МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

ПАТТЕРНЫ ПРОГРАММИРОВАНИЯ

Методические рекомендации к лабораторным работам для студентов направления подготовки 09.03.04 «Программная инженерия» дневной формы обучения

Могилев 2023

УДК 621.01
ББК 36.4
П20

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «31» августа 2023 г., протокол № 1

Составитель канд. техн. наук, доц. Ю. В. Вайнилович

Рецензент Ю. С. Романович

Методические рекомендации разработаны на основе рабочей программы по дисциплине «Паттерны программирования» для студентов направления подготовки 09.03.04 «Программная инженерия» и предназначены для использования при проведении лабораторных работ.

Учебное издание

ПАТТЕРНЫ ПРОГРАММИРОВАНИЯ

Ответственный за выпуск В. В. Кутузов

Корректор А. А. Подошевко

Компьютерная верстка Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс. Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

© Белорусско-Российский
университет, 2023

Содержание

1 Основные правила выполнения лабораторных работ 4
2 Лабораторная работа № 1. Язык моделирования UML. Построение диаграмм классов 4
3 Лабораторная работа № 2. Язык моделирования UML. Построение диаграмм взаимодействия 7
4 Лабораторная работа № 3. Разработка программ с использованием принципа единственной обязанности (SRP) и принципа открытости/закрытости (OCP) 7
5 Лабораторная работа № 4. Разработка программ с использованием принципа подстановки Лисков (LSP), принципа разделения интерфейсов (ISP) и принципа инверсии (DIP)................................ 8
6 Лабораторная работа № 5. Разработка программ с использованием порождающих паттернов 9
7 Лабораторная работа № 6. Разработка программ с использованием структурных паттернов 14
8 Лабораторная работа № 7. Разработка программ с использованием паттернов поведения 19
9 Лабораторная работа № 8. Разработка программ с использованием шаблонов GRASP 24
Список литературы 28

1 Основные правила выполнения лабораторных работ

1 Ознакомиться с целью предстоящей работы и условием индивидуального задания.

2 Изучить вопросы раздела «Теоретические сведения».

3 Разработать логическую модель решения задачи индивидуального задания, используя язык моделирования UML.

4 Набрать и отладить текст программы на ПК в среде Visual Studio.NET.

5 Подготовить отчёт по лабораторной работе. Отчёт оформляется на листах формата А4, он должен содержать:

- титульный лист;
- цель работы;
- текст индивидуального задания;
- логическую модель решения задачи;
- листинг разработанной программы;
- скриншоты с результатами выполнение программы на ПК;
- выводы о проделанной работе.

6 Защитить лабораторную работу. Защита включает в себя демонстрацию работы программы преподавателю и ответы на его вопросы по теме лабораторной работы в объёме методических рекомендаций. После защиты лабораторной работы преподаватель ставит на титульном листе свою подпись и дату. Только после этого лабораторная работа считается полностью выполненной, и студент может приступить к выполнению следующей работы.

2 Лабораторная работа № 1. Язык моделирования UML. Построение диаграмм классов

Цель работы

Получить навыки построения диаграмм классов на языке моделирования UML.

Теоретические сведения

Диаграммы классов [2, с. 280–295].

Индивидуальные задания

Для заданного варианта задания нарисовать UML-диаграмму классов.

1 Система Факультатив. Преподаватель объявляет запись на Курс. Студент записывается на Курс, обучается и по окончании Преподаватель выставля-

ет Оценку, которая сохраняется в Архиве Студентов. Преподавателей и Курсов при обучении может быть несколько.

2 Система Платежи. Клиент имеет Счет в банке и Кредитную Карту (КК). Клиент может оплатить Заказ, сделать платеж на другой Счет, заблокировать КК и аннулировать Счет. Администратор может заблокировать КК за превышение кредита.

3 Система Больница. Пациенту назначается лечащий Врач. Врач может сделать назначение Пациенту (процедуры, лекарства, операции). Медсестра или другой Врач выполняют назначение. Пациент может быть выписан из Больницы по окончании лечения, при нарушении режима или при иных обстоятельствах.

4 Система Вступительные экзамены. Абитуриент регистрируется на Факультет, сдает Экзамены. Преподаватель выставляет Оценку. Система подсчитывает средний балл и определяет Абитуриентов, зачисленных и учебное заведение.

5 Система Библиотека. Читатель оформляет Заказ на Книгу. Система осуществляет поиск в Каталоге. Библиотекарь выдает Читателю Книгу на абонемент или в читальный зал. При невозвращении Книги Читателем он может быть занесен Администратором в черный список.

6 Система Конструкторское бюро. Заказник представляет Техническое Задание (ТЗ) на проектирование многоэтажного Дома. Конструктор регистрирует ТЗ, определяет стоимость проектирования и строительства, выставляет Заказчику Счет за проектирование и создает Бригаду Конструкторов для выполнения Проекта.

7 Система Телефонная станция. Абонент оплачивает Счет за разговоры и Услуги, может попросить Администратора сменить номер и отказаться от услуг. Администратор изменяет номер, Услуги и временно отключает Абонента за неуплату.

8 Система Автобаза. Диспетчер распределяет заявки на Рейсы между Водителями и назначает для этого Автомобиль. Водитель может сделать заявку на ремонт. Диспетчер может отстранить Водителя от работы. Водитель делает отметку о выполнении Рейса и состоянии Автомобиля.

9 Система Интернет-магазин. Администратор добавляет информацию о Товаре. Клиент делает и оплачивает Заказ на Товары. Администратор регистрирует Продажу и может занести неплательщиков в «черный список».

10 Система Железнодорожная касса. Пассажир делает Заявку на станцию назначения, время и дату поездки. Система регистрирует Заявку и осуществляет поиск подходящего Поезда. Пассажир делает выбор Поезда и получает Счет на оплату. Администратор вводит номера Поездов, промежуточные и конечные станции, цены.

11 Система Городской транспорт. На Маршрут назначаются Автобус, Троллейбус или Трамвай. Транспортные средства должны двигаться с определенным для каждого Маршрута интервалом. При поломке на Маршрут должен выходить резервный транспорт или увеличиваться интервал движения.

12 Система Аэрофлот. Администратор формирует летную Бригаду (пилоты, штурман, радист, стюардессы) на Рейс. Каждый Рейс выполняется Самолетом с определенной вместимостью и дальностью полета. Рейс может быть отменен из-за погодных условий в Аэропорту отлета или назначения. Аэропорт назначения может быть изменен в полете из-за технических неисправностей, о которых сообщил командир.

13 Система Периодические издания. Читатель может сделать Заявку, предварительно выбрав периодические издания из списка. Система подсчитывает сумму для оплаты. Читатель оплачивает заявку. Администратор добавляет Заявку в «черный список», если Клиент не оплачивает её в определённый срок.

14 Система Заказ гостиницы. Клиент оставляет Заявку на Номер, указав количество мест в номере, класс апартаментов и время пребывания. Администратор рассматривает Заявку, подтверждает или отклоняет ее. Результат просматривает Клиент. В случае подтверждения Заявки Клиент оплачивает услуги.

15 Система Жилищно-коммунальные услуги. Квартиросъемщик отправляет Заявку, в которой указывает род работ, масштаб и желаемое время выполнения. Диспетчер формирует соответствующую Бригаду и регистрирует её в Плане работ. Диспетчер может отклонить Заявку в случае занятости всех Бригад.

16 Система Прокат автомобилей. Клиент выбирает Автомобиль из списка доступных, заполняет форму Заказа, указывая паспортные данные, срок аренды. Администратор может отклонить Заявку, указав причины отказа. При подтверждении Заявки Клиент оплачивает Заказ. Система выписывает сумму. В случае повреждения Автомобиля Клиентом Администратор вносит соответствующие пометки.

Контрольные вопросы

1 Для чего предназначена диаграмма классов?

2 Перечислите основные графические символы, используемые при построении диаграммы классов.

3 Как на диаграмме классов представляются отношения наследования и агрегации?

3 Лабораторная работа № 2. Язык моделирования UML. Построение диаграмм взаимодействия

Цель работы

Получить навыки построения диаграмм взаимодействия объектов классов на языке моделирования UML.

Теоретические сведения

Диаграммы последовательности [2, с. 260–277].

Индивидуальные задания

Нарисовать диаграмму взаимодействия объектов программной системы, реализованной при выполнении лабораторной работы № 1.

Контрольные вопросы

1 Для чего предназначена диаграмма взаимодействия?
2 Объясните последовательность построения диаграммы взаимодействия объектов программной системы.
3 На примере построенной диаграммы последовательностей расскажите порядок объектов программы.

4 Лабораторная работа № 3. Разработка программ с использованием принципа единственной обязанности (SRP) и принципа открытости/закрытости (OCP)

Цель работы

Получить навыки разработки программ с использованием принципа единственной обязанности (SRP) и принципа открытости/закрытости (OCP).

Теоретические сведения

Принцип единственной обязанности [2, с. 153–158].
Принципа открытости/закрытости [2, с. 160–171].

Индивидуальные задания

Разработать программу решения задачи из лабораторной работы № 1 в виде консольного приложения с использованием принципа единственной обязан-

ности (SRP) и принципа открытости/закрытости (OCP).

Контрольные вопросы

1 Объясните назначение принципов объектно-ориентированного проектирования SOLID.

2 Перечислите принципы, входящие в SOLID.

3 Сформулируйте принцип единственной обязанности. Объясните основную идею, лежащую в основе этого принципа, приведите пример использования.

4 Сформулируйте принцип открытости/закрытости. Объясняете основную идею, лежащую в основе этого принципа, приведите пример использования.

5 Лабораторная работа № 4. Разработка программ с использованием принципа подстановки Лисков (LSP), принципа разделения интерфейсов (ISP) и принципа инверсии (DIP)

Цель работы

Получить навыки разработки программ с использованием принципа подстановки Лисков (LSP), принципа разделения интерфейсов (ISP) и принципа инверсии (DIP).

Теоретические сведения

Принцип подстановки Лисков [2, с. 173–189].
Принцип разделения интерфейсов [2, с. 200–212].
Принцип инверсии зависимостей [2, с. 191–199].

Индивидуальные задания

Разработать программу решения задачи в соответствии с выданным вариантом в виде консольного приложения с использованием принципа подстановки Лисков (LSP), принципа разделения интерфейсов (ISP) и принципа инверсии (DIP).

Варианты заданий

1 Студент, преподаватель, заведующий кафедрой, персона.
2 Небоскрёб, дача, коттедж, жилое здание.
3 Организация, страховая компания, нефтегазовая компания, завод.
4 Журнал, книга, печатное издание, учебник.
5 Тест, экзамен, выпускной экзамен, испытание.

6 Строительное сооружение, театр, производственный корпус, гостиница.
7 Игрушка, телевизор, товар, молоко.
8 Квитанция, накладная, документ, счёт.
9 Автомобиль, поезд, самолёт, транспортное средство.
10 Республика, монархия, королевство, государство.
11 Корабль, пароход, парусник, корвет.
12 Двигатель, бензиновый двигатель, дизельный двигатель, реактивный двигатель.
13 Деталь, узел, механизм, изделие.
14 Млекопитающее, парнокопытное, птица, животное.
15 Выпускник вуза, бакалавр, магистр, инженер.

Контрольные вопросы

1 Сформулируйте принцип подстановки Лисков. Объясните основную идею, лежащую в основе этого принципа, приведите пример использования.
2 Сформулируйте принцип разделения интерфейсов. Объясните основную идею, лежащую в основе этого принципа, приведите пример использования.
3 Сформулируйте принцип инверсии. Объясните основную идею, лежащую в основе этого принципа, приведите пример использования.

6 Лабораторная работа № 5. Разработка программ с использованием порождающих паттернов

Цель работы

Научиться разрабатывать программы с использованием структурных шаблонов проектирования из каталога GoF: Строитель, Абстрактная фабрика, Фабричный метод.

Теоретические сведения

Абстрактная фабрика – порождающий шаблон проектирования, позволяющий изменять поведение системы, варьируя создаваемые объекты, при этом сохраняя интерфейсы.

Он позволяет создавать целые группы взаимосвязанных объектов, которые, будучи созданными одной фабрикой, реализуют общее поведение.

Шаблон реализуется созданием абстрактного класса `Factory`, который представляет собой интерфейс для создания компонентов системы (например, для оконного интерфейса он может создавать окна и кнопки). Затем пишутся классы, реализующие этот интерфейс. Этот шаблон предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретные классы.

Как и в реальной жизни, фабрика имеет некую специализацию, создавая товары или устройства какого-либо определенного типа.

Фабрика, которая выпускает, например, мебель, не может производить, например, еще и компоненты для смартфонов.

В программировании фабрика объектов может создавать только объекты определенного типа, которые используют единый интерфейс.

Самыми главными преимуществами данного паттерна в C#, является упрощение создания объектов различных классов, использующих единый интерфейс.

Паттерн предоставляет интерфейс для создания семейств, связанных между собой, или независимых объектов, конкретные классы которых неизвестны.

От класса «абстрактная фабрика» наследуются классы конкретных фабрик, которые содержат методы создания конкретных объектов-продуктов, являющихся наследниками класса «абстрактный продукт», объявляющего интерфейс для их создания.

Клиент пользуется только интерфейсами, заданными в классах «абстрактная фабрика» и «абстрактный продукт» (рисунок 6.1).

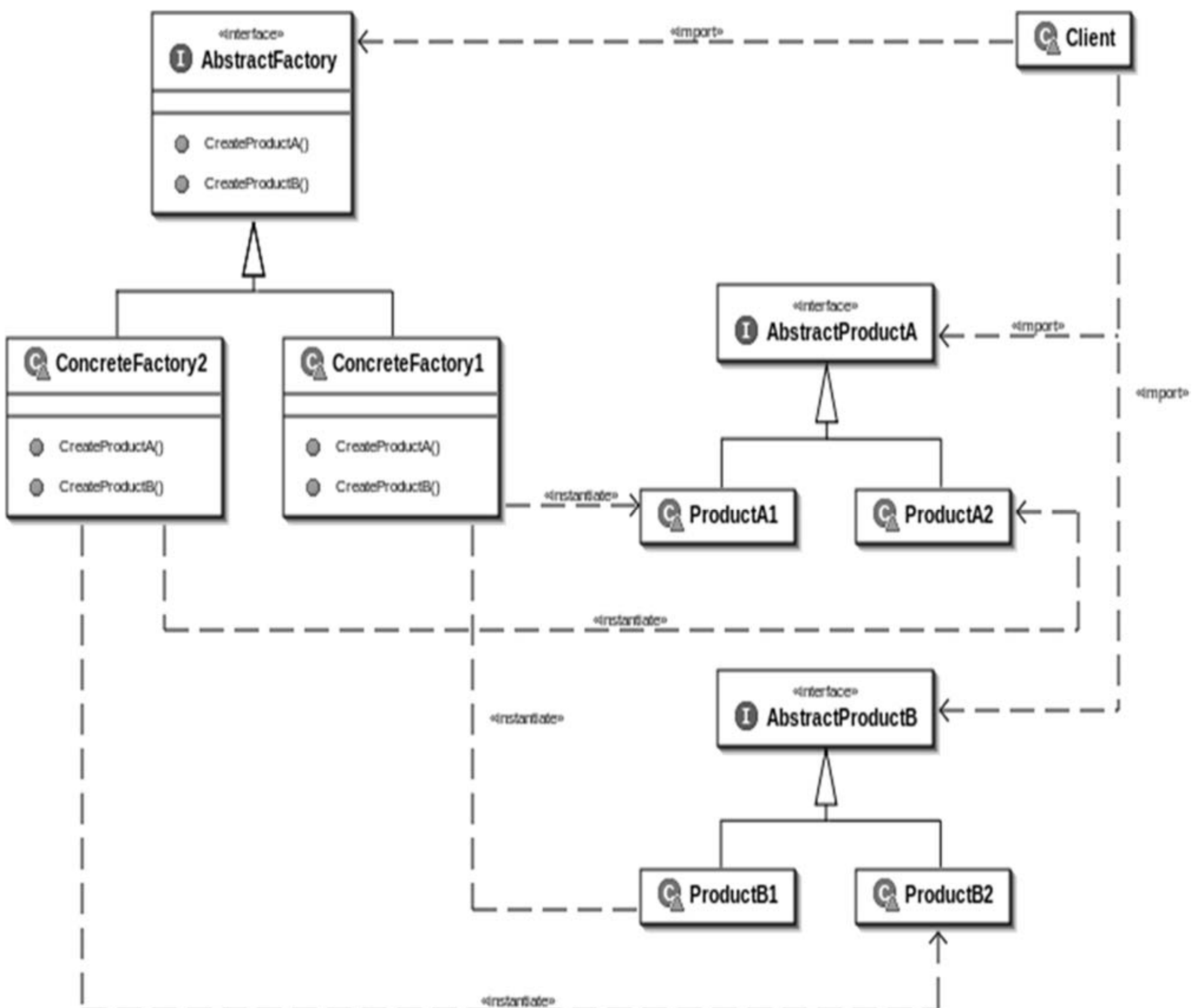

Рисунок 6.1 – UML-диаграмма шаблона Абстрактный продукт

Фабричный метод – порождающий шаблон проектирования, предоставляющий подклассам интерфейс для создания экземпляров некоторого класса.

В момент создания наследники могут определить, какой класс создавать.

Иными словами, Фабрика делегирует создание объектов наследникам родительского класса.

Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

Шаблон определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать (рисунок 6.2).

Фабричный метод позволяет классу делегировать создание подклассов.

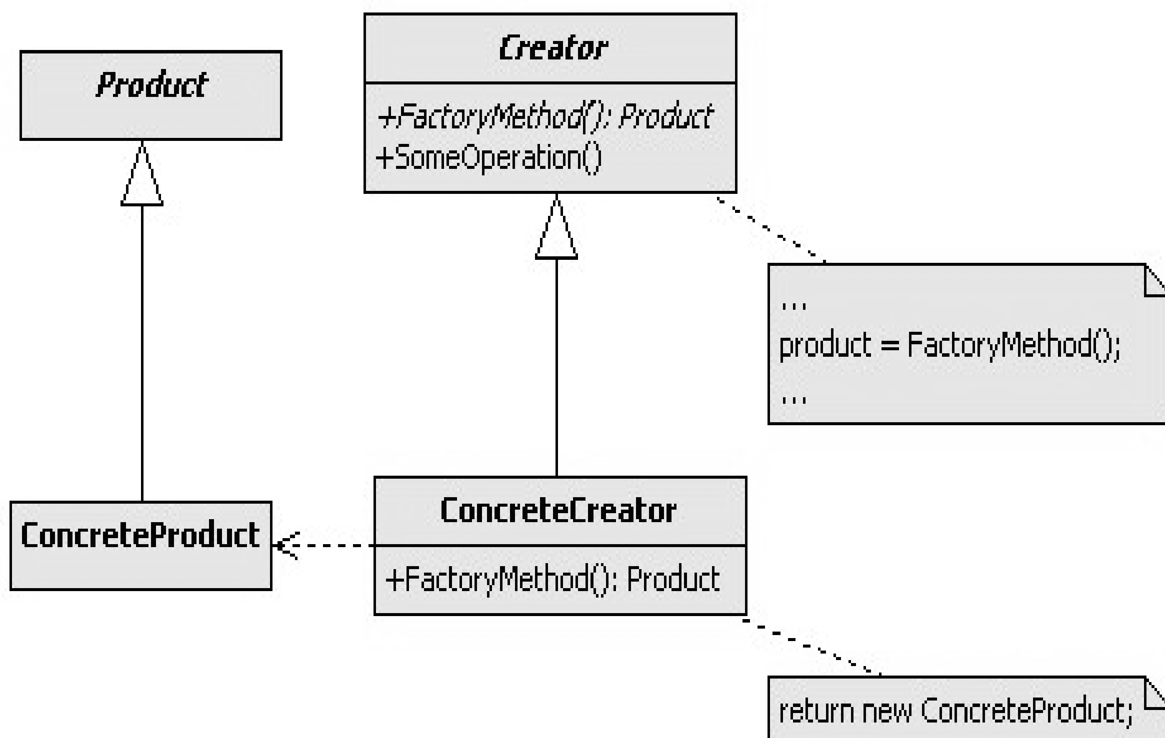

Рисунок 6.2 – UML-диаграмма шаблона Фабричный метод

Используется, когда:

– классу заранее неизвестно, объекты каких подклассов ему нужно создавать;

– класс спроектирован так, чтобы объекты, которые он создаёт, специфицировались подклассами;

– класс делегирует свои обязанности одному из нескольких вспомогательных подклассов, и планируется локализовать знание о том, какой класс принимает эти обязанности на себя.

Участники паттерна:

– `Product` – продукт; определяет интерфейс объектов, создаваемых абстрактным методом;

– `ConcreteProduct` – конкретный продукт, реализует интерфейс `Product`;

– `Creator` – создатель, объявляет фабричный метод, который возвращает объект типа `Product`. Может также содержать реализацию этого метода «по умолчанию»; может вызывать фабричный метод для создания объекта типа `Product`;

– `ConcreteCreator` – конкретный создатель; переопределяет фабричный метод таким образом, чтобы он создавал и возвращал объект класса `ConcreteProduct`.

Строитель (Builder) – шаблон проектирования, который инкапсулирует создание объекта и позволяет разделить его на различные этапы.

Паттерн Строитель рекомендуется использовать в следующих случаях:

– когда процесс создания нового объекта не должен зависеть от того, из каких частей этот объект состоит и как эти части связаны между собой;

– когда необходимо обеспечить получение различных вариаций объекта в процессе его создания.

UML-диаграмма паттерна показана на рисунке 6.3.

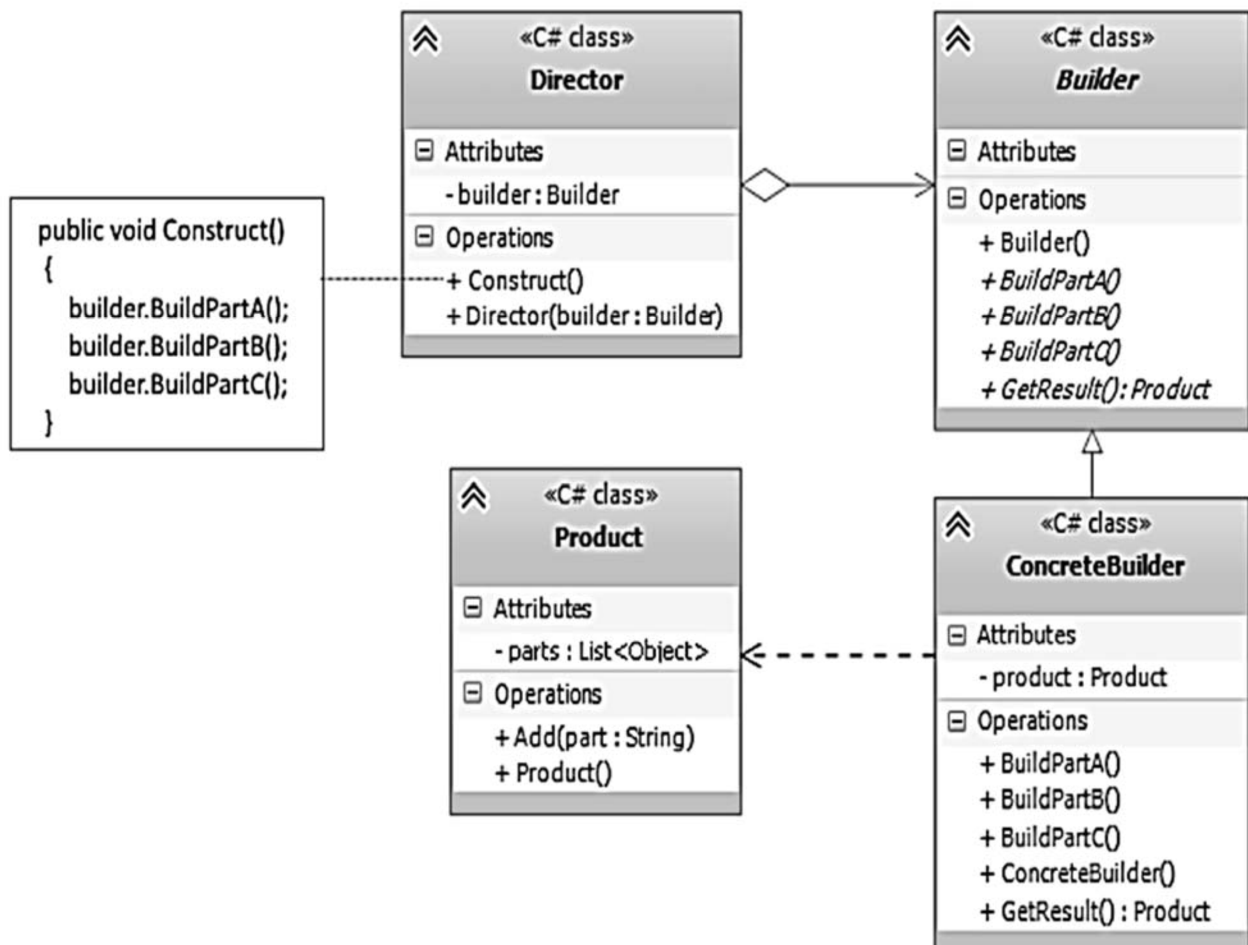

Рисунок 6.3 – UML-диаграмма шаблона Строитель

Участники паттерна:

– `Product`: представляет объект, который должен быть создан. В данном случае все части объекта заключены в списке `parts`;

– `Builder`: определяет интерфейс для создания различных частей объекта `Product`;

– `ConcreteBuilder`: конкретная реализация `Buildera`. Создает объект `Product` и определяет интерфейс для доступа к нему;

– `Director`: распорядитель – создает объект, используя объекты `Builder`.

Индивидуальные задания

Для указанного варианта задания необходимо нарисовать UML-диаграмму классов реализуемой программы и разработать консольное приложение на языке C#.

1 Паттерн `Builder`. Имеется текст статьи в формате TXT. Статья состоит из заголовка, фамилий авторов, самого текста статьи и хеш-кода текста статьи. Написать приложение, позволяющее конвертировать документ в формате TXT в документ формата XML, необходимо также проверять корректность хеш-кода статьи.

2 Паттерн `Abstract Factory`. Разработать систему Кинопрокат. Пользователь может выбрать определённую киноленту, при заказе киноленты указывается язык звуковой дорожки, который совпадает с языком файла субтитров. Система должна поставлять фильм с требуемыми характеристиками, причём при смене языка звуковой дорожки должен меняться и язык файла субтитров и наоборот.

3 Паттерн `Factory Method`. Фигуры игры «тетрис». Реализовать процесс случайного выбора фигуры из конечного набора фигур. Предусмотреть появление супер-фигур с большим числом клеток, чем обычные.

4 Паттерн `Builder`. Разработать модель системы Автомобиль. Написать код приложения, который будет позволять порождать как серийные автомобили, так и автомобили по специальному заказу. Использовать шаблон.

5 Паттерн `Builder`. Разработать модель системы Музыкальный коллектив. Написать код приложения, позволяющий создавать певческие, танцевальные и смешанные коллективы.

6 Паттерн `Builder`. Разработать модель системы Комплексный обед. Написать код приложения, позволяющий создавать как стандартные комплексные обеды, так и обеды, в которые включены дополнительные блюда из меню.

7 Паттерн `Builder`. Имеется текст статьи в формате TXT. Статья состоит из заголовка, фамилий авторов, самого текста статьи и хеш-кода текста статьи. Написать приложение, позволяющее конвертировать документ в формате TXT в документ формата XML, необходимо также проверять корректность хэш-кода статьи.

8 Паттерн `Abstract Factory`. Написать код приложения, позволяющий универсально записывать данные о совершенном телефонном звонке в базу данных и xml-файл, а также считывать эту информацию. Использовать также шаблон DAO.

9 Паттерн `Abstract Factory`. Написать код приложения, позволяющий сохранять регистрационные данные пользователя в базе данных. Состав регистрационных данных у каждого пользователя может быть различен. Использовать также шаблон DAO.

10 Паттерн `Abstract Factory`. Разработать систему Кинопрокат. Пользователь может выбрать определенную киноленту, при заказе киноленты указывается язык звуковой дорожки, который совпадает с языком файла субтитров. Система должна поставлять фильм с требуемыми.

11 Паттерн `Factory Method`. Фигуры игры «тетрис». Реализовать процесс случайного выбора фигуры из конечного набора фигур. Предусмотреть появление супер-фигур с большим числом клеток, чем обычные.

12 Паттерн `Abstract Factory`. Проект «Заводы по производству автомобилей». В проекте должно быть реализована возможность создавать автомобили различных типов на разных заводах.

13 Паттерн `Factory Method`. Проект «Фабрика смартфонов». В проекте должно быть реализовано создание смартфонов с различными характеристиками. Пример использования шаблона см. в разделе 4.

Контрольные вопросы

1 Каково назначение шаблона Строитель?
2 Какая проблема решается с помощью шаблона Строитель?
3 В чём заключается решение, предлагаемое в шаблоне Строитель?
4 Для чего предназначен шаблон Абстрактная фабрика?
5 Какое решение предлагается в шаблоне Абстрактная фабрика?
6 Какую проблему позволяет решить шаблон Фабричный метод?

7 Лабораторная работа № 6. Разработка программ с использованием структурных паттернов

Цель работы

Научиться разрабатывать программы с использованием таких структурных шаблонов проектирования из каталога GoF, как Адаптер и Фасад.

Теоретические сведения

Шаблон Адаптер.

Адаптер – структурный шаблон проектирования, предназначенный для преобразования интерфейса класса к другому интерфейсу, на который рассчитан клиент.

Проблема. Как обеспечить взаимодействие несовместимых интерфейсов или как создать единый устойчивый интерфейс для нескольких классов с раз-

ными интерфейсами?

Решение. Преобразовать исходный интерфейс класса к другому виду с помощью промежуточного объекта-адаптера.

Основными участниками решения являются:

– `Client` – клиент, использующий целевой интерфейс;

– `ITarget` – целевой интерфейс;

– `Adapter` – адаптер, реализующий интерфейс `ITarget`;

– `Adapted` – адаптируемый класс, имеющий интерфейс несовместимый с интерфейсом клиента.

Работа клиента с адаптируемым объектом происходит следующим образом:

– клиент обращается с запросом к объекту-адаптеру `Adapter`, вызывая его метод `Request()` через целевой интерфейс `ITarget`;

– адаптер `Adapter` преобразует запрос в один или несколько вызовов к адаптируемому объекту `Adapted`;

– клиент получает результаты вызова, не зная ничего о преобразованиях, выполненных адаптером.

Шаблон Адаптер имеет следующие разновидности:

– адаптер объекта применяет для адаптации одного интерфейса к другому композицию объектов адаптируемого класса (рисунок 7.1);

– адаптер класса использует для адаптации наследование адаптируемому классу (рисунок 7.2).

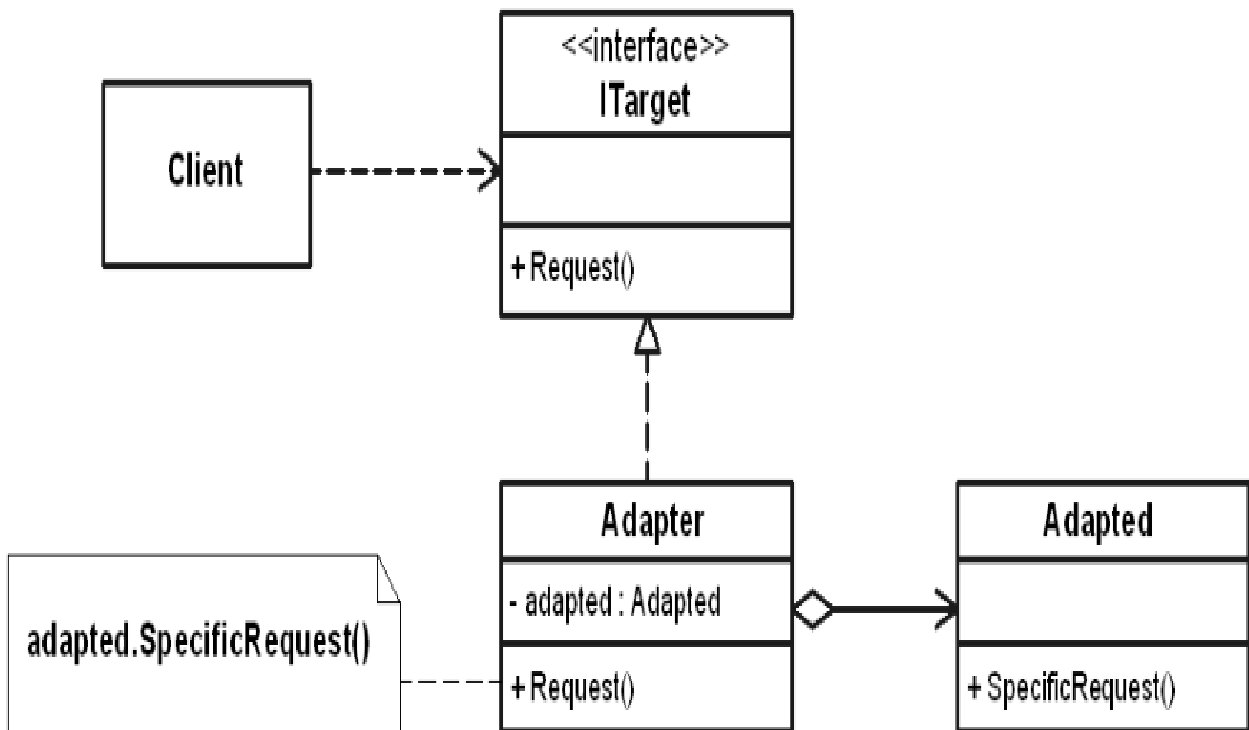

Рисунок 7.1 – Диаграмма классов шаблона Адаптер объекта

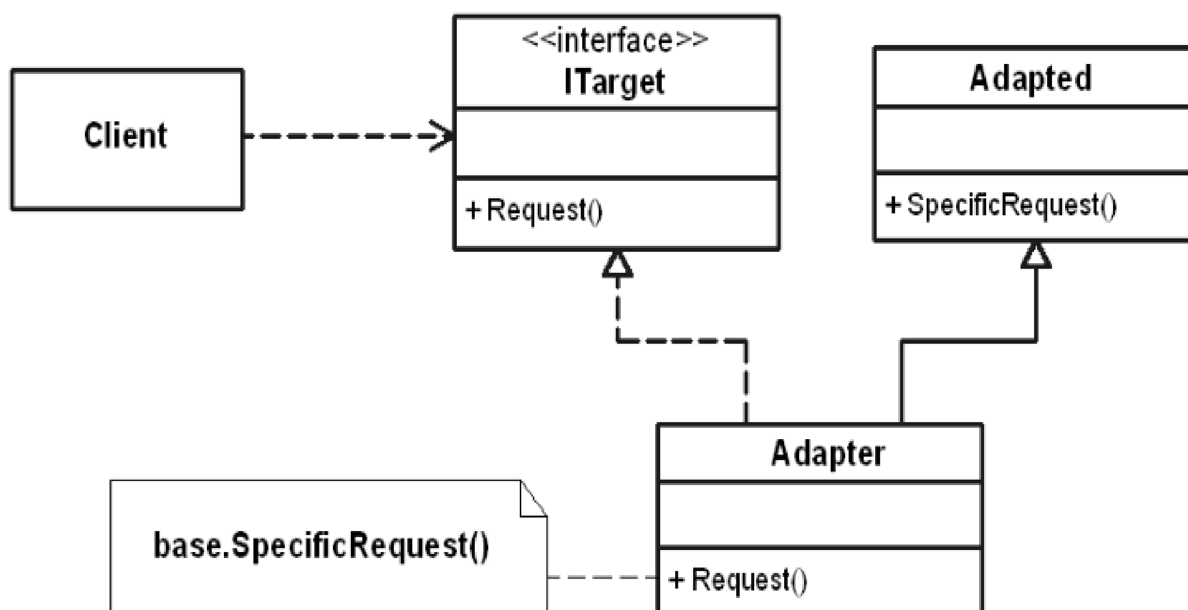

Рисунок 7.2 – Диаграмма классов шаблона Адаптер класса

Результаты. Система становится независимой от интерфейса внешних классов (компонентов, библиотек). При переходе на использование внешних классов не требуется переделывать всю систему, достаточно переделать один класс `Adapter`.

Результаты применения адаптеров классов и объектов различаются.

Адаптер объекта:

– позволяет работать объекту `Adapter` со многими адаптируемыми объектами (например, с объектами класса `Adapted` и его производных классов);

– отличается сложностью при замещении операций класса `Adapted` (для этого может потребоваться создать класс производный от `Adapted` и добавить в класс `Adapter` ссылку на этот производный класс).

Адаптер классов:

– обеспечивает простой доступ к элементам адаптируемого класса, поскольку Adapter является производными классом от `Adapted`;

– характеризуется легкостью изменения адаптером операций адаптируемого класса `Adapted`;

– обладает возможностью работы только с одним адаптируемым классом (возможность адаптировать классы, производные от `Adaptee` отсутствует).

Отличие реализации шаблона Адаптер класса будет заключаться только в коде класса Adapter.

Шаблон Фасад.

Фасад – структурный шаблон проектирования, позволяющий скрыть сложность подсистемы путем сведения всех возможных вызовов к одному объекту (фасадному объекту), делегирующему их объектам подсистемы (рисунок 7.3).

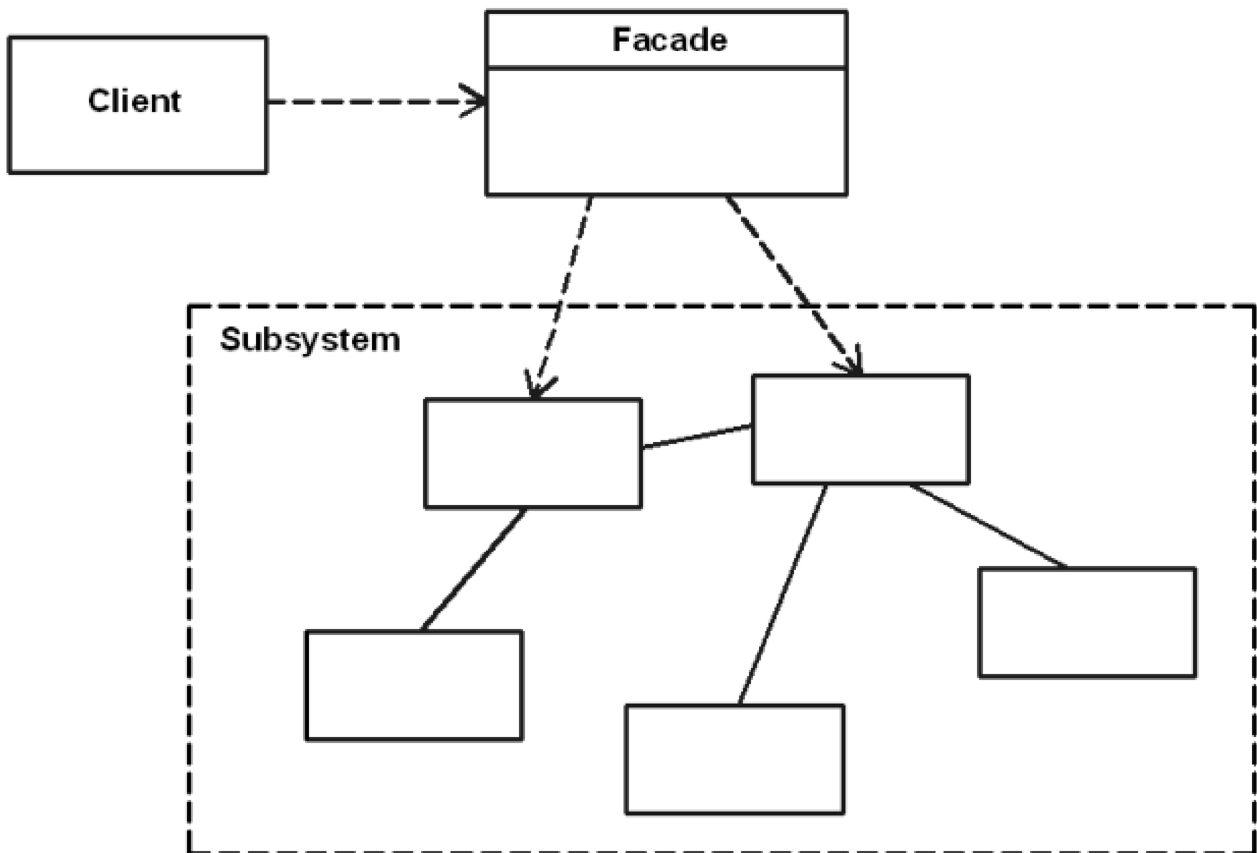

Рисунок 7.3 – Диаграмма классов шаблона Фасад

Проблема. Как обеспечить унифицированный интерфейс с подсистемой, если нежелательна высокая степень связанности с этой подсистемой или реализация подсистемы может измениться?

Решение. Определить одну точку взаимодействия с подсистемой – фасадный объект, обеспечивающий единый упрощенный интерфейс с подсистемой и возложить на него обязанность по взаимодействию с классами подсистемы.

Участники решения:

– `Client` – взаимодействует с фасадом и не имеет доступа к классам подсистемы;

– `Facade` – перенаправляет запросы клиентов к классам подсистемы;

– Классы подсистемы – выполняют работу, порученную объектом `Facade`, ничего не зная о существовании фасада, то есть не хранят ссылок на него.

Результаты. Клиенты изолируются от классов (компонентов) подсистемы, что уменьшает число объектов, с которыми клиенты взаимодействуют, и упрощает работу с подсистемой; снижается степень связанности между клиентами и подсистемой, что позволяет изменять классы подсистемы, не затрагивая при этом клиентов.

Индивидуальные задания

1 Изучить пример проектирования программной системы с использованием паттерна Адаптер [3, с. 117–123].

2 Изучить пример проектирования программной системы с использованием паттерна Фасад [3, с. 123–133].

3 Разработать библиотеку классов, которая содержит указанные классы (таблица 7.1), задействованные в шаблоне Адаптер. Для адаптера объектов атрибуты класса должны быть реализованы как автоматические свойства, а для шаблона Адаптер классов – как защищённые поля.

4 Разработать библиотеку классов, которая должна содержать класс-фасад и заданный набор классов (таблица 7.2). В фасаде необходимо задать ссылки на другие классы библиотеки.

5 Добавить в решение консольное приложение, которое для реализованных шаблонов играет роль клиента. Продемонстрировать в консольном приложении работу шаблонов проектирования.

Таблица 7.1 – Варианты заданий для разработки приложения с использованием шаблона Адаптер

<table>
<tr><th>Номер варианта</th><th>Тип адаптера</th><th>Требуемый интерфейс</th><th>Адаптируемый класс</th></tr>
<tr><td>1</td><td>Объектов</td><td>+ CalculateDp(T0 : int, dT : int) : double – определить изменение давления при заданной начальной температуре T0 и изменении температуры dT;
+ ModifMass(dm : double): void – изменить массу газа в баллоне на величину dm;
+ GetData() : string – возвращает строку с данными об объекте</td><td rowspan="2">Баллон с газом.
Атрибуты:
Volume : double – объём баллона, м^3;
Mass : double – масса газа, кг;
Molar : double – молярная масса газа, кг/моль.
Операции:
+ GetPressure(T : int) : double – определить давление в баллоне при заданной температуре газа T;
+ AmountOfMatter() : double – определить количество вещества;
+ ToString() : string – возвращает строку с данными об объекте</td></tr>
<tr><td>2</td><td>Классов</td><td>+ ModifVolume(dV : double) : void – изменить объём баллона на величину dV;
+ GetDp(T0 : int, T1 : int) : double – определить изменение давления при изменении температуры с T0 до T1;
+Passport() : string – возвращает строку с данными об объекте</td></tr>
</table>

Таблица 7.2 – Варианты заданий для разработки приложения с использованием шаблона Фасад

Номер варианта	Данные для разработки приложения на основе шаблона Фасад
1	*Расчёт страхового взноса за недвижимость*. Классы (типы недвижимости): квартира, таун-хаус, коттедж. Параметры: срок страхования, жилплощадь, м^2, число проживающих, год постройки здания, износ здания, %
2	*Расчёт ежедневной нормы потребления килокалорий*. Классы (Тип телосложения): астеник, нормостеник, гиперстеник. Параметры: рост, вес, возраст, пол, группа физической активности (низкая, средняя и высокая активность)
3	*Расчёт стоимости туристической путевки*. Классы (виды путевок): пляжный отдых, экскурсия, горные лыжи. Параметры: длительность, страна, гостиница (число звезд), рацион питания (двухразовый, трехразовый, всё включено)

Контрольные вопросы

1 Какие шаблоны проектирования называют структурными?

2 Каково назначение структурного шаблона Адаптер?

3 Каким образом осуществляется взаимодействие клиента с адаптируемым классом в шаблоне Адаптер?

4 В чём заключается различие между Адаптером объектов и Адаптером классов?

5 Какая проблема решается с помощью шаблона Фасад?

6 В чём заключается решение, предлагаемое в шаблоне Фасад?

7 К каким последствиям приводит использование шаблона Фасад?

8 Лабораторная работа № 7. Разработка программ с использованием паттернов поведения

Цель работы

Научиться разрабатывать программы с использованием таких поведенческих шаблонов проектирования, как Стратегия, Состояние и Шаблонный метод.

Теоретические сведения

Поведенческие шаблоны проектирования. Диаграммы конечных автоматов UML.

Поведенческие шаблоны (англ. Behavioral Patterns) предназначены для определения алгоритмов и способов реализации взаимодействия различных объектов и классов.

Одним из средств описания поведения программных объектов в языке UML являются диаграммы конечных автоматов (диаграммы состояний).

Диаграммы конечных автоматов (англ. state machine diagrams) UML отображают жизненный цикл объекта с помощью состояний, событий и переходов.

Основными элементами диаграмм конечных автоматов являются:

– состояние – это ситуация во время жизни объекта, в которой он удовлетворяет определённым условиям, выполняет определённую деятельность или находится в ожидании событий. Состояния изображаются на диаграмме в виде прямоугольников со скруглёнными углами. Специальными видами состояний являются начальное и конечное состояния, изображаемые соответственно символами ● и ◉;

– переход – это отношение между двумя состояниями, указывающее на то, что объект из первого состояния перейдёт во второе, при выполнении определённого условия. Переходы на диаграммах конечных автоматов изображают стрелками, ведущими от одного состояния к другому;

– событие – это значимое или заслуживающее внимания происшествие, которое может инициализировать переход объекта от одного состояния к другому. Событием может являться поступление сигнала, выполнение какого-либо условия, истечение определённого периода времени. События разделяют на внешние, внутренние и временные.

Пример простой диаграммы конечных автоматов для объекта «входная дверь» показан на рисунке 8.1.

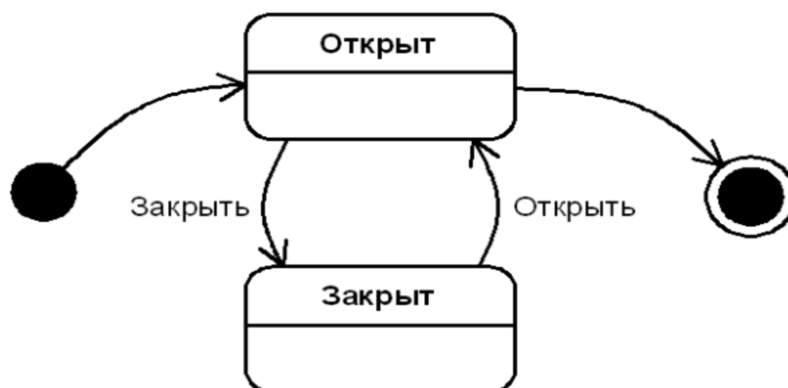

Рисунок 8.1 – Простая диаграмма конечных автоматов UML

Шаблон Состояние.

Шаблон Состояние (англ. `State`) управляет изменением поведения объекта в зависимости от его внутреннего состояния.

Проблема. Как изменять поведение объекта в зависимости от его внутреннего состояния?

Решение. Определить для каждого состояния отдельный класс со стандартным интерфейсом.

Структура. Диаграмма классов шаблона Состояние представлена на рисунке 8.2.

Участники шаблона:

– `State` – абстрактный класс, определяющий общий интерфейс для всех

конкретных состояний;

– `StateMachine` – класс с несколькими внутренними состояниями, хранит экземпляр класса State;

– `ConcreteStateA`, `ConcreteStateB` – классы конкретных состояний, обрабатывающие запросы от класса `StateMachine`; каждый класс предоставляет собственную реализацию обработки запроса.

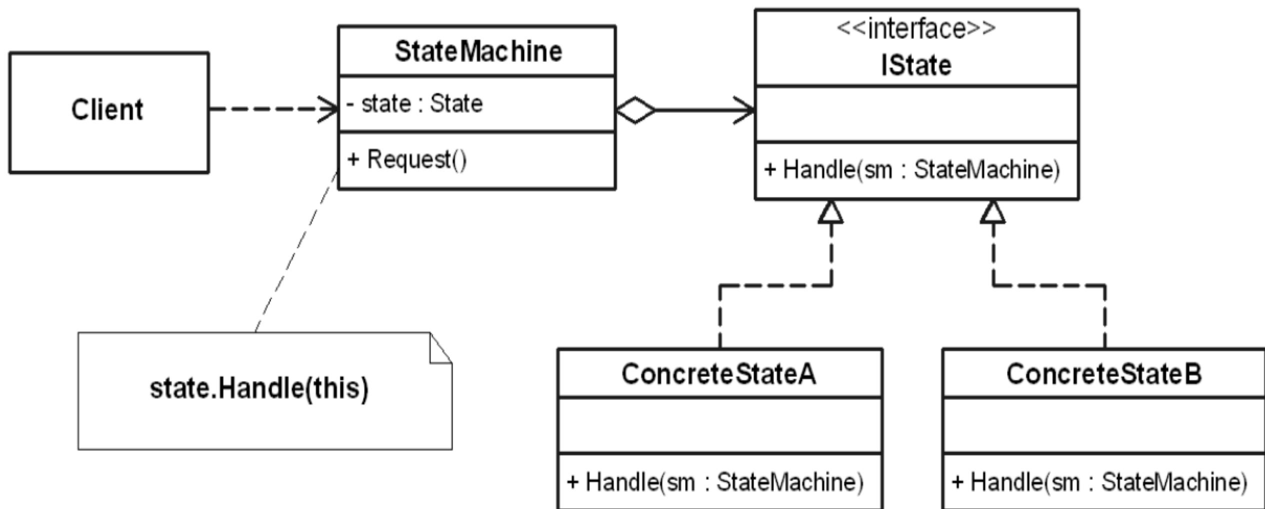

Рисунок 8.2 – Диаграмма классов шаблона Состояние

Шаблон Шаблонный метод.

Шаблон Стратегия (англ. `Strategy` или `Policy`) позволяет менять алгоритм независимо от клиентов, которые его используют.

Проблема. Как спроектировать изменяемые, но надёжные алгоритмы (стратегии)?

Решение. Определить для каждого алгоритма (стратегии) отдельный класс со стандартным интерфейсом.

Структура. Диаграмма классов шаблона Стратегия показана на рисунке 8.3.

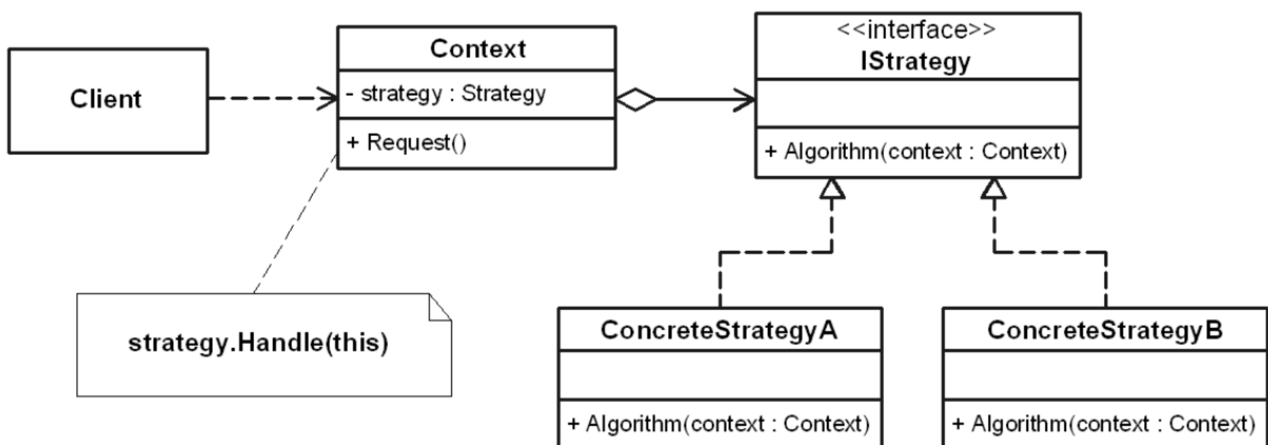

Рисунок 8.3 – Диаграмма классов шаблона Стратегия

Участники шаблона:

– `Strategy` – класс, определяющий общий для всех поддерживаемых алгоритмов интерфейс;

– `Context` – класс, хранящий ссылку на экземпляр класса Стратегия;

– `ConcreteStrategyA`, `ConcreteStrategyB` – классы, представляющие конкретные стратегии, использующие интерфейс класса `Strategy` для реализации алгоритма.

Шаблонный метод.

Шаблон «Шаблонный метод» (англ. `Template Method`) определяет основу алгоритма (шаблонный метод) в базовом классе и позволяет производным классам переопределять отдельные шаги алгоритма, не изменяя его структуру в целом.

Проблема. Как определить алгоритм и реализовать возможность переопределения некоторых шагов алгоритма в производных классах без изменения общей структуры алгоритма?

Решение. Создать абстрактный класс, определяющий следующие операции:

– шаблонный метод, который задаёт основу алгоритма;

– абстрактные операции, которые замещаются в производных классах для реализации шагов алгоритма.

Структура. Диаграмма классов шаблона Шаблонный метод показана на рисунке 8.4.

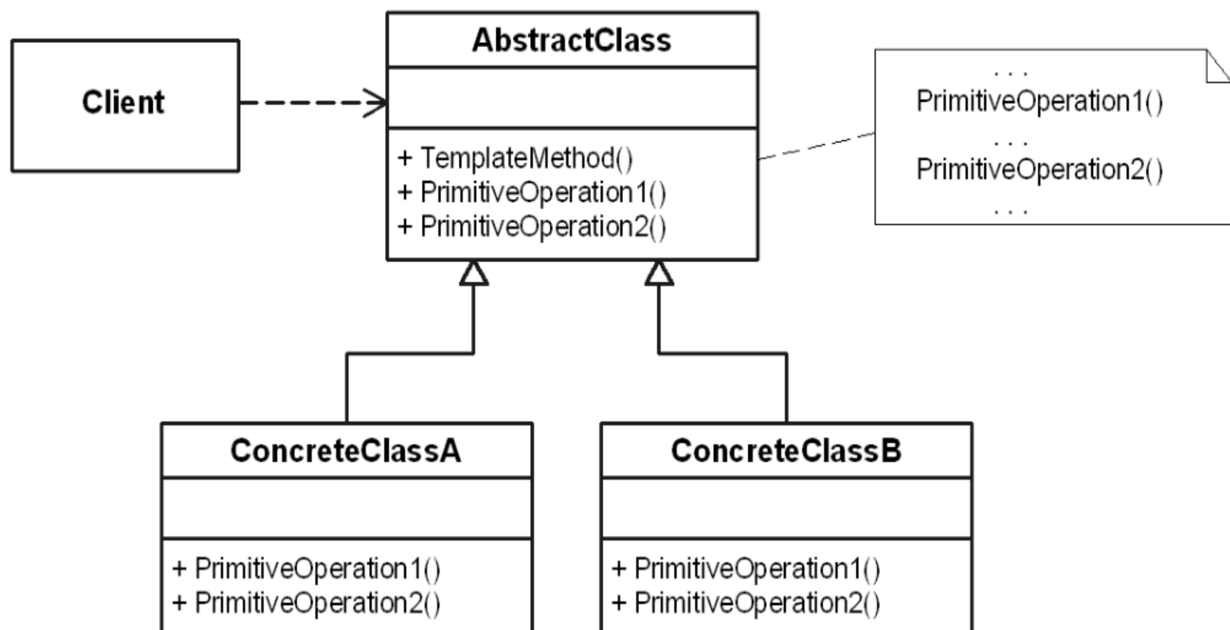

Рисунок 8.4 – Диаграмма классов шаблона Шаблонный метод

Участники шаблона:

– `AbstractClass` определяет абстрактные частные операции и содержит шаблонный метод, который вызывает эти операции;

– `ConcreteClassA`, `ConcreteClassB` реализуют частные операции, вызываемые шаблонным методом.

Индивидуальные задания

1 Изучить пример проектирования программной системы с использованием паттерна Состояние [3, с. 143–154].

2 Разработать диаграмму конечных автоматов для заданного класса (таблица 8.1). Описать в форме таблицы варианты реакции экземпляра класса на операции, вызываемые в указанных состояниях.

3 Разработать библиотеку классов, включающую необходимые классы для реализации шаблона Состояние (класс Конечный автомат, интерфейс Состояние, классы Конкретные состояния).

4 Разработать приложение Windows Forms для управления состояниями экземпляров класса Конечный автомат.

Таблица 8.1 – Варианты заданий для разработки приложения с использованием шаблона Состояние

Номер варианта	Классы, их атрибуты и операции	Состояния
1	*Телефон*. *Атрибуты*: номер, баланс, вероятность поступления звонка. *Операции*: позвонить, ответить на звонок, завершить разговор, пополнить баланс	Ожидание, Звонок, Разговор, Заблокирован (баланс отрицательный)
2	*Банкомат*. *Атрибуты*: ID, общая сумма денег в банкомате, вероятность отсутствия связи с банком. *Операции*: ввести PIN-код, внять заданную сумму, завершить работу, загрузить деньги в банкомат	Ожидание, Аутентификация пользователя, Выполнение операций, Заблокирован (денег нет)
3	*Грузовой лифт*. *Атрибуты:* текущий этаж, грузоподъёмность, вероятность отключения электроэнергии. *Операции*: вызвать на заданный этаж, загрузить, разгрузить, восстановить подачу энергии	Покой, Движение, Перегружен, Нет питания

Контрольные вопросы

1 Каково назначение поведенческих шаблонов проектирования?

2 Какие основные элементы используются на диаграммах конечных автоматов UML?

3 Для чего предназначен поведенческий шаблон Состояние?

4 Какое решение предлагается в шаблоне Состояние?

5 Что понимают под стратегией в шаблоне Стратегия?

6 Какое решение предлагается в шаблоне Стратегия?

7 Какую проблему позволяет решить шаблон Шаблонный метод?

9 Лабораторная работа № 8. Разработка программ с использованием шаблонов GRASP

Цель работы

Получить навыки разработки программ с использованием шаблонов GRASP.

Теоретические сведения

В разрабатываемой программной системе могут быть определены десятки и сотни различных классов, на которые могут быть возложены сотни и тысячи обязанностей.

Во время объектно-ориентированного проектирования при определении принципов взаимодействия объектов необходимо распределить обязанности между классами. При правильном выполнении этой задачи система становится гораздо проще для понимания, поддержки и расширения. Кроме того, появляется возможность повторного использования уже разработанных компонентов в последующих приложениях.

Проектирование на основе обязанностей (англ. `Responsibility-Driven Design, RDD`) – это общий подход к проектированию классов, в котором считается, что классы программной системы имеют определённые обязанности и должны взаимодействовать с другими классам для их выполнения.

В общем случае выделяют два типа обязанностей: знание `(knowing)` и действие `(doing)`.

Обязанности, относящиеся к знаниям объекта (вытекают из модели предметной области):

– наличие информации о закрытых инкапсулированных данных;
– наличие информации о связанных объектах;
– наличие информации о следствиях или вычисляемых величинах.

Обязанности, относящиеся к действиям объекта:

– выполнение некоторых действий самим объектом (например, создание экземпляра или выполнение вычислений);
– инициирование действий других объектов;
– управление действиями других объектов и их координирование.

Выявить и описать основные принципы распределения обязанностей, положенные в основу подхода `RDD`, позволяют общие шаблоны распределения обязанностей `(`англ. `General Responsibility Assignment Software Patterns, GRASP).`

К основным шаблонам `GRASP` относятся:

– `Information Expert` – информационный эксперт;
– `Creator` – создатель экземпляров класса;
– `Low Coupling` – слабая связанность;
– `High Cohesion` – сильное сцепление.

Шаблоны Information Expert и Creator.

Шаблон Information Expert является наиболее общим шаблоном GRASP и при распределении обязанностей используется гораздо чаще других шаблонов.

Проблема. Каков наиболее общий принцип распределения обязанностей между классами при объектно-ориентированном проектировании?

Решение. Назначить обязанность информационному эксперту – классу, который обладает достаточной информацией для выполнения этой обязанности.

Результаты. Поддержка инкапсуляции; для выполнения требуемых задач объекты используют собственные данные. Более простое понимание и поддержка системы классов, моделирующих заданную систему.

Создание объектов в объектно-ориентированной системе является одним из наиболее стандартных видов деятельности. Поэтому при распределении обязанностей, связанных с созданием объектов, следует руководствоваться шаблоном Creator.

Проблема. Какой класс должен отвечать за создание нового экземпляра выбранного класса A?

Решение. Назначить классу B обязанность создавать объекты другого класса A, если выполняется одно из следующих условий:

– класс B агрегирует или содержит объекты A;
– класс B активно использует объекты A;
– класс B обладает данными инициализации для объектов A.

Результаты. Использование шаблона не увеличивает число связей между классами, поскольку созданный класс, как правило, виден только для класса-создателя. Если процедура создания объекта достаточно сложная, то предпочтительно использовать такой шаблон, как Фабрика.

Шаблоны Low Coupling и High Cohesion.

Шаблон Слабая связанность позволяет снизить влияние изменений в одном классе системы на другие связанные с ним классы.

Степень связанности (англ. coupling) – это мера, определяющая, насколько жестко один элемент программной системы связан с другими элементами, либо каким количеством данных о других элементах он обладает.

Класс с низкой степенью связанности (слабо связанный) зависит от небольшого числа других классов.

Класс с высокой степенью связанности (жестко связанный) зависит от множества других классов. Наличие таких классов нежелательно, поскольку оно приводит к возникновению следующих проблем:

– изменения в связанных классах приводят к изменениям в данном классе;
– затрудняется понимание каждого класса в отдельности;
– усложняется повторное использование, поскольку для этого требуется дополнительный анализ классов, с которыми связан данный класс.

Проблема. Как уменьшить влияние изменений, вносимых в данный класс, на другие классы?

Решение. Распределить обязанности между классами таким образом, чтобы степень связанности системы оставалась низкой.

Результаты. Улучшаются возможности для повторного использования классов системы.

Появляется возможность поручить разработку отдельных частей системы разным разработчикам.

При злоупотреблении шаблоном система будет состоять из набора изолированных сложных классов, самостоятельно выполняющих все операции, и набора классов, хранящих данные.

Шаблон «Сильное сцепление».

Шаблон «Сильное сцепление» позволяет создавать простые для понимания классы, обладающие возможностью повторного использования.

В терминах объектно-ориентированного проектирования функциональное сцепление (англ. `cohesion`) – это мера связанности или сфокусированности обязанностей класса (подсистемы).

Класс обладает высокой степенью сцепления (сильным сцеплением), если его обязанности тесно связанны между собой и он не выполняет непомерных объёмов работы.

Класс с низкой степенью сцепления (слабым сцеплением) выполняет много несвязанных между собой обязанностей. Такие классы приводят к возникновению следующих проблем:

- сложность понимания системы;
- сложность при повторном использовании;
- сложность поддержки;
- ненадёжность, постоянная подверженность изменениям.

Проблема. Как обеспечить сфокусированность обязанностей классов, их управляемость и ясность для понимания?

Решение. Обеспечить высокую степень сцепления при распределении обязанностей.

Результаты. Классы с высокой степенью сцепления просты в поддержке и повторном использовании.

В некоторых случаях неоправданно использовать высокое сцепление для распределённых серверных объектов.

Индивидуальные задания

1 Изучить пример проектирования программной системы «Железнодорожные перевозки» [3, с. 88–109].

2 На основе заданных классов предметной области, а также их обязанностей (таблица 9.1), разработать диаграмму классов UML, в которой применены такие шаблоны GRASP, как `Information Expert` и `Creator`.

3 Создать проект библиотеки классов, в которой должны присутствовать модули классов, отмеченных на диаграмме классов. Реализовать атрибуты и опера-

ции классов, а также отношения между ними с помощью средств языка C#.

4 Добавить в решение проект консольного приложения и связать его с полученной библиотекой классов. Продемонстрировать в консольном приложении выполнение классами требуемых обязанностей.

5 Дополнительно добавить в решение проект приложения Windows Forms. Реализовать в приложении представление данных об объектах и возможность добавления новых объектов.

6 Обеспечить возможность сохранения данных путём сериализации объектов в XML-документ.

Таблица 9.1 – Индивидуальные задания

Номер варианта	Классы	Обязанности
1	Гостиница. Гостиничный номер. Клиент	*Знать.* Стоимость одного дня проживания. Число дней проживания. *Делать.* Добавить клиента в номер. Удалить клиента из номера. Определить общее число клиентов. Определить суммарную плату за проживание
2	Цех. Станок. Рабочий (может обслуживать несколько станков)	*Знать.* Какие станки, обслуживаются рабочим. Надбавка за обслуживание более чем одного станка. *Делать.* Добавить рабочего в цех. Добавить рабочему станок. Определить зарплату (зависит от числа обслуживаемых станков). Определить суммарные расходы на зарплату
3	Магазин. Товар. Продавец-консультант	*Знать* Какие товары, проданы сотрудником. Бонус за продажу единицы товара. *Делать* Добавить товар в магазин. Продать товар. Определить зарплату (зависит от числа проданных товаров). Определить общую выручку

Контрольные вопросы

1 В чём заключается подход проектирования на основе обязанностей `(RDD)`?

2 Какие выделяют виды обязанностей объекта?

3 Какие выделяют основные шаблоны GRASP?

4 Какую проблему решает шаблон `Expert` и в чём заключается это решение?

5 Какие классы должны отвечать за создание объектов согласно шаблону `Creator`?

6 К каким проблемам приводит использование в системе классов с высокой степенью связывания?

7 Что понимают под функциональным сцеплением в объектно-ориентированным проектировании?

8 Чем плохо использование классов с низкой степенью сцепления?

Список литературы

1 **Шаллоуей, А.** Шаблоны проектирования. Новый подход к объектно-ориентированному анализу и проектированию / А. Шаллоуей. – Москва: Вильямс, 2002. – 288 с.

2 **Мартин, Р.** Принципы, паттерны и методики гибкой разработки на языке C#: пер. с англ. / Р. Мартин, М. Мартин. – Санкт-Петербург: Символ-Плюс, 2011. – 768 с.

3 Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. – Санкт-Петербург: Питер, 2016. – 366 с.

4 **Тепляков, С.** Паттерны проектирования на платформе .NET / С. Тепляков. – Санкт-Петербург: Питер, 2015. – 320 с.