Научная статья
Original article
УДК 004.415.2

АНАЛИЗ ПОВЕДЕНЧЕСКИХ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

ANALYSIS BEHAVIORAL DESIGN PATTERNS OF MOBILE APPLICATION

Курганова Александра Геннадьевна, магистрант, Московский государственный технический университет имени Н.Э. Баумана, г. Москва, kurganova06.1998@gmail.com

Kurganova Alexandra Gennadievna, Master's student, Bauman Moscow State Technical University, Moscow, kurganova06.1998@gmail.com

Аннотация. В данной статье рассматривается анализ поведенческих паттернов проектирования при разработке мобильного приложения. Паттерн проектирования — это часто встречающееся решение определенной проблемы при проектировании архитектуры программ. Перед встраиванием паттернов в мобильное приложение необходимо сначала их выбрать. Для выбора определенных паттернов проектирования сперва необходимо провести их анализ, который включает в себя: классификацию всех паттернов проектирования на группы, разбор сути каждого из паттернов и практическое применение каждого из них.

Annotation This article discusses the analysis of behavioral design patterns in the development of a mobile application. A design pattern is a common solution to a specific problem when designing a program architecture. Before embedding patterns in a mobile

application, you must first select them. To select certain design patterns, it is first necessary to analyze them, which includes: classification of all design patterns into groups, analysis of the essence of each of the patterns and practical application of each of them.

Ключевые слова: паттерн проектирования, мобильное приложение, поведенческие паттерны проектирования, анализ, классификация, практическое применение

Keywords: design pattern, behavioral design patterns, mobile application, analysis, classification, practical application

Введение

При разработке мобильного приложения прежде всего изучают предметную область, после чего начинают проектировать общую структуру приложения и продумывать различные типовые проблемы, которые будут возникать на различных стадиях проектирования. Для решения типовых проблем создают концепцию, которая являются обобщенной и высокоуровой. Поскольку это концепция, то ее можно использовать множество раз без привязки к определенной программе. Концепции этих решений назвали паттернами проектирования.

Для выбора определенных паттернов проектирования для приложения сперва необходимо провести их анализ, а именно: понять на какие группы делятся все паттерны проектирования, в чем их суть и каково практическое применение каждого из них. Анализ паттернов проектирования будет разобран в данной статье.

Группы паттернов проектирования и их состав

Все паттерны проектирования делится на три группы:

1. порождающие
2. структурные
3. поведенческие

Поведенческие паттерны проектирования решают задачи эффективного и безопасного взаимодействия между различными объектами программы.

К поведенческим паттернам относят паттерны:

1. цепочка обязанностей
2. команда
3. итератор
4. посредник
5. снимок
6. наблюдатель
7. состояние
8. стратегия
9. шаблонный метод
10. посетитель

Поведенческие паттерны проектирования

Цепочка обязанностей

Цепочка обязанность – порождающий паттерн проектирования, который позволяет передавать запросы по цепочке, причем каждый последующий обработчик либо сам обрабатывает полученное ему действие, или передает его ниже для выполнения следующим объектом.

Данный паттерн проектирования может быть полезен при решении задачи описанной ниже, которая требует определенной последовательности действий.

Есть сайт аукциона, на котором представлены различные категории вещей в зависимости от статуса пользователя. Для того, чтобы отобразить пользователю страницу с каталогом, которая подходит его статусу, необходимо сделать несколько проверок, подтверждающих статус пользователя. С появлением новых пользователей разных статусов, программный код проверки статусов пользователя значительно увеличится и усложниться.

Цепочка обязанностей позволяет решить данную проблему путем создания объектов, отвечающих за отдельную проверку. Помимо этого у каждого объекта будет ссылка на следующий объект в цепочке для возможности передачи ему проверки в случае невозможности выполнения действия самому. Благодаря созданию единой цепочки с ссылками объекта на каждого последующего обработчика появляется гарантия обработки каждого из посланного события.

Команда

Поведенческий паттерн проектирования команда позволяет превращать запросы в объекты, благодаря чему эти объекты можно передавать в качестве аргументов вызова функций. Помимо этого объекты можно ставить в очередь, отменять и логировать.

Данный паттерн может быть полезен при решении задачи, описанной ниже.

При разработке приложения для многофункционального редактора заметок необходимо создать множество различных кнопок: для профиля, для контекстного меню, для текстового редактора заметок. Если сделать единый класс кнопки для всего пользовательского интерфейса и множество подклассов для обработки событий на нажатия по каждой из кнопок. Неудобство данного подхода заключается в том, что в разных частях приложения может оказаться одна и та же или похожая по функционалу кнопка. Тогда обработчик придется дублировать в каждый подкласс кнопки.

Решением данной проблемы является внедрения паттерна команда. Необходимо создать класс для хранения объекта команды, который в свою очередь будет перенаправлять вызов обработчика кнопки соответствующему объекту слоя бизнес-логики. В классе для хранения объекта команды будут собраны различные команды кнопок всего пользовательского интерфейса приложения.

Итератор

Итератор – поведенческий паттерн проектирования, благодаря которому можно последовательно проходить по объектам не раскрывая их реализацию.

Паттерн итератор подходит для решения проблемы обхода по различным видам коллекции. Чтобы не писать в классе коллекции множество разных методов обхода по ней, нужно создать отдельный класс, который будет итерироваться по коллекции. Этот класс будет отслеживать текущую позицию в коллекции, по скольким еще элементам осталось пройти и состояние обхода. Благодаря этому по коллекции может проходить сразу несколько итераторов и в дальнейшем функционал обхода легко расширить за счет добавления нового итератора с методом обхода.

Посредник

Посредник – поведенческий паттерн проектирования, который позволяет уменьшить связанность множества объектов между собой путем перемещения этих связей в отдельный класс, называемый посредником.

Паттерн посредник может быть полезен при решении задачи, описанной ниже.

На сайте форума есть множество элементов форм пользовательского интерфейса. Все эти формы, как правило, состоят из похожих между собой полей для ввода поля или выбора ответа с проверкой данных после их ввода. Если функционал проверки каждого такого поля написать рядом с ним, то в дальнейшем нельзя будет создать такой же элемент, придется его дублировать, но с другой логикой проверки.

Для решения данной проблемы можно перенести проверки полей в класс. Благодаря этому созданный класс будет принимать событие от элемента пользовательского интерфейса, а элемент пользовательского интерфейса будет зависеть только от созданного класса, а не логики, прописанной ему напрямую.

Снимок

Данный поведенческий паттерн позволяет сохранять и восстанавливать прошлые состояния объектов без раскрытия их реализации.

Данный паттерн может быть полезен при решении задачи, описанной ниже.

При разработке приложения для многофункционального редактора заметок необходимо реализовать функционал отмены какого-либо действия, сделанным пользователем в текстовом редакторе заметок. Для возможности такого функционала необходимо сохранять последнее состояние текстового редактора. Для этого можно сохранить значения каждого из его полей. Минус такого подхода в том, что тогда любой другой класс программы, сможет получить доступ к последнему состоянию текстового редактора.

Для решения данной проблемы необходимо применить поведенческий паттерн снимок, благодаря которому создании копии состояния объекта делегируется самому объекту, который владеет этим состоянием. Благодаря этому не возникает нарушения принципа инкапсуляции объектно-ориентированного программирования. Создание копии необходимо поместить в отдельный объект с ограниченным интерфейсом, позволяющим читать и восстанавливать данные, при этом открытым для класса, к которому он относится. С помощью объекта для создания снимка другие классы смогут получать копии при необходимости лишь через ограниченный интерфейс, не трогая при этом основной класс.

Наблюдатель

Наблюдатель – поведенческий паттерн проектирования, который предоставляет механизм подписки и отписки на объект с реагированием на события, которые происходят в других объектах.

Данный паттерн полезен при описании следующей задачи.

В магазине есть товар, который очень нужен покупателю. Для того, чтобы покупатель не приходил каждый раз в магазин в ожидании завоза товара, магазин может прислать клиенту оповещение когда интересующий его товар доставят в магазин. Для реализации такой системы подписок необходимо хранить внутри объекта издателя список ссылок на объекты подписчиков. Издатель должен предоставить методы, благодаря которым подписчики могли бы подписываться или отписываться от него. В результате такой реализации когда издатель будет получать событие, подписчики по определенным методам смогут получить эти события.

Состояние

Состояние – поведенческий паттерн проектирования, благодаря которому объекты при изменении состояния могут менять свое поведение, создавая ощущение, что был изменен сам объект.

Мобильное приложение имеет собственный жизненный цикл, в ходе которого приложения активно, находится в фоновом режиме или же закрыто. Находясь в разных состояниях приложение может по-разному реагировать на одни и те же события. Например при получении новой подписки в социальной сети, если приложение активно, то пользователю должно прийти уведомление. А если приложение закрыто, то оповещение не должно придти. Проблема заключается не только в множестве поведений на одно и то же событие, что затрудняет программную реализацию, но и в увеличении поддержки числа состояний приложений, например когда оно собирается стать активным или наоборот собирается стать неактивным.

Для решения данной проблемы был создан паттерн состояние , благодаря которому можно создать классы для каждого состояния объекта, которые будет хранить поведение, соответствующие различным состояниям. Можно также создать отдельные классы для каждого состояния и один общий, который будет делегировать работу определенному состоянию. Благодаря этому и отдельные

классы состояний и состояния будут знать друг о друге и инициировать переходы между собой.

Стратегия

Стратегия – поведенческий паттерн проектирования, который определяет семейство алгоритмов и помещает каждый алгоритм в отдельный класс, в результате чего алгоритмы можно взаимозаменять во время исполнения программы.

Данный паттерн полезен при разработке приложений с алгоритмами, которые часто изменяются. Примером такого приложения может послужить навигатор, в котором есть несколько способов маршрутов: пешеходный, на машине, на общественном транспорте, велосипеде и другие. С помощью паттерна стратегия можно поместить каждый маршрут в отдельные стратегии, которые имеет общий интерфейс. Таким образом достаточно сменять стратегии между собой в зависимости от выбранного маршрута.

Шаблонный метод

Шаблонный метод – поведенческий паттерн, который определяет только скелет алгоритма, делегируя какую-то конкретную деталь подклассу не меняя общей структуры.

С помощью паттерна шаблонного метода можно разбить алгоритм на определенные шаги и часть этих шагов сделать универсальными для множества объектов. При этом все шаги алгоритма будут последовательно соединены между собой обеспечивая корректный результат работы алгоритма при работе с различными объектами.

Посетитель

Посетитель – поведенческий паттерн проектирования, благодаря которому возможно добавлять в программу новые действия не изменяя объекты, над которыми будут выполняться действия.

Данный паттерн проектирования может быть полезен в случае если в программе по работе с разными форматами документов потребовался бы функционал экспортирования в определенный формат. Для того, чтобы не изменять уже существующие классы документов, можно создать объект, который будет принимать различные объекты формата документов и реализовать механизм экспортирования. При этом исходные классы с форматами документов следует изменить, добавив нужный метод для механизма поиска определенного класса формата документов для объекта, занимающимся экспортом.

Заключение

В результате проведения подробного анализа поведенческих паттернов проектирования, выявлены их суть, достоинства, недостатки и практическое применение. После проведения анализа появилась возможность не только выбрать определенные паттерны проектирования, которые могут быть встроены в разрабатываемое мобильное приложение, но и грамотно их внедрить в уже работающую систему.

Литература

1. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес; [пер. с англ.: А. Слинкин науч. ред.: Н. Шалаев]. — Санкт-Петербург [и др.] : Питер, 2014. — 366 с. : ил. ; 24 см.
2. Head First. Паттерны проектирования. Обновленное юбилейное издание / Эрик Фримен, Элизабет Робсон, Кэти Сьерра, Берт Бейтс — Санкт-Петербург [и др.] : Питер, 2019. — 656 с: ил. – (Серия «Head First O’Reilly»).

3. [Электронный ресурс] 2019. – Режим доступа: http://design-pattern.ru/, свободный.
4. [Электронный ресурс] – 2017 г. – Режим доступа: https://habr.com/ru/company/mailru/blog/325492/, свободный.
5. [Электронный ресурс] – 2018 г. – Режим доступа: https://medium.com/educative/the-7-most-important-software-design-patterns-d60e546afb0e, свободный.
6. [Электронный ресурс] – 2017 г. – Режим доступа: https://www.raywenderlich.com/477-design-patterns-on-ios-using-swift-part-1-2, свободный.
7. [Электронный ресурс] – 2018 г. – Режим доступа: https://medium.com/educative/the-7-most-important-software-design-patterns-d60e546afb0e, свободный.
8. [Электронный ресурс] – 2020 г. – Режим доступа: https://blog.usejournal.com/design-patterns-in-ios-swift-977776513f66?gi=9e1a6d020306, свободный.
9. [Электронный ресурс] – 2018 г. – Режим доступа: https://medium.com/@lubabahasnain93/design-patterns-in-swift-part-i-creational-design-pattern-18d4be82092f, свободный.
10. Florent Vilmart Hands-On Design Patterns with Swift: by Florent Vilmart /Florent Vilmart, Giordano Scalzo, Sergio De Simone; — Санкт-Петербург [и др.] : Packt Publishing, 2018. — 414 с.
11. [Электронный ресурс] – 2019 г. – Режим доступа: https://khawerkhaliq.com/blog/swift-design-patterns-command-pattern/, свободный.
12. [Электронный ресурс] – 2020 г. – Режим доступа: https://betterprogramming.pub/implement-the-strategy-design-pattern-in-swift-5d9c3f221277, свободный.
13. Adam Freeman. Pro Design Patterns in Swift / Adam Freeman – New York: Apress, 2015. — 414 с.

14. [Электронный ресурс] – 2020 г. – Режим доступа: https://exyte.com/blog/understanding-ios-design-patterns, свободный.
15. [Электронный ресурс] – 2017 г. – Режим доступа: https://brightinventions.pl/blog/swift-facade-pattern/, свободный.
16. [Электронный ресурс] – 2020 г. – Режим доступа: https://www.jellyfishtechnologies.com/blog/builder-design-pattern-swift/, свободный.

References

1. Techniques of object-oriented design. Design patterns / E. Gamma, R. Helm, R. Johnson, D. Vlissides; [trans. from English: A. Slinkin scientific editor: N. Shalaev]. - St. Petersburg [et al.] : St. Petersburg, 2014. - 366 p. : ill. ; 24 cm.
2. Head First. Design patterns. Updated anniversary edition / Eric Freeman, Elizabeth Robson, Kathy Sierra, Bert Bates - St. Petersburg [et al.] : St. Petersburg, 2019. - 656 pp. - (Series "Head First O'Reilly").
3. [Electronic resource] 2019. - Access mode: http://design-pattern.ru /, free.
4. [Electronic resource] - 2017 - Access mode: https://habr.com/ru/company/mailru/blog/325492 /, free.
5. [Electronic resource] - 2018 - Access mode: https://medium.com/educative/the-7-most-important-software-design-patterns-d60e546afb0e, free.
6. [Electronic resource] - 2017 - Access mode: https://www.raywenderlich.com/477-design-patterns-on-ios-using-swift-part-1-2, free.
7. [Electronic resource] - 2018 - Access mode: https://medium.com/educative/the-7-most-important-software-design-patterns-d60e546afb0e, free.
8. [Electronic resource] - 2020 - Access mode: https://blog.usejournal.com/design-patterns-in-ios-swift-977776513f66 ?gi=9e1a6d020306, free.
9. [Electronic resource] - 2018 - Access mode: https://medium.com/@lubabahasnain93/design-patterns-in-swift-part-i-creational-design-pattern-18d4be82092f, free.

10. Florent Vilmart Hands-On Design Patterns with Swift: by Florent Vilmart /Florent Vilmart, Giordano Scalzo, Sergio De Simone; - Saint Petersburg [et al.] : Packt Publishing, 2018— - 414 p.
11. [Electronic resource] - 2019 - Access mode: https://khawerkhaliq.com/blog/swift-design-patterns-command-pattern /, free.
12. [Electronic resource] - 2020 - Access mode: https://betterprogramming.pub/implement-the-strategy-design-pattern-in-swift-5d9c3f221277, free.
13. Adam Freeman. Pro Design Patterns in Swift / Adam Freeman - New York: Apress, 2015— - 414 p.
14. [Electronic resource] - 2020 - Access mode: https://exyte.com/blog/understanding-ios-design-patterns, free.
15. [Electronic resource] - 2017 - Access mode: https://brightinventions.pl/blog/swift-facade-pattern /, free.
16. [Electronic resource] - 2020 - Access mode: https://www.jellyfishtechnologies.com/blog/builder-design-pattern-swift /, free.

© Курганова А.Г., 2022 Научно-образовательный журнал для студентов и преподавателей «StudNet» №4/2022.

Для цитирования: Курганова А.Г., АНАЛИЗ ПОВЕДЕНЧЕСКИХ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ// Научно-образовательный журнал для студентов и преподавателей «StudNet» №4/2022.