第4章 类和对象

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- 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

面向过程

- 面向过程的程序设计中，问题被看作一系列需要完成的功能模块，函数（泛指高级语言实现功能模块的实体）用于完成这些任务，解决问题的焦点是编写函数，函数是面向过程的，它关注如何依据规定的条件完成指定的任务。
- 在多函数程序中，许多重要的数据被放置在全局数据区，这样它们可以被所有的函数访问（每个函数还可以具有它们自己的局部数据），这种数据和对数据的操作相分离的结构很容易造成全局数据在没商量的情况下被改动，因而程序的正确性不易保证。

- 面向对象的程序设计将数据和对数据的操作行为封装在一起，作为一个相互依存、不可分割的整体--类。类中的大多数数据只能为本类的行为使用，类会提供公开的外部接口与外界进行通信。
- 类是抽象的数据类型，用类创建对象。
- 程序的执行，表现为一组对象之间的交互通信。对象之间通过公共接口进行通信，从而完成系统功能。
- 面向对象的程序模块间关系简单，程序的独立性高、数据安全。面向对象的显著特点包括：封装性、继承性和多态性。

4.1 封装的意义

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- 封装：把对象的属性和操作结合为一个独立的整体，并尽可能隐藏对象的内部细节。
- “封装”的两个含义
  - 把对象的全部属性和全部服务结合在一起，形成一个不可分割的独立单位
  - 实现“信息隐蔽”，尽可能隐藏对象的内部细节，对外界形成一个边界，只保留有限的外部接口与外界进行联系。
- 对象的数据封装特性彻底消除了面向过程的方法中数据与操作相分离带来了的种种问题，提高了程序的可复用性和可维护性，减轻了程序员维护与操作分离的数据的负担。

对象的数据封装性还可以把对象的私有数据和公有数据分离开，保护了私有数据，减少了模块间的干扰，达到降低程序复杂性、提高可控性的目的。

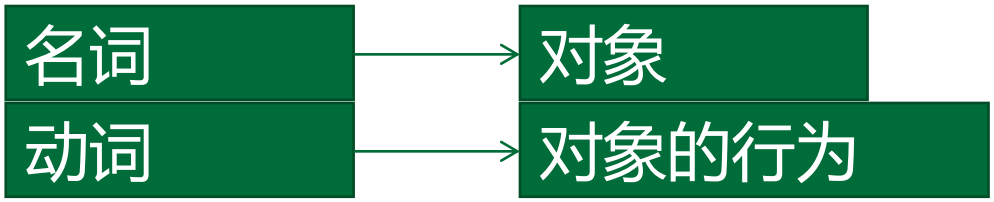
- 封装的目的在于把对象的设计者和对象的使用者分开，使用者不必知晓行为实现的细节，只需用设计者提供的接口来访问该对象。
- “黑盒”特性：一个类可以修改数据存储的方式，但只要仍提供相同的方法操作数据，其他对象就不会知道或者不会关心底层所发生的变化。就像一个公司如何运作客户通常无从知晓，也不需要知晓，无论它的内部如何调整，只要这个公司对外办公的业务接口没有发生变化，客户就不会受到影响。

- 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。

4.2.1 面向对象的分析

- “抽象”是面向对象设计中的重要环节。
- 面向过程程序设计：自顶向下，逐步求精
- 面向对象设计：从设计类开始，然后向类中添加方法
- 面向对象的思维方式：以对象为中心，分析对象的行为、状态，抽取出类的设计。

- 有一个酒店，酒店有若干客房，向客户提供查询、入住、退房等功能。

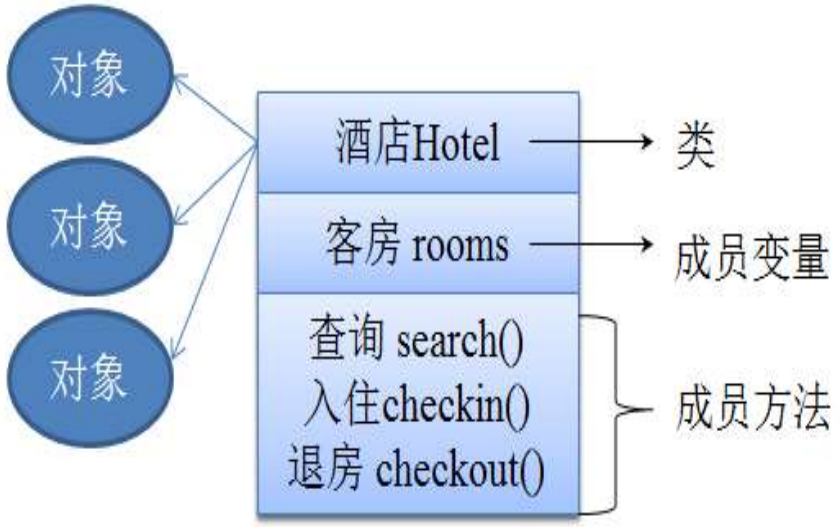


酒店 客房 客户

查询

入住

退房



1．定义类的语法格式

```
[类的修饰符] class 类名 [extends 父类名] {
        ……      //类体
}
```

【例4-1】定义一个酒店类Hotel。

（1）类的访问控制符
（2）数据成员（成员变量）：记录对象性质和状态的变量
（3）数据成员的set和get方法
（4）构造方法

- 1. 数据成员

  [修饰符] 数据类型 成员名[=默认值];

例:

- **private** String hotelName;
- **private** String[][] rooms;

数据类型	关键字	缺省数值
布尔型	boolean	false
字符型	char	'\u0000'
字节型	byte	0
短整型	short	0
整型	int	0
长整型	long	0
浮点型	float	0.0F
双精度型	double	0.0D
引用类型	类、接口	null

- 2. 方法
  - 一般是对类中的数据成员进行操作
  - 数据访问公共接口：如果类中的数据成员是private型的，则往往定义public的方法来设置数据成员的值或读取数据成员的值，set/get方法
    - 存：设置private成员变量的取值－setXxxx(xxx){}
    - 取：读取private成员变量的取值－xxx getXxxx(){}
  - 业务方法

- 3. 构造方法
  - 构造方法名与类名相同
  - 构造方法一般用于初始化类的对象
  - 创建类的对象时，new运算符为该对象分配内存，并调用构造方法来初始化该对象
    - 如果一个类中未定义构造方法，则编译时系统会自动提供一个缺省的无参的构造构造方法，其方法体为空。

      public 类名( ){ }
    - 至少写一个无参的构造方法

- this
  - 区分成员变量和方法的局部变量
- this()
  - 调用本类的其他构造方法

- 对象是类的一个实例
  - 类是抽象的，对象是具体的

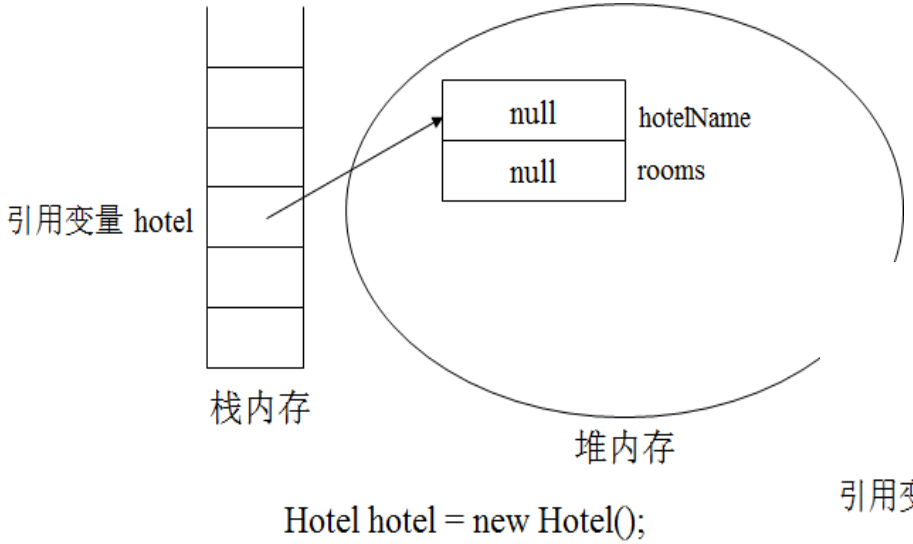
  - 酒店：类
  - 某个酒店：对象

4.3.1 对象和引用的关系

- 对象：通过new关键字调用某个构造方法创建，为该对象分配内存空间，并按照构造方法的方法体对对象的数据成员赋初值，创建好的对象在堆内存中。

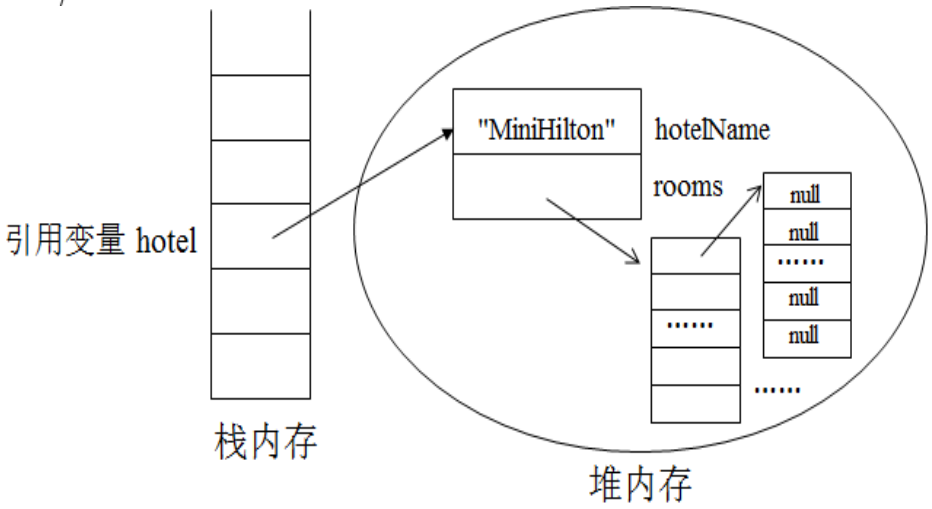
- 引用：Java不允许直接访问堆内存中的对象，只能通过对象的引用变量操作该对象，引用变量在栈内存中

- 【例】酒店问题中的对象和引用。



Hotel hotel = new Hotel();

垃圾回收机制



Hotel hotel = new Hotel("MiniHilton", new String[10][20]);

4.3.2 this引用

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- this：代表一个引用，指向正在调用该方法的当前对象。

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

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

练习

【题目】编写一个学生类，包括学号、姓名、性别、年龄

（1）编写合理的重载构造方法。

（2）编写各数据成员的set、get方法。

（3）编写测试类创建几个学生，打印他们的信息。

（4）在测试类中创建学生数组存储学生对象，打印数组中每个学生的信息。

```java
package com.company;

public class Student {
    private int studentNumber;
    private String name;
    private int age;
    private int gender; // 0代表女 1代表男

    public Student() {
        super();
    }

    public Student(int studentNumber, String name, int age, int gender) {
        super();
        this.studentNumber = studentNumber;
        this.name = name;
        this.age = age;
        this.gender = gender;
    }

    public void setStudentNumber(int studentNumber) {
        this.studentNumber = studentNumber;
    }

    public int getStudentNumber() { return studentNumber; }

    public void setName(String name) { this.name = name; }

    public String getName() { return this.name; }

    public void setAge(int age) { this.age = age; }

    public int getAge() { return this.age; }

    public void setGender(int gender) { this.gender = gender; }

    public int getGender() { return this.gender; }

}
```

练习

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

```java
package com.company;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<Student> list = new ArrayList<>();
        int[] studentNumArray = {1120213927, 1120213925, 1120213924, 1120213923, 1120213922, 1120213921, 1120213920, 1120213842};
        String[] nameArray = {"ZhangSan", "LiSi", "KangKang", "LiMing", "WangGang", "ZhaoXin", "LiuWei", "BaiZhanTang"};
        int[] ageArray = {28, 21, 27, 19, 22, 24, 20, 16};
        int[] genderArray = {0, 0, 1, 1, 1, 1, 0, 0};
        for (int i = 0; i < nameArray.length; i++) {
            list.add(new Student(studentNumArray[i], nameArray[i], ageArray[i], genderArray[i]));
        }
        //数组存储学生对象
        Student[] studentArray = new Student[nameArray.length];
        for (int i =0; i < nameArray.length; i++) {
            studentArray[i] = list.get(i);
            System.out.println("************************");
            System.out.println("学号: " + studentArray[i].getStudentNumber());
            System.out.println("姓名: " + studentArray[i].getName());
            System.out.println("年龄: " + studentArray[i].getAge());
            System.out.println("性别: " + (studentArray[i].getAge() == 0 ? "女" : "男"));
        }


    }
}
```

【例4-2】在酒店类Hotel中定义一个方法compareTo，比较两个酒店房间数的多少。

```
public class Hotel {
    private String hotelName;   //酒店名
    private String[][] rooms;

    public int compareTo(Hotel anotherHotel){}
}
```

```
hotel1.compareTo(hotel2);
```

this　this：代表一个引用，指向正在调用该方法的当前对象

```
public int compareTo(Hotel anotherHotel){
    int thisRoomsCount = this.rooms.length * this.rooms[0].length;
    int anotherRoomsCount =
          anotherHotel.rooms.length * anotherHotel.rooms[0].length;
    return thisRoomsCount-anotherRoomsCount;
}

public static void main(String[] args) {
    Hotel hotel1 = new Hotel("MiniHilton", new String[10][20]);
    Hotel hotel2 = new Hotel("MiniStarwood", new String[15][20]);
    int res = hotel1.compareTo(hotel2);

    if(res>0){
        System.out.println("酒店1的房间更多 ");
    }else if(res<0){
        System.out.println("酒店2的房间更多");
    }else{
        System.out.println("两个酒店的房间一样多");
    }
}
```

作业题

编写一个坐标系中的“点”类Point。

（1）编写构造方法用x、y坐标初始化某个点public Point(int x, int y)

（2）重载构造方法初始化对角线上的点，public Point(int x)

（3）编写distance()方法计算当前点到原点的距离 public double distance()

（4）重载distance()方法，计算当前点到另外一个坐标的距离：

public double distance(int x, int y)

（5）重载distance方法，计算当前点到另外一个点的距离：

public double distance(Point other)

（6）编写测试类PointTest，创建几个点，计算它们之间的距离。

4.5 关于static

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- 当在类中声明一个成员时，可以指定它是为一个类的各个对象各自拥有(实例成员)，还是为一个类的所有对象共享(类范围的成员)。

- 类范围的成员称为静态成员，以关键字static声明。

- 静态成员变量最大的特性：不属于某个具体的对象，是所有对象所共享的

- 不属于某个具体的对象，是类的属性，所有对象共享的，不存储在某个对象的空间中。static修饰的成员变量与对象的构造无关，它是放在方法区的，而对象是放在堆区的

- static数据成员：为类的对象所共享的数据
- static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。既可以通过对象访问，也可以通过类名访问，但一般更推荐使用类名访问

- 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
- 生命周期伴随类的一生(即：随类的加载而创建，随类的卸载而销毁)

【例4-3】定义含有static数据成员的Person类。

4.5.1 static成员

```
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
        }
}
```

【练习】编写一个学生类，包括学号、姓名、性别、年龄和记录学生总数的数据成员。

- （1）编写合理的重载构造方法。
- （2）编写各数据成员的set、get方法。
- （3）在测试类中创建学生数组存储学生对象，打印数组中每个学生的信息和当前学生总数。

```java
public class Student {
    private static final double pi = 3.14;
    private int studentNumber;
    private String name;
    private int age;
    private int gender; // 0代表女 1代表男
    private static int studentTotalNum = 0;

    public Student() {
        super();
        setStudentTotalNum();
    }

    public Student(int studentNumber, String name, int age, int gender) {
        super();
        this.studentNumber = studentNumber;
        this.name = name;
        this.age = age;
        this.gender = gender;
        setStudentTotalNum();
    }

    public void setStudentNumber(int studentNumber) { this.studentNumber = studentNumber; }

    public int getStudentNumber() { return studentNumber; }

    public void setName(String name) { this.name = name; }

    public String getName() { return this.name; }

    public void setAge(int age) { this.age = age; }

    public int getAge() { return this.age; }

    public void setGender(int gender) { this.gender = gender; }

    public int getGender() { return this.gender; }

    public void setStudentTotalNum() { this.studentTotalNum++; }

    public int getStudentTotalNum() { return this.studentTotalNum; }
```

4.5.1 static成员

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

```java
package com.student;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<Student> list = new ArrayList<>();
        int[] studentNumArray = {1120213927, 1120213925, 1120213924, 1120213923, 1120213922, 1120213921, 1120213920, 1120213842};
        String[] nameArray = {"ZhangSan", "LiSi", "KangKang", "LiMing", "WangGang", "ZhaoXin", "LiuWei", "BaiZhanTang"};
        int[] ageArray = {28, 21, 27, 19, 22, 24, 20, 16};
        int[] genderArray = {0, 0, 1, 1, 1, 1, 0, 0};
        for (int i = 0; i < nameArray.length; i++) {
            list.add(new Student(studentNumArray[i], nameArray[i], ageArray[i], genderArray[i]));
        }
        //数组存储学生对象
        Student[] studentArray = new Student[nameArray.length];
        for (int i =0; i < nameArray.length; i++) {
            studentArray[i] = list.get(i);
            System.out.println("************************");
            System.out.println("学号: " + studentArray[i].getStudentNumber());
            System.out.println("姓名: " + studentArray[i].getName());
            System.out.println("年龄: " + studentArray[i].getAge());
            System.out.println("性别: " + (studentArray[i].getAge() == 0 ? "女" : "男"));
            System.out.println("学生总数: " + studentArray[i].getStudentTotalNum());
        }
    }
}
```

4.5.2 变量的使用规则

- 数据成员/局部变量?
- 数据成员：属于每个对象/属于类?
- 方法局部变量/代码块局部变量?

- 数据成员存在于堆内存中，其释放由Java的垃圾回收机制控制），导致更大的内存开销；同时也扩大了变量的作用域，使程序的内聚性降低（软件设计的基本原则是高内聚、低耦合）

- 【例4-4】变量的使用规则示例。

- 代码段1：

```
public class TestScope1 {
        static int i;      // 定义一个类数据成员作为循环变量
        public static void main(String[] args) {
                for(i=0; i<10; i++){
                        System.out.println(i);
                }
        }
}
```

- 代码段2：

```
public class TestScope2 {
        public static void main(String[] args) {
                int i;    // 定义一个方法局部变量做循环变量
                for(i=0; i<10; i++){
                        System.out.println(i);
                }
        }
}
```

- 代码段3：

```
public class TestScope3 {
        public static void main(String[] args) {
                for(int i=0; i<10; i++){   // 定义一个代码块局部变量作为循环变量
                        System.out.println(i);
                }
        }
}
```

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。
- （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。
- （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。

4.5.3 static代码块

- 如果在加载类时希望先进行一些特殊的初始化动作，可以使用static定义一个代码块，将期望最早执行的初始化任务写在代码块中。

- 【例4-5】静态代码块示例。

静态代码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。

```
public class StaticBlockTest {
        static{
                System.out.println("static代码区，类正在被加载
...");
        }

        public StaticBlockTest() {
                System.out.println("创建类的对象...");
        }

        public static void main(String[] args) {
                new StaticBlockTest();
                new StaticBlockTest();
        }
}
```

4.5.4 类常量的定义

- 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 ……

```
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];
        }
……
}
```

- “包” 方式：包与磁盘的文件系统结构相对应，一个包就相当于一个文件夹，包中的类相当于文件夹下的文件。

- Java用为类定义不同的包，即定义不同的存储位置的方式解决同名类的冲突。同时包也提供了类的分类管理，使类可以按功能、来源等分为不同的集合，便于组织和使用。

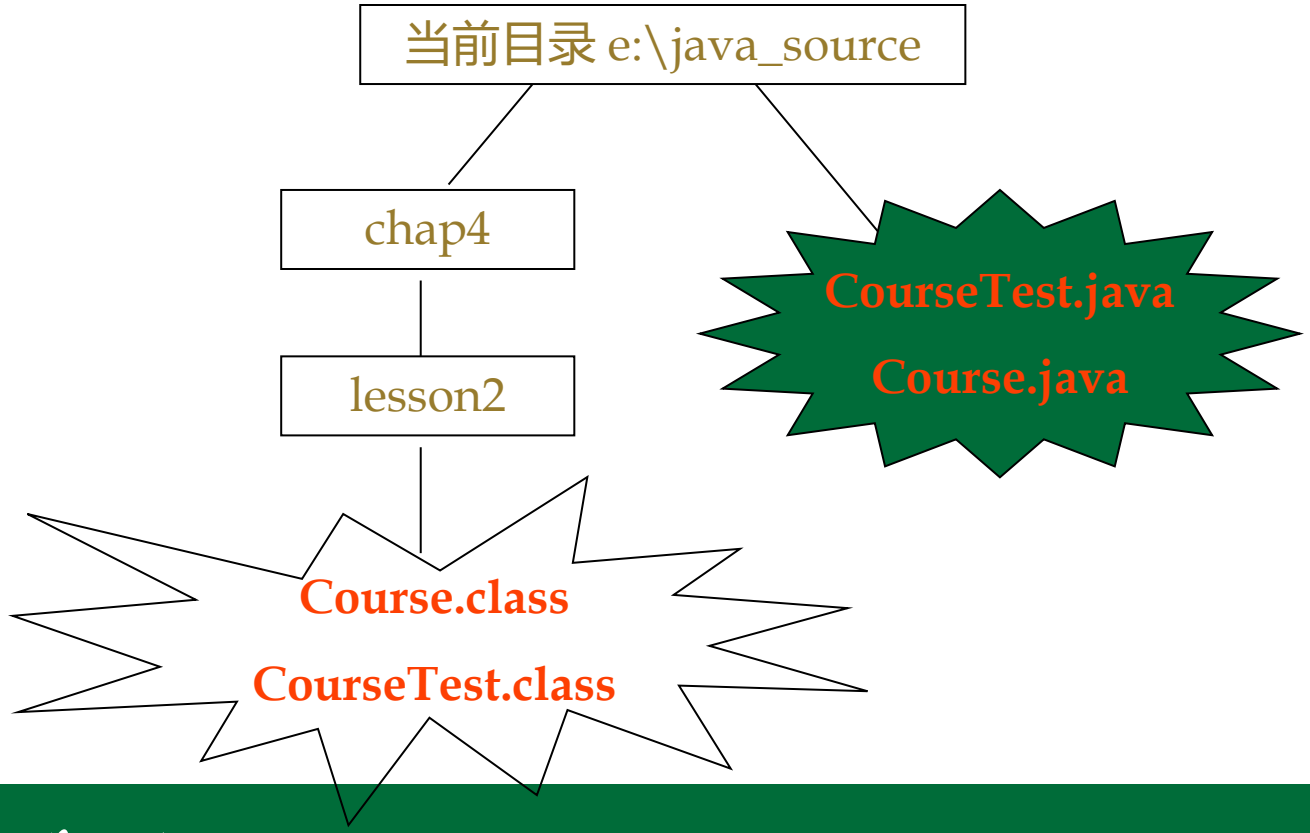
- java.lang：这个包下包含了Java语言的核心类，如String、Math、System等，这个包下的类在程序运行时自动导入。
- java.util：包含了大量实用的工具类和集合等，如Scanner、Date、Arrays、List等。
- java.text：包含了一些与Java格式化相关的类。
- java.awt：包含了构建图形用户界面的类。
- java.io：包含了输入/输出相关的类。
- java.sql：包含了JDBC数据库相关操作的类。
- javax.swing：包含了轻量级的构建图形用户界面的类。

4.6.1 包的创建

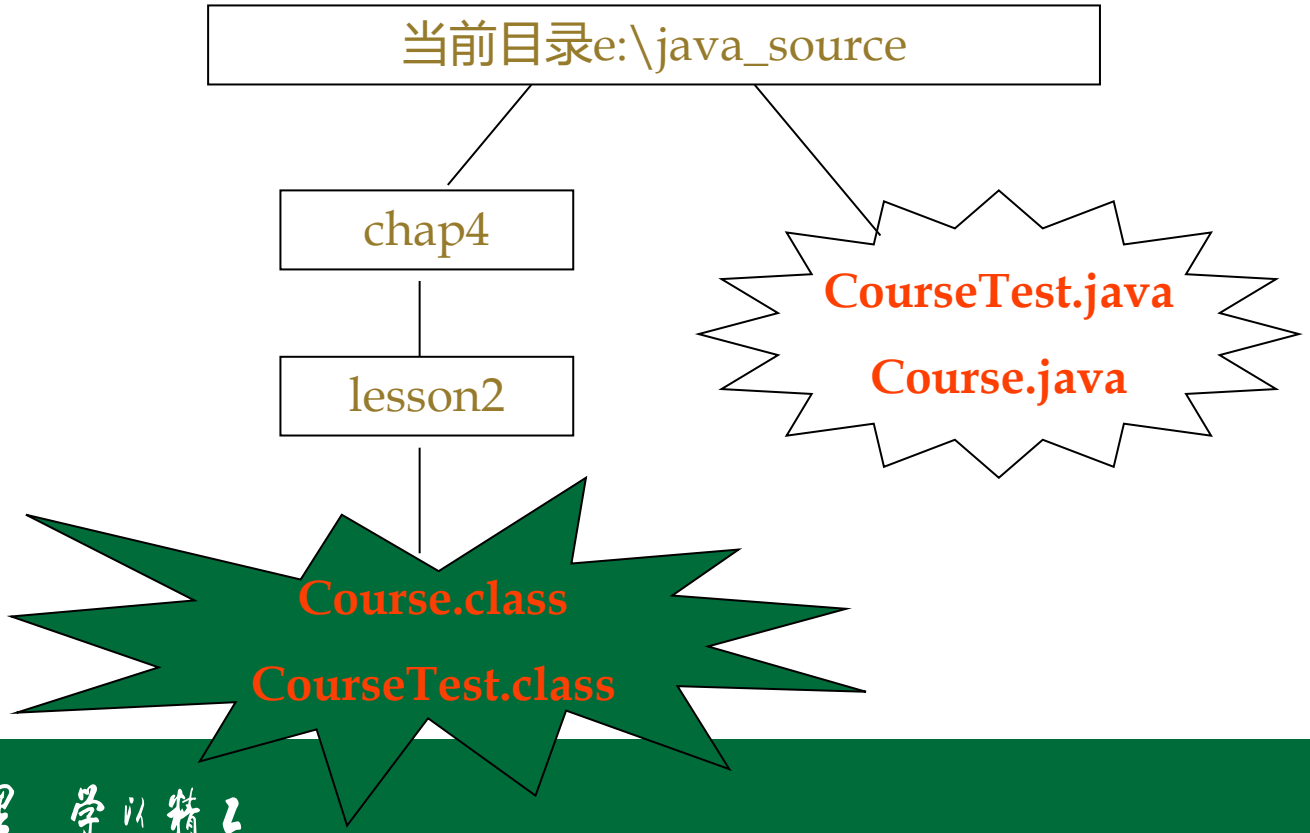
- 创建包：Java文件的第一条语句
- 语句格式
  package 包名[.包名[.包名]…];
- Java包的名字都是由小写字母组成，“.”指明包（文件夹）的层次。

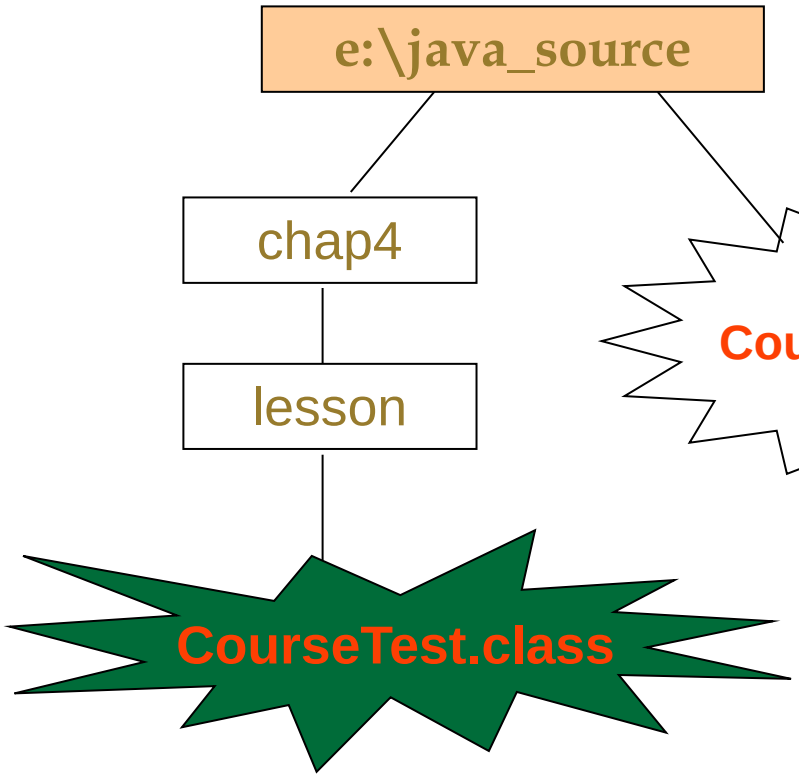
例如：　pakage　chap4;
　　package　chap4.lesson1;
　　包　子包

- javac –d . Course.java
- javac –d . CourseTest.java



- 带包定义的类的执行
  - 需要在包的上级目录执行类的class文件
  - java chap4.lesson2.CourseTest





- 设置环境变量classpath：指定寻找类字节码的路径－包的上级文件夹。

classpath=e:\java_source;

设置classpath后，在任意路径都可以执行包下面的类。

4.6.2 类的导入

- 同一个包中的类可以互相访问：无需导入
- 一个类要访问位于另一个包中的类（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。

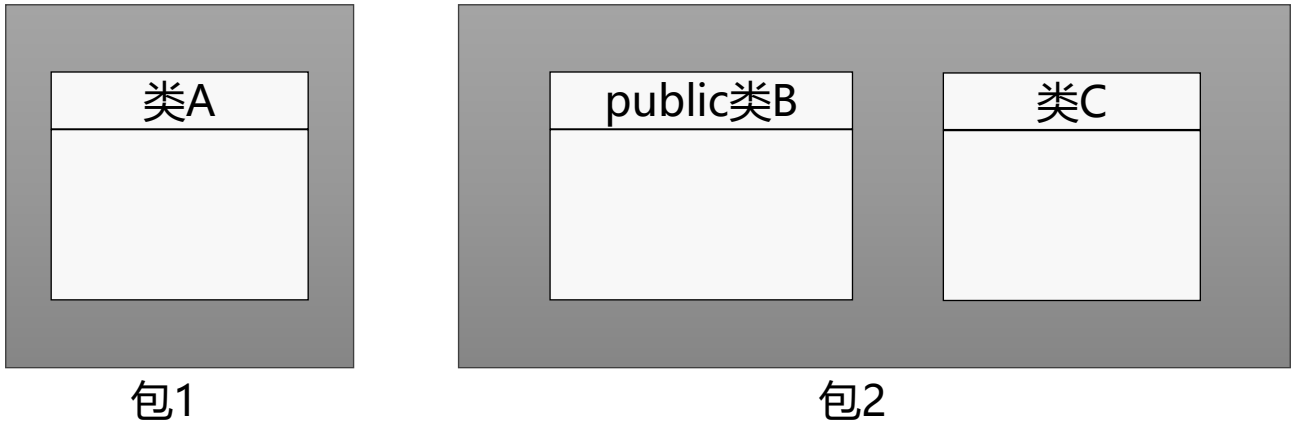
```
import java.awt.*;
import java.awt.event.*;
```

- 类的访问控制：public | 缺省

[类修饰符] class 类名

- public：该类**可以**被所有类访问（包内或包外均可）
  - 如果使用者与被使用者**在同一个包**下：直接使用
  - 如果使用者与被使用者**不在同一个包**下：通常先导入，再使用
- 缺省：该类只能在当前包中被使用

4.6.2 类的导入



类A可以访问类B吗? ✓　　类B可以访问类A吗? ✗　　类C可以访问类A吗? ✗

类A可以访问类C吗? ✗　　类B可以访问类C吗? ✓　　类C可以访问类B吗? ✓

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

1．编译

```
d:\java_source>  javac  –d  .  Hotel.java
```

参数1 “-d”：指定编译时在文件系统中创建与包对应的文件夹结构，必须使用。

参数2：指定类文件（包结构+class文件）的存储位置。该参数由程序设计者按需指定。

参数3：指定被编译的源文件。一定要保证按照路径可以找到被编译的源文件。

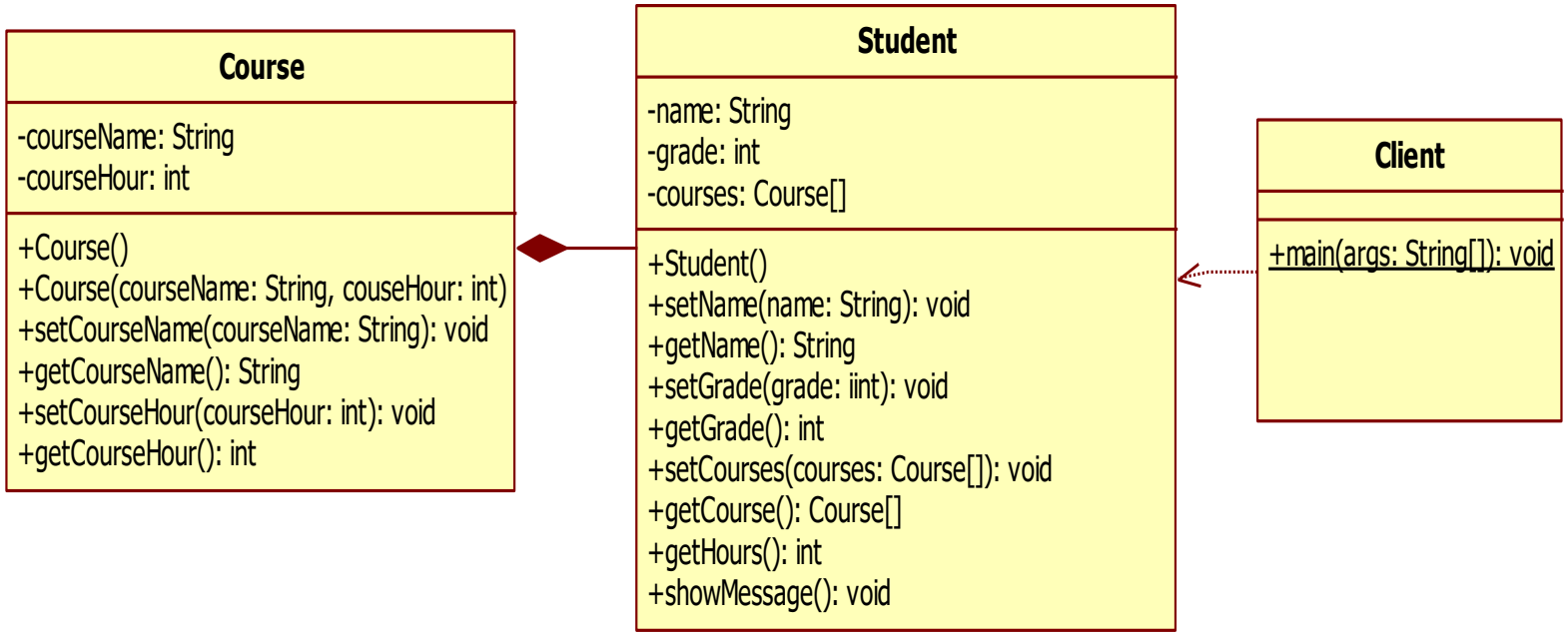
2．执行

- 完整的类名：包名.类名

- 执行class文件
  - 在当前路径下可以按照包名结构找到被执行的类文件
  - 必须使用完整的类名

4.7.1 类的设计—组合关系

- 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。



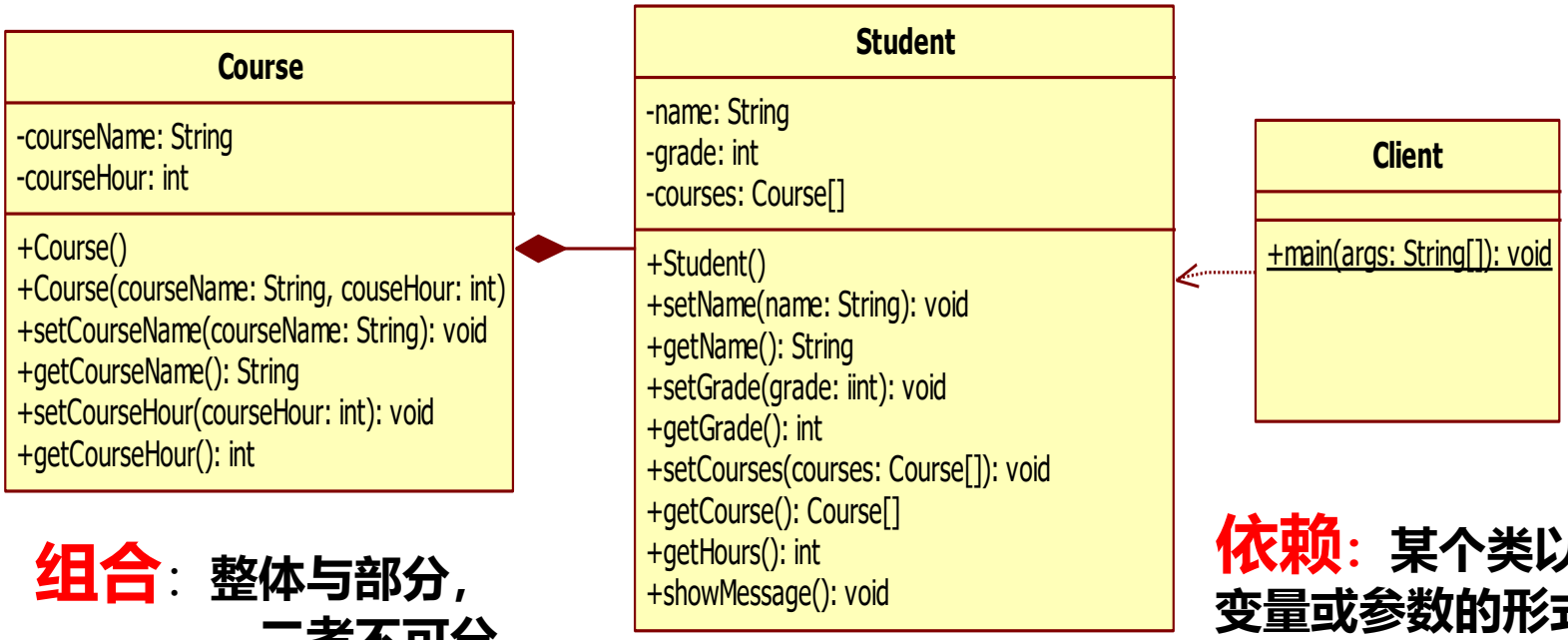
- UML图：Unified Modeling Language 统一建模语言
- 展示面向对象中的类之间的关系，每种关系的表达都有特定的符号。

<table>
<tr><th colspan="2">关 系</th><th colspan="2">Java中的表现与实现</th></tr>
<tr><td colspan="2">泛化（Generalization）</td><td colspan="2">extends继承</td></tr>
<tr><td colspan="2">实现（Realization）</td><td colspan="2">implements实现</td></tr>
<tr><td rowspan="3">关联</td><td>组合（Composition）</td><td rowspan="3">某个类以成员变量的形式出现在另一个类中</td><td>整体与部分，二者不可分
例如学生和学生证</td></tr>
<tr><td>聚合（Aggregation）</td><td>整体与部分，相对独立
例如学校和学生</td></tr>
<tr><td>关联（Association）</td><td>其他</td></tr>
<tr><td colspan="2">依赖（Dependency）</td><td colspan="2">某个类以局部变量或参数的形式出现在另一个类中</td></tr>
</table>

类与类之间的关系主要有6种：类与类之间的关系，从泛化→实现→组合→聚合→关联→依赖，关系越来越弱。

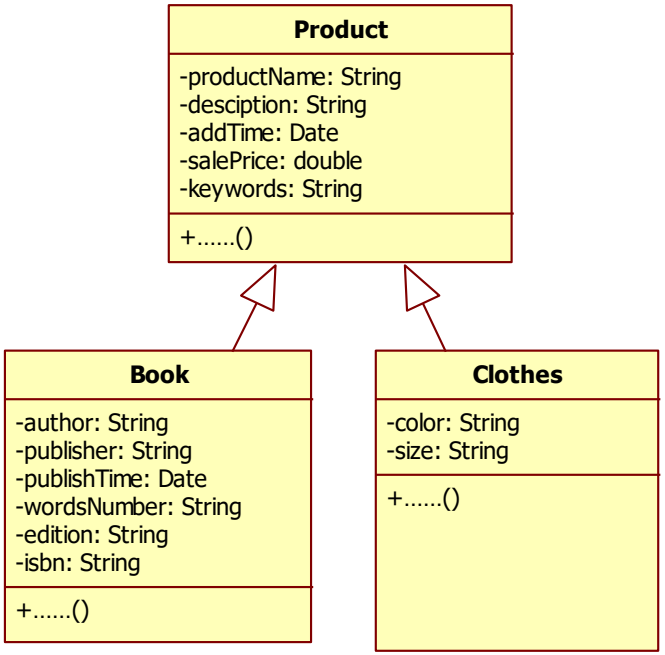
<table>
<tr><th colspan="2">关 系</th><th colspan="2">Java中的表现与实现</th></tr>
<tr><td colspan="2">泛化（Generalization）</td><td colspan="2">extends继承</td></tr>
<tr><td colspan="2">实现（Realization）</td><td colspan="2">implements实现</td></tr>
<tr><td rowspan="3">关联</td><td>组合（Composition）</td><td rowspan="3">某个类以成员变量的形式出现在另一个类中</td><td>整体与部分，二者不可分
例如学生和学生证</td></tr>
<tr><td>聚合（Aggregation）</td><td>整体与部分，相对独立
例如学校和学生</td></tr>
<tr><td>关联（Association）</td><td>其他</td></tr>
<tr><td colspan="2">依赖（Dependency）</td><td colspan="2">某个类以局部变量或参数的形式出现在另一个类中</td></tr>
</table>

关联关系：指某个类以成员变量的形式出现在另一个类中，而组合、聚合是关联的特殊表现，是引用类与被引用类之间存在整体和部分的关系。

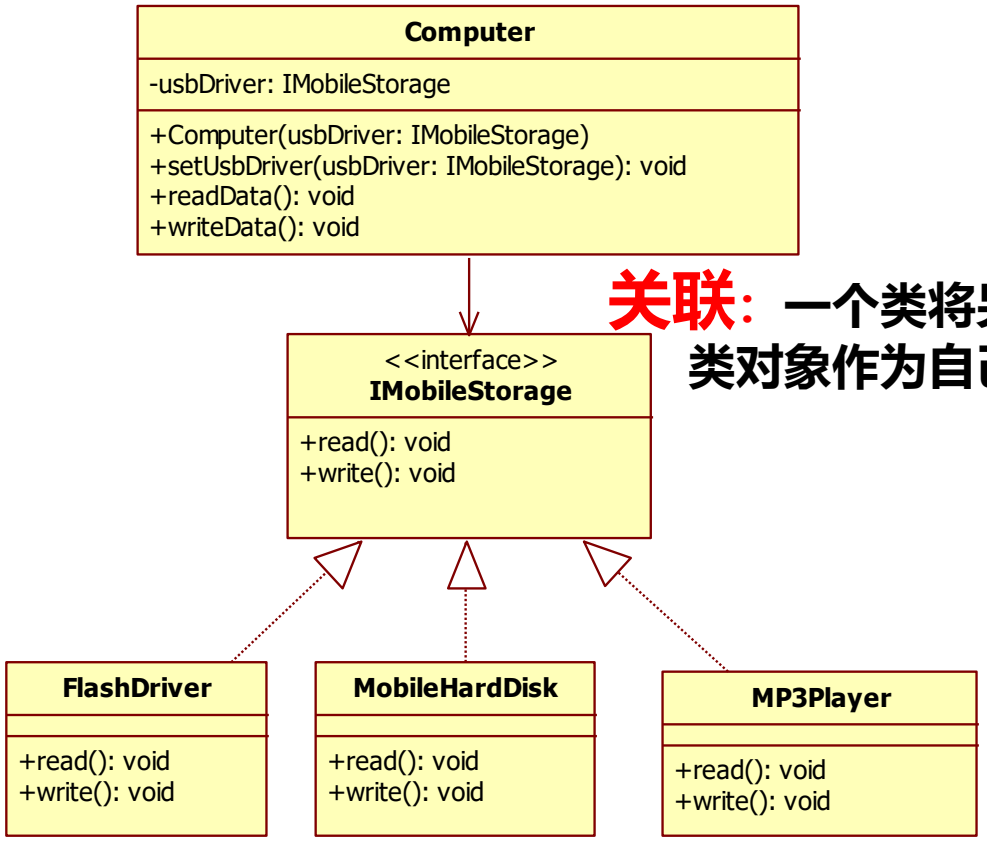


组合：整体与部分，二者不可分

依赖：某个类以局部变量或参数的形式出现在另一个类中



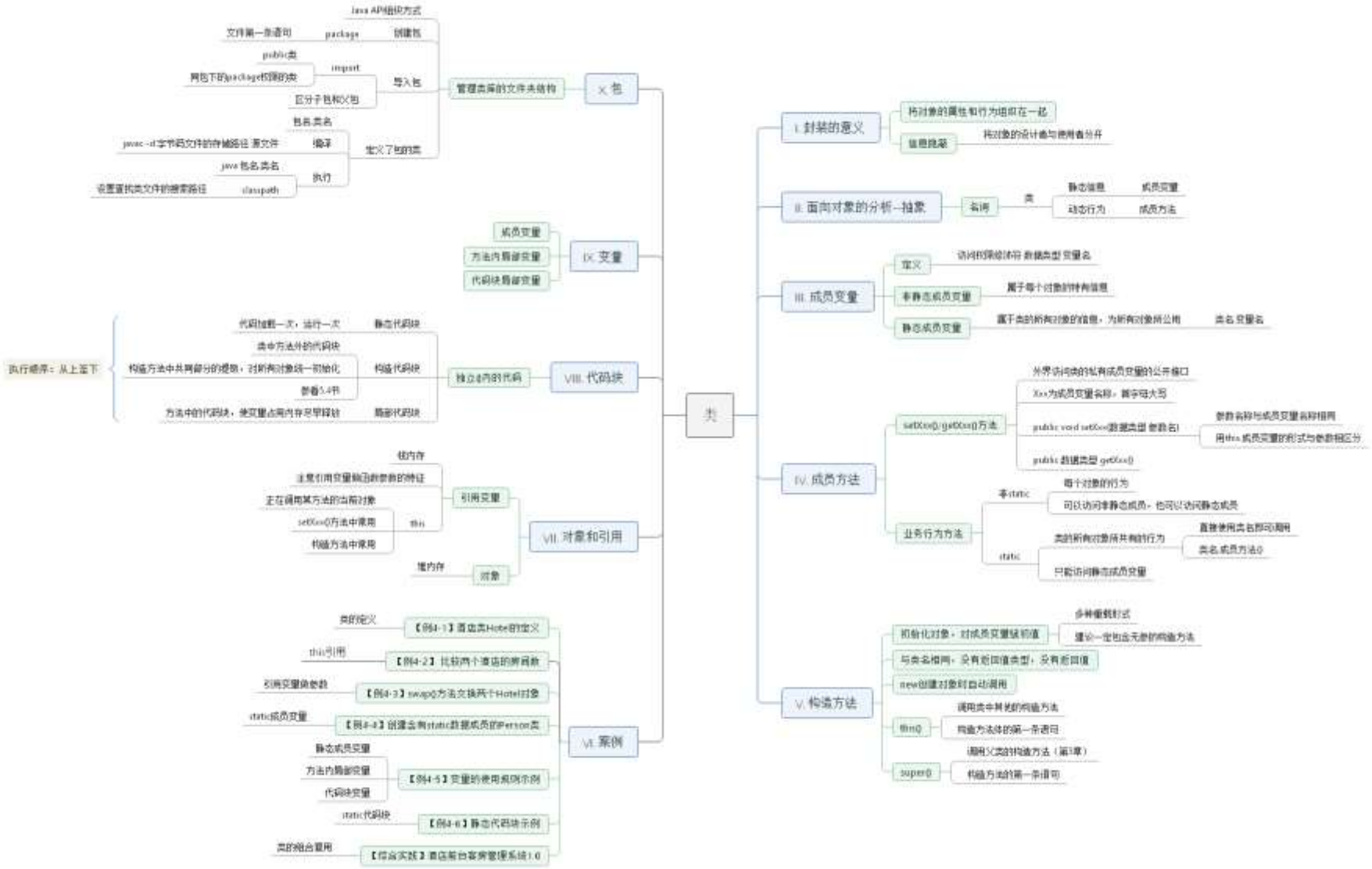
泛化：extends继承



关联：一个类将另一个类对象作为自己属性

实现：implement

本章思维导图


类
X. 包
管理类库的文件夹结构
Java API组织方式
创建包
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
设置查找类文件的搜索路径
classpath
IX. 变量
成员变量
方法内局部变量
代码块局部变量
VIII. 代码块
独立结构的代码
静态代码块
代码随着一次，运行一次
类中方法外的代码块
构造代码块
构造方法中共同部分的提取，对所有对象统一初始化
普通代码块
方法中的代码块，使变量占用内存尽早释放
局部代码块
执行顺序：从上至下
VII. 对象和引用
引用变量
栈内存
主要引用变量指向数据的特征
正在调用某方法的当前对象
this
setXxx()方法中常用
构造方法中常用
对象
堆内存
VI. 案例
类的定义
【例4-1】酒店类Hotel的定义
this引用
【例4-2】比较两个酒店的房间数
引用变量做参数
【例4-3】swap()方法交换两个Hotel对象
static成员变量
【例4-4】创建含有static数据成员的Person类
静态成员变量
方法内局部变量
代码块变量
【例4-5】变量的使用规则示例
static代码块
【例4-6】静态代码块示例
类的组合复用
【综合实践】酒店前台客房管理系统1.0
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
动态行为
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3章）
构造方法的第一条语句