面向对象第四天：

回顾：

1. 抽象方法：
    - 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：
    - abstract修饰，包含抽象方法的类必须是抽象类，不能被实例化
    - 需要被继承，派生类：重写所有抽象方法
    - 意义：代码复用，可以包含抽象方法，为派生类提供统一的入口(名字一致)，强制必须重写
3. 接口：
    - interface定义，抽象方法，不能被实例化，
    - 需要被实现，实现类：必须重写所有抽象方法
    - 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：
    - 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：若想访问引用类型数组元素的属性/方法，需要通过元素去打点来调用

精华笔记：

1. 多态：多种形态
    - 向上造型/自动类型转换：
        - 超类型的引用指向派生类的对象
        - 能点出来什么，看引用的类型----------这是规定，记住它
    - 向下转型/强制类型转换，成功的条件只有如下两种：
        - 引用所指向的对象，就是该类型
        - 引用所指向的对象，实现了该接口或继承了该类
    - 强转时若不符合如上条件，则发生ClassCastException类型转换异常

    > 建议：在强转之前先通过instanceof来判断引用的对象是否是该类型
    >
    > 注意：instanceof返回boolean结果，它为true的条件就是强转成功的条件
    >
    > 何时需要强转：若想访问的属性/行为在超类中没有，则需要强制类型转换

2. 成员内部类：应用率低，了解
    - 类中套类，外面的称为外部类，里面的称为内部类
    - 内部类通常只服务于外部类，对外不具备可见性
    - 内部类对象通常在外部类中创建
    - 内部类可以直接访问外部类的成员，在内部类中有个隐式的引用指向创建它的外部类对象

    > 隐式的引用：外部类名.this

    - 何时用：若A类(Baby)只让B类(Mama)用，并且A类(Baby)还想访问B类(Mama)的成员时，可以设计成员内部类

3. 匿名内部类：应用率高，掌握
    - 何时用：若想创建一个派生类的对象，并且对象只创建一次，可以设计为匿名内部类，可以大大简化代码
    - 注意：匿名内部类中不能修改外面局部变量的值
    - 小面试题：
        - 问：内部类有独立的.class吗?
        - 答：有
4. package和import：
    - package：声明包
        - 作用：避免类的命名冲突
        - 规定：同包中的类不能同名，但不同包中的类可以同名。
        - 类的全称：包名.类名。包名常常有层次结构
        - 建议：包名所有字母都小写
    - import：导入类
        - 同包中的类可以直接访问，但不同包中的类不能直接访问，若想访问：
            - 先import导入类，再访问类----------建议
            - 类的全称----------太繁琐、不建议

笔记：

1. 多态：多种形态
    - 向上造型/自动类型转换：
        - 超类型的引用指向派生类的对象
        - 能点出来什么，看引用的类型----------这是规定，记住它
    - 向下转型/强制类型转换，成功的条件只有如下两种：
        - 引用所指向的对象，就是该类型
        - 引用所指向的对象，实现了该接口或继承了该类
    - 强转时若不符合如上条件，则发生ClassCastException类型转换异常

    > 建议：在强转之前先通过instanceof来判断引用的对象是否是该类型
    >
    > 注意：instanceof返回boolean结果，它为true的条件就是强转成功的条件
    >
    > 何时需要强转：若想访问的属性/行为在超类中没有，则需要强制类型转换

```
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 Master {
    void feed(Animal animal){ //喂动物
        animal.eat();
    }
}

package ooday04;
/**
```

```
 * 演示多态
 */
public class Test {
    public static void main(String[] args) {
        /*
        Animal o = new Dog("小黑",2,"黑"); //向上造型
        Dog g = (Dog)o;   //引用o所指向的对象，就是Dog类型
        Swim s = (Swim)o; //引用o所指向的对象，实现了Swim接口
        //Fish f = (Fish)o; //运行时会发生ClassCastException类型转换异常

        System.out.println(o instanceof Dog);  //true
        System.out.println(o instanceof Swim); //true
        System.out.println(o instanceof Fish); //false
        */

        /*
        Animal o1 = new Dog("小黑",2,"黑");
        //o1能强转为:Dog,Swim,Animal

        Animal o2 = new Fish("小黑",2,"黑");
        //o2能强转为:Fish,Swim,Animal

        Animal o3 = new Chick("小黑",2,"黑");
        //o3能强转为:Chick,Animal
        */

        /*
        //演示向上造型(多态)的第2点应用:
        Master master = new Master();
        Dog dog = new Dog("小黑",2,"黑");
        Chick chick = new Chick("小花",3,"花");
        Fish fish = new Fish("小金",1,"金");
        master.feed(dog); //在传参的同时，系统自动做了向上造型
        master.feed(chick);
        master.feed(fish);
        */

        //演示向上造型(多态)的第1点应用:
        //Animal o = new Animal(); //编译错误，抽象类不能被实例化
        Animal[] animals = new Animal[5];
        animals[0] = new Dog("小黑",2,"黑"); //向上造型
        animals[1] = new Dog("小白",1,"白");
        animals[2] = new Fish("小金",1,"金");
        animals[3] = new Fish("小花",2,"花");
        animals[4] = new Chick("小灰",3,"灰");
        for(int i=0;i<animals.length;i++){ //遍历所有动物
            System.out.println(animals[i].name); //输出每个动物的名字
            animals[i].eat();   //每个动物吃饭
            animals[i].drink(); //每个动物喝水

            if(animals[i] instanceof Dog){
                Dog dog = (Dog)animals[i];
                dog.lookHome();
            }
            if(animals[i] instanceof Chick){
```

```
                Chick chick = (Chick)animals[i];
                chick.layEggs();
            }
            if(animals[i] instanceof Swim){ //适用于所有实现Swim接口的(会游泳的)
                Swim s = (Swim)animals[i];
                s.swim();
            }
        }
    }
}
```

2. 成员内部类：应用率低，了解
    - 类中套类，外面的称为外部类，里面的称为内部类
    - 内部类通常只服务于外部类，对外不具备可见性
    - 内部类对象通常在外部类中创建
    - 内部类可以直接访问外部类的成员，在内部类中有个隐式的引用指向创建它的外部类对象

        > 隐式的引用：外部类名.this

    - 何时用：若A类(Baby)只让B类(Mama)用，并且A类(Baby)还想访问B类(Mama)的成员时，可以设计成员内部类

```
package ooday04;
//成员内部类
public class InnerClassDemo {
    public static void main(String[] args) {
        Mama m = new Mama();
        //Baby b = new Baby(); //编译错误，内部类对外不具备可见性
    }
}

class Mama{ //外部类
    String name;
    void create(){
        Baby b = new Baby(); //内部类对象通常在外部类中创建
    }
    class Baby{ //内部类
        void show(){
            System.out.println(name); //简写
            System.out.println(Mama.this.name); //完整写法,Mama.this指外部类对象
            //System.out.println(this.name); //编译错误，this指当前Baby对象
        }
    }
}
```

3. 匿名内部类：应用率高，掌握
    - 何时用：若想创建一个派生类的对象，并且对象只创建一次，可以设计为匿名内部类，可以大大简化代码
    - 注意：匿名内部类中不能修改外面局部变量的值
    - 小面试题：

- 问：内部类有独立的.class吗?
- 答：有

```
package ooday04;
//匿名内部类
public class AnonInnerClassDemo {
    public static void main(String[] args) {
        //1)创建了Aoo的一个派生类，但是没有名字
        //2)为该派生类创建了一个对象，名为o1，向上造型为Aoo类型
        //  ----new Aoo(){};是在创建Aoo的派生类对象
        //3)大括号中的为派生类的类体
        Aoo o1 = new Aoo(){};

        //1)创建了Aoo的一个派生类，但是没有名字---另一个派生类了
        //2)为该派生类创建了一个对象，名为o2，向上造型为Aoo类型
        //3)大括号中的为派生类的类体
        Aoo o2 = new Aoo(){};

        int num = 5;
        num = 6;
        //1)创建了Boo的一个派生类，但是没有名字
        //2)为该派生类创建了一个对象，名为o3，向上造型为Boo类型
        //3)大括号中的为派生类的类体
        Boo o3 = new Boo(){
            void show(){ //重写Boo类的show()方法
                System.out.println("showshow");
                //num = 55; //编译错误，匿名内部类中不能修改外面局部变量的值
            }
        };
        o3.show(); //通过派生类对象o3来调用派生类类体中的show()方法
    }
}

abstract class Boo{
    abstract void show();
}

abstract class Aoo{
}
```

4. package和import：
   - package：声明包
     - 作用：避免类的命名冲突
     - 规定：同包中的类不能同名，但不同包中的类可以同名。
     - 类的全称：包名.类名。包名常常有层次结构
     - 建议：包名所有字母都小写
   - import：导入类
     - 同包中的类可以直接访问，但不同包中的类不能直接访问，若想访问：
       - 先import导入类，再访问类----------建议
       - 类的全称----------太繁琐、不建议

补充：

1. 多态的实际应用：
    - 将不同对象(狗、鱼、鸡)统一封装到一个数组(动物数组)中来访问，实现代码复用
    - 将超类型(Animal)作为参数或返回值类型，传递或返回派生类(Dog/Fish/Chick)对象，以扩大方法的应用范围(所有Animal)，实现代码复用
2. 隐式的引用：
    - this：指代当前对象
    - super：指代当前对象的超类对象
    - 外部类名.this：指代当前对象的外部类对象
3. 明日单词：

```
1)public:公开的
2)private:私有的
3)protected:受保护的
4)final:最终的
5)static:静态的
6)enum:枚举
7)season:季节
8)PI:圆周率
9)spring:春
10)summer:夏
11)autumn:秋
12)winter:冬
13)value:值
```