软件工程课后习题答案

（适用于《软件工程》朴勇，周勇编著，2019 年版）

目录

第 1 章　软件工程概述

1. 软件工程主要包括哪些内容？

软件工程是从技术和管理两个方面开发和维护计算机软件的一门学科。IEEE 对软件工程的定义是：将系统化、规范化、可量化的工程原则和方法应用于软件的开发、运行和维护及对其中方法的理论研究，其主要目标是高效开发高质量的软件，降低开发成本。

软件工程知识体系包含两个部分：开发过程和支持过程；10 个主要的知识域，分别是：软件需求、软件设计、软件构造、软件测试、软件维护、软件配置管理、软件工程管理、软件工程过程、软件工程工具与方法、软件质量。

2. 面向对象分析方法优于传统方法的根本原因是什么？可否借助图 1.4 或其他实例给出自己的理解？

面向对象方法是一种动态的思想，其出发点和基本原则是尽可能模拟人类习惯的思维方式，将现实世界中的实体抽象为对象（Object），对象中同时封装了实体的静态属性和动态方法。面向对象分析设计的方式使得业务领域中实体及实体之间的关系与对象及其关系保持一致，做到了概念层与逻辑层的相互协调，更要强调的是各种逻辑关系在结构上的稳定性，通过稳定的结构来提高应对各种变化的能力。

技术上，对象融合了数据及在数据之上的操作，所有的对象按照类（Class）进行划分，类是对象的“抽象”；类与类之间可以构成“继承”的层次关系；对象之间的互相联系是通过消息机制实现的，确保了对信息的“封装”，使得对象之间更为独立。

同时，面向对象的分析过程既包含了由特殊到一般的归纳思维过程，也有由一般到特殊的演绎思维过程，而且对象是更为独立的实体，可以更好地进行“重用”。

3. UML 包含哪些重要的模型？它们在系统开发的解空间中作用如何？

UML2.0 具体包括以下模型：

1. 用例图：用于表示系统与使用者（或其他外部系统）之间的交互，有助于将需求映射到系统；
2. 活动图：用于表示系统中顺序和并行的活动；
3. 类图：用于表示类、接口及其之间的关系；
4. 对象图：用于表示类图中定义的对象的实例，其配置是对系统的模拟；
5. 顺序图：用于表示重要对象之间的互动顺序；
6. 通信图：用于表示对象交互的方法和需要支持交互的连接；
7. 时序图：用于表示重点对象之间的交互时间安排；
8. 交互概况图：用于将顺序图、通信图和时序图收集到一起，以捕捉系统中发生的重要交互情况；
9. 组成结构图：用于表示类或组件的内部，可以在特定的上下文中描述类间的关系；
10. 组件图：用于表示系统内的重要组件和彼此间交互所用的接口；
11. 包图：用于表示类与组件集合的分级组织；
12. 状态图：用于表示整个生命周期中对象的状态和可以改变状态的事件；
13. 部署图：用于表示系统最终怎样被部署到真实的世界中。

第 2 章　软件开发过程

1. 总结敏捷生命周期模型与传统瀑布模型主要的不同点及适用情况。

敏捷生命周期模型处理需求和技术变化主要通过增量和迭代过程。在每一次周期结束时，都交付用户一个可用的、可部署的系统，每次迭代周期尽可能短，以便能及时频繁地处理需求变化和用户反馈。

而传统瀑布模型中，用户只有在开发早期及开发结束后，才有机会接触系统，且由于文档驱动式的开发方式，模型缺少灵活性，变更很不容易，变更来的越迟，付出的代价也越大。

瀑布模型是一种计划驱动的模型，在对系统整体上的把控和协调开发过程中，上具有一定的优势，因此瀑布模型比较适合规模较大的系统开发或者分布式的开发模式。而敏捷生命周期模型更适合规模中小、需求变化频繁的系统开发，强调团队的作用，因此更适合集中式的开发模式。

2. 什么是用例？它与功能的含义有什么不同？举例说明。

用例（Use Case）是指用户通过系统完成的有价值的目标。用例不是一个具体的功能，一个用例是用户与系统的完整交互，可能涉及多个功能的组合，不同的用例可能会涉及相同的功能组合，但意义却不同。

例如，用户按下计算器上的减号键，这是一个功能；用户按下“3-2=”的序列并计算出结果，这是一个用例。

3. 简述 DevOps 的基本原理和任务。

DevOps 是一组过程、方法与系统的统称，用于促进开发、技术运营和质量保障部门之间的沟通、协作与整合。它的出现是由于软件行业日益清晰地认识到：为了按时交付软件产品和服务，开发和运维工作必须紧密合作。

第 3 章　需求分析

1. 在软件开发过程中，问题发现得越晚，修正起来越困难，付出的代价越高。分析原因，至少给出两个理由，并简短说明。

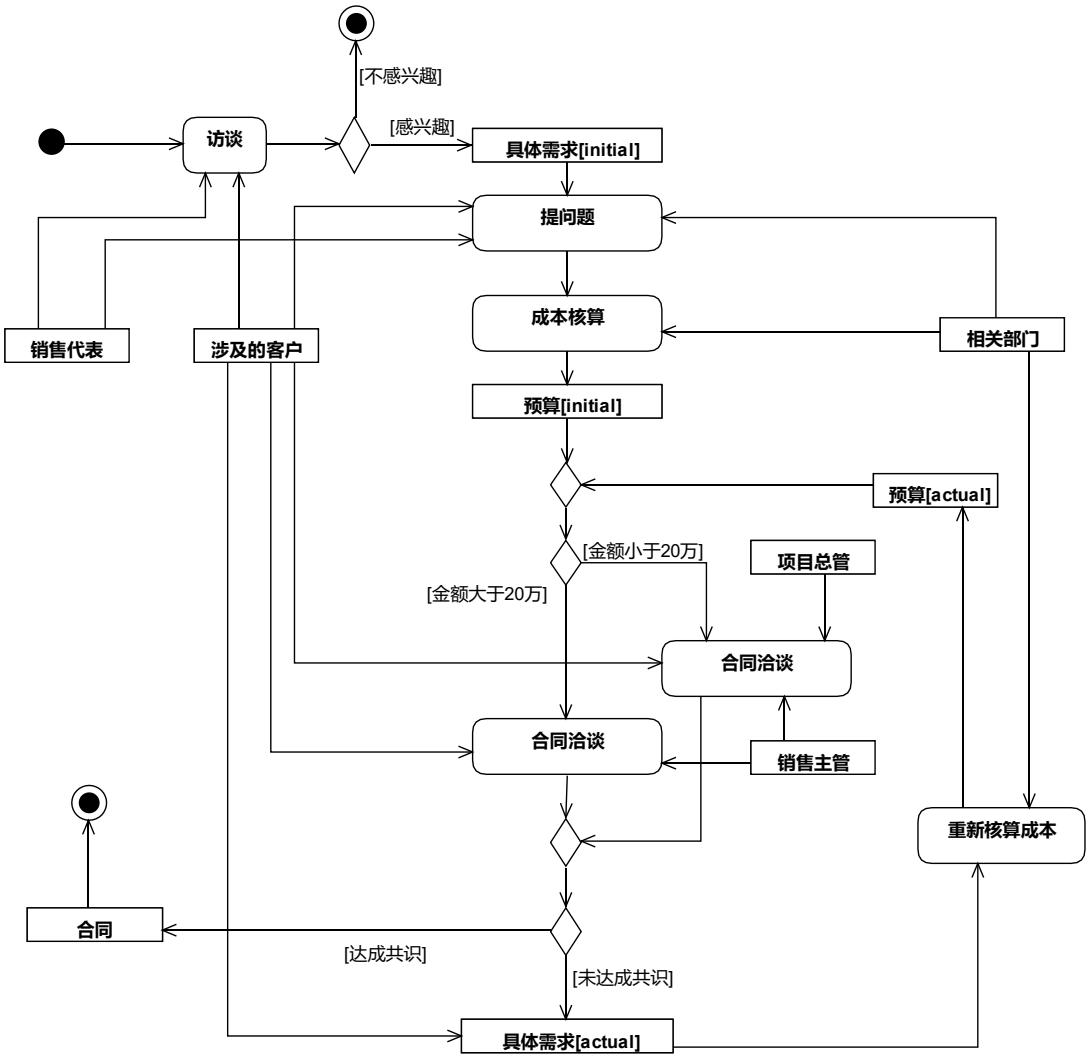
问题发现得越晚，所涉及的部分就越多，因为软件开发已经接近完成，问题可能会对软件开发产生全局性的影响，导致整个项目的失败；而前期发现问题则可以较快地修正，因为此时往往程序还没有开发完成，和问题相关的部分也相对较少，修改起来成本也更小。

2. 图 3.10 通过活动图描述了合同签订的流程，请对其进行扩展，满足以下要求。

（1）客户对销售的产品并不感兴趣。

（2）对于金额小于 20 万元的合同，在谈判的过程中需要部门经理的参与，考虑到项目管理的费用，由他们决定是否按照当前的合同金额签订合同。

（3）相关部门在成本核算前会提出一些问题，需要销售人员向客户询问，并澄清结果。



3. 给出以下需求描述的用例图。

（1）一个音像商店准备开发软件系统，用于向客户销售或者租借电影光盘。

（2）音像商店向多家订购商订购光盘，然后分类存储在系统中，订购了上千张光盘；还可以根据客户的请求向订购商订购光盘。

（3）所有的电影光盘用条码来管理，条码的号码是光盘的唯一标识。

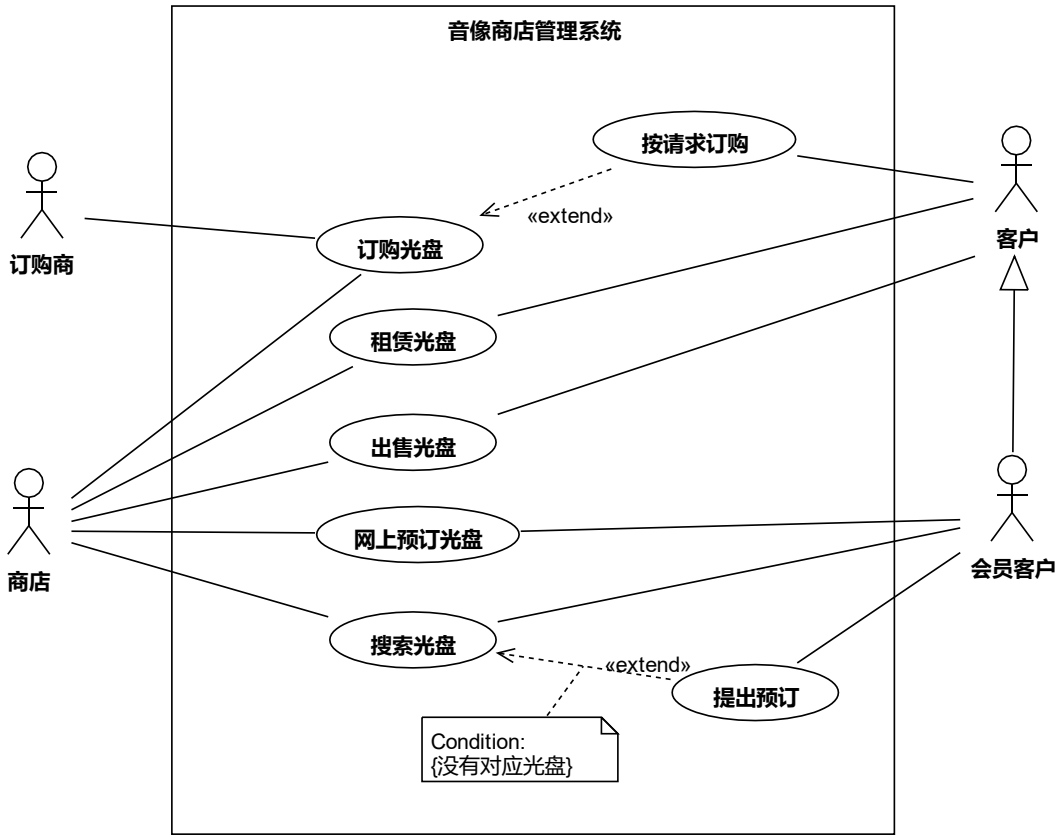
（4）音像商店可以向客户销售或租赁电影光盘。使用条码扫描来支持销售或租赁。

（5）音像商店建立会员制，会员客户购买电影光盘可以享受折扣。会员卡也使用条码来管理。

（6）会员可以通过网络预订电影光盘，并在指定的日期来取。

（7）会员可以利用灵活的搜索机制找到喜欢的电影，如果没有对应光盘，可以提出预订。

使用 UML 给出上述需求描述的用例图。要求绘制规范，尤其注意“角色—用例”和“用例—用例”之间的关系。



4. 根据某毕业设计选题系统的功能描述，使用 UML 建模技术，完成需求分析的用例图，包括系统的用例及其子用例（如果有，需要标记与主用例的关系）和角色（Actor）。

（1）教师信息维护：教务员录入老师的基本信息；教师信息包括教师 ID、教师姓名、教师职称、联系方式、邮箱地址等，可从 Excel 中导入；指导教师的联系方式在学生选题功后，才能公开给学生。

（2）学生信息维护：教务员录入和维护学生信息，学生信息包括学号（学生 ID）、学生姓名、班级。

（3）登录：学生、教师、教务员都需要输入 ID 和密码登录系统，使用权限范围内的用例，可以修改个人密码。

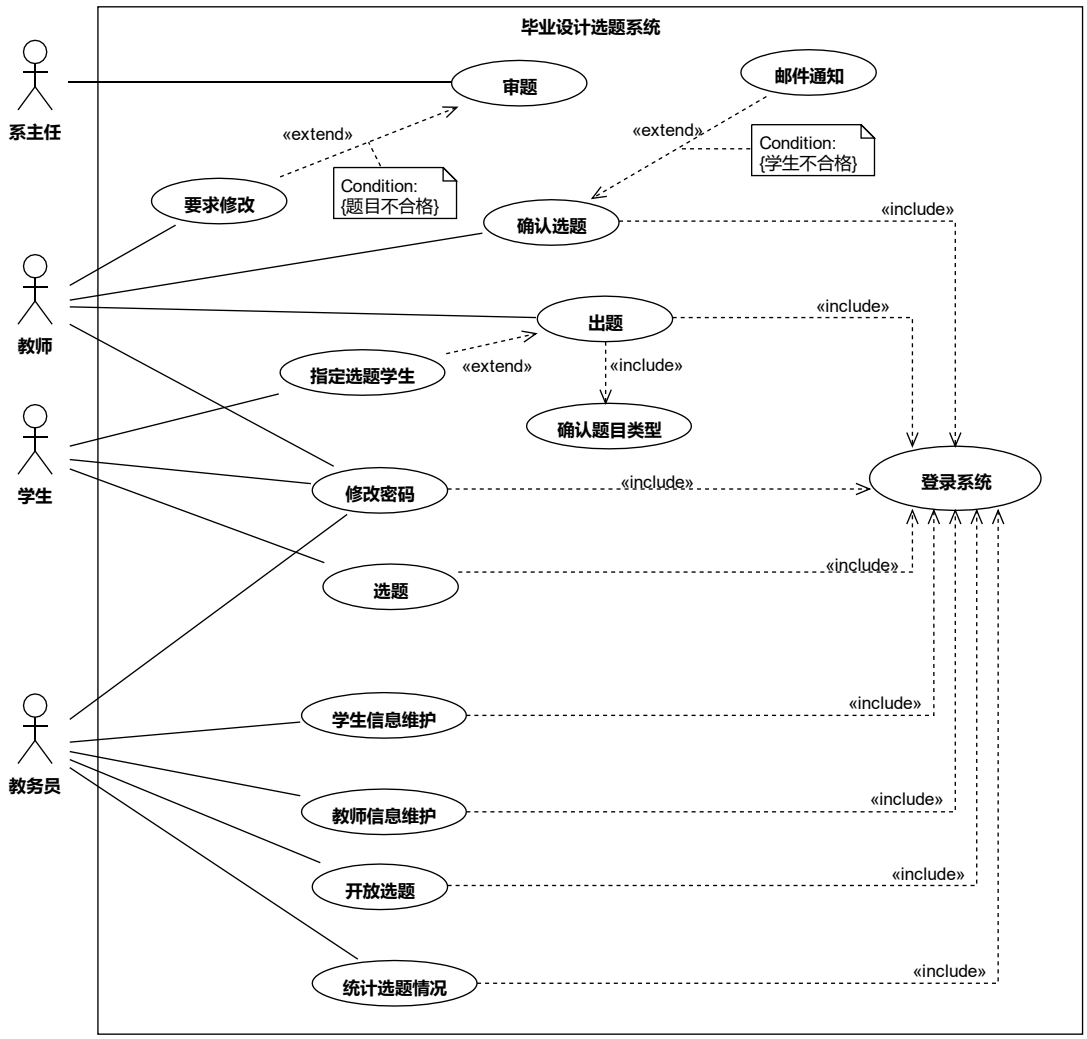
（4）出题：教师使用此功能登记和维护毕业设计的题目。子功能是在出题过程中，要确定题目的类型，如校内或者校外，可选的是直接指定该题目的选题学生。

（5）审题：系主任负责对所有该系教师出的题目进行审核，合格的题目可以发布，不合格的题目要求教师修改。

（6）开放选题：教务员将所有审核通过的题目公开，供学生选择。

（7）选题：学生浏览公开的题目列表，根据题目要求和个人兴趣及特点，选择相应的题目。

（8）确认选题：教师审查自己所出题目的选题情况，对合格的学生予以确认，将不合格的学生删除，并发送邮件通知。



5. 以下给出了“老年人监护系统”中的用例及其描述，使用 UML 用例图描述该系统，并给出用例之间的联系。

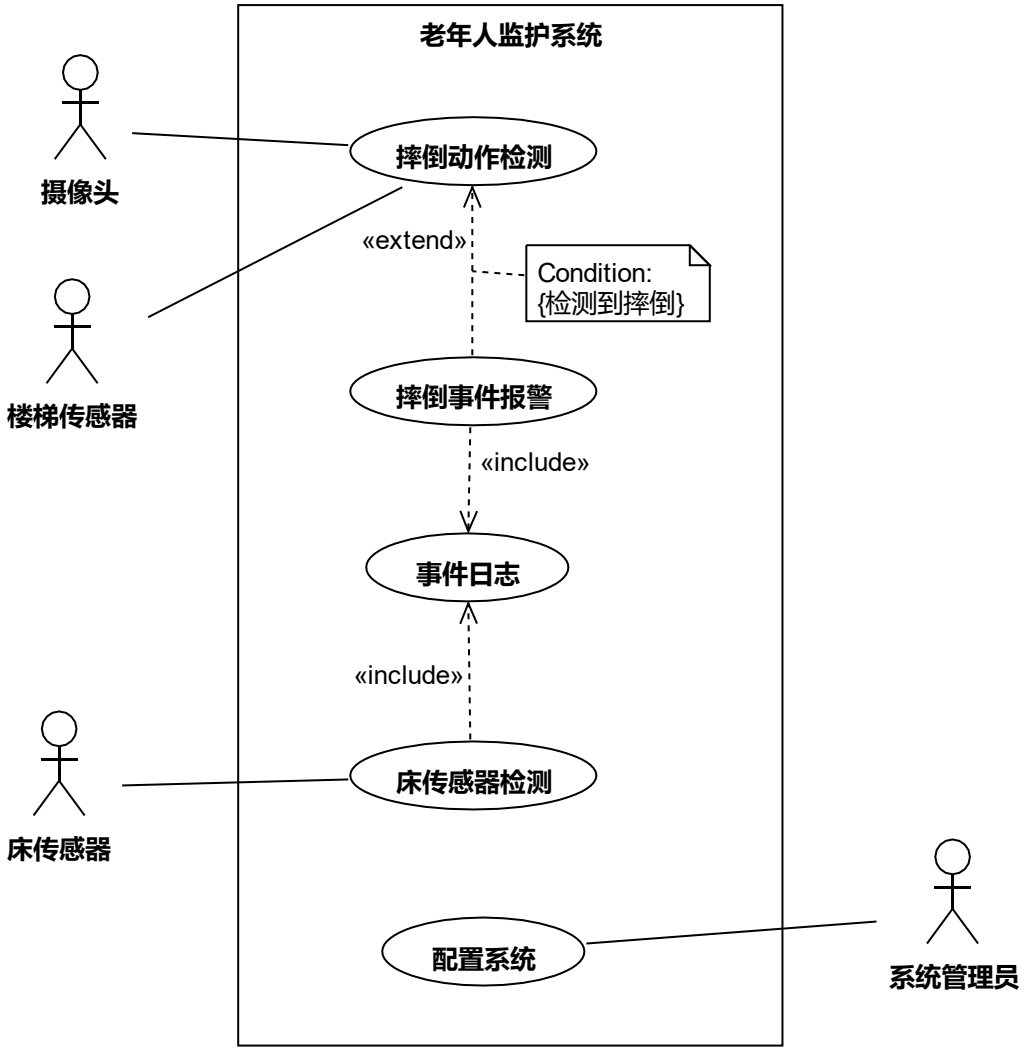
（1）摔倒动作检测：从楼梯传感器和摄像头中获取输入数据，用以检测是否有人摔倒。

（2）摔倒事件报警：如果检测到某位老人摔倒，那么发送一条报警消息到手机上，同时该报警信息会被发送到用例“事件日志”中进行记录。

（3）事件日志：将发生的事件记录在数据库中。

（4）床传感器监测：从安装在床位上的床传感器中获取脉搏、呼吸等数据，并发送到用例“事件日志”中处理。

（5）配置系统：系统管理员对系统进行各种配置操作。

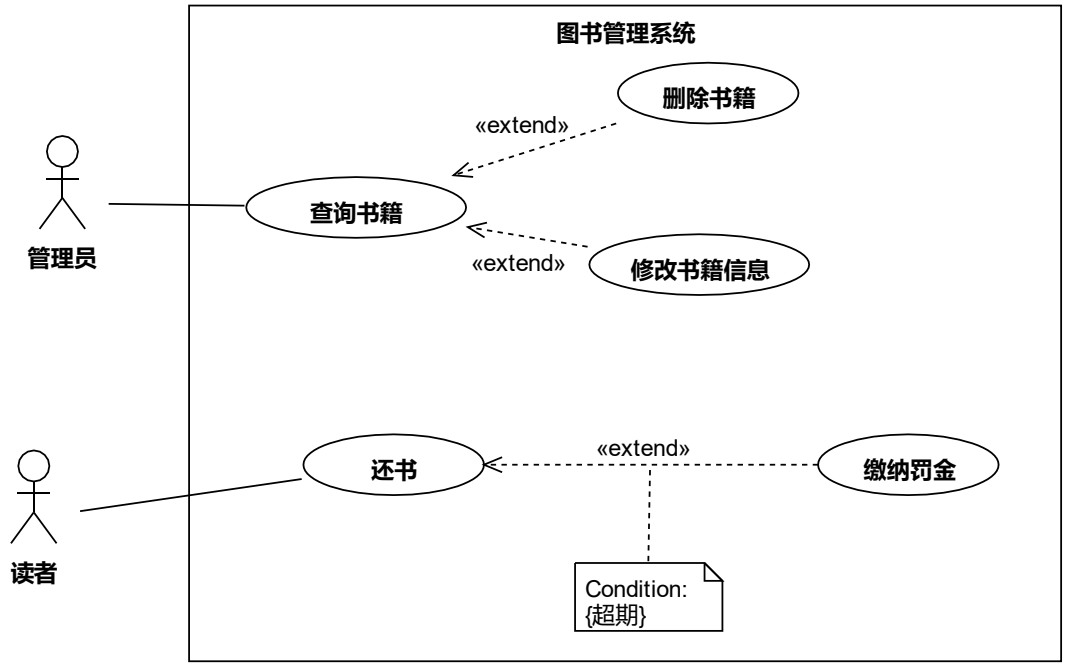


6. 在图书管理系统中：

（1）管理员可进行“删除书籍”和“修改书籍信息”这两个操作，并且这两个操作在执行前都必须先进行“查询书籍”操作。

（2）读者可以“还书”，这是一个基础操作如果读者所借书籍超期，还书时要缴纳罚金，即当书籍“超期”时，将执行“缴纳罚金”操作。

要求：画出上述系统的 UML 用例图。



第 4 章　软件架构的构建

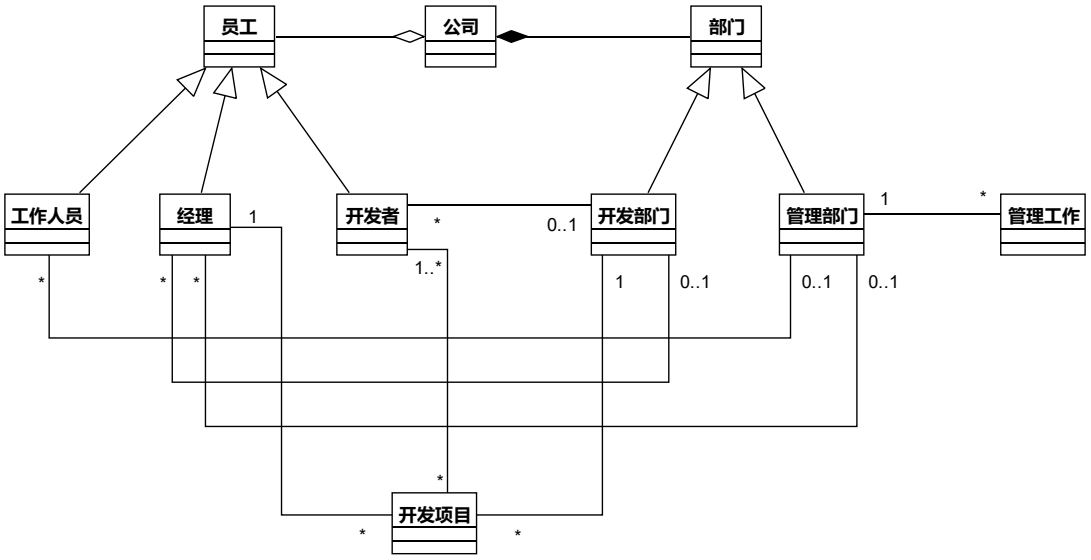
某报告生成系统的处理流程如下：首先从配置文件中加载必要的参数，然后读取 Word 模板文件，模板文件定义了报告的基本格式及相关数据变量的定义及其引用。模板中的内容以数据流的方式依次经过预处理、数据加载及文档生成三个环节，最后产生 doc 或 pdf 格式的报告，如图 4.15 所示。该系统的架构设计采用何种架构模式较为合适？为什么？这种模式有什么优点（两个）？

使用管道与过滤器风格较为合适。因为在该系统的处理流程中，每个构件都有一组输入和输出，下一个阶段的输入是上一个阶段的输出，因此适用管道与过滤器风格。该架构的优点是具有良好的信息隐藏性和模块独立性，从而产生高内聚，低耦合的特点。

第 5 章　类的分析与设计

1. 针对下述描述建立类模型，画出该系统的分析类图。

某软件公司下属的部门分为开发部门和管理部门两类，每个部门由唯一的部门名字确定。每个开发部门可开发多个软件项目，每个管理部门承担公司的若干项日常管理工作。公司的员工分为经理、工作人员和开发者三类。开发部门包括经理和开发者，管理部门包括经理和工作人员。开发项目时，每个项目只能由一位经理主持，但一位经理可主持多个开发项目；每个开发者可参加多个开发项目，每个开发项目也需要多个开发者参与。



2. 以下两个类图的含义有何不同？

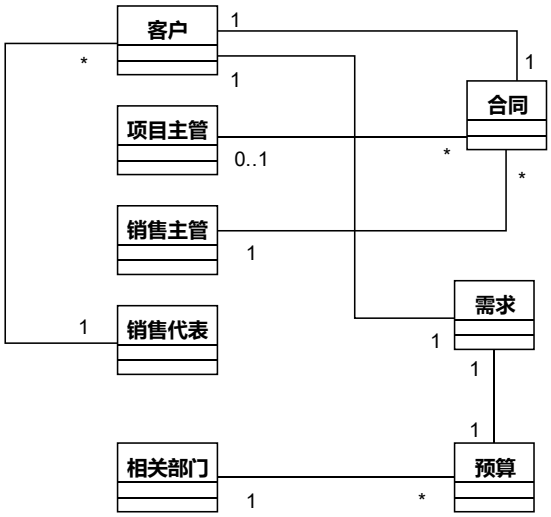
上图表示学生和教室之间的关联关系，即学生在教室上课。这种关联关系是双向的，并且一位学生可以在多个教室有课，一间教室也可以让多个学生上课。

下图表示学生和教室之间的依赖关系，即学生在教室自习。依赖关系是单向的，从学生指向教室，表示在自习活动中学生使用教室。依赖关系与关联关系的主要区别还在于依赖是动态的，体现的是对象的瞬时使用关系，具有偶然性、临时性，且并不需要进行保持。

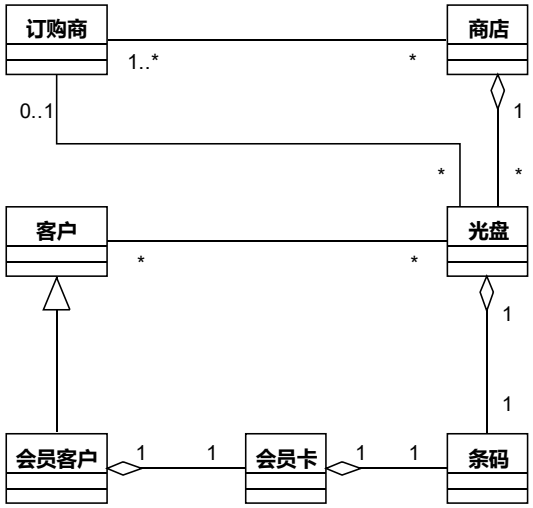
此例中，学生使用教室上自习是比较随意的，自习时产生依赖关系，自习结束后依赖关系就解除了。而学生在教室上课则是较为固定的，是双向的关联关系。

3. 针对第 3 章习题 2、3、4 的描述，在它们用例图的基础上分别给出各自的分析类图。

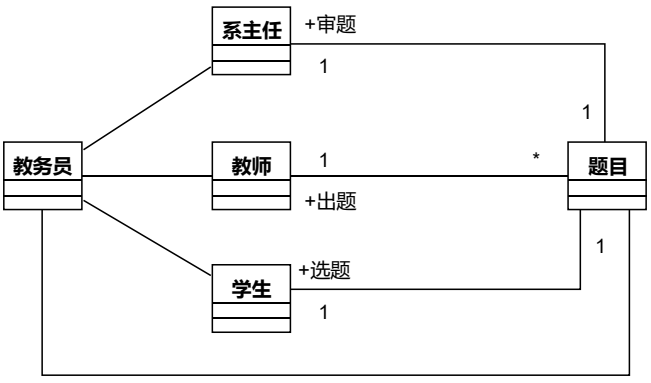
3.2：



3.3：



3.4：

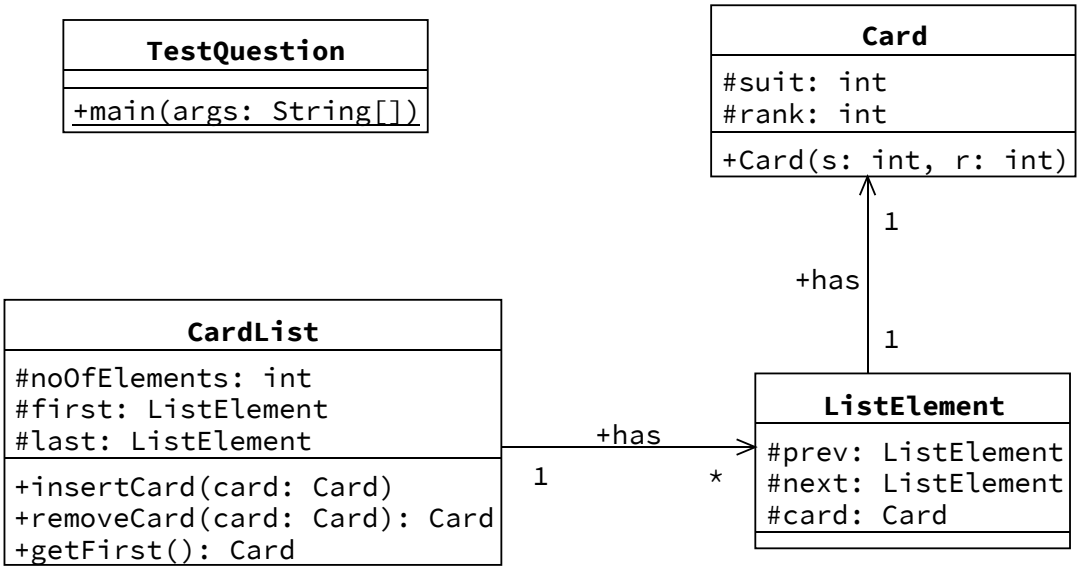


第 6 章　代码生成

1. 详细说明图 6.19 所示的类图中两个关联关系表示的含义。

级队与班级是组合关系，即班级组成了级队。班级与级队有同样的生存期，一旦级队类的对象被销毁，该级队的所有班级类的对象也都会被销毁。教师与班级是一对多的关联关系，一名教师可以作为班主任管理多个班级，一个班级只有一位班主任。

2. 下面给出的代码 6.14 是使用 Java 语言对 4 个类的实现。对这些代码使用逆向工程，要求使用 UML 给出它们的设计类图。



3. 某位系统分析人员针对高校学生选课系统的需求设计了分析类图的草图，如图 6.20 所示。

（1）教务人员通过职工号和密码登录系统，录入课程信息。课程信息包括：课程编号、课程名称、授课时间、授课地点。并为每位教师排课：一名教师可以教授 0 门或多门课程，1 门课程也可以由 1 或多名教师教授。

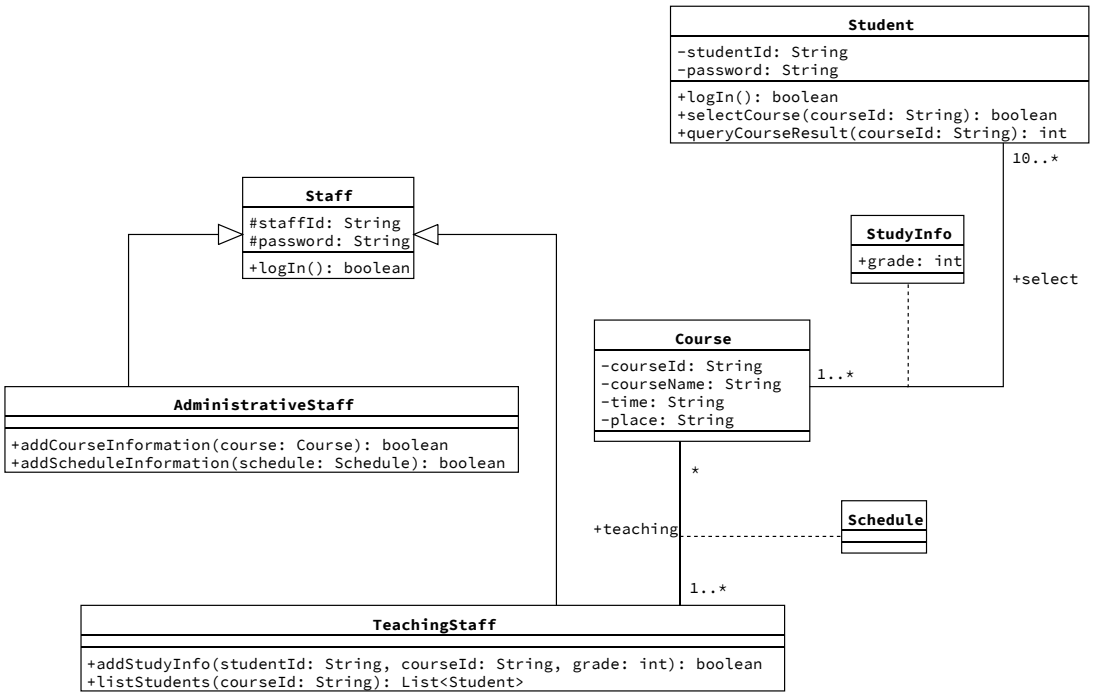
（2）学生通过学号和密码登录系统，选择课程。系统要求 1 门课程至少要有 10 名学生选课才能开课，1 名学生至少要选择一门课程。

（3）教师通过职工号和密码登录系统，获取课程的学生名单。

（4）课程结束后，教师通过该系统录入学生成绩。

（5）学生可通过该系统查询自己的课程成绩。

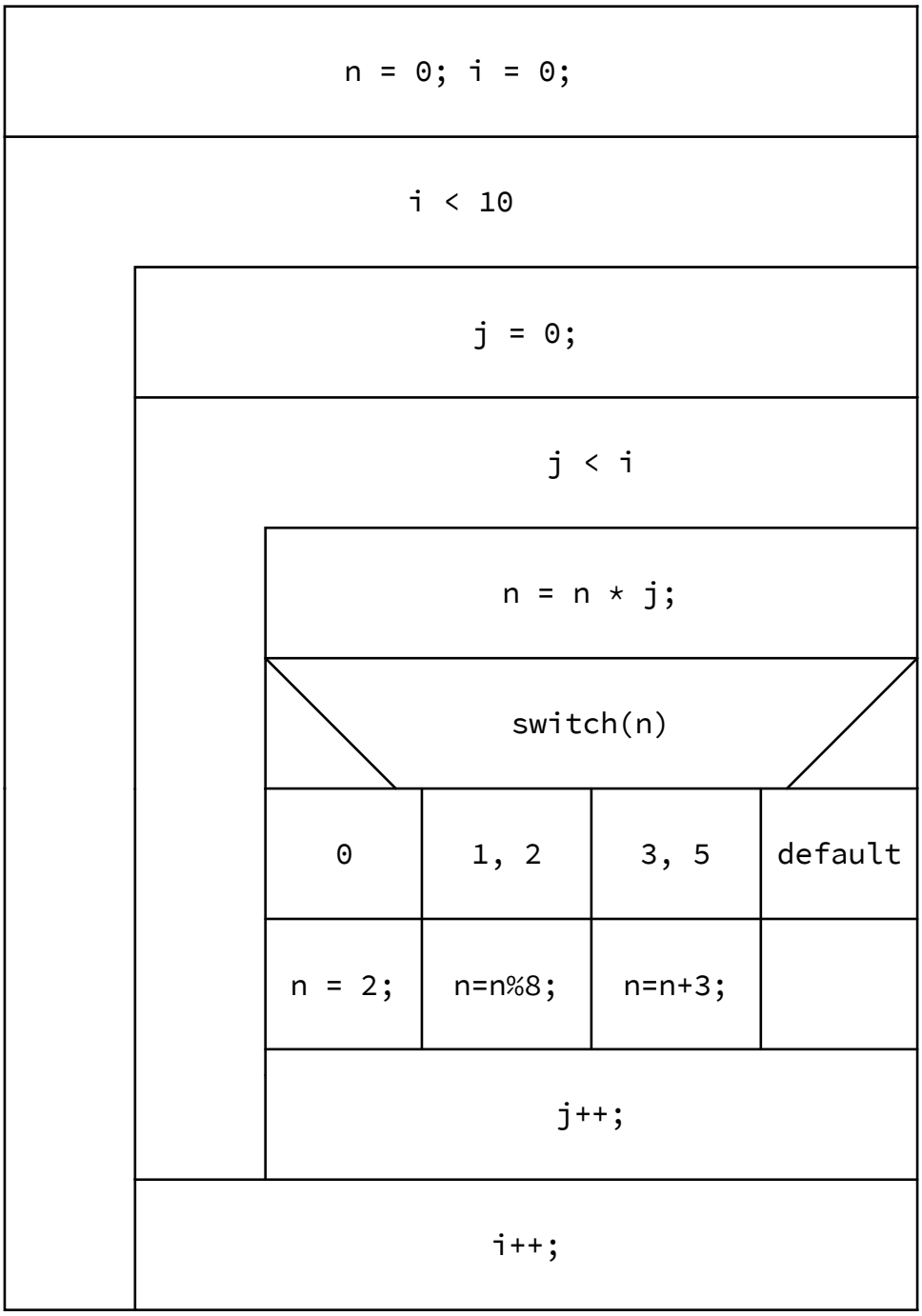
请基于图 6.20 进行细化和完善，尽可能补充需要的信息，完成该系统的设计类图，并尝试使用 Java 或 C++语言编写该系统的代码框架。



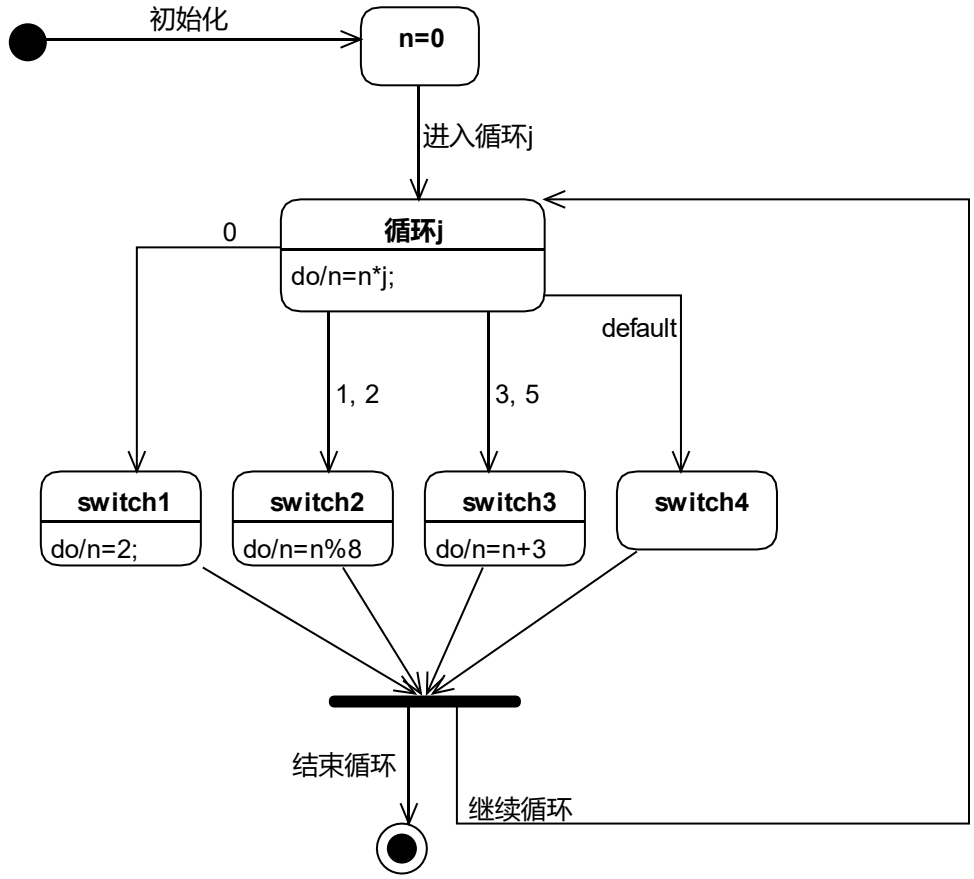
第 7 章　类的详细设计

1. 给出下面代码段的程序盒图，并使用状态转换图对该代码段中变量 n 的状态变化进行描述。

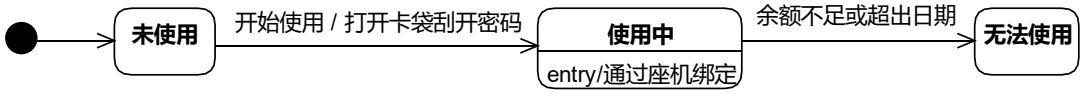
程序盒图：



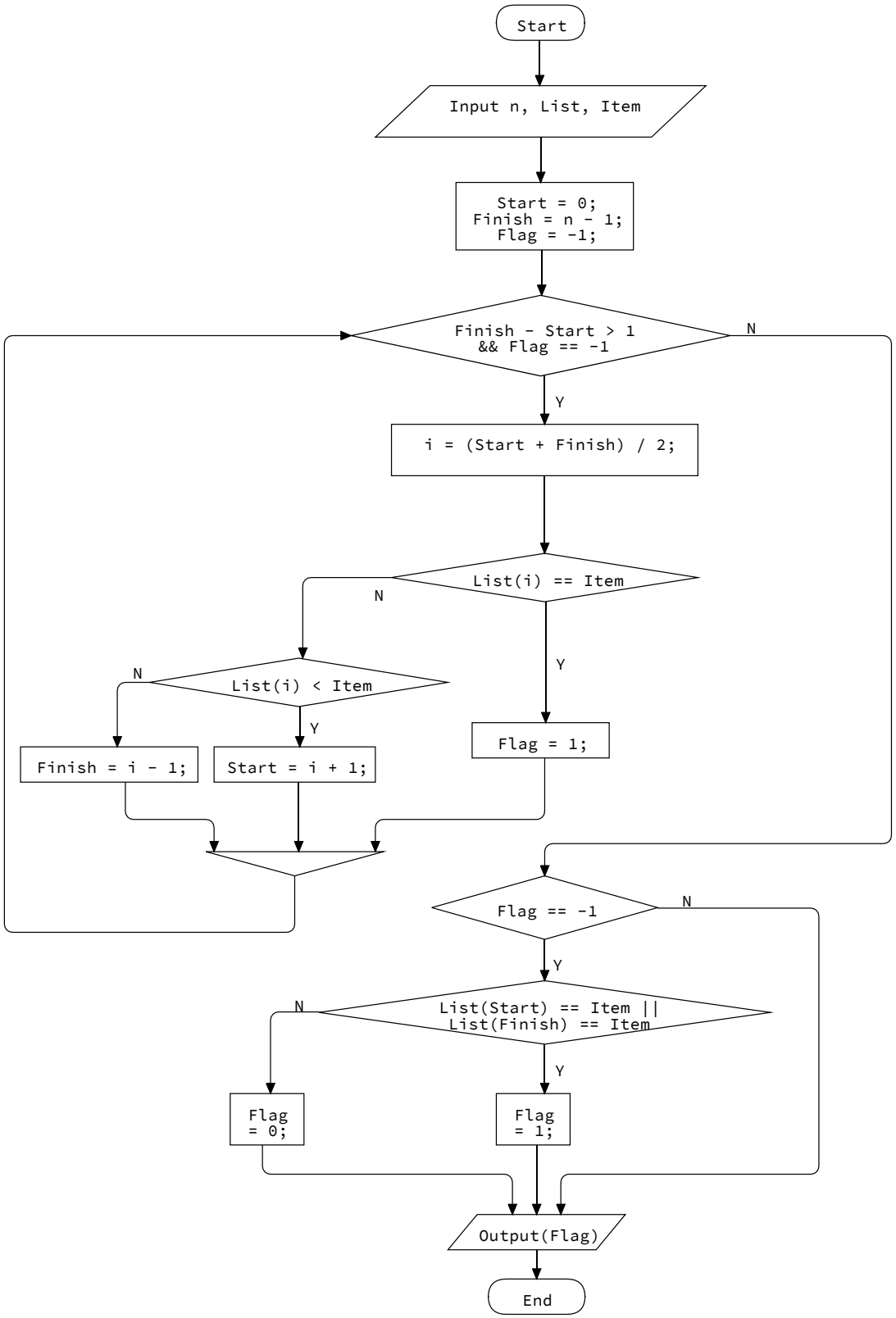
状态转换图：



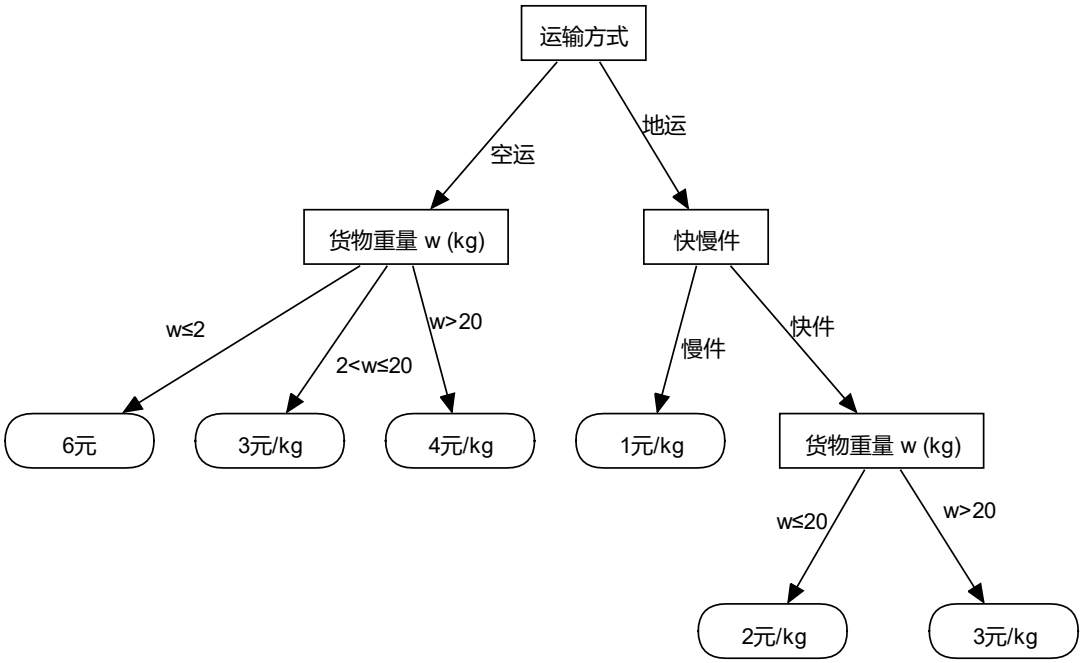
2. IP 长途卡的使用过程如下：未使用前，卡密封在塑料卡袋中，密码被遮挡住；一旦开始使用，使用者必须打开卡袋、刮开密码，通过座机绑定使用长途卡；一旦卡上余额不足或者超出长途卡的使用日期范围，长途卡将无法使用。根据以上描述，分析 IP 长途卡的状态，绘制其状态图。



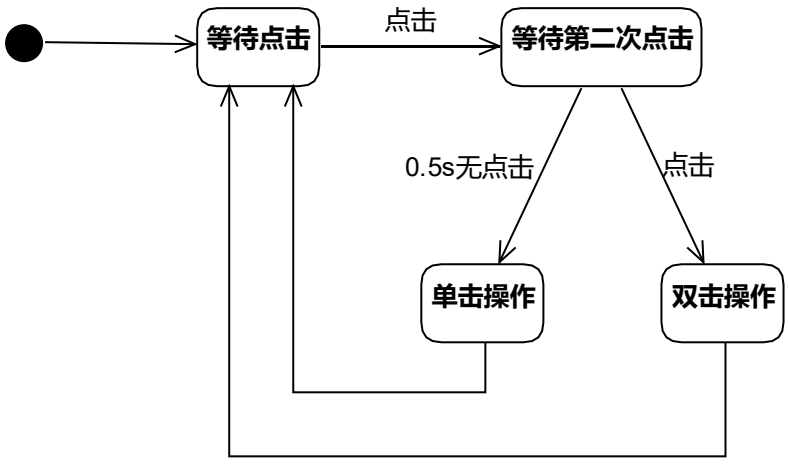
3. 绘制该伪代码对应的程序流程图。



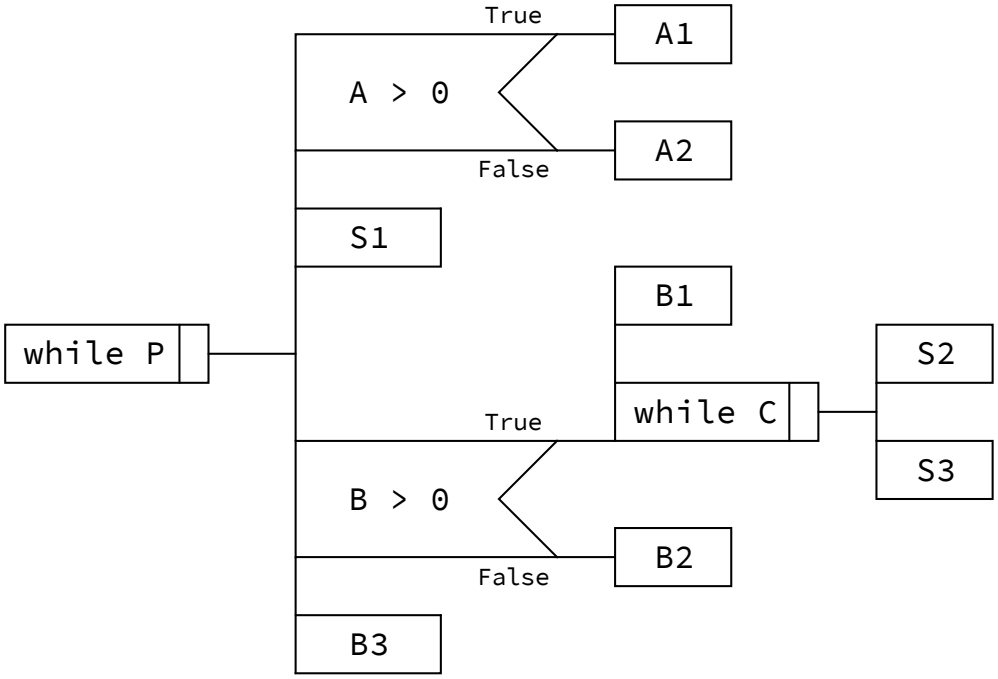
4. 某公司承担空中和地面运输业务。计算货物托运费的规定如下。
[空运]若货物重量小于等于 2kg，则一律收费 6 元；若货物重量大于 2kg 而又小于 20kg，则收费 3 元/kg；若货物重量大于 20kg，则收费 4 元/kg。
[地运]若为慢件，则收费为 1 元/kg。若为快件，当重量小于等于 20kg 时，收费为 2 元/kg；当货物重量大于 20kg 时，收费 3 元/kg。
请画出以上规则的判定树。



5. 绘制一个鼠标单击事件的状态图，将间隔时间或发生的次数作为条件和动作执行的依据。当一个对象被单击选中并且存储在一个变量中时，若在 0.5s 内此对象又被单击，则被认为是一次双击操作；若单击的是另外的对象或者该对象在至少 0.5s 后才被再次单击，则被认为是对该对象的单击操作。



6. 画出下面用 PDL 写出的程序的 PAD 图。



7. 基于类图 5.9，使用 OCL 完善以下约束：

（1）每个项目的计划开始时间不得晚于 2018 年 9 月 5 日。

（2）每个项目不能作为其本身的前驱项目添加到前驱项目列表中。

（3）每个项目的任务集合中不能同时有超过 3 个完成进度不足 50%的项目。

```
context ProjectComponent inv:
  self.startPlanned <= "2018-09-05"

context Project::addPredecessor(project : Project)
  pre: project != self

context Project inv:
  self.selectedTask
    -> select(s | s.compeletePct < 50)
    -> size() <= 3
```

第 8 章　设计优化

1. 图 8.25 所示的类图的设计中使用了观察者设计模式，每个券商（Broker）关注的是股票（Stock）的价格，它们可以将感兴趣的股票设定为自选股，并通过方法 newStock()购买指定的股票。
如果股票的价格发生了变化，所有选定该股票为自选股的券商将会得到通知，包括该股票的名字，这里假定每个股票有唯一的名字。使用 Java 语言对该类图进行实现，要求券商得到股价改变的通知后，输出该券商的名字、股票的名字及股票的最新价格。可将该系统的界面设计为如图 8.26 所示的模式。

BrokerInterface.java

```
package com.nikesu.course.softwareengineering.observer;

public interface BrokerInterface {
    public void update(String stockname);
}
```

Broker.java

```
package com.nikesu.course.softwareengineering.observer;

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

public class Broker implements BrokerInterface {
    private String brokername;
    private List<Stock> stocks = new ArrayList<Stock>();

    public Broker(String brokername) {
        this.brokername = brokername;
        return;
    }

    public void newStock(Stock stock) {
        stocks.add(stock);
        stock.register(this);
    }

    public String getName() {
        return brokername;
    }

    @Override
    public void update(String stockname) {
        System.out.println(brokername + '\t' + stockname);
    }
}
```

Stock.java

```
package com.nikesu.course.softwareengineering.observer;

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

```
public class Stock {
    private String stockname;
    private int val;
    private List<BrokerInterface> observers = new
ArrayList<BrokerInterface>();

    public Stock(String stockname) {
        this.stockname = stockname;
        this.val = 0;
        return;
    }

    public void register(BrokerInterface b) {
        observers.add(b);
        return;
    }

    public void notifyAllObservers() {
        for (BrokerInterface broker : observers) {
            broker.update(stockname);
        }
        return;
    }

    public int getVal() {
        return this.val;
    }

    public void setVal(int value) {
        this.val = value;
        notifyAllObservers();
        return;
    }

    public String getName() {
        return this.stockname;
    }
}
```

ObserverDemo.java

```
package com.nikesu.course.softwareengineering.observer;

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

public class ObserverDemo {
    private static List<Stock> stocks = new ArrayList<Stock>();
    private static List<Broker> brokers = new ArrayList<Broker>();
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        while (true) {
            System.out.println("1. 新建券商");
            System.out.println("2. 新建股票");
            System.out.println("3. 券商自选股票");
```

```
            System.out.println("4. 更改股票价格");
            String a = scanner.next();
            switch(a) {
            case "1":
                System.out.print("姓名: ");
                String brokername = scanner.next();
                brokers.add(new Broker(brokername));
                break;
            case "2":
                System.out.print("名字: ");
                String stockname = scanner.next();
                stocks.add(new Stock(stockname));
                break;
            case "3":
                System.out.println("选择券商和股票: ");
                for (int i = 0; i < brokers.size(); i++) {
                    System.out.println(i + "\t" +
                            brokers.get(i).getName());
                }
                for (int i = 0; i < stocks.size(); i++) {
                    System.out.println(i + "\t" +
                            stocks.get(i).getName());
                }
                int b = scanner.nextInt();
                int s = scanner.nextInt();
                brokers.get(b).newStock(stocks.get(s));
                break;
            case "4":
                System.out.println("选择股票, 输入新价格: ");
                for (int i = 0; i < stocks.size(); i++) {
                    System.out.println(i + "\t" +
                            stocks.get(i).getName());
                }
                s = scanner.nextInt();
                int p = scanner.nextInt();
                stocks.get(s).setVal(p);
                break;
            default:
                System.out.println("Bye.");
               System.exit(0);
            }
        }
    }
}
```

2. 图 8.27 是合成（Composite）模式的结构。合成模式把部分与整体的关系用树结构进行表示，使用户对单对象和组合对象的使用具有一致性。请给出一个合成模式的具体应用，并使用 Java 语言（或其他面向对象语言）进行实现。

一个文件系统就是一个典型的合成模式系统。文件系统是一个树结构，树的节点有两种，一种是树枝节点，即目录，有内部树结构；另一种是文件，即树叶节点，没有内部树结构。

又如下面这段代码使用合成模式实现了一个图形类，该类的一个实例既可是一个椭圆形，也可是一个图形类对象的列表。该类有一个绘制方法 print()。

CompositeDemo.java

```
import java.util.ArrayList;

/** "Component" */
interface Graphic {
    //Prints the graphic.
    public void print();
}

/** "Composite" */
class CompositeGraphic implements Graphic {
    //Collection of child graphics.
    private final ArrayList<Graphic> childGraphics = new ArrayList<>();

    //Adds the graphic to the composition.
    public void add(Graphic graphic) {
        childGraphics.add(graphic);
    }

    //Prints the graphic.
    @Override
    public void print() {
        for (Graphic graphic : childGraphics) {
            graphic.print();  //Delegation
        }
    }
}

/** "Leaf" */
class Ellipse implements Graphic {
    //Prints the graphic.
    @Override
    public void print() {
        System.out.println("Ellipse");
    }
}

/** Client */
public class CompositeDemo {
    public static void main(String[] args) {
        //Initialize four ellipses
        Ellipse ellipse1 = new Ellipse();
        Ellipse ellipse2 = new Ellipse();
        Ellipse ellipse3 = new Ellipse();
        Ellipse ellipse4 = new Ellipse();

        //Creates two composites containing the ellipses
```

```
        CompositeGraphic graphic2 = new CompositeGraphic();
        graphic2.add(ellipse1);
        graphic2.add(ellipse2);
        graphic2.add(ellipse3);

        CompositeGraphic graphic3 = new CompositeGraphic();
        graphic3.add(ellipse4);

        //Create another graphics that contains two graphics
        CompositeGraphic graphic1 = new CompositeGraphic();
        graphic1.add(graphic2);
        graphic1.add(graphic3);

        //Prints the complete graphic (Four times the string "Ellipse").
        graphic1.print();
    }
}
```

3. 熟悉和掌握表 8.1 中给出的其他设计模式。

   好的，熟悉了。

第 9 章　实现技术

1. 以下 XML 文档是 well-formed 的吗？请指出错误并加以改正。

不是 well-formed 的，改正如下：

```
<?xml version="1.0" encoding="GB2312"?>
<users>
    <user id="1">
        <name>John</name>
        <password>123</password>
        <roles>
            <role>admin</role>
        </roles>
    </user>
    <user id="2">
        <name>Mike</name>
        <password>abc</password>
        <roles>
            <role>guest</role>
            <role>buyer</role>
        </roles>
    </user>
</users>
```

2. 对于以下给定的 XML 文档，编写一段 Java 程序，分别使用 DOM 和 SAX 两种方式计算所有书目的价格之和。

DomParser.java

```
package com.nikesu.course.softwareengineering.xml;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

public class DomParser {
    private DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
    private DocumentBuilder builder = null;
    private String path;

    public DomParser(String path) {
        this.path = path;
    }

    public void parse() {
        double sum = 0.0;
        try {
            builder = factory.newDocumentBuilder();
```

```
            Document document = builder.parse(path);
            NodeList nodelist = document.getElementsByTagName("price");
            for (int i = 0; i < nodelist.getLength(); i++) {
                Node node = nodelist.item(i);
                sum += Double.parseDouble(node.getFirstChild()
                        .getNodeValue());
            }
        } catch(Exception e) {
            e.printStackTrace();
        }
        System.out.println("DOM Parser: sum = " + sum);
    }
}
```

SaxParser.java

```
package com.nikesu.course.softwareengineering.xml;

import java.io.File;
import java.util.Stack;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

class SaxHandler extends DefaultHandler {
    private Stack<String> stack = new Stack<>();
    private double sum = 0.0;

    @Override
    public void startElement(String uri, String localName,
            String qName, Attributes attributes) throws SAXException {
        stack.push(qName);
    }

    @Override
    public void characters(char[] ch, int start, int length)
            throws SAXException {
        if ("price".equals(stack.peek())) {
            sum += Double.parseDouble(new String(ch, start, length));
        }
    }

    @Override
    public void endElement(String uri, String localName,
            String qName) throws SAXException {
        stack.pop();
        if (stack.isEmpty()) {
            System.out.println("SAX Parser: sum = " + sum);
        }
    }
}

public class SaxParser {
    private SAXParserFactory factory = SAXParserFactory.newInstance();
```

```
    private SAXParser parser = null;
    private String path;

    public SaxParser(String path) {
        this.path = path;
    }

    public void parse() {
        try {
            parser = factory.newSAXParser();
            parser.parse(new File(path), new SaxHandler());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

XmlDemo.java

```
package com.nikesu.course.softwareengineering.xml;

public class XmlDemo {
    public static void main(String args[]) {
        DomParser domParser = new DomParser("data\\books.xml");
        SaxParser saxParser = new SaxParser("data\\books.xml");
        domParser.parse();
        saxParser.parse();
        return;
    }
}
```

3. 针对上题中使用 XML 格式的数据，设计关系模型并保存在数据库中，使用 Java 通过 JDBC 计算上题中要求的结果。

jdbcDemo.java

```java
package com.nikesu.course.softwareengineering.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class jdbcDemo {
    public static void main(String[] args) throws Exception {
        double sum = 0.0;

        String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
        String URL = "jdbc:mysql://localhost:3306/BOOK?" +
                "useSSL=false&serverTimezone=Asia/Shanghai";
        String USERNAME = "root";
        String PASSWORD = "123456";
        String query = "SELECT price FROM books;";
        Class.forName(JDBC_DRIVER);

        Connection connection =
                DriverManager.getConnection(URL, USERNAME, PASSWORD);
        Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery(query);
        while (resultSet.next()) {
            sum += resultSet.getDouble("price");
        }
        System.out.println("sum = " + sum);
    }
}
```

第 10 章　交互设计

1. 本章描述的 ISO 9241 对话原则适用于所有软件系统中的界面设计。请针对每个对话原则，根据自己对某软件系统（如腾讯 QQ、新浪微博、淘宝等）的使用体验，分别给出 2 个正面的应用示例并简单描述。

腾讯 QQ，当用户鼠标悬浮在“联系人”图标上时，会在鼠标旁边弹出带有“联系人”三字的说明，符合自我描述性；当用户在某个群聊被禁言，其相应的群聊界面的输入框将变灰，且有禁言中的相关提示，以明示用户不可输入，符合与用户期望一致性。

2. 扁平化设计（Flat Design）就是在进行界面设计的过程中，去除所有具有三围突出效果的风格和属性，即去除下落式阴影、梯度变化、表面质地差别，以及所有具有三维效果的设计风格。很多流行的系统都采用了扁平化的设计方式，如 iOS、Android 和微软 Metro 风格的系统。请结合交互设计的原则，解释扁平化界面设计的优势和不足。

优势：扁平化设计去除了冗余、厚重和繁杂的装饰效果，仅留下有用的信息和按钮，使界面变得更加干净整齐，使用起来更加简洁，更符合任务适应性；

不足：过于简洁的界面对于新用户来讲较为陌生，与拟物化的设计风格相比，易学性较差。

第 11 章　软件测试

1. 图 11.15 所示的流程图描述了某子程序的处理流程，现要求用白盒测试方法对子程序进行测试，并回答以下问题：

（1）满足条件覆盖的测试数据集，是否一定能满足分支覆盖？请举例。

（2）给出满足多条件组合覆盖的最小测试用例组。

（1）不一定。如测试用例 a=1， b=-4 和 a=-1， b=4 就满足条件覆盖，但不满足分支覆盖。

（2）{{a=1， b=-4}，{a=-4,b=1}，{a=0，b=0}}

2. 阅读代码 11.21，回答下列问题。

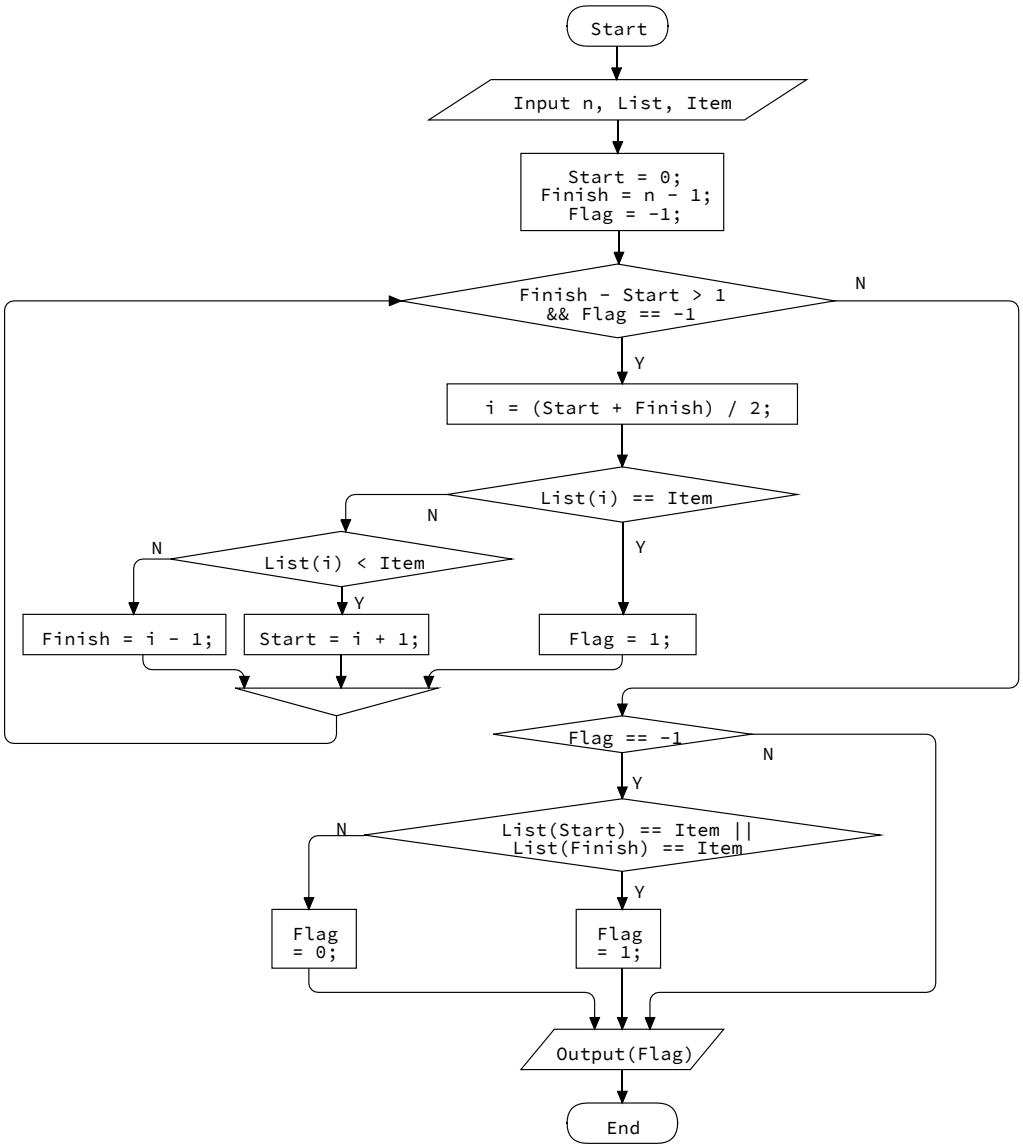
（1）绘制该代码对应的程序流程图；

（2）绘制该代码对应的控制流图；

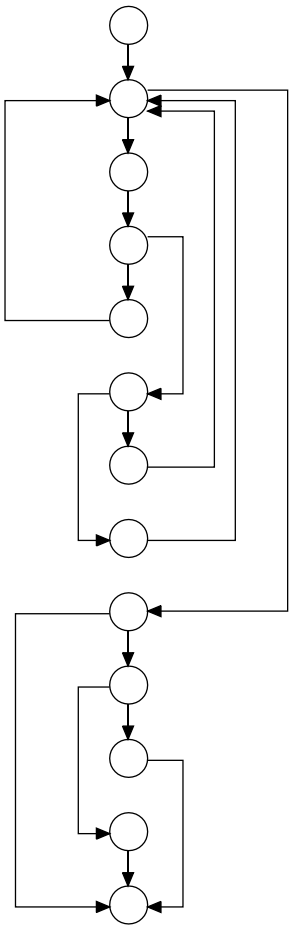
（3）计算环形复杂度；

（4）使用条件覆盖标准设计该程序的测试用例。

（1）



（2）



（3）McCabe = 6

（4）测试用例如下：

```
List = {1, 2, 3}, Item = 0
List = {1, 2, 3}, Item = 1
List = {1, 2, 3}, Item = 2
List = {1, 2, 3}, Item = 3
```

3. 某函数完成教师课时津贴标准的计算。其输入参数有两个：教师职称及是否为外聘教师；输出为该教师的津贴标准。具体如下。
本校专职教师的每课时津贴费：教授 50 元，副教授 40 元，讲师 30 元，助教 20 元；外聘兼职教师每课时津贴费：教授 50 元，副教授 50 元，讲师 30 元，助教 30 元。
使用等价类分析方法给出该函数的弱等价类测试用例。

1. 专职教授；
2. 专职副教授；
3. 兼职讲师；
4. 兼职助教。

4. 图 11.16 所示是某杂志社稿件处理系统中稿件类（Article）的状态图，根据该图回答问题。
（1）给出 Article 类的定义，包括属性和方法。
（2）列出所有需要测试的类状态。
（3）列出所有需要测试状态的转换。
（4）从初始态开始（各属性置为空值），一篇名为 *A Good Paper* 的稿件投稿，由于符合杂志领域范围，通过初审，然后送外审，外审专家认为质量不错，但需要继续修改，论文修改后达到要求，被录用。为以上场景开发一个测试驱动类，编写代码。

（1）

```
public class Article {
    public String tname;
    public boolean inDomain;
    public boolean qualityOK;
    public boolean isApproved;

    public void submit(String tname) {
        this.tname = tname;
    }

    public void initialCheck() {

    }

    public void review() {

    }

    public void refine() {

    }

    public void reject() {

    }

    public void accept() {

    }
}
```

（2）Submitted, InitialChecked, PeerReviewed, Refined, Rejected, Acceptted

（3）

状态 A	事件	状态 B
Submitted	initialCheck()	InitialChecked
InitialChecked	review()	PeerReviewed
	reject()	Rejected
PeerReviewed	refine()	Refined
	reject()	Rejected
Refined	accept()	Acceptted
	reject()	Rejected

(4)

```
public class ArticleTest {
    public static void main(String[] args) {
        Article article = new Article();
        article.inDomain = true;
        article.qualityOK = true;
        article.isApproved = true;

        article.submit("A Good Paper");
        article.initialCheck();
        article.review();
        article.refine();
        article.accept();
        return;
    }
}
```

第 12 章　软件项目级管理

1. 根据版本管理的机制设计一个版本仓库，并使用工具实现它。

好的。

2. 选择使用第 3 章习题中描述的一个系统需求，完成系统规模、工作量和成本估算的过程并给出结果。

未调整的功能点数

任务需求	输入 3-4-6	查询 3-4-6	输出 4-5-7	内部数据 7-10-15	外部文件 5-7-10	合计
查询书籍	3	3	5	7	5	23
修改书籍信息	6	-	-	7	7	20
还书	3	4	4	7	7	25
缴纳罚金	4	4	4	7	7	26
合计						94

调整后的功能点数

未调整的功能点数（UFP）		113	备注
影响因子	与其它系统的交互（0~5）	0	无交互
	分布式数据的处理（0~5）	3	-
	事物的高处理率（0~5）	4	-
	处理逻辑		
	计算复杂性（0~10）	1	-
	控制复杂性（0~5）	1	-
	出错处理（0~10）	4	-
	业务逻辑（0~5）	2	-
	可重用性（0~5）	1	-
	可移植性（0~5）	1	-
	可维护性（0~5）	5	-
综合影响合计（DI）		22	-
影响因子（TCF）		0.92	-
调整后功能点数（FP）		86.48	-

不妨令系数 $A = 3$，程序长度（千代码行） $Size = 1$，工作量系数 $EM_i = 1, 1 \le i \le 7$，正则化因子 $B = 0.91$，比例系数 $SF_i = 3.5, 1 \le j \le 5$，可得

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_j = 1.085$$

从而工作量

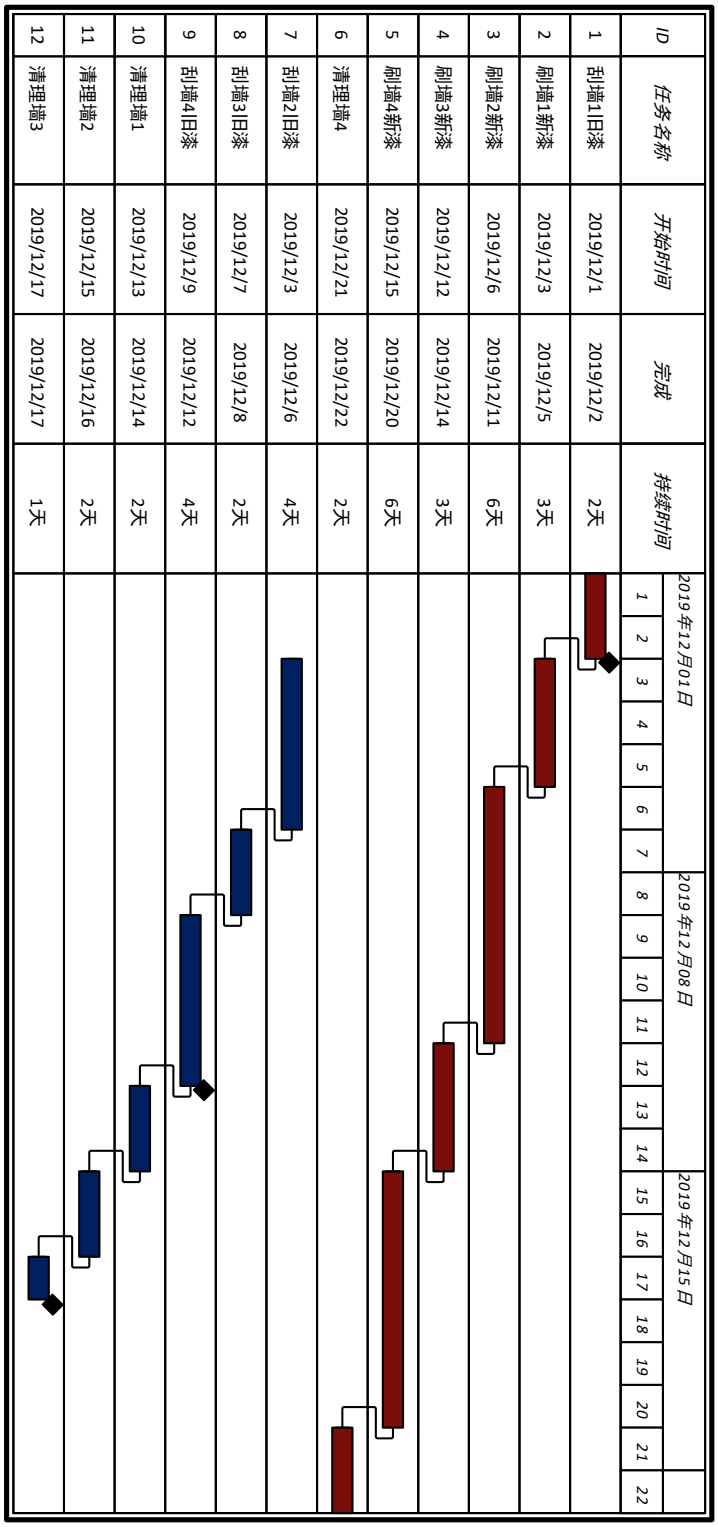
$$PM = A \times Size^E \times \prod_{i=1}^{n} EM_i = 3$$

3. 给定如图 12.22 所示的工程网络图，补充计算每个工作包的计划安排，并指出其关键路径。

关键路径是：

刮墙 1 旧漆->刷墙 1 新漆->刷墙 2 新漆->刷墙 3 新漆->刷墙 4 新漆->清理墙 4。

4. 在上题的基础上，为 2 人的团队使用甘特图制定计划安排，要求指定每项任务的责任人，并标出可能的里程碑位置。



5. 根据表 12.10 中项目的成本数据，计算出项目的 BAC，然后根据 50/50 和 0/100 规则，计算 BCWS、ACWP、BCWP，并进一步计算 SPI 和 CPI，说明在 7 月 1 日时项目的状况如何。

$BAC = 5 + 20 + 80 + 30 + 40 + 50 = 225$（千元），在 7 月 1 日时，工作任务 T1 至 T5 均全部完成，T6 还没有开始。由此计算 $BCWS$，$ACWP$ 和 $BCWP$：

工作任务	$BCWS$（千元）	$ACWP$（千元）	$BCWP$（千元） 50/50 规则	$BCWP$（千元） 0/100 规则
T1	5	10	5	5
T2	20	15	20	20
T3	80	60	80	80
T4	30	40	30	30
T5	40	45	40	40
T6	0	0	0	0
合计	175	170	175	175

注意到 7 月 1 日时，所有任务要么是已经完成，要么是还未开始，因此使用 50/50 规则和 0/100 规则计算出的 $BCWP$ 是一样的。

由 $SPI = BCWP/BCWS = 1$ 可知，项目在 7 月 1 日时的实际进度与计划进度相同；

由 $CPI = BCWP/ACWP > 1$ 可知，项目在 7 月 1 日时的实际费用低于预算费用。

6. 软件质量保证的工作任务有哪些？

（1）编制项目质量保证计划。

（2）参与编写项目的软件过程描述。

（3）评审软件工程活动，以验证其是否符合规定。

（4）审核指定的软件工程产品，以验证是否遵守规定。

（5）确保根据文档化的规程，记录和处理软件工程和工程产品中的偏差，包括在项目计划、过程描述、适用的标准或软件工程工作产品中存在的偏差。

（6）记录各种不符合项并报告给上层管理人员。

第 13 章　软件过程管理及改进

1. CMMI 阶段式表述中 5 个等级的关注点分别是什么？

1. 初始级：开发的初级阶段，没有引入任何系统化的过程控制。开发过程存在很大的随意性，开发结果存在较大的不确定性，并且难以理解和回顾，开发过程经常返工，工作量翻倍是常态。被动地等待问题的出现以及救火式的处理，项目成败极大的依赖员工的技能和个人的承诺。

2. 已管理级：这个阶段的项目可以再现，总结出了项目开发的特点和管理经验，项目管理起到了重要的作用。引入了关键的子过程：需求管理（REQM）、项目计划（PP）、项目跟踪和控制（PMC）、过程和产品质量保证（PPQA）、配置管理（CM）、供应商协议管理（SAM）和度量与分析（MA）。但仍然是被动的问题应对方式。

3. 已定义级：所有已有的过程都进行了统一的文档化，它们能够被公司范围所理解和利用，并为其它项目提供一个统一的框架。公司范围对过程的分析综合和协调是管理的核心，为此需要下面的子过程的支持：需求定义（RD）、验证（Val）、确认（Ver）、技术方案（TS）、风险管理（RSKM）、组织级过程聚焦（OPF）、组织级过程定义（OPD）、组织级培训（OT）、集成项目管理（IPM）、决策分析和决定（DAR）。这是一个主动的问题应对方式。

4. 已量化管理级：为了能够识别出哪些过程改动会带来什么样的质量变化，需要对过程和产品质量进行量化的度量。每个过程的表现需要能够度量，并基于量化结果进行自我分析的能力，通过以下两个子过程进行支持：组织级过程性能（OPP）和量化项目管理（QPM）。

5. 优化级：在此阶段，公司能够在基于前面的基础阶段之上确定更为合理的优化目标和及时识别并做出必要的过程调整。包含的子过程包括：组织级革新与实施（OID）和原因分析与解决（CAR）。

2. CMMI 中过程域 PA 分为哪些类型？每一种类型中都有哪些过程域？

CMMI 中过程域主要分为四大类 22 个，如下表所示：

缩写	英文名称	中文名称	领域类别	成熟度等级
CAR	Causal Analysis and Resolution	因果分析与解决	支持类	5
CM	Configuration Management	配置管理	支持类	2
DAR	Decision Analysis and Resolution	决策分析与解决	支持类	3
IPM	Integrated Project Management	集成项目管理	项目管理类	3
MA	Measurement and Analysis	测量与分析	支持类	2
OPD	Organizational Process Definition	组织级过程定义	过程管理类	3
OPF	Organizational Process Focus	组织级过程焦点	过程管理类	3
OPM	Organizational Performance Management	组织的绩效与管理	过程管理类	5
OPP	Organizational Process Performance	组织过程性能	过程管理类	4
OT	Organizational Training	组织培训	过程管理类	3
PMC	Project Monitoring and Control	项目监督与控制	项目管理类	2
PP	Project Planning	项目计划	项目管理类	2
PPQA	Process and Product Quality Assurance	过程和产品质量保证	支持类	2
QPM	Quantitative Project Management	量化的项目管理	项目管理类	4
REQM	Requirements Management	需求管理	项目管理类	2
RSKM	Risk Management	风险管理	项目管理类	3
SAM	Supplier Agreement Management	供应商协议管理	项目管理类	2
RD	Requirement Development	需求开发	工程类	3
TS	Technical Solution	技术解决方案	工程类	3
PI	Product Integration	产品集成	工程类	3
VAL	Validation	确认	工程类	3
VER	Verification	验证	工程类	3

3. 举例说明 CMMI 的过程域 PA 的结构及其作用。

过程域的描述使用了一个统一的框架结构。每个过程域都需要有目的陈述、简介、相关过程域，然后要有特定目标及其达成目标的特定实践，特定实践中包含了一系列具体实施的子实践及使用的工作产品实例。还需要有通用目标，通用目标的达成依赖于实施通用实践，每个通用实践也包含一系列的子实践和对通用实践的详细说明。

以过程域“项目计划”为例，它的目的陈述为“项目计划（Project Planning, PP）的目的在于建立并维护定义项目活动的计划”；简介为“项目计划是有效管理项目的关键之一。项目计划过程域包括以下活动……”;特定目标包括“项目计划参数的估算得到建立和维护”等；特定实践包括“估算项目范围”等。

4. 简单说明 PSP 的含义。

个体软件过程（Personal Software Process, PSP）是一种可用于控制、管理和改进个人工作方式的自我持续改进过程，是一个包括软件开发表格、指南和规程的结构化框架。PSP 的目标是培养开发者在尽可能准确设置的时间段中完成高质量的工作。

5. 简单说明 TSP 的含义。

团队软件过程（Team Software Process, TSP）对软件过程的定义、度量和改进提出了一整套原则、策略和方法，把 CMMI 要求实施的管理与 PSP 要求开发者具有的技巧结合起来，旨在按时交付高质量的软件，并把成本控制在预算的范围之内。

TSP 是一个框架，在这个框架中，个人可以将自己的工作过程和技能结合团队成熟的过程管理技术，完成高质量的工作。