面向对象-Day02（继承、抽象类）

今日学习内容

- this关键字
- 构造器和setter方法的选用
- 继承思想
- 方法覆盖
- 抽象方法和抽象类
- Object类的常用方法

今日学习目标

- 掌握使用this解决二义性
- 了解如何选用构造器和setter方法设置对象内容
- 理解继承的作用
- 必须掌握继承的语法
- 了解子类可以继承父类那些成员
- 掌握方法覆盖的判断规则和如何覆盖方法
- 掌握super关键字的含义
- 掌握抽象方法的定义和特点
- 掌握抽象类的定义和特点
- 掌握如何覆盖Object类的toString方法
- 掌握equals方法和==的区别

2、封装思想

2.1 this 关键字(一)

2.1.1 this关键字（掌握）

之前说过，变量名称或方法参数名称，要见名知意，下列两个set方法的参数名，就显得太LOW了。

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

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

    public String getName() {
        return name;
    }

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

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

```
}
```

不就是设置名字和年龄吗，如果此时把参数名分别改为name和age。

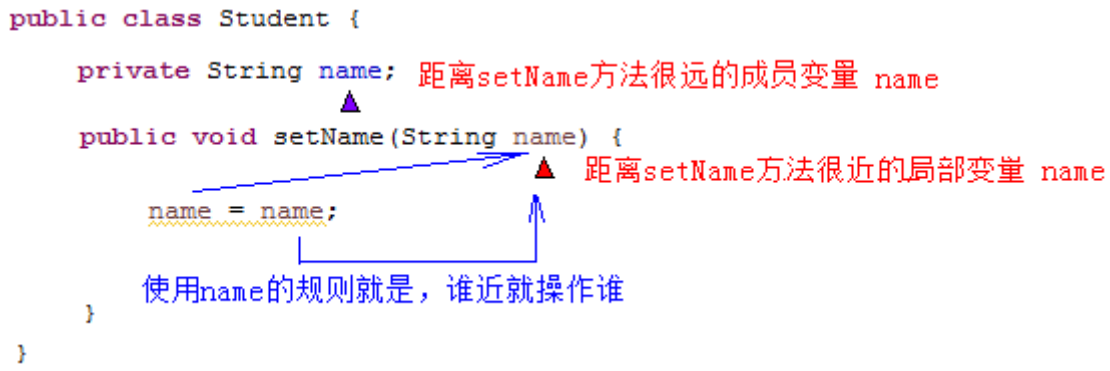
```
public class Student {
    private String name;
    private int age;

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

    public String getName() {
        return name;
    }

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

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

此时会发现参数根本就设置不进去，name和age打印出来都是各自的初始值，运行测试类的结果如下：

```
null,0
```

先回忆方法的参数属于局部变量这个结论，导致参数设置不进去的原因是：

局部变量和成员变量同名，此时在方法中调用变量时根据**就近原则**，优先使用局部变量，示意图如下。

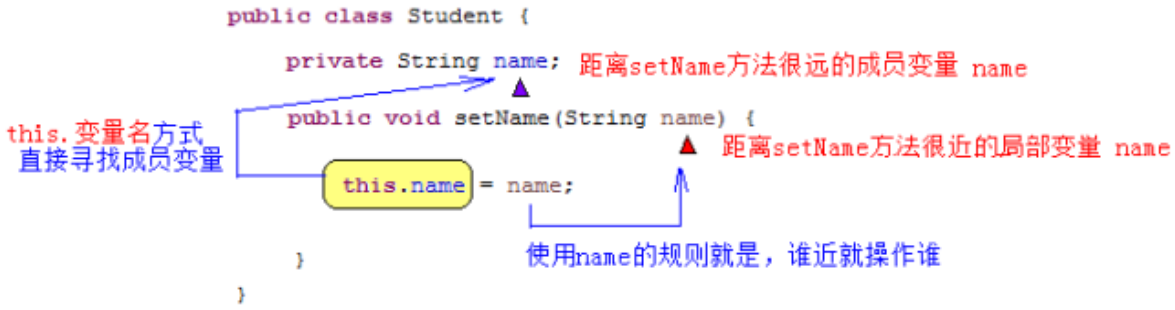


可以看出setName方法中两次使用的name，都是直接寻找距离自己最近的形参name，就相当于把参数name的值设置给参数name，根本就没有把参数值设置给成员变量。

> 提示：
>
> 当在一个作用域访问变量时，首先在当前作用域查找该变量，
> 如果能找到，不继续查找。
> 如果在当前作用域找不到该变量，尝试去上一层作用域查找，如果找到，停止查找，如果找不到，
> 继续上一层查找，依次类推。这整个过程形成一个查找链，这个称为作用域链。

该问题，更专业的叫法是**局部变量和成员变量存在二义性**，也就是变量名有歧义。为了解决该问题——有请this关键字。

使用 **this.变量名** 的语法，此时访问的就**是成员变量**，this的其他操作，后面再讲。



具体代码如下：

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

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

    public String getName() {
        return name;
    }

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

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

2.1.2 使用构造器还是setter方法（了解）

构造器和setter方法都可以给对象设置数据：

- 构造器，在创建对象的时候设置初始数据，只能初始化一次。
- setter方法，创建对象后再设置初始数据，可以设置多次。

3、继承思想

需求：使用面向对象的知识定义出老师（Teacher）、学生（Student）、员工（Employee）三个类：

- 老师：拥有名字、年龄、级别三个状态，有授课和休息两个功能
- 学生：拥有名字、年龄、学号三个状态，有学习和休息两个功能
- 员工：拥有名字、年龄、入职时间三个状态，有工作和休息两个功能

代码截图如下：

Teacher.java

```java
public class Teacher {
    private String name;
    private int age;
    private int level;//级别

    public void teach(){
        System.out.println("授课");
    }
    public void rest(){
        System.out.println("休息");
    }

    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 int getLevel() {
        return level;
    }

    public void setLevel(int level) {
        this.level = level;
    }
}
```

Student.java

```java
public class Student {
    private String name;
    private int age;
    private String sn;//学号

    public void study(){
        System.out.println("学习");
    }
    public void rest(){
        System.out.println("休息");
    }

    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 String getSn() {
        return sn;
    }

    public void setSn(String sn) {
        this.sn = sn;
    }
}
```

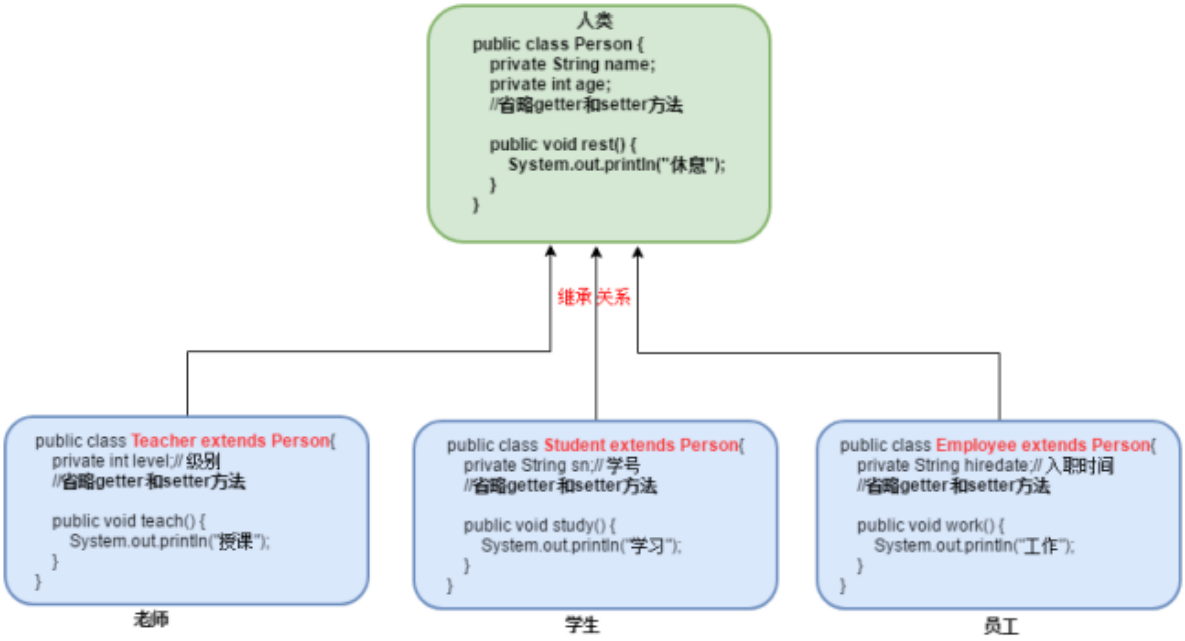
Employee.java

```java
public class Employee {
    private String name;
    private int age;
    private String hiredate;//入职日期

    public void work(){
        System.out.println("工作");
    }
    public void rest(){
        System.out.println("休息");
    }

    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 String getHiredate() {
        return hiredate;
    }

    public void setHiredate(String hiredate) {
        this.hiredate = hiredate;
    }
}
```

此时，发现三个类中的存在着大量的共同代码，而我们要考虑的就是如何解决代码重复的问题。

面向对象的继承思想，可以解决多个类存在共同代码的问题。

继承关系设计图：



记住几个概念：

- 被继承的类，称之为父类
- 继承父类的类，称之为子类
- 父类：存放多个子类共同的字段和方法
- 子类：存放自己**特有的**(独有的)字段和方法

经过继承后，子类拥有了父类的特征和行为，也即：**子类拥有了父类的字段和方法。**

3.1.继承语法（重点）

在java程序中，如果一个类需要继承另一个类，此时使用extends关键字。

```
public class 父类名 {
    // 存放多个子类共同的字段和方法
}

public class 子类名 extends 父类名 {
    // 存放自己特有的(独有的)字段和方法
}
```

注意：Java中类只支持单继承，但是支持多重继承。也就是说一个子类只能有一个直接的父类，父类也可以再有父类。

- 下面是错误的写法！ Java中的类只支持单继承。

```
class SuperClass1 { }
class SuperClass2 { }

// 错误用法
class SubClass extends SuperClass1,SuperClass2 {

}
```

- 下面代码是正确的。一个父类可以有多个子类。

```
class SuperClass{}
class SubClass1 extends SuperClass{ }
class SubClass2 extends SuperClass{ }
```

- 下面代码是正确的，支持多重继承。

```
class A {}
class B extends A {}
class C extends B
```

例如：大学生 extends 学生，学生 extends 人类，我们就说，大学生具有人类的特征和行为。

- Object类是Java语言的根类，任何类都是Object的子类，要么是直接子类，要么是间接子类（后讲）

```
public class  Person {

}
等价于
public class Person extends Object {

}
```

3.1.1. 继承操作（重点）

父类代码：

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

```
    public void rest() {
        System.out.println("休息");
    }

    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;
    }
}
```

子类代码：

```
public class Student extends Person{
    private String sn;  // 学号

    public void study() {
        System.out.println("学习");
    }

    public String getSn() {
        return sn;
    }

    public void setSn(String sn) {
        this.sn = sn;
    }
}
```

测试代码：

```
public class ExtendsDemo {
    public static void main(String[] args) {
        // 创建学生对象
        Student stu = new Student();
        // 设置字段信息
        stu.setName("will");     //继承于父类
        stu.setAge(17);          //继承于父类
        stu.setSn("s_123");

        // 调用方法
        stu.study();
        stu.rest();              //继承于父类
    }
}
```

3.1.2 继承相关的访问修饰符（了解）

关键字	本类	同包子类	同包其他类	不同包子类	不同包其他类
private	√	×	×	×	×
默认	√	√	√	×	×
protected	√	√	√	√	×
public	√	√	√	√	√

```
private ： 理解为私有的，其修饰的成员(字段和方法)只能在本类被访问，出了本类就不能被外界访问。
默认 ： 其修饰的成员在同包才能被访问，所以也称为包访问权限。
protected ： 理解为受保护的成员， ① 同包可访问 ② 只要存在继承关系都可以被访问。
public ： 公共的，在项目的任何地方都可以被访问。

访问权限的关系
private < 默认 < protected < public
```

3.1.3 子类可以继承到父类哪些成员（了解）

子类继承父类之后，可以拥有到父类的某一些成员（字段和方法），**关键还要根据访问修饰符来判断**：

- **如果父类中的成员使用public和protected修饰，子类都能继承.**
- 如果父类和子类在同一个包中，使用默认访问修饰的成员，此时子类可以继承到
- **如果父类中的成员使用private修饰，子类继承不到。private只能在本类中访问**
- **父类的构造器，子类也不能继承，因为构造器必须和当前的类名相同**

记住一句话：子类继承父类的非私有成员（字段和方法），但构造器除外。

```
/**
 * 解析
 * 子类继承父类的成员，由于父类的成员是私有的，不能被子类直接访问，所以，我们常常说私有成员子类继承不到。
 */
public class Father {
    private void test() { ... }
    public void showInfo() { ... }
}

public class Son extends Father {

}
// 思考：子类继承了父类的什么方法?
```

3.2. 方法覆盖（掌握）

子类继承了父类，可以拥有父类的部分方法和成员变量。**可是当继承父类的某个方法不适合子类本身的特征时，此时怎么办?** 比如鸵鸟（Ostrich）是鸟类（Bird）中的一个特殊品种，所以鸵鸟类是鸟类的一个子类，但是鸟类有飞翔的功能，但是对应鸵鸟，飞翔的行为显然不适合于它。

父类：

```
public class Bird {
    public void fly() {
        System.out.println("飞呀飞...");
    }
}
```

子类：

```
public class Ostrich extends Bird {

}
```

测试类：

```
public class OverrideDemo {
    public static void main(String[] args) {
        // 创建鸵鸟对象
        Ostrich os = new Ostrich();
        // 调用飞翔功能
        os.fly();
    }
}
```

运行结果：

```
飞呀飞...
```

上述代码从**语法是正确的，但从逻辑上是不合理的**，因为鸵鸟不能飞翔，此时怎么办？——方法覆盖操作。

3.2.1. 方法覆盖操作（重点掌握）

当子类存在一个和父类一模一样的方法时，我们就称之为子类覆盖了父类的方法，也称之为重写（override）。

那么我们就可以在子类方法体中，重写编写逻辑代码。

```
public class Ostrich extends Bird {
    public void fly() {
        System.out.println("扑扑翅膀,快速奔跑...");
    }
}
```

运行测试代码：

```
扑扑翅膀,快速奔跑...
```

重写方法的调用顺序：

通过子类对象调用方法时，先在子类中查找有没有对应的方法，若存在就执行子类的，若子类不存在就执行父类的，如果父类也没有，报错。

> 开发意识：什么时候用方法覆盖？

> 当子类继承父类的方法不能符合自身需要时，子类就可以根据自身需要对父类的同名方法进行覆盖。

方法覆盖的细节：(理解掌握)

1. 方法签名必须相同 (方法签名= 方法名 + 参数类型)
2. 子类方法的返回值类型：要么和父类方法的返回类型相同，要么是父类方法返回值类型的子类 (了解)
3. 子类方法的访问权限 >= 父类方法访问权限
    - 如果父类方法是private，子类方法不能覆盖。==> 覆盖建立在继承的基础上，没有继承，就不能覆盖。

上述的方法覆盖细节真多，记不住，那么记住下面这句话就万事OK了。

精华：直接拷贝父类中方法的定义粘贴到子类中，再重新编写子类方法体，打完收工！ (ctrl + o)

3.2.2. 覆盖中super关键字（掌握）

问题：当在子类中的某一个方法中需要去调用父类中**被覆盖的方法**，此时得使用super关键字。

```
public class Ostrich extends Bird {

    public void fly() {
        System.out.println("扑扑翅膀,快速奔跑...");
    }

    public void sayHi() {
        super.fly(); // 调用父类被覆盖的方法
        fly();       // 调用本类中的方法
    }
}
```

如果调用被覆盖的方法不使用super关键字，此时调用的是本类中的方法。

- super关键字表示父类对象的意思，更多的操作，后面再讲。
- super.fly() 可以翻译成调用父类对象的fly方法。

综合案例：使用继承和方法覆盖知识完成 ，要求：

- 定义Person类 和 Teacher类，Teacher 继承 Person 类。
- 输出老师的基本信息（打印姓名，年龄，级别信息）。

定义 Person 类

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

    public Person() { }

    // setter and getter ( 省略 )
```

```
    // 输出人类对象的基本信息
    public void showInfo(){
        System.out.println("name:" + name);
        System.out.println("age:" + age);
    }
}
```

定义 Teacher 类继承 Person

```
public class Teacher extends Person{
    private int level;

    public Teacher(){ }

    // setter and getter ( 省略 )

    @Override
    public void showInfo() {
        super.showInfo();
        System.out.println("level = " + level);
    }
}
```

定义测试类

```
 public class OverrideDemo {
    public static void main(String[] args) {
        Teacher teacher = new Teacher();
        teacher.setName("kallen");
        teacher.setAge(20);
        teacher.setLevel(1);

        // 调用子类覆盖的方法
        teacher.showInfo();
    }
}
```

3.3. 抽象方法和抽象类（掌握）

需求：求圆（Circle）和矩形（Rectangle）两种图形的面积。

分析：无论是圆形还是矩形，还是其他形状的图形，只要是图形，都有面积，也就说图形都有求面积的功能，那么我们就可以把定义一个图形（Graph）的父类，该类拥有求面积的方法，但是作为图形的概念，而并不是某种具体的图形，那么怎么求面积是不清楚的，姑且先让求面积的getArea方法返回0。

父类代码：

```
public class Graph {
    public double getArea() {
        return 0.0;
    }
}
```

子类代码（圆形）：

```
public class Circle extends Graph {
    private int r;  //半径

    public void setR(int r) {
        this.r = r;
    }

    public double getArea() {
        return 3.14 * r * r;
    }
}
```

子类代码（矩形）：

```
public class Rectangle extends Graph {
    private int width;  // 宽度
    private int height; // 高度

    public void setWidth(int width) {
        this.width = width;
    }
    public void setHeight(int height) {
        this.height = height;
    }

    public double getArea() {
        return width * height;
    }
}
```

测试代码：

```
public class GraphDemo {
    public static void main(String[] args) {
        // 圆
        Circle c = new Circle();
        c.setR(10);
        double ret1 = c.getArea();
        System.out.println("圆的面积：" + ret1);

        // 矩形
        Rectangle r = new Rectangle();
        r.setWidth(5);
        r.setHeight(4);
        double ret2 = r.getArea();
        System.out.println("矩形的面积：" + ret2);
    }
}
```

运行结果如下：

```
圆的面积：314.0
矩形的面积：20.0
```

3.3.1. 引出抽象方法（了解）

问题1：既然不同的图形求面积的算法是不同的，所以**必须**要求每一个图形子类去覆盖getArea方法，如果没有覆盖，应该以语法报错的形式做提示。

问题2：在Graph类中的getArea方法的**方法体没有任何存在意义**，因为不同图形求面积算法不一样，子类必须要覆盖getArea方法。

要满足上述对方法的要求，就得使用abstract来修饰方法，被abstract修饰的方法具备两个特征：

- 该方法没有方法体
- 要求子类必须覆盖该方法

这种方法，我们就称之为**抽象方法。**

3.3.2. 抽象方法和抽象类（重点掌握）

使用abstract修饰的方法，称为抽象方法。

```
public abstract 返回类型 方法名（参数）;
```

抽象方法的特点：

- 使用abstract修饰，没有方法体，留给子类去覆盖
- 抽象方法必须定义在抽象类中

使用abstract修饰的类，成为抽象类。

```
public abstract class 类名 {

}
```

一般的，抽象类以Abstract作为类名前缀，如AbstractGraph，一看就能看出是抽象类。

抽象类的特点：

- 抽象类不能创建对象，调用没有方法体的抽象方法没有任何意义
- 抽象类中可以同时拥有抽象方法和普通方法
- 抽象类要有子类才有意义，**子类必须实现父类的所有的抽象方法**，除非子类也是抽象类。

> 提示：
>
> 子类覆盖抽象父类的抽象方法，更专业的说法叫做实现（implement）

父类代码：

```
public abstract class AbstractGraph {
    public abstract double getArea();   // 没有方法体
}
```

子类代码：

```
public class Circle extends AbstractGraph {
    private int r;// 半径

    public void setR(int r) {
        this.r = r;
    }

    public double getArea() {   //覆盖父类抽象方法
        return 3.14 * r * r;    //编写方法体
    }
}
```

测试类没有改变。

3.4. Object类和常用方法（掌握）

Object本身表示对象类的意思，是Java中的根类，要么是一个类的直接父类，要么就是一个类的间接父类。

```
public class  A {

}
其实等价于
public class  A extends Object{

}
```

因为所有类都是Object类的子类， 所有类的对象都可以调用Object类中的方法，常见的方法：

- boolean equals(Object obj)：拿当前调用该方法的对象和参数obj做比较

在Object类中的equals方法和“ == ”符号相同，都是比较对象是否是同一个的存储地址。

```
public class ObjectDemo {
    public static void main(String[] args) {
        //创建Person对象p1
        Person p1 = new Person();
        //创建Person对象p2
        Person p2 = new Person();

        //比较p1和p2的内存地址是否相同
        boolean ret1 = p1 == p2;
        boolean ret2 = p1.equals(p2);
        System.out.println(ret1);   //false
        System.out.println(ret2);   //false
    }
}
```

官方建议：每个类都应该覆盖equals方法去比较我们关心的数据，而不是内存地址。

- String toString()：表示把对象中的字段信息转换为字符串格式

打印对象时其实打印的就是对象的toString方法

```
Person p = new Person();
p.setName("will");
p.setAge(17);
System.out.println(p);
System.out.println(p.toString());
```

其中：

```
System.out.println(p);
等价于
System.out.println(p.toString());
```

打印格式如：

```
cn.wolfcode._04_object.Person@15db9742
```

默认情况下打印的是对象的hashCode值（也就是内存地址），但是我们更关心对象中字段存储的数据。

官方建议：应该每个类都应该覆盖toString返回我们关心的数据，如：

```
public class Person {
    private String name;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

    public String toString() {
        return "Person [name=" + name + ", age=" + age + "]";
    }
}
```

此时打印对象，看到的是该对象的字段信息。

```
Person [name=will, age=17]
```

可以通过IDEA生成toString方法，刚开始一定要手写。

== 符号到底比较的是什么：

- 比较基本数据类型：比较两个值是否相等
- 比较对象数据类型：比较两个对象是否是同一块内存空间

每一次使用new关键字，都表示在堆中创建一块新的内存空间。