day1-面向对象入门(类和对象)

今日内容

- 面向对象编程思想
- 类的定义
- 对象的创建和使用
- 对象的内存图
- 成员变量和局部变量的区别
- 封装
- 构造方法
- API

教学目标

- 能够知道类和对象的关系
- 能够完成类的定义及使用
- 能够知道对象在内存中的初始化过程
- 能够知道局部变量和成员变量的区别
- 能够知道private关键字的特点
- 能够知道this关键字的作用
- 能够知道构造方法的格式和注意事项
- 能够完成一个标准类代码的编写及测试
- 能够知道帮助文档的使用步骤

知识点1-类和对象

知识点--1. 面向对象和面向过程编程思想

目标

- 理解面向过程和面向对象编程思想

路径

- 面向过程编程思想
- 面向对象编程思想
- 举例对比2种编程思想

讲解

编程思想其实就是编程思路,我们开发中2种经典的编程思想就是面向过程编程思想和面向对象编程思想.

面向过程编程思想

- **强调的是过程,必须清楚每一个步骤,然后按照步骤一步一步去实现**

面向对象编程思想

- 强调的是对象, 通过调用对象的行为来实现功能，而不是自己一步一步的去操作实现。

举例对比2种编程思想

- 洗衣服:
  - 面向过程：把衣服脱下来-->找一个盆-->放点洗衣粉-->加点水-->浸泡10分钟-->揉一揉-->清洗衣服-->拧干-->晾起来
  - 面向对象: 把衣服脱下来-->给女朋友去洗
- 吃饭
  - 面向过程: 买菜--->洗菜--->炒菜--->吃
  - 面向对象: 找个饭店-->10块钱
- java程序: 需求:打印数组中所有的元素,打印格式为: [元素1，元素2，元素3，元素，...，元素n]

```
private static void method01() {
    // 1.面向过程来实现该需求
    // 1.1 定义一个数组,并且初始化数组中的元素
    int[] arr = {10, 20, 30, 40, 50};

    // 1.2 按照指定格式去打印
    // 1.2.1 先打印一个左中括号 "["
    System.out.print("[");

    // 1.2.2 循环遍历元素
    for (int i = 0; i < arr.length; i++) {
        // 1.2.3 判断遍历的元素是否是最后一个元素
        int e = arr[i];
        if (i == arr.length - 1) {
            // 1.2.5 如果是最后一个元素,那么打印格式就是 "元素]"
            System.out.print(e+"]");
        } else {
            // 1.2.4 如果不是最后一个元素,那么打印格式就是 "元素, "
            System.out.print(e+", ");
        }
    }


    // 2.面向对象来实现该需求
    // Arrays数组的工具类,toString()方法,可以帮助我们按照该指定格式打印数组
    System.out.println(Arrays.toString(arr));// [10, 20, 30, 40, 50]
}
```

小结

- 面向过程:是一种编程思想
- 面向对象:是一种编程思想
- 区别:
  - 面向过程:注重的是步骤,必须清楚每一个步骤,按照步骤一步一步去实现
  - 面向对象:注重的是对象,无须清楚每一个步骤,只需要使用对象调用行为来完成需求

知识点--2. 类的概述

目标

- 能够理解什么是类,以及类由什么组成

路径

- 类的概述
- 类的组成

讲解

类的概述

- 类是用来描述一类具有**共同属性和行为事物的统称**。所以其实类在客观世界里是不存在的，**是抽象的**，只是用来描述数据信息的。
- 人类: 描述信息
- 手机类: 描述信息
- 学生类: 描述信息
- 狗类: 描述信息
- ...

类的组成

- 属性：就是该事物的状态信息。
- 行为：就是该事物能够做什么。

举例

- 手机类
    - 属性：品牌、价格...。
    - 行为：打电话、发短信...。

小结

- 类是用来描述一群具有共同属性和行为事物的统称,类是抽象的,看不见,摸不着的,用来描述数据信息的
- 类的组成:
    - 属性
    - 行为

知识点--3. 对象的概述

目标

- 理解什么是对象

路径

- 对象的概念
- 举例

讲解

对象的概念

- 对象是类的一个实例(并不是你的女朋友哈)，**具体存在的，看得见摸得着的**，并且具备该类事物的属性和行为
    - 对象的属性:对象的属性具有特定的值
    - 对象的行为:对象可以操作的行为
- 人类: 描述信息 抽象
    - 对象: [22班-李睿]同学
    - 对象: 彭老师

- 手机类: 描述信息
  - 对象: 李睿同学手上这台手机
- 学生类: 描述信息
  - 对象: 22班-李睿同学
- 狗类: 描述信息
  - 对象: 李睿同学家的那条狗
- ...

举例

- 对象: 你手上拿的这台手机
  - 属性：华为、1999。 对象的属性具体的值,类中的属性没有具体的值
  - 行为：使用打电话功能,使用发短信功能。对象可以使用行为

小结

- 对象是类的实例,具体存在的,看得见摸得着的
- 对象的属性是有具体的值
- 对象的行为其实就是可以使用的功能\行为

知识点--4. 类和对象的关系

目标

- 理解类和对象的关系

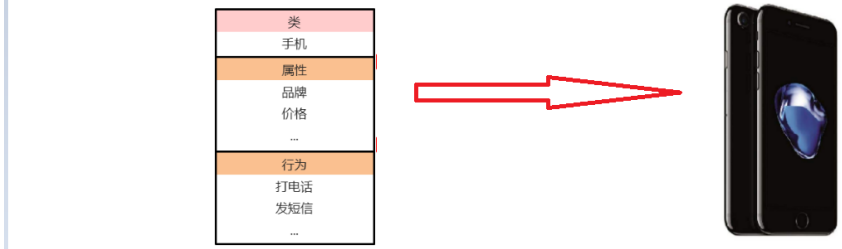
路径

- 类和对象的关系

讲解

类和对象的关系

- 类是对一类具有共同属性和行为的事物的统称,是抽象的
- 对象是一类事物的具体实例,看得见,摸的着的,真实存在的实体,是具体的
- 类是对象的抽象,对象是类的实体



小结

- 对象是根据类来创建的,类中有什么,对象就有什么,可以把类看成是对象的数据类型
- int num = 10; 10是int类型
- 类 变量名 = 对象; 对象是属于左边类 类型

知识点--5. 类的定义【应用】

目标

- 掌握如何定义一个类

路径

- 回顾类的组成
- 类的定义步骤
- 类的定义格式
- 举例

讲解

复习类的组成

类的组成是由属性和行为两部分组成

- 属性：该类事物的状态信息,在类中通过成员变量来体现（类中方法外的变量）
- 行为：该类事物有什么功能,在类中通过成员方法来体现（和前面的方法相比去掉static关键字即可）

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

类的定义格式

```
public class 类名 {// 定义一个类
    // 类里面:属性(成员变量),行为(成员方法)
    // 定义成员变量
    数据类型 变量名1;
    数据类型 变量名2;
    ...

    // 定义成员方法
    方法;  去掉static
}
```

举例

- 定义一个手机类,类名为(Phone),类的属性有:品牌(brand),价格(price),类的行为:打电话(call),发短信(sendMessage)

```
/*
    手机类:
        类名:
        手机(Phone)

        成员变量:
        品牌(brand)
        价格(price)
```

```
        成员方法:
        打电话(call)
        发短信(sendMessage)
 */
public class Phone {
    //成员变量
    String brand;
    int price;

    //成员方法
    public void call() {
        System.out.println("打电话");
    }

    public void sendMessage() {
        System.out.println("发短信");
    }
}
```

小结

- 定义类的格式

```
public class 类名 {// 定义一个类
    // 类里面:属性(成员变量),行为(成员方法)
    // 定义成员变量
    数据类型 变量名1;
    数据类型 变量名2;
    ...

    // 定义成员方法
    方法;  去掉static
}
```

知识点--6. 对象的创建和使用

目标

- 掌握对象的创建和对象的使用

路径

- 对象的创建
- 对象的使用
- 案例演示

讲解

对象的创建

- 创建对象的格式:
  - `类名 对象名 = new 类名();`
  - 类其实就是对象的数据类型,类是引用数据类型

- 例: Phone p1 = new Phone (); 创建了一个手机对象(Phone类的对象)

对象的使用

- 调用成员的格式:
  - 访问成员变量
    - 获取成员变量的值: 对象名.成员变量名
    - 给成员变量赋值: 对象名.成员变量名=值;
  - 访问成员方法
    - 对象名.成员方法();

案例演示

```
/*
    局部变量: 定义在方法中的变量
    成员变量: 定义在类中,方法外的变量

    定义一个类的格式:
        public class 类名{
            属性(状态信息)--成员变量
            格式: 数据类型 变量名;

            行为(功能)--成员方法
            格式: 去掉static的方法,后期可以加上
        }

    注意:
        暂时类中只能定义成员变量和定义成员方法,不能写输出语句,循环语句,条件控制语句...
 */
public class Phone {
    // 属性---成员变量
    String brand;// 品牌
    int price;// 价格

    // 行为(功能)--成员方法

    /**
     * 打电话的功能
     * @param phoneNum 电话号码
     */
    public void call(String phoneNum){
        System.out.println("正在给"+phoneNum+"打电话...");
    }

    /**
     * 发短信的功能
     * @param phoneNum 电话号码
     * @param message  短信内容
     */
    public void sendMessage(String phoneNum,String message){
        System.out.println("正在给"+phoneNum+"发送短信,短信的内容是:"+message);
    }

}
public class Test {// 用来执行程序的
```

```java
    // 程序的入口main方法
    public static void main(String[] args) {
        /*
            对象的创建:  类名 对象名 = new 类名();
            对象的使用:
                对象访问成员变量: 对象名.成员变量名
                    获取成员变量的值: 对象名.成员变量名
                    给成员变量赋值: 对象名.成员变量名 = 值;

                对象访问成员方法:  对象名.成员方法名(实参);


            结论: 成员变量有默认值
                整数类型: 默认值是0
                浮点类型: 默认值是0.0
                布尔类型: 默认值是false
                字符类型: 默认值是不可见字符  '\u0000'
                引用数据类型:默认值是 null
         */
        // 创建一个Phone对象
        Phone p1 = new Phone();

        // 使用p1对象访问成员变量
        // 获取成员变量的值:
        System.out.println(p1.brand);// null
        System.out.println(p1.price);// 0

        // 给成员变量赋值
        p1.brand = "华为";
        p1.price = 1999;

        System.out.println(p1.brand);// 华为
        System.out.println(p1.price);// 1999

        String str = p1.brand;
        System.out.println(str);// 华为

        System.out.println("=====================================");
        // 使用p1对象访问成员方法
        p1.call("10086");
        p1.sendMessage("10086","请问一下联通的客服电话号码是多少?");

    }

    // 对象的创建
    private static void method01() {
        // 创建Phone类的对象
        Phone p1 = new Phone();// p1对象
        Phone p2 = new Phone();// p2对象

        // 创建Student类的对象
        Student stu1 = new Student();// stu1对象
        Student stu2 = new Student();// stu2对象

        // 复盘之前的结论: 对象是根据类来创建的,类是对象的数据类型(引用数据类型)
        // 新的结论: 对象和对象之间是相互独立的
        // 新的结论: 一个类可以创建多个对象
    }
```

```
}
```

小结

```
    创建对象的格式:
        类名 对象名 = new 类名();
    使用对象:
        访问类的成员变量:
                获取成员变量的值: 对象名.成员变量名
                给成员变量赋值:    对象名.成员变量名 = 值;

        访问类的成员方法:
            成员方法无返回值:  对象名.成员方法(实参);
            成员方法有返回值:
                                对象名.成员方法(实参);  直接调用
                                数据类型 变量名 = 对象名.成员方法(实参); 赋值调用
                                System.out.println(对象名.成员方法(实参));输出调用

    类中的成员变量是有默认值的:
        整数类型      默认值          0
        小数类型      默认值          0.0
        字符类型      默认值          不可见字符 '\u0000'
        布尔类型      默认值          false
        引用数据类型  默认值          null
```

实操--7. 学生对象-练习

需求

- 首先定义一个学生类，然后定义一个学生测试类，在学生测试类中通过对象完成成员变量和成员方法的使用

分析

- 定义学生类
  - 成员变量：姓名，年龄...
  - 成员方法：学习，做作业...
- 测试类
  - 创建main方法,在main 方法中创建学生对象
  - 使用学生对象访问成员变量和访问成员方法

实现

```
public class Student {
    // 成员变量: 姓名, 年龄,...
    String name;// 姓名
    int age;// 年龄

    // 成员方法: 学习, 做作业,...
    /**
     * 学习的功能
     */
    public void study(){
        System.out.println("学生正在学习...");
    }
```

```
    /**
     * 做作业的功能
     */
    public void doHomeWork(){
        System.out.println("学生正在做作业...");
    }
}
public class Test {
    public static void main(String[] args) {
        // 创建一个Student对象  类名 对象名 = new 类名();
        Student stu1 = new Student();

        // 使用Student对象访问成员变量  对象名.成员变量名
        System.out.println(stu1.name+","+stu1.age);// null,0

        stu1.name = "张三";// 把张三赋值给stu1对象的name成员变量
        stu1.age = 18;// 把18赋值给stu1对象的age成员变量

        System.out.println(stu1.name+","+stu1.age);// 张三,18

        System.out.println("+============================");
        // 使用Student对象访问成员方法  对象名.成员方法名(实参);
        stu1.study();
        stu1.doHomeWork();
    }
}
```

小结

略

知识点--8. 单个对象内存图

目标

- 掌握单个对象的内存图

路径

- 查看程序案例
- 绘制内存图

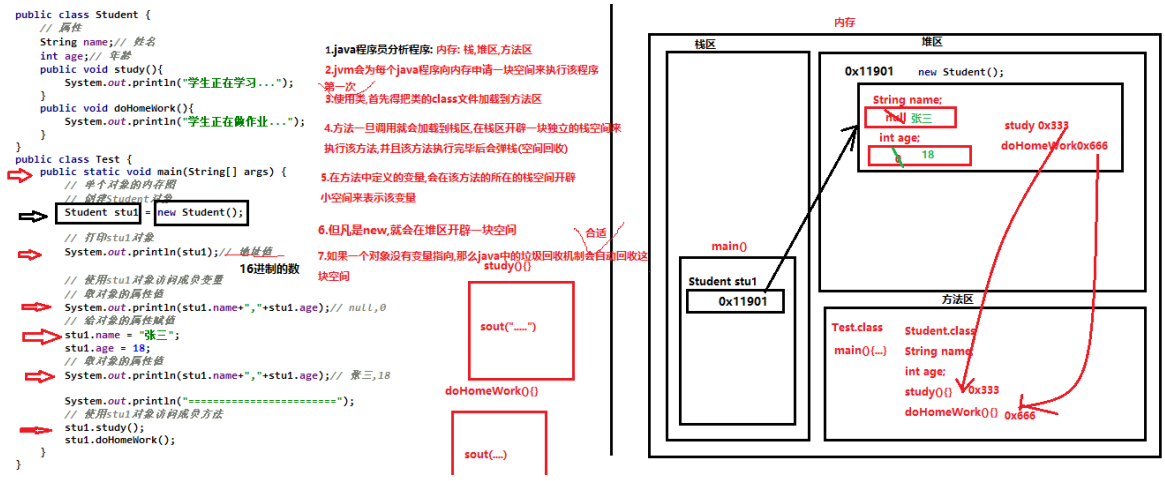
讲解

查看程序案例

代码

```
public class Student {
    // 属性
    String name;// 姓名
    int age;// 年龄

    // 行为
    /**
     * 学习的功能
```

```
     */
    public void study(){
        System.out.println("学生正在学习...");
    }

    /**
     * 做作业的功能
     */
    public void doHomeWork(){
        System.out.println("学生正在做作业...");
    }
}
public class Test {
    public static void main(String[] args) {
        // 单个对象的内存图
        // 创建Student对象
        Student stu1 = new Student();

        // 打印stu1对象
        System.out.println(stu1);// 地址值

        // 使用stu1对象访问成员变量
        // 取对象的属性值
        System.out.println(stu1.name+","+stu1.age);// null,0
        // 给对象的属性赋值
        stu1.name = "张三";
        stu1.age = 18;
        // 取对象的属性值
        System.out.println(stu1.name+","+stu1.age);// 张三,18

        System.out.println("========================");
        // 使用stu1对象访问成员方法
        stu1.study();
        stu1.doHomeWork();
    }
}
```



小结

- 只要创建对象,就会在堆区开辟一块空间
- 只要调用方法,就会在栈区开辟一块空间,用来执行该方法

知识点--9. 多个对象内存图【理解】

目标

- 掌握多个对象的内存图

路径

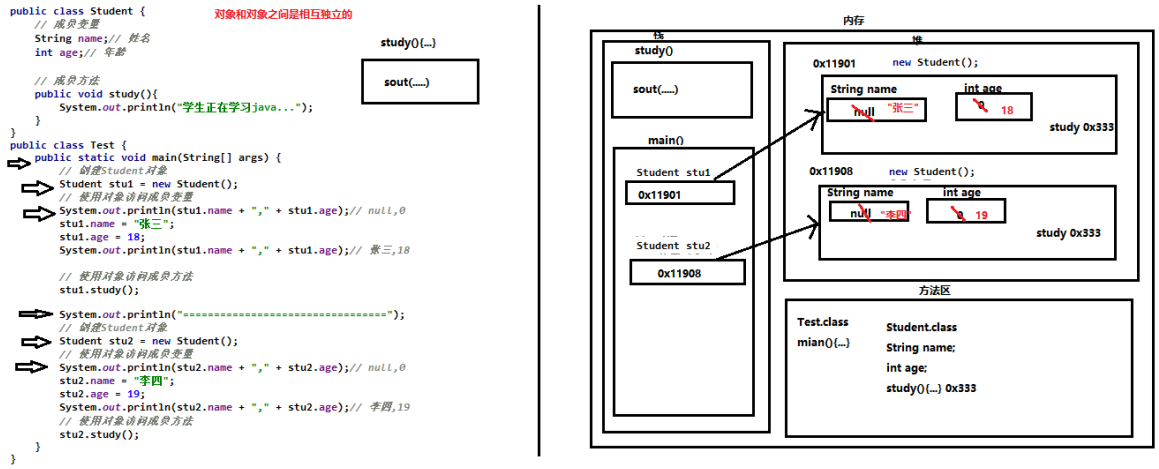
- 查看程序案例
- 绘制内存图

讲解

查看程序案例

```
public class Student {
    // 成员变量
    String name;// 姓名
    int age;// 年龄

    // 成员方法
    public void study(){
        System.out.println("学生正在学习java...");
    }
}
public class Test {
    public static void main(String[] args) {
        // 创建Student对象
        Student stu1 = new Student();

        // 使用对象访问成员变量
        System.out.println(stu1.name + "," + stu1.age);// null,0

        stu1.name = "张三";
        stu1.age = 18;

        System.out.println(stu1.name + "," + stu1.age);// 张三,18

        // 使用对象访问成员方法
        stu1.study();

        System.out.println("================================");
        // 创建Student对象
        Student stu2 = new Student();

        // 使用对象访问成员变量
        System.out.println(stu2.name + "," + stu2.age);// null,0

        stu2.name = "李四";
        stu2.age = 19;
        System.out.println(stu2.name + "," + stu2.age);// 李四,19

        // 使用对象访问成员方法
        stu2.study();

    }
}
```

绘制内存图



小结

- 多个对象在堆内存中，都有不同的内存划分，成员变量存储在各自的内存区域中，成员方法多个对象共用的一份
- 凡是new就会重新在堆区开辟一块新空间
- 对象和对象之间的关系是相互独立的

知识点--10. 多个变量指向相同对象内存图【理解】

目标

- 掌握多个变量指向相同对象内存图

路径

- 查看程序案例
- 绘制内存图

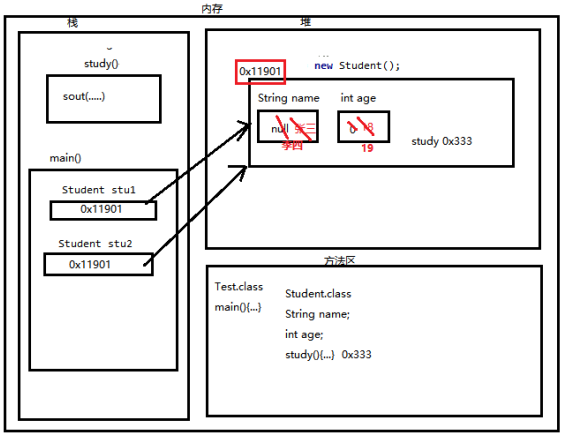
讲解

查看程序案例

```
public class Student {
    // 成员变量
    String name;// 姓名
    int age;// 年龄

    // 成员方法
    public void study(){
        System.out.println("学生正在学习java...");
    }
}

public class Test {
    public static void main(String[] args) {
        // 创建Student对象
        Student stu1 = new Student();

        // 使用对象访问成员变量
        System.out.println(stu1.name + "," + stu1.age);// null,0

        stu1.name = "张三";
```

```
        stu1.age = 18;

        System.out.println(stu1.name + "," + stu1.age);// 张三,18

        // 使用对象访问成员方法
        stu1.study();

        System.out.println("==================================");
        // 创建Student对象
        Student stu2 = stu1;

        // 使用对象访问成员变量
        System.out.println(stu2.name + "," + stu2.age);// 张三,18

        stu2.name = "李四";
        stu2.age = 19;
        System.out.println(stu2.name + "," + stu2.age);// 李四,19
        System.out.println(stu1.name + "," + stu1.age);// 李四,19

        // 使用对象访问成员方法
        stu2.study();

    }
}
```

绘制内存图



小结

- 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。

知识点--11. 成员变量和局部变量的区别【理解】

目标

- 理解成员变量和局部变量的区别

路径

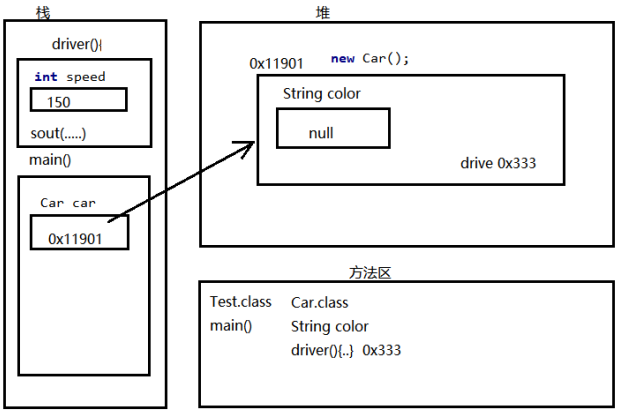
- 成员变量和局部变量的区别

讲解

- 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，随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）

```
public class Car {

    String color;// 颜色  成员变量   默认值null

    public void drive(){
        int speed = 150;// 速度 局部变量
        System.out.println(speed);
    }

}

public class Test {

    /*
        成员变量和局部变量的区别:
            定义的位置不同: 成员变量定义在类中方法外,局部变量定义在方法中
            在内存中的位置不同: 成员变量是在堆区,局部变量是在栈区
            生命周期不同:
                成员变量是随着对象的创建而存在,随着对象的销毁而销毁
                局部变量是随着方法的调用而存在,随着方法调用完毕而销毁
            默认值不同:
                成员变量有默认值
                局部变量没有默认值,不赋值不能直接使用
     */
    public static void main(String[] args) {
        Car car = new Car();
        System.out.println(car.color);
        car.drive();
    }
}
```

```
public class Car {

    String color;// 颜色  成员变量

    public void drive(){
        int speed = 150;// 速度 局部变量
        System.out.println(speed);
    }

}

public class Test {

    public static void main(String[] args) {
        Car car = new Car();

        car.drive();

    }
}
```



小结

略

知识点--2. 封装

知识点--2.1 private关键字

目标

- 理解private关键字的含义和使用格式

路径

- private的含义
- private的使用格式
- 案例

讲解

private的含义

- 概述: private是一个权限修饰符，代表最小权限。
- 特点:
  - 可以修饰成员变量和成员方法。
  - 被private修饰后的成员变量和成员方法，只在本类中才能访问。

private的使用格式

```
// private关键字修饰成员变量
private 数据类型 变量名 ;

// private关键字修饰成员方法
private 返回值类型 方法名(参数列表){
    代码
}
```

案例

```
public class Student{

    private String name;

    private int age;

    public void study(){
        System.out.println("努力学习Java...");
    }
    private void work(){
        System.out.println("努力敲代码...");
    }
}
```

小结

```
- private的含义: private是一个权限修饰符,表示最小的权限
    - private的使用: 修饰成员变量和成员方法
        修饰成员变量的格式:  private 数据类型 变量名;
        修饰成员方法的格式:  private 返回值类型 方法名(参数列表){...}

- 特点: 被private修饰的成员变量或者成员方法,只能在本类中访问
```

知识点--2.2 对属性封装的步骤

目标

- 理解对属性封装的步骤

路径

- 为什么要对属性进行封装
- 对属性封装的步骤

讲解

为什么要对属性进行封装

```
public class Student {
    // 类的属性 ---> 成员变量
    // 姓名(name)
    String name;

    // 年龄(age)
    int age;

    // 类的行为 ---> 成员方法
    public void show(){
        System.out.println("姓名:"+name+",年龄:"+age);
    }
}

public class Demo1Student {
    public static void main(String[] args) {
        // 创建Student对象
        Student stu = new Student();
```

```
        // 给stu对象的属性赋值
        stu.name = "冰冰";
        // stu.age = 18;
        stu.age = -18;

        // 调用show()方法
        stu.show();// 姓名:冰冰,年龄:-18

    }
}
```

- 通过对象名直接访问成员变量的方式来对属性赋值,会存在数据安全隐患,应该怎么解决呢?
- 解决方式: 不让外界直接访问成员变量(也就是要对属性进行封装)

对属性封装的步骤

1. 使用private修饰成员变量

```
public class Student {
    // 类的属性 ---> 成员变量
    // 姓名(name)
    private String name;

    // 年龄(age)
    private int age;

    // 类的行为 ---> 成员方法
    public void show(){
        System.out.println("姓名:"+name+",年龄:"+age);
    }
}
```

2. 对需要访问的成员变量,提供对应的`getXxx`方法(获取属性的值) 、 `setXxx` 方法(给属性赋值)。

小结

- 略

知识点--2.3 set和get方法

目标

- 掌握set和get方法的书写

路径

- set和get方法的介绍
- set和get方法的书写

讲解

set和get方法的介绍

- 由于属性使用了private关键字修饰,在其他类中无法直接访问,所以得提供公共的访问方法,我们把这张方法叫做set和get方法
    - get方法: 提供“get变量名()”方法，用于获取成员变量的值，方法用public修饰

- set方法: 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

set和get方法的书写

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

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

  public String getName() {
    return name;
  }

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

  public int getAge() {
    return age;
  }
}
```

小结

```
set方法的书写规律:
    1.set方法一定是一个公共的方法(public)
        2.set方法一定没有返回值(void)
        3.set方法的方法名一定是set+属性名,并且属性名首字母大写
        4.set方法一定有参数
        5.set方法一定会给属性赋值

        get方法的书写规律:
    1.get方法一定是一个公共的方法(public)
        2.get方法一定有返回值,并且返回值类型与获取的属性类型一致
        3.get方法的方法名一定是get+属性名,并且属性名首字母大写
        4.get方法一定没有参数
        5.get方法一定会返回属性的值
```

知识点--2.4 this关键字

目标

- 理解this关键字的含义和使用

路径

- 问题
- this关键字的含义和使用

讲解

问题

我们发现 `setXxx` 方法中的形参名字并不符合见名知意的规定，那么如果修改与成员变量名一致，是否就见名知意了呢？代码如下：

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

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

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

经过修改和测试，我们发现新的问题，成员变量赋值失败了。也就是说，在修改了`setXxx()` 的形参变量名后，方法并没有给成员变量赋值！这是由于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，我们只能使用this关键字，来解决这个重名问题。

this的含义和使用

- this含义: this代表当前调用方法的引用，哪个对象调用this所在的方法，this就代表哪一个对象
- this关键字其主要作用是区分同名的局部变量和成员变量
  - 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 {
    private String name;
    private int age;

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

    public String getName() {
      return name;
    }

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

    public int getAge() {
      return age;
    }
  }
  ```

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

小结

```
this关键字:
    1.作用: 用来区分同名的成员变量和局部变量
    2.格式: this.成员变量名
    3.this含义:代表当前对象
      当前对象: 谁调用this所在的方法,谁就是当前对象
```

知识点--2.5 this内存原理

目标

- 加深对this的理解

路径

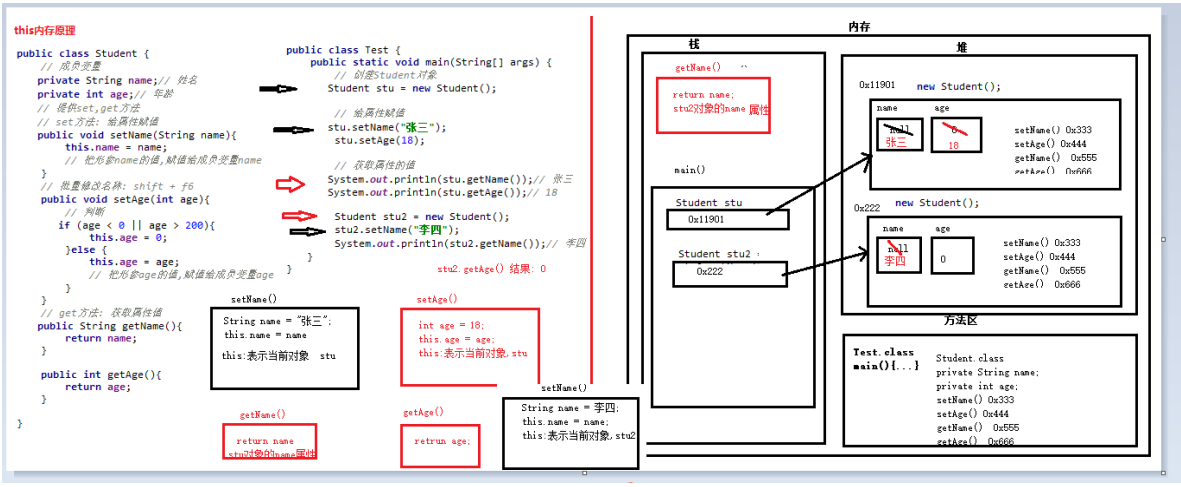
- 查看案例代码
- 绘制内存图

讲解

代码

```
public class Student {
    // 属性
    private String name;// 姓名
    private int age;// 年龄

    // 提供set方法,为了能够让外界给属性赋值
    public void setName(String name){
        this.name = name;
    }

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

    // 提供get方法,为了能够让外界可以获取属性的值
    public String getName(){
        return name;
    }

    public int getAge(){
        return age;
    }

    // 成员方法
    public void show(){
        System.out.println("姓名:"+name+",age:"+age);
    }
}
```

```java
public class Demo1Student {
    public static void main(String[] args) {
        // 创建Student对象
        Student stu = new Student();

        // 给Student对象属性赋值
        stu.setName("冰冰");
        stu.setAge(18);

        // 调用show方法
        stu.show();
    }
}
```



小结

略

知识点--2.6 封装概述

目标:

- 理解封装的概念

路径:

- 封装概述

讲解:

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

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
- 例如:成员变量使用private修饰，提供对应的getXxx()/setXxx()方法

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

小结

略

知识点--3. 构造方法

知识点--3.1 构造方法概述

目标

- 能够理解构造方法的作用和能够定义构造方法

路径

- 构造方法的概述
- 构造方法的定义

讲解

构造方法的概述

- 构造方法是一种特殊的方法,主要是完成对象的创建和对象数据的初始化

构造方法的定义

- 格式

```
// 空参构造方法
修饰符 类名(){

}

// 有参构造方法
修饰符 类名(参数列表){
    // 方法体
}
```

- 特点:
  - 构造方法的写法上，方法名与它所在的类名相同
  - 构造方法没有返回值，所以不需要返回值类型，甚至不需要void
- 示例代码:

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

    // 构造方法:
    // 空参构造方法
    // 创建对象,属性为默认值
    public Student(){//空参构造
```

```
        System.out.println("空参构造方法执行了...");
    }

    // 有参构造方法
    public Student(String name,int age){// 满参构造方法
        // 给属性赋值
        this.name = name;
        this.age = age;
    }

    public Student(String name){// 有参构造方法
        // 给属性赋值
        this.name = name;
    }

    // 方法
    public void show(){
        System.out.println("姓名:"+name+",年龄:"+age);
    }

}

/*
    测试类
 */
public class Demo1Student {
    public static void main(String[] args) {
        // 使用构造方法
        // 创建Student对象:  类名 对象名 = new 类名();
        // 调用空参构造方法创建对象,对象的属性为默认值
        Student stu1 = new Student();// 等于号的右边其实就是调用空参构造方法
        stu1.show();// 姓名:null,年龄:0

        // 调用满参构造方法创建对象,对象的属性会被赋值
        Student stu2 = new Student("冰冰",18);
        stu2.show();// 姓名:冰冰,年龄:18

        // 调用有参构造方法创建对象,对象的部分属性会被赋值
        Student stu3 = new Student("小泽老师");
        stu3.show();// // 姓名:小泽老师,年龄:0

    }
}
```

小结

构造方法的概述

- 构造方法是一种特殊的方法,主要是完成对象的创建和对象数据的初始化

构造方法的定义

- 格式:

  空参构造方法

  ```
  修饰符 类名(){

  }
  ```

  有参构造方法

```
                修饰符 类名(参数){
                    方法体(给属性赋值)
                }
        - 特点:
            1.构造方法的方法名和类名一致
            2.构造没有返回值,连void都没有
            3.构造方法可以重载
  调用构造方法: 通过new来调用
```

知识点--3.2 构造方法的注意事项

目标

- 理解构造方法的注意事项,并以后开发中知道如何避免

路径

- 构造方法的注意事项

讲解

- 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
- 构造方法可以重载，既可以定义参数，也可以不定义参数。
- 示例代码

```
/*
    学生类
 */
class Student {
    private String name;
    private int age;

    public Student() {}

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

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

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

    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 s1 = new Student();
        s1.show();

        //public Student(String name)
        Student s2 = new Student("林青霞");
        s2.show();

        //public Student(int age)
        Student s3 = new Student(30);
        s3.show();

        //public Student(String name,int age)
        Student s4 = new Student("林青霞",30);
        s4.show();
    }
}
```

小结

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

- 构造方法可以重载，既可以定义参数，也可以不定义参数。
- 定义构造方法的时候,不要写返回值,连void都不能有
- 定义构造方法的时候,构造方法名和类名一定要一致

知识点--3.3 标准类制作

目标

- 掌握标准类的制作

路径

- 标准类的组成
- 案例演示

讲解

标准类的组成

`JavaBean` 是 Java语言编写类的一种标准规范。符合 `JavaBean` 的类，要求类必须是公共的，属性使用private修饰,并且具有无参数的构造方法，提供用来操作成员变量的 `set` 和 `get` 方法。

```
public class ClassName{
  //成员变量  private
  //构造方法
  //无参构造方法【必须】
  //满参构造方法【建议】
  //getXxx()
  //setXxx()
  //成员方法
}
```

案例演示

- 需求：定义标准学生类，要求分别使用空参和有参构造方法创建对象，空参创建的对象通过setXxx赋值，有参创建的对象直接赋值，并通过show方法展示数据。
- 示例代码：

```
public class Student {
    // 成员变量 private
    public  String name;
    public  int age;

    // 生成构造方法和set\get方法的快捷键：  alt+insert

    // 空参构造方法
    public Student() {
    }

    // 满参构造方法
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }


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

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

    // get方法
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    // 成员方法 功能方法
    public void show(){
        System.out.println(name+","+age);
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        // 需求:要求分别使用空参和有参构造方法创建对象，空参创建的对象通过setXxx赋值，有参创建的对象直接赋值，并通过show方法展示数据。
        // 使用空参构造方法创建对象
        Student stu1 = new Student();

        // 使用有参构造方法创建对象
        Student stu2 = new Student("张三",18);

        // 使用stu1和stu2调用show方法打印各自属性的值
        stu1.show();// null,0
        stu2.show();// 张三,18

        // 空参创建的对象只能通过setXxx赋值
        stu1.setName("李四");
        stu1.setAge(19);
        stu1.show();// 李四,19

    }
}
```

小结

略

知识点--4. API

目标

- 了解API的概念和使用步骤

路径

- API的概念
- API的使用步骤
- 演示API的使用

讲解

API的概念

- 什么是API

  API (Application Programming Interface)： 应用程序编程接口。Java API是一本程序员的`字典`，是JDK中提供给我们使用的类的**说明文档**。这些类将底层的代码实现封装了起来，我们不需要关心这些类是如何实现的，只需要学习这些类如何使用即可。所以我们可以通过查询API的方式，来学习Java提供的类，并得知如何使用它们。

  - API其实就是jdk中核心类库的说明文档
  - 对于jdk中的核心类库只需要知道如何使用,无须关心他是如何实现的

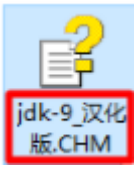
API的使用步骤

1. 打开API帮助文档。
2. 点击显示，找到索引，看到输入框。
3. 你要找谁？在输入框里输入，然后回车。
4. 看包。java.lang下的类不需要导包，其他需要。

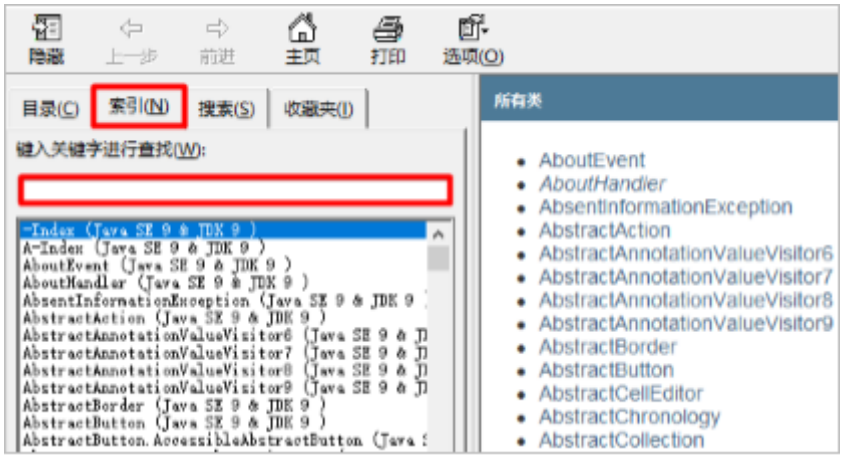
5. 看类的解释和说明。
6. 看构造方法。
7. 看成员方法。

演示API的使用

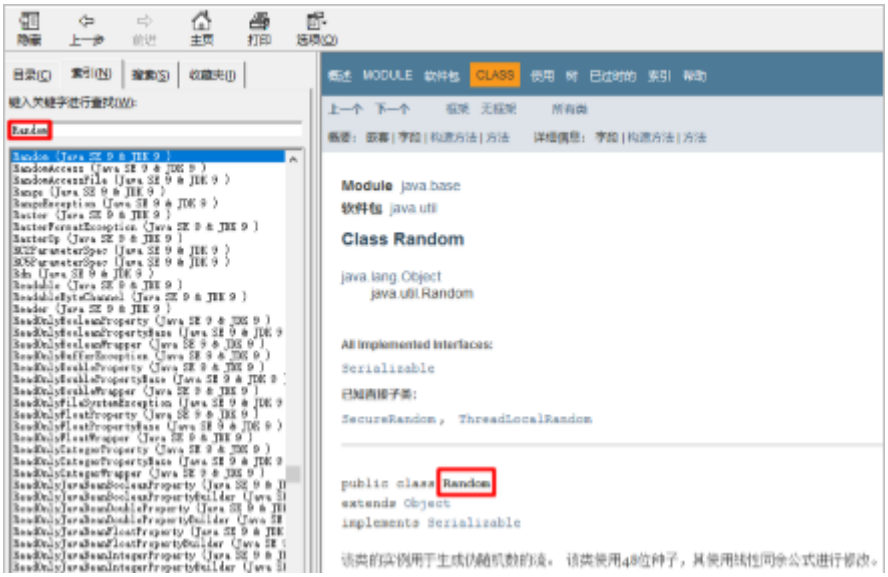
- 打开帮助文档



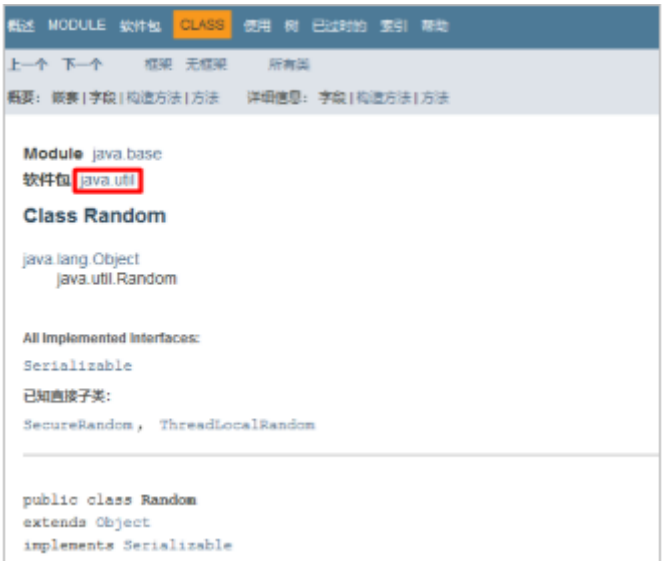
- 找到索引选项卡中的输入框



- 在输入框中输入Random



- 看类在哪个包下



- 看类的描述

- 看构造方法

- 看成员方法


```
public class Test {
```

```
    public static void main(String[] args) {
        /*
            API使用步骤:
                1.打开API
                2.点击"显示"
                3.点击"索引"
                4.在输入框中输入要查找的类,回车
                5.查看包
                6.查看类的解释说明
                7.查看构造方法
                8.查看成员方法

            使用系统提供的类导包:  在java.lang包下的类不需要导包,其他都需要导包

            举例: Scanner
                1.查看包  java.util  使用时需要导包
                2.查看类的解释说明:  一个简单的文本扫描器,可以扫描基本数据类型和字符串类型的
数据
                3.查看构造方法:
                    Scanner(InputStream source)
                    System.in: 表示键盘录入
                4.查看成员方法: 功能
                     boolean nextBoolean()
                     byte nextByte()
                     double nextDouble()
                     float nextFloat()
                     int nextInt()
                     long nextLong()
                     short nextShort()
                     String nextLine() 可以获取一行字符串 包括空格,制表符,回车...
                     String next()  只能获取连续的一个字符串,不能获取含有空格,制表符,回
车...

         */
        Scanner sc = new Scanner(System.in);
        // int i = sc.nextInt();
        // System.out.println(i);

        // long l = sc.nextLong();
        // System.out.println(l);

        // double d = sc.nextDouble();
        // System.out.println(d);

        //String str1 = sc.next();
        //System.out.println(str1);

        String str2 = sc.nextLine();
        System.out.println(str2);
    }
}
```

小结

略

总结