接口和内部类

一 内容回顾（列举前一天重点难点内容）

1.1 教学重点:

```
1.掌握多态的基本概念
2.掌握多态的优缺点
3.掌握多态中的向上转型,向下转型
4.掌握抽象类的基本概念
5.掌握抽象类的基本使用
```

1.2 教学难点:

```
1.多态思想的理解
    理解思想不是一朝一夕的事情,大家可以先学会基本的语法,使用,再慢慢的不断深入理解
2.多态中instanceof的理解使用
```

二 教学目标

```
1.掌握接口的基本概念
2.掌握接口的基本使用
3.掌握接口中的多态
4.掌握内部类的分类
5.掌握内部类的基本使用
6.了解接口的新特性
7.了解匿名内部类
```

三 教学导读

3.1. 接口

宏观上来讲，接口是一种标准。例如，我们生活中常见的USB接口。电脑通过USB接口连接各种外设的设备，每一个接口不用关心连接的外设设备是什么，只要这个外设的设备实现了USB的标准，就可以连接到电脑上。

从程序上来讲，接口代表了某种能力，类似于生活中的合同。而在接口中定义的各个方法，表示这个能力的具体要求，类似于合同中的条款。

3.2. 内部类

内部类，即定义在类内的类。在Java中，可以在类的内部在定义一个完整的类。

- 内部类编译后，可以生成独立的字节码文件。
- 内部类可以直接访问外部类的私有成员，而不用破坏封装。
- 可为外部类提供必要的内部功能组件。

四 教学内容

4.1. 接口(会)

4.1.1. 接口的定义

定义接口，需要使用到关键字`interface`

```java
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 接口的定义
 */
public interface MyInterface {
    public static final String INTERFACE_FIELD = "value";   // 接口中的属性定义
    public abstract void method();                // 接口中的方法定义
}
```

接口中可以定义:

- 属性
  - 接口中的属性，默认都是静态常量，访问权限都是public。
- 方法
  - 接口中的方法，默认都是抽象方法，访问权限都是public。

注意:

一般接口中不写成员变量,只写方法,相当于制定规则,所以又将接口称为方法列表

接口的作用

让java从单继承间接的实现了多继承,扩充了原来的功能,我们可以认为接口是类的补充.

4.1.2. 接口和抽象类的异同

相同点

- 都可以编译成字节码文件。
- 都不能创建对象。
- 都可以声明引用。
- 都具备Object类中所定义的方法。
- 都可以写抽象方法。

不同点

- 接口中所有的属性，都是公开静态常量，缺省使用 public static final 修饰。
- 接口中所有的方法，都是公开抽象方法，缺省使用 public abstract 修饰。
- 接口中没有构造方法、构造代码段、静态代码段。

4.1.3. 接口的实现

接口，需要让类实现，表示这个类具有了这个接口定义的能力。因为接口中有很多的抽象方法，因此类在实现接口的时候，如果不是抽象类，必须要重写实现接口中所有的抽象方法。

```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 接口的实现
 */
public interface MyInterface {


```

```
    public static final String INTERFACE_FIELD = "value";    // 接口中的属性定义
    public abstract void method();                           // 接口中的方法定义
}

// 非抽象类实现接口，必须重写实现接口中所有的抽象方法
class MyInterfaceImpl implements MyInterface {
    @Override
    public void method() {}
}

// 抽象类实现接口，接口中的抽象方法，可以实现，也可以不实现
abstract class MyInterfaceAbstractImpl implements MyInterface {}
```

一个类可以实现多个接口

我们使用接口进行行为的约束，规则的制定。接口表示一组能力，那么一个类可以接受多种能力的约束。类比于合同来说，一个人是可以签订多份合同的。因此，一个类可以实现多个接口。实现多个接口的时候，必须要把每一个接口中的方法都实现了。

注意：如果一个类实现的多个接口中，有相同的方法，实现类只需要实现一次即可。

> 例如，一个人签订了两份合同，两份合同中都要求月收入2W，那么这个人如果月收入2W，则两份合同中的条款可以同时实现。

```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
```

```
 * @Description 多个接口的实现
 */
interface CookService {
    void cook();
}
interface NursemaidService {
    void cook();
    void wash();
}
class Nurse implements NursemaidService, CookService {
    // 这个cook方法，在这两个接口中都定义了，那么只需要实现一次即可
    @Override
    public void cook() {
        System.out.println("做饭");
    }
    @Override
    public void wash() {
        System.out.println("洗衣服");
    }
}
```

> 需要注意的是：
>
> 如果两个接口中定义了两个同名、同参数的方法，但是返回值不同。那么类是没有办法同时实现这两个接口的。因为在同时实现的时候，无法最终确定这个方法的返回值类型。就好比一个人签订了两份合同，A合同要求性别男，B合同要求性别女。这是有冲突的条件。因此类无法同时实现。

4.1.4. 接口的继承

接口之间也是存在继承关系的，与类的继承相似，子接口可以继承到父接口中所有的成员的。

与类的继承不同，接口之间的继承是多继承。也就是说，一个接口是可以有多个父接口的。子接口可以继承到所有的父接口中的成员。

```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 多个接口的实现
 */
interface SuperInterface1 {
    void method1();
}
interface SupeInterface2 {
    void method2();
}
// 此时，在这个接口中，继承到了所有的父接口中的方法，同时定义了自己独有的方法
// 实现类在实现这个接口的时候，需要实现 method1、method2、method3 三个方法
interface SubInterface extends SuperInterface1, SuperInterface2 {
    void method3();
}
```

注意：

上述，一个类实现多个接口的时候，多个接口中不能存在有冲突的方法。接口在继承父接口的时候，也不允许同时继承两个有方法冲突的接口。

父类与接口的功能如何分配?

```
一般父类中放的是主要功能,接口中放的是额外的功能,接口作为父类的补充.
```

当一个类实现的接口中出现了相同的方法,子类中实现方法的时候会不会混淆?

```
不会,接口中的方法都是抽象的,要通过子类写具体的实现.我们在调用方法时,最终看的功能,而功能只有子类中的一份.
```

4.1.5. 接口的多态

接口的引用，可以指向实现类的对象。与类的多态相似，同样存在向上转型和向下转型。

- 向上转型：实现类类型转型为接口类型。
  - 是一个隐式转换，不需要任何的修饰。
  - 向上转型后的接口引用，只能访问接口中的成员。
- 向下转型：接口类型转型为实现类类型。
  - 是一个显式转换，需要强制类型转换。
  - 向下转型后的实现类引用，将可以访问实现类中的成员。

```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 接口的继承
 */
interface CanWalk {
    void walk();
}
abstract class Animal {
    public abstract void eat();
}
class Dog extends Animal implements CanWalk {
```

```
    @Override
    public void eat() {}
    @Override
    public void walk() {}
}

class Test {
    public static void main(String[] args) {
        // 1. 实例化一个Dog对象
        Dog dog = new Dog();          // 将狗看成狗看待
        // 2. 将Dog对象转型为Animal类型
        Animal animal = dog;          // 将狗看成动物看待
        // 3. 将Dog对象转型为CanWalk类型
        CanWalk walk = dog;           // 将狗看成会走的东西看待
    }
}
```

多态体现

与类相同，向上转型后的接口的引用，调用接口中的方法的时候，实际调用的是实现类中的重写实现。

4.1.6. 接口的新特性(了解)

在Java8中，为接口添加了两个新特性：

- static方法：可以在接口中定义静态方法，静态方法不是抽象方法，是有实现部分的。同时，这个静态方法，只能由当前的接口名字调用，接口的引用和实现类都是不能使用的。

```
/**
```

```
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 接口新特性
 */
interface MyInterface {
    public static void method1() {
        System.out.println("接口中的静态方法实现");
    }
}

class Person implements MyInterface{
    //这里不可以重写method1方法
}

public class Demo8 {
    public static void main(String[] args) {
        MyInterface.method1();  //正确
        MyInterface m = new Person();
        m.method1();  //错误
        Person.method1();//错误
    }
}
```

- default方法：修饰接口中的方法，default修饰的方法可以添加默认的实现部分。此时，实现类在实现接口的时候，对于这些方法可以重写，也可以不重写。

```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 接口新特性
 */
```

```java
interface MyInterface {
    public default void method2() {
        System.out.println("接口中的方法默认实现");
    }
}

class Person implements MyInterface{
    //可以重写method2方法
}

public class Demo8 {
    public static void main(String[] args) {
        MyInterface m = new Person();
        m.method2();  //正确
        Person.method2();//错误
    }
}
```

4.2 内部类(了解)

4.2.1. 内部类的分类(会)

内部类，按照定义的位置和修饰符不同，可以分为:

- 成员内部类
- 静态内部类
- 局部内部类
- 匿名内部类

4.2.2. 成员内部类(会)

4.2.2.1. 概念

定义在一个类的内部的类.内部类的地位与外部类的成员变量,成员方法平等,内部类也可以看做是外部类的成员,成员之间可以相互调用

4.2.2.2. 使用

- 外部类的一个成员部分，创建内部类对象时，必须依赖外部类对象。
- Outer outer = new Outer();
- Inner inner = outer.new Inner();
- Inner inner1 = new Outer().new Inner();

4.2.2.3. 特点

- 书写位置：与属性、方法平级别，且没有使用static修饰的类。
- 访问权限：内部类可以是任意的访问权限。
- 成员内部类中，不能写静态属性、静态方法。
- 编译之后生成的字节码文件格式：外部类$内部类.class
- 实例化对象，需要借助外部类的对象完成。

4.2.2.4. 示例代码

```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 成员内部类
```

```java
 */
class OuterClass {
    public String name;
    public class InnerClass {
        public String name;
        public void show(String name) {
            System.out.println(name);          // 访问参数 name
            System.out.println(this.name);         // 访问内部类属性 name
            System.out.println(OuterClass.this.name); // 访问外部类属性 name
        }
    }

    public void  test(){
        System.out.println("Outer-show");

        InnerClass inner = new InnerClass();
        inner.show();
    }
}
class Program {
    public static void main(String[] args) {
        // 实例化外部类对象
        OuterClass outer = new OuterClass();

       //调用内部类的方法的方式
       //第一种:借助于外部类的方法实现
       outer.test();
       //方式二:借助外部类对象，实例化内部类对象
        //引用:外部类.内部类
        //构成:外部类对象的引用.new 内部类的构造方法
```

```
35          OuterClass.InnerClass inner = outer.new
    InnerClass();
36          inner.show();
37      }
38  }
```

4.2.3. 静态内部类(了解)

4.2.3.1. 概念

在类的内部定义，与实例变量、实例方法同级别的，使用static修饰的类。

4.2.3.2. 使用

- 不依赖外部类对象，可以直接创建或通过类名访问。
- Outer.Inner inner = new Outer.Inner();

4.2.3.3. 特点

- 书写位置：和类中的属性、方法平级，且使用关键字 static 修饰
- 静态内部类中，可以写属性、方法、构造方法...
- 静态内部类中，可以写静态属性、方法
- 编译之后生成的字节码文件，格式：外部类$内部类.class
- 对象的实例化，不需要借助外部类对象。

4.2.3.4. 实例代码

```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 静态内部类
 */
public class OuterClass {
  //静态内部类不一定有静态方法,有静态方法的一定是静态内部类
    static class InnerClass {
        String name;
        public  void show(String name) {
            System.out.println(name);
            System.out.println(this.name);
        }
    }
}
class Test {
    public static void main(String[] args) {
        // 1. 实例化静态内部类的对象
        //构成:  new  +  外部类名字.内部类的构造方法
        OuterClass.InnerClass innerClass = new
OuterClass.InnerClass();
        // 2. 可以通过导包的形式,
        //    先导包 import 包.OuterClass.InnerClass
        // InnerClass innerClass = new InnerClass();
        innerClass.show("aaa");
    }
}
```

4.2.4. 局部内部类(了解)

4.2.4.1. 概念

定义在外部类的方法中，作用范围和创建对象范围仅限于当前方法。

4.2.4.2. 特点

- 局部内部类访问外部类当前方法中的局部变量时，因无法保障变量的生命周期与自身相同，变量必须修饰为final。
- 不能使用访问权限修饰符修饰。
- 书写位置：写在一个类的方法内部，作用域仅限于当前方法。
- 局部内部类，编译后生成的字节码文件格式：外部类$序号内部类名.class

4.2.4.3. 示例代码

```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 局部内部类
 */
public class Program {
    public static void main(String[] args) {
        Outer1 outer1 = new Outer1();
        outer1.show();
    }
}

class  Outer1{
```

```
14      public void  show(){
15          System.out.println("Outer-show");
16          // 定义一个局部变量
17          // 如果这个局部变量，被包裹在了一个局部代码段中（比如局部内部类、匿名内部类），此时这个局部变量会被隐式的定义为final
18          int height=0;
19          //局部内部类
20          // 在一个类的方法中，直接定义一个内部类
21          class  Inner{
22              public  void run(){
23                  System.out.println("Inner-run"+height);
24              }
25          }
26
27        //创建局部内部类对象
28          Inner inner = new Inner();
29          inner.run();
30      }
31  }
```

4.2.4.4. 局部内部类的作用

通过局部内部类实现了功能的私有化,并对方法内部的代码进行了整理,增强了代码的可读性和可操作性.

示例代码

```
1  class Test{
2      public void play() {
3
4  //当我们想将这两个方法变成play的私有功能时,发现play中不能直接写方法的定义,所以写入局部内部类
5  //          public void gongneng1(){
6  //              System.out.println("功能1");
```

```
7  //          }
8  //          public void gongneng2(){
9  //              System.out.println("功能2");
10 //          }
11          class Inner{
12            //通过编写gongneng1,gongneng2两个方法,将play的整体功能实现了分类整理
13              public void gongneng1(){
14                  System.out.println("功能1");
15              }
16              public void gongneng2(){
17                  System.out.println("功能2");
18              }
19          }
20
21          Inner inner = new Inner();
22          inner.gongneng1();
23          inner.gongneng2();
24      }
25
26      public void run(){
27         //因为两个方法是play的局部内部类方法.play之外不可见
28 //          gongneng1();
29 //          gongneng2();
30      }
31  }
```

4.2.4.5. 局部内部类所在方法中局部变量的使用(了解)

- final:被final修饰的变量会被放在常量区,而常量区的值存在的时间要大于局部变量所在的方法,相当于从原来的基础上扩大了作用域
- 当方法中同时存在局部内部类与局部变量时,局部变量的使用范围就会从原来的基础上进行扩大.

原因:在当前程序执行时,程序会默认让final去修饰height.所以当局部变量所在的方法结束的时候,变量没有被释放,保存的值还在.

- 关于变量前面的final的说明:
  - 前提:变量必须与局部内部类同时存在.并且在局部内部类中使用了当前的局部变量
  - 在jdk1.7之前要想保住局部变量的值,要手动添加final
  - 在jdk1.7之后,程序执行时,java的内部机制已经在变量的前面默认添加了final

示例代码

模拟jdk1.7时局部变量前面有final修饰时的情况

结论:发现虽然show方法已经结束,但是我们仍然可以拿到age的值.

```
public class Demo4 {
    public static void main(String[] args) {
        Outer4 outer4 = new Outer4();
        outer4.show();
        outer4.eat();//show方法已经结束,但是 获取age的值 age = 6
    }
}

class Outer4{
    Object object = null;
    public void show() {
     int  age = 6;
        class Inner{//局部内部类

            public void run() {
                System.out.println("跑"+age);
```

```
            }

            @Override
            public String toString() {
                // TODO Auto-generated method stub
                return "toString    "+age;
            }
        }

        //show的内部使用局部内部类
        object = new Inner();//多态

    }

    public void eat(){
        System.out.println(object.toString());
        System.out.println("eat");
    }
}
```

4.2.5. 匿名内部类(了解)

4.2.5.1. 概念

匿名内部类(对象):定义在一个类的方法中的匿名子类对象,属于局部内部类

其实这个概念我们可以分成两部分看

先学习匿名子类对象:没有名字的子类对象

再学习匿名内部类对象:一个类的方法中的匿名子类对象

4.2.5.2. 特点

- 一切特征与局部内部类相同。
- 必须继承一个父类或者实现一个接口。
- 定义类、实现类、创建对象的语法合并，只能创建一个该类的对象。

4.2.5.3. 匿名内部类作用

- 当只用到当前子类的一个实例对象的时候,定义好马上使用,使用完立刻释放
- 当不好起名字的时候
- 可以更好的定义运行时的回调(知道即可)

4.2.5.4. 创建匿名子类对象

- 创建方式

  第一种方式:使用已有的子类创建匿名子类对象

  ```
  使用场景:已经创建好的子类可以多次使用,适用于相同的功能被多次调用
  ```

  第二种方式:直接使用Animal创建匿名子类对象

  ```
  构成: new +  父类的名字/接口的名字 + () + {写当前子类的成员} + ;

  使用场景:只能使用一次,使用完会被当做垃圾回收,适用于每次都使用不同的功能
  ```

- 示例代码

```
/**
 * @Author 千锋大数据教学团队
```

```
 * @Company 千锋好程序员大数据
 * @Description 局部内部类
 */
public class Demo6 {
    public static void main(String[] args) {
        Animal animal = new Animal();
        //匿名对象
        new Animal().eat();

        //匿名子类对象
        //第一种方式:使用已有的子类创建匿名子类对象
        new Dog().eat();
        //第二种方式:直接使用Animal创建匿名子类对象
        //直接创建没有名字的Animal的匿名子类对象
        new Animal(){
            @Override
            public void eat() {
                System.out.println("匿名子类对象-eat");
            }
        }.eat();
    }
}

//研究匿名子类对象
class Animal {
    public void eat() {
        System.out.println("fu-eat");
    }
}
class Dog extends Animal
{
    public void eat() {
        System.out.println("zi-eat");
```

```
    }
}
```

4.2.5.5. 创建匿名内部类

- 说明

  代码中的Animal对象就是一个匿名内部类

  我们可以用匿名内部类做方法的参数或返回值

  匿名内部类的父类可以是父类也可以是父接口

- 示例代码

```
public class Demo6 {
    public static void main(String[] args) {
        //测试匿名内部类
        Test1 test1 = new Test1();

test1.canShuTest();//com.qf.test.Animal@1b6d3586

test1.canShuTest1();//com.qf.test.Test1$2@4554617c  使用外部类+$+序号表示当前的匿名内部类
    }
}

class Animal {
    public void eat() {
        System.out.println("fu-eat");
    }
}

//研究匿名内部类
class  Test1{
    public  void  show(){
```

```
19          //匿名内部类
20          new Animal(){
21              @Override
22              public void eat() {
23                  System.out.println("匿名子类对象-eat");
24              }
25          }.eat();
26      }
27
28      //普通的匿名对象作为参数
29      public void canShuTest(){
30          System.out.println(new Animal());
31      }
32      //匿名内部类作为参数
33      public  void  canShuTest1(){
34          System.out.println(new Animal(){
35              @Override
36              public void eat() {
37                  System.out.println("eat");
38              }
39          });
40      }
41
42      //普通的匿名对象作为返回值
43      public Animal fanHuiZhiTest(){
44          return  new Animal();
45      }
46
47      //匿名内部类作为返回值
48      public Animal fanHuiZhiTest1(){
49          return  new Animal(){
50  //              public void jump(){
51  //
```

```
52  //              }
53          };
54      }
55  }
```

注意:除了new Object类是匿名对象,其他所有类的匿名对象本质上都是匿名子类对象.

4.2.6. 内部类作用(会)

- 间接实现了多继承
- 方便定义
- 只有外部类可以访问创建的内部类的属性和方法,包括私有方法
- 同一个包中其他的类不可见,有了很好的封装性

示例代码

要求:要让X同时继承来自A和B的内容,并且A和B没有关系.

```
1  class  A{}
2  class  B{}
3  class  X extends  A{
4      class  Y extends  B{}
5  }
```