Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный технологический университет»

Кафедра информационных систем и технологий

**Семинар «Шаблоны проектирования»**

**«Паттерны проектирования. Поведенческие шаблоны»**

Минск 2023

**Цель поведенческих паттернов проектирования:**

Поведенческие шаблоны необходимы для определения способов и алгоритмов реализации взаимодействия объектов и классов. Их использование позволяет сделать программный продукт более гибким, за счет упрощения взаимодействия между объектами и снижения уровня связности системы.

**Виды поведенческих паттернов:**

* Цепочка обязанностей (Chain of responsibility);
* Команда (Command);
* Итератор (Iterator);
* Посредник (Mediator);
* Хранитель (Memento);
* Наблюдатель (Observer);
* Состояние (State);
* Стратегия (Strategy);
* Шаблонный метод (Template method);
* Посредник (Visitor).

**Цепочка обязанностей**

Паттерн объединяет несколько объектов в цепочку, по которой последовательно передаются запросы. Каждый объект из цепочки может обработать запрос сам, а может передать его следующему объекту цепочки.

Когда используют:

* Когда программа должна иметь возможность обработки запросов разными способами, при этом не известно, какие конкретно запросы придут и какие обработчики для них нужны;
* Когда необходимо выполнение обработчиков в определенном порядке.

Обработчики реализуют один абстрактный класс Handler, что содержит ссылку на самого себя (обозначена как successor) для делегирования обязанностей по обработке следующему в цепочке объекту. Метод handlerRequest() отвечает за эту делегацию.

Преимуществом является уменьшение зависимости между клиентом и обработчиком.

Недостаток данного паттерна заключается в том, что поступивший запрос может остаться не обработанным.

Пример использования: в некоторых языках программирования шаблон используется для обработки исключений. При возникновении исключения идет проверка на наличие механизма его обработки, в ином случае идет передача в стек вызовов. В стеке вызовов процесс будет повторяться до тех пор, пока не будет найден код для обработки исключения или пока не останется объектов для передачи запроса.

**Команда**

Паттерн преобразует запросы в объекты, позволяя передавать их как аргументы при вызове методов, выстраивать запросы в очередь, отменять операции.

Когда используют:

* Когда запросы должны обрабатываться в разном порядке и/или в разное время;
* При необходимости сохранения истории запросов;
* Когда обработчик должен быть отделен от вызывающей стороны.

Клиент создает объекты конкретных команд. Отправитель хранит ссылку на объект команды и обращается к нему в том случае, если необходимо выполнить какое-то действие. Отправитель работает с командами исключительно через их общий интерфейс. Команда описывает этот интерфейс (обозначен как ConcreteCommand). Отправитель не знает какую команду использует, т.к. получает готовый объект команды от клиента. Получатель содержит бизнес-логику программы.

Из преимуществ: шаблон позволяет реализовать отмену и повтор команд, отложенный запуск команд, собирает сложные команды из простых.

Паттерн усложняет код программы, так как вводятся дополнительные классы.

Пример использования: шаблон можно использовать для облегчения асинхронной обработки алгоритмов, путем передачи очереди заданий для обработки таким образом, чтобы очередь не знала о фактическом результате выполнения. Объект, который ставится в очередь, реализует свой алгоритм в рамках интерфейса, ожидаемого очередью.

**Итератор**

Шаблон нужен для получения доступа к элементам составного объекта без доступа к его внутреннему представлению.

Когда используют:

* Если требуется доступ к элементам без открытия доступа ко всему представлению;
* При необходимости единого интерфейса для обхода;
* Когда нужны множественные или параллельные обходы элементов.

Итератор описывает интерфейс для доступа и обхода элементов коллекции. Конкретный итератор нужен для реализации алгоритма обхода конкретной коллекции. Коллекция возвращает новый экземпляр определенного конкретного итератора, связывая его с текущим объектом коллекции. Клиент взаимодействует со всеми объектами через интерфейсы коллекции и итератора.

Данный шаблон упрощает классы хранения данных, позволяет перемещаться по структуре данных в разные стороны, а также реализовывать разные способы обхода.

Однако, если можно обойтись циклом, то необходимость в паттерне отпадает.

**Посредник**

Снижает связность множества компонентов, работающих совместно, что убирает нужду объектов вызывать друг друга напрямую.

Когда используют:

* Когда связь между наборами объектов сложна и хорошо определена;
* Когда существует слишком много взаимосвязей, в следствие чего нужна общая точка управления.

Посредник определяет интерфейс для обмена информацией с объектами Colleague. Конкретный посредник координирует действия объектов Colleague. Каждый класс Colleague знает о своем объекте Посредник, и все объекты обмениваются информацией только с Посредником.

Посредник предлагает упростить взаимодействие между компонентами; удалить зависимости между компонентами, позволяя повторно их использовать; централизовать управление компонентами.

Пример использования: приложения для управления почтовой рассылкой. Идет отслеживание, кто и на что подписан, и предоставляется точка доступа, через которую администратор рассылки может отправлять сообщения всем или отдельным подписчикам.

**Хранитель**

Захватывает и извлекает внутреннее состояние объекта, чтобы его можно были восстановить позже, не нарушая инкапсуляции.

Когда используют:

* Если есть необходимость в сохранении внутреннего состояния объекта с последующим восстановлением;
* Если границы инкапсуляции должны быть сохранены;
* Когда внутреннее состояние не может быть раскрыто интерфейсами без раскрытия реализации.

Создатель делает снимки своего состояния и может воспроизводить прошлое состояние, если передать в него готовый снимок. Хранитель – простой объект данных, что содержит состояние создателя, Опекун знает, когда делает снимок создателя и когда его нужно восстанавливать.

Паттерн не нарушает инкапсуляции исходного объекта. Упрощает структуру исходного объекта.

Из недостатков: требует много памяти, если клиенты слишком часто создают снимки; в некоторых языках программирования сложно гарантировать, чтобы только исходный объект имел доступ к состоянию снимка.

**Наблюдатель**

Паттерн позволяет одному или нескольким объектам получать уведомления об изменениях состояния других объектов в системе.

Когда используют:

* Если изменение состояния одного/нескольких объектов должно вызывать реакцию других объектов;
* Для управления массовой рассылкой.

Наблюдатель передает запрос одновременно всем заинтересованным получателям, но позволяет им динамически подписываться/отписываться от оповещений.

Преимущества: издатели не зависят от конкретных классов подписчиков (и наоборот); можно подписывать/отписывать получателей на лету.

Недостатки: подписчики оповещаются в случайном порядке.

Пример использования: любой графический интерфейс. При вызове пользователем события, например, нажатия кнопки, элемент управления перебирает все наблюдатели и посылает каждому из них уведомление.

**Состояние**

Данный шаблон позволяет объектам менять свое поведение в зависимости от своего состояния.

Когда используют:

* Если поведение объекта должно зависеть от его состояния;
* Переходы между состояниями должны быть явными;
* Поведение объекта с его состоянием связывают сложные условия.

Контекст хранит ссылку на объект состояния и передает ему часть работы, которая зависит от состояний. Состояние описывает общий интерфейс для всех конкретных состояний (А и В). Конкретные состояния реализуют поведения, связанные с определенным состоянием контекста.

Из преимуществ: упрощает код контекста; концентрирует в одном месте код, связанный с определенным состоянием.

Из недостатков: может усложнить код в случае, когда состояний мало и они редко меняются.

Пример использования: электронное письмо может иметь разные состояния, каждое из которых влияет на то, как объект обрабатывает разные функции. Если состояние «не отправлено», то вызов функции отправки сообщения отправит его, а если вызвать функцию отмены отправки, то она либо вызовет ошибку, либо просто не выполнится.

**Стратегия**

Определяет семейство схожих алгоритмов и помещает каждый из них в собственный класс, благодаря чему алгоритмы можно взаимозаменять во время исполнения программы.

Когда используют:

* Если единственным отличием между родственными классами является их поведение;
* Если необходимо несколько версий алгоритма;
* Когда поведение класса должно быть определено во время выполнения;
* Когда алгоритмы получают доступ/используют данные, что не должны быть доступны вызывающему коду.

Контекст хранит ссылку на объект конкретной стратегии и работает с ним через общий интерфейс стратегий (обозначен как IStrategy). Стратегия определяет общий для всех вариаций алгоритма интерфейс. Конкретные стратегии реализуют различные вариации алгоритма.

Шаблон позволяет заменять алгоритмы на лету; изолирует код и данные алгоритмов от остальных классов; отделяется от наследования и делегирования.

Но, усложняет программу за счет введенных дополнительных классов и клиент должен знать разницу между стратегиями для выбора подходящей.

Пример использования: при импорте данных в новую систему на основе набора данных могут выполняться разные алгоритмы проверки. Настроив импорт на использование стратегий, можно убрать логику выбора набора проверки и отделить импорт от кода проверки. Это позволит динамически вызывать одну/несколько стратегий во время импорта.

**Шаблонный метод**

Вид поведенческого паттерна, который определяет структуру алгоритма, позволяя реализующим классам определять фактическое поведение.

Когда используют:

* Когда общее поведение подклассов должно быть локализовано в общем классе;
* Если есть нужда в единой абстрактной реализации алгоритма;
* Когда большинство/все подклассы должны реализовывать поведение;
* Если родительские классы должны иметь возможность единообразно вызывать поведение в своих подклассах.

Абстрактный класс определяет шаги алгоритма и содержит шаблонный метод, состоящий из вызовов этих шагов. Конкретный класс переопределяет все/некоторые шаги алгоритма. Конкретные класс не переопределяют сам шаблонный метод.

Шаблонный метод облегчает повторное использование кода.

Однако, есть ограничение в виде «скелета» существующего алгоритма; с ростом количества шагов данный паттерн становится тяжело поддерживать.

В качестве примера можно использовать аналогию из жизни. При строительстве дома строители используют что-то схожее с шаблонным методом. Есть проект с шагами строительства, но, могут быть внесены незначительные изменения, чтобы дом был немного отличным от другого.

**Посетитель**

Паттерн применяет одну или несколько операций к набору объектов во время выполнения, отделяя операции от структуры объекта.

Когда используют:

* Если над структурой объекта выполняется множество несвязанных операций;
* Когда структура объекта не должна меняться, но операции, выполняемые над ней, могут;
* При наличии операций, которые должны выполняться над конкретными классами структуры объекта;
* Когда допустимо раскрытие внутреннего состояния или операций структуры объекта.

Посетитель описывает общий интерфейс для всех типов посетителей. Он объявляет набор методов, отличающихся типом входящего параметра, которые нужны для запуска операции для всех типов конкретных элементов.

Конкретные посетители реализуют поведение для всех типов элементов. Элемент описывает метод принятия посетителя. Конкретные элементы реализуют методы принятия посетителя. Клиентом является коллекция или сложны составной объект.

Преимущества: объединяет родственные операции в одном классе; посетитель может накапливать состояние при обходе структуры элементов; упрощает добавление операций.

Недостатки: иерархия элементов часто меняется; использование паттерна может привести к нарушению инкапсуляции элементов.

Пример использования: страховой агент, нуждающийся в новых клиентах. Он посещает дома на улице, но для разных типов домов у него разные предложения.

**Итоги**

Были изучены 10 поведенческих паттернов проектирования. Для каждого из паттернов были рассмотрены назначение, цели использования, графическое представление, преимущества и недостатки, пример использования.