day1-面向对象入门(类和对象)

今日内容

- 类和对象
  - 面向对象编程思想
  - 类的概述和定义----->掌握
  - 对象的创建和使用--->重点重点 掌握
  - 对象的内存图------>理解
  - 成员变量和局部变量
- 封装
  - private关键字使用
  - set\get方法
  - this关键字
- 构造方法
  - 定义和使用
  - JavaBean标准类制作
- 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--->切菜---->炒菜--->吃
  - 面向对象: 找个饭店-->20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]

```
public class Test {
    public static void main(String[] args) {
        /*
            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。
         */
        // 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]
        // 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};

        // 面向过程:
        // 2.循环遍历数组
        for (int i = 0; i < arr.length; i++) {
            // 3.在循环中,获取遍历出来的元素
            int e = arr[i];
            // 4.判断该元素:
            if (i == 0) {
                // 4.1 如果该元素是第一个元素,打印格式: [ + 元素 + 逗号空格   不换行
                System.out.print("[" + e + ", ");
            } else if (i == arr.length - 1) {
                // 4.2 如果该元素是最后一个元素,打印格式: 元素 + ]
                System.out.println(e + "]");
            } else {
                // 4.3 如果该元素是中间元素,打印格式为: 元素 + 逗号空格    不换行
                System.out.print(e + ", ");
            }
        }

        System.out.println("==================================");

        // 面向对象:
        // jdk的api中有一个Arrays类toString()方法,可以帮助我们按照这种格式打印数组中的所有元素
        System.out.println(Arrays.toString(arr));
    }
```

```
}
```

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:注重的是过程,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。
- 手机类---描述手机
- 人类---- 描述人

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...。
- 人类:
    - 属性: 姓名,年龄,性别....
    - 行为:吃饭,睡觉,.....

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----共同拥有的
    - 行为----共同拥有的

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

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

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)

```
public class Phone {
    //属性(成员变量): 数据类型 变量名;
    /**
     * 品牌
     */
    String brand;
    /**
     * 价格
     */
    double price;

    //行为(成员方法): 去掉static

    /**
     * 打电话的功能
     * @param phoneNum 电话号码
     */
    public void call(String phoneNum){
        System.out.println("正则给"+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,短信内容是:"+message);
    }
}
```

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

```
package com.itheima.demo3_对象的创建和使用;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 10:27
 */
public class Phone {
    //属性(成员变量): 数据类型 变量名;
    /**
     * 品牌
```

```
     */
    String brand;
    /**
     * 价格
     */
    double price;

    //行为(成员方法): 去掉static

    /**
     * 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,短信内容是:"+message);
    }

    // 为了演示有返回值的方法调用
    public int show(String str){
        System.out.println("有参数有返回值的方法:"+str);
        return 100;
    }
}

public class Test {
    public static void main(String[] args) {
        /*
            对象的创建和使用:
                对象的创建:
                    对象创建格式: 类名 对象名 = new 类名();
                    结论: 类其实也是一种数据类型,是引用数据类型
                对象的使用:
                    访问成员变量:
                        给成员变量赋值:  对象名.成员变量名 = 值;
                        获取成员变量的值: 对象名.成员变量名

                    访问成员方法:
                        调用方法: 有返回值的方法,无返回值的方法
                        无返回值的方法:
                                直接调用: 对象名.方法名(实参);
                        有返回值的方法:
                                直接调用: 对象名.方法名(实参);
                                赋值调用: 数据类型 变量名 = 对象名.方法名(实参);
                                输出调用: System.out.println(对象名.方法名(实参));
         */
        // 创建Phone类的对象
        Phone p1 = new Phone();
        // 给p1对象的brand成员变量赋值
        p1.brand = "华为";
```

```
        // 给p1对象的price成员变量赋值
        p1.price = 999.8;

        // 获取p1对象的brand成员变量的值
        System.out.println(p1.brand);
        // 获取p1对象的price成员变量的值
        System.out.println(p1.price);

        // 无返回值的成员方法
        // 使用p1对象调用call方法
        p1.call("10086");
        // 使用p1对象调用sendMessage方法
        p1.sendMessage("10086","请问一下联通的客服电话号码是多少?");

        System.out.println("==============================");
        // 有返回值的方法
        // 直接调用
        p1.show("itheima");

        // 赋值调用
        int res = p1.show("itcast");// 100
        System.out.println("res:"+res);// 100

        // 输出调用
       System.out.println(p1.show("java"));// 100

        /*
        之前访问变量
        int num;
        num = 10;
        System.out.println(num);*/
        /*
        之前访问访问
            方法分类: 无参数无返回值,有参数无返回值,有参数有返回值,无参数有返回值
            调用方法: 有返回值的方法,无返回值的方法
                无返回值的方法:
                        直接调用: 方法名(实参);
                有返回值的方法:
                        直接调用: 方法名(实参);
                        赋值调用: 数据类型 变量名 = 方法名(实参);
                        输出调用: System.out.println(方法名(实参));
        */
    }
}
```

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
```

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

```java
package com.itheima.demo4_类和对象的练习;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 11:23
 */
public class Student {
    // 成员变量: 属性
    /**
     * 姓名
     */
    String name;
    /**
     * 年龄
     */
    int age;

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

```
    }

    /**
     * 做作业的功能
     */
    public void doHomeWork(){
        System.out.println("学生正在做作业敲代码...");
    }
}

package com.itheima.demo4_类和对象的练习;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 11:22
 */
public class Test {
    public static void main(String[] args) {
        // 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
        // 创建学生对象
        Student stu = new Student();

        // 访问成员变量
        stu.name = "冰冰";
        stu.age = 18;
        System.out.println(stu.name+","+stu.age);// 冰冰,18

        // 访问成员方法
        stu.study();
        stu.doHomeWork();

    }
}
```

小结

略

知识点--成员变量默认值

```
public class Student {
    /**
     * 姓名
     */
    String name;
    /**
     * 年龄
     */
    int age;
    /**
     * 分数
     */
    double score;
    char c;
}
```

```java
package com.itheima.demo5_成员变量默认值;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 11:47
 */
public class Test {
    public static void main(String[] args) {
        // 变量的要使用一定要赋值
        //int num;// 局部变量:定义在方法中的变量
        //System.out.println(num);// 编译报错,局部变量没有默认值

        /*
            成员变量的默认值:
                整数类型: 默认值是0
                小数类型: 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'
                引用类型: 默认值是null
         */
        // 创建Student对象
        Student stu = new Student();
        // 访问成员变量
        System.out.println(stu.name);// null
        System.out.println(stu.age);// 0
        System.out.println(stu.score);// 0.0
        System.out.println("="+stu.c+"=");
    }
}
```

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

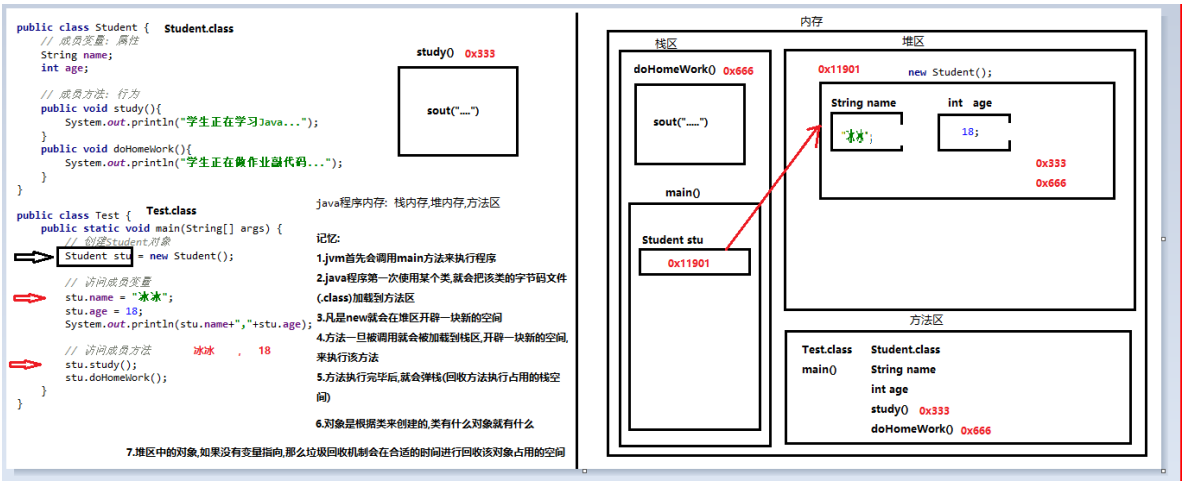
查看程序案例

代码

```java
package com.itheima.demo6_单个对象内存图;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 11:23
 */
public class Student {
    // 成员变量: 属性
```

```
    /**
     * 姓名
     */
    String name;
    /**
     * 年龄
     */
    int age;

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

    /**
     * 做作业的功能
     */
    public void doHomeWork(){
        System.out.println("学生正在做作业敲代码...");
    }
}
```

```
package com.itheima.demo6_单个对象内存图;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 11:54
 */
public class Test {
    public static void main(String[] args) {
        // 创建Student对象
        Student stu = new Student();
        System.out.println(stu);// 十六进制数地址值

        // 访问成员变量
        stu.name = "冰冰";
        stu.age = 18;
        System.out.println(stu.name+","+stu.age);

        // 访问成员方法
        stu.study();
        stu.doHomeWork();
    }
}
```



小结

- 只要创建对象,就会在堆区开辟一块空间(凡是new就会在堆区开辟一块新的空间)
- 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

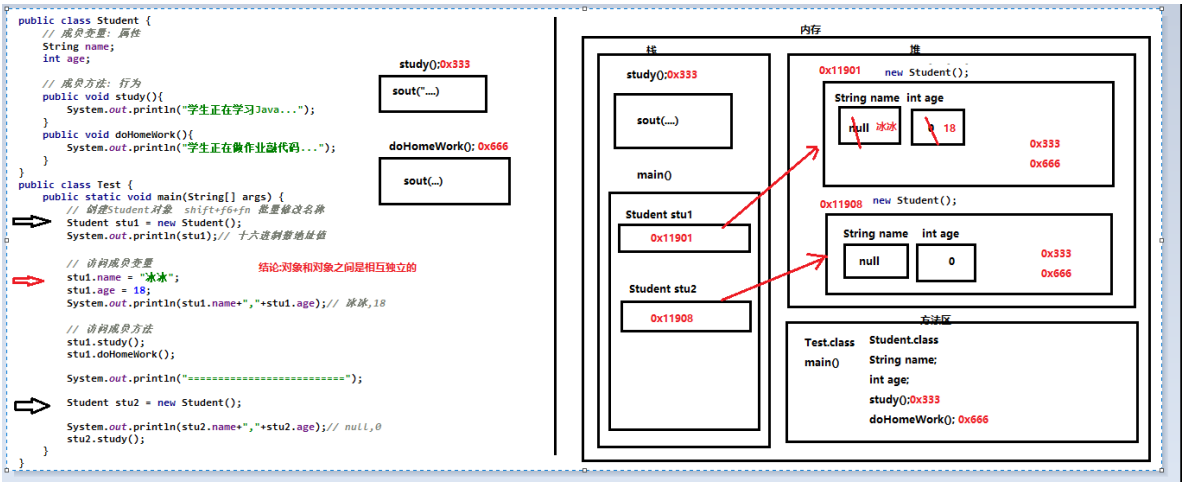
讲解

查看程序案例

```
package com.itheima.demo7_多个对象内存图;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 11:23
 */
public class Student {
    // 成员变量: 属性
    /**
     * 姓名
     */
    String name;
    /**
     * 年龄
     */
    int age;

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

```
    /**
     * 做作业的功能
     */
    public void doHomeWork(){
        System.out.println("学生正在做作业敲代码...");
    }
}
package com.itheima.demo7_多个对象内存图;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 11:54
 */
public class Test {
    public static void main(String[] args) {
        // 创建Student对象  shift+f6+fn 批量修改名称
        Student stu1 = new Student();
        System.out.println(stu1);// 十六进制数地址值

        // 访问成员变量
        stu1.name = "冰冰";
        stu1.age = 18;
        System.out.println(stu1.name+","+stu1.age);// 冰冰,18

        // 访问成员方法
        stu1.study();
        stu1.doHomeWork();

        System.out.println("==========================");

        Student stu2 = new Student();

        System.out.println(stu2.name+","+stu2.age);// null,0
        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对象的内存区域中，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

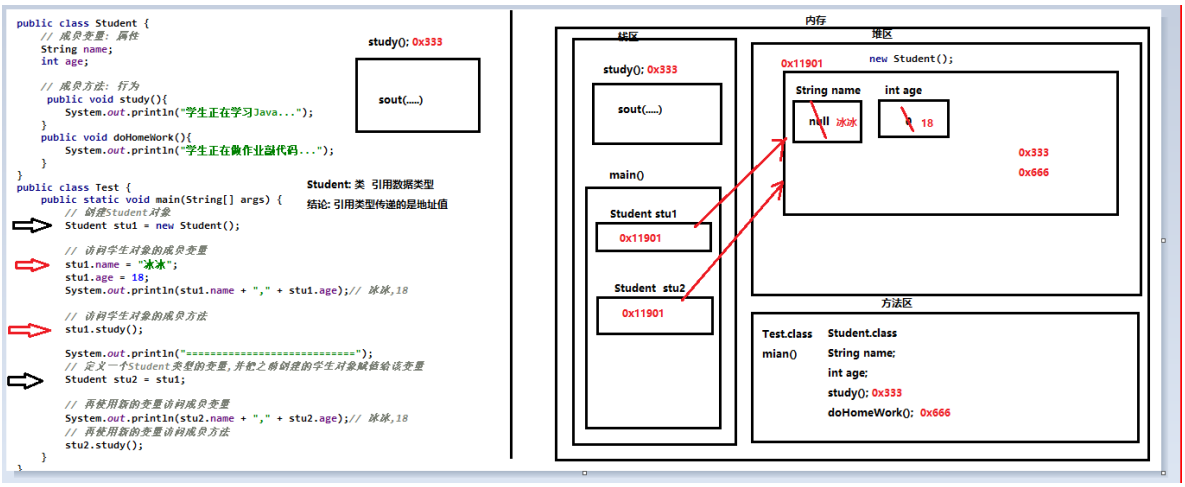
讲解

查看程序案例

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

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

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

        // 访问学生对象的成员变量
        stu1.name = "冰冰";
        stu1.age = 18;
        System.out.println(stu1.name + "," + stu1.age);// 冰冰,18

        // 访问学生对象的成员方法
```

```
        stu1.study();

        System.out.println("============================");
        // 定义一个Student类型的变量,并把之前创建的学生对象赋值给该变量
        Student stu2 = stu1;

        // 再使用新的变量访问成员变量
        System.out.println(stu2.name + "," + stu2.age);// 冰冰,18
        // 再使用新的变量访问成员方法
        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。
- **引用类型传递的是地址值**

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

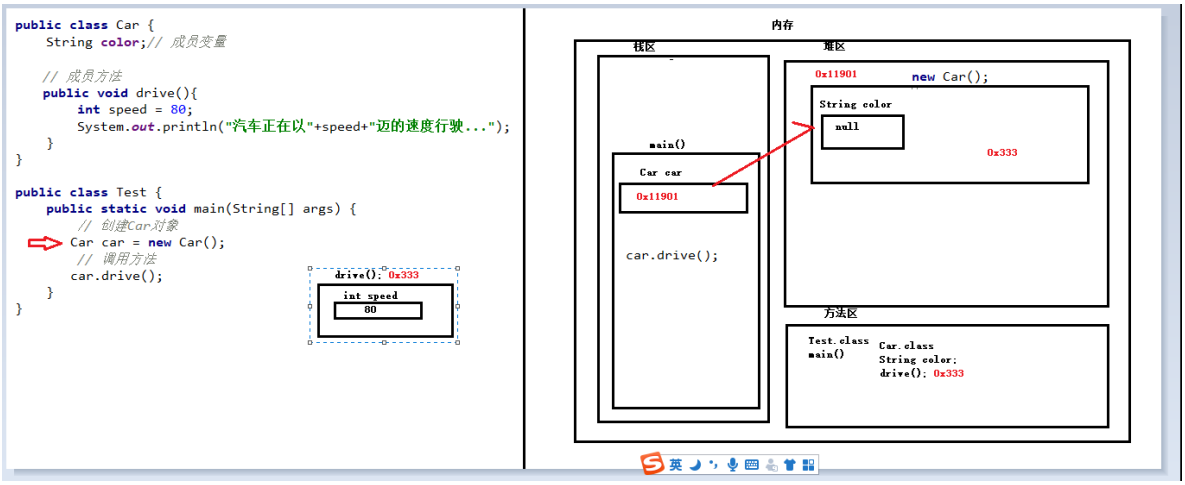
路径

- 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;// 成员变量

    // 成员方法
    public void drive(){
        int speed = 80;
        System.out.println("汽车正在以"+speed+"迈的速度行驶...");
    }
}


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) {
         // 创建Car对象
        Car car = new Car();
        // 调用方法
        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 {
    /**
     * 姓名
```

```
     */
    private String name;
    /**
     * 年龄
     */
    private int age;


    private void study(){
        System.out.println("正在学习java");
    }

    public void show(){
        // 只能在本类中访问
        System.out.println(name+","+age);
    }
}

package com.itheima.demo10_private关键字;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 15:03
 */
public class Test {
    public static void main(String[] args) {
        /*
            private关键字:
                概述:是一个权限修饰符,最小的权限
                特点:
                    1.private可以修饰成员变量和成员方法
                    2.被private修饰后的成员变量和成员方法，只在本类中才能访问。
                使用:
                    修饰成员变量格式:  private 数据类型 变量名;
                    修饰成员方法格式:  private 返回值类型 方法名(形参列表){方法体}
         */
        // 创建Student类对象
        Student stu1 = new Student();

        // 直接访问stu1的成员变量
        //stu1.name = "冰冰";// 编译报错,因为没有访问权限
        //stu1.age = 18;// 编译报错,因为没有访问权限

        // 直接访问stu1的成员方法
        //stu1.study();// 编译报错,因为没有访问权限

    }
}
```

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 为什么要对属性进行封装

目标

- 理解对属性封装

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 {
    /**
     * 姓名
     */
    String name;
    /**
     * 年龄
     */
     int age;
}
package com.itheima.demo11_为什么要对属性进行封装;

/**
 * @Author：pengzhilin
 * @Date: 2020/9/5 15:16
 */
public class Test {
    public static void main(String[] args) {
        /*
            为什么要对属性进行封装:
                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\隐藏)
            对成员变量隐藏的步骤:
                1.使用private关键字修饰成员变量
                2.提供公共的访问方法:
                    给成员变量赋值的公共方法(set方法)
                    获取成员变量值的公共方法(get方法)
         */
        // 创建Student对象
        Student stu1 = new Student();

        // 访问成员变量
        stu1.name = "冰冰";
        // 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?
```

```
        stu1.age = -18;
        System.out.println(stu1.name + "," + stu1.age);// 冰冰,-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
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

- 使用private修饰成员变量,来隐藏成员变量不被外界直接访问
- 为private修饰的成员变量,提供公共的访问方式(set\get方法)

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

```
package com.itheima.demo12_set和get方法;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 15:16
 */
public class Student {
    /**
     * 姓名
     */
    private String name;
    /**
     * 年龄
     */
     private int age;
```

```
     // 提供给成员变量赋值的方法-set方法
    public void setName(String s){
        name = s;
    }

    public void setAge(int a){
        if (a < 0 || a > 150){
            age = -1;
            System.out.println("您的数据不合法!");
        }else{
            age = a;
        }
    }
     // 提供获取成员变量值的方法-get方法
    public String getName(){
        return name;
    }

    public int getAge(){
        return age;
    }
}

package com.itheima.demo12_set和get方法;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 15:36
 */
public class Test {
    public static void main(String[] args) {
        /*
            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: 使用private修饰,并提供公共的访问方法
         */
        // 创建Student对象
        Student stu1 = new Student();

        // 访问成员变量
        // 隐藏属性后的方式
        stu1.setName("冰冰");
        stu1.setAge(-18);
        System.out.println(stu1.getName()+","+stu1.getAge());// 冰冰,-1

        // 没有隐藏属性之前的方式
        //stu1.name = "冰冰";
        //stu1.age = -18;
        //System.out.println(stu1.name + "," + stu1.age);// 冰冰,-18
    }
}
```

小结

- 对成员变量的封装:
    - 使用private修饰成员变量

- 提供公共的访问方法(set赋值方法,get取值方法)

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：

  ```
  package com.itheima.demo13_this关键字;

  /**
   * @Author: pengzhilin
   * @Date: 2020/9/5 15:16
  ```

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

     // 提供给成员变量赋值的方法-set方法
    public void setName(String name){
        this.name = name;
    }

    public void setAge(int age){
        if (age < 0 || age > 150){
            this.age = -1;
            System.out.println("您的数据不合法!");
        }else{
            this.age = age;
        }
    }
     // 提供获取成员变量值的方法-get方法
    public String getName(){
        return name;
    }

    public int getAge(){
        return age;
    }
}

package com.itheima.demo13_this关键字;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 15:36
 */
public class Test {
    public static void main(String[] args) {
        /*
            问题1:set方法的形参名不能起到知名达意(不符合标识符命名规范)
            解决1:把形参名修改成符合命名规范
            问题2:set方法的形参名改为符合命名规范后,发现set方法无法给成员变量赋值
            解决2:使用this关键字来区别同名的成员变量和局部变量
                格式: this.成员变量名
                this表示谁: 哪个对象调用this所在的方法,this就表示哪个对象

           结论:
                1.如果成员方法中有与成员变量同名的局部变量,那么就需要使用this关键字来区
分
                2.如果成员方法中没有与成员变量同名的局部变量,那么就不需要使用this关键字
来区分(直接使用成员变量即可)
         */
        // 创建Student对象
        Student stu1 = new Student();
```

```
        // 访问成员变量
        // 隐藏属性后的方式
        stu1.setName("冰冰");
        stu1.setAge(-18);
        System.out.println(stu1.getName()+","+stu1.getAge());// 冰冰,-1

        Student stu2 = new Student();
        stu2.setName("空空");

    }
}
```

> 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
      结论:哪个对象调用this所在的方法,this就表示哪个对象
```

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

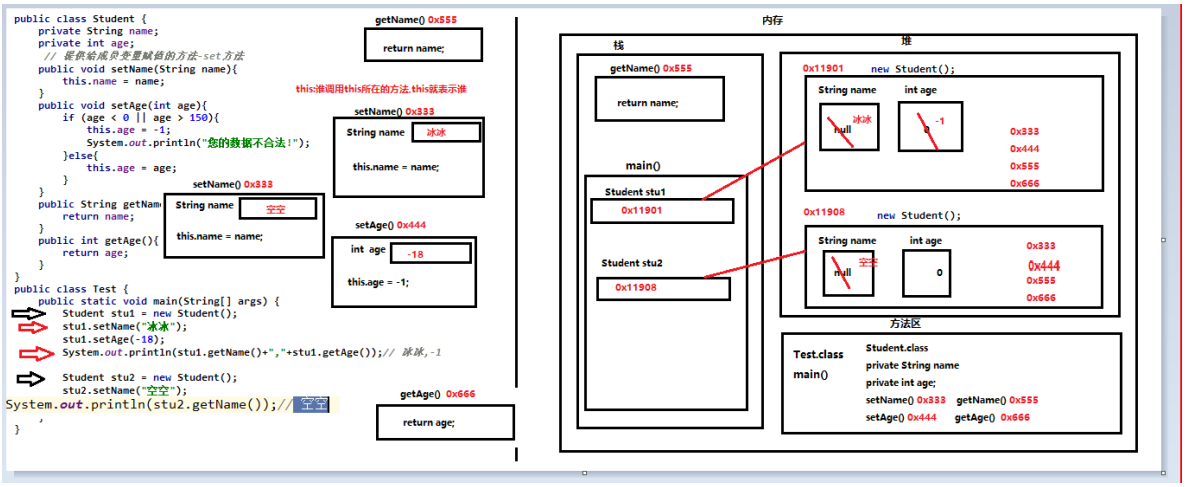
讲解

代码

```
package com.itheima.demo13_this关键字;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 15:36
 */
public class Test {
    public static void main(String[] args) {
        /*
            问题1:set方法的形参名不能起到知名达意(不符合标识符命名规范)
            解决1:把形参名修改成符合命名规范
            问题2:set方法的形参名改为符合命名规范后,发现set方法无法给成员变量赋值
            解决2:使用this关键字来区别同名的成员变量和局部变量
                格式: this.成员变量名
                this表示谁: 哪个对象调用this所在的方法,this就表示哪个对象
```

```
            结论:
                1.如果成员方法中有与成员变量同名的局部变量,那么就需要使用this关键字来区分
                2.如果成员方法中没有与成员变量同名的局部变量,那么就不需要使用this关键字来区分(直接使用成员变量即可)
             */
            // 创建Student对象
            Student stu1 = new Student();

            // 访问成员变量
            // 隐藏属性后的方式
            stu1.setName("冰冰");
            stu1.setAge(-18);
            System.out.println(stu1.getName()+","+stu1.getAge());// 冰冰,-1

            Student stu2 = new Student();
            stu2.setName("空空");
            System.out.println(stu2.getName()+","+stu2.getAge());// 空空,0

        }
}


package com.itheima.demo13_this关键字;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 15:16
 */
public class Student {
    /**
     * 姓名
     */
    private String name;
    /**
     * 年龄
     */
     private int age;

     // 提供给成员变量赋值的方法-set方法
    public void setName(String name){
        this.name = name;
    }

    public void setAge(int age){
        if (age < 0 || age > 150){
            this.age = -1;
            System.out.println("您的数据不合法!");
        }else{
            this.age = age;
        }
    }
     // 提供获取成员变量值的方法-get方法
    public String getName(){
        return name;
    }

    public int getAge(){
```

```
            return age;
        }
    }
```



小结

- this表示当前对象(哪个对象调用this所在的方法,this就表示哪个对象)

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:

```
package com.itheima.demo14_构造方法的使用;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 16:48
 */
public class Student {
    /**
     * 姓名
     */
    private String name;
    /**
     * 年龄
     */
    private int age;
```

```
    // 构造方法
    public Student(){
        System.out.println("空参方法");
    }

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

    public String getName(){
        return name;
    }

    public int getAge(){
        return age;
    }
}
package com.itheima.demo14_构造方法的使用;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 16:44
 */
public class Test {
    public static void main(String[] args) {
        /*
            构造方法:
                概述:构造方法是一个特殊的方法,主要用来创建对象并给属性赋值.
                定义:
                    无参构造方法:
                        权限修饰符 类名(){
                        }
                    有参构造方法:
                        权限修饰符 类名(形参列表){
                            给属性赋值
                        }
                特点:
                    1.构造方法没有返回值类型,连void不能写
                    2.构造方法的名字就是类名
                    3.通过new来调用构造方法
                使用: 通过new来调用
         */
        // 通过调用空参构造方法创建对象
        Student stu1 = new Student();
        System.out.println(stu1.getName()+","+stu1.getAge());// null,0

        // 通过调用有参构造方法创建对象
        Student stu2 = new Student("冰冰",18);
        System.out.println(stu2.getName()+","+stu2.getAge());// 冰冰,18

    }
}
```

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属性的初始化

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

 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

```
package com.itheima.demo15_构造方法的注意事项;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 16:58
 */
public class Student {
    /**
     * 姓名
     */
    private String name;
    /**
     * age
     */
    private int age;
```

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

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

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

    // 有参构造方法
    public Student(int age){
        this.age = age;
    }

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

    public int getAge(){
        return age;
    }
}

package com.itheima.demo15_构造方法的注意事项;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 16:56
 */
public class Test {
    public static void main(String[] args) {
        /*
            构造方法的注意事项:
                1.构造方法没有返回值,连void都不能写
                2.构造方法名和类名一致
                3.如果一个类没有定义构造方法,系统会自动生成一个空参构造方法
                4.如果一个类定义了构造方法,系统就不会自动生成一个空参构造方法
                5.构造方法可以重载
                6.构造方法只能给属性赋值一次,而set方法可以给属性赋值无数次
                  因为调用构造方法,就会创建一个新的对象

         */
         //调用空参构造方法创建对象
        Student stu1 = new Student();

        // 通过有参构造方法创建对象
        Student stu2 = new Student("冰冰",18);
        Student stu3 = new Student("冰冰",18);

        System.out.println(stu2.getAge());// 18
        // 通过set方法给属性赋值
        stu2.setAge(19);
```

```
        System.out.println(stu2.getAge());// 19
        stu2.setAge(20);
        System.out.println(stu2.getAge());// 20

    }
}
```

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
- 构造方法只能给属性赋值一次,不能重复赋值,可以谁有set方法给属性重复赋值
- 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：

```
package com.itheima.demo16_标准类;
```

```java
/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 17:10
 */
public class Student {
    // 成员变量--private
    /**
     * 姓名
     */
    private String name;
    /**
     * 年龄
     */
    private int age;

    // 空参构造方法  alt+insert--->Constructor
    public Student() {
    }

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

    // set\get方法 alt+insert---> setter and getter
    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;
    }

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

}
```

小结

- 标准类
    - 成员变量----private
    - 空参构造
    - 满参构造

    - set\get方法
    - 成员方法

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。

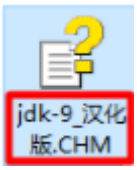
    - API其实就是jdk中核心类库的说明文档
    - 对于jdk中的核心类库只需要知道如何使用,无须关心他是如何实现的

API的使用步骤

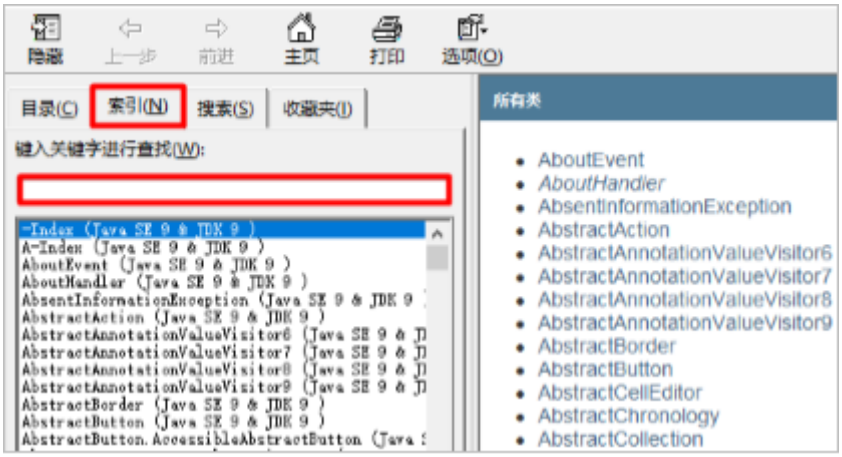
1. 打开API帮助文档。
2. 点击显示，找到索引，看到输入框。
3. 你要找谁？在输入框里输入，然后回车。
4. 看包。java.lang下的类不需要导包，其他需要。
5. 看类的解释和说明。
6. 看构造方法。
7. 看成员方法。

演示API的使用

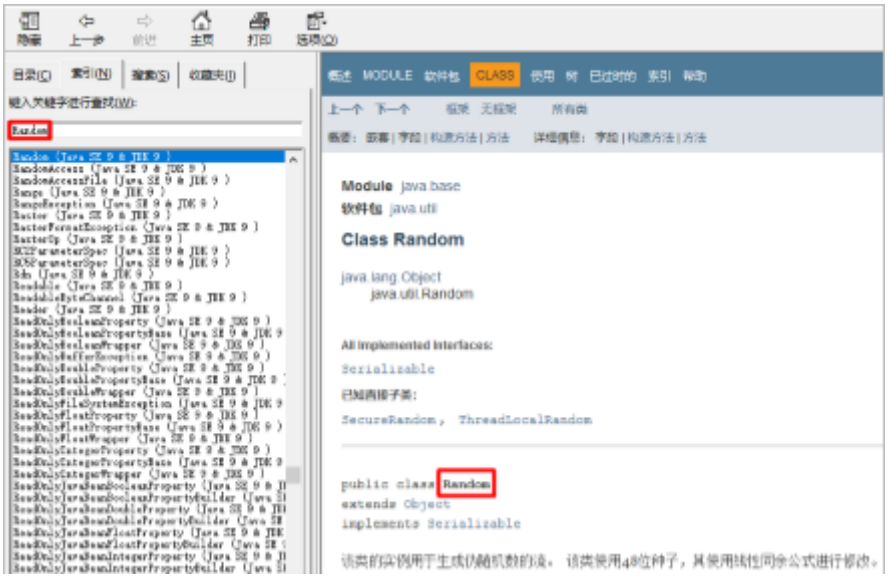
- 打开帮助文档



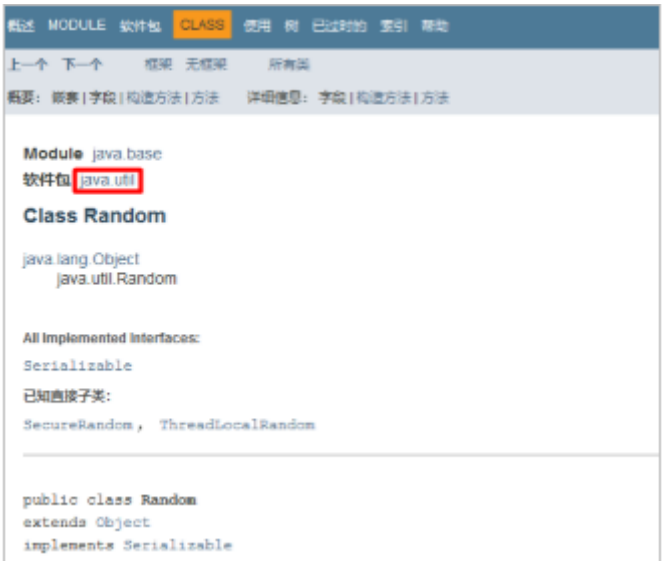
- 找到索引选项卡中的输入框



- 在输入框中输入Random



- 看类在哪个包下



- 看类的描述

Class Random

java.lang.Object
  java.util.Random

All Implemented Interfaces:
Serializable

已知的直接子类:
SecureRandom, ThreadLocalRandom

public class Random
extends Object
implements Serializable

- 看构造方法

构造方法摘要

构造方法	
Constructor	描述
Random()	创建一个新的随机数生成器。
Random(long seed)	使用单个 long种子创建一个新的随机数生成器。

- 看成员方法

Modifier and Type	方法	描述
DoubleStream	doubles()	
DoubleStream	doubles(double randomNumberOrigin, double randomNumberBound)	
DoubleStream	doubles(long streamSize)	
DoubleStream	doubles(long streamSize, double randomNumberOrigin, double randomNumberBound)	
IntStream	ints()	
double	nextDouble()	
float	nextFloat()	
double	nextGaussian()	
int	nextInt()	
int	nextInt(int bound)	方法名，参数

```
package com.itheima.demo17_API;

import java.util.Scanner;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/5 17:24
 */
public class Test {
    public static void main(String[] args) {
        /*
            api的使用步骤:
                1.打开api文档
                2.点击显示
                3.点击索引,在输入框中,输入要查找的类\接口
                4.查看类的包   如果在java.lang包就不需要导包,其余都需要导包
                5.查看类的解释说明
                6.查看类的构造方法
                7.查看类的成员方法
```

```
            举例: Scanner类
                1.查看类的包          java.util  导包
                2.查看类的解释说明     是一个文本扫描器,可以扫描基本类型的数据和字符串
                3.查看类的构造方法     Scanner(InputStream source)
                4.查看类的成员方法
                     byte nextByte()
                     short nextShort()
                     short nextInt()
                     Long nextLong()
                    boolean nextBoolean()
                     double nextDouble()
                    float nextFloat()

                     String nextLine()  可以获取一行字符串    空格,回车,tab键都可以获取
                     String next()      可以获取单个字符串    空格,回车,tab键都不可以获
取

         */
        Scanner sc = new Scanner(System.in);
        /*System.out.println("请输入一个整数:");
        int num = sc.nextInt();
        System.out.println(num);*/

       /* System.out.println("请输入一个小数:");
        double numD = sc.nextDouble();
        System.out.println(numD);*/

        //System.out.println("请输入一个字符串:");
       /* String str = sc.nextLine();
        System.out.println(str);*/

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

        System.out.println("请输入年龄:");
        int age = sc.nextInt();
        System.out.println("年龄:"+age);// 18

        System.out.println("请输入姓名:");
        String name = sc.next();
        System.out.println("姓名:"+name);

        //String name = sc.nextLine();
        //System.out.println("姓名:"+name);
    }
}
```

小结

略

总结

必须练习:

1.类的定义
2.对象的创建和使用
3.对象的内存图-----建议理解透彻
4.制作标准类

- 能够知道类和对象的关系
  类是对象的抽象,对象是类的实例
  对象是根据类来创建的,类中有什么对象就有什么
  对象和对象之间是相互独立

- 能够完成类的定义及使用
  格式:

```
public class 类名{
    成员变量
    成员方法
}
```

  创建对象: 通过new调用构造方法创建对象
  对象的使用:
    访问成员变量:对象名.成员变量名
    访问成员方法:
      无返回值方法: 对象名.方法名(实参);
      有返回值方法:
        直接调用: 对象名.方法名(实参);
        赋值调用: 数据类型 变量名 = 对象名.方法名(实参);
        输出调用: System.out.println(对象名.方法名(实参));

- 能够知道对象在内存中的初始化过程
  画图
- 能够知道局部变量和成员变量的区别
  位置不同,内存中位置不同,生命周期不同,默认值不同
  位置不同: 成员变量在类中,方法外;局部变量在方法中
  内存中位置不同: 成员变量在堆区; 局部变量在栈区
  生命周期不同;
    成员变量随着对象的创建而存在,随着对象的销毁而销毁
    局部变量随着方法的调用而存在,随着方法的执行完毕而销毁
  默认值不同: 成员变量有默认值,局部变量没有默认值

- 能够知道private关键字的特点
  被修饰的成员变量或者成员方法只能在本类中访问

- 能够知道this关键字的作用
  区别同名的成员变量和局部变量

- 能够知道构造方法的格式和注意事项
  1.构造方法没有返回值,连void都不能写
  2.构造方法名和类名一致
  3.构造方法可以重载
  4.构造方法通过new来调用
  5.构造方法只能给属性赋值一次
  6.如果一个类中没有定义构造方法,系统就会默认生成空参构造方法,
    如果一个类中定义了构造方法,系统就不会默认生成空参构造方法,
- 能够完成一个标准类代码的编写及测试
  成员变量--private
  空参构造方法
  满参构造方法(建议)
  set\get方法
  成员方法

- 能够知道帮助文档的使用步骤
  打开api
  点击显示
  点击索引,在输入框中输入要查找的内容
  查看包
  查看类的解释说明
  查看类的构造方法
  查看类的成员方法