# Строитель (builder)

## Суть паттерна

**Строитель** — это порождающий паттерн проектирования, который позволяет создавать сложные объекты пошагово. Строитель даёт возможность использовать один и тот же код строительства для получения разных представлений объектов.

## Проблема

Представьте сложный объект, требующий кропотливой пошаговой инициализации множества полей и вложенных объектов. Код инициализации таких объектов обычно спрятан внутри монструозного конструктора с десятком параметров. Либо ещё хуже — распылён по всему клиентскому коду.

Чтобы не плодить подклассы, вы можете создать гигантский конструктор, принимающий уйму параметров для контроля над создаваемым продуктом. Действительно, это избавит вас от подклассов, но приведёт к другой проблеме.

## Решение

Паттерн Строитель предлагает вынести конструирование объекта за пределы его собственного класса, поручив это дело отдельным объектам, называемым *строителями*.

Паттерн предлагает разбить процесс конструирования объекта на отдельные шаги. Чтобы создать объект, вам нужно поочерёдно вызывать методы строителя. Причём не нужно запускать все шаги, а только те, что нужны для производства объекта определённой конфигурации.

Зачастую один и тот же шаг строительства может отличаться для разных вариаций производимых объектов. Например, деревянный дом потребует строительства стен из дерева, а каменный — из камня.

В этом случае вы можете создать несколько классов строителей, выполняющих одни и те же шаги по-разному. Используя этих строителей в одном и том же строительном процессе, вы сможете получать на выходе различные объекты.

***Директор***

Вы можете пойти дальше и выделить вызовы методов строителя в отдельный класс, называемый *директором*. В этом случае директор будет задавать порядок шагов строительства, а строитель — выполнять их.

Отдельный класс директора не является строго обязательным. Вы можете вызывать методы строителя и напрямую из клиентского кода. Тем не менее, директор полезен, если у вас есть несколько способов конструирования

продуктов, отличающихся порядком и наличием шагов конструирования. В этом случае вы сможете объединить всю эту логику в одном классе.

Такая структура классов полностью скроет от клиентского кода процесс конструирования объектов. Клиенту останется только привязать желаемого строителя к директору, а затем получить у строителя готовый результат.

# Структура

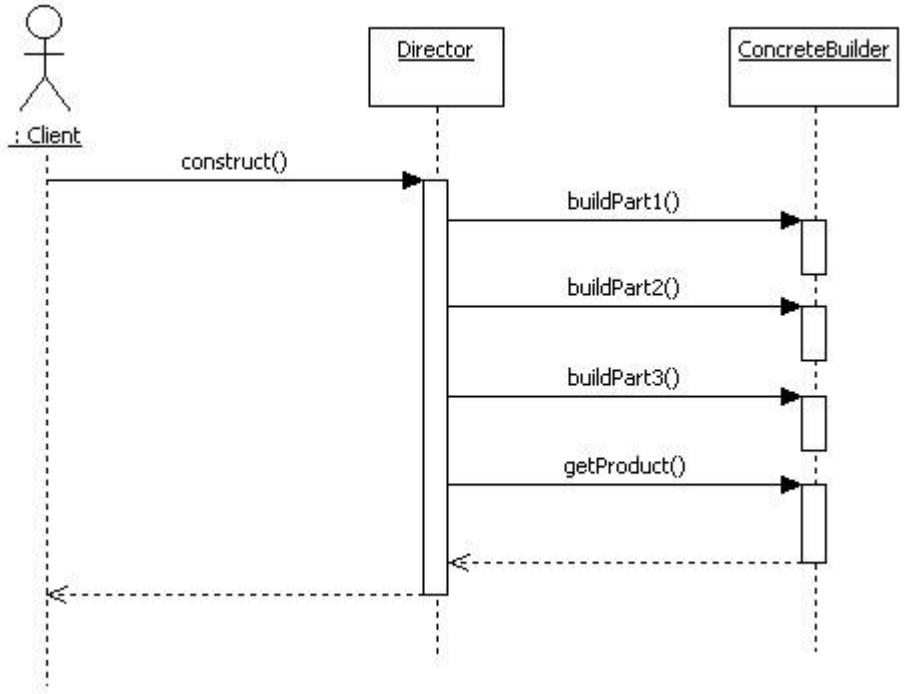
**Интерфейс строителя** объявляет шаги конструирования продуктов, общие для всех видов строителей.

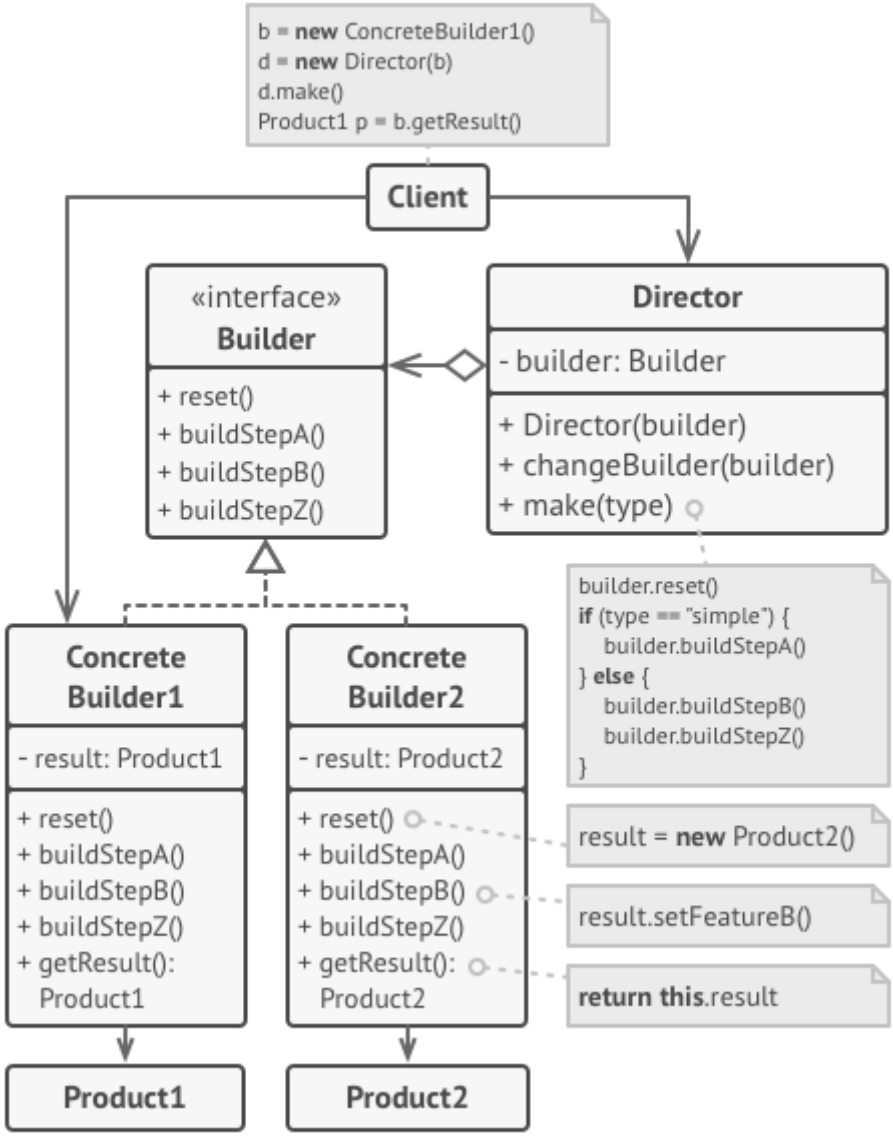
**Конкретные строители** реализуют строительные шаги, каждый по-своему. Конкретные строители могут производить разнородные объекты, не имеющие общего интерфейса.

**Продукт** — создаваемый объект. Продукты, сделанные разными строителями, не обязаны иметь общий интерфейс.

**Директор** определяет порядок вызова строительных шагов для производства той или иной конфигурации продуктов.

Обычно **Клиент** подаёт в конструктор директора уже готовый объект-строитель, и в дальнейшем данный директор использует только его. Но возможен и другой вариант, когда клиент передаёт строителя через параметр строительного метода директора. В этом случае можно каждый раз применять разных строителей для производства раз личных представлений объектов.






## Преимущества:

- Позволяет создавать продукты пошагово.
- Позволяет использовать один и тот же код для создания различных продуктов.
- Изолирует сложный код сборки продукта от его основной бизнес-логики.

## Недостатки:

- Усложняет код программы из-за введения дополнительных классов.
- Клиент будет привязан к конкретным классам строителей, так как в интерфейсе строителя может не быть метода получения результата.