# State

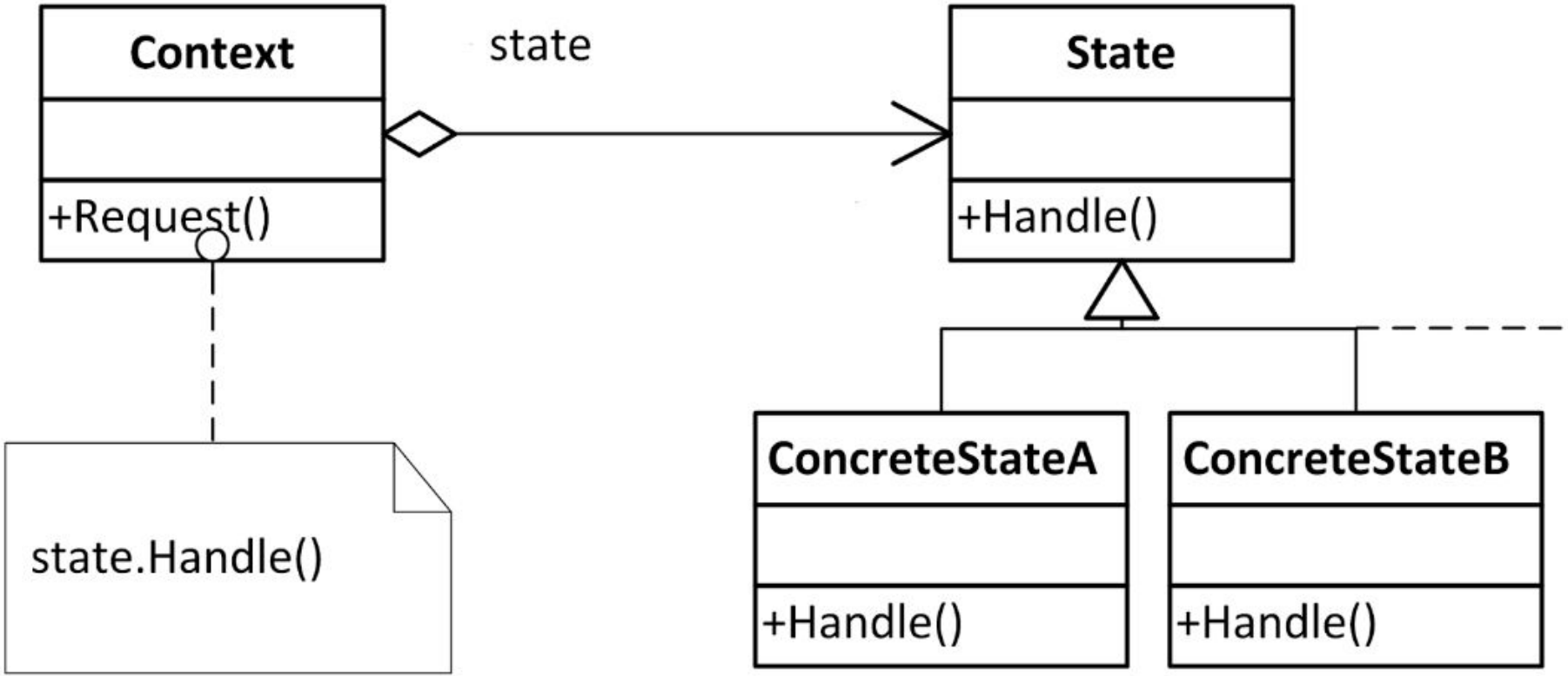
# Мотивация

Основная идея паттерна в том, чтобы ввести абстрактный класс State для представления различных состояний.

При этом паттерн позволяет объекту варьировать свое поведение в зависимости от внутреннего состояния.
Извне создается впечатление, что изменился класс объекта.

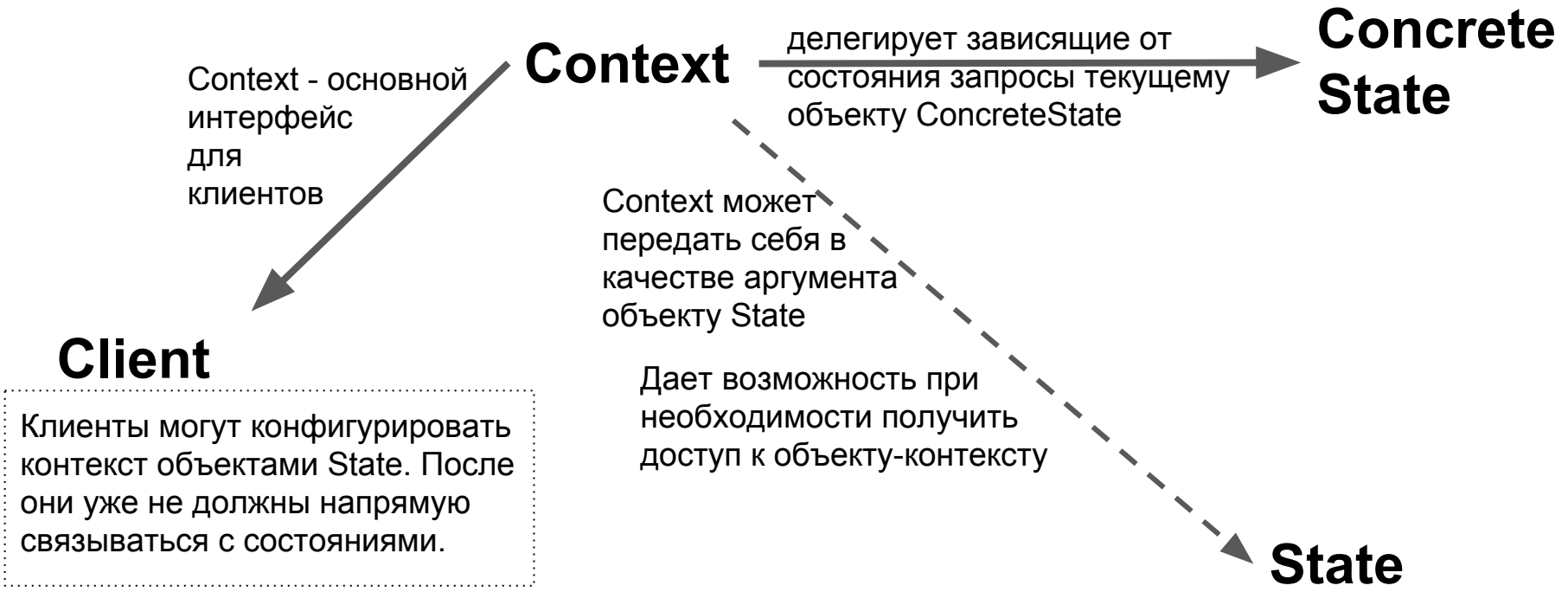
# Когда используем?

- Когда поведение объекта должно зависеть от его состояния и может изменяться динамически во время выполнения
- Когда возникает много условий


Context
+Request()
state
State
+Handle()
state.Handle()
ConcreteStateA
+Handle()
ConcreteStateB
+Handle()

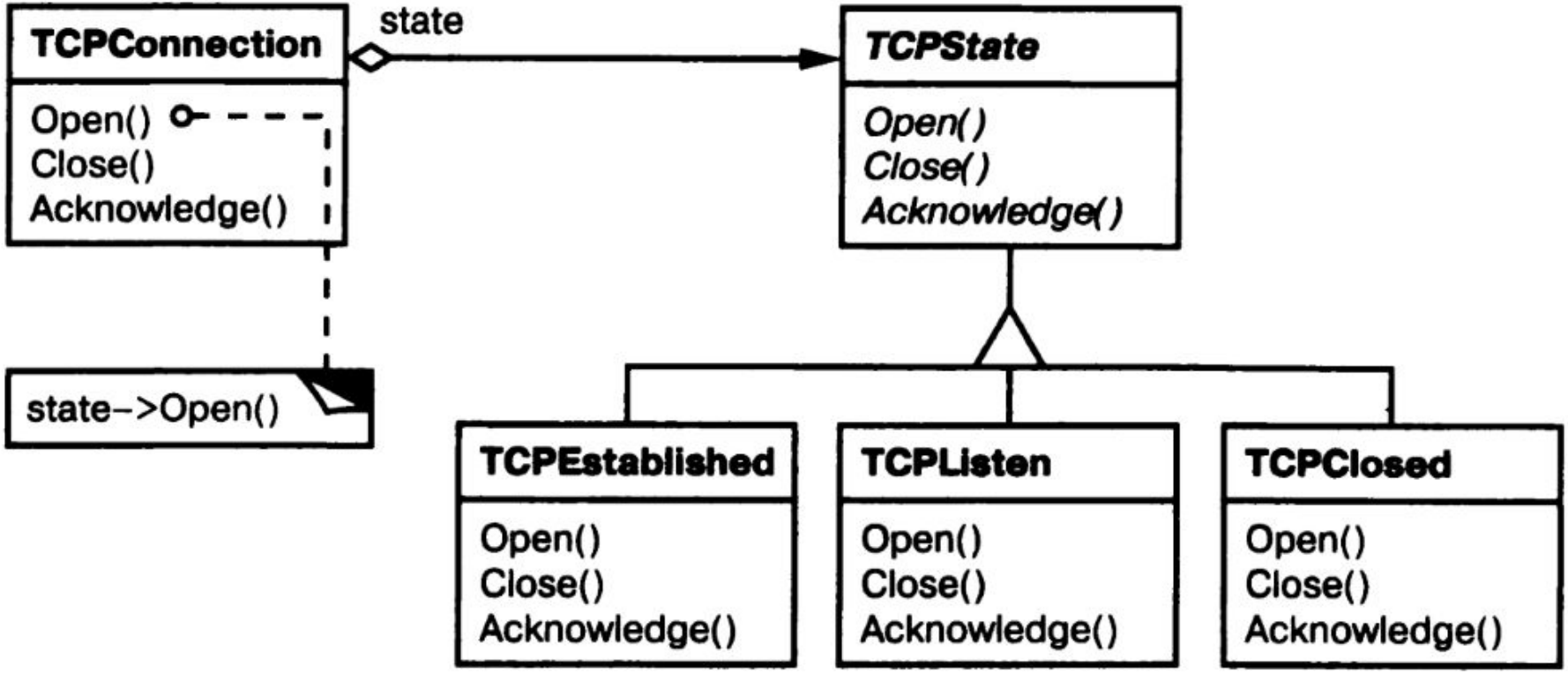
# Участники:

- Context
  - интерфейс для клиентов
  - хранит экземпляр ConcreteState, которым определяется текущее состояние
- State
  - интерфейс для инкапсуляции поведения, ассоциированного с конкретным состоянием контекста
- Подклассы ConcreteState
  - реализация поведения, ассоциированного с некоторым состоянием Context


Context
Context - основной интерфейс для клиентов
делегирует зависящие от состояния запросы текущему объекту ConcreteState
Concrete State
Context может передать себя в качестве аргумента объекту State
Дает возможность при необходимости получить доступ к объекту-контексту
State
Client
Клиенты могут конфигурировать контекст объектами State. После они уже не должны напрямую связываться с состояниями.

# Пример

- Класс TCPConnection - Context
- Состояния - ConcreteState
  - Established(установлено)
  - Listening(прослушивание)
  - Closed(закрыто)
- Интерфейс TCPState - State
  - Open()
  - Close()
  - Acknowledge()


TCPConnection
Open()
Close()
Acknowledge()
state
TCPState
Open()
Close()
Acknowledge()
state->Open()
TCPEstablished
Open()
Close()
Acknowledge()
TCPListen
Open()
Close()
Acknowledge()
TCPClosed
Open()
Close()
Acknowledge()

# Вопросы реализации

- Что определяет переходы между состояниями?
- Как создавать и удалять объекты?

# Переходы между состояниями

- В целом, более гибким и правильным подходом будет позволить самим подклассам State определять следующее состояние и момент перехода
  - добавляем в Context интерфейс, позволяющий объектам State установить состояние контекста
- Недостаток: каждый из подклассов должен знать еще хотя бы об одном подклассе

# Результаты

- локализует зависящее от состояния поведение и делит его на части, соответствующие состояниям
  - логика распределяется между подклассами State; больше не заключена в монолитные операторы if и switch
- делает явными переходы между состояниями
- объекты состояния можно разделять
  - State становится приспособленцем, у которого нет внутреннего состояния, а есть только поведение

# Минус

- При большом количестве состояний - большое количество классов

# Родственные паттерны

- Flyweight (Приспособленец)
  - подсказывает, как и когда можно разделять объекты класса State
- Singleton (Одиночка)
  - объекты State часто бывают одиночками