# 第5章 面向对象程序设计进阶

## 5.1 实验目的

- 掌握 static 关键字，理解类层次和对象层次的区别。
- 掌握 final 关键字，理解 final 代表进化终止。
- 掌握 abstract 关键字，熟悉抽象类和抽象方法的设计技巧。
- 掌握 interface 关键字和接口的基本概念。
- 掌握 package 关键字和包的概念及使用技术。

## 5.2 相关知识

在 Java 中，static 关键字用来定义类层次的成员变量和成员方法，这种类层次的成员被类加载器装入后就可以直接使用，不需要创建对象实例，并且在内存中只有一个副本，所有对象共享使用，相对于类层次成员，对象层次的成员只有创建了对象后才能使用，并且每个对象都有自己的副本。final 关键字用来修饰不可变成员，即成员的值或行为将来不可改变，具体来说，如果用 final 修饰类，则此类不能派生子类，相当于该类的进化终止了；如果用 final 修饰方法，则此方法在其子类中不能被重写，此方法进化终止；如果用 final 修饰变量，则变量的值不能再修改，变量变为常量。Java 使用 abstract 关键字来定义抽象类和抽象方法，抽象的意思就是设计不完整，还需要后续的设计来继续完善。

Java 中没有多继承，只有单一继承，为了使一个对象具有多种行为，Java 提供了接口的概念，用 interface 关键字来定义接口，跟现实生活一样，接口代表的是规范和标准，所以只能包含公有的常量和公有抽象的方法。

前两章实验的重点内容是理解面向对象设计中的基本原理，本章实验对 Java 中面向对象的实现技术进行进阶学习。抽象和继承进阶中将学习抽象类，封装进阶中将介绍单态设计模式，多态进阶中将学习接口，以及对应主教材补充包和枚举等实验内容。

## 5.3 实验内容

视频讲解

### 5.3.1 验证实验

(1) 理解 static、final 关键字。static 关键字修饰的成员是类层次，即通过类名来使用所

有对象共享的成员。而 final 关键字是“最终”，不可更改的意思，代表进化终止。输入以下代码后编译运行，查看运行结果。

```
class Static_final {
      static int i = 10;
      static final int k = 20;
      static {    i = i + 5; }
      public static void main(String[] args) {
         System.out.println("i = " + i);
         System.out.println("k = " + k);
         k = 30;                                          //此句编译错，必须去掉
      }
      static {    i = i/2;     }
}
```

请解释为什么 k＝30；语句编译会出错。_____。

请解释为什么 i 的输出值不是 15。_____。

请说明 final 关键字和 static 关键字的作用。_____

_____

_____。

(2) 理解 abstract 关键字。用抽象类和抽象方法代表设计不具体、不完善的类和方法，主要用在高层抽象中，以便提供统一的接口。输入以下程序并以 InheDemo.java 存盘，编译并运行，同时学习并掌握继承的工作原理。

```
abstract class Employee{                                  //抽象类
  String EmpName;
  char sex;
  double EmpSal;
  Employee(String en, char s, double es){
  EmpName = en;   sex = s;   EmpSal = es;
  }
public String getName(){
  return EmpName;
}
public char getSex(){
  return sex;
}
public abstract double getSal();
public   void setSal(int basSal){
    EmpSal = basSal;
}}
/**********************************/
class Worker extends Employee{
  char category;
  boolean dressAllowance;
  Worker(String en, char s, double es, char c, boolean d){
  super(en,s,es);
  category = c;   dressAllowance = d;
  }
public char getCategory(){
  return category;
```

```
}
public boolean getDressAll(){
 return dressAllowance;
}
public double getSal(){
  return EmpSal;
}}
/**********************************/
class Superior extends Employee {
  int experience;
  boolean vehicle;
  double medicalAllowance;
  Superior(String en, char s, double es, int e, boolean v, double ma){
  super(en,s,es);
  experience = e;
  vehicle = v;
  medicalAllowance = ma;
}
public int getExp(){
  return experience;
}
public boolean getVehicle(){
  return vehicle;
}
public double getMedicalAll(){
  return medicalAllowance;
}
public   double getSal(){
   return EmpSal * 4 + 1000 + medicalAllowance;
}}
/**********************************/
class Officer extends Superior{
  double travelAllowance;
  Officer(String en, char s,double es,int e, boolean v, double ma,double ta){
  super(en,s,es,e,v,ma);
  travelAllowance = ta;
}
public double getTravelAll(){
  return travelAllowance;
}
public double getSal(){
   return EmpSal * 2 + 200 + travelAllowance + medicalAllowance;
}}
/**********************************/
class Manager extends Superior{
    double clubAllowance;
  Manager(String en,char s,double es, int e, boolean v,double ma, double ca){
    super(en,s,es,e,v,ma);
    clubAllowance = ca;
}
public double getClubAll(){
     return clubAllowance;
}
public double getSal(){
```

```
        return EmpSal * 5 + 2000 + medicalAllowance + clubAllowance;
}}
/**********************************/
class InheDemo{
 public static void main(String args[])     {
    Worker w = new Worker("M.John",'M',1200.50,'B',true);
    System.out.println("工人信息: ");
    System.out.println("姓名: " + w.getName());
    System.out.println("性别: " + w.getSex());
    System.out.println("薪资: " + w.getSal());
    System.out.println("类别: " + w.getCategory());
    if(w.getDressAll()) System.out.println("提供服装津贴");
    else   System.out.println("未提供服装津贴");
    Officer o = new Officer("S.David",'F',2500.70,15,true,345.60,200);
    System.out.println("\n主任信息: ");
    System.out.println("姓名: " + o.getName());
    System.out.println("性别: " + o.getSex());
    System.out.println("薪资: " + o.getSal());
    System.out.println("工作经验: " + o.getExp() + "年");
    if(o.getVehicle())   System.out.println("提供交通工具");
    else   System.out.println("未提供交通工具");
    System.out.println("医疗津贴: " + o.getMedicalAll());
    System.out.println("出差津贴: " + o.getTravelAll());
    Manager m = new Manager("ArnoldShwaz",'M',4500.70,15,true,345.60,300);
    System.out.println("\n经理信息: ");
    System.out.println("姓名: " + m.getName());
    System.out.println("性别: " + m.getSex());
    System.out.println("薪资: " + m.getSal());
    System.out.println("工作经验: " + m.getExp() + "年");
    if(m.getVehicle())   System.out.println("提供交通工具");
    else   System.out.println("未提供交通工具");
    System.out.println("医疗津贴: " + m.getMedicalAll());
    System.out.println("会员津贴: " + m.getClubAll());
  }
}
```

① 读懂程序,完成图 5-1 所示的员工继承层次图,并填空。

② 分析程序的执行流程并写出程序执行结果。

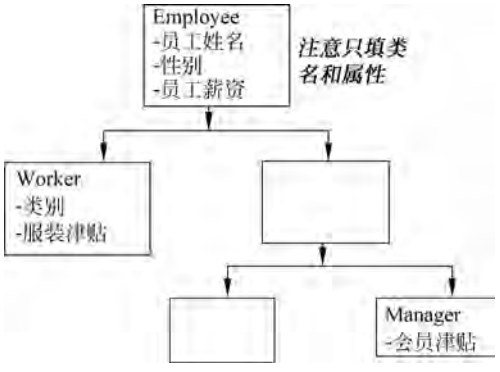


图 5-1　员工继承层次图

（3）理解接口 interface。Java 中 interface 关键字用来定义接口或界面，此接口实际上是一种规范或标准。在接口中不关心细节，只关心功能和相应的数据指标要求，所以在 Java 的 interface 接口中只存在两类东西：公有抽象的方法和公有静态的常量。

```
//interfacedemo.java
interface Computable {
   int M = 10;
   int f(int x);
  public abstract int g(int x,int y);
}
class A implements Computable {
 public int f(int x){ return M + 2 * x;}
 public int g(int x,int y){return M * (x + y);}
}
class B implements Computable {
 public int f(int x){return x * x * x;}
 public int g(int x,int y){return x * y * M;}
}
public class interfacedemo {
 public static void main(String[ ] args){
   Computable a = new A();                                    //可换为 A a = new A();
   Computable b = new B();                                    //可换为 B b = new B();
   System.out.println(a.M);
   System.out.println("" + a.f(20) + ", " + b.g(12,2));
   System.out.println(b.M);
   System.out.println("" + b.f(20) + ", " + b.g(12,2));
 }
}
```

接口(interface)可以看成是抽象类的一种特例，接口中的所有方法都必须是抽象的。接口中的方法定义默认为 public abstract 类型，接口中的成员变量类型默认为 public static final。

（4）理解包的使用。Java 中的包用来组织类和接口及资源，包的层次结构严格地对应于操作系统的目录结构，并注意以下编译和运行方式。

① 用 javac　－d　.　－classpath　.　*.java　　//编译

② 用 java　－classpath　.　包名.类名　　//执行

如本例：编译：javac　－d　.　Trangle.java

javac　flyhorse.java

运行：java　－cp　.　flyhorse

用记事本输入以下程序并以相应的文件名存盘。

```
//Trangle.java
package  www.horsefly;
public class Trangle  {
  double sideA,sideB,sideC;
  boolean flag;
  public Trangle(double a,double b,double c)  {
    sideA = a;sideB = b;sideC = c;
    if(a + b > c&&a + c > b&&c + b > a)  {
```

```
        System.out.println("我是一个三角形");
        flag = true;
      }  else  {
        System.out.println("我不是一个三角形");
        flag = false;
      }
    }
  public void jsmj () {
        if(flag)  {
          double p = (sideA + sideB + sideC)/2.0;
          double area = Math.sqrt(p * (p - sideA) * (p - sideB) * (p - sideC)) ;
          System.out.println("是一个三角形,能计算面积");
          System.out.println("面积是:" + area);
        }  else {
           System.out.println("不是一个三角形,不能计算面积");
        }
    }
  public void set(double a, double b, double c) {
      sideA = a;sideB = b;sideC = c;
      if(a + b > c&&a + c > b&&c + b > a) {
        flag = true;
      }  else  {
        flag = false;
      }
    }
}
//flyhorse.java
import www.horsefly.Trangle;
import java.util.Date;
public class flyhorse {
     public static void main(String args[ ])  {
          Trangle trangle = new Trangle(12,3,104);
          trangle.jsmj( );
          trangle.set(3,4,5);
          trangle.jsmj( );
          Date 今天 = new Date( );
          System.out.println("今天是:" + 今天);
     }
}
```

flyhorse 本地编译运行结果如图 5-2 所示。


图 5-2　本地编译运行结果

本地编译后生成的目录结构如图 5-3 所示。

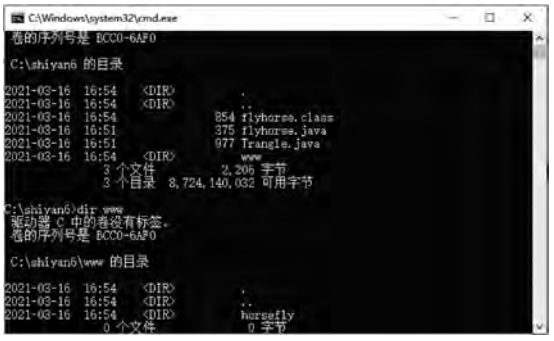


图 5-3　本地编译后的目录结构

请解释 import www.horsefly.Trangle;语句的作用：_____。

程序中 package www.horsefly;这条语句的含义是：

_____。

编译后，Trangle.class 所在的路径是：_____。

请回答在 Java 中，包名和操作系统的目录之间有什么关系：_____。

(5) 带包移植。将带包的程序移植到鲲鹏云服务器上，因为操作系统的不同，可能对目录和路径的使用也不相同，所以在远程鲲鹏云服务器上把上面的实验运行一遍，看看有哪些不同。

① 首先，登录华为云，单击“控制台”，选择原来购买区域的服务器，选择“更多”→“开机”选项，弹出如图 5-4 所示对话框，单击“是”按钮，等待服务器启动成功。

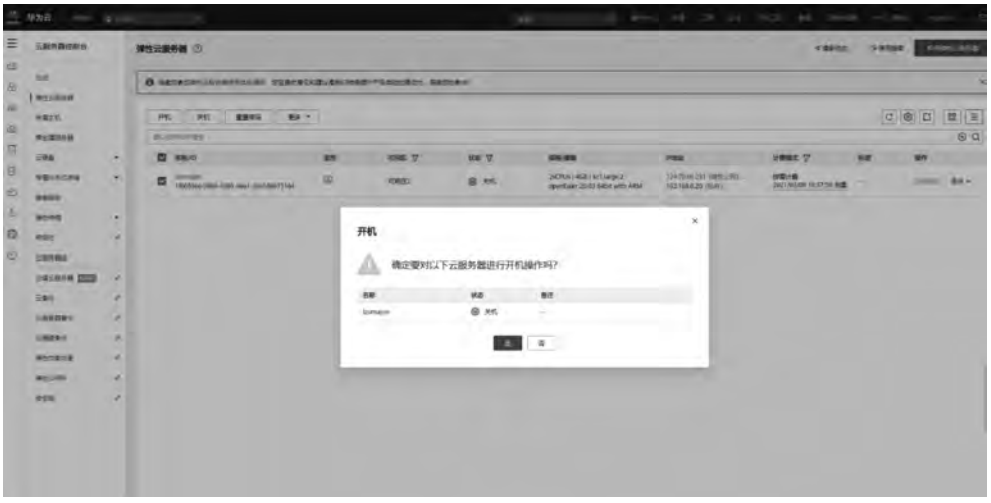


图 5-4　远程开机

② 打开 WinSCP 软件，输入远程鲲鹏云服务器的 IP 地址和 root 密码，等待连接成功后，如图 5-5 所示，将 flyhorse.java 和 Trangle.java 拖(复制)到右侧鲲鹏云服务器/root/java/目录下。

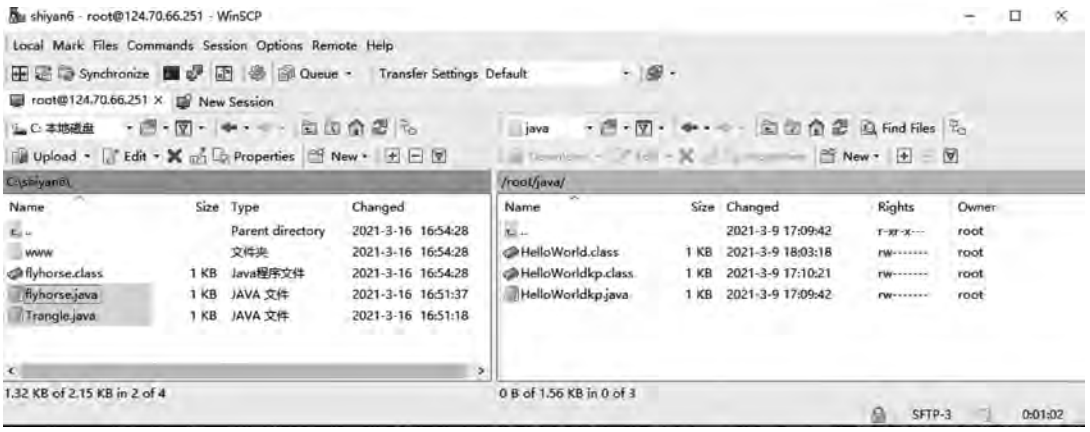


图 5-5　用 WinSCP 上传源程序文件

③ 双击 PuTTY，输入服务器 IP 地址和 root 密码，连接到远程鲲鹏云服务器，切换到 Java 目录下，用 ls 命令查看源程序是否已经上传成功，然后用以下命令编译和运行，运行结果如图 5-6 所示，然后再用 ls 命令查看是否多了一个 www 目录。


图 5-6　远程鲲鹏云服务器编译运行结果

④ 如果不继续在服务器上工作或做实验，一定记得关闭服务器，如图 5-7 所示，选择“更多”→“关机”选项，在弹出的对话框中选中“强制关机”复选框，再单击“是”按钮，然后就可以关闭浏览器了。

请读者解释 javac　-encoding　utf8　-d　.　*.java 代码的含义。

虽然本地采用 x86 架构的 Windows 系统，即一般是 Intel 的指令集，而鲲鹏云服务器采用的是鲲鹏指令集和 openEuler 操作系统，但 Java 程序在本地和远程鲲鹏云服务器不做任何修改就可以编译和运行，运行结果也是一致的，这是为什么？


图 5-7　关闭远程鲲鹏云服务器

## 5.3.2　填空实验

（1）理解单态设计模式。单态设计模式就是指设计的类不能随意地通过 new 来创建对象，在程序运行期间，只存在一个对象，该对象可完成所有的相关逻辑操作，不需要另外创建对象。单态设计模式有很多写法，主教材中介绍了饿汉模式，在实验教材中另介绍一种懒汉模式，懒汉模式就是类加载完成后并不马上创建对象，只有当进程需要该对象时才创建对象。如下面的示例，请参考主教材知识完成填空。

```
//Singleton.java
public class Singleton {
    private static Singleton instance = null;
    _________ Singleton() {                          //封装构造方法
    }

    public static Singleton getInstance(){
      if (instance == null) {
          ____________________;                      //创建 Singleton 对象
      }
      return instance;
    }
    public void displayinfo(){
        System.out.println("Hello, I am a Singleton Object!");
    }
}
//TestSingleton.java
public class TestSingleton {
   public static void main(String[] args) {
    //Singleton obj1 = new Singleton();              //无法创建对象
    Singleton obj = ___________________;            //获得 Singleton 对象
   __________;                                      //通过该对象调用 displayinfo()方法
   }
}
```

（2）理解接口和动态加载执行。以下程序是一个 Java 多态接口动态加载示例程序。

该程序是一个通用程序，用来计算每一种交通工具行驶 1000 千米所需的时间，已知每种交通工具的参数都是 3 个整数 A、B、C 的表达式。现有两种工具：Car 和 Plane，其中 Car 的速度运算公式为：A * B/C，Plane 的速度运算公式为：A＋B＋C。将来如果增加第 3 种交通工具的时候，不必修改此通用程序，只需要编写新的交通工具的程序即可。

**注意：**

① 充分利用接口的概念，用类对象充当参数。根据需求可以设计一个接口 Common 和三个类：一个主控类 ComputerTime，两个交通工具类 Plane、Car，未来增加新的交通工具增加新的类即可。其运行过程是，从命令行输入类 ComputerTime 的 4 个参数，第一个是交通工具的类型，第二、第三、第四个参数分别是整数 A、B、C，举例如下。

- 计算 Plane 的时间：java ComputerTime Plane 20 30 40
- 计算 Car 的时间：java ComputerTime Car 23 34 45

② 实例化对象除了主教材中学习的 new 构造方法，还有另外一种方法，通过动态加载类的方式实现：Class.forName(str).newInstance();，例如需要实例化一个 Plane 对象，只要调用 Class.forName(类名).newInstance()即可。

请填空完成程序，并学习包的使用、接口的使用、动态加载类等知识。

```
//Common.java
package CalTime.vehicle.all;
public interface Common {
   double runTimer(double a, double b, double c);  //定义接口方法
}
//文件 Plane.java
package CalTime.vehicle;
_____                                    //导入 Common 接口
public class Plane implements Common {
   public double runTimer(double a, double b, double c) {
      return (a + b + c);
   }
}
//文件 Car.java
package CalTime.vehicle;
import CalTime.vehicle.all.Common;
public class Car implements Common {
   _____                                 //仿照上面 Plane 类填写
   _____
   _____
}
//文件 ComputerTime.java
import CalTime.vehicle.all.Common;
import java.lang.*;
public class ComputerTime {
   public static void main(String args[]) {
      System.out.println("交通工具：" + args[0]);
      System.out.println(" 参数 A: " + args[1]);
      System.out.println(" 参数 B: " + args[2]);
      System.out.println(" 参数 C: " + args[3]);
      double A = Double.parseDouble(args[1]);
      double B = Double.parseDouble(args[2]);
      double C = Double.parseDouble(args[3]);
```

```
        double v,t;
        try {
            Common d = (Common)
    Class.forName("CalTime.vehicle." + args[0]).newInstance();//该语句的意思是动态加载
    //CalTime.vehicle.Car 类并创建一个实例对象,假设输入的参数是 Car.用接口 Common 定义的
    //引用 d 指向该对象
            v = d.runTimer(A,B,C);                    //调用 d 对象实现的接口方法 runTimer()
            t = 1000/v;
            System.out.println("平均速度: " + v + " km/h");
            System.out.println("运行时间: " + t + " 小时");
        } catch(Exception e)    {
            System.out.println("class not found");
        }
    }
}
```

请在当前目录编译，编译命令：javac -d . Common.java Car.java Plane.java ComputerTime.java

然后运行，运行命令：java -cp . ComputerTime Car 100 50 70

java -cp . ComputerTime Plane 130 50 70

请将运行结果抄在下面：

### 5.3.3 设计实验

（1）给 5.3.2 节填空实验第(2)题中的通用程序 ComputerTime 增加一种新的交通工具 Ship，假设该 Ship 工具的速度运算公式为：A+B/C，请设计类程序 Ship.java，编译后运行下列命令：

```
java ComputerTime Ship 40 100 25
```

输出结果如图 5-8 所示。

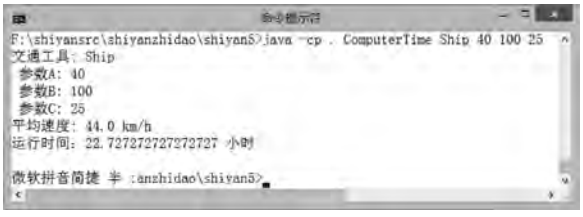


图 5-8 新交通工具 Ship 的运行时间

（2）请参考抽象类 Disk，设计两个类 SSD 和 SATA 分别表示一般固态硬盘和基于 SATA 口的固态硬盘，最后使用 TestDisk 类测试效果。

```
//Disk.java
public abstract class Disk {                          //抽象类 Disk
//不管什么磁盘,都具备"价格、厂家、型号"这几个属性,子类直接继承即可
```

```
    protected Float price;                          // 价格
    protected String manufacturer;                  // 厂家
    protected String model;                         // 型号
//不同的磁盘,读写方式差别很大,所以声明为 abstract,强制子类必须改写
    abstract void read();
    abstract void write();
//如下方法是各种磁盘通用的,直接继承即可,无须改写
    public String getManufacturer(){
        return manufacturer;
   }
    public void setManufacturer(String manufacturer){
        this.manufacturer = manufacturer;
    }
    public String getModel(){
        return model;
    }
    public void setModel(String model){
        this.model = model;
    }
    public Float getPrice(){
        return price;
    }
    public void setPrice(Float price){
        this.price = price;
    }
}
//TestDisk.java
public class TestDisk {
    public static void main(String[] args) {
        Disk disk;
        //disk = new Disk();          //编译错误: Cannot instantiate the type Disk
        disk = new SSD();
        disk.read();                  // OK, 最终调用 read(), 输出"SSD read, very fast"
        disk.write();                 // OK, 最终调用 write(), 输出"SSD write, very fast"
        System.out.println(disk.getManufacturer());
                                      // OK, 最终调用 getManufacturer(), 输出"Seagate"
        System.out.println(disk.getModel());
                                      // OK,最终调用 getModel(),输出"ssd"
        System.out.println(disk.getPrice());
                                      // OK,最终调用 getPrice(),输出"500.5"
        disk = new SATA();
        disk.read();                  // OK, 最终调用 read(), 输出"SATA read, very slow"
        disk.write();                 // OK, 最终调用 write(), 输出"SATA write, very slow"
        System.out.println(disk.getManufacturer());
                                      // OK, 最终调用 getManufacturer(), 输出"Maxtor"
        System.out.println(disk.getModel());
                                      // OK, 最终调用 getModel(),输出"sata"
        System.out.println(disk.getPrice());
                                      // OK, 最终调用 getPrice(),输出"100.5"
    }
}
```