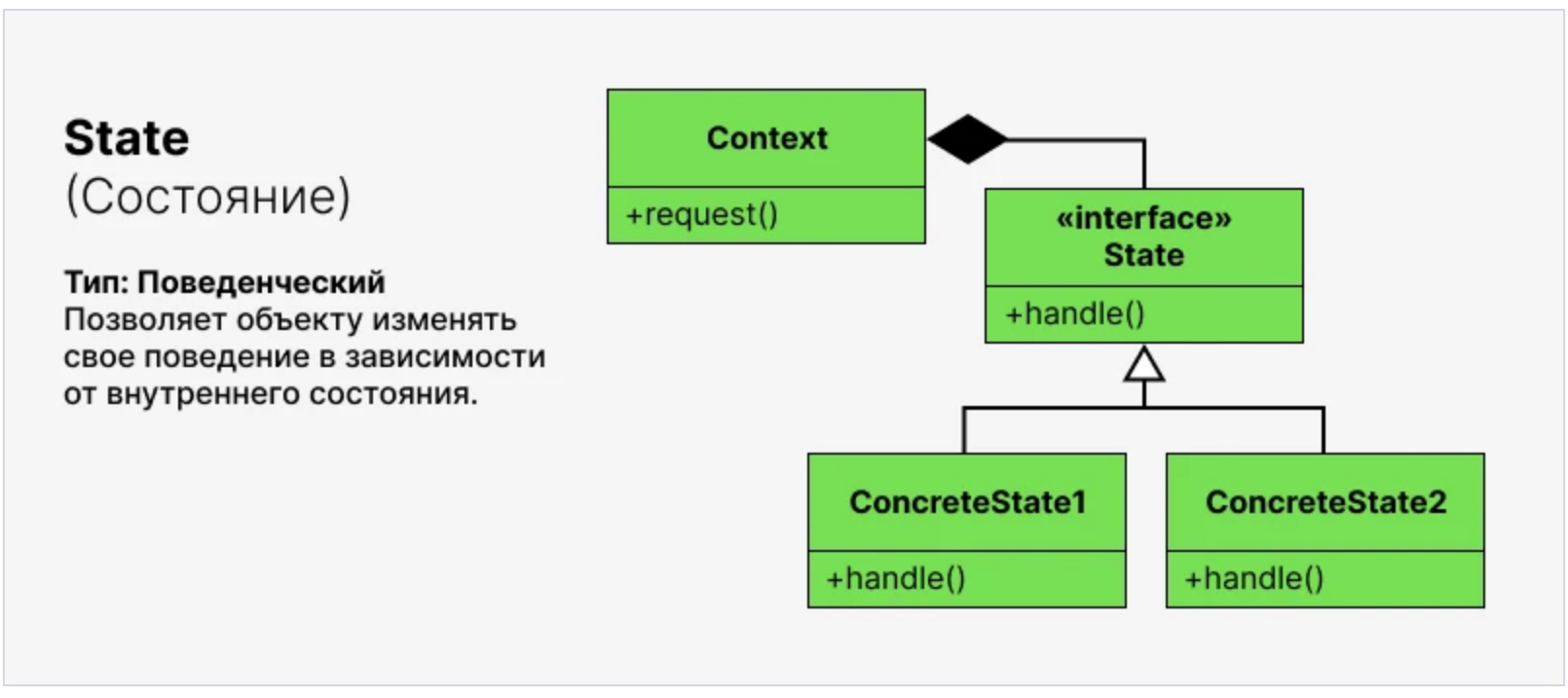
# Поведенческие паттерны, часть 2

JSP & Servlets
17 уровень, 1 лекция

ОТКРЫТА

## 2.1 State

**Состояние (State)** — поведенческий шаблон проектирования. Используется в тех случаях, когда во время выполнения программы объект должен менять своё поведение в зависимости от своего состояния.




Паттерн состоит из 3 блоков:

Context — класс, объекты которого должны менять своё поведение в зависимости от состояния.

State — интерфейс, который должен реализовать каждое из конкретных состояний. Через этот интерфейс объект Context взаимодействует с состоянием, делегируя ему вызовы методов. Интерфейс должен содержать средства для обратной связи с объектом, поведение которого нужно изменить.

Для этого используется **событие** (паттерн Publisher — Subscriber). Это необходимо для того, чтобы в процессе выполнения программы заменять объект состояния при появлении событий. Возможны случаи, когда сам Context периодически опрашивает объект состояния на наличие перехода.

ConcreteState1, ConcreteState2 — классы конкретных состояний. Должны содержать информацию о том, при каких условиях и в какие состояния может переходить объект из текущего состояния. Например, из ConcreteState1 объект может переходить в состояние ConcreteState2 и ConcreteState3, а из ConcreteState2 — обратно в ConcreteState1 и так далее. Объект одного из них должен содержать Context при создании.

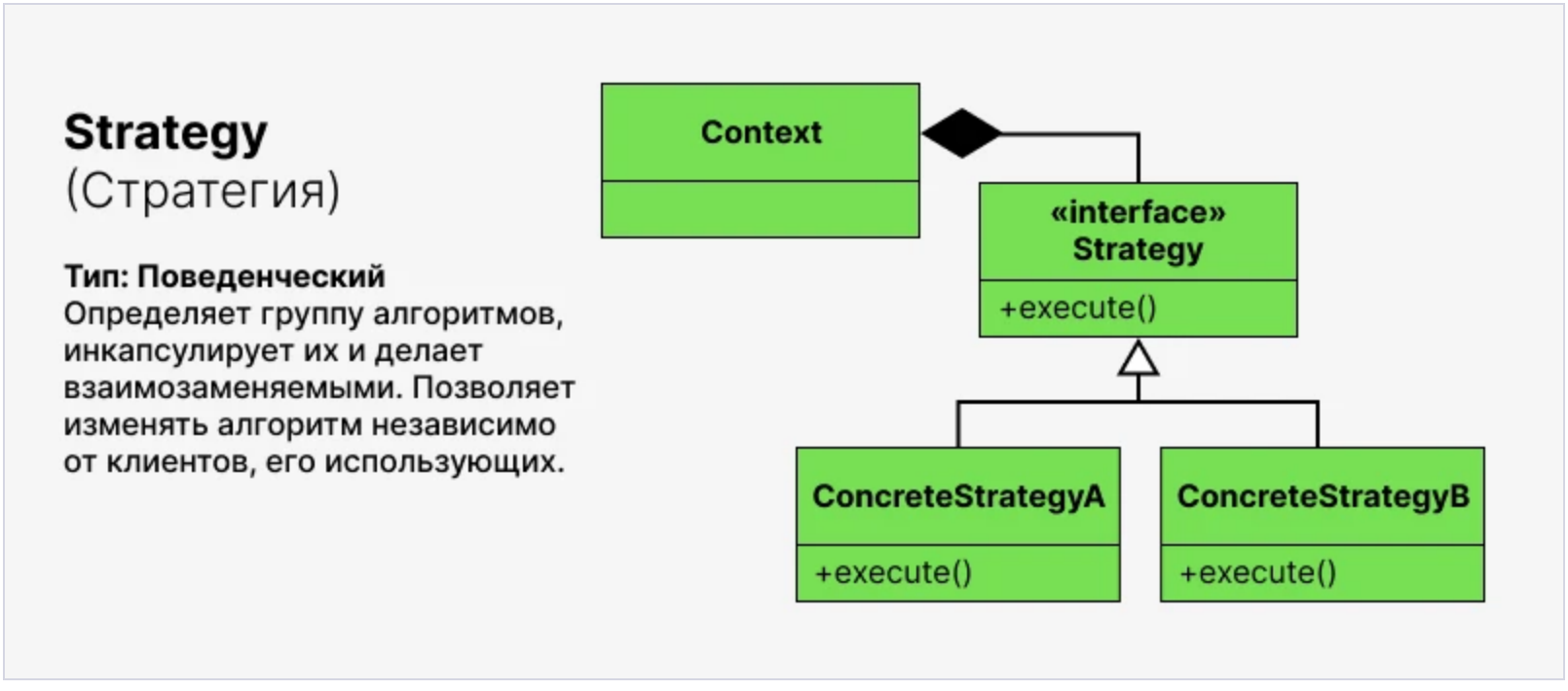
Например, ты пишишь игру, где персонаж может бегать, плавать и летать. Если твой персонаж попал в воду, то разумно ограничить его поведение в воде: он теперь не может стрелять, но у него сохранились какие-то действия: плыть вперед, вправо, влево и т.п.

Состояние твоего персонажа можно описать объектом State, у которого есть методы, которые можно вызывать и которые будут что-то делать. И после того, как твой персонаж залез в воду, ты просто меняешь у него внутри ссылку на другой объект State — и он меняет свое состояние.

## 2.2 Strategy

**Стратегия (Strategy)** — поведенческий шаблон проектирования, предназначенный для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм путём определения соответствующего класса.

Шаблон Strategy позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые его используют.



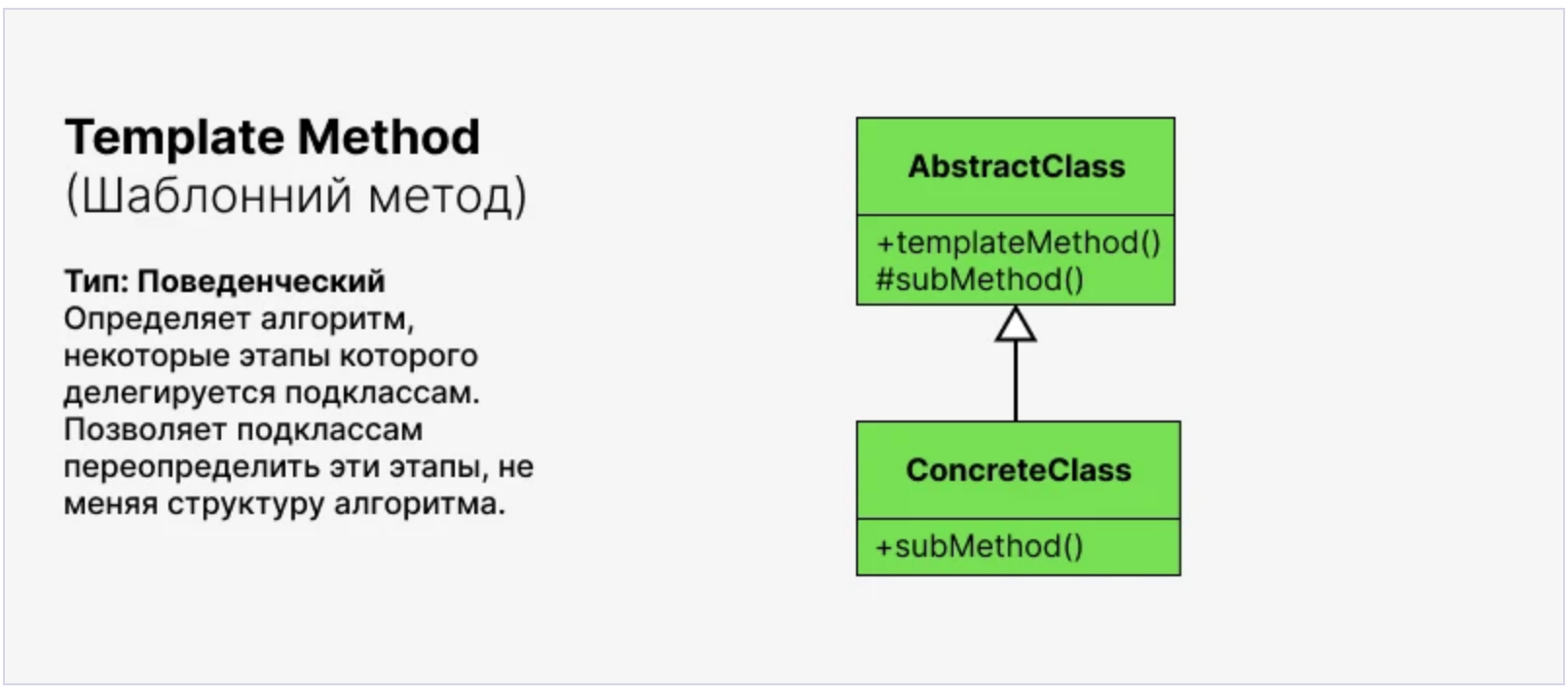

Паттерн Стратегия позволяет использовать различные бизнес-правила или алгоритмы в зависимости от контекста. Применяется в случаях, когда в одном и том же месте в зависимости от текущего состояния системы (или её окружения) должны использоваться различные алгоритмы.

**Сильные стороны:**

- инкапсуляция реализации различных алгоритмов, система становится независимой от возможных изменений бизнес-правил;
- вызов всех алгоритмов одним стандартным образом;
- отказ от использования переключателей и/или условных операторов.

Этот паттерн чем-то похож на паттерн State, однако тут акцент сделан не на состояние, а на поведение. Допустим, персонаж в твоей игре может менять оружие. Тогда при смене оружия можно просто поменять ссылку на объект, которые описывает, как это оружие работает.

## 2.3 Template Method




**Abstract class** (абстрактный класс) — определяет абстрактные операции, замещаемые в наследниках для реализации шагов алгоритма; реализует шаблонный метод, определяющий скелет алгоритма. Шаблонный метод вызывает замещаемые и другие, определенные в Abstract class, операции.

**Concrete class** (конкретный класс) — реализует замещаемые операции необходимым для реализации способом. Concrete class предполагает, что инвариантные шаги алгоритма будут выполнены в AbstractClass.

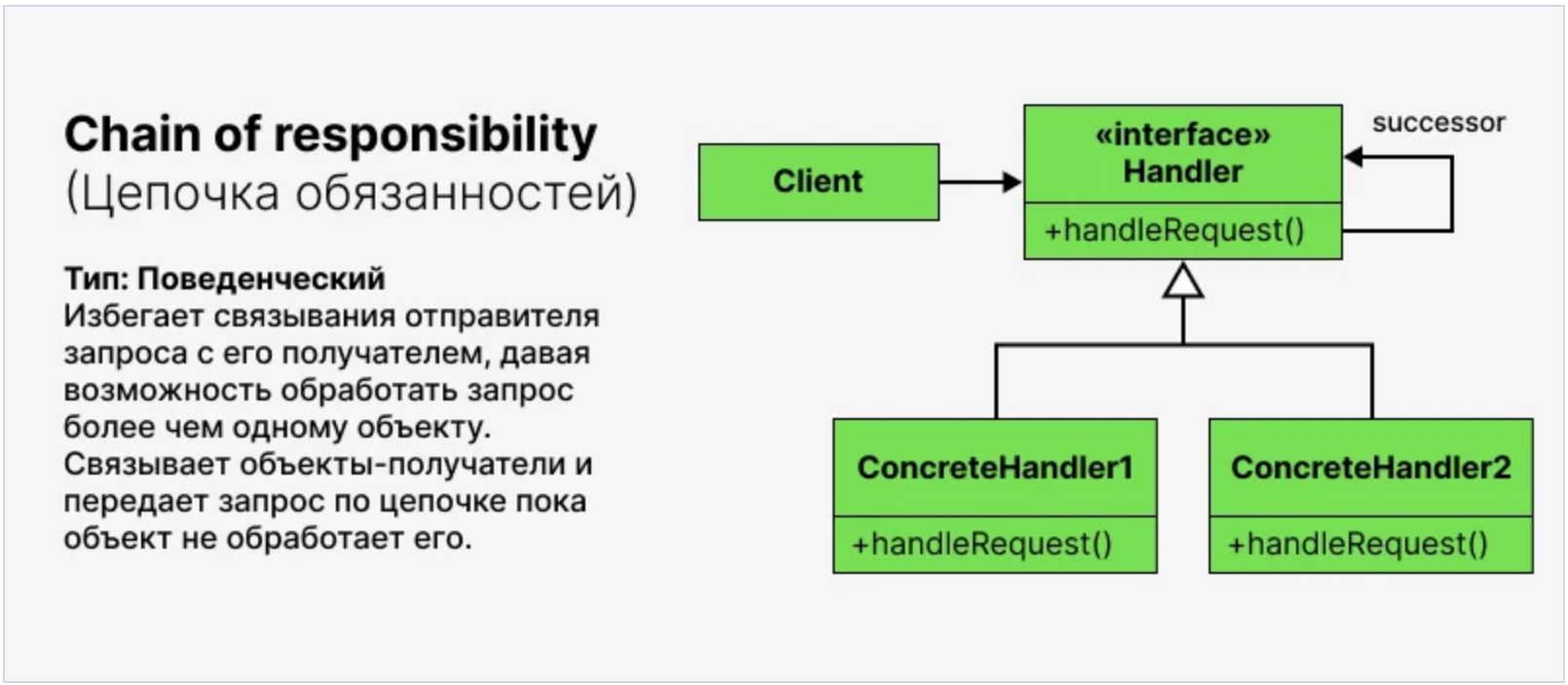
Этот паттерн часто используется, когда надо:

- Однократное использование инвариантной части алгоритма с оставлением изменяющейся части на усмотрение наследникам.
- Локализация и вычленение общего для нескольких классов кода для избегания дублирования.
- Разрешение расширения кода наследниками только в определенных местах.

Да, этот паттерн описывает использование пары: абстрактный класс и его реализация.

## 2.4 Chain of Responsibility

**Цепочка обязанностей (Chain of responsibility)** — поведенческий шаблон проектирования, предназначенный для организации в системе уровней ответственности

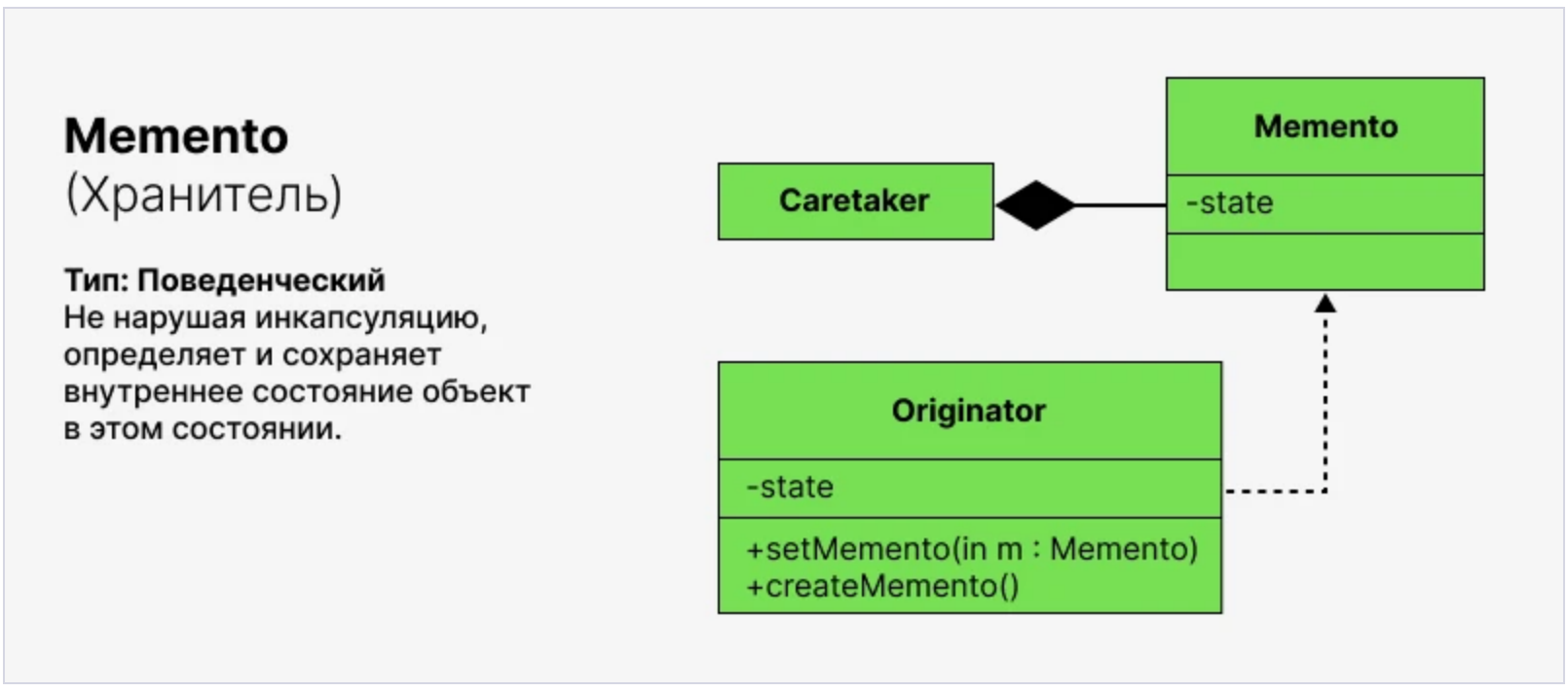



Шаблон рекомендован для использования в условиях, когда:

- в разрабатываемой системе имеется группа объектов, которые могут обрабатывать сообщения определенного типа;
- все сообщения должны быть обработаны хотя бы одним объектом системы;
- сообщения в системе обрабатываются по схеме “обработай сам или перешли другому”, то есть одни сообщения обрабатываются на том уровне, где они получены, а другие пересылаются объектам другого уровня.

## 2.5 Memento

**Хранитель (Memento)** — поведенческий шаблон проектирования, позволяющий не нарушая инкапсуляцию зафиксировать и сохранить внутреннее состояние объекта так, чтобы позднее восстановить его в это состояние.




Шаблон Хранитель используется, когда:

- необходимо сохранить снимок состояния объекта (или его части) для последующего восстановления;
- прямой интерфейс получения состояния объекта раскрывает детали реализации и нарушает инкапсуляцию объекта.

‹ Предыдущая лекция

Следующая лекция ›

+10

**Комментарии** популярные новые старые

**JavaCoder**

Введите текст комментария

У ЭТОЙ СТРАНИЦЫ ЕЩЕ НЕТ НИ ОДНОГО КОММЕНТАРИЯ

ЯЗЫК ИНТЕРФЕЙСА

Русский

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ

ДОСТУПНО В Google Play

Загрузите в App Store