普通高等教育“十三五”规划教材（软件工程专业）

软件设计模式实用教程

Software

主　编　车战斌
副主编　李勇军　高　亮

◎ 选取熟悉的业务场景完成每个模式的引题。
◎ 整本教材以实际开发应用贯穿关键知识点。
◎ 具有大量多种形式的课后习题。

中国水利水电出版社
www.waterpub.com.cn

普通高等教育“十三五”规划教材（软件工程专业）

软件设计模式实用教程

主　编　车战斌

副主编　李勇军　高　亮

中国水利水电出版社
www.waterpub.com.cn

·北京·

内 容 提 要

软件设计模式是软件工程前人经验的积累与总结，它为构建易维护和便复用的软件而诞生。本书结合大量的应用实例分析和讲解每一个常用的设计模式，贴近生活，力求通俗易懂，并且在真实项目实例的引导下学会合理运用设计模式。

本书分为3个部分，共6章内容：第1部分（第1章、第2章）为基础知识，包括UML类图讲解和设计原则等；第2部分（第3章、第4章、第5章）为设计模式讲解，包括6种常用的创建型设计模式、7种常用的结构型设计模式和10种常用的行为型设计模式；第3部分（第6章）为综合案例，使用多种模式混合解决实际应用问题。

本书适合作为高等学校计算机专业的软件开发课程教材，也可作为一线开发人员、高等院校计算机及软件等相关专业师生、IT培训机构讲师和学员、业余软件开发人员、设计模式研究人员以及爱好者的参考用书。

图书在版编目（CIP）数据

软件设计模式实用教程 / 车战斌主编. -- 北京 : 中国水利水电出版社, 2019.3
普通高等教育“十三五”规划教材. 软件工程专业
ISBN 978-7-5170-7230-0

Ⅰ. ①软… Ⅱ. ①车… Ⅲ. ①软件设计－高等学校－教材 Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2018)第273708号

策划编辑：石永峰　　责任编辑：张玉玲　　封面设计：李　佳

书　名	普通高等教育“十三五”规划教材（软件工程专业） **软件设计模式实用教程** RUANJIAN SHEJI MOSHI SHIYONG JIAOCHENG
作　者	主　编　车战斌 副主编　李勇军　高　亮
出版发行	中国水利水电出版社 （北京市海淀区玉渊潭南路1号D座　100038） 网址：www.waterpub.com.cn E-mail：mchannel@263.net（万水） sales@waterpub.com.cn 电话：（010）68367658（营销中心）、82562819（万水）
经　售	全国各地新华书店和相关出版物销售网点
排　版	北京万水电子信息有限公司
印　刷	三河市鑫金马印装有限公司
规　格	184mm×260mm　16开本　17印张　422千字
版　次	2019年3月第1版　2019年3月第1次印刷
印　数	0001—3000册
定　价	48.00元

前　言

什么是软件设计？如何进行软件设计？依据设计类图如何写代码？怎么使用设计模式？……

这是很多开发人员或者设计人员曾有过的感慨，尤其是初级设计人员或初识设计模式人员。

目前市面也有不少关于设计模式的书籍，有的主要是针对重点院校，完全是理论讲解并且针对每个模式讲解的篇幅相对偏少；有些教材易懂，作为入门教材比较好，但多数是翻译版，不能完全忠实于原文，并且文中详细讲解的模式数量相对少，没有针对性的课后习题；还有一些书籍，作为入门参考教材挺好，但其中的引题基本上都是生活中的例子，这样不符合软件设计思维，并且也没有针对性练习题。上述教材对于以培养应用型软件工程人才为目标的高等院校，不能很好地满足课程目标。为了解决只会编写代码，而不知道规范且想快速上手设计的初学者来说，本书可以为你答疑解惑。

本书以随手拈来的生活实例为最好的设计（Java 代码引题），结合项目实例讲解设计模式，讲解如何通过模式来解决上述生活问题，让读者能够快速提升自己的开发和设计能力，真正地理解和掌握每一个设计模式。

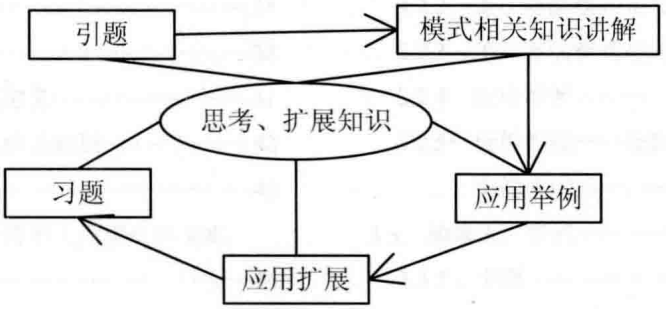
本书的组织

本书分为 3 个部分，主要讲授面向对象设计中使用的 UML 相关知识及设计原则，随后对设计模式进行总体的介绍，然后从创建型、结构型、行为型三种分类下常用的设计模式进行深入浅出地讲解，最后以 2 个综合案例讲解混合模式的使用。

第 1 部分是基础知识，包括第 1 章、第 2 章，由车战斌、李勇军执笔。该部分主要是进行 UML 中常用类图及设计原则的详细讲解。

第 2 部分是设计模式，包括第 3 章、第 4 章、第 5 章，其中第 3 章创建型模式由高亮执笔，第 4 章结构型模式由李勇军执笔，第 5 章行为型模式，由余雨萍、郭丽执笔。该部分主要是讲解常用的设计模式。

第 2 部分对于每个模式的讲解，力求通俗易懂，真实场景应用，每个模式讲解的基本结构如下：



第 3 部分是综合案例（第 6 章），由高亮、郭丽执笔。该部分主要是讲解使用多种模式来解决问题的方法。

本书特点

（1）选取熟悉的业务场景完成每个模式的引题；
（2）整本教材以实际开发应用贯穿关键知识点；
（3）具有大量多种形式的课后习题。

本书风格

语句及案例接“地气”，通俗易懂，以常见案例的 Java 代码作为引题，随后介绍相关模式的知识，最后以类似案例进行分析讲解，并配以大量多种形式的习题。

本书由车战斌教授主编并统稿，李勇军、高亮任副主编，参与编写的还有余雨萍、郭丽。在本书编写过程中，本书编者进行了多次规划、组稿和方案讨论工作，并提出了许多建设性意见，在此一并表示感谢。

由于编者水平有限，书中错误或不妥之处在所难免，恳请广大读者批评指正，欢迎通过电子邮箱 yongjunli@zut.edu.cn 来信告知。

编　者

2019 年 1 月

目　　录

前言

第一部分　基础知识

第 1 章　UML 类图讲解 …… 1
1.1　UML 中类的表示法 …… 1
1.2　UML 中类之间的关系 …… 4
1.3　如何阅读类图 …… 9
1.4　本章小结 …… 12
第 2 章　设计原则介绍 …… 13
2.1　单一职责原则 …… 13
2.1.1　引题 …… 13
2.1.2　相关知识 …… 14
2.1.3　应用 …… 15
2.2　里氏替换原则 …… 16
2.2.1　引题 …… 16
2.2.2　相关知识 …… 17
2.2.3　应用 …… 17
2.3　开-闭原则 …… 20
2.3.1　引题 …… 20
2.3.2　相关知识 …… 23
2.3.3　应用 …… 24
2.4　依赖倒置原则 …… 26
2.4.1　引题 …… 26
2.4.2　相关知识 …… 27
2.4.3　应用 …… 27
2.5　接口隔离原则 …… 28
2.5.1　引题 …… 28
2.5.2　相关知识 …… 30
2.5.3　应用 …… 30
2.6　迪米特法则 …… 31
2.6.1　引题 …… 31
2.6.2　相关知识 …… 32
2.6.3　应用 …… 33
2.7　合成/聚合复用原则 …… 34
2.7.1　引题 …… 34
2.7.2　相关知识 …… 35
2.7.3　应用 …… 36
2.8　本章小结 …… 37

第二部分　设计模式

第 3 章　创建型模式 …… 38
3.1　简单工厂模式 …… 38
3.1.1　引题 …… 38
3.1.2　简单工厂模式定义 …… 41
3.1.3　简单工厂模式相关知识 …… 42
3.1.4　应用举例 …… 43
3.1.5　应用扩展——简单工厂模式在 JDK 中的应用 …… 44
3.2　工厂方法模式 …… 44
3.2.1　引题 …… 44
3.2.2　工厂方法模式定义 …… 45
3.2.3　工厂方法模式相关知识 …… 46
3.2.4　应用举例 …… 47
3.2.5　应用扩展——反射在工厂方法模式中的应用 …… 49
3.3　抽象工厂模式 …… 51
3.3.1　引题 …… 51
3.3.2　抽象工厂模式定义 …… 51

3.3.3 抽象工厂模式相关知识……54
3.3.4 应用举例……54
3.3.5 应用扩展——抽象工厂模式在 JDK 中的应用……57
3.4 单例模式……58
3.4.1 引题……58
3.4.2 单例模式定义……59
3.4.3 单例模式相关知识……60
3.4.4 应用举例……60
3.4.5 应用扩展——单例模式在多线程中的应用……61
3.5 原型模式……63
3.5.1 引题……63
3.5.2 原型模式定义……65
3.5.3 原型模式相关知识……66
3.5.4 应用举例……67
3.5.5 应用扩展——浅复制与深复制……69
3.6 建造者模式……74
3.6.1 引题……74
3.6.2 建造者模式定义……75
3.6.3 建造者模式相关知识……77
3.6.4 应用举例……78
3.6.5 应用扩展——建造者模式在 Java API 中的应用……80
3.7 本章小结……80
3.8 习题……80
第 4 章 结构型模式……83
4.1 装饰者模式……83
4.1.1 引题……83
4.1.2 装饰者模式定义……86
4.1.3 装饰者模式相关知识……88
4.1.4 应用举例……88
4.1.5 应用扩展——装饰者模式在 Java API 中的应用……96
4.2 代理模式……97
4.2.1 引题……97
4.2.2 代理模式定义……100
4.2.3 代理模式相关知识……101
4.2.4 应用举例……102
4.2.5 应用扩展——代理模式在 Java API 中的应用……105
4.3 适配器模式……106
4.3.1 引题……106
4.3.2 适配器模式定义……106
4.3.3 适配器模式相关知识……108
4.3.4 应用举例……108
4.3.5 应用扩展——适配器模式在 Java API 中的应用……114
4.4 外观模式……114
4.4.1 引题……114
4.4.2 外观模式定义……115
4.4.3 外观模式相关知识……117
4.4.4 应用举例……117
4.4.5 应用扩展——外观模式在 Java API 中的应用……122
4.5 组合模式……122
4.5.1 引题……122
4.5.2 组合模式定义……125
4.5.3 组合模式相关知识……127
4.5.4 应用举例……127
4.5.5 应用扩展——组合模式在 Java API 中的应用……134
4.6 桥接模式……135
4.6.1 引题……135
4.6.2 桥接模式定义……137
4.6.3 桥接模式相关知识……138
4.6.4 应用举例……139
4.6.5 应用扩展——桥接模式在 Java API 中的应用……143
4.7 享元模式……143
4.7.1 引题……143
4.7.2 享元模式定义……144
4.7.3 享元模式相关知识……146
4.7.4 应用举例……147

4.7.5 应用扩展——享元模式在Java API 中的应用 …… 153
4.8 本章小结 …… 153
4.9 习题 …… 153
第 5 章 行为型模式 …… 160
5.1 观察者模式 …… 160
5.1.1 引题 …… 160
5.1.2 观察者模式定义 …… 163
5.1.3 观察者模式相关知识 …… 165
5.1.4 应用举例 …… 166
5.1.5 应用扩展——观察者模式在Java APJ 中的应用 …… 167
5.2 迭代器模式 …… 169
5.2.1 引题 …… 169
5.2.2 迭代器模式定义 …… 173
5.2.3 迭代器模式相关知识 …… 175
5.2.4 应用举例 …… 176
5.2.5 应用扩展——迭代器模式在Java JDK 中的应用 …… 179
5.3 策略模式 …… 181
5.3.1 引题 …… 181
5.3.2 策略模式的定义 …… 186
5.3.3 策略模式相关知识 …… 187
5.3.4 应用举例 …… 188
5.3.5 应用扩展——策略模式在 JDK中的应用 …… 190
5.4 模板方法模式 …… 191
5.4.1 引题 …… 191
5.4.2 模板方法模式定义 …… 191
5.4.3 模板方法模式相关知识 …… 193
5.4.4 应用举例 …… 193
5.4.5 应用扩展——模板方法模式在Java API 中的应用 …… 195
5.5 命令模式 …… 195
5.5.1 引题 …… 195
5.5.2 命令模式定义 …… 196
5.5.3 命令模式相关知识 …… 198
5.5.4 应用举例 …… 198
5.5.5 应用扩展——命令模式在 Java API中的应用 …… 201
5.6 状态模式 …… 201
5.6.1 引题 …… 201
5.6.2 状态模式定义 …… 201
5.6.3 状态模式相关知识 …… 203
5.6.4 应用举例 …… 203
5.6.5 应用扩展——状态模式在 Java API中的应用 …… 207
5.7 责任链模式 …… 207
5.7.1 引题 …… 207
5.7.2 责任链模式定义 …… 208
5.7.3 责任链模式相关知识 …… 209
5.7.4 应用举例 …… 210
5.7.5 应用扩展——责任链模式在 Java API中的应用 …… 212
5.8 解释器模式 …… 212
5.8.1 引题 …… 212
5.8.2 解释器模式定义 …… 212
5.8.3 解释器模式相关知识 …… 214
5.8.4 应用举例 …… 214
5.9 备忘录模式 …… 217
5.9.1 引题 …… 217
5.9.2 备忘录模式定义 …… 217
5.9.3 备忘录模式相关知识 …… 219
5.9.4 应用举例 …… 220
5.9.5 应用扩展 …… 222
5.10 中介者模式 …… 222
5.10.1 引题 …… 222
5.10.2 中介者模式定义 …… 223
5.10.3 中介者模式相关知识 …… 225
5.10.4 应用举例 …… 225
5.11 本章小结 …… 229
5.12 习题 …… 230

第三部分　综合案例

第 6 章　案例——学生信息管理系统 ····· 236

6.1　学生信息管理系统——抽象工厂模式与单例模式结合 ····· 236

6.1.1　系统需求 ····· 236

6.1.2　模式应用分析 ····· 236

6.1.3　类设计 ····· 237

6.1.4　详细编码 ····· 238

6.2　数据库连接池——动态代理模式与单例模式相结合 ····· 254

6.2.1　需求分析 ····· 254

6.2.2　动态代理模式与单例模式实现数据库连接池 ····· 255

6.2.3　数据库连接池的使用 ····· 262

6.3　小结 ····· 263

参考文献 ····· 264

第一部分　基础知识

第 1 章　UML 类图讲解

软件设计模式，又称设计模式，是一套被反复使用、多数人知晓、经过分类编目，以及具有代码设计经验的总结。但由于模式本身具有抽象性，为了保证模式描述的统一性，本书采用 UML（Unified Modeling Language，统一建模语言）的方式进行描述。

UML 现已纳入了 OMG（Object Management Group，对象管理组织）标准，成为业务、应用和系统架构的标准可视化建模语言。而 UML 中使用较多的图是类图。类图是使用最广泛的一种模型，用来表述系统中各个类的类型以及其间存在的各种静态关系。

设计时，类图用来记录类的结构，这些类构成了系统的架构。那么如何有效地理解和掌握类图，从而更好地进行系统的设计？本章将从类的表示法、类之间的基本关系和类图的阅读上进行介绍，以便读者在学习设计模式之前对类有一个整体的认识。

1.1　UML 中类的表示法

类是对一组具有相同属性、操作、关系和语义的描述。关系是类之间的，语义是蕴藏的，因此对于一个类而言，其关键特性是属性（成员变量）和操作（成员方法）。图 1-1 是 UML 中类图的表示法，从图中可看出类是用一个矩形表示的。它包含三个分栏，从上至下每个分栏分别写入类的名称、属性和操作。

类名
-属性：属性类型
+方法()：方法返回值类型

图 1-1　UML 中类的表示法

1. *名称*

名称是一个字符串，用于区别于其他类，表示方法有两种：

- 简单名：仅是一个单独的名称，如订单 Order 等。
- 全名：也称为路径名，就是在类名前加上包的名称，如 java::awt::Rectangel、businessRule::Order 等。

对于类名称的命名规则，UML 中并未明确定义，只要是由字符、数字、下划线组成的唯一的字符串即可，但在实际应用中，有一个普遍采用的命名规则：

大驼峰法（印帕斯卡命名法）：使用大写字母开头、混合大小写，每个单词以大写开始。避免使用特殊符号，尽可能避免使用缩写。

2. 属性

属性是一个名词，描述类实例中包含的特征信息，同时表明了对象的唯一性。创建对象时，属性可以有初始值。在面向对象编程中，它一般实现为类的成员变量。

从图 1-1 中可看出，属性前面有一个修饰，用来表示它的可见性（详见可见性），一般来说，属性的可见性均为私有 private，这样才符合面向对象的“封装”思想。

属性名的命名规则虽不是硬性规定，但通常都习惯于采用小驼峰法标识，也即把第一个单词的首字母小写，为进行区分，通常属性名的第一个字母是小写的。

3. 操作

操作是类提供的服务，是访问本对象属性或关系的一种途径，也是影响其他对象属性或关系的唯一途径。通俗点说，操作就是定义了对象所能做的事情。在面向对象编程语言中，它通常以成员方法的形式实现。

从图 1-1 中可看出，操作前面也有一个修饰，用来表示它的可见性，为向其他类提供服务，操作通常应声明为公有 public。并且操作在表示时，可只写出操作名，也可将其所需的参数写出来，即写出成员方法的完整签名。

操作名的命名规则也未硬性规定，通常习惯采用和属性名相同的命名规则。

4. 可见性

类的属性和操作都有相似的可见性定义，各种编程语言对可见性的处理并不完全相同。UML 中将可见性归纳为四类：

- 公有：除了类本身以外，属性和操作对其他类也是可见的。属性的公有可见性应尽量少用，公有意味着将类的属性暴露给外部，这与面向对象的封装原则是矛盾的。暴露给外部的内容越多，对象越容易受影响，越容易形成高耦合度。
- 保护：属性和操作只对类本身、它的子类或友元（取决于具体编程语言）是可视的。保护可见性用于保护属性和操作使其不被超出上述范围的外部类使用，防止行为的耦合和封装变得松散。
- 私有：属性和操作只对类本身和类的友元（取决于具体编程语言）是可见的。私有可见性可用在不希望子类继承属性和操作的情况下。它提供了从超类对子类去耦的方法，并且减少了删除或排除未使用继承操作的需要。
- 实施：属性和操作只在类本身的内部是可视的（取决于具体编程语言）。实施可见性最具有限制性，当只有类本身才可使用属性和操作时，才使用这种可见性，它是私有可见性的变体，在有些 UML 建模工具中并不支持。

【示例】

下面构建一个学生类。不管是在学生成绩管理系统、图书管理系统，亦或是教务管理系统中都不乏有学生出现的地方。假定现在需要构造一个学生类，此类具有学号、姓名以及一个静态成员学生数量等属性，要求能够通过构造方法对学生信息进行初始化和学生个数累加，分别设置和获取学生各个属性，并使用一个方法输出学生的所有属性值。

构造学生类时，由于要求能够对学生各个属性进行设置和获取，因此学生的学号、姓名等属性均应设为私有。为方便程序人员进行编码工作，通常使用相应语言的标识符来标记，本片使用 Java 语言，现设计的学生类图如图 1-2 所示。

通常对于属性（成员变量）进行获取和设置的方法不显示在类图中，因此学生类图也可以表示为如图 1-3 所示。

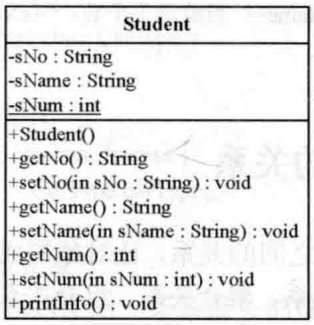


图 1-2　学生类图

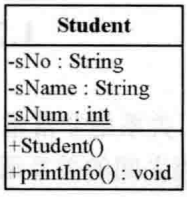


图 1-3　学生类图

Student 类所对应的代码如下：

```
public class Student {
    private String sNo;
    private String sName;
    private static int sNum;
    public Student(){
        sNo = "20180767102";
        sName = "杨帅";
        sNum++;
    }
    /*获取学生学号*/
    public String getNo() {
        return sNo;
    }
    /*设置学生学号*/
    public void setNo(String sNo) {
        this.sNo = sNo;
    }
    /*获取学生姓名*/
    public String getName() {
        return sName;
    }
    /*设置学生姓名*/
    public void setName(String sName) {
        this.sName = sName;
    }
    /*获取学生当前个数*/
    public int getNum() {
        return sNum;
    }
    /*设置学生当前个数*/
    public void setNum(int sNum) {
        Student.sNum = sNum;
    }
```

```
    /*打印学生信息*/
    public void printInfo(){
        //方法体可依据实际情况自行来编写
        System.out.println("学号："+sNo+" 姓名："+sName+" 当前学生个数："+sNum);
    }
}
```

1.2 UML 中类之间的关系

在 UML 中，关系是非常重要的语义。它抽象出对象之间的关系，让对象构成某个特定的结构。类图中，类之间的关系通常包括关联关系、依赖关系、聚合关系、组合关系、泛化关系和实现关系。需要注意的是：不同的 UML 建模工具中关系的表示法与 UML 标准存在差异，本教材中统一使用 Microsoft Visio 进行相关图例的绘制。以下将对类之间的关系进行介绍。

1. 关联关系

UML 中关联关系使用一条直线表示，如图 1-4（a）所示。它描述不同类的对象之间的结构关系，它在一段时间内将多个类的实例连接在一起。简单地说，关联关系描述了某个对象在一段时间内一直“知道”另一个对象的存在。它体现的是两个类或类与接口之间语义级别的一种强依赖关系，比如我和我的好朋友，这种关系比依赖更强、不存在依赖关系的偶然性、关系也不是临时的，一般是长期的，而且双方的关系一般是平等的。

标准的 UML 对于关联关系的表示法只使用一条直线来表示，它表明两个类对象可以互相“知道”，但有时为表明 A“知道”B，但 B 并不“知道”A，即关系是单向的，可使用一条带箭头的直线来表示，如图 1-4（b）所示。当然关联关系还存在自关联和多维关联，本书中暂不涉及。

（a）　　（b）

图 1-4　关联关系表示法

关联关系在面向对象语言中实现时，表示类 A 引用了一个类型为关系类 B 的全局变量，即类 B 对象常常作为属性在类 A 中进行引用。

【示例】

服装生产中针对每一款号的衣服均会有多个工序，而在每道工序进行加工时，都会在相应的车种上进行加工，这也就是说工序加工时是需要知道车种信息，因此工序 Process 和车种 CarType 之间的关系则是一种关联关系并且是一种单向关系，如图 1-5 所示。

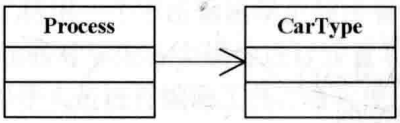


图 1-5　工序和车种之间的关系

在程序世界中，工序 Process 使用车种 CarType 只需要将 CarType 作为 Process 的属性即可。参考代码如下：

CarType 类：

```
public class CarType {
}
```

Process 类：

```
public class Process {
    CarType carType;
}
```

2. 依赖关系

依赖关系是使用一条带箭头的虚线表示，如图 1-6 所示。此关系可描述为一个对象的修改会导致另一个对象的修改。与关联不同的是，依赖关系除了“知道”其对象的存在，还会“使用”其对象的属性或方法，即依赖是一种特殊的关联关系。依赖关系具有偶然性、临时性，但类 B 的变化会影响到类 A。比如当要写字时，需要借助笔，此时写字者和笔之间的关系就是依赖，笔的变化会影响写字者的写字操作。

同样，依赖也有单向依赖和双向依赖之分，但是依赖关系却不像关联关系那样有带箭头和不带箭头之分，均使用带箭头的虚线。因为在面向对象设计中，双向依赖是一种非常不好的结构，在设计时应当保持单向依赖，杜绝双向依赖关系的产生。

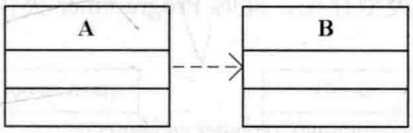


图 1-6　依赖关系表示法

依赖关系在面向对象编程语言中进行实现时，类 B 对象常常在类 A 的某一个操作中使用。

【示例】

程序员使用电脑是常事，也就是说作为程序员（Programmer）在编程时需要使用电脑（Computer），这时程序员就依赖于电脑。程序员使用电脑的类图如图 1-7 所示。

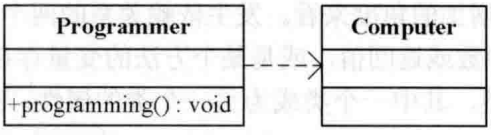


图 1-7　程序员使用电脑类图

在程序世界中，Programmer 类如何使用 Computer 类呢？主要是有 3 种形式：

第 1 种：Computer 类是公有的，Programmer 类直接调用它。这种情况下就如同单位公共电脑，程序员无需做申请，都可以使用。代码如下：

Computer 类：

```
public class Computer{

}
```

Programmer 类：

```
class Programmer{
        public void programming(){
         //此处不用声明变量，直接调用 Computer 类的方法
        }
}
```

第 2 种：Computer 类是 Programmer 类中某个方法的局部变量。代码如下：

Computer 类：

```
class Computer {
}
```

Programmer 类：

```
class Programmer {
    public void programming(){
        Computer computer = new Computer();
    }
}
```

Programmer 有一个 programming 方法，Computer 类作为该方法的变量来使用。即 Computer 类的持有者是 programming 方法，只有该方法被调用时才被实例化。

第 3 种：Computer 类保持不变，只是 Programmer 类使用 Computer 类的方式进行修改。例如，作为形式参数或返回值类型存在。此时 Programmer 类代码如下：

Programmer 类：

```
class Programmer {
    public Computer programming(Computer computer){
        return null;
    }
}
```

这时 Computer 类被 Programmer 类的一个方法所持有，Computer 类对象生命周期会随着方法执行的结束而结束。

在依赖关系中，使用较多的是第 3 种方式。

从上述讲解来看，关联和依赖的区别主要表现在 2 个方面：

（1）从类的属性是否增加的角度来看。发生依赖关系的两个类不会增加属性。其中一个类作为另一个类的方法的参数或返回值，或是某个方法的变量存在。

发生关联关系的两个类，其中一个类成为另一个类的属性，而属性是一种更为紧密的耦合，成为长久的持有关系。

（2）从关系的生命周期来看。依赖关系是仅当类的方法被调用时而存在，随着方法的结束而结束。

关联关系是当类实例化的时候即产生，当类销毁时结束。相比依赖来言，关联关系的生存期更长。

3. 聚合关系

聚合关系用一条带空心菱形箭头的直线表示，如图 1-8（a）所示，它表明 A 聚合到 B 上，或者说 B 由 A 聚合而成。聚合关系主要表示实体对象之间的关系，表达整体由部分构成的语

义，即 has-a 关系。如图 1-8（b）所示一个部门由许多员工构成。

（a）　　（b）

图 1-8　聚合关系表示法

聚合关系中的整体和部分不是强依赖，即使整体不存在，部分仍然存在，例如部门撤销以后，人员依然存在。不同的 UML 建模工具中对于聚合的表示符号并不完全一致，如 Visio 中可以为聚合关系指定有向性，即是双向关系还是单向关系。

聚合关系在面向对象编程语言中进行实现时，和关联关系是一致的，往往是类 A 以集合的形式在类 B 中作为属性进行引用。

【示例】

这里引用程杰《大话设计模式》里举的大雁的例子。

大雁喜欢热闹害怕孤独，所以它们一直过着群居的生活，这样就有了雁群，每一只大雁都有自己的雁群，每个雁群都有好多大雁，大雁 Goose 与雁群 GooseGroup 之间的关系就是聚合关系，如图 1-9 所示。

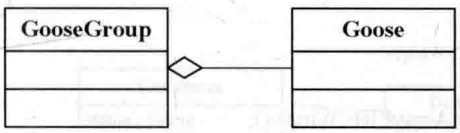


图 1-9　雁群与大雁的聚合关系

雁群和大雁关系的代码如下：

Goose 类：

```
public class Goose {

}
```

GooseGroup 类：

```
public class GooseGroup {
    public List<Goose> gooses;
    public GooseGroup(List<Goose> gooses){
        this.gooses = gooses;      //仅赋值，不实例化
    }
}
```

4. 组合关系

组合关系使用一条带实心菱形箭头的直线表示，如图 1-10（a）所示，它表明 A 组合成 B，或者说 B 由 A 组合而成。组合关系用于表示实体对象关系，表达整体拥有部分的语义。如图 1-10（b）所示，1 个母公司可拥有多个子公司。

（a） （b）

图 1-10 组合关系表示法

组合关系是一种强依赖的特殊聚合关系，如果整体不存在，则部分也将消亡。如母公司解体了，子公司也将不复存在。

组合关系在面向对象编程语言中进行实现时，和关联关系一致，往往是类 A 以集合的形式在类 B 构造方法中进行实例化。

【示例】

接着聚合中的大雁示例来举例，每只大雁都有两只翅膀，大雁 Goose 与雁翅 Wing 的关系则是组合关系，如图 1-11 所示。

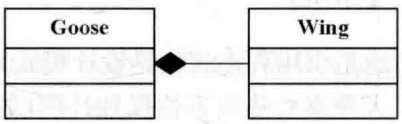


图 1-11 大雁与雁翅的组合关系

大雁与雁翅之间关系的代码如下：

Wing 类：

```
public class Wing {

}
```

Goose 类：

```
public class Goose {
    public List<Wing> wings;
    public Goose(){
        wings = new ArrayList<Wing>();
    }
}
```

通过例子可以看出，聚合关系明显没有组合紧密，大雁不会因为它们的群主将雁群解散而无法生存，而雁翅则无法脱离大雁而单独生存，这也说明组合关系的类具有相同的生命周期。

聚合和组合的关系区别主要体现在 2 个方面：

（1）构造方法不同。聚合类的构造方法中包含另一个类的实例作为参数，因为构造方法中传递另一个类的实例，因此示例中的大雁类可以脱离雁群独立存在；组合类的构造方法包含另一个类的实例化，因为在构造方法中进行实例化，因此两者紧密耦合在一起，同生同死，翅膀类不能脱离大雁类存在。

（2）信息的封装性不同。在聚合关系中，使用者可同时了解到雁群类和大雁类，因为他们是独立的；而在组合关系中，使用者只认识到大雁类，根本不知道翅膀类的存在，因为翅膀类封装在大雁类中。

5. 泛化关系

泛化关系使用一条带空心箭头的直线表示，如图 1-12 所示，它表明 A 继承了 B。

泛化关系说明两个对象之间的继承关系，是从后代类到其祖先类的关系。

泛化关系在面向对象编程语言中进行实现时，主要是在类声明时使用，如在 Java 语言中实现图 1-12 泛化关系，则类 A 的声明应为：class A extends B。