day02 【复习回顾、继承、抽象类模板设计模式】

今日内容

- 面向对象复习
  - 制作标准类
  - 创建对象和使用对象
  - 对象的内存图
  - 匿名对象
- 继承----------重点\掌握
  - 如何继承
  - 继承后成员访问特点
- 抽象方法
- 模板设计模式
- final 关键字----掌握
  - 修饰类\方法\变量
- static关键字----掌握
  - 修饰成员变量\成员方法

教学目标

- 能够写出类的继承格式
- 能够说出继承的特点
- 能够说出子类调用父类的成员特点
- 够说出方法重写的概念
- 能够说出`this`可以解决的问题
- 能够说出`super`可以解决的问题
- 描述抽象方法的概念
- 写出抽象类的格式
- 写出抽象方法的格式
- 能够说出父类抽象方法的存在意义
- 描述`final`修饰的类的特点
- 描述`final`修饰的方法的特点
- 描述`final`修饰的变量的特点

第一章 面向对象复习

知识点--1.1 类和对象

目标:

- 掌握如何定义一个标准类以及创建并使用对象

路径:

- 定义一个标准类
- 创建并使用对象

讲解:

定义一个标准类

- 定义类的格式:

```
public class 类名{
    // 成员变量(private修饰)
    // 构造方法(空参,满参)
    // set\get方法
    // 成员方法(行为)
}
```

- 案例

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

    // 构造方法
    public Student(){

    }

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

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

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

    public String getName(){
        return name;
    }

    public int getAge(){
        return age;
    }

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

创建并使用对象

- 创建对象的格式

```
通过调用构造方法创对象:
    类名 对象名 = new 类名(实参);
```

- 使用对象:
  - 对象访问成员变量

    ```
    对象名.成员变量名
    ```

  - 对象访问成员方法

    ```
    对象名.成员方法名(实参);
    ```

- 案例

```
public class Test {
    public static void main(String[] args) {
        /*
            定义一个标准类
            使用对象
         */
        // 创建对象
        Student stu1 = new Student();
        Student stu2 = new Student("张三", 18);

        // 访问成员变量
        stu1.setName("李四");
        stu1.setAge(19);

        // 访问成员方法
        stu1.show();// 李四,19
        stu2.show();// 张三,18
    }
}
```

小结:

- 定义一个标准类
- 创建对象
- 使用对象
  - 访问成员变量: 对象名.成员变量名
  - 访问成员方法:
    - 无返回值的方法: 对象名.成员方法名(实参);
    - 有返回值的方法:
      - 直接调用: 对象名.成员方法名(实参);
      - 赋值调用: 数据类型 变量名 = 对象名.成员方法名(实参);

- 输出调用: System.out.println(对象名.成员方法名(实参));

知识点--1.2 对象的内存图

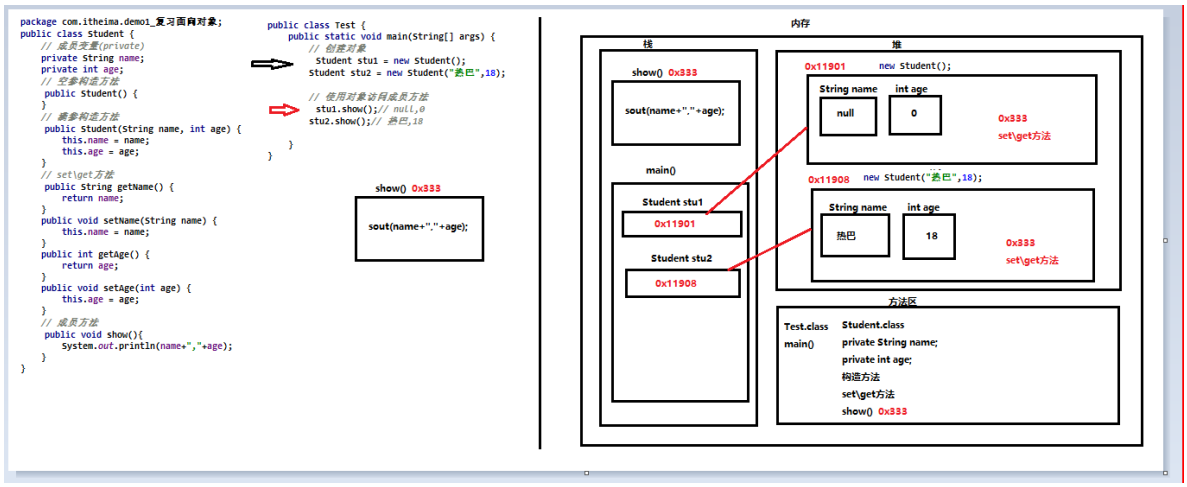
目标

- 理解对象的内存图

路径

- 对象的内存图

讲解



小结

- 只要是new对象就会在堆区开辟一块独立的空间
- 只要调用方法,方法就会被加载进栈
- 只要方法执行完毕,方法就会被弹栈

知识点--1.3 匿名对象

目标

- 理解什么是匿名对象并会使用匿名对象

路径

- 匿名对象的概述
- 使用匿名对象

讲解

匿名对象的概述

什么是匿名对象：就是指"没有名字"的对象。

```
有名字的对象：
    Student stu = new Student();
    stu.show();
    stu.study();
匿名对象：
    new Student();
```

使用匿名对象

- 特点:匿名对象只能使用一次

```
package com.itheima.demo2_匿名对象;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/6 9:04
 */
public class Test {
    public static void main(String[] args) {
        /*
            匿名对象:
                概述:没有名字的对象
                特点:匿名对象只能使用一次
                使用场景:当某个类的对象只需要使用一次的时候,就可以使用该类的匿名对象
                        例如:方法的参数,方法的返回值
         */
        // 创建对象
        Student stu1 = new Student("热巴",18);// 有名字的对象
        stu1.show();
        stu1.show();

        System.out.println("==================================");
        //匿名对象
        new Student("热巴",18).show();// 没有名字的对象
        new Student("热巴",18).show();// 没有名字的对象

        System.out.println("==================================");
        // 调用method1方法
        Student stu2 = new Student("热巴",18);// 0x11901
        method1(stu2);// 有名字的对象传参
        method1(new Student("热巴",18));// 匿名对象的方式传参数

        System.out.println("==================================");
        Student stu3 = method2();// 0x11908
        stu3.show();// 丽颖,18

    }

    public static void method1(Student stu){// 0x11901
        stu.show();
    }


    public static Student method2(){
        //Student stu = new Student("丽颖",18);// 0x11908
        //return stu;// 0x11908
```

```
            return new Student("丽颖",18);
        }


    }
```

小结

- 匿名对象：就是指"没有名字"的对象。
- 特点: 只能使用一次

第二章 继承

面向对象语言的三大特征:封装,继承,多态

知识点--2.1 继承概述

目标:

- 能够理解什么继承

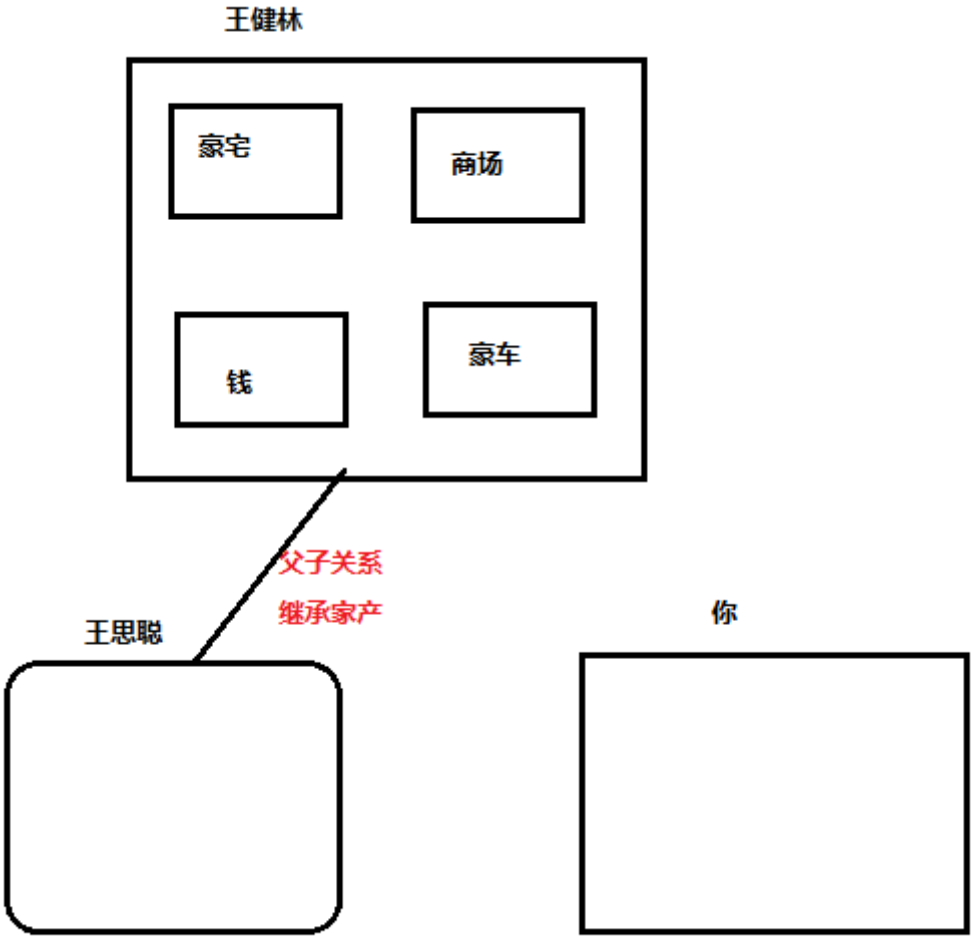
路径:

- 为什么要有继承
- 继承的含义
- 继承的好处

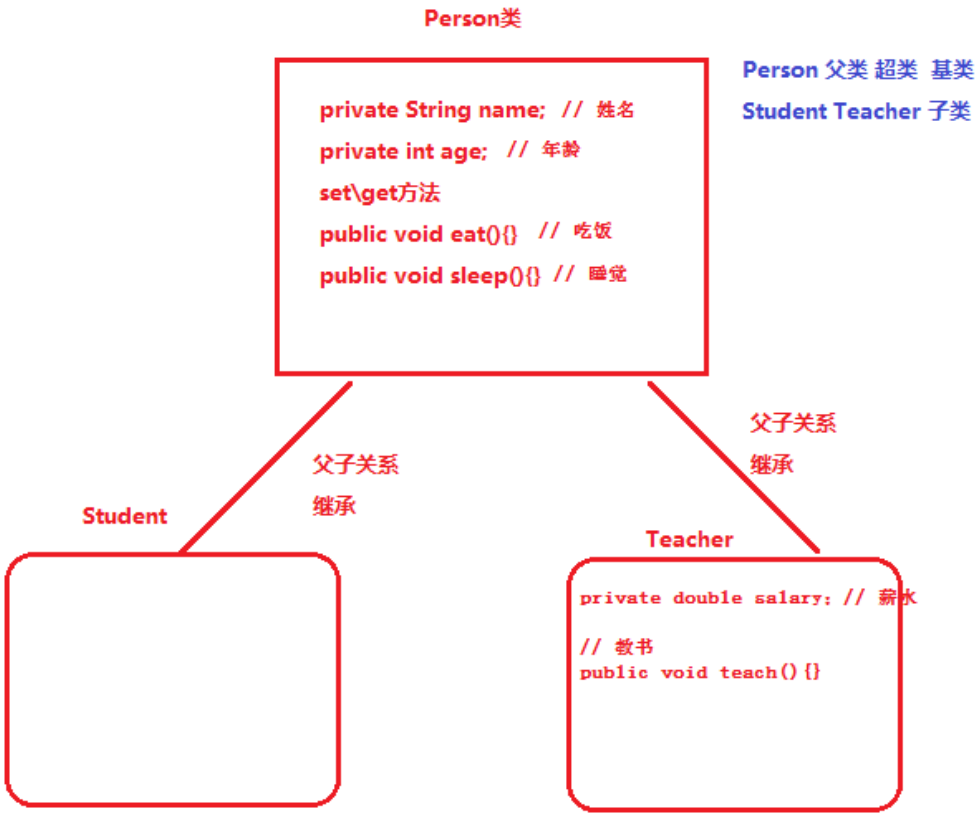
讲解:

2.1.1 为什么要有继承

现实生活中，为什么要有继承?


王健林
豪宅
商场
钱
豪车
父子关系
继承家产
王思聪
你

程序中为什么要有继承?


Person类
private String name; // 姓名
private int age; // 年龄
set\get方法
public void eat(){} // 吃饭
public void sleep(){} // 睡觉
Person 父类 超类 基类
Student Teacher 子类
父子关系
继承
Student
父子关系
继承
Teacher
private double salary; // 薪水
// 教书
public void teach(){}

2.1.2 继承的含义

继承：在java中指的是“一个类”可以“继承自”“另一个类”。 "被继承的类"叫做: 父类/超类/基类，"继承其他类的类"叫做:子类。继承后，“子类”中就“拥有”了“父类”中所有的成员(成员变量、成员方法)。 “子类就不需要再定义了”。

2.1.3 继承的好处

1. 提高**代码的复用性**（减少代码冗余，相同代码重复利用）。
2. 使类与类之间产生了关系。

小结

- **继承**：在java中指的是“一个类”可以“继承自”“另一个类”。 "被继承的类"叫做: 父类/超类/基类，"继承其他类的类"叫做:子类。继承后，“子类”中就“拥有”了“父类”中所有的成员(成员变量、成员方法)。 “子类就不需要再定义了”。

知识点--2.2 继承的格式

目标:

- 能够掌握如何实现继承

路径:

- 继承的格式
- 继承的演示

讲解:

继承的格式

通过 `extends` 关键字，可以声明一个子类继承另外一个父类，定义格式如下：

```
class 父类 {
    ...
}

class 子类 extends 父类 {
    ...
}
```

需要注意：Java是单继承的，一个类只能继承一个直接父类，并且满足is-a的关系,例如:Dog is a Animal, Student is a Person

继承的演示

```
人类:
public class Person {
    // 成员变量
    String name;
    int age;

    // 功能方法
    public void eat(){
```

```
        System.out.println("吃东西...");
    }

    public void sleep(){
        System.out.println("睡觉...");
    }
}
老师类: extends 人类
public class Teacher extends Person {
    double salary;// 独有的属性
    public void teach(){}// 独有的方法
}
学生类: extends 人类
public class Student extends Person{

}
Dog:  extends 人类
public class Dog extends Person{// 语法上是可以的,但不符合现实逻辑(不符合is a的关系)

}
测试:
public class Test {
    public static void main(String[] args) {
        Teacher t = new Teacher();
        System.out.println(t.name);
        System.out.println(t.age);
        t.eat();
        t.sleep();
    }
}
```

小结

- ```
  public class 子类名 extends 父类名{

  }
  ```
- 通过继承可以**将一些共性的属性,行为抽取到一个父类中**,子类只需要继承即可,提供了代码的复用性

知识点--2.3 继承后成员访问规则

目标:

- 能够掌握继承后成员访问规则

路径:

- 继承后构造方法的访问规则
- 继承后私有成员的访问规则
- 继承后非私有成员的访问规则

讲解:

继承后构造方法的访问规则
  ```

- 构造方法不能被继承

```
class Fu {
    // 构造方法
    Fu(){}
    Fu(String name,int age){}
}

class Zi extends Fu{

}

public class Test {
    public static void main(String[] args) {
        /*
            构造方法的访问规则:父类的构造方法不能被子类继承
            私有成员的访问规则:
            非私有成员的访问规则:
         */
        //Zi zi = new Zi("张三",18);// 编译报错,因为没有继承
    }
}
```

继承后私有成员的访问规则

- 父类的"私有成员"可以被子类继承，但子类不能被直接访问。

```
public class Fu{
    private int num = 100;//私有成员，只能在父类内部使用。
    private void method(){
        System.out.println("私有成员方法");
    }
}
public class Zi extends Fu{

}
public class Demo {
    public static void main(String[] args) {
        Zi z = new Zi();
        System.out.println(z.num);// 编译错误
        z.method();// 编译错误
    }
}
```

继承后非私有成员的访问规则

- 当通过"子类"访问非私有成员时，先在子类中找，如果找到就使用子类的，找不到就继续去"父类"中找。

```
public class Fu{
    int money = 100;
    public void method(){
        System.out.println("Fu 类中的成员方法method");
    }
```

```
}
public class Zi extends Fu{
    int money = 1;
     public void method(){
        System.out.println("Zi 类中的成员方法method");
    }
}
public class Demo{
    public static void main(String[] args){
        Zi z = new Zi();
        System.out.println(z.money);//1
        z.method();// Zi 类中的成员方法method
    }
}
```

小结

- 构造方法不能被继承
- 父类的"私有成员"可以被子类继承，但子类不能被直接访问。
- 当通过"子类"访问非私有成员时，先在子类中找，如果找到就使用子类的，找不到就继续去"父类"中找。

知识点--2.4 方法重写

目标:

- 能够正确对父类中的方法进行重写

路径:

- 方法重写的概念
- 重写的注意事项

讲解:

方法重写的概念

方法重写：子类中出现与父类一模一样的方法时（返回值类型，方法名和参数列表都相同），会出现覆盖效果，也称为重写或者复写。**声明不变，重新实现。**

```
package com.itheima.demo6_方法重写;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/6 10:21
 */
class Fu{
    public void method(){
        System.out.println("Fu method");
    }
}
class Zi extends Fu{

    @Override
    public void method() {
```

```
        System.out.println("Zi method");
    }

    public void show(){
        System.out.println("Zi show");
    }
}
public class Test {
    public static void main(String[] args) {
        /*
            方法重写:
                方法重写 ：子类中出现与父类一模一样的方法时（返回值类型，方法名和参数列表都相同），
                                会出现覆盖效果，也称为重写或者复写。声明不变，重新实现。
                注意事项:
                    1.一定要是父子类关系
                    2.子类中重写的方法返回值类型,方法名,参数列表一定要和父类一模一样
                    3.子类中重写的方法可以使用@Override注解进行标识,如果不是重写的方法使用@Override注解标识就会报错
                        建议开发中重写的方法使用@Override注解标识,这样可以提高代码的可读性
                    4.子类重写父类的方法的访问权限不能低于父类的访问权限
                        访问权限: public > protected > 默认(空) > private
         */
        Zi zi = new Zi();
        zi.method();
    }
}
```

重写的注意事项

- 方法重写是发生在子父类之间的关系。
- 子类方法重写父类方法，返回值类型、方法名和参数列表都要一模一样。
- 子类方法重写父类方法，必须要保证权限大于等于父类权限。
    - 访问权限从大到小: public protected (默认) private
- 使用@Override注解，检验是否重写成功，重写注解校验！
    - 建议重写方法都加上这个注解，一方面可以提高代码的可读性，一方面可以防止重写出错！

方法重写的使用场景

```
package com.itheima.demo7_方法重写的使用场景;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/6 10:38
 */
class Fu{
    public void sport(){
        System.out.println("Fu 运动的方式跑步");
    }

    public void run(){
        System.out.println("Fu 第1圈");
        System.out.println("Fu 第2圈");
        System.out.println("Fu 第3圈");
```

```
    }
}

class Zi extends Fu{
    // 子类方法的实现和父类方法的实现完全不同
    @Override
    public void sport() {
        System.out.println("Zi 运动的方式游泳");
    }

    // 子类方法的实现要保留父类方法的功能,但要在父类功能的基础之上额外增加功能
    @Override
    public void run() {
        // 让父类的方法执行=====复制父类的代码过来
        super.run();// 调用父类的方法

        // 额外增加的代码
        System.out.println("Zi 第4圈");
        System.out.println("Zi 第5圈");
        System.out.println("Zi 第6圈");
        System.out.println("Zi 第7圈");
        System.out.println("Zi 第8圈");
        System.out.println("Zi 第9圈");
        System.out.println("Zi 第10圈");
    }
}

public class Test {
    public static void main(String[] args) {
        /*
            方法重写的使用场景:
                当父类的方法无法满足子类的需求的时候,子类就会去重写父类的方法
         */
        // 创建子类对象
        Zi zi = new Zi();
        // 调用运动的方法
        zi.sport();
        // 调用跑步的方法
        zi.run();


    }
}
```

小结

- 方法重写:子类中出现与父类一模一样的方法时（返回值类型，方法名和参数列表都相同）
- 使用场景: 当父类的某个方法,子类有不同的实现,那么就可以重写该方法
- 建议: 校验方法重写或者标识方法重写,可以使用@Override注解

知识点--2.5 this和super关键字

目标:

- 掌握super和this 的用法

路径:

- this和super关键字的介绍
- this关键字的三种用法
- super关键字的三种用法

讲解:

this和super关键字的介绍

- this：存储的“当前对象”的引用；
  - this可以访问：本类的成员属性、成员方法、构造方法；
- super：存储的“父类对象”的引用；
  - super可以访问：父类的成员属性、成员方法、构造方法；

this关键字的三种用法

- this访问本类成员变量: **this.成员变量**

```
public class Student{
    String name = "张三";
    public void show(){
        String name = "李四";
        System.out.println("name = " + name);// 李四
        System.out.println("name = " + this.name);// 张三
    }
}
```

- this访问本类成员方法: **this.成员方法名();**

```
public class Student{
    public void show(){
        System.out.println("show方法...");
        this.eat();
    }
    public void eat(){
        System.out.println("eat方法...");
    }
}
```

- this访问本类构造方法: **this()可以在本类的一个构造方法中，调用另一个构造方法**

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

    public Student(String name) {
        this();//当使用this()调用另一个构造方法时，此代码必须是此构造方法的第一句有效代码。
        System.out.println("有参构造方法...");
    }
}
```

```
public class Demo {
    public static void main(String[] args) {
        Student stu2 = new Student();
    }
}
```

super关键字的三种用法

- super访问父类的成员变量: super.父类成员变量名

```
class Fu{
    int num = 100;
}

class Zi extends Fu{
    int num = 10;

    public void show(){
        int num = 1;
        System.out.println("局部变量num:"+num);// 1
        System.out.println("Zi 类中的num:"+this.num);// 10
        System.out.println("Fu 类中的num:"+super.num);// 100

      }
}
```

- super访问父类的成员方法: super.成员方法名();

```
class Fu{
    public void method1(){
        System.out.println("Fu method1...");
    }
}

class Zi extends Fu{
    public void show(){
        // 访问父类的method1方法
        super.method1();
    }

    @Override
    public void method1(){
        super.method1();// 调用父类的method1方法
        System.out.println("Zi method1...");
    }
}
```

- super访问父类的构造方法: super()

```
public class Fu{
    public Fu(){
        System.out.println("Fu 类的空参构造方法..");
    }
    public Fu(String name, int age) {
        System.out.println("Fu 类的有参构造方法..");
    }
```

```
}
public class Zi extends Fu{
    public Zi(){
        super();// 调用父类的空参构造方法
        System.out.println("Zi 类的空参构造方法..");
    }
    public Zi(String name,int age){
        super(name,age);// 调用父类的有参构造方法
         System.out.println("Zi 类的有参构造方法..");
    }
}
public class Demo {
    public static void main(String[] args) {
        Zi zi = new Zi();
        System.out.println("----------------------");
        Zi z2 = new Zi("刘德华", 17);
    }
}
```

小结

- this关键字的三种用法:
  this可以访问本类的成员变量: this.成员变量　　一般用来区分同名的成员变量和局部变量
  this可以访问本类的成员访问: this.成员方法名(实参);
  this可以访问本类的构造方法:
  空参构造: this();
  有参构造: this(实参);
  注意:
  1.只能在本类的构造方法中使用this调用其他构造方法
  2.在本类的构造方法中使用this调用其他构造方法,必须放在该构造方法的第一行,否则会报错
  3.两个构造方法不能使用this同时相互调用

- super关键字的三种用法:
  super可以访问父类的成员变量: super.成员变量　　一般用来区分父子类中同名的成员变量
  super可以访问父类的成员方法: super.成员方法(实参);　　一般用来在子类中访问父类的成员方法
  super可以访问父类的构造方法:
  空参构造: super();
  有参构造: super(实参);
  注意:
  1.子类的构造方法默认会调用父类的空参构造方法
  2.super访问父类的构造方法,可以用来初始化从父类继承过来的属性
  3.在子类的构造方法中,使用super调用父类的构造方法,必须放在子类构造方法的第一行

知识点-- 2.6 super的注意事项

目标

- 关于super的注意事项

路径

- super的注意事项一
- super的注意事项二

讲解

super的注意事项一

- super访问成员变量和成员方法: 优先去父类中找,如果有就直接使用,如果没有就去爷爷类中找,如果有,就用,依次类推...

```
class Ye{
    int num = 10;
    public void method(){
        System.out.println("Ye method");
    }
}
class Fu extends Ye{
    int num = 100;
    public void method(){
        System.out.println("Fu method");
    }
}
class Zi extends Fu{
    int num = 1000;
    public void show(){
        System.out.println(super.num);
        super.method();
    }
}

public class Test {
    public static void main(String[] args) {

        Zi zi = new Zi();
        zi.show();
    }
}
```

super的注意事项二

- 子类的构造方法默认会调用父类的空参构造方法,如果父类中的没有空参构造方法,只定义了有参构造方法,会编译报错

```
class Fu1{
    public Fu1(){
        System.out.println("Fu1 空参构造");
    }

    public Fu1(int num){
        System.out.println("Fu1 有参构造");
    }
}

class Zi1 extends Fu1{
```

```
    public Zi1(){
        // super();
    }

    public Zi1(int num){
        // super();
    }
}

// 问题: super调用父类的构造方法有什么用?
class Person{
    private String name;
    private int age;

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

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

class Student extends Person{
    public Student(String name,int age){
       super(name,age);
    }
}

public class Test2 {
    public static void main(String[] args) {
        /*
            super的注意事项二
              1.子类的构造方法默认会调用父类的空参构造方法
              2.如果父类中的没有空参构造方法,只定义了有参构造方法,会编译报错
            问题: super调用父类的构造方法有什么用?
            结果: 为了在创建子类对象的时候,初始化从父类继承过来的属性
         */
        // 通过调用子类的空参构造方法,创建子类对象
        // Zi1 zi = new Zi1();

        // 通过调用子类的有参构造方法,创建子类对象
        //Zi1 zi = new Zi1(100);

        // 创建Student类的对象
        Student stu = new Student("张三", 18);
        stu.show();

    }
}
```

小结

- super访问成员变量和成员方法: 优先去父类中找,如果有就直接使用,如果没有就去爷爷类中找,如果有,就用,依次类推...
- 子类的构造方法默认会调用父类的空参构造方法,如果父类中的没有空参构造方法,只定义了有参构造方法,会编译报错
- 子类构造方法中使用super调用父类的构造方法,是为了在创建子类对象的时候,初始化从父类继承过来的属性

知识点--2.7 继承体系对象的内存图

目标:

- 理解继承体系对象的内存图

路径:

- 继承体系内存图原理
- 书写继承案例
- 根据案例绘制内存图

讲解:

- 继承体系内存图原理---父类空间优先于子类对象产生

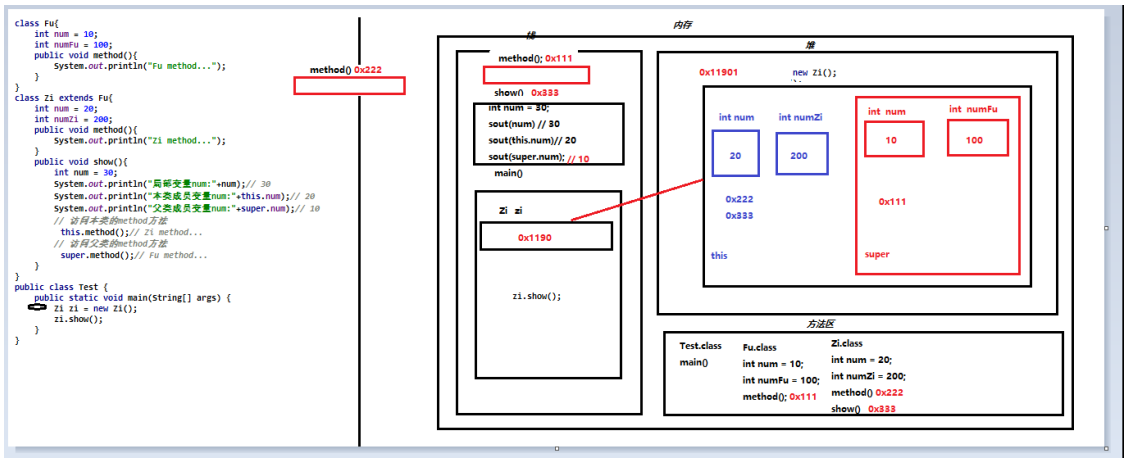
  在每次创建子类对象时，先初始化父类空间，再创建其子类对象本身。目的在于子类对象中包含了其对应的父类空间，便可以包含其父类的成员，如果父类成员非private修饰，则子类可以随意使用父类成员。代码体现在子类的构造方法调用时，一定先调用父类的构造方法。

- 书写继承案例

```
class Fu{
    int num = 10;
    int numFu = 100;
    public void method(){
        System.out.println("Fu method...");
    }
}
class Zi extends Fu{
    int num = 20;
    int numZi = 200;
    public void method(){
        System.out.println("Zi method...");
    }
    public void show(){
        int num = 30;
        System.out.println("局部变量num:"+num);// 30
        System.out.println("本类成员变量num:"+this.num);// 20
        System.out.println("父类成员变量num:"+super.num);// 10
        // 访问本类的method方法
        this.method();// Zi method...
        // 访问父类的method方法
        super.method();// Fu method...
    }
}
public class Test {
    public static void main(String[] args) {
        Zi zi = new Zi();
```

```
            zi.show();
        }
    }
```

- 根据案例绘制内存图



小结:

略

知识点--2.8 继承的特点

目标:

- 继承的特点

路径:

- Java只支持单继承，不支持多继承。
- 一个类只能有一个父类,但是可以有多个子类。
- 可以多层继承。

讲解:

1. Java只支持单继承，不支持多继承。

```
  // 一个类只能有一个父类，不可以有多个父类。
class A {

}
class B {

}
class C1 extends A {// ok

}
class C2 extends A, B {// error

}
```

1. 一个类只能有一个父类,但可以有多个子类。

```
  // A可以有多个子类
class A {

}
class C1 extends A {

}
class C2 extends  A {

}
```

1. 可以多层继承。

```
class A /*extends Object*/{// 爷爷    默认继承Object类

}
class B extends A {// 父亲

}
class C extends B {// 儿子

}
```

> 补充: 顶层父类是Object类。所有的类默认继承Object，作为父类。
>
> class A {} 默认继承Object类 直接继承Object类
>
> class B extends A{} B的父类就是A,但是A的父类是Object类 间接继承Object类
>
> java中所有类都是直接或者间接继承Object,所有类都是Object类的子类

小结:

- 类的继承只能是单继承,不能多继承,但是可以多层继承
- java中所有类都是直接或者间接继承Object,所有类都是Object类的子类

第三章 抽象类

知识点--3.1 抽象类的概述和定义

目标

- 理解抽象类的概述和抽象类的定义

路径

- 抽象类的概述和特点
- 抽象类的定义
- 抽象类中的成员

讲解

抽象类的概述

- 概述: 使用abstract关键字修饰的类就是抽象类

- 特点: 这种类不能被创建对象,它就是用来做父类的,被子类继承的

抽象类的定义

- 格式:

```
修饰符 abstract class 类名{

}
```

- 例如:

```
public abstract class Person{

}
```

抽象类中的成员

- 成员变量
- 成员方法
- 构造方法
- **抽象方法**

```
package com.itheima.demo13_抽象类的概述和定义;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/6 14:42
 */
public abstract class Animal {
    // 成员变量
    private String name;
    private int age;
    // 构造方法
    public Animal(){

    }
    public Animal(String name,int age){
        this.name = name;
        this.age = age;
    }
    // 成员方法

    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;
    }

    public void show(){
        System.out.println(name+","+age);
    }
    // 抽象方法 ---??

}
package com.itheima.demo13_抽象类的概述和定义;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/6 14:38
 */
public class Test {
    public static void main(String[] args) {
        /*
            抽象类:
                概述:使用abstract关键字修饰的类就是抽象类
                特点:抽象类不能创建对象,主要用来给子类继承的
                格式:
                    public abstract class 类名{
                        成员变量
                        构造方法
                        成员方法
                        抽象方法
                    }
                抽象类成员:
                        成员变量
                        构造方法
                        成员方法
                        抽象方法
                普通类和抽象类的区别:
                   1.普通类可以创建对象,抽象类不可以创建对象
                   2.普通类没有抽象方法,抽象类有抽象方法
         */
        //Animal anl1 = new Animal();// 编译报错,抽象类不能创建对象
        //Animal anl2 = new Animal("旺财",2);// 编译报错,抽象类不能创建对象
    }
}
```

小结

- 使用abstract关键字修饰的类就是抽象类
- 抽象类中的成员
  - 成员变量
  - 构造方法
  - 成员方法
  - 抽象方法
- 抽象类的特点
  - 不能创建对象,主要用来给子类继承的

知识点--3.2 抽象方法的概述和定义

目标

- 掌握抽象方法的概述和定义

路径

- 抽象方法的概述
- 抽象方法的定义

讲解

抽象方法的概述

- 没有方法体,使用abstract修饰的方法就是抽象方法

抽象方法的定义

```
修饰符 abstract 返回值类型 方法名(形参列表);
例如:
    public abstract void work();
```

抽象方法的作用: 强制要求子类重写的

```
package com.itheima.demo14_抽象方法的概述和定义;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/6 14:42
 */
public abstract class Animal {
    // 成员变量
    private String name;
    private int age;
    // 构造方法
    public Animal(){

    }
    public Animal(String name, int age){
        this.name = name;
        this.age = age;
    }
    // 成员方法

    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;
    }

    // 所有子类显示信息的方法实现都是一样的
    public void show(){
        System.out.println(name+","+age);
    }

    // 抽象方法 ---
    // 因为所有子类吃东西的方法实现不一样
    public abstract void eat();

}
```

```
package com.itheima.demo14_抽象方法的概述和定义;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/6 14:53
 */
public class Dog extends Animal {
    @Override
    public void eat() {
        System.out.println("狗吃骨头...");
    }
}
```

```
package com.itheima.demo14_抽象方法的概述和定义;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/6 14:55
 */
public class Cat extends Animal {

    @Override
    public void eat() {
        System.out.println("猫吃鱼...");
    }
}
```

```
package com.itheima.demo14_抽象方法的概述和定义;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/6 14:47
 */
public class Test {
    public static void main(String[] args) {
        /*
            抽象方法:
                概述: 使用abstract修饰,并且没有方法体的方法
                格式: 修饰符 abstract 返回值类型 方法名(形参列表);
                抽象方法的使用场景:如果父类中某个方法,所有子类都有不同的实现,那么就可以把该方法定义为抽象方法
                抽象方法的作用: 强制要求子类重写

         */
```

```
        Dog d = new Dog();
        d.eat();

        Cat c = new Cat();
        c.eat();
    }
}
```

小结

- 抽象方法: 没有方法体,使用abstract修饰的方法就是抽象方法
- 抽象方法定义格式: 修饰符 abstract 返回值类型 方法名(形参列表);
- 使用场景:如果父类中某个方法,所有子类都有不同的实现,那么就可以把该方法定义为抽象方法
- 抽象方法的作用: 强制要求子类重写

知识点--3.3 抽象类的注意事项

目标

- 理解抽象类的特点

路径

- 抽象类的特点

讲解

- 抽象类不能被创建对象，就是用来做“父类”，被子类继承的。
- 抽象类不能被创建对象，但可以有“构造方法”——为成员变量初始化。
- 抽象类中可以没有抽象方法,但抽象方法必须定义在抽象类中
- 子类继承抽象类后,必须重写抽象类中所有的抽象方法,否则子类必须也是一个抽象类

```
package com.itheima.demo15_抽象类的注意事项;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/6 15:00
 */
abstract class Animal{
    private String name;
    private int age;

    public Animal() {
    }

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

    public void show(){
        System.out.println(name+","+age);
    }
```

```
    // 抽象类没有抽象方法
}
class Dog extends Animal{
    public Dog() {
        super();
    }

    public Dog(String name, int age) {
        super(name, age);
    }
}

abstract class Person{
    // 抽象方法
    public abstract void eat();
    public abstract void drink();

}

//普通子类继承抽象类后,必须重写抽象类中所有的抽象方法
class Student extends Person{

    @Override
    public void eat() {
        // ...
    }

    @Override
    public void drink() {
        // ...
    }
}

//抽象子类继承抽象类后,可以不用重写抽象类中的抽象方法
 abstract class Teacher extends Person{
    @Override
    public void eat() {
        // ... 可以重写...
    }
}


public class Test {
    public static void main(String[] args) {
        /*
            抽象类的注意事项:
            - 抽象类不能被创建对象，就是用来做“父类”，被子类继承的。
            - 抽象类不能被创建对象，但可以有“构造方法”--为成员变量初始化。
            - 抽象类中可以没有抽象方法,但抽象方法必须定义在抽象类中(抽象类中不一定有抽象方法,
但抽象方法一定在抽象类中)
            - 子类继承抽象类后,必须重写抽象类中所有的抽象方法,否则子类必须也是一个抽象类

         */
        // 抽象类不能被创建对象，就是用来做“父类”，被子类继承的。
        //Animal anl = new Animal();

        // 抽象类不能被创建对象，但可以有“构造方法”--为成员变量初始化。
        Dog d = new Dog("旺财", 2);
```

```
        d.show();// 旺财,2
    }
}
```

小结

- 抽象类不能创建对象,一般用来作为父类,供子类继承
- 抽象类不能被创建对象，但可以有“构造方法”——为成员属性初始化。
- 抽象类中可以没有抽象方法,但抽象方法必须定义在抽象类中
- 子类继承抽象类后,必须重写抽象类中所有的抽象方法,否则子类必须也是一个抽象类

知识点--3.4 模板设计模式

目标:

- 理解模板设计模式

路径:

- 设计模式概述
- 模板设计模式概述
- 模板模式的实现步骤
- 案例演示

讲解:

设计模式概述

- 设计模式就是解决一些问题时的固定思路，也就是代码设计思路经验的总结。

模板设计模式概述

- 针对某些情况,在父类中指定一个模板,然后根据具体情况,在子类中灵活的具体实现该模板

```
public abstract class Person{
    // 有方法体的方法: 通用模板
    public void sleep(){
        System.out.println("两眼一闭,就睡觉...");
    }

    // 没有方法体的方法(抽象方法):  填充模板(要子类重新实现的)
   public abstract void eat();
}
```

- **抽象类体现的就是模板设计思想，模板是将通用的东西在抽象类中具体的实现，而模板中不能决定的东西定义成抽象方法**，让使用模板（继承抽象类的类）的类去**重写抽象方法实现需求**

模板模式的实现步骤

- 定义抽象父类作为模板
- 在父类中定义"模板方法"--- 实现方法(通用模板)+抽象方法(填充模板)
- 子类继承父类,重写抽象方法(填充父类的模板)
- 测试类:

- 创建子类对象，通过子类调用父类的“实现的方法”+ “子类重写后的方法” e

案例演示

假如我现在需要定义新司机和老司机类，**新司机和老司机都有开车功能，开车的步骤都一样**，只是驾驶时的姿势有点不同，`新司机:开门,点火,双手紧握方向盘,刹车,熄火`，`老司机:开门,点火,右手握方向盘左手抽烟,刹车,熄火`。那么这个时候我们就可以将固定流程写到父类中，不同的地方就定义成抽象方法，让不同的子类去重写

分析:

- 司机类
  - 开车方法: 确定实现--通用模板
    - 开门
    - 点火
    - (姿势)
    - 刹车
    - 熄火
  - 姿势方法: 不确定实现--填充模板
- 新司机类继承司机类,重写姿势方法
- 老司机类继承司机类,重写姿势方法

```
package com.itheima.demo16_模板设计模式案例;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/6 15:30
 */
// 父类
public abstract class Driver {
    // 开车方法 通用模板
    public void driveCar(){
        System.out.println("开门");
        System.out.println("点火");
        // 姿势??
        ziShi();
        System.out.println("刹车");
        System.out.println("熄火");
    }

    // 姿势方法  填充模板
    public abstract void ziShi();
}
```

现在定义两个使用模板的司机:

```
public class NewDriver extends Driver {
    @Override
    public void ziShi() {
        System.out.println("双手紧握方向盘");
    }
}

public class OldDriver extends Driver {
    @Override
    public void ziShi() {
        System.out.println("右手握方向盘左手抽烟");
    }
}
```

编写测试类

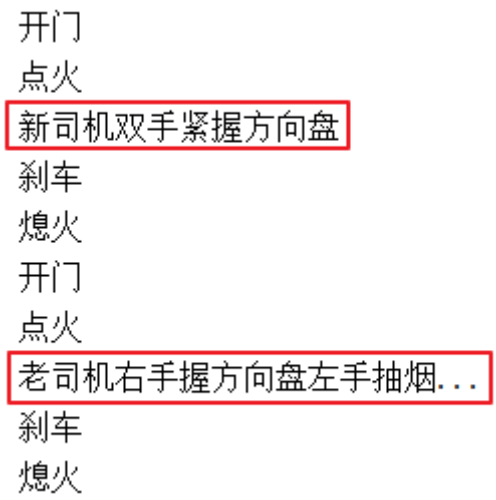
```
public class Test {

    public static void main(String[] args) {
        // 创建新司机对象
        NewDriver d1 = new NewDriver();
        d1.driveCar();

        // 创建老司机对象
        OldDriver d2 = new OldDriver();
        d2.driveCar();
    }

}
```

运行效果



可以看出，模板模式的优势是，模板已经定义了通用架构，使用者只需要关心自己需要实现的功能即可！非常的强大！

小结

- 定义一个抽象类作为父类
- 在抽象类中,该模板可以确定的功能,就定义成一个有方法体的方法,作为通用模板
- 在抽象类中,该模板不可以确定的功能,就定义成一个抽象方法,作为填充模板

- 让需要使用该模板的类,去继承该抽象类,填充模板\使用模板

第四章 final关键字

知识点-- final关键字的概述和使用

目标:

- final关键字的概述和使用

路径:

- final关键字的概述
- final关键字的使用

讲解:

final关键字的概述

final： 不可改变。可以用于修饰类、方法和变量。

- 类：被修饰的类，不能被继承。
- 方法：被修饰的方法，不能被重写。
- 变量：被修饰的变量，就只能赋值一次,不能被重新赋值。

final关键字的使用

修饰类

格式如下：

```
修饰符 final class 类名 {

}
例如:
public final class FinalClassFu {
}
public class FinalClassZi /*extends FinalClassFu*/ {
    // FinalClassFu类被final修饰了,所以不能被继承
}
```

查询API发现像 `public final class String` 、 `public final class Math` 、 `public final class Scanner` 等，很多我们学习过的类，都是被final修饰的，目的就是供我们使用，而不让我们所以改变其内容。

修饰方法

格式如下：

```
修饰符 final 返回值类型 方法名(参数列表){
    //方法体
}
```

重写被 `final` 修饰的方法，编译时就会报错。

```
public class FinalMethodFu {
    public final void show(){

    }
}
public class FinalMethodZi extends FinalMethodFu {

    /*@Override
    public void show() {

    }*/
    // 无法重写父类中的show方法,因为父类中的show方法被final修饰了
}
```

修饰变量

局部变量——基本类型

基本类型的局部变量，被final修饰后，只能赋值一次，不能再更改。代码如下：

```
public class FinalDemo1 {
    public static void main(String[] args) {
         // final修饰基本数据类型
        final int NUM = 10;
        // NUM = 20;// 编译报错,final修饰的变量只能赋值一次,不能重复赋值
    }
}
```

局部变量——引用类型

引用类型的局部变量，被final修饰后，只能指向一个对象，地址不能再更改。但是不影响对象内部的成员变量值的修改，代码如下：

```
public class FinalDemo2 {
    public static void main(String[] args) {
        // 引用类型
        final Student stu = new Student("张三",18);
        //stu = new Student("李四",19);// 编译报错
        stu.setAge(19);
    }
}
```

成员变量

成员变量涉及到初始化的问题，初始化方式有两种，只能二选一：

1. 显示初始化；

```
public class FinalVariable {
    final int NUM1 = 10;
}
```

2. 构造方法初始化。

```
public class FinalVariable {
    final int NUM2;
    public FinalVariable(int NUM2){
        this.NUM2 = NUM2;
    }
    public FinalVariable(){
        this.NUM2 = 10;
    }
}
```

> 被final修饰的常量名称，一般都有书写规范，所有字母都**大写**。

小结：

- final修饰类，类不能被继承。
- final修饰方法，方法不能被重写。
- final修饰变量，变量不能被改值,只能赋值一次.

第五章 static关键字

知识点-- static关键字

目标:

- 之前咋们写main方法的时候,使用过了一个static关键字,接下来我们来学习一下static关键字

路径:

- static关键字概述
- static关键字的使用

讲解:

1.1 static关键字概述

static是一个静态修饰符关键字，表示静态的意思,可以修饰成员变量和成员方法以及代码块。

1.2 static关键字的使用

static修饰成员变量

当 `static` 修饰成员变量时，该变量称为**类变量**。该类的每个对象都**共享**同一个类变量的值。任何对象都可以更改该类变量的值，但也可以在不创建该类的对象的情况下对类变量进行操作。

定义格式：

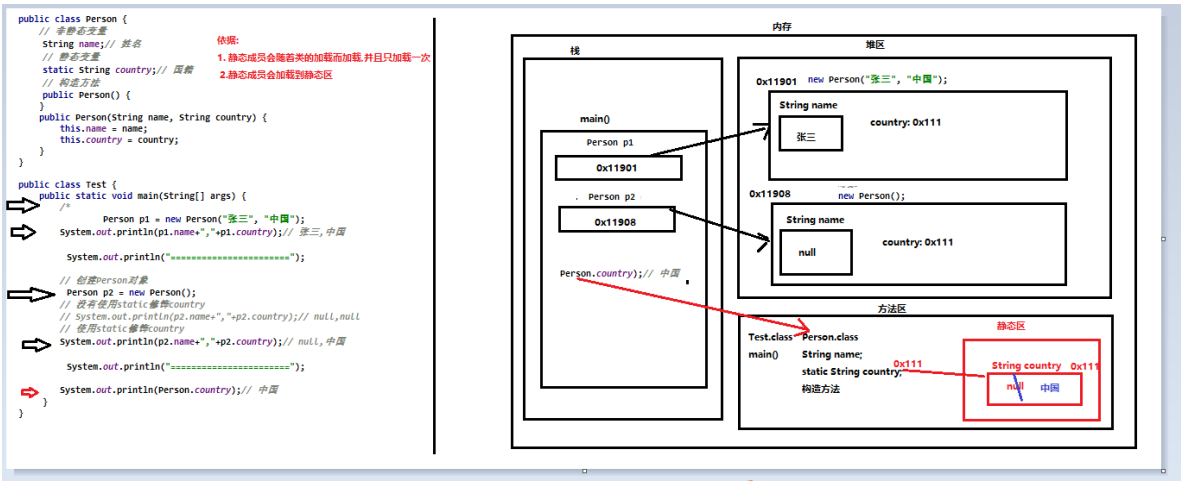
```
static 数据类型 变量名;
```

静态成员变量的访问方式:

```
对象名.静态成员变量名; 不推荐
类名.静态成员变量名;  推荐
```

案例:

```java
public class Person {
    // 非静态变量
    String name;// 姓名
    // 静态变量
    static String country;// 国籍

    // 构造方法

    public Person() {
    }

    public Person(String name, String country) {
        this.name = name;
        this.country = country;
    }
}

public class Test {
    public static void main(String[] args) {
        // 创建Person对象
        Person p1 = new Person("张三", "中国");
        System.out.println(p1.name+","+p1.country);// 张三,中国

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

        // 创建Person对象
        Person p2 = new Person();
        // 没有使用static修饰country
        // System.out.println(p2.name+","+p2.country);// null,null
        // 使用static修饰country
        System.out.println(p2.name+","+p2.country);// null,中国

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

        System.out.println(Person.country);// 中国
    }
}
```



static修饰成员方法

概述

- 被static修饰的方法会变成静态方法,也称为类方法,该静态方法可以使用类名直接调用。

格式

```
修饰符 static 返回值类型 方法名 (参数列表){
    // 执行语句
}
```

访问方式

```
对象名.方法名(实参);
类名.方法名(实参);    推荐
```

```
public class Person {
    // 非静态方法
    public void method1(){
        System.out.println("Person method1...");
    }

    // 静态方法
    public static void method2(){
        System.out.println("Person method2...");
    }
}
public class Test {
    public static void main(String[] args) {
        /*
            static修饰成员方法:
                格式:修饰符 static 返回值类型 方法名(形参列表){方法体}
                特点:被static修饰的成员方法叫做静态成员方法
                使用:
                    对象名.静态成员方法名(实参);
                    类名.静态成员方法名(实参); ----->推荐
         */
        Person p = new Person();
        p.method2();

        // 类名.静态成员方法名(实参);
        Person.method2();

    }
}
```

静态方法调用的注意事项:

- 静态方法中不能出现this关键字
- 静态方法中只能直接访问静态成员变量和静态成员方法
- 静态方法中不能直接访问非静态成员变量和非静态成员方法
- 非静态方法中可以直接访问一切成员变量和成员方法

```
package com.itheima.demo19_static修饰成员方法;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/6 16:33
```

```
 */
public class ChinesePeople {
    // 非静态成员变量
    String name;// 姓名
    // 静态成员变量
    static String country;// 国籍

    // 非静态方法
    public void method1(){
        System.out.println("非静态 method2方法");
    }

    public void method2(){
        // 非静态方法中可以直接访问一切成员变量和成员方法
        System.out.println(name);
        System.out.println(country);
        method1();
        method4();

        System.out.println("非静态 method2方法");
    }

    // 静态方法
    public static void method3(){
        //静态方法中不能直接访问非静态成员变量和非静态成员方法
        //System.out.println("非静态的成员变量:"+name);// 编译报错
        //method1();// 编译报错

        //静态方法中只能直接访问静态成员变量和静态成员方法
        System.out.println("静态成员变量:"+country);
        method4();

        // 静态方法中不能出现this关键字
        //System.out.println(this.name);// 编译报错
        //System.out.println(this.country);// 编译报错
        System.out.println("非静态 method3方法");
    }

    public static void method4(){
        System.out.println("非静态 method4方法");
    }
}

package com.itheima.demo19_static修饰成员方法;

/**
 * @Author: pengzhilin
 * @Date: 2020/9/6 16:32
 */
public class Test {
    public static void main(String[] args) {
        /*
            概述: 被static修饰的方法就是静态方法,否则就是非静态方法
            static修饰成员方法: 在方法的返回值类型前面加上static
            访问静态方法:
                对象名.静态方法名(参数);  不推荐
                类名.静态方法名(参数);    推荐
            注意事项:
```

```
                1.静态方法中只能直接访问静态成员变量和静态成员方法
                2.静态方法中不能直接访问非静态成员变量和非静态成员方法
                3.非静态方法中可以直接访问一切成员变量和成员方法
                4.静态方法中不能出现this关键字
         */
        ChinesePeople.method3();

        //ChinesePeople p = new ChinesePeople();
        //p.method2();

        /*// 对象名.静态方法名(参数);  不推荐
        ChinesePeople p1 = new ChinesePeople();
        p1.method3();
        p1.method4();

        // 类名.静态方法名(参数);    推荐
        ChinesePeople.method3();
        ChinesePeople.method4();*/
    }

}
```

以后开发中static的应用

概述

以后的项目中，通常会需要一些“全局变量”或者“全局的工具方法”，这些全局变量和方法，可以单独定义在一个类中，并声明为static(静态)的，可以很方便的通过类名访问

例如:

```
public class Utils {
    // "全局变量"
    public static final int WIDTH = 800;
    public static final int HEIGHT = 800;


    // "全局方法"
    // 找int数组中的最大值
    public static int getArrayMax(int[] arr){
        int max = arr[0];
        for (int i = 0; i < arr.length; i++) {
            if(arr[i] > max){
                max = arr[i];
            }
        }
        return max;
    }
}

public class Test {
    public static void main(String[] args) {
        /*
            以后的项目中，通常会需要一些“全局变量”或者“全局的工具方法”，这些全局变量和方法，
            可以单独定义在一个类中，并声明为static(静态)的，可以很方便的通过类名访问

            工具类
```

```
         */
        System.out.println(Utils.width);
        System.out.println(Utils.height);

        int[] arr = {23,34,545,56};
        System.out.println(Utils.getArrayMax(arr));
    }
}
```

小结:

```
static修饰成员方法:
    格式: 在返回值类型前面加static关键字
    使用: 类名.静态方法名(实参);
    注意事项:
        1.静态方法中不能出现this
        2.静态方法中只能直接访问静态成员变量和成员方法
        3.非静态方法中可以直接访问一切成员变量和成员方法
static修饰成员变量:
    格式: static 数据类型 变量名;
    使用; 类名.静态成员变量名
    特点; 被static修饰的变量会被该类的所有对象共享
```

总结

- 能够写出类的继承格式
  `public class 子类名 extends 父类名{}`

- 能够说出继承的特点
  子类拥有父类的所有成员变量和成员方法

- 能够说出子类调用父类的成员特点
  构造方法: 无法继承
  私有成员: 可以继承,但无法直接访问
  非私有成员: 可以继承,优先在子类中查找,如果有,就直接使用,如果没有,就去父类中找,依次类推...

- 够说出方法重写的概念
  子类中出现和父类一模一样的方法(返回值类型,方法名,参数列表),该方法就是重写的方法

- 能够说出this可以解决的问题
  this的使用:
  访问本类的成员变量; this.本类成员变量 区分同名的成员变量和局部变量
  访问本类的成员方法; this.本类成员方法(实参)
  访问本类的构造方法:
  空参构造:this();
  有参构造:this(实参);
  注意;
  1.this调用本类的构造方法需要放在第一行
  2.本类的构造方法不能同时使用this相互调用
  3.只能在本类的构造方法中调用

- 能够说出super可以解决的问题

super的使用:
访问父类的成员变量; super.父类成员变量　区分父子类中同名的成员变量
访问父类的成员方法; super.父类成员方法(实参)　区分父子类中同名的成员方法
访问父类的构造方法:
空参构造:super();
有参构造:super(实参);
注意;
1.super调用父类的构造方法需要放在第一行
2.子类构造方法默认会调用父类的空参构造方法
3.只能在子类的构造方法中调用父类的构造方法

- 描述抽象方法的概念
使用abstract修饰,并且没有方法体

- 写出抽象类的格式
```
public abstract class 类名{
    成员变量
    成员方法
    构造方法
    抽象方法
}
```
- 写出抽象方法的格式
在返回值类型前面加上abstract

- 能够说出父类抽象方法的存在意义
强制要求子类重写

- 描述final修饰的类的特点
不能被继承

- 描述final修饰的方法的特点
不能被重写

- 描述final修饰的变量的特点
只能赋值一次,不能重复赋值
- static关键字
static修饰成员方法:
格式: 在返回值类型前面加static关键字
使用: 类名.静态方法名(实参);
注意事项:
1.静态方法中不能出现this
2.静态方法中只能直接访问静态成员变量和成员方法
3.非静态方法中可以直接访问一切成员变量和成员方法
static修饰成员变量:
格式: static 数据类型 变量名;
使用; 类名.静态成员变量名
特点; 被static修饰的变量会被该类的所有对象共享