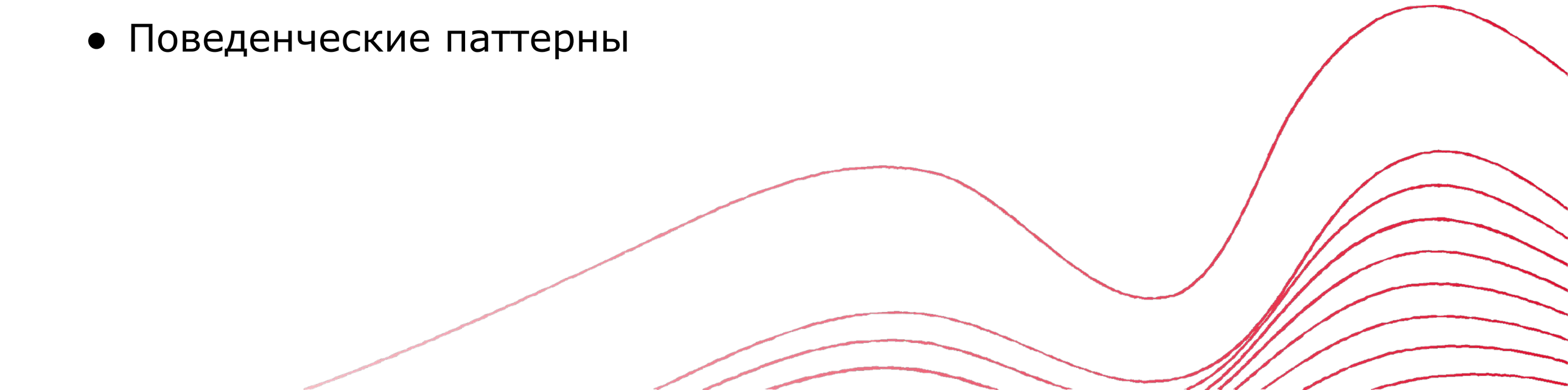
Лекция 16

Паттерны

- Что такое паттерны?
- Порождающие паттерны
- Структурные паттерны
- Поведенческие паттерны



Паттерны. Важные ссылки:

Сайт, на котором объяснение паттернов происходит через картинки и простые примеры практически в игровой форме. Playgrounds для паттернов. Я бы рекомендовал погрузится на недельку в этот сайт и попробовать все, что там описано:
https://refactoring.guru/ru/design-patterns/catalog

Шпаргалка по паттернам, полезно перед собеседованием, но изучать по этой статье было бы сложно:
https://stfalcon.com/ru/blog/post/ios-patterns

Та самая, легендарная книга:
https://docs.google.com/file/d/0B6GuCegBf3X3Tm1rZl9BUTduQm8/edit?resourcekey=0-ME3Ni9D9Wae8zLuAbNPx6w

Паттерны на примерах, репозиторий с кодом:
https://github.com/ochococo/Design-Patterns-In-Swift

Книга которую советовали мне для изучения паттернов когда я начинал:
https://fabulabook.com/product/paterny-proyektuvannya/?utm_source=Google%20Shopping&utm_campaign=Fabula&utm_medium=cpc&utm_term=18989&gclid=CjwKCAjwyIKJBhBPEiwAu7zll4jS2EfuO5VmvZUu13tLeRc9tmNg_rj4hAqroYYDRuHVNfnxK0j4ORoC2cEQAvD_BwE

Порождающие паттерны

1. **Singleton** (одиночка) - Гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.
   https://refactoring.guru/ru/design-patterns/singleton/swift/example

2. **Abstract Factory** (абстрактная фабрика) - Позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.
   https://refactoring.guru/ru/design-patterns/abstract-factory/swift/example

3. **Factory Method** (фабричный метод) - Определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.
   https://refactoring.guru/ru/design-patterns/factory-method/swift/example

4. **Builder** - Позволяет создавать сложные объекты пошагово. Строитель дает возможность использовать один и тот же код строительства для получения разных представлений объектов.
   https://refactoring.guru/ru/design-patterns/builder/swift/example

5. **Prototype** (прототип) - Позволяет копировать объекты, не вдаваясь в подробности их реализации.
   https://refactoring.guru/ru/design-patterns/prototype/swift/example

Структурные паттерны

1. **Facade** - Предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку.

https://refactoring.guru/ru/design-patterns/facade/swift/example

2. **Decorator** - Позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки».

https://refactoring.guru/ru/design-patterns/decorator/swift/example

3. **Adapter** - Позволяет объектам с несовместимыми интерфейсами работать вместе.

https://refactoring.guru/ru/design-patterns/adapter/swift/example

4. **Bridge** - Разделяет один или несколько классов на две отдельные иерархии — абстракцию и реализацию, позволяя изменять их независимо друг от друга.

https://refactoring.guru/ru/design-patterns/bridge/swift/example

5. **Proxy** - Позволяет подставлять вместо реальных объектов специальные объекты-заменители. Эти объекты перехватывают вызовы к оригинальному объекту, позволяя сделать что-то до или после передачи вызова оригиналу.

https://refactoring.guru/ru/design-patterns/proxy/swift/example

6. **Composite** (компоновщик) - Позволяет сгруппировать объекты в древовидную структуру, а затем работать с ними так, как будто это единичный объект.

https://refactoring.guru/ru/design-patterns/composite/swift/example

7. **Flyweight** (легковес) - Позволяет вместить бóльшее количество объектов в отведенную оперативную память. Легковес экономит память, разделяя общее состояние объектов между собой, вместо хранения одинаковых данных в каждом объекте.

https://refactoring.guru/ru/design-patterns/flyweight/swift/example

Поведенческие паттерны

1. **Observer** (наблюдатель) - Создает механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.

https://refactoring.guru/ru/design-patterns/observer/swift/example

2. **Strategy** (стратегия) - Определяет семейство схожих алгоритмов и помещает каждый из них в собственный класс, после чего алгоритмы можно взаимозаменять прямо во время исполнения программы.

https://refactoring.guru/ru/design-patterns/strategy/swift/example

3. **Chain of Responsibility** (цепочка обязанностей) - Позволяет передавать запросы последовательно по цепочке обработчиков. Каждый последующий обработчик решает, может ли он обработать запрос сам и стоит ли передавать запрос дальше по цепи.

https://refactoring.guru/ru/design-patterns/chain-of-responsibility/swift/example

4. **Command** (команда) - Преобразует запросы в объекты, позволяя передавать их как аргументы при вызове методов, ставить запросы в очередь, логировать их, а также поддерживать отмену операций.

https://refactoring.guru/ru/design-patterns/command/swift/example

5. **Memento** (снимок) - Позволяет делать снимки состояния объектов, не раскрывая подробностей их реализации. Затем снимки можно использовать, чтобы восстановить прошлое состояние объектов.

https://refactoring.guru/ru/design-patterns/memento/swift/example

Поведенческие паттерны

6. **Iterator** - Даёт возможность последовательно обходить элементы составных объектов, не раскрывая их внутреннего представления.
https://refactoring.guru/ru/design-patterns/iterator/swift/example

7. **Mediato** (Посредник) - Позволяет уменьшить связанность множества классов между собой, благодаря перемещению этих связей в один класс-посредник.
https://refactoring.guru/ru/design-patterns/mediator/swift/example

8. **Visitor** (посетитель) - Позволяет создавать новые операции, не меняя классы объектов, над которыми эти операции могут выполняться.
https://refactoring.guru/ru/design-patterns/visitor/swift/example

9. **Template Method** (Шаблонный метод) - Определяет скелет алгоритма, перекладывая ответственность за некоторые его шаги на подклассы. Паттерн позволяет подклассам переопределять шаги алгоритма, не меняя его общей структуры.
https://refactoring.guru/ru/design-patterns/template-method/swift/example

10. **State** (Состояние) - Позволяет объектам менять поведение в зависимости от своего состояния. Извне создаётся впечатление, что изменился класс объекта.
https://refactoring.guru/ru/design-patterns/state/swift/example

Паттерны, с которыми мы уже знакомы

Delegate - реализовывается через протоколы у Apple. Пример: UITableViewDelegate, UITextFieldDelegate.

Singleton - обычно мы используем их через shared проперти в iOS разработке.

Адаптер - в swift этот паттерн реализуют протоколы. Также в качестве примера адаптера из пройденного материала можно привести конвертер данных.

Фасад - неплохим примером фасада являются наши менеджеры (NetworkManager, DataManager).

Декоратор - пример реализации в swift - любые extension.

Наблюдатель (observer), local push notifications: https://www.raywenderlich.com/21458686-local-notifications-getting-started

Все ссылки:

Сайт, на котором объяснение паттернов происходит через картинки и простые примеры практически в игровой форме. Playgrounds для паттернов. Я бы рекомендовал погрузится на недельку в этот сайт и попробовать все, что там описано:
https://refactoring.guru/ru/design-patterns/catalog

Шпаргалка по паттернам, полезно перед собеседованием, но изучать по этой статье было бы сложно:
https://stfalcon.com/ru/blog/post/ios-patterns

Та самая, легендарная книга:
https://docs.google.com/file/d/0B6GuCegBf3X3Tm1rZl9BUTduQm8/edit?resourcekey=0-ME3Ni9D9Wae8zLuAbNPx6w

Паттерны на примерах, репозиторий с кодом:
https://github.com/ochococo/Design-Patterns-In-Swift

Книга которую советовали мне для изучения паттернов когда я начинал:
https://fabulabook.com/product/paterny-proyektuvannya/?utm_source=Google%20Shopping&utm_campaign=Fabula&utm_medium=cpc&utm_term=18989&gclid=CjwKCAjwyIKJBhBPEiwAu7zll4jS2EfuO5VmvZUu13tLeRc9tmNq_rj4hAgroYYDRuHVNfnxK0j4ORoC2cEQAvD_BwE

Web Academy

CREATE YOUR IT FUTURE.

WITH WEB ACADEMY