Лабораторная работа №1

Изучение базовых шаблонов проектирования

Цель работы: освоить применение базовых шаблонов проектирования при решении задачи, связанной с разработкой файловой базы данных, на примере паттерна делегирование.

Теоретический материал

В некоторых случаях использование наследования приводит к созданию неудачного проекта. Хотя и менее удобное, делегирование представляет собой универсальный способ расширения классов. Делегирование применимо во многих ситуациях, где наследование терпит фиаско.

Наследование - распространенный способ расширения и многократного использования функциональности класса. Делегирование представляет собой более общий подход к решению задачи расширения возможностей поведения класса. Этот подход заключается в том, что некоторый класс вызывает методы другого класса, а не наследует их. Во многих ситуациях, не позволяющих использовать наследование, возможно применение делегирования.

Наследование, например, подходит для описания отношений «is-a-kind-of» («Это разновидность...»), так как они очень статичны по своей природе. Однако отношения «is-a-role-played-by» («Это роль, которую играет...») неудобно моделировать при помощи наследования. Экземпляры класса могут играть сразу несколько ролей.

Более серьезная проблема заключается в том, что один и тот же субъект может играть различные комбинации ролей в разное время. Отношения, описываемые наследованием, являются статичными и не меняются со временем. Чтобы в условиях использования наследования смоделировать различные комбинации ролей на протяжении некоторого времени, один и тот же субъект должен быть представлен множеством разных объектов в разные моменты времени.

Моделирование динамически изменяющихся ролей при помощи наследования вызывает большие затруднения.

Для реализации паттерна «Делегирование» просто используется ссылка на экземпляр класса.

Самый лучший способ убедиться, что делегирование будет легко реализовать, - это сделать явными его структуру и назначение. Одним из решений этой задачи является реализация делегирования с использованием шаблона Interface.

Основным недостатком делегирования является его меньшая структурированность, чем наследования. Отношения между классами, построенные с применением делегирования, менее очевидны, чем представленные при помощи наследования, однако в большинстве случаев это обеспечивает значительно большую гибкость.

Задание на лабораторную работу

Разработать приложение, обладающее графическим интерфейсом и реализующее файловую базу данных в соответствии с заданием (Приложение 1). Архитектура базы данных и приложения должна быть спроектирована таким образом, чтобы было адекватно применение паттерна «Делегирование». Язык разработки – Java. Оформление лабораторной работы должно быть выполнено в соответствии с требованиями, приведенными в Приложении 2.

Лабораторная работа №2

Изучение порождающих шаблонов проектирования

Цель работы: освоить применение пораждающих шаблонов проектирования при решении задачи, связанной с разработкой файловой базы данных, на примере паттерна абстрактная фабрика.

Теоретический материал

Абстрактная фабрика — это порождающий паттерн проектирования, который позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.

Предположим, необходимо создать каркас пользовательского интерфейса, который должен работать со многими оконными системами, например, MS Windows, Linux или MacOS, т.е. на любой платформе с сохранением свойственного платформе внешнего вида и стиля (look and feel). Для каждого типа элемента окна (текстовое поле, кнопка, список и т.д.) создают абстрактный класс, а затем объявляют конкретный подкласс каждого такого класса для каждой поддерживаемой платформы. В целях надежности нужно гарантировать, что все созданные объекты элементов окна предназначены для нужной платформы. Здесь вступают в игру абстрактные классы-фабрики.

Абстрактный класс-фабрика определяет методы для создания экземпляра каждого абстрактного класса, представляющего элемент окна пользовательского интерфейса. Конкретные классы-фабрики представляют собой конкретные подклассы абстрактного класса-фабрики, реализующего методы для создания экземпляров конкретных классов элементов окна одной и той же платформы.

С точки зрения общего контекста абстрактный класс-фабрика и его конкретные подклассы организуют набор конкретных классов, работающих с разными, но связанными продуктами.

Абстрактная фабрика объявляет список продуктов, которые может запрашивать клиентский код. Конкретные фабрики относятся к различным операционным системам и создают элементы одного и того же вида.

В самом начале, программа определяет, какая из фабрик соответствует текущей ОС. Затем, создаёт эту фабрику и отдаёт её клиентскому коду. В дальнейшем, клиент будет работать только с этой фабрикой, чтобы исключить несовместимость возвращаемых продуктов.

Клиентский код не зависит от конкретных класс фабрик и элементов интерфейса. Он общается с ними через абстрактные интерфейсы. Благодаря этому, клиент может работать любой разновидностью фабрик и элементов интерфейса.

Чтобы добавить в программу новую вариацию элементов (например, для поддержки Linux), вам не нужно трогать клиентский код. Достаточно создать ещё одну фабрику, производящую эти элементы.

Таким образом, абстрактная фабрика скрывает от клиентского кода подробности того, как и какие конкретно объекты будут созданы. Но при этом клиентский код может работать со всеми типами создаваемых продуктов, так как их общий интерфейс был заранее определён.

Задание на лабораторную работу

Разработать приложение, обладающее графическим интерфейсом и реализующее файловую базу данных в соответствии с заданием (Приложение 1). Архитектура базы данных и приложения должна быть спроектирована таким образом, чтобы было адекватно применение паттерна «Абстрактная фабрика». Язык разработки – Java. Оформление лабораторной работы должно быть выполнено в соответствии с требованиями, приведенными в Приложении 2.

Лабораторная работа №3

Изучение разделяющих шаблонов проектирования

Цель работы: освоить применение разделяющих шаблонов проектирования при решении задачи, связанной с разработкой файловой базы данных, на примере паттерна "Интерфейс, только для чтения".

Теоретический материал

Часто возникает необходимость, чтобы объект изменялся только некоторыми его клиентами. Шаблон "Интерфейс, только для чтения" (Read-Only Interface) гарантирует, что клиенты, которым не позволено изменять объект, не будут его изменять благодаря тому, что они получают доступ к этому объекту через интерфейс, который не содержит каких-либо методов, способных изменить объект.

Данный паттерн может быть использован в следующих случаях:

- Существует класс, экземпляры которого одни классы могут изменять, а другие - нет.
- Нужно, чтобы экземпляры некоторого класса изменялись бы экземплярами других классов, находящихся в разных пакетах. Это означает объявление методов, которые модифицируют экземпляры классов пакета, закрытыми - не самый пригодный выбор для ограничения количества классов, которые могут вызвать модифицирующие методы.
- Необходимо заставить клиентов класса обращаться к этому классу через определяемый самим программистом интерфейс. В целом, если проектируется класс и его клиенты, это правильная политика. Но в некоторых ситуациях это может быть неудобно. Клиентские клас-

сы могут разрабатываться третьими фирмами, или просто нежелательно менять интерфейс, используемый клиентскими классами.

Несмотря на то, что этот шаблон представляет собой хорошую защиту от ошибок программирования, но он не способен воспрепятствовать злонамеренному программированию.

Задание на лабораторную работу

Разработать приложение, обладающее графическим интерфейсом и реализующее файловую базу данных в соответствии с заданием (Приложение 1). Архитектура базы данных и приложения должна быть спроектирована таким образом, чтобы было адекватно применение паттерна «Интерфейс, только для чтения». Язык разработки – Java. Оформление лабораторной работы должно быть выполнено в соответствии с требованиями, приведенными в Приложении 2.

Лабораторная работа №4

Изучение структурных шаблонов проектирования

Цель работы: освоить применение структурных шаблонов проектирования при решении задачи, связанной с разработкой файловой базы данных, на примере паттерна итератор.

Теоретический материал

Шаблон «Итератор» определяет интерфейс, который объявляет методы для последовательного доступа к объектам коллекции. Класс, осуществляющий доступ к коллекции только через этот интерфейс, не зависит от класса, реализующего этот интерфейс, и от класса коллекции.

Предположим, что создаются классы, которые позволяют просматривать инвентарь на товарном складе. Здесь должен применяться пользовательский интерфейс, который даст возможность пользователю просматривать описание, количество, местонахождение и другую информацию об инвентарной единице.

Классы просмотра инвентаря должны быть частью специализированного приложения. Поэтому они не должны зависеть от реального класса, предоставляющего коллекции предметов инвентаризации. Чтобы обеспечить такую независимость, проектируется интерфейс, позволяющий пользовательскому интерфейсу последовательно обращаться к коллекции инвентарных единиц, ничего не зная об использующемся реальном классе коллекции.

Данный паттерн может быть использован в следующих случаях:

- Класс нуждается в доступе к содержимому коллекции, не становясь зависимым от класса, который используется для реализации коллекции.
- Классу нужен универсальный способ доступа к содержимому множества коллекций.

Кроме методов, проверяющих наличие и считывающих следующий элемент коллекции, часто используются следующие методы:

- проверка наличия и считывание предыдущего элемента коллекции;
- перемещение к первому или последнему элементу коллекции;
- получение количества элементов обхода.

Во многих случаях алгоритм обхода класса-итератора требует доступа к внутренней структуре данных класса коллекции. По этой причине классы-итераторы часто реализуются в виде закрытого внутреннего класса, принадлежащего классу коллекции.

Класс коллекции может предоставлять различные объекты итерации, которые обходят коллекцию по-разному. Например, класс коллекции, поддерживающий ассоциацию между объектами ключей и объектами значений, может иметь одни методы создания итераторов, которые обходят только объекты ключей, и другие методы для создания итераторов, которые обходят только объекты значений.

Задание на лабораторную работу

Разработать приложение, обладающее графическим интерфейсом и реализующее файловую базу данных в соответствии с заданием (Приложение 1). Архитектура базы данных и приложения должна быть спроектирована таким образом, чтобы было адекватно применение паттерна «Итератор». Язык разработки – Java. Оформление лабораторной работы должно быть выполнено в соответствии с требованиями, приведенными в Приложении 2.

Лабораторная работа №5

Изучение поведенческих шаблонов проектирования

Цель работы: освоить применение поведенческих шаблонов проектирования при решении задачи, связанной с разработкой файловой базы данных, на примере паттерна наблюдатель.

Теоретический материал

Шаблон «Наблюдатель» позволяет организовать динамическую регистрацию зависимостей между объектами. В результате объект будет оповещать зависящие от него объекты об изменении своего состояния.

Предположим, что создается ПО для компании, которая изготавливает детекторы дыма, сенсоры движения и другие приборы безопасности. Чтобы с выгодой использовать преимущества нового рынка, компания планирует ввести новую линию устройств. Такие устройства смогут посылать сигнал на карту безопасности, которая может быть установлена на большинстве компьютеров. Ожидается, что компании, изготавливающие контролирующие системы безопасности, будут внедрять такие устройства и карты в свои системы. Чтобы облегчить процесс внедрения карт в контролирующие системы, нужно создать удобный пользовательский API.

API должен достаточно легко внедряться в программы будущих заказчиков с тем, чтобы они могли получать извещения от карты безопасности. Работа API не должна требовать от заказчиков изменения архитектуры уже имеющегося у них ПО. API может знать о ПО заказчика только то, что по крайней мере один или, возможно, несколько объектов будут иметь метод, который должен вызываться при получении извещения от устройства безопасности.

Данный паттерн может быть использован в следующих случаях:

- Реализуются два разных независимых класса. Экземпляр одного такого класса должен иметь возможность извещать другие объекты об изменена своего состояния. Экземпляр другого класса должен быть оповещен, когда связанный с ним объект изменяет состояние. Однако эти два класса не должны работать друг с другом. Чтобы быть многократно используемыми, им не следует иметь никакой непосредственной информации друг о друге.
- Есть отношение зависимости «один к нескольким», которое может потребовать от объекта оповещения некоторых зависящих от него объектов об изменении его состояния.
- Для передачи извещений и задания их приоритета нужна некоторая логика. Эта логика не зависит ни от отправителя, ни от получателя извещений.

Шаблон позволяет объекту передавать извещения другим объектам таким образом, что ни отправитель, ни получатель извещений ничего не знают о классах друг друга.

Существуют некоторые ситуации, когда использование данного паттерна может приводить к непредвиденным и нежелательным результатам:

- Если объект должен передать извещения большому количеству объектов, передача извещений может потребовать много времени. Это объясняется тем, что один объект может иметь множество наблюдателей, непосредственно зарегистрировавшихся на получение его извещений. Это может также произойти, когда какой-то объект имеет множество косвенных наблюдателей и его извещения каскадно передаются другими объектами. Иногда можно уменьшить негативные последствия такой ситуации, делая передачу извещений асинхронной, выполняющейся в своем собственном потоке. Однако асин-

хронная передача извещений способна породить свои собственные проблемы.

- Более серьезная проблема связана с циклическими зависимостями. Объекты вызывают методы notify друг друга до тех пор, пока не переполнится стек и не будет сгенерирована ошибка StackOverflowError. Несмотря на всю серьезность этой проблемы, она может быть легко решена посредством задания внутреннего флага в одном из классов, участвующих в цикле.

Задание на лабораторную работу

Разработать приложение, обладающее графическим интерфейсом и реализующее файловую базу данных в соответствии с заданием (Приложение 1). Архитектура базы данных и приложения должна быть спроектирована таким образом, чтобы было адекватно применение паттерна «Наблюдатель». Язык разработки – Java. Оформление лабораторной работы должно быть выполнено в соответствии с требованиями, приведенными в Приложении 2.

Лабораторная работа №6

Изучение порождающих шаблонов проектирования

Цель работы: освоить применение шаблонов многопоточного программирования при решении задачи, связанной с разработкой файловой базы данных, на примере паттерна будущее.

Теоретический материал

Паттерн «Будущее» предполагает следующий механизм работы. Используется объект, который инкапсулирует результат вычислений и позволяет скрыть от клиентов, синхронно или асинхронно выполняется вычисление. Этот объект будет содержать метод чтения результатов вычислений. Метод ожидает получения результата, если вычисления производятся асинхронно, и продолжает или выполняет вычисление, если оно является синхронным и еще не завершено.

Предположим, что проектируется класс, который позволит программе получать текущие данные о погоде для данной местности. Ожидается, что общей сферой применения такого класса является отображение информации о погоде среди некоторой другой информации. Например, такой класс может использоваться как часть сервлета, который генерирует код HTML для Web-страницы, показывающей информацию о погоде наряду с новостями.

Время, которое понадобится для получения информации о погоде, будет меняться в значительной степени. Оно может составлять долю секунды или, возможно, несколько минут. Нужно спроектировать этот класс таким образом, чтобы минимизировать влияние на клиентов этого класса.

Можно создать экземпляры класса погоды, которые будут отправлять события заинтересованным объектам в том случае, когда этот класс получает затребованную информацию о погоде. Однако такой подход нежелателен, потому что если клиенты класса погоды должны будут получать асинхронный вызов, извещающий о наступлении события, то клиенты класса сильно усложнятся.

Класс клиента вынужден будет реализовать метод по получению новой информации о погоде таким образом, чтобы обеспечить безопасность потока выполнения.

Вместо того чтобы заставить метод прямо возвращать результат вычисления, вынуждаем его возвращать объект, который инкапсулирует это вычисление. Этот объект будет содержать метод, вызываемый в том случае, когда нужно будет получить результат вычисления. Вызывающие этот метод не знают, выполняется ли вычисление синхронно или асинхронно по отношению к их потоку.

Задание на лабораторную работу

Разработать приложение, обладающее графическим интерфейсом и реализующее файловую базу данных в соответствии с заданием (Приложение 1). Архитектура базы данных и приложения должна быть спроектирована таким образом, чтобы было адекватно применение паттерна «Будущее». Язык разработки – Java. Оформление лабораторной работы должно быть выполнено в соответствии с требованиями, приведенными в Приложении 2.

Приложение 1. Варианты лабораторных заданий

Разработать программу базы данных в соответствии с вариантом, представленным в таблице.

Номер варианта	**База данных**
1	Книги
2	Автомобили
3	Отдел кадров
4	Деканат
5	Спортивная секция
6	Кадастровая служба
7	Музей
8	Школа
9	Магазин
10	Фабрика

Приложение 2. Требования к оформлению лабораторных работ

По каждой лабораторной работе необходимо составить отчет, который должен содержать:

- титульный лист;
- название и цель работы;
- лабораторное задание в соответствии с вариантом;
- ход выполнения лабораторной работы, включающий словесное описание алгоритма работы программы или его графическое представление;
- листинг программы;
- результаты выполнения программы для различных случаев;
- выводы.

Отчет должен предоставляться в печатном виде в соответствии с представленными выше требованиями.