Java Design Patterns

Strategy

Тема

Strategy

Суть паттерна

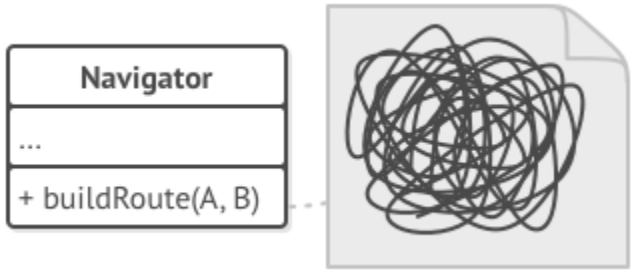
Стратегия

Стратегия — это поведенческий паттерн проектирования, который определяет семейство схожих алгоритмов и помещает каждый из них в собственный класс. После чего, алгоритмы можно взаимно заменять прямо во время исполнения программы.

Проблема

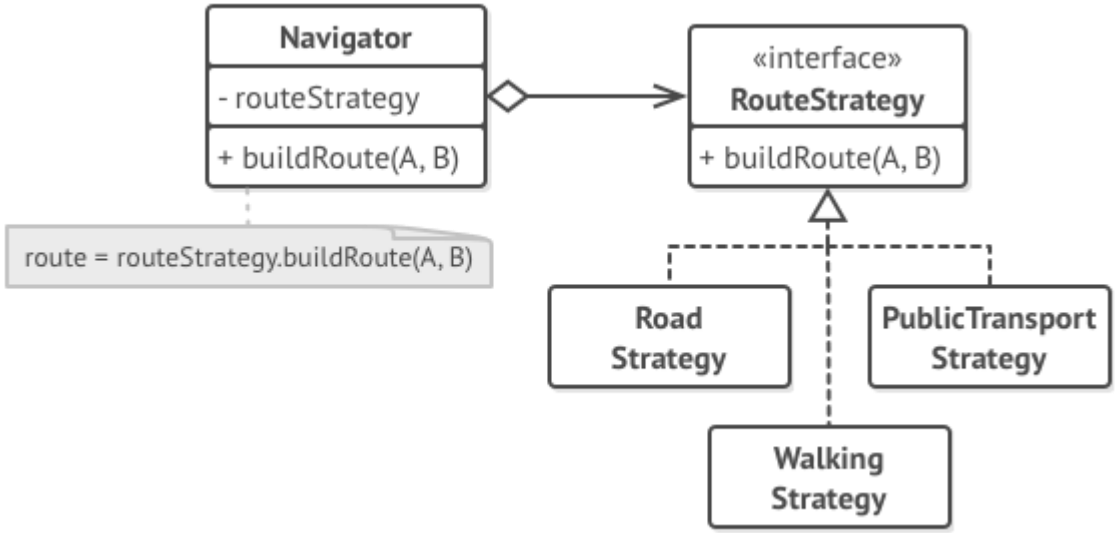
Постановка задачи

Вы решили написать приложение-навигатор для путешественников. Он должен показывать красивую и удобную карту, позволяющую с лёгкостью ориентироваться в незнакомом городе.



Решение задачи

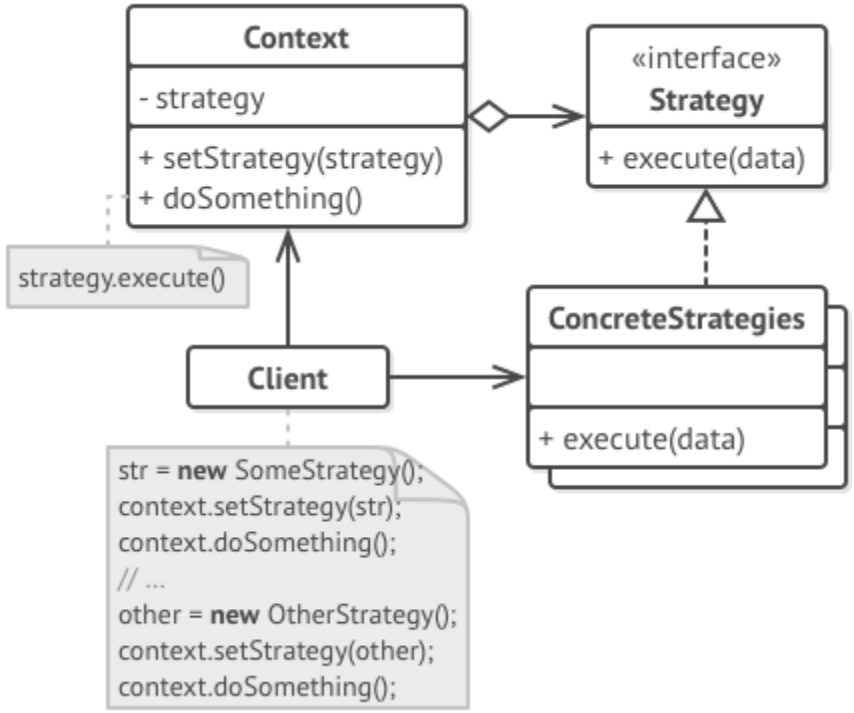
Паттерн Стратегия предлагает определить семейство схожих алгоритмов, которые часто изменяются или расширяются, и вынести их в собственные классы, называемые стратегиями.



Структура

Структура паттерна

1. Контекст хранит ссылку на объект конкретной стратегии, работая с одним объектом через общий интерфейс стратегий
2. Стратегия определяет интерфейс, общий для всех вариаций алгоритма. Контекст использует этот интерфейс для вызова алгоритма.
3. Конкретные стратегии реализуют различные вариации алгоритма.
4. Во время выполнения программы, контекст получает вызовы от клиента и делегирует их объекту конкретной стратегии.
5. Обычно, клиент должен создать объект конкретной стратегии и передать его в контекст: либо через конструктор, либо в какой-то другой решающий момент, используя сеттер. Благодаря этому, контекст не знает о том, какая именно стратегия сейчас выбрана.



Применимость

Применение паттерна

1. Когда вам нужно использовать разные вариации какого-то алгоритма внутри одного объекта.
2. Когда у вас есть множество похожих классов, отличающихся только некоторым поведением.
3. Когда вы не хотите обнажать детали реализации алгоритмов для других классов.
4. Когда различные вариации алгоритмов реализованы в виде развесистого условного оператора. Каждая ветка такого оператора представляет вариацию алгоритма.

Шаги реализации

Алгоритм реализации паттерна

1. Определите алгоритм, который подвержен частым изменениям. Также подойдёт алгоритм, имеющий несколько вариаций, которые выбираются во время выполнения программы.
2. Создайте интерфейс стратегий, описывающий этот алгоритм. Он должен быть общим для всех вариантов алгоритма.
3. Поместите вариации алгоритма в собственные классы, которые реализуют этот интерфейс.
4. В классе контекста создайте поле для хранения ссылки на текущий объект-стратегию, а также метод для её изменения. Убедитесь в том, что контекст работает с этим объектом только через общий интерфейс стратегий.
5. Клиенты контекста должны подавать в контекст соответствующий объект-стратегию, когда хотят, чтобы тот вёл себя определённым образом.

Плюсы и недостатки

Плюсы:

- Горячая замена алгоритмов на лету.
- Изолирует код и данные алгоритмов от остальных классов.
- Уход от наследования к делегированию.
- Реализует *принцип открытости/закрытости*.

Минусы:

- Усложняет программу за счёт дополнительных классов.
- Клиент должен знать, в чём разница между стратегиями, чтобы выбрать подходящую.

Отношение с другими паттернами

- Мост, Стратегия и Состояние (а также слегка и Адаптер) имеют схожие структуры классов — все они построены на принципе «композиции», то есть делегирования работы другим объектам. Тем не менее, они отличаются тем, что решают разные проблемы. Помните, что паттерны — это не только рецепт построения кода определённым образом, но и описание проблем, которые привели к данному решению.
- Команда и Стратегия похожи по духу, но отличаются масштабом и применением:
  - *Команду* используют, чтобы превратить любые разнородные действия в объекты. Параметры операции превращаются в поля объекта. Этот объект теперь можно логировать, хранить в истории для отмены, передавать во внешние сервисы и так далее.
  - С другой стороны, *Стратегия* описывает разные способы сделать одно и то же действие, позволяя взаимозаменять эти способы в каком-то объекте контекста.
- Стратегия меняет поведение объекта «изнутри», а Декоратор изменяет его «снаружи».
- Шаблонный метод использует наследование, чтобы расширять части алгоритма. Стратегия использует делегирование, чтобы изменять алгоритм. *Шаблонный метод* работает на уровне классов. *Стратегия* позволяет менять логику отдельных объектов.
- Состояние можно рассматривать как надстройку над Стратегией. Оба паттерна используют композицию, чтобы менять поведение основного объекта, делегируя работу вложенным объектам-помощникам. Однако в *Стратегии* эти объекты не знают друг о друге и никак не связаны. В *Состоянии* сами конкретные состояния могут переключать контекст.

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association