面向对象第三天：

回顾：

1. 继承：
    - 代码复用、extends、超类:共有的 派生类:特有的
    - 派生类可以访问派生类的和超类的，但超类不能访问派生类的
    - 单一继承，具有传递性
    - java规定：构造派生类之前必须先构造超类
        - 若派生类的构造方法中没有调用超类构造方法，则默认super()调超类无参构造方法
        - 若派生类的构造方法中调用了超类构造方法，则不再默认提供

        > super()调用超类构造方法，必须位于派生类构造方法的第1行

2. super：指代当前对象的超类对象
    - super.成员变量名----------访问超类的成员变量
    - super.方法名()----------调用超类的方法
    - super()----------调用超类的构造方法
3. 方法的重写(override/overriding)：
    - 发生在父子类中，方法名相同，参数列表相同
    - 重写方法被调用时，看对象的类型

精华笔记：

1. 抽象方法：
    - 由abstract修饰
    - 只有方法的定义，没有具体的实现(连{}都没有)
2. 抽象类：
    - 由abstract修饰
    - 包含抽象方法的类必须是抽象类，但不包含抽象方法的类也可以声明为抽象类
    - 抽象类不能被实例化(new对象)
    - 抽象类是需要被继承的，派生类：
        - 必须重写所有抽象方法----------变不完整为完整
        - 也声明为抽象类----------一般不这么用
    - 抽象类的意义：
        - 封装共有的属性和行为----------代码复用
        - 可以包含抽象方法，为所有派生类统一入口(名字统一)，强制必须重写
3. 接口：
    - 是一种引用数据类型
    - 由interface定义

- 只能包含抽象方法(常量、默认方法、静态方法、私有方法------暂时搁置)
- 不能被实例化
- 接口是需要被实现/继承的，实现类/派生类：必须重写接口中的所有抽象方法
  - 注意：重写接口中的方法时，必须加public(先记住)
- 一个类可以实现多个接口，用逗号分隔。若又继承又实现时，应先继承后实现
- 接口可以继承接口

4. 引用类型数组：------记住它和基本类型数组的两种区别即可
   - 区别1：给引用类型数组的元素赋值时，需要new个对象
   - 区别2：访问引用类型数组的元素的属性/行为时，需要打点访问
5. 综合练习：达内员工管理系统

笔记：

1. 抽象方法：
   - 由abstract修饰
   - 只有方法的定义，没有具体的实现(连{}都没有)
2. 抽象类：
   - 由abstract修饰
   - 包含抽象方法的类必须是抽象类，但不包含抽象方法的类也可以声明为抽象类
   - 抽象类不能被实例化(new对象)
   - 抽象类是需要被继承的，派生类：
     - 必须重写所有抽象方法----------变不完整为完整
     - 也声明为抽象类----------一般不这么用
   - 抽象类的意义：
     - 封装共有的属性和行为----------代码复用
     - 可以包含抽象方法，为所有派生类统一入口(名字统一)，强制必须重写

```
public abstract class Animal {
    String name;
    int age;
    String color;
    Animal(String name,int age,String color){
        this.name = name;
        this.age = age;
        this.color = color;
    }

    void drink(){
        System.out.println(color+"色的"+age+"岁的"+name+"正在喝
水...");
    }
    abstract void eat();
}

public class Dog extends Animal{
    Dog(String name,int age,String color){
```

```
        super(name,age,color);
    }

    void lookHome(){
        System.out.println(color+"色的"+age+"岁的狗狗"+name+"正在看
家...");
    }
    void eat(){
        System.out.println(color+"色的"+age+"岁的狗狗"+name+"正在吃肯
头...");
    }
}

public class Chick extends Animal {
    Chick(String name,int age,String color){
        super(name,age,color);
    }
    void layEggs(){
        System.out.println(color+"色的"+age+"岁的小鸡"+name+"正在下
蛋...");
    }
    void eat(){
        System.out.println(color+"色的"+age+"岁的小鸡"+name+"正在吃小
米...");
    }
}

public class Fish extends Animal {
    Fish(String name,int age,String color){
        super(name,age,color);
    }

    void eat(){
        System.out.println(color+"色的"+age+"岁的小鱼"+name+"正在吃小
虾...");
    }
}
```

3. 接口：
    - 是一种引用数据类型
    - 由interface定义
    - 只能包含抽象方法(常量、默认方法、静态方法、私有方法------暂时搁置)

```
interface Inter {
    abstract void show();
    void test(); //接口中的方法默认都是抽象的，前面默认有abstract
    //void say(){} //编译错误，抽象方法不能有方法体
}
```

    - 不能被实例化

```
public class InterfaceDemo {
    public static void main(String[] args) {
        //Inter o = new Inter(); //编译错误，接口不能被实例化
    }
}
```

- 接口是需要被实现/继承的，实现类/派生类：必须重写接口中的所有抽象方法
  - 注意：重写接口中的方法时，必须加public(先记住)

```
interface Inter {
    abstract void show();
    void test(); //接口中的方法默认都是抽象的，前面默认有abstract
    //void say(){} //编译错误，抽象方法不能有方法体
}
class InterImpl implements Inter {
    public void show(){ //重写接口中的抽象方法时，必须加public
    }
    public void test(){
    }
}
```

- 一个类可以实现多个接口，用逗号分隔。若又继承又实现时，应先继承后实现

```
//演示接口的多实现
interface Inter1{
    void show();
}
interface Inter2{
    void test();
}
abstract class Aoo{
    abstract void say();
}
class Boo extends Aoo implements Inter1,Inter2{
    public void show(){}
    public void test(){}
    void say(){}
}
```

- 接口可以继承接口

```
//演示接口继承接口
interface Inter3{
    void show();
}
interface Inter4 extends Inter3{
    void test();
}
class Coo implements Inter4{
    public void test(){}
    public void show(){}
}
```

```
public abstract class Animal {
    String name;
    int age;
    String color;
    Animal(String name,int age,String color){
        this.name = name;
        this.age = age;
        this.color = color;
    }

    void drink(){
        System.out.println(color+"色的"+age+"岁的"+name+"正在喝水...");
    }
    abstract void eat();
}

public interface Swim {
    /** 游泳 */
    void swim();
}

public class Dog extends Animal implements Swim {
    Dog(String name,int age,String color){
        super(name,age,color);
    }

    void lookHome(){
        System.out.println(color+"色的"+age+"岁的狗狗"+name+"正在看家...");
    }
    void eat(){
        System.out.println(color+"色的"+age+"岁的狗狗"+name+"正在吃肯
头...");
    }
    public void swim(){
        System.out.println(color+"色的"+age+"岁的狗狗"+name+"正在游泳...");
    }
}

public class Fish extends Animal implements Swim {
    Fish(String name,int age,String color){
        super(name,age,color);
    }

    void eat(){
        System.out.println(color+"色的"+age+"岁的小鱼"+name+"正在吃小
虾...");
    }
    public void swim(){
        System.out.println(color+"色的"+age+"岁的小鱼"+name+"正在游泳...");
    }
}

public class Chick extends Animal {
    Chick(String name,int age,String color){
        super(name,age,color);
```

```
    }
    void layEggs(){
        System.out.println(color+"色的"+age+"岁的小鸡"+name+"正在下蛋...");
    }
    void eat(){
        System.out.println(color+"色的"+age+"岁的小鸡"+name+"正在吃小
米...");
    }
}
public class SwimTest {
    public static void main(String[] args) {
        Dog dog = new Dog("小黑", 2, "黑");
        dog.eat();      //Dog类重写之后的
        dog.drink();    //复用Animal类的
        dog.swim();     //Dog类重写之后的
        dog.lookHome(); //Dog类所特有的

        Chick chick = new Chick("小白", 1, "白");
        chick.eat();     //Chick类重写之后的
        chick.drink();   //复用Animal类的
        chick.layEggs(); //Chick类所特有的

        Fish fish = new Fish("小金", 1, "金");
        fish.eat();   //Fish类重写之后的
        fish.drink(); //复用Animal类的
        fish.swim();  //Fish类重写之后的

    }
}
```

4. 引用类型数组：------记住它和基本类型数组的两种区别即可
    - 区别1：给引用类型数组的元素赋值时，需要new个对象
    - 区别2：访问引用类型数组的元素的属性/行为时，需要打点访问

```
public class RefArrayDemo {
    public static void main(String[] args) {
        Dog[] dogs = new Dog[3];
        dogs[0] = new Dog("小黑",2,"黑");
        dogs[1] = new Dog("小白",1,"白");
        dogs[2] = new Dog("小灰",3,"灰");
        System.out.println(dogs[0].name); //输出第1只狗狗的名字
        dogs[1].age = 4; //修改第2只狗狗的年龄为4岁
        dogs[2].swim(); //第3只狗狗在游泳
        for(int i=0;i<dogs.length;i++){ //遍历dogs数组
            System.out.println(dogs[i].name); //输出每只狗狗的名字
            dogs[i].eat(); //每只狗狗吃饭
        }

        Chick[] chicks = new Chick[2];
        chicks[0] = new Chick("小花",1,"花");
        chicks[1] = new Chick("大花",2,"白");
        for(int i=0;i<chicks.length;i++){ //遍历chicks数组
            System.out.println(chicks[i].name);
            chicks[i].layEggs();
```

```
        }

        Fish[] fish = new Fish[4];
        fish[0] = new Fish("小金",2,"金");
        fish[1] = new Fish("大金",4,"白");
        fish[2] = new Fish("小绿",1,"绿");
        fish[3] = new Fish("小红",3,"红");
        for(int i=0;i<fish.length;i++){ //遍历fish数组
            System.out.println(fish[i].color);
            fish[i].drink();
        }

        /*
        //声明Dog型数组dogs，包含3个元素，每个元素都是Dog类型，默认值为null
        Dog[] dogs = new Dog[3];
        //声明Chick型数组chicks，包含3个元素，每个元素都是Chick类型，默认值为
null
        Chick[] chicks = new Chick[3];
        //声明Fish型数组fish，包含2个元素，每个元素都是Fish类型，默认值为null
        Fish[] fish = new Fish[2];
         */
    }
}
```

5. 综合练习：达内员工管理系统

需求：
1)教研总监:名字、年龄、工资、上班打卡()、下班打卡()、完成工作()、
解决企业问题()、培训员工()、编辑书籍()
2)讲师:名字、年龄、工资、上班打卡()、下班打卡()、完成工作()、
解决企业问题()、培训员工()、编辑书籍()
3)项目经理:名字、年龄、工资、上班打卡()、下班打卡()、完成工作()、
编辑书籍()
4)班主任:名字、年龄、工资、上班打卡()、下班打卡()、完成工作()

设计：
1)雇员超类:名字、年龄、工资、上班打卡(){}、下班打卡(){}、abstract 完成工作()
2)企业顾问接口： 解决企业问题()、培训员工()
3)技术作者接口： 编辑书籍()
4)教研总监类，继承雇员超类，实现企业顾问接口和技术作者接口： 重写4个抽象方法
5)讲师类，继承雇员超类，实现企业顾问接口和技术作者接口： 重写4个抽象方法
6)项目经理类，继承雇员超类，实现技术作者接口： 重写2个抽象方法
7)班主任类，继承雇员超类： 重写1个抽象方法

补充：

1. 设计规则：----------适合初学者
   - 将所有派生类共有的属性和行为，抽到超类中----------抽共性
   - 若派生类的行为/代码都一样，设计普通方法

     若派生类的行为/代码不一样，设计抽象方法

- 将部分派生类共有的行为，抽到接口中
  - 接口对继承单根性的扩展----------实现多继承
  - 接口是一种标准、规范，若实现了某接口就具备某个的功能，若不实现接口就不具备那个功能----------后期才能理解得更好

2. 类间关系：
   - 类和类------------------------------继承
   - 接口和接口------------------------继承
   - 类和接口--------------------------实现
3. null：表示空，没有指向任何对象。
   - 若引用的值为null，则该引用不能再进行任何操作了，若操作则发生NullPointerException空指针异常。
4. 明日单词：

```
1)polymorphic:多态
2)cast:转换
3)Master:主人
4)feed:喂
5)instanceof:实例
6)Inner:内部
7)Outer:外部
8)anonymous:匿名
```