第4章　封装与类

本章知识点

- 利用数据抽象和数据隐藏技术创建类
- 创建和使用对象
- 对属性和方法进行访问
- 方法的重载
- 构造方法及其使用
- this引用的用法
- static方法和属性的使用
- 类的组合方法
- 包的创建和使用

面向对象和面向过程的区别

面向对象和面向过程的区别

举例： 吃饭：

面向过程：去超市买菜—洗菜—切菜—炒菜—菜撑出来—吃

面向对象：上饭店吃，我--服务员（点菜）--厨师（做菜）--服务员（端菜）--我（吃）。

面向过程：强调的是每一个功能的步骤。

面向对象：强调的是对象，然后由对象去调用功能。

* 世界是由什么组成的？

名胜



动物 ， 植物……

人


物品





引入：面向对象程序设计思想

面向对象的方法学认为世界是由各种各样具有自己的运动规律和内部状态的对象所组成的，复杂的对象可以由简单的对象组合而成，整个世界都是由不同的对象经过层层组合构成的。

因此，人们应当按照面向对象的方法学来理解世界，直接通过对象及其相互关系来反映世界。

引入：面向对象程序设计思想

面向对象程序设计方法将数据和对数据的操作封装在一起，作为一个整体来处理。

面向对象的开发方法达到了软件工程的三个主要目标：重用性、灵活性和扩展性。

引入：面向对象程序设计思想

引入：面向对象程序设计思想

引出二个问题：

第一，用什么封装？

第二，什么是封装？如何封装?

本节课我们要解决问题：“第一，用什么封装？”。

- 什么是类？
- 类对象的创建；
- 引用类；

4.2 定义类

类作为一个抽象的数据类型，用来描述相同类型的对象。面向对象编程就是定义这些类。

1.类的概念

简单地说：

类是对对象的抽象描述；

对象是表示现实世界中某个具体的事物。

张三

将对象抽象为类

对类进行实例化

类（CLASS）
如：
“Man class ”

类和对象概述

有各自不同的状态

状态	行为
名字：保时捷	行驶
产地：德国	刹车
排量：3.0	
颜色：灰色	

状态	行为
名字：法拉利	行驶
产地：意大利	刹车
排量：4.0	
颜色：红色	

类和对象概述

类的实例化，是由类具体化对象

状态	行为
名字：保时捷	行驶
产地：德国	刹车
排量：3.0	
颜色：灰色	

对象

归纳一组相似对象的共性

属性

产地

车体颜色

行驶

刹车

…

方法

类

小汽车

4.2 定义类

4.2.2 使用class定义类

1．定义类的语法格式

[修饰符] class 类名 [extends 父类名] [implements 接口名列表]
{
    类成员变量声明;
    类方法声明;
}

Java中的类

类可以看成是创建Java对象的模板。

通过右边一个简单的类来理解下Java中类的定义：

```
public class Dog
{ String static breed;
int age;
 String color;
 void barking(){ }
void hungry(){ }
void sleeping(){ }
 }
```

Java中的类（总结）

一个类可以包含以下类型变量：

- **局部变量**：在方法、构造方法或者语句块中定义的变量被称为局部变量。变量声明和初始化都是在方法中，方法结束后，变量就会自动销毁。
- **成员变量**：成员变量是定义在类中，方法体之外的变量。这种变量在创建对象的时候实例化。成员变量可以被类中方法、构造方法和特定类的语句块访问。
- **类变量**：类变量也声明在类中，方法体之外，但必须声明为static类型。

一个类可以拥有多个方法，在上面的例子中：barking()、hungry()和sleeping()都是Dog类的方法。

例：定义一个表示二维平面上点的类

```
class Point（判断该类包含的类型变量）
{
   private int x,y;
   public void setPoint(int a,int b)
 {
     x=a;
     y=b;
   }
  public int getX() { return x; }
  public int getY() { return y; }
   public String toString()
    { return "["+x+","+y+"]"; }
}
```

4.3 对象和引用

- 对象的创建和使用
  - *创建对象*

[修饰符] 类名 对象名=new 类名(实参列表);

或

[修饰符] 类名 对象名;

对象名=new 类名(实参列表);

- 对象的创建和使用
  - *对象的使用—对象的实例化*

Point　thePoint;

注意：类属于复合数据类型，因此，在声明对象时，系统并没有为对象分配空间，用户需要应用new完成分配空间的任务。

thePoint=new　Point( );

- **对象的创建和使用**

  - *对象的使用---对象的引用*

    - 引用成员变量

      对象名.成员变量名

    - 引用方法

      对象名.方法名(参数列表)

例：定义一个表示圆形的类，能够计算圆面积和周长。

```
class Circle1
{
  float r;
  final double PI=3.14159;
  public double area()                    //计算面积
 {
     return PI*r*r;
 }
  public void  setR(float x)              //设置半径
   {
      r=x;
    }
```

```
public double  perimeter()            //计算周长
  {     return 2*PI*r;  }
public static void main(String args[])
 {    double x,y;
 Circle1 cir=new Circle1(); //创建Circle1类的对象cir
  cir.setR(12.35f);          //引用cir对象的setR()方法
  x=cir.area();              //引用cir对象的area()方法
  y=cir.perimeter();   //引用cir对象的perimeter()方法
  System.out.println("圆面积="+x+"\n圆周长="+y);
  }
}
```

程序运行结果如下：

圆面积=479.163190376011

圆周长=77.597275396842 96

第4章　封装与类

面向对象（2）

回顾

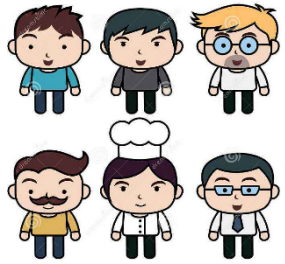
- 面向对象研究--对象
- 面向对象程序设计方法：
  1.创建类

```
类的定义格式：
[修饰符] Class   类名
{
        成员变量；（属性）
        成员方法；（行为）
}
```

将对象抽象为类

各个对象的
公共属性和行为
如：张三、李四等

类（CLASS）
如：
“男人类”



```
public class  Man
{  string name;
   string Id;
   string  job ;...
   public double worktime() {};
   public double freetime() {};…
}
```

回顾

- 面向对象研究----对象
- 面向对象程序设计方法：

  1.创建类

  2.创建类对象　语法格式如下：

类名　对象名=new　类名(实参列表)

Man Man1 =new Man();

```
public class Man
{ string name;
  string Id;
  string job ;...
  public double worktime() {};
  public double freetime() {};…
}
```

回顾

- 面向对象研究---对象
- 面向对象程序设计方法：

  1.创建类

  2.创建类对象

  3.引用类对象

◆引用成员变量

对象名.成员变量名

◆引用方法

对象名.方法名(参数列表)

```
public class  Man{
 string name;
  string Id;
  string  job ;...
public double worktime() {};
public double freetime() {};…
```

```
public class  Demo {
public static void main(String args[])
Man  Man1 =new Man();
 Man1.name=“张三” ;
 Man1.name=“李四” ;
 Man1.worktime();}
```

```
 public class  Man   ----- Man.java
{  string name;
   string Id;
   string  job ;
   public void  setname(string x) {   name=x;    //设置姓名};
   public double worktime() { System.out.println(name+”的工作时间是”+”8:30~17:00”)};
    public double freetime() { System.out.println(name+”的空闲时间是”+”17:00以后”)};
}
```

```
public class  Demo  ------ Demo.java
{public static void main(String args[])
     {  Man Man1 =new Man（）；
         Man1.name=“张三”;
         Man1.setname(“王五”);
        Man1.worktime();
     }
}
```

运行结果：

王五的工作时间是

8:30~17:00

本节课内容

本节知识点：

- 类、对象的应用（面向对象和面向过程的应用）
- 构造方法

重点、难点：

- 类、对象的应用（面向对象）
- 构造方法

一、类、对象的应用

类、对象的应用（编写程序）

案例1：将大象放进冰箱。

一般解决法：

1.打开冰箱门

2.装进大象

3.关闭冰箱门

一、类、对象的应用

类、对象的应用（编写程序）

案例1：将大象放进冰箱。

面向过程：

1.打开冰箱门
2.装进大象
3.关闭冰箱门

```
public class  Demo
{public static void main(String args[])
   {
    System.out.println(“打开冰箱门”);
    System.out.println(“装进大象”);
    System.out.println(“关闭冰箱门”);
   }
}
```

```
class Demo
{public static void main(String args[])
    {
     System.out.println(“打开冰箱门”);
     System.out.println(“装进大象”);
     System.out.println(“关闭冰箱门”);
  public static void open(){
     System.out.println(“打开冰箱门”); }
 public static voidn in(){
     System.out.println(“装进大象”); }
public static void close(){
     System.out.println(“关闭冰箱门”); }
  }
}
```

```
class Demo
{public static void main(String args[])
   {
    System.out.println("打开冰箱门");
    System.out.println("装进大象");
    System.out.println("关闭冰箱门");
 public static void open(){
    System.out.println("打开冰箱门"); }
 public static void in(){
    System.out.println("装进大象"); }
public static void close(){
    System.out.println("关闭冰箱门"); }
  }
}
```

一、类、对象的应用

```
class  Demo
{public static void main(String args[])
    { open();
       in();
      close();
  public static void open(){
     System.out.println(“打开冰箱门”)；}
 public static void in(){
     System.out.println(“装进大象”)；}
public static void close(){
     System.out.println(“关闭冰箱门”)；}
  }
}
```

一、类、对象的应用

应用（编写程序）

案例1：　将大象放进冰箱。

面向对象：（大象放进冰箱的分析）

1.有哪些类（由对象归纳得类）

大象、冰箱、主类（ **Demo** ）

一、类、对象的应用

应用（编写程序）

案例1：　将大象放进冰箱。

面向对象：（大象放进冰箱的分析）

1.有哪些类（由对象归纳得类）

大象、冰箱、主类（ **Demo** ）

2.每个类有哪些东西

大象：进去

冰箱：开门、关门

主类： **main方法**

一、类、对象的应用

应用（编写程序）

案例1：　将大象放进冰箱。

面向对象：（大象放进冰箱的分析）

1.有哪些类（由对象归纳得类）

大象、冰箱、主类（ **Demo** ）

2.每个类有哪些东西

大象：进去

冰箱：开门、关门

主类： **main方法**

3.类与类直接的关系是什么？

Demo中使用大象和冰箱类的功能

一、类、对象的应用

```
public class Big{
public static void in(){
    System.out.println(“装进大象”
}
```

```
public class Bx{
public static void open(){
    System.out.println(“打开冰箱门
; }
public static void close(){
    System.out.println(“关闭冰箱门”);
; }
}
```

```
public class Demo
{public static void main(String
args[]) {
 // 调用冰箱的开门
 //调用大象进入
 //调用冰箱的关门
  }
}
```

一、类、对象的应用

总结

- 面向对象的开发

就是不断的创建对象、使用对象、指挥对象做事情。

（注意：java中基本的单位是：类）

一、类、对象的应用

总结

- 面向对象的开发

  就是不断的创建对象、使用对象、指挥对象做事情。-----使用类

  （注意：java中基本的单位是：类）

一、类、对象的应用

```
public class Big{
public static void in(){
    System.out.println(“装进大象”
}
```

```
public class Bx{
public static void open(){
    System.out.println(“打开冰箱门
; }
public static void close(){
    System.out.println(“关闭冰箱门”);
; }
}
```

```
public class Demo
{public static void main(String
args[]) {
// 调用冰箱的开门
 //调用大象进入
 //调用冰箱的关门
}
}
```

一、类、对象的应用

总结

- 面向对象的开发

  就是不断的创建对象、使用对象、指挥对象做事情。-----使用类

  （注意：java中基本的单位是：类）

- 面向对象的设计

一、类、对象的应用

总结

- 面向对象的开发

  就是不断的创建对象、使用对象、指挥对象做事情。-----使用类

  （注意：java中基本的单位是：类）

- 面向对象的设计

  其实就是在管理和维护对象之间的关系

一、类、对象的应用

```
public class Big{
public static void in(){
    System.out.println(“装进大象”
}
```

```
public class Bx{
public static void open(){
    System.out.println(“打开冰箱门
; }
public static void close(){
    System.out.println(“关闭冰箱门
; }
}
```

```
public class Demo
{public static void main(String
args[]) {
  Bx bb=new Bx();
  bb. open();// 调用冰箱的开门
  Big aa=new Big();
  aa.in();  //调用大象进入
  bb. Close;//调用冰箱的关门
   }
}
```

运行结果：
开的冰箱门
装进大象
关闭冰箱门

一、类、对象的应用

总结

- 面向对象的开发

  就是不断的创建对象、使用对象、指挥对象做事情。-----使用类

  （注意：java中基本的单位是：类）

- 面向对象的设计

  其实就是在管理和维护对象之间的关系

- 面向对象的特征

  封装、继承、多态、[抽象]

一、类、对象的应用

案例二：用手机打电话。

分析： 手机：

特征：名字、价格、颜色

行为：打电话、发送消息、打游戏

```
Public class Phone{
    string name;
    int price;
    string color;

  void call(){System.out.println("打电话"); }
   void sendMessage(){System.out.println("发消息"); }
  void playGame(){System.out.println("玩游戏"); }
}
```

一、类、对象的应用

```
Public class phoneDemo{
public static void main(String args[]){
    Phone p=new phone();
    p.call(“马云”);
    p.sendMessage();
    p.palyGame();
}
 }
```

一、类、对象的应用

```
Public class phone{
        string name;
        int price;
        string color;

    void call(string name){System.out.println("打电话给
"+name); }
    void sendMessage(){System.out.println("发消息"); }
   void playGame(){System.out.println("玩游戏"); }
     }
```

一、类、对象的应用

```
Public class phoneDemo{
public static void main(String args[]){
    Phone p=new phone();
    p.call(“马云”);
    p.sendMessage();
    p.palyGame();
}
 }
```

运行结果：
打电话给马云
发消息
玩游戏

一、类、对象的应用

```
Public class phoneDemo{
public static void main(String args[]){
    Phone p=new phone();
    p.name="华为";
    p.price=3999;
    p.color="红色"
    p.call("马云");
    p.sendMessage();
    p.palyGame();
}
}
```

一、类、对象的应用

```
Public class phone{
        string name;
        int price;
        string color;

     void call(string name){System.out.println("打电话"+name); }
     void sendMessage(){System.out.println("发消息"); }
    void playGame(){System.out.println("玩游戏"); }
    viod show(){System.out.println(name+" "+ price"+" "+color)}
        }
```

一、类、对象的应用

```
Public class phoneDemo{
public static void main(String args[]){
    Phone p=new phone();
    p.name="华为";
    p.price=3999;
    p.color="红色"
    p.call("马云");
    p.sendMessage();
    p.palyGame();
    p.show();
}
}
```

运行结果：
打电话给马云
发消息…
玩游戏…
华为 3999 红色

一、类、对象的应用

练习：

- 建立一个汽车AutoMobile类，包括轮子个数（NumbersOfWheels），汽车颜色(autoMobileColor)，车身重量(autoMobileWeight)、速度(speed)等成员变量。并通过创建对象，实现：汽车具有加速（speedUp()），减速(speedDown)，停车(stop())等功能。

二、构造方法(补充)

定义：类中有一种特殊的成员方法，其方法名与类名相同，称为构造方法。

下面是一个构造方法示例：

```
 public class Puppy
{
public Puppy() {}
 public Puppy(String name)
{
// 这个构造器仅有一个参数name}
 }
```

二、构造方法(补充)—为什么要有构造方法

构造方法和对象的初始化

当使用**new**运算符实例化一个对象时，系统为对象创建内存区域并自动调用构造方法初始化成员变量。

下面是一个构造方法示例：

```
 public class Puppy
{
public Puppy() {}
 public Puppy(String
name){
// 这个构造器仅有一个参
数：name }
 }
```

下面是**new运算符实例化一个对象**

```
Puppy p=new  Puppy()
Puppy p1=new  Puppy("张
三")
```

（补充）Java实例——定义构造方法

```
class Student
{
  String name;
  String address;
  int grade;
  Student(String x1,String x2,int y)
 { //定义构造方法
     name=x1;
    address=x2;
    grade=x3;
}
```

```
public static void main(String args[])
{ Student zhang;    //声明并创建zhang对象
 zhang=new Student("张三","西安市兴庆路1号",3);
  Student wang;    //声明并创建wang对象
  wang=new Student("王五","西安市翠华路12号",4);

   System.out.println(zhang.name+zhang.address+zhang.g
   rade);
   System.out.println(wang.name+wang.address+wang.grad
   e);
 }
}
```

二、构造方法(补充)

（补充）构造方法的特点

- 构造方法名与类名相同；
- 构造方法没有返回值；
- 在创建一个对象的时候，至少要调用一个构造方法。
- 构造方法不能显式地直接调用；
- 构造方法的主要作用是对对象初始化。

Java实例—用构造方法初始化成员变量

```
class Triangle
{
   int x,y,z;
   public  Triangle(int i,int j,int k)          //声明构造方法
  {     x=i; y=j; z=k;   }
    public static boolean judge(Triangle m)
   { if(Math.sqrt(m.x*m.x+ m.y*m.y)==Math.sqrt(m.z*m.z))
              //引用Math类库的sqrt()方法
             return true;
          else
             return false;
  }}
```

```
public static void main(String args[]){
    Triangle t1=new Triangle(3,4,5);  //实例化对象t1，调用构造方法对其进行初始化
    if(judge(t1))          //调用judge()方法，判断t1的成员变量是
                           //否能构成直角三角型的3个边长
        System.out.println("这是一个直角三角形");
    else
        System.out.println("这不是一个直角三角形");
    }
}
```

二、构造方法(补充)

（补充）构造方法的特点

- 构造方法名与类名相同；
- 构造方法没有返回值；
- 在创建一个对象的时候，至少要调用一个构造方法。
- 构造方法不能显式地直接调用
- 构造方法的主要作用是对对象初始化。

- 每个类都有构造方法。如果没有显式地为类定义构造方法，Java编译器将会为该类提供一个默认构造方法。

（补充）缺省构造方法的使用

```
class Student
{
  String name;                                    //成员变量
  String address;                                 //成员变量
  int score;
//成员变量
  public void setMessage(String x1,String x2, int x3) //成员方法
  {
    name=x1;
    address=x2;
    score=x3;
  }
```

（补充）构造方法的特点

```
public static void main(String args[])
 {
    Student s1=new Student();   //创建Student类对象s1
    System.out.println(s1.name+"   "+s1.address+"
                                "+s1.score);
                                   //输出缺省构造方法的初始化结果
    s1.setMessage("张三","西安市兴庆路1号",75);
                        // 调用成员方法给成员变量赋值
    System.out.println(s1.name+"   "+s1.address+"
                                   "+s1.score);
   }
}
```

程序运行结果如下：

null　null　　0

张三　西安市兴庆路1号　75

二、构造方法(补充)

（补充）构造方法的特点

- 构造方法名与类名相同；
- 构造方法没有返回值；
- 在创建一个对象的时候，至少要调用一个构造方法。
- 构造方法不能显式地直接调用
- 构造方法的主要作用是对对象初始化。
- 每个类都有构造方法。如果没有显式地为类定义构造方法，Java编译器将会为该类提供一个默认构造方法。

- 一个类中可以定义多个构造方法，但各构造方法的参数表不能相同，即各构造方法的参数个数不同或参数类型不同。

（补充）*使用多个构造方法*

```
class Time1
{
  private int hour;                  //0-23
  private int minute;                 //0-59
  private int second;                 //0-59
  public Time1()
   { setTime (0,0,0); }
  public Time1(int hh)
    { setTime (hh,0,0); }
  public Time1 (int hh, int mm)
    { setTime (hh,mm,0); }
  public Time1(int hh, int mm, int ss)
    { setTime (hh,mm,ss); }
```

```
public void setTime (int hh, int mm, int ss)
   {
    hour = ((hh >= 0 && hh < 24) ? hh : 0);
    minute = ((mm >= 0 && mm < 60) ? mm : 0);
    second = ((ss >= 0 && ss < 60) ? ss : 0);}

public String toString()
   {
    return (hour + ":" +(minute < 10 ? "0" : "") + minute +
  ":" +
            (second < 10 ? "0" : "") + second);
   }
  }
```

```
public class MyTime1
 {
  private static Time1 t0, t1, t2, t3;
  public static void main(String args[])
{
    t0=new Time1();
    t1=new Time1(11);
    t2=new Time1(22, 22);
    t3=new Time1(11, 22, 33);
    System.out.println(" t0= " + t0.toString());
    System.out.println("t1= " + t1.toString());
    System.out.println("t2= " + t2.toString());
    System.out.println("t3= " + t3.toString());
} }
```

程序运行结果如下：

t0= 0:00:00

t1= 11:00:00

t2= 22:22:00

t3= 11:22:33

（补充）使用无参数的构造方法

```
class Time
 {  private int hour;      //0-23
  private int minute;    //0-59
  private int second;    //0-59
  public Time()
    { setTime(0,0,0); }
   public void setTime(int hh, int mm, int ss)
    {  hour = ((hh >= 0 && hh <24) ? hh : 0);
     minute = ((mm >= 0 && mm < 60) ? mm : 0);
     second = ((ss >= 0 && ss < 60) ? ss : 0);
    }
```

```
public String toString()
   {
     return (hour + ":" + (minute < 10 ? "0" : "") + minute +
  ":" +          (second < 10 ? "0" : "") + second );
    }
 }
public class MyTime
{
  public static void main(String args[])
  {
    Time time=new Time();
    time.setTime(11,22,33);
    System.out.println(" set time =" + time.toString());
   }
  }
```

运行结果如下：

```
set time =11:22:33
```

（补充）

- **对象的销毁**

通过new运算符实例化对象时，系统为对象分配所需的存储空间，存放其属性值。但内存空间有限，不能存放无限多的对象。为此，Java提供了资源回收机制，自动销毁无用对象，回收其所占用的存储空间。如果需要主动释放对象，或在释放对象时需要执行特定操作，则在类中可以定义finalize()方法。

```
public void finalize()
{
  方法体;
}
```

第4章　封装与类

面向对象（3）

上节课内容

本节知识点：

- 类、对象的应用
- 构造方法

重点、难点：

- 类、对象的应用
- 构造方法

回顾---构造方法（使用多个构造方法）

```
class Time1
{
  private int hour;                  //0-23
  private int minute;                 //0-59
  private int second;                 //0-59
  public Time1()
   { setTime (0,0,0); }
  public Time1(int hh)
    { setTime (hh,0,0); }
  public Time1 (int hh, int mm)
    { setTime (hh,mm,0); }
  public Time1(int hh, int mm, int ss)
    { setTime (hh,mm,ss); }
```

回顾---构造方法（使用多个构造方法）

```
public void setTime (int hh, int mm, int ss)
   {
    hour = ((hh >= 0 && hh < 24) ? hh : 0);
    minute = ((mm >= 0 && mm < 60) ? mm : 0);
    second = ((ss >= 0 && ss < 60) ? ss : 0);}

public String toString()
   {
    return (hour + ":" +(minute < 10 ? "0" : "") + minute +
  ":" +
            (second < 10 ? "0" : "") + second);
   }
  }
```

回顾---构造方法（使用多个构造方法）

```
public class MyTime1
 {
  private static Time1 t0, t1, t2, t3;
  public static void main(String args[])
{
    t0=new Time1();
    t1=new Time1(11);
    t2=new Time1(22, 22);
    t3=new Time1(11, 22, 33);
    System.out.println(" t0= " + t0.toString());
    System.out.println("t1= " + t1.toString());
    System.out.println("t2= " + t2.toString());
    System.out.println("t3= " + t3.toString());
} }
```

回顾---构造方法（使用多个构造方法）

程序运行结果如下：

t0= 0:00:00

t1= 11:00:00

t2= 22:22:00

t3= 11:22:33

回顾---构造方法（使用无参数的构造方法

```
class Time
 {  private int hour;      //0-23
  private int minute;    //0-59
  private int second;    //0-59
  public Time()
    { setTime(0,0,0); }
   public void setTime(int hh, int mm, int ss)
    {  hour = ((hh >= 0 && hh <24) ? hh : 0);
     minute = ((mm >= 0 && mm < 60) ? mm : 0);
     second = ((ss >= 0 && ss < 60) ? ss : 0);
    }
```

回顾---构造方法（使用无参数的构造方法

```
public String toString()
   {  return (hour + ":" + (minute < 10 ? "0" : "") +
  minute + ":" +  (second < 10 ? "0" : "") + second );
   }  }
public class MyTime
  {
  public static void main(String args[])
  { Time time=new Time();
    time.setTime(11,22,33);
    System.out.println(" set time =" + time.toString());
   } }
```

回顾---构造方法（使用无参数的构造方法

运行结果如下：

```
set time =11:22:33
```

本节课内容

本节知识点：

- this关键字的用法
- 类的封装

重点、难点：

- this关键字的用法
- 类的封装

4.3.2 this引用---定义

- this：代表一个引用，指向正在调用该方法的当前对象。

```
hotel2.setHotelName("MiniStarwood");
```

4.3.2 this引用---定义

- this：代表一个引用，指向正在调用该方法的当前对象。

```
hotel2.setHotelName("MiniStarwood");
```

```
public void setHotelName(String hotelName)
{
    this.hotelName = hotelName;
}
```

4.3.2 this引用---定义

- this：代表一个引用，指向正在调用该方法的当前对象。

```
hotel2.setHotelName("MiniStarwood");
```

```
public void setHotelName(String hotelName)
{
    this.hotelName = hotelName;
}
```

4.3.2 this引用---用法（好处）

```
 class Student
{
  String name; //成员变量
  String address;  //成员变量
  int score;  //成员变量
public void setMessage(String x1,String x2, int x3)
   //成员方法
  {name=x1;
    address=x2;
    score=x3;}
}
```

4.3.2 this引用---用法（好处）

```
public void setMessage(String x1,String x2, int x3)
 //成员方法
 { name=x1;
   address=x2;
   score=x3;}
```

```
public void setMessage(String name,String
 address, int score) //成员方法
 { this.name= name;
   this.address= address;
   this.score= score;}
```

观察this关键字的动向（好处）

```
public  class TestThis{
   TestThis(){
System.out.println(this)}
Public static viod main(String[] args){
 TestThis tt=new TestThis();}
}
```

aa.TestThis@de6ced

观察this关键字的动向（好处）

```
public  class TestThis{
   TestThis(){
System.out.println(this);}
   void show(){
System.out.println(this);}

public static viod main(String[] args){
 TestThis tt=new TestThis();
  tt.show();}
}
```

```
aa.TestThis@de6ced
aa.TestThis@de6ced
```

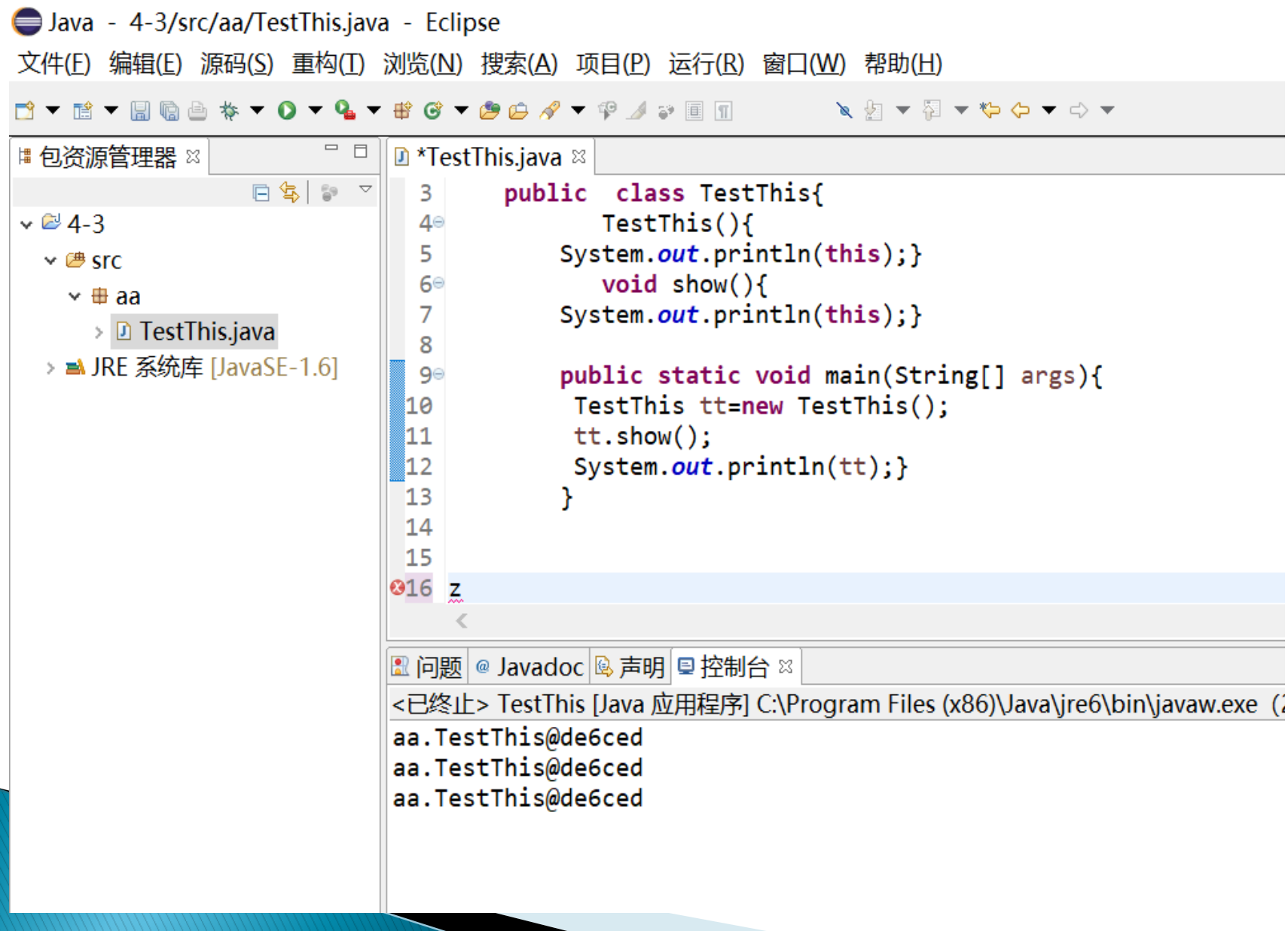
观察this关键字的动向（好处）

```
public  class TestThis{
   TestThis(){
System.out.println(this);}
   void show(){
System.out.println(this);}

public static viod main(String[] args){
 TestThis tt=new TestThis();
 tt.show();
 System.out.println(tt);}
}
```

```
aa.TestThis@de6ced
aa.TestThis@de6ced
aa.TestThis@de6ced
```

观察this关键字的动向（好处）



观察this关键字的动向（好处）

```
public  class TestThis{
String name; int age;
TestThis(){System.out.println(this);}
TestThis(String name,int age) {name=name;age=age;}
void show(){System.out.println(this);}
void show1(){System.out.println(name+" "+age);}
public static viod main(String[] args){
 TestThis tt=new TestThis();
 tt.show();
 System.out.println(tt);
TestThis tt1=new TestThis("王健林",18);
tt1.show1();
} }
```

因为不知道name是当前对象的name还是变量的name，就使用就近原则

```
aa.TestThis@de6ced
aa.TestThis@de6ced
aa.TestThis@de6ced
null 0
```

观察this关键字的动向（好处）

```
public  class TestThis{
String name; int age;
TestThis(){System.out.println(this);}
TestThis(String name,int age)
  {this.name=name;this.age=age;}
void show(){System.out.println(this);}
void show1(){System.out.println(name+" "+age);}
public static viod main(String[] args){
 TestThis tt=new TestThis();
 tt.show();
 System.out.println(tt);
TestThis tt1=new TestThis("王健林",18);
tt1.show1();
} }
```

```
aa.TestThis@de6ced
aa.TestThis@de6ced
aa.TestThis@de6ced
王健林 18
```

this关键字的应用（例子）

```
public class Boy {
String name;
Boy(){this("张三丰");}
Boy(String name){
this.name=name;}
void  show(){
System.out.println(this.name);}
public static void main(String[] args){
Boy b=new Boy();
b.show();}}
```

张三丰

this关键字的应用（例子）

```
public class Boy {
String name;
Boy(){//表示调用本类中的有参构造
this("张三丰");}
Boy(String name){
//表示当前对象
this.name=name;}
void  show(){System.out.println(this.name);}
public static void main(String[] args){
Boy b=new Boy();
b.show();}}
```

张三丰

this关键字的应用（例子）

```
public class Boy {
String name;
Boy(){//表示调用本类中的有参构造
this("张三丰");}
Boy(String name){
//表示当前对象
this.name=name;
this.show();}
void  show(){System.out.println(this.name);}
public static void main(String[] args){
Boy b=new Boy();
}}
```

张三丰

This关键字（重点）

总结：

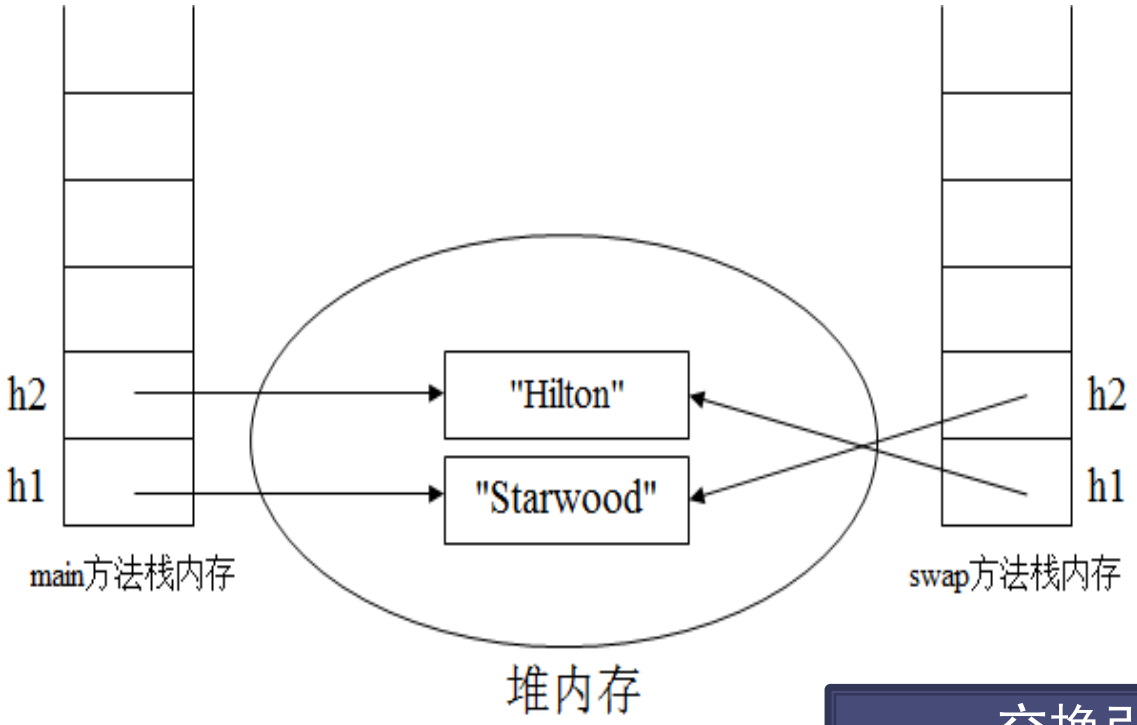
- 对于构造方法来说，this关键字代表当前正在构造的对象。
- 对于成员方法来说，this关键字代表当前正在调用的对象。

使用场合：

- 当我们形参变量名和成员变量名相同的时候，在我们方法体的内部会优先选择变量使用，此时就需要使用this.的方式明确使用的成员变量而不是形参变量（重中之重）。
- 在我们构造方法的第一行使用this()的方式可以调用本类中其他构造方法（了解）。

4.4 方法的参数传递（方法时已讲）

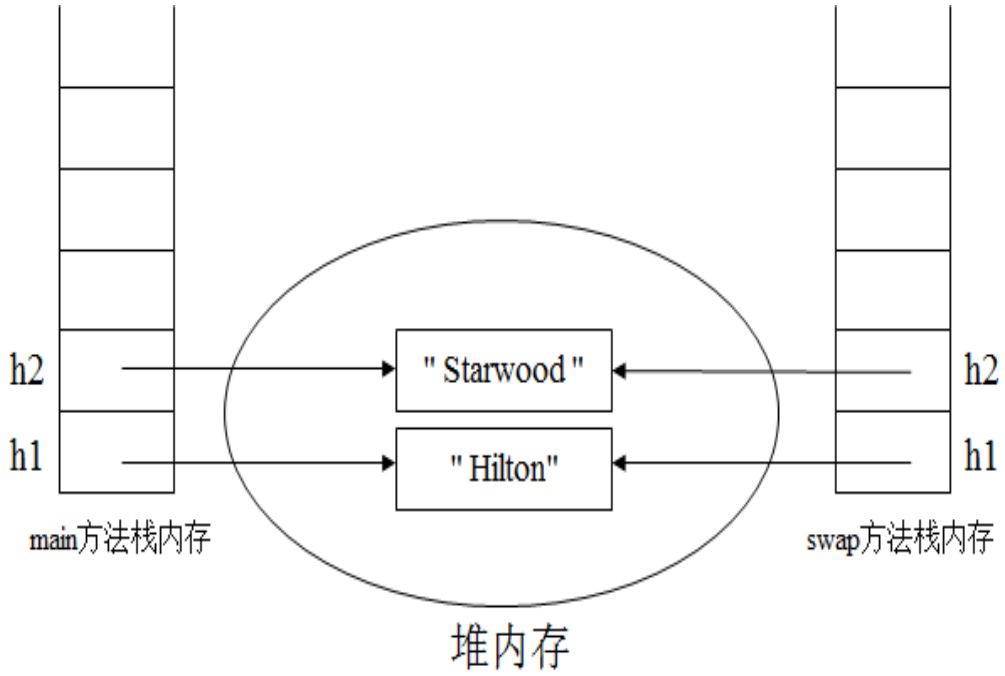
【例4-2】设计swap()方法交换两个Hotel的对象。



交换引用变量

4.4 方法的参数传递（方法时已讲）

【例4-2】设计swap()方法交换两个Hotel的对象。



交换引用变量指向的数据

类的封装

1、为什么要封装？
2、什么是封装？
3、怎么封装？
4、访问修饰符。

类的封装---为什么要封装？

代码如果不受限制，很多属性值是无效的。封装就是为了保证属性值有效的技术。

类的封装---什么是封装？

- 通常情况下我们可以在测试类中给成员变量赋值一些合法但不合理的数值，而程序的执行过程无法检测和提示，和我们现实的实际应用不太符合。

- 为了避免上面的情况，我们就需要对成员变量的赋值操作可以进行合理的封装和测试，该方式就是封装，其实封装就是保证我们变量值合理的机制。

类的封装---怎么要封装？

封装的流程：

- 私有化成员变量，使用private 关键字修饰。
- 需要提供共有的get成员变量和set成员变量的方法，在方法体中来进行合理判断
- 需要在构造方法中调用set成员变量进行合理的判断

类的封装—访问修饰符

- 成员访问权限
  - public（公有）：被public修饰的成员变量和成员方法可以在所有类中访问。
  - protected（保护）：被protected修饰的成员变量和成员方法可以在声明他们的类中访问，在该类的子类中访问，也可以在与该类位于同一包的类中访问，但不能在位于其它包的非子类中访问

类的封装—访问修饰符

- 缺省：缺省指不使用权限修饰符。不使用权限修饰符修饰的成员变量和方法可以在声明他们的类中访问，也可以在与该类位于同一包的类中访问，但不能在位于其它包的类中访问。
- private（私有）：被private修饰的成员变量和成员方法只能在声明他们的类中访问，而不能在其他类（包括其子类）中访问。

类的封装—访问修饰符

访问控制	本类	同一包中的类	其他包中子类	其他包中的类
public	√	√	√	√
protected	√	√	√	×
缺省	√	√	×	×
private	√	×	×	×

包的概念

包的定义：

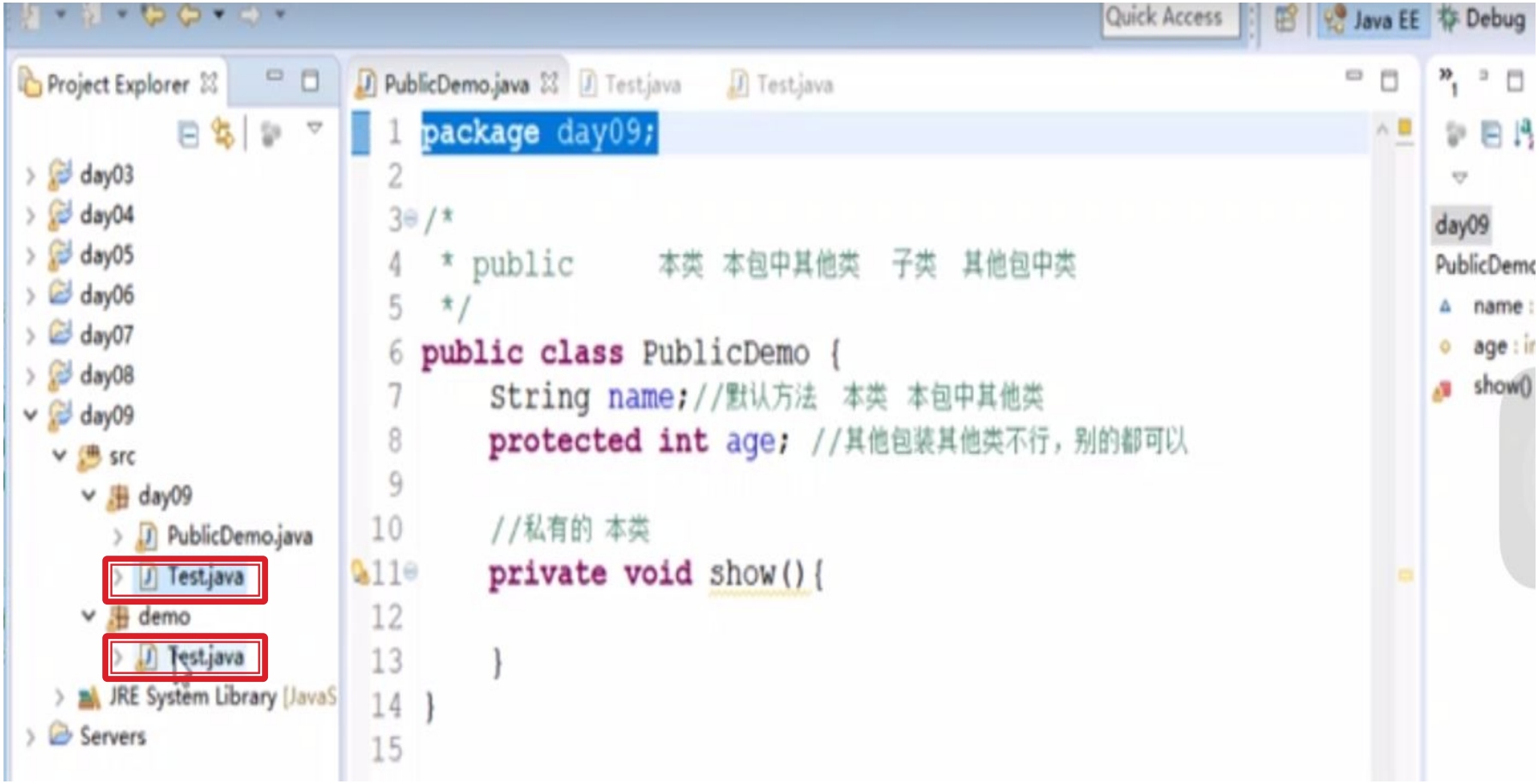
package 包名—表示创建单层包

package 包名1.包名2……包名n—表示创建多层 包也就是我们的多层目录

包的作用（好处）：

为了管理文件方便，避免文件同名引起的冲突

包的概念


Quick Access
Java EE
Debug
Project Explorer
day03
day04
day05
day06
day07
day08
day09
src
day09
PublicDemo.java
Test.java
demo
Test.java
JRE System Library [JavaS
Servers
PublicDemo.java
Test.java
Test.java
1 package day09;
2
3 /*
4 * public 本类 本包中其他类 子类 其他包中类
5 */
6 public class PublicDemo {
7 String name;//默认方法 本类 本包中其他类
8 protected int age; //其他包装其他类不行，别的都可以
9
10 //私有的 本类
11 private void show(){
12
13 }
14 }
15
day09
PublicDemo
name :
age : i
show()

封装类--- Girl.java

```
Public class Girl {
   String name;
   int age;
   boolean flag;
public void show(){
  System.out.println("我是"+name+ " ， 今年"+age+ "岁了，目前 " +( flag ? "入住": "没入住"));}
}
```

封装类--- TestGirl.java

```
public class TestGirl{
public static void main (String[] args){
Girl g=new Girl();
g.show();
}}
```

可以使用有参构造来创建对象（赋值/传参）

```
我是null，今年0岁了，目前 没入住
```

封装类--- Girl.java

```
Public class Girl {
   String name;
   int age;
   boolean flag;
public Girl(String name,int age, boolean flag){
 this.name=name;
 this. age=age;
 this. flag =flag;}
public void show(){
  System.out.println("我是"+name+ "， 今年"+age+ "目前" +( flag ? "入住": "没入住"));}
 }
```

封装类--- TestGirl.java

```
public class TestGirl{
public static void main (String[] args){
Girl g=new Girl();
g.show();
}
}
```

一旦自己编写构造，系统就不提供默认的无参构造，所以报错。

建议：有有参构造的时候，加上无参构造。

封装类--- Girl.java

```
Public class Girl {
   String name;
   int age;
   boolean flag;
public Girl(){}
public Girl(String name,int age, boolean flag){
 this.name=name;
 this. age=age;
 this. flag =flag;}
public void show(){
  System.out.println("我是"+name+ "， 今年"+age+ "目前
" +( flag ? "入住": "没入住"));}
 }
```

封装类--- TestGirl.java

```
public class TestGirl{
public static void main (String[] args){
Girl g=new Girl();
g.show();
Girl g1=new Girl("凤姐",-18,true);
g1.show();
    }
}
```

```
我是null，今年0岁了，目前 没入住
我是凤姐，今年-18岁了，目前 入住
```

类的封装---什么是封装？

- 通常情况下我们可以在测试类中给成员变量赋值一些合法但不合理的数值，而程序的执行过程无法检测和提示，和我们现实的实际应用不太符合。

- 为了避免上面的情况，我们就需要对成员变量的赋值操作可以进行合理的封装和测试，该方式就是封装，其实封装就是保证我们变量值合理的机制。

类的封装---怎么要封装？

封装的流程：

- 私有化成员变量，使用private 关键字修饰。
- 需要提供共有的get成员变量和set成员变量的方法，在方法体中来进行合理判断
- 需要在构造方法中调用set成员变量进行合理的判断

封装类--- Girl.java

```
Public class Girl {//1.私有化成员变量
   private String name;
   private int age;
   private boolean flag;
public Girl(){}
public Girl(String name,int age, boolean flag){
 this.name=name;
 this. age=age;
 this. flag =flag;}
public void show(){
  System.out.println("我是"+name+ "， 今年"+age+ "目前 " +( flag
? "入住": "没入住"));}
 }
```

封装类--- Girl.java

```
Public class Girl { //1.私有化成员变量
  private String name;     private int age;     private boolean flag;
public Girl(){}
public Girl(String name,int age, boolean flag){
 this.name=name;   this. age=age;   this. flag =flag;}
public void show(){ System.out.println("我是"+name+ " ， 今年
"+age+ "目前 " +( flag ? "入住": "没入住"));}
//2.提供公共的get和set方法，并且在方法体中进行合理的判断
public String getName(){  return name;  }
   public void setName(String name){ this.name=name; }
   public int getAge(){  return age;  }
   public void setAge(int age){ if(age>0 && age<=18){
    this.age=age;}else{System.out.println("年龄不合理");}}
   public boolean getFlag(){  return flag; }
   public void setFlag(){  this.flag=flag; } }
```

封装类--- TestGirl.java

```
public class TestGirl{
public static void main (String[] args){
Girl g=new Girl();
g.show();
Girl g1=new Girl("凤姐",-18,true);
g1.show();
    }
}
```

```
我是null，今年0岁了，目前 没入住
我是凤姐，今年-18岁了，目前 入住
```

封装类--- TestGirl.java

```
public class TestGirl{
public static void main (String[] args){
Girl g=new Girl();
g.show();
Girl g1=new Girl("凤姐",-18,true);
g1.setAge(19);
g1.show();
    }
}
```

```
我是null，今年0岁了，目前 没入住
年龄不合理
我是凤姐，今年-18岁了，目前 入住
```

封装类--- Girl.java

```
Public class Girl {//1.私有化成员变量
  private String name;     private int age;     private boolean flag;
public Girl(){}
public Girl(String name,int age, boolean flag){
 this.name=name;   this. age=age;   this. flag =flag;}
public void show(){ System.out.println("我是"+name+ " ， 今年
"+age+ "目前 " +( flag ? "入住": "没入住"));}
//2.提供公共的get和set方法，并且在方法体中进行合理的判断
public String getName(){  return name;  }
   public void setName(String name){ this.name=name; }
   public int getAge(){  return age;  }
   public void setAge(int age){ if(age>0 && age<=18){
    this.age=age;}else{System.out.println("年龄不合理");}}
   public boolean getFlag(){  return flag; }
   public void setFlag(){  this.flag=flag; } }
```

类的封装---怎么要封装？

封装的流程：

- 私有化成员变量，使用private 关键字修饰。
- 需要提供共有的get成员变量和set成员变量的方法，在方法体中来进行合理判断
- 需要在构造方法中调用set成员变量进行合理的判断

封装类--- Girl.java

```
Public class Girl {//1.私有化成员变量
  private String name;     private int age;     private boolean flag;
public Girl(){}
public Girl(String name,int age, boolean flag){
 setName(name); setAge(age); setFlag(flag);}
public void show(){ System.out.println("我是"+name+ " ， 今年
"+age+ "目前 " +( flag ? "入住": "没入住"));}
//2.提供公共的get和set方法，并且在方法体中进行合理的判断
public String getName(){  return name;  }
   public void setName(String name){ this.name=name; }
   public int getAge(){  return age;  }
   public void setAge(int age){ if(age>0 && age<=18){
    this.age=age;}else{System.out.println("年龄不合理");}}
   public boolean getFlag(){  return flag; }
   public void setFlag(){  this.flag=flag; } }
```

封装类--- TestGirl.java

```
public class TestGirl{
public static void main (String[] args){
Girl g=new Girl();
g.show();
Girl g1=new Girl("凤姐",-18,true);
g1.show();}
}
```

```
我是null，今年0岁了，目前 没入住
年龄不合理
我是凤姐，今年0岁了，目前 入住
```

封装类--- Girl.java

```
Public class Girl {//1.私有化成员变量
  private String name;  private int age=18;  private boolean flag;
public Girl(){}
public Girl(String name,int age, boolean flag){
 setName(name); setAge(age); setFlag(flag);}
public void show(){ System.out.println("我是"+name+ "，今年
"+age+ "目前 " +( flag ? "入住": "没入住"));}
//2.提供公共的get和set方法，并且在方法体中进行合理的判断
public String getName(){  return name;  }
   public void setName(String name){ this.name=name; }
   public int getAge(){  return age;  }
   public void setAge(int age){ if(age>0 && age<=18){
    this.age=age;}else{System.out.println("年龄不合理");}}
   public boolean getFlag(){  return flag; }
   public void setFlag(){  this.flag=flag; } }
```

封装类--- TestGirl.java

```
public class TestGirl{
public static void main (String[] args){
Girl g=new Girl();
g.show();
Girl g1=new Girl("凤姐",-18,true);
g1.show();
}
}
```

保护了成员变量的初始值，并且避免了成员变量赋值一些合法但不合理的数值

```
我是null，今年18岁了，目前 没入住
年龄不合理
我是凤姐，今年18岁了，目前 入住
```

封装—练习

- 自定义person类，实现该的封装，特征：姓名，年龄，国籍，提供一个打印特征的方法。

- 自定义Testperson类，在main方法中创建person类的对象并打印特性。

第4章　封装与类

面向对象（4）

本节课内容

本节知识点：

- static关键字
- 综合实践--酒店前台客房管理系统
- 类的继承（本书第五章）

重点、难点：

- static关键字
- 类的继承（本书第五章）

4.5 关于static--定义（为什么出现）

- 基本概念

通常情况下成员变量都隶属于对象层级，每个对象都拥有独立的内存空间来记录自己独有的成员变量，当所有对象的成员变量值都完全一样的时候，若每个对象单独记录会造成内存空间的浪费。

解决上述问题，应该将该成员变量由对象层级提升到类层级，在内存空间中只保留一份而且被所有对象所共享，为了实现该效果使用static关键字来进行修饰，表示静态的含义。

4.5.1 static成员--（使用）

- static数据成员（带有static的成员变量）：为类的对象所共享的数据。（举例）

例:static数据变量和非static成员变量的对比

```
class Student1
 {
  String name;                  //非static成员变量
  String address;               //非static成员变量
  static int count=0;           // static数据变量
  public Student1(String m, String a )
  { name=m;
    address=a;
    count=count+1;
  }
```

```
 public static void main(String args[])
{Student1 p1=new Student1("李明","西安市未央区");
 Student1 p2=new Student1("张敏", "上海市闽行区");
 System.out.println(p1.name+" "+p1.address+"
"+p1.count);
 Student1.count=Student1.count+1;
 System.out.println(p2.name+" "+p2.address+"
"+p2.count);
 p1.count=p1.count-1;
 System.out.println(p2.name+" "+p2.address+"
"+p2.count);
   }
}
```

```
李明 西安市未央区 2
张敏 上海市闽行区 3
张敏 上海市闽行区 2
```

4.5.1 static成员--（使用）

- static数据成员（带有static的成员变量）：为类的对象所共享的数据
- static方法（带有static的成员方法）：工具方法，不必创建对象直接使用类名即可调用。

  Math.PI

  Math.random()

  Math.sin()

虽然static成员也可以通过对象来引用，但是，绝对不鼓励这个方式。强烈建议使用**类名.成员**的形式进行存取，以区别于非static成员。

例： static方法和非static方法的对比

```
class Course
{ String no;                  //非static变量：课程编号
   int score;                 //非static变量：成绩
   static int sum=0;  //static变量变量：总成绩
   public  Course(String n, int s)
   { no=n;
      score=s; }
 public static void summarize(int s)  //static变量方
  法：统计总成绩
 {  sum+=s; }
}
```

```
public class Statistic
{  public static void main(String args[])
      { Course c1,c2;
        c1=new Course("210",90);
        Course.summarize(90);
        System.out.println("sum="+ c1.sum);
        c2=new Course("300",80);
        c2.summarize(80);
        System.out.println("sum="+Course.sum); }
    }
```

```
sum=90
sum=170
```

```
public class Statistic
{  public static void main(String args[])
     {
   Course.summarize(90);
      System.out.println("sum="+ Course.sum);
   Course.summarize(80);
      System.out.println("sum="+Course.sum); }
   }
```

```
sum=90
sum=170
```

4.5.1 static成员--（使用的注意事项）

- 在static方法中不允许使用非static成员
- 在非static方法中既可以使用非static成员，也可以使用static成员

【例4-3】定义含有static数据成员的Person类。

4.5.1 static成员--（使用）

```
public class Person {
        public static String nationality="Chinese";  //static成员
        private String name;    //非static成员
 public static String getNationality() {
return name+":"+nationality;     //static方法访问非static成员，报错
        }
public static void setNationality(String nationality) {  //static方法访问
static成员
                Person.nationality = nationality;
        }
        public void sayHello(){ //非static方法可以引用static成员
                System.out.println("hello,"+nationality+"!");
        }
       public static void main(String[] args) {
new Person().sayHello();  //创建Person类的匿名对象调用sayHello方法
```

总结static使用

- 使用方式

（1）对于非静态的成员方法来说，既可以访问非静态的成员同时也可以访问静态的成员；（成员：成员变量+成员方法）

（2）对于静态的成员方法来说，只能访问静态的成员不能访问非静态的成员；

（执行静态方法时候可以还没有创建对象，非静态成员隶属于对象层级）

（3）只有被所有对象共享的内容才能加static，static不能随便加。

4.5.2 变量的使用规则

- （1）如果某个信息需要在类的多个方法之间共享，则将其定义为数据成员即静态成员变量。
- （2）如果变量用于描述对象的静态信息，而且这个信息是与每个对象相关的，将其定义为对象的数据成员即非静态成员变量。
- （3）如果信息是与类相关的，即所有这个类的对象都具有相同的信息，将其定义为类的数据成员即静态成员变量。

例子：证明加static---对象共享

- 自定义Singleton类，实现该类的封装
- 自定义TestSingleton类，在main方法中有且只能得到Singleton类中的一个对象。

例子：证明加static---对象共享

```
//自定义Singleton类

public class Singleton {
}
```

例子：证明加static---对象共享

```
// 创建TestSingleton类

public class TestSingleton {
public static void main(String[] args){
Singleton s1=new Singleton();
Singleton s2=new Singleton();
System.out.println(s1);
System.out.println(s2);
System.out.println(s1==s2);}
}
```

```
aa.Singleton@c17164
aa.Singleton@1fb8ee3
false
```

例子：证明加static---对象共享

```
public class Singleton {
    //1.自定义无参构造，使用private关键字修饰
     private Singleton(){    }
   //2.需要提供一个共用的get成员变量的方法来将成员变量返回出去

}
```

例子：证明加static---对象共享

```
public class Singleton {
    private static Singleton s=new Singleton();
   //1.自定义无参构造，使用private关键字修饰
    private Singleton(){
    }
//2.需要提供一个共用的get成员变量的方法来将成员变量返回出去

}
```

例子：证明加static---对象共享

```
public class Singleton {
    private static Singleton s=new Singleton();
   //1.自定义无参构造，使用private关键字修饰
    private Singleton(){
    }
//2.需要提供一个共用的get成员变量的方法来将成员变量返回出去
    public static Singleton getInstance(){
    return s;
    }
}
```

例子：证明加static---对象共享

```
public class TestSingleton {
public static void main(String[] args){
Singleton s1=Singleton.getInstance();
Singleton s2=Singleton.getInstance();
System.out.println(s1);
System.out.println(s2);
System.out.println(s1==s2);
}
}
```

```
aa.Singleton@c17164
aa.Singleton@c17164
true
```

4.5.3 static代码块

静态语句块
构造块
构造方法体

```
Public class testBlock{
   {System.out.println(“构造块”); }
    public testBlock(){
          System.out.println(“构造方法体”); }
    static{
          System.out.println(“静态语句块”); }
public static void main(String[] args){
         testBlock tb=new testBlock();
          }
}
```

静态块一般用来在类加载以后初始化一些静态资源时候使用，如：加载配置文件

4.5.4 类常量的定义（已讲）

- final：放在变量声明前，表示该变量一旦被赋值后，就不能再改变其取值，即通常意义上的符号常量。
- static：如果这个常量属于类的每一个对象，则可以在其定义前加上修饰。

- static final ……

4.5.4 类常量的定义（已讲）

```
public class Hotel {
        private static final int HEIGHT=10;  //层数
        private static final int WIDTH=12;    //客房数

        private String hotelName;  //酒店名
        private String[][] rooms;    //酒店客房
        public Hotel(){
                rooms = new String[HEIGHT][WIDTH];
        }
……
}
```

4.6 包（再次说明）

1.包的定义：

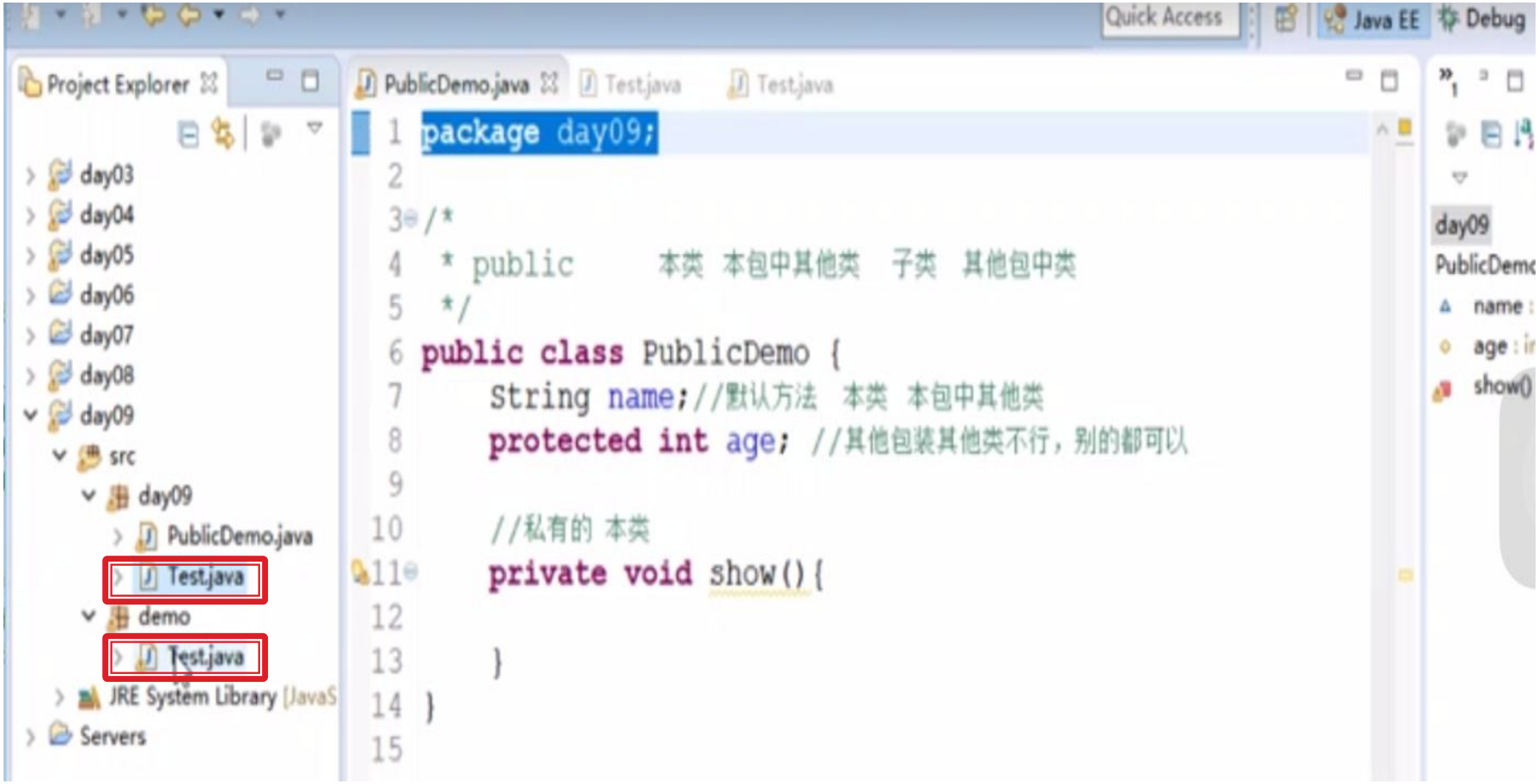
package 包名---表示创建单层包

package 包名1.包名2……包名n---表示创建多层包也就是我们的多层目录

2.包的作用（好处）：

为了管理文件方便，避免文件同名引起的冲突

包的概念（再次说明）


Quick Access
Java EE
Debug
Project Explorer
day03
day04
day05
day06
day07
day08
day09
src
day09
PublicDemo.java
Test.java
demo
Test.java
JRE System Library [JavaS
Servers
PublicDemo.java
Test.java
Test.java
1 package day09;
2
3 /*
4 * public 本类 本包中其他类 子类 其他包中类
5 */
6 public class PublicDemo {
7 String name;//默认方法 本类 本包中其他类
8 protected int age; //其他包装其他类不行，别的都可以
9
10 //私有的 本类
11 private void show(){
12
13 }
14 }
15
day09
PublicDemo
name :
age : i
show()

3. 类的导入

- 同一个包中的类可以互相访问
- 一个类要访问位于另一个包中的类（假设这个类允许被访问）：通过import语句导入类

- import不能自动引入包的子包，必须用显式声明的方式导入子包。

```
import java.awt.*;
import java.awt.event.*;
```

4.7 综合实践---酒店前台客房管理系统

- 设计一个酒店前台客房管理系统包括：酒店客房状态的查询，用户的入住、退房等功能。

- 系统命令如下

  search all：查询并输出酒店所有客房的状态。

  search 客房编号:查询该客房状态。

  in 客房编号 用户名：用户入住，例如“in 0306 Grace”表示姓名为Grace的客人入住0306客房。

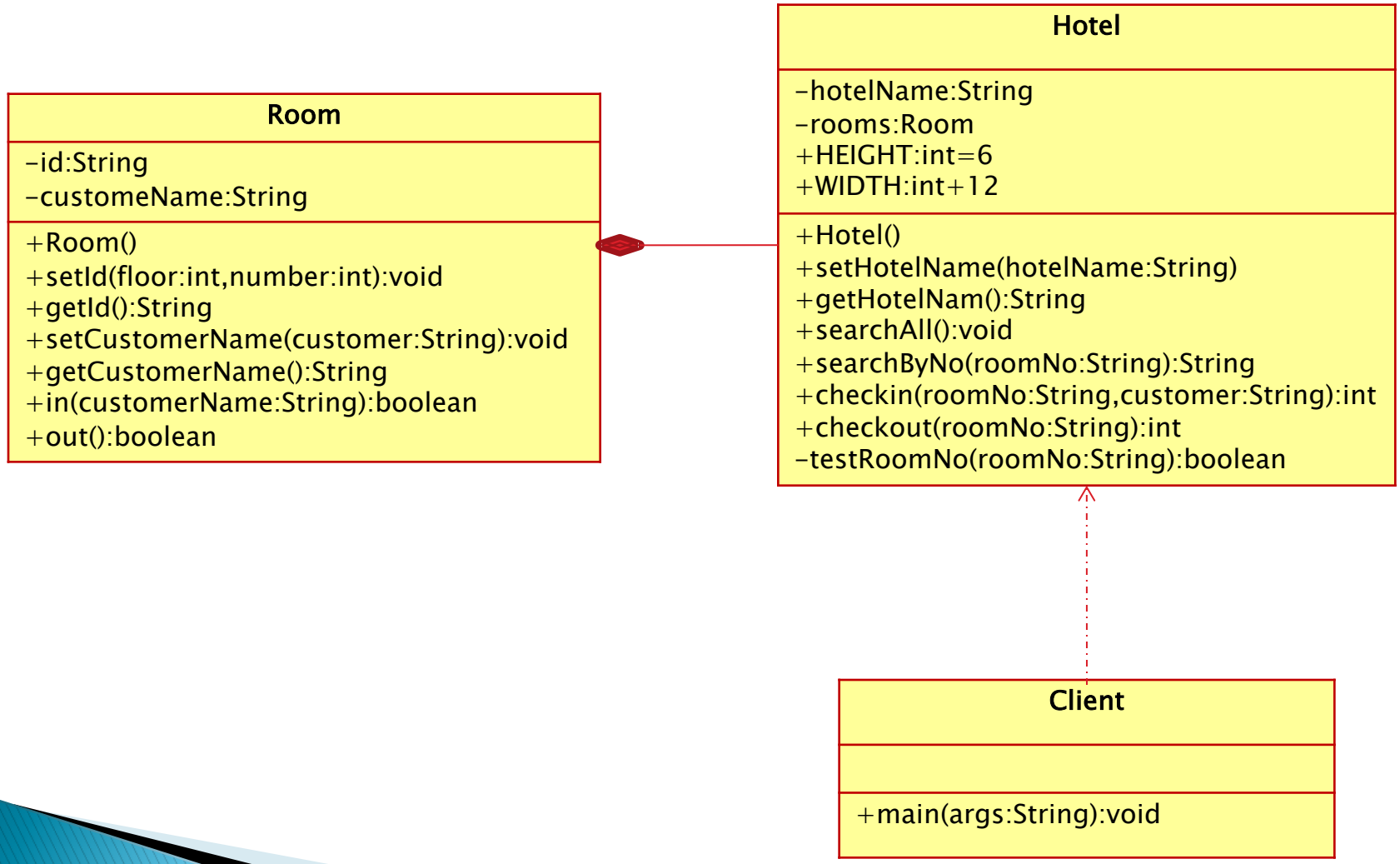
  quit：退出程序

4.7.1类的设计—组合关系

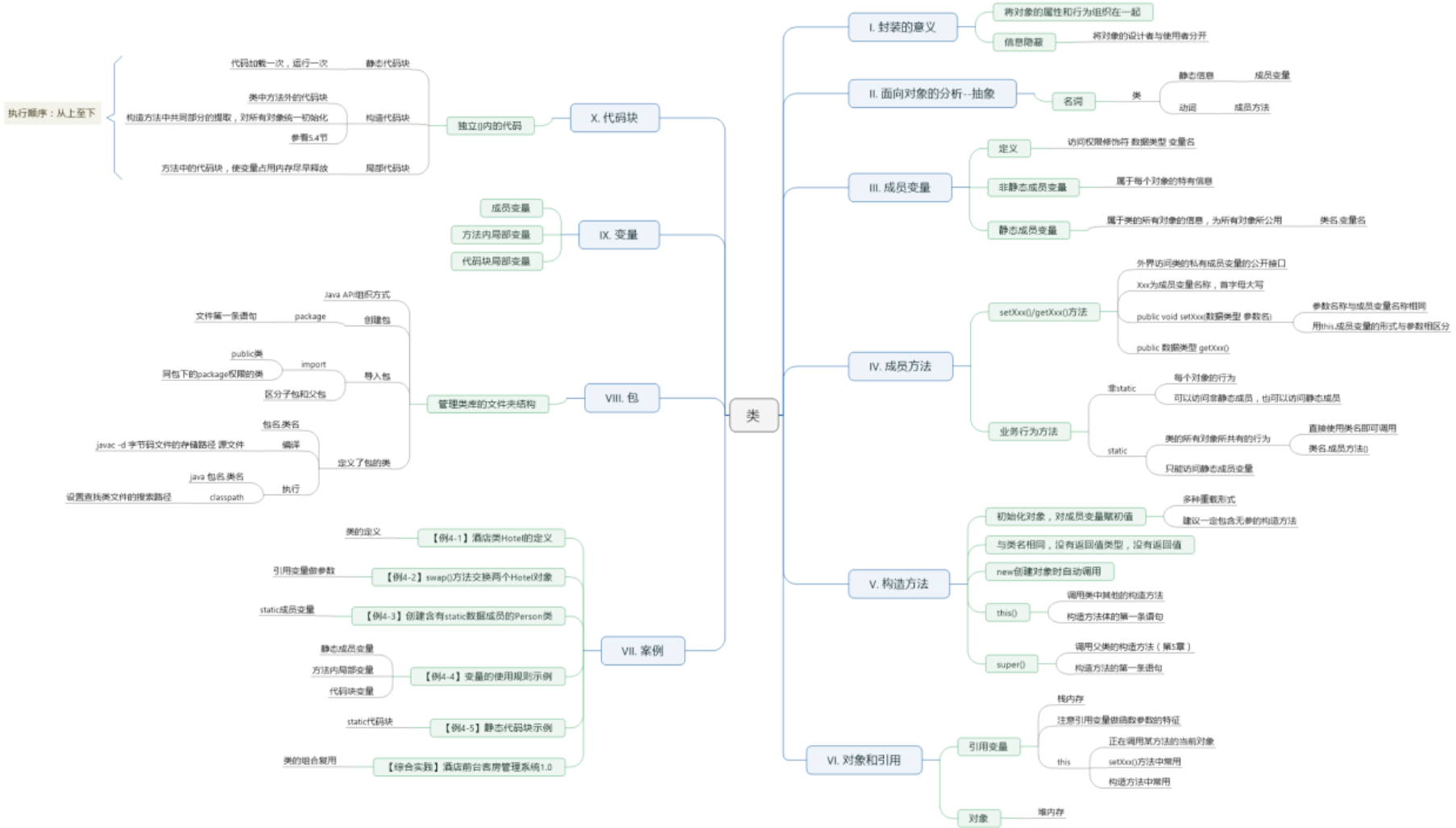
设计一个酒店前台客房管理系统

- 组合关系：“has a”，一个类的成员可以是其他类的引用。
- “HAS-A”关系体现了OO设计中类的专属性，类的专用化程度越高，在其他应用中就越可能被复用，
- OO设计不提倡用单独的类来完成大量不相关的操作。
- 组合是OO设计中大量存在的类间关系。

4.7.1 类的设计—组合关系



本章思维导图


类
I. 封装的意义
将对象的属性和行为组织在一起
信息隐藏
将对象的设计者与使用者分开
II. 面向对象的分析--抽象
名词
类
静态信息
成员变量
动词
成员方法
III. 成员变量
定义
访问权限修饰符 数据类型 变量名
非静态成员变量
属于每个对象的特有信息
静态成员变量
属于类的所有对象的信息，为所有对象所公用
类名.变量名
IV. 成员方法
setXxx()/getXxx()方法
外界访问类的私有成员变量的公开接口
Xxx为成员变量名称，首字母大写
public void setXxx(数据类型 参数名)
参数名称与成员变量名称相同
用this.成员变量的形式与参数相区分
public 数据类型 getXxx()
业务行为方法
非static
每个对象的行为
可以访问非静态成员，也可以访问静态成员
static
类的所有对象所共有的行为
直接使用类名即可调用
类名.成员方法()
只能访问静态成员变量
V. 构造方法
初始化对象，对成员变量赋初值
多种重载形式
建议一定包含无参的构造方法
与类名相同，没有返回值类型，没有返回值
new创建对象时自动调用
this()
调用类中其他的构造方法
构造方法体的第一条语句
super()
调用父类的构造方法（第5章）
构造方法的第一条语句
VI. 对象和引用
引用变量
栈内存
注意引用变量做函数参数的特征
this
正在调用某方法的当前对象
setXxx()方法中常用
构造方法中常用
对象
堆内存
X. 代码块
独立{}内的代码
执行顺序：从上至下
静态代码块
代码加载一次，运行一次
构造代码块
类中方法外的代码块
构造方法中共同部分的提取，对所有对象统一初始化
参看5.4节
局部代码块
方法中的代码块，使变量占用内存尽早释放
IX. 变量
成员变量
方法内局部变量
代码块局部变量
VIII. 包
管理类库的文件夹结构
创建包
Java API组织方式
package
文件第一条语句
导入包
import
public类
同包下的package权限的类
区分子包和父包
定义了包的类
包名.类名
编译
javac -d 字节码文件的存储路径 源文件
执行
java 包名.类名
classpath
设置查找类文件的搜索路径
VII. 案例
【例4-1】酒店类Hotel的定义
类的定义
【例4-2】swap()方法交换两个Hotel对象
引用变量做参数
【例4-3】创建含有static数据成员的Person类
static成员变量
【例4-4】变量的使用规则示例
静态成员变量
方法内局部变量
代码块变量
【例4-5】静态代码块示例
static代码块
【综合实践】酒店前台客房管理系统1.0
类的组合复用