## 实验二 Java类和对象的设计

## 一、实验目的

掌握面向对象的编程思想、类与对象；

掌握类的封装性、继承性和多态性的作用；

掌握成员变量和成员方法的特性、构造方法**、toString方法、equals方法的使用**；

掌握this、super、final、static等关键字的使用；

掌握接口及接口对象的使用；

掌握包的定义与使用；

掌握四种访问控制权限private、default、protected、public。

## 二、案例

## 案例2-1 自定义学生类

1. 案例描述

掌握类定义的方式；掌握如何在类中定义成员变量和成员方法

1. 需求分析

在面向对象的思想核心就是对象，在程序中创建对象的前提是需要定义一个类。为了让初学者掌握类的定义方式，本案例将设计一个表示学生的类，该类具有表示姓名的属性name和表示年龄的属性age，同时还具有表示说话行为的方法speak()，用于输出学生的姓名和年龄。

1. 设计思路（实现原理）

1）使用class关键字定义一个表示学生类型的类，类名为Student。

2）在Student类中定义两个成员变量name和age，分别用来表示姓名和年龄。其中，name的数据类型为String，变量age的数据类型为int。

3）在Student类中定义一个表示说话行为的speak()方法，用于输出学生的姓名和年龄。

4、案例实现

class Student{

String name;

int age;

void speak() {

System.out.println("我的名字是 "+name+"，今年 "+age+"岁");

}

}

**5、案例总结**

1）Java语言严格区分大小写，class和Class是不同的，在定义类时只能使用class关键字

2）在Student类中，成员变量name是String类型，String表示一个字符串，后面的章节会详细讲解

3）思考一下：自己定义一个手机(Phone)类，在类中定义品牌(brand)和价格(price)属性，定义打电话的call()方法，代码如下所示：

public class Phone {

String brand;

double price;

void call(){

System.out.println("hi,how are you doing");

}

}

## 案例2-2 类的封装

1. 案例描述

了解为什么要对类进行封装；了解如何实现类的封装

1. 需求分析

在案例2-1中，s1对象的年龄是可以随便赋值的，如果将age的值赋值为-30，显然违背了事实。为了解决这类问题，我们需要对类进行封装，防止外界对类中的成员变量随意访问。为了让初学者更好地掌握类的封装，本案例将使用private关键字对成员变量name和age进行私有化，同时分别提供一个setName(String n)和setAge(int a)方法用于外界的访问，其中setAge(int a)中需要对age进行判断。

1. 设计思路（实现原理）
2. 编写测试类Example02，将属性age的值设为-30，演示不合理现象。
3. 对Student类进行修改，将name和age属性使用private修饰，然后定义getName()、setName(String n)、getAge()和setAge(int a)四个对外访问name和age的方法。
4. 在setAge(int a)方法中对传入的参数进行检查，如果输入值为负数，则打印出“设置的年龄不合法”，如果不为负数，才将其设置为age属性的值。

4）对Example02类进行修改，在main()方法中创建Student类的实例对象，通过调用对象的setName(String n)和setAge(int a)方法来设置的name属性和age属性值，并调用speak()方法。

4、案例实现

1、定义Example02类，代码如下所示：

public class Example02 {

public static void main(String[] args) {

Student s1 = new Student();

s1.name = "小新";

s1.age = -30;

s1.speak();

}

}

运行结果如图2-1所示。

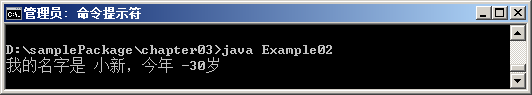


图2-1 运行结果

从上图可以看出，当将age的值设置为-30后，程序不会报错，但却违背了现实。

2、对Student类进行封装，其代码如下：

class Student {

private String name = "张三";

private int age = 19;

public String getName() {

return name;

}

public void setName(String n) {

name = n;

}

public int getAge() {

return age;

}

public void setAge(int a) {

// 对传入的参数进行检查

if(a < 0){

System.out.println("设置的年龄不合法");

}else{

age = a;

}

}

void speak() {

System.out.println("我的名字是" + name + "，今年" + age + "岁");

}

}

public class Example02 {

public static void main(String[] args) {

Student s1 = new Student();

s1.setName("小新");

s1.setAge(-30);

s1.speak();

}

}

运行结果如图2-2所示。

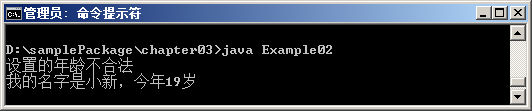


图2-2 运行结果

5、案例总结

1）Student的name和age属性使用private关键字修饰为私有后，在Example02类中不能再使用s1.name和s1.age的方式访问这两个属性，只能通过public类型的setName(String n)和setAge(int a)方法进行访问。

2）在上面的代码中，调用setAge(int a)方法时的传入参数为-30，由于参数小于0，会打印出“设置的年龄不合法”，并不会将负数赋值给age属性。由此可见，只要实现了封装就能对外界的访问进行控制，避免对私有变量随意修改而引发问题。

## 案例2-3 定义构造方法（含重载）

1、案例描述

掌握当未定义构造方式，系统会自动分配无参构造方法；

掌握如有定义构造方法，系统不再自动分配无参构造方法；

掌握有参构造方法的定义方式；

掌握构造方法的重载。

2、需求分析

如果希望在创建对象的时候直接为其属性赋值，可以定义有参的构造方法。有参构造方法指的是在初始化对象时，接受外部传入的值并赋给对象的属性。

和普通方法一样，构造方法也可以重载。不同的构造方法，可以为不同的属性进行赋值。本案例将通过创建对象的方式演示不同构造方法的使用方式，并根据构造方法的输出结果对构造方法的重载进行学习。

3、设计思路（实现原理）

1. 在案例2-1的基础上，定义在类中定义四个重载的构造方法，包括无参的构造方法，接收一个String类型参数的构造方法，接收int类型参数的构造方法，接收String类型和int类型两个参数的构造方法。
2. 编写一个测试类Example04，在main()方法中通过这4个构造方法分别创建4个对象。
3. 打印这些对象的信息。

4、案例实现

1） 对Student类进行修改，代码如下所示：

**public** **class** Student {

**private** **int** age;

**private** String name;

**public** Student() {

age = 0;

name = "未命名";

}

**public** Student(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

**public** Student(String name) {

**this**.name = name;

}

**public** Student(**int** age) {

**this**.age = age;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

2）定义Example03类，代码如下所示：

**public** **class** Example03 {

**public** **static** **void** main(String[] args) {

Student s1, s2, s3, s4;

s1 = **new** Student();

s2 = **new** Student(18);

s3 = **new** Student("java");

s4 = **new** Student("student", 20);

System.***out***.println("s1的姓名:" + s1.getName() + ",年龄：" + s1.getAge());

System.***out***.println("s2的姓名:" + s2.getName() + ",年龄：" + s2.getAge());

System.***out***.println("s3的姓名:" + s3.getName() + ",年龄：" + s3.getAge());

System.***out***.println("s4的姓名:" + s4.getName() + ",年龄：" + s4.getAge());

}

}

运行结果如图2-3所示。

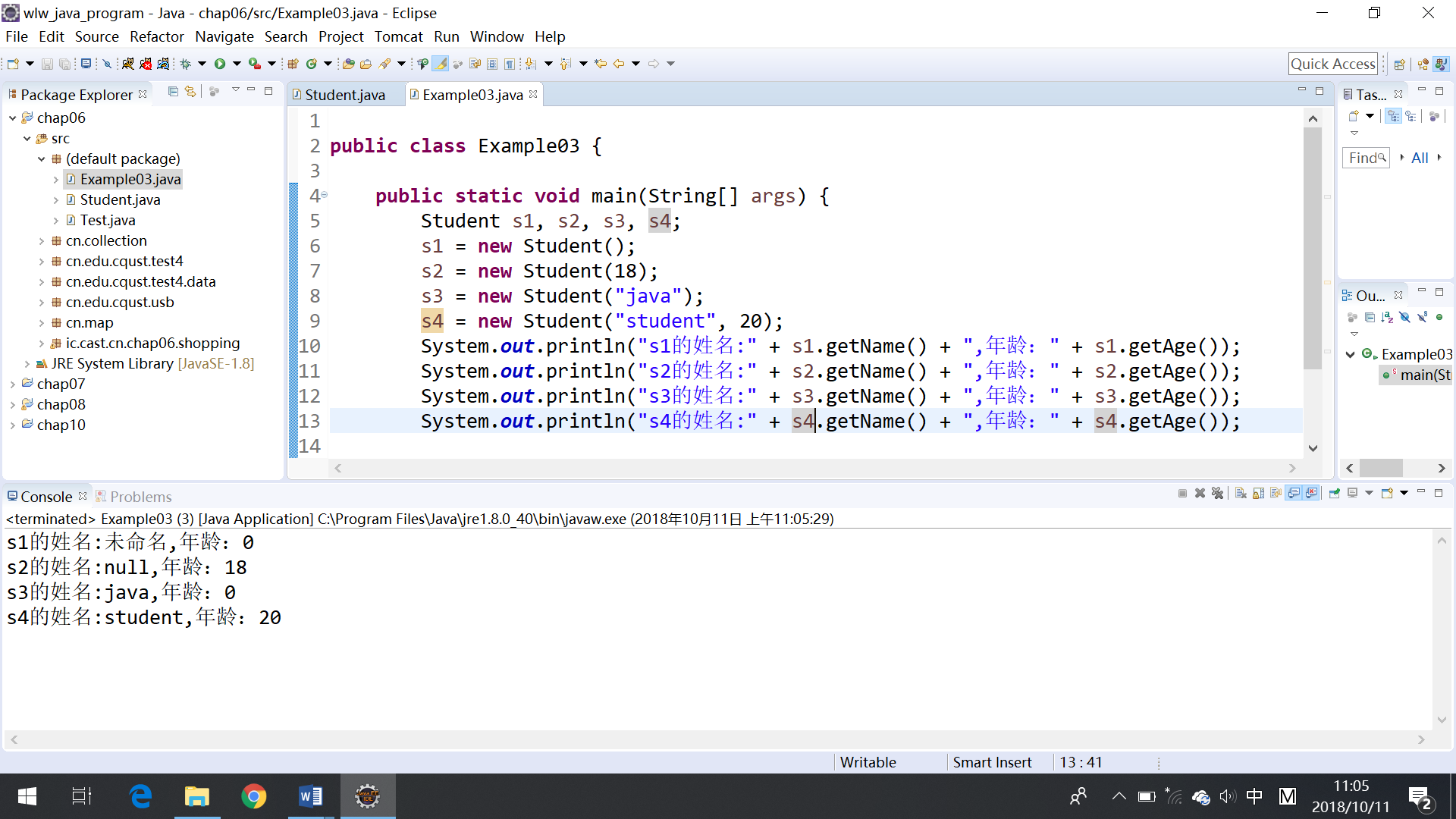


图2-3 运行结果

5、案例总结

一个类中可以定义多个重载的构造方法，在创建对象时，根据传入参数的不同会调用相应的构造方法。

## 案例2-4 类的继承

1、案例描述

了解类继承的意义和作用；

掌握如何实现类的继承。

2、需求分析

在程序中，继承描述的是事物之间的所属关系，通过继承可以使多种事物之间形成一种关系体系。

为了让初学者熟悉类的继承，本案例将演示类的继承并编写测试类验证子类是否拥有父类的可继承成员。

3、设计思路（实现原理）

1）设计两个类Student和Teacher

2）抽取两个类共同的内容（如：吃饭、睡觉）封装到一个类Person中，各自特有的部分保留在各自类中。

3）让学生类继承Person类，老师类也继承Person。

4）编写测试类Example04，测试Student类和Teacher是否继承了Person类的成员。

4、案例实现

1） 抽取父类Person，实现继承关系，代码如下：

class Person {

String name;//姓名

// 吃饭的功能

void eat() {

System.out.println("吃饭 ");

}

// 睡觉的功能

void sleep() {

System.out.println("睡觉");

}

}

class Student extends Person {

// 学号

int sid;

}

class Teacher extends Person {

// 工号

int tid;

// 教课的功能

void teach() {

System.out.println("老师教课");

}

}

2）定义测试类Example04，代码如下：

public class Example04{

public static void main(String[] args) {

Student s = new Student();

s.eat();

s.sleep();

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

Teacher t = new Teacher();

t.eat();

t.sleep();

t.teach();

}

}

运行结果如图2-4所示。

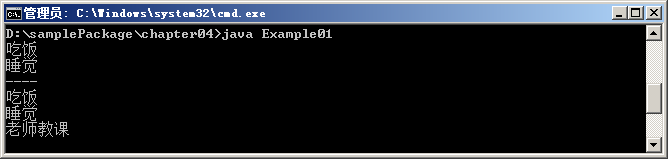


图2-4运行结果

5、案例总结

1）在Java中，多个类可以继承一个父类，但是一个类不能直接继承多个类，一个类只能有一个直接父类。

2）父类是由子类不断抽取而来的，不断地抽取就形成了体系结构，这个结构称为继承体系结构。

3）子类在继承父类的时候，会自动拥有父类所有的成员。4）继承的好处是划分了类的层次性，实现了代码重用、扩展了程序功能。

## 案例2-5 方法的重写

1. 案例描述

了解方法重写的意义和作用

掌握如何进行方法重写

1. 需求分析

在继承关系中，子类会自动继承父类中定义的方法，但有时在子类中需要对继承的方法进行一些

修改，即对父类的方法进行重写。为了让初学者掌握方法的重写，本案例将编写一个类NewPhone，该类继承Phone类并对Phone类的call()方法进行重写。

1. 设计思路（实现原理）

1）定义一个类Phone，编写方法call()，表示打电话功能

2）定义一个Phone的子类NewPhone，重写父类call()方法，让它除了打电话功能外还具有开启语言和关闭语言功能。

3）编写一个测试类Example05，分别调用重写前和重写后的call()方法

4、案例实现

1）定义Phone及其子类NewPhone，子类对父类中的call()方法重写，代码如下：

class Phone {

void call() {

System.out.println("打电话");

}

}

class NewPhone extends Phone {

void call() {

System.out.println("开启语音");

super.call();

System.out.println("关闭语音");

}

}

2）定义测试类Example05，代码如下：

public class Example05{

public static void main(String[] args) {

System.out.println("--重写前--");

Phone phone = new Phone();

phone.call();

System.out.println("--重写后--");

Phone newPhone = new NewPhone();

newPhone.call();

}

}

运行结果如图2-5所示。

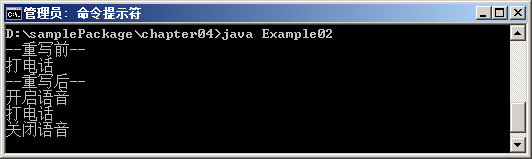


图2-5 运行结果

5、案例总结

1）子类中需要对继承自父类的方法进行一些修改，这时就用到方法重写。

2）在子类中重写的方法需要和父类被重写的方法具有相同的方法名、参数列表以及返回值类型。

3）子类方法的访问修饰权限不能小于父类的。

4）重写的主要优点是能够定义子类特有的特征。

## 案例2-6 接口的实现

1. 案例描述

掌握如何实现接口

1. 需求分析

由于接口中的方法都是抽象方法，因此不能通过实例化对象的方式来调用接口中的方法。此时需要定义一个类，并使用implements关键字实现接口中所有的方法。为了上初学者掌握如何实现接口，本案例将设计一个类，使用关键字implements实现Inter接口中的方法。

1. 设计思路（实现原理）

1）设计名为Inter的接口，接口中定义一个初始值为20的num常量和method()抽象方法。

2）编写Inter接口的实现类InterImpl，使用implements关键字实现接口中的方法。

3）编写测试类Example10，测试类中对接口的实现类进行实例化，并调用接口中的方法。

4、案例实现

1）编写接口Inter，InterImpl类使用implements实现了接口，代码如下：

//定义一个接口

interface Inter {

int num = 20;

void method();

}

// 使用implements关键字实现接口

class InterImpl implements Inter {

void show() {

System.out.println(num);

}

public void method() {

System.out.println("InterImpl method");

}

}

2）定义测试类Example06，代码如下：

class Example06{

public static void main(String[] args) {

//通过实现类实例化

InterImpl ii = new InterImpl();

ii.show();

ii.method();

}

}

运行结果如图2-6所示。

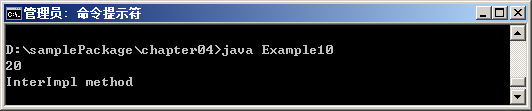


图2-6 运行结果

5、案例总结

1）接口使用interface关键字来定义。

2）接口没有构造方法，接口中的变量缺省都是使用public static final修饰的，即全局常量，接口中的方法都是public abstract修饰的，即抽象方法。

3）定义一个类，并使用implements关键字实现接口中的方法，一个类可以实现多个接口。

4）由于接口中的方法都是抽象方法，因此不能直接通过实例化对象的方式来调用接口中的方法，需要在类实现接口后借助类来间接实例化。

## 案例2-7 接口的继承

1、案例描述

掌握接口继承的方式及其特点

2、需求分析

在程序中，可以定义一个接口使用extends关键字去继承另一个接口。为了加深初学者对结果的理解，本案例将演示接口与接口、类与接口的继承和实现关系。

3、设计思路（实现原理）

1）设计接口Phone代表传统手机，在接口中定义receiveMessages()和call()抽象方法。

2）设计接口SmartPhone代表智能手机，在接口中定义faceTime()方法，并使用extends关键字继承Phone接口，使其具有Phone接口的所有功能。

3）设计类MIPhone表示小米手机，类中定义useMIUI()方法，并实现SmartPhone接口的所有方法。

4）编写测试类Example07，测试类中对MIPhone进行实例化，并访问小米手机具有的各种功能。

4、案例实现

1）编写接口Phone和SmartPhone、类MIPhone。代码如下：

interface Phone {

void receiveMessages();

void call();

}

interface SmartPhone extends Phone {

void faceTime();

}

class MIPhone implements SmartPhone {

public void receiveMessages() {

System.out.println("接收短息");

}

public void call() {

System.out.println("语音通话");

}

public void faceTime() {

System.out.println("视频通话");

}

public void useMIUI() {

System.out.println("使用MIUI");

}

}

2）定义测试类Example07，代码如下：

public class Example11 {

public static void main(String[] args) {

MIPhone miPhone = new MIPhone();

miPhone.receiveMessages();

miPhone.call();

miPhone.faceTime();

miPhone.useMIUI();

}

}

运行结果如图2-7 所示。

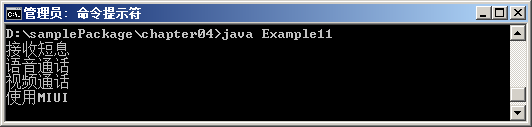


图2-7 运行结果

5、案例总结

接口与接口之间的继承关系，可以是单继承，也可以是多继承；一个接口可以通过extends关键字继承多个接口，接口之间用逗号隔开。

## 案例2-8 Object类

1. 案例描述

了解什么是Object类

掌握Object类中常用方法的使用

1. 需求分析

在JDK中提供了一个Object类，它是所有类的父类，即每个类都直接或间接继承自该类。Object类自身的equals()方法比较的是内存地址值，相当于“==”，但是在实际开发中我们很少比较两个对象的地址值，这时候常常就需要根据自身需求来重写equals()方法。为了方便初学者学习Object类，本案例将演示如何重写equals()方法。

1. 设计思路（实现原理）

1）设计Student类，定义sid变量表示学号，重写equals()方法，判断进行比较的两个学生对象学号是否相同，相同即为同一个学生，否则不是。

2）编写测试类Example08，创建两个Student的对象Lily和Lucy，并分别调用setter方法为学号赋值，最后通过打印语句，输出这两个对象通过“==”和重写后的equals()方法比较后的结果。

4、案例实现

1）编写Student类，重写equals()方法。代码如下：

class Student {

Student() {}

// 学号

private int sid;

public void setSid(int sid) {

this.sid = sid;

}

public int getSid() {

return sid;

}

// 我们的需求是学号相同的人就是同一个人.

public boolean equals(Object obj)

{

// 判断是否为当前对象

if (this == obj) {

return true;

}

// 为了呈现的健壮性,加入类型判断

if (!(obj instanceof Student)) {

return false;

}

Student s = (Student) obj;

return (this.sid == s.sid);

}

}

2）定义测试类Example08，为学号赋值，并打印使用“==”和重写equals()方法的比较结果，代码如下：

public class Example08 {

public static void main(String[] args) {

Student Lily = new Student();

Lily.setSid(200);

Student Lucy = new Student();

Lucy.setSid(200);

System.out.println("Lily == Lucy"+"\t\t"+(Lily == Lucy));

System.out.println("Lily.equals(Lucy)"+"\t"+Lily.equals(Lucy));

}

}

运行结果如图2-8所示。

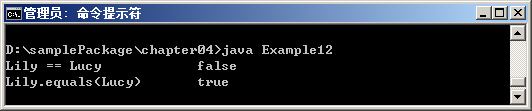


图2-8 运行结果

5、案例总结

1）Object类是Java中所有类的父类，每个类都直接或间接继承自该类。

2）Object类中的equals()方法比较的是内存地址值，和“==”比较结果一致。底层代码如下：

public boolean equals(Object obj) {

return (this == obj);

}

实际开发中很少比较内存地址值，所以我们常常会根据自身需求重写equals()方法。

## 案例2-9 多态的作用

1. 案例描述

了解多态的概念、多态的好处和弊端

掌握如何实现多态

1. 需求分析

在Java中为了实现多态，允许使用一个父类类型的变量来引用一个子类类型的对象，根据被引用子类对象特征的不同，得到不同的运行结果。多态提高代码了的扩展性和可维护性。为了帮助初学者掌握如何实现多态，本案例将实现一个多态程序来演示如何实现多态。

1. 设计思路（实现原理）

1）编写Animal接口，接口中定义sleep()方法。

2）Cat类实现Animal接口的方法，并定义另一个方法catchMouse()

3）编写测试类Example09，使Cat对象指向父类Animal的引用 ，并通过该引用调用sleep()方法。

4、案例实现

1）编写Animal接口及其实现类Cat。代码如下：

interface Animal {

void sleep();

}

class Cat implements Animal {

void catchMouse() {

System.out.println("抓老鼠");

}

public void sleep() {

System.out.println("睡觉");

}

}

2）定义测试类Example09，Cat对象指向Animal引用，并使用父类Animal引用调用sleep()方法。代码如下：

public class Example09 {

public static void main(String[] args) {

// 多态

Animal animal = new Cat();

animal.sleep();

}

}

运行结果如图2-9所示。

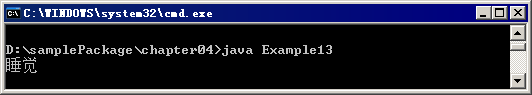


图2-9 运行结果

5、案例总结

1）多态是指对象在不同时刻表现出来的不同状态，在Java中，允许使用一个父类类型的变量来引用一个子类类型的对象。

2）多态的前提：

(1) 要有继承关系。

(2) 要有方法重写。

(3) 要有父类引用指向子类对象。

3）多态的好处是提高代码的扩展性和可维护性。

4）多态的弊端是父类或者接口只能调用自己具备的成员，不能访问子类特有的成员。

## 案例2-10 对象的类型转换

1、案例描述

掌握如何实现对象的类型转换

2、需求分析

在多态的学习中，涉及到将子类对象当做父类类型使用的情况，这时进行强制类型转换可能会出现出错。为了让初学者熟悉对象的类型转换，本案例将演示一个子类与父类之间类型转换过程中出现的错误。

编写一个接口Animal，接口中定义sleep()方法，编写两个类Cat和Pig分别实现接口，再编写一个测试类Example10实现对象的类型转换。

3、设计思路（实现原理）

1）编写Animal接口，接口中定义sleep()方法。

2）Cat类和Pig类分别实现Animal接口的方法

3）编写测试类Example10对对象的类型进行转换，使Cat向上转型为Animal类型，然后再让Animal类型转换为Pig类型。

4、案例实现

1）编写Animal接口及其实现类Cat。代码如下：

interface Animal {

void sleep();

}

class Cat implements Animal {

public void sleep() {

System.out.println("猫晚上睡觉");

}

}

class Pig implements Animal {

public void sleep() {

System.out.println("猪整天睡觉");

}

}

2） 定义测试类Example10，Cat对象指向Animal引用，并使用父类Animal引用调用sleep()方法。代码如下：

public class Example10 {

public static void main(String[] args) {

Animal animal = new Cat();

Pig pig = (Pig) animal;

pig.sleep();

}

}

运行结果如图2-10所示。

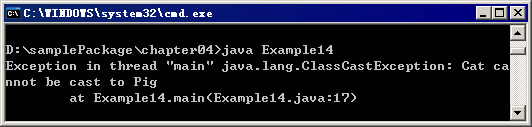


图2-10 错误提示

5、案例总结

1）向下类型转换时，需要注意被转换对象的真实类型是否和需要转换的对象类型相同，否则容易报类型转换异常(java.lang.ClassCastException)。

2）Java提供了一个关键字instanceof，它可以判断一个对象是否为某个类(或接口)的实例或者子类实例，语法格式如下：

对象(或者对象引用变量) instanceof 类(或接口)

3）进行向上类型转换，子类对象当做父类类型使用时，不能通过父类变量去调用子类中某些方法，否则会抛出编译异常。

## 三、实验内容

1. 定义一个类MyProgram，包含两个属性：一个是private的整型属性data、一个是private的String类型属性str，封装这两个属性的四个方法setData( )和getData( )、setStr( )和getStr( )；将这两个属性转变为字符串的方法是toString( )。编写程序，使用MyProgram类，实现数据的访问和修改，并调用toString( )方法显示该类的属性。

[基本要求] 编写完整程序。

[思考问题] 试述程序中各个方法的作用

2. 定义一个类实现银行帐户的概念，包括的变量有“帐号”"和“存款余额”，包括的方法有“存款”、“取款”、“查询余额”和“显示帐号”。定义主类，创建帐户类的对象，并完成相应操作。

[基本要求] 编写完整程序。

[思考问题] 试述程序中各个方法的作用。

3. 有图形类的父类Shape，参照圆Circle类补充完整正方性Square和三角形Triangle类，并分析运行结果。

class Shape {

void draw() {}

void erase() {}

}

class Circle extends Shape {

void draw() { System.out.println("Circle.draw()");}

void erase() { System.out.println("Circle.erase()");}

}

class Square extends Shape {

void draw() { }

void erase() { }

}

class Triangle extends Shape {

void draw() { }

void erase() { }

}

public class Shapes {

public static Shape randShape() {

switch((int)(Math.random() \* 3)) {

default: // To quiet the compiler

case 0: return new Circle();

case 1: return new Square();

case 2: return new Triangle();

}

}

public static void main(String[] args) {

Shape[] s = new Shape[9];

// Fill up the array with shapes:

for(int i = 0; i < s.length; i++)

s[i] = randShape();

// Make polymorphic method calls:

for(int i = 0; i < s.length; i++)

s[i].draw();

}

}

4. 有两个类：MobileManagement和Mobile，分别描述如表2.1所示两部手机名称及价格，类MobileManagement在包cn.edu.cqust.test4中，而Mobile在包cn.edu.cqust.test4.data中。它们代码如下。运行MobileManagement.java，使程序能够显示两部手机的价格和数量。

[基本要求] 在空白处填写相关代码并修改上面程序，使程序能够显示两部手机的价格和数量，运行结果如图2-11 所示。

表2.1 手机价格表

|  |  |  |
| --- | --- | --- |
| 序号 | 名称 | 价格（元） |
| 1 | 荣耀8 | 1099 |
| 2 | 小米Max2 | 1699 |

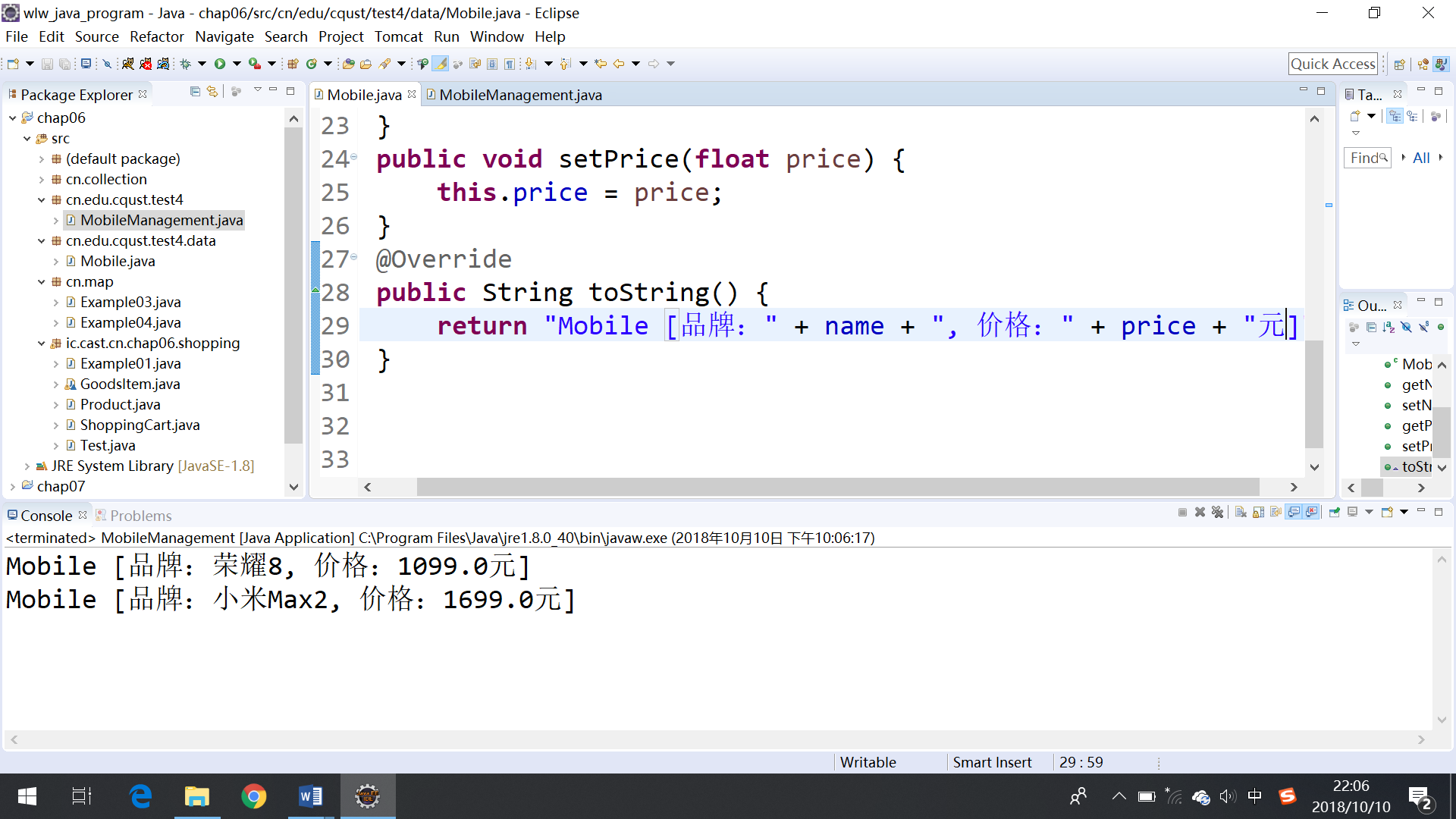


图2-11 运行结果

程序Mobile.java源代码：

**public** **class** Mobile {

/\*\* Holds the name of the mobile. \*/

**private** String name;

/\*\* Holds the price of the mobile. \*/

**private** **float** price;

/\*\* Creates a new mobile object. \*/

**public** Mobile(String name, **float** price) {

**this**.name = name;

**this**.price = price;

}

/\*\* Gets the name of the mobile. \*/

**public** String getName() {

**return** name;

}

/\*\* Gets the price of the mobile. \*/

**public** **float** getPrice() {

**return** price;

}

}

程序MobileManagement.java源代码：

**public** **class** MobileManagement {

/\*\* Defines the entry point of the application. \*/

**public** **static** **void** main(String[] args) {

// Creates two mobile phone objects.

Mobile mobile1 = **new** Mobile("荣耀8", 1099);

Mobile mobile2 = **new** Mobile("小米Max2", 1699);

}

}

**5.USB接口程序设计**

通常人们使用的计算机上都有USB接口，鼠标、键盘、麦克风等都可以连接到USB接口中使用。在计算机启动时，这些设备也随之启动；当计算机关闭时，这些设备也会随之关闭。鼠标、键盘、麦克风等USB接口设备都启动后，计算机才开机成功；当这些设备都关闭后，计算机才关机成功。

读所编写的USB接口程序，该程序模拟了计算机的开机和关机过程，写出运行结果。

程序代码：

**⑴ USB接口，定义了设备的启动和关闭功能（USB.java）**

**package** cn.edu.cqust.usb;

**public** **interface** USB {

**void** turnOn();//启动

**void** turnOff();//关闭

}

**⑵ 鼠标类，实现了USB接口（Mouse.java）**

**package** cn.edu.cqust.usb;

**public** **class** Mouse **implements** USB {

@Override

**public** **void** turnOn() {

System.***out***.println("鼠标启动了......");

}

@Override

**public** **void** turnOff() {

System.***out***.println("鼠标关闭了......");

}

}

**⑶ 麦克风类，实现了USB接口（Mic.java）**

**package** cn.edu.cqust.usb;

**public** **class** Mic **implements** USB {

@Override

**public** **void** turnOn() {

System.***out***.println("麦克风启动了......");

}

@Override

**public** **void** turnOff() {

System.***out***.println("麦克风关闭了......");

}

}

**⑷ 键盘类，实现了USB接口（KeyBoard.java）**

**package** cn.edu.cqust.usb;

**public** **class** KeyBoard **implements** USB {

@Override

**public** **void** turnOn() {

System.***out***.println("键盘启动了......");

}

@Override

**public** **void** turnOff() {

System.***out***.println("键盘关闭了......");

}

}

**⑸ 计算机类，拥有4个USB接口，具有添加USB设备、开机、关机的功能（Computer.java）**

**package** cn.edu.cqust.usb;

**public** **class** Computer {

**private** USB[] usbArr=**new** USB[4];

//向计算机上连接一个USB设备

**public** **void** add(USB usb){

//遍历所有的插槽

**for**(**int** i=0;i<usbArr.length;i++){

//如果发现一个空的

**if**(usbArr[i]==**null**){

usbArr[i]=usb;

**break**;

}

}

}

//计算的开机功能

**public** **void** powerOn(){

//遍历所有的插槽

**for**(**int** i=0;i<usbArr.length;i++){

//如果发现有设备

**if**(usbArr[i]!=**null**){

//将USB设备启动

usbArr[i].turnOn();

}

}

System.***out***.println("计算机开机成功！！！");

}

//计算的关机功能

**public** **void** powerOff(){

//遍历所有的插槽

**for**(**int** i=0;i<usbArr.length;i++){

//如果发现有设备

**if**(usbArr[i]!=**null**){

//将USB设备关闭

usbArr[i].turnOn();

}

}

System.***out***.println("计算机关机成功！！！");

}

}

**⑹ 运行类（测试类）（Test.java）**

**package** cn.edu.cqust.usb;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

//实例化计算机对象

Computer computer=**new** Computer();

//向计算机中添加鼠标、麦克风、键盘

computer.add(**new** Mouse());

computer.add(**new** Mic());

computer.add(**new** KeyBoard());

//启动计算机

computer.powerOn();

//关闭计算机

computer.powerOff();

}

}

6、请简述this关键字的作用，并举例说明；

7、请简述super关键字的作用，并举例说明；

8、请简述final关键字的作用，并举例说明。

9、请简述static关键字的作用，并举例说明。