1. 类和对象

面向对象和面向过程的思想对比:

面向过程: 是一种以过程为中心的编程思想,实现功能的每一步,都是自己实现的

面向对象: 是一种以对象为中心的编程思想,通过指挥对象实现具体的功能

1.1 类和对象的关系

客观存在的事物皆为对象,所以我们也常常说万物皆对象。

• 类

- o 类的理解
 - 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
 - 类是对象的数据类型,类是具有相同属性和行为的一组对象的集合
 - 简单理解: 类就是对现实事物的一种描述
- 。 类的组成
 - 属性: 指事物的特征,例如: 手机事物(品牌,价格,尺寸)
 - 行为: 指事物能执行的操作,例如: 手机事物(打电话,发短信)
- 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
 - o 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**

1.2 类的定义【应用】

类的组成是由属性和行为两部分组成

- 属性: 在类中通过成员变量来体现 (类中方法外的变量)
- **行为**: 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

```
1
   public class Student {
 2
      // 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
 3
 4
      String name;
 5
      int age;
 6
      // 行为 : 学习
 7
       // 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.
 8
9
      public void study(){
          System.out.println("学习");
10
11
12
```

1.3 对象的创建和使用

- 创建对象的格式:
 - o 类名 对象名 = new 类名();
- 调用成员的格式:
 - o 对象名.成员变量
 - o 对象名.成员方法();
- 示例代码:

```
package com.itheima.object1;
1
2
    public class TestStudent {
3
4
           创建对象的格式:
 5
                  类名 对象名 = new 类名();
 6
 7
           调用成员变量的格式:
                  对象名.变量名
8
9
           调用成员方法的格式:
10
                  对象名.方法名();
        */
11
       public static void main(String[] args) {
12
           // 类名 对象名 = new 类名();
13
14
           Student stu = new Student();
15
           // 对象名.变量名
           // 默认初始化值
16
           System.out.println(stu.name); // null
17
           System.out.println(stu.age); // 0
18
19
20
           stu.name = "张三";
21
           stu.age = 23;
22
23
           System.out.println(stu.name); // 张三
24
           System.out.println(stu.age); // 23
25
           // 对象名.方法名();
26
```

```
      27
      stu.study();

      28
      // com.itheima.object1.Student@b4c966a

      29
      // 全类名(包名 + 类名)

      30
      System.out.println(stu);

      31
      }

      32
      }
```

1.4 案例-手机类的创建和使用

需求: 首先定义一个手机类,然后定义一个手机测试类,在手机测试类中通过对象完成成员变量和成员方法的使用

分析:

成员变量:品牌,价格成员方法:打电话,发短信

• 示例代码:

```
1
    package com.itheima.test1;
 2
 3
    public class Phone {
        // 品牌, 价格
 4
        String brand;
 5
        int price;
 6
 8
        // 打电话,发短信
        public void call(String name){
 9
            System.out.println("给"+name+"打电话");
10
        }
11
12
        public void sendMessage(){
13
            System.out.println("群发短信");
15
        }
16
    }
```

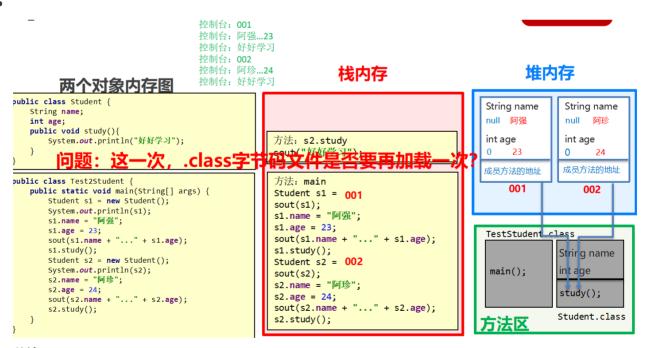
```
package com.itheima.test1;
 1
 2
    public class TestPhone {
 3
        public static void main(String[] args) {
           // 1. 创建对象
           Phone p = new Phone();
 6
 7
           // 2. 给成员变量进行赋值
           p.brand = "大米";
 8
 9
           p.price = 2999;
10
           // 3. 打印赋值后的成员变量
           System.out.println(p.brand + "..." + p.price);
11
12
           // 4. 调用成员方法
           p.call("阿强");
13
           p.sendMessage();
14
15
        }
16
   }
```

2. 对象内存图

2.1 单个对象内存图【理解】



2.2 多个对象内存图【理解】



• 总结:

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

2.3 多个对象指向相同内存图【理解】

控制台: 阿珍...阿珍

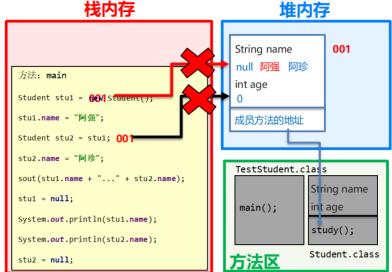
控制台: NullPointerException

控制台: 阿珍

两个引用指向同一对象内存图

```
public class Student {
    String name;
    int age;
    public void study(){
        System.out.println("好好学习");
    }
}

public class Test3Student {
    public static void main(String[] args) {
        Student stu1 = new Student();
        stu1.name = "阿强";
        Student stu2 = stu1;
        stu2.name = "阿珍";
        sout(stu1.name + "..." + stu2.name);
        stu1 = null;
        //System.out.println(stu1.name);
        System.out.println(stu2.name);
        stu2 = null;
```



• 总结:

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。

3. 成员变量和局部变量

3.1 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 (桟内存)
- **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)

4. 封装

4.1 private关键字

概述: 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

示例代码:

```
1 /*
2 学生类
```

```
3
   */
 4
    class Student {
 5
        //成员变量
        String name;
 6
 7
        private int age;
 8
 9
        //提供get/set方法
10
        public void setAge(int a) {
            if(a<0 || a>120) {
11
12
                System.out.println("你给的年龄有误");
            } else {
13
14
                age = a;
15
            }
16
        }
17
        public int getAge() {
18
19
           return age;
20
        }
21
        //成员方法
22
        public void show() {
23
            System.out.println(name + "," + age);
24
25
        }
26
    }
27
        学生测试类
28
29
30
    public class StudentDemo {
31
        public static void main(String[] args) {
32
            //创建对象
33
            Student s = new Student();
           //给成员变量赋值
34
           s.name = "林青霞";
35
           s.setAge(30);
36
37
            //调用show方法
38
            s.show();
        }
39
40
    }
```

4.2 private关键字的使用

• 需求:

- 。 定义标准的学生类,要求name和age使用private修饰
- o 并提供set和get方法以及便于显示数据的show方法
- 。 测试类中创建对象并使用, 最终控制台输出 林青霞, 30

• 示例代码:

```
5
        //成员变量
 6
        private String name;
 7
        private int age;
 8
 9
        //get/set方法
10
        public void setName(String n) {
11
            name = n;
12
13
        public String getName() {
14
15
            return name;
16
        }
17
18
        public void setAge(int a) {
19
            age = a;
20
        }
21
22
        public int getAge() {
23
            return age;
24
25
26
        public void show() {
27
            System.out.println(name + "," + age);
28
29
30
        学生测试类
31
32
    public class StudentDemo {
33
34
        public static void main(String[] args) {
35
            //创建对象
            Student s = new Student();
36
37
            //使用set方法给成员变量赋值
38
39
            s.setName("林青霞");
40
            s.setAge(30);
41
42
            s.show();
43
            //使用get方法获取成员变量的值
44
            System.out.println(s.getName() + "---" + s.getAge());
45
            System.out.println(s.getName() + "," + s.getAge());
46
47
48
        }
49
    }
```

4.3 this关键字【应用】

概述:this修饰的变量用于指代成员变量,其主要作用是(区分局部变量和成员变量的重名问题)

- 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

代码实现:

```
public class Student {
 2
        private String name;
        private int age;
 3
 4
        public void setName(String name) {
 5
 6
             this.name = name;
 8
 9
        public String getName() {
10
             return name;
11
        }
12
13
         public void setAge(int age) {
             this.age = age;
14
        }
15
16
        public int getAge() {
17
18
             return age;
19
20
        public void show() {
21
22
             System.out.println(name + "," + age);
23
        }
24
```

4.4 this内存原理【理解】

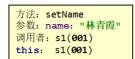
- 注意:this代表当前调用方法的引用,哪个对象调用的方法,this就代表哪一个对象
- 图解:

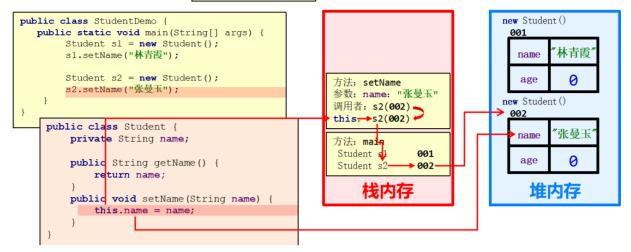
.

this内存原理

```
new Student()
public class StudentDemo {
  public static void main(String[] args) {
       Student s1 = new Student();
                                                                                            林青霞
       s1.setName("林青霞");
                                                                                     name
     public class Student {
                                                                                     age
                                                                                              0
        private String name;
                                                      方法: setName
                                                      <u>参数→</u>name: "林青霞"
         public String getName() {
                                                     调用者: s1(001)
            return name;
                                                     this. >s1(001)
         public void setName(String name) {
                                                     方法: main,
            this.name = name;
                                                                  ▶ 001
                                                      Student s1-
                                                          栈内存
                                                                                       堆内存
```

this内存原理





4.5 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

5. 构造方法

5.1 构造方法的格式和执行时机

格式注意:

- 。 方法名与类名相同, 大小写也要一致
- o 没有返回值类型,连void都没有
- o 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

• 示例代码:

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

```
4
 5
        //构造方法
 6
        public Student() {
            System.out.println("无参构造方法");
 7
 8
 9
10
        public void show() {
11
            System.out.println(name + "," + age);
12
13
14
        测试类
15
16
17
    public class StudentDemo {
18
        public static void main(String[] args) {
            //创建对象
19
20
            Student s = new Student();
21
            s.show();
22
        }
23
```

5.2 构造方法的作用

• 用于给对象的数据(属性)进行初始化

```
1
   package com.itheima.constructor;
 2
3
   public class Student {
       /*
4
          格式:
 5
 6
                 1. 方法名需要跟类名相同,大小写也要一致
                 2. 没有返回值类型, 连void都没有
 8
                 3. 没有具体的返回值(不能由return带回具体的结果)
9
        */
10
11
       private String name;
12
13
       private int age;
14
       // 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
15
16
       public Student(){}
17
       // 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
18
       public Student(String name, int age){
19
          this.name = name;
20
21
          this.age = age;
          System.out.println("我是Student类的构造方法");
22
23
       }
24
       public void show(){
25
           System.out.println(name + "..." + age);
26
27
```

```
28 }
```

```
1
    package com.itheima.constructor;
2
3
    public class TestStudent {
        public static void main(String[] args) {
4
            Student stu1 = new Student("张三",23);
5
            stu1.show();
6
7
            Student stu2 = new Student();
8
9
        }
10
   }
```

5.3 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

5.4 标准类的代码编写和使用

代码:

```
package com.itheima.test3;
 1
 2
 3
 4
        JavaBean类: 封装数据
 5
    public class Student {
 6
 7
        private String name;
        private int age;
 8
 9
10
        public Student() {
11
        }
12
        public Student(String name, int age) {
13
14
            this.name = name;
            this.age = age;
15
```

```
16
17
18
        public String getName() {
            return name;
19
20
        }
21
22
        public void setName(String name) {
23
            this.name = name;
        }
24
25
        public int getAge() {
26
27
            return age;
28
        }
29
        public void setAge(int age) {
30
31
            this.age = age;
32
        }
33
        public void show(){
34
            System.out.println(name + "..." + age);
35
36
        }
37
    }
38
```

```
1
    package com.itheima.test3;
2
3
    public class TestStudent {
       public static void main(String[] args) {
4
           // 1. 无参数构造方法创建对象,通过setXxx方法给成员变量进行赋值
5
           Student stu1 = new Student();
 6
           stu1.setName("张三");
7
           stu1.setAge(23);
8
9
           stu1.show();
10
           // 2. 通过带参数构造方法,直接给属性进行赋值
11
12
           Student stu2 = new Student("李四",24);
13
           stu2.show();
14
       }
15
   }
16
```