student{
    //成员变量
    String name;
    int age;
    //成员方法
    public void study() {
        System.out.println("好好学习，天天向上");
    }
    public void eat() {
        System.out.println("好好吃饭");
    }
}
```

1.4 对象的使用

1.4.1 对象的使用格式

创建对象：

```
类名 对象名 = new 类名();
```

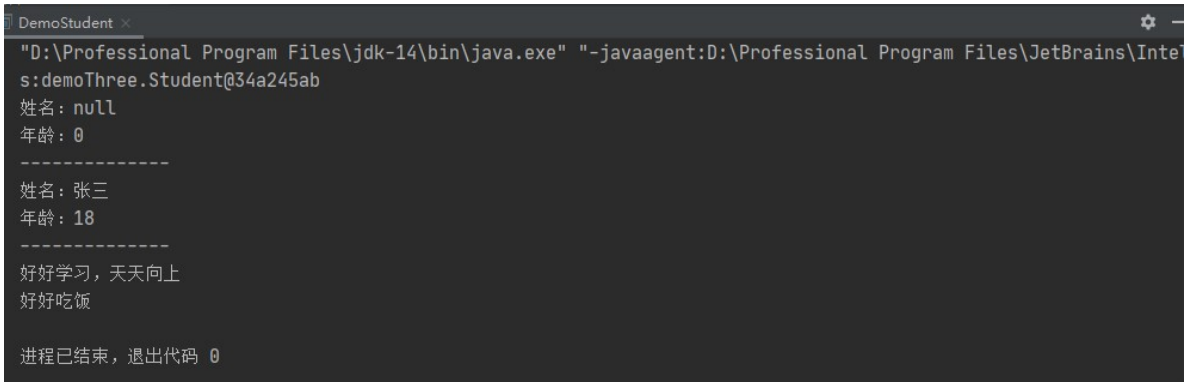
使用对象访问类中的成员：

```
对象名.成员变量;
对象名.成员方法();
```

例：

```
public class DemoStudent{
    public static void main(String[] args) {
        //创建对象格式
        Student s = new Student();
```

```
        System.out.println("s:"+s);
        //首次输出成员变量值
        System.out.println("姓名："+s.name);
        System.out.println("年龄："+s.age);
        System.out.println("--------------");
        //给成员变量赋值
        s.name = "张三";
        s.age = 18;
        //再次输出成员变量值
        System.out.println("姓名："+s.name);
        System.out.println("年龄："+s.age);
        System.out.println("--------------");
        //调用成员方法
        s.study();
        s.eat();
    }
}
```

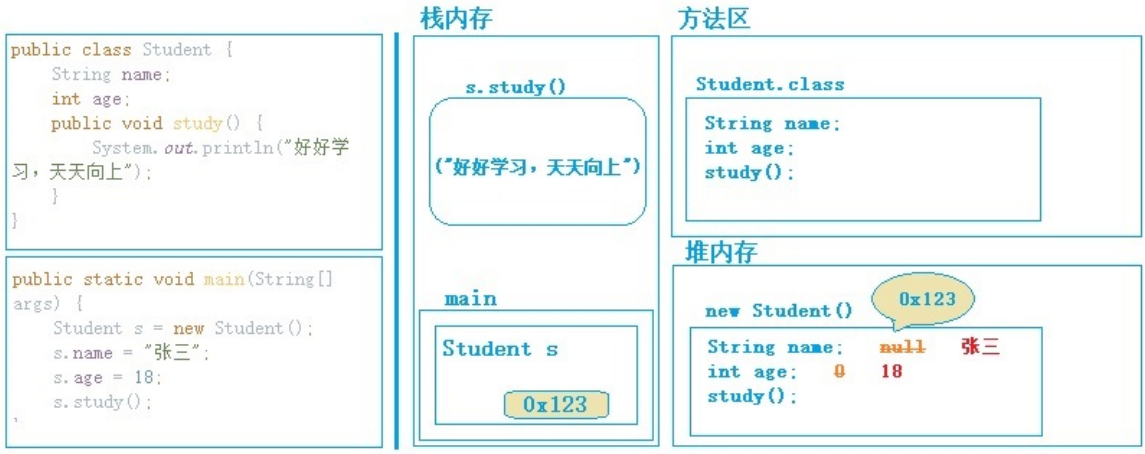


1.4.2 成员变量的默认值

	数据类型	默认值
基本类型	整数（byte，short，int，long）	0
	浮点数（float，double）	0.0
	字符（char）	'\u0000'
	布尔（boolean）	false
引用类型	数组，类，接口	null

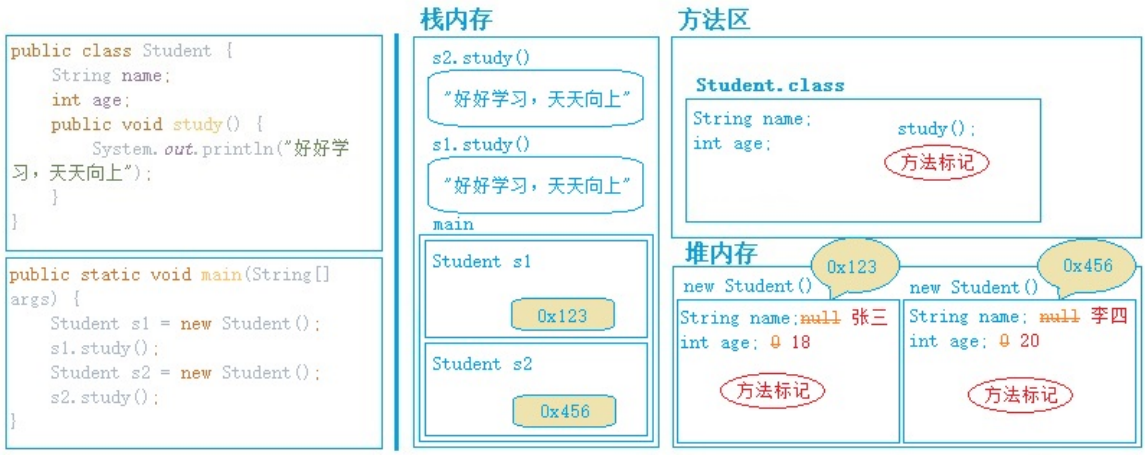
1.5 对象内存图

1.5.1 一个对象，调用一个方法内存图



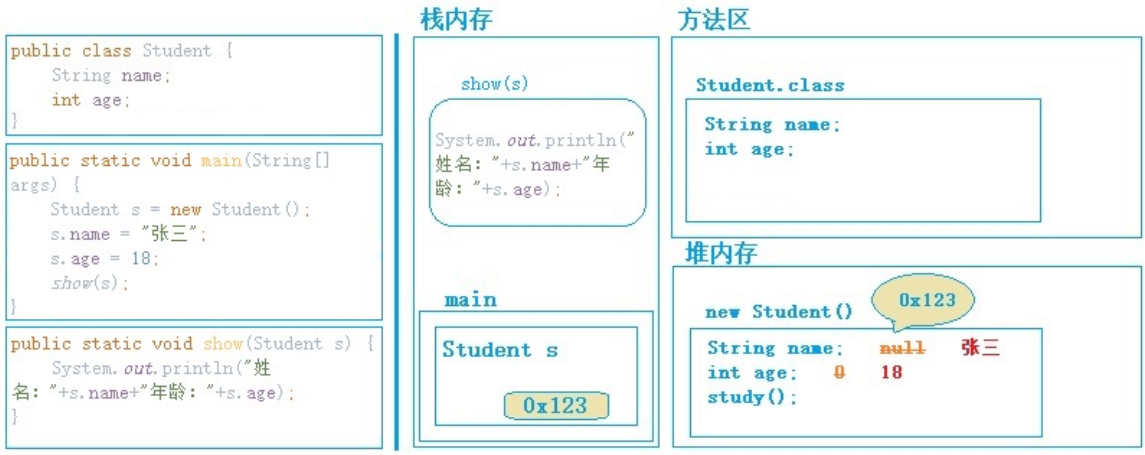
> 通过上图，可以理解，在栈内存中运行的方法，遵循“先进后出，后进先出”的原则。变量s指向堆内存中的空间，寻找方法信息，去执行该方法

1.5.2 两个对象，调用同一方法内存图



> 对象调用方法时，更具对象中方法标记（地址值），去类中寻找方法信息。这样哪怕是多个对象，方法信息只保存一份，节约内存空间

1.5.3 一个引用，作为参数传递到方法中内存图



> 引用类型作为参数，传递的是地址值

1.6 成员变量和局部变量的区别

- 在类中的位置不同【重点】

    - 成员变量：类中，方法外
    - 局部变量：方法中或者方法声明上（形式参数）
- 作用范围不一样【重点】
    - 成员变量：类中
    - 局部变量：方法中
- 初始化值的不同【重点】
    - 成员变量：有默认值
    - 局部变量：没有默认值，必须先定义，赋值，最后使用
- 在内存中的位置不同【了解】
    - 成员变量：堆内存
    - 局部变量：栈内存
- 生命周期不一样【了解】
    - 成员变量：随着对象的创建而存在，随着对象的消失而消失
    - 局部变量：随着方法的调用而存在，随着方法的调用完毕而消失

第二章 封装

面向对象编程语言是对客观世界的模拟，客观世界里成员变量都是隐藏在对象内部的，外界无法直接操作和修改，封装可以被认为是一个保护屏障，防止该类的代码和数据被其他类随意访问，要访问该类的数据，必须通过指定的方式。适当的封装可以让代码更容易理解与维护，也加强了代码的安全性。

- 原则：将**属性隐藏**起来，若是需要访问某个属性，**提供公共方法**对其访问

2.1封装的步骤

1. 使用`private`关键字来修饰成员变量
2. 对需要访问的成员变量，提供对应的一对`getter`、`setter`方法

2.2 封装的操作（`private`关键字）

2.2.1 `private`的含义

1. `private`是一个权限修饰符，代表最小权限
2. 可以修饰成员变量和成员方法
3. 被`private`修饰后的成员变量和成员方法，只在本类中才能访问

2.2.2 `private`的使用格式

```
private 数据类型 变量名;
```

例：

1. 使用`private`修饰成员变量

```
public class Student {
    private String name;
    private int age;
}
```

2. 提供`getter`、`setter`方法，可以访问成员变量

```
public class Student {
    String name;
    int age;

    public String getName() {
        return name;
    }

    public void setName(String n) {
        name = n;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int a) {
        age = a;
    }
}
```

2.3 封装优化（`this`关键字）

在上述代码中，`setter`方法中的形参名字并不符合见名知意的规定，但是如果修改与成员变量名一致，又会造成形参变量名与成员变量名重名，导致成员变量名被隐藏，方法中的变量名，无法访问到成员变量，从而赋值失败。所以，我们使用`this`关键字，来解决重名问题

2.3.1 `this`的含义

`this`代表所在类的当前对象的引用（地址值），即对象自己的引用

- 方法被哪个对象调用，方法中的`this`就代表那个对象。即**谁在调用，`this`就代表谁**

2.3.2 `this`的使用格式

```
this.成员变量名;
```

使用`this`修饰方法中的变量，解决成员变量被隐藏的问题，代码如下：

```
public class Student {
    String name;
    int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
```

```
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

> 注：方法中只有只有一个变量名时，默认也是使用`this`修饰，可以省略不写

2.4 封装优化（构造方法）

当一个对象被创建时候，构造方法用来初始化该对象，给对象的成员变量赋初始值

> 注：无论是否自定义构造方法，所有的类都有构造方法，因为Java自动提供了一个无参构造方法，一旦自己定义了构造方法，Java自动提供的无参构造方法就会失效

2.4.1 构造方法的定义格式

```
修饰符 构造方法名(参数列表) {
    //方法体
}
```

构造方法的写法上，**方法名与它所在类名相同。**它没有返回值，所以不需要返回值类型，`void`也不需要

例：

```
public class Student {
    private String name;
    private int age;
    //无参构造方法
    public Student() {}
    //有参构造方法
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

注：

- 如果你不提供构造方法，系统会给出无参构造方法
- 如果你提供了构造方法，系统将不再提供无参构造方法
- 构造方法是可以重载的，即可以定义参数，也可以不定义参数

2.5 标准代码（`JavaBean`）

`JavaBean`是Java语言编写类的一种标准规范。符合`JavaBean`的类，要求类必须是具体的和公开的，并且具有无参构造方法，提供用来操作成员变量的`getter`、`setter`方法，以学生类为例，标准代码如下：

```
public class Student {
    String name;
```

```
    int age;

    public Student() {
    }

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

测试类代码：

```
public class DemoStudent {
    public static void main(String[] args) {
        //使用无参构造
        Student s1 = new Student();
        s1.setName("张三");
        s1.setAge(18);
        System.out.println(s1.getName()+"---"+s1.getAge());

        //使用有参构造
        Student s2 = new Student("李四",20);
        System.out.println(s2.getName()+"---"+s2.getAge());
    }
}
```

运行结果：