Паттерн “Строитель”(“Builder”)

НАЗНАЧЕНИЕ:

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.

Паттерн “Строитель”(“Builder”)

МОТИВАЦИЯ:

Используйте “Строитель”, чтобы избавиться от гигантских конструкторов с десятками параметров. Он делает код создания объекта пошаговым, понятным и защищенным от ошибок.

“Строитель” позволяет создавать разные представления одного и того же сложного объекта, управляя процессом его конструирования пошагово. Один и тот же процесс строительства может создать совершенно разные продукты.

Паттерн "Строитель" — это как сборка бургера: вы говорите "добавь котлету", "добавь сыр", "не клади лук", и в результате получаете нужный вам продукт, не вникая в детали кухни.

Паттерн “Строитель”(“Builder”)

ТАК ПЛОХО

```cpp
#include <iostream>

class Vehicle {
public:
    Vehicle(const std::string& name, const std::string&
body, int numwheels, int numdoors, bool caterpillars) {

//Параметров много. А в ином случае может быть и еще
больше.

        this->name = name;
        this->body = body;
        this->numwheels = numwheels;
        this->numdoors = numdoors;
        this->caterpillars = caterpillars;
    }

    void showVehicle() const {
        std::cout << "Vehicle: " << name << std::endl <<
std::endl;
        std::cout << "body: " << body << "," <<
"numwheels: " << numwheels << "," << "numdoors: " <<
numdoors << "," << "caterpillars: " << (caterpillars ?
"yes" : "no") << "." << std::endl;
    }

private:
    std::string name;
    std::string body;
    int numwheels;
    int numdoors;
    bool caterpillars;
};
```

```cpp
int main() {

    // ПЛОХО: Длинные конструкторы с множеством параметров
(представьте что их не 5, а 20)

    Vehicle excavator("Catpill 12309", "Excavator body", 20, 2,
true);
    Vehicle moto("Honda CBR1000", "Moto body", 2, 0, false);

    // Еще хуже: если нужно создать объект по частям. Приходится
использовать временные переменные или создавать неполный объект


    std::string name = "Kamaz";
    std::string body = "Truck body";
    int wheels = 6;
    int doors = 2;
    bool caterpillars = false;
    Vehicle truck(name, body, wheels, doors, caterpillars);

    excavator.showVehicle();
    std::cout << std::endl;
    moto.showVehicle();
    std::cout << std::endl;
    truck.showVehicle();
}
```

```
body: Excavator body,numwheels: 20,numdoors: 2,caterpillars: yes.

Vehicle: Honda CBR1000

body: Moto body,numwheels: 2,numdoors: 0,caterpillars: no.

Vehicle: Kamaz

body: Truck body,numwheels: 6,numdoors: 2,caterpillars: no.
```

Паттерн “Строитель”(“Builder”)

Проблемы без паттерна “Строитель”:

1)Длинные конструкторы - много параметров, легко перепутать порядок.

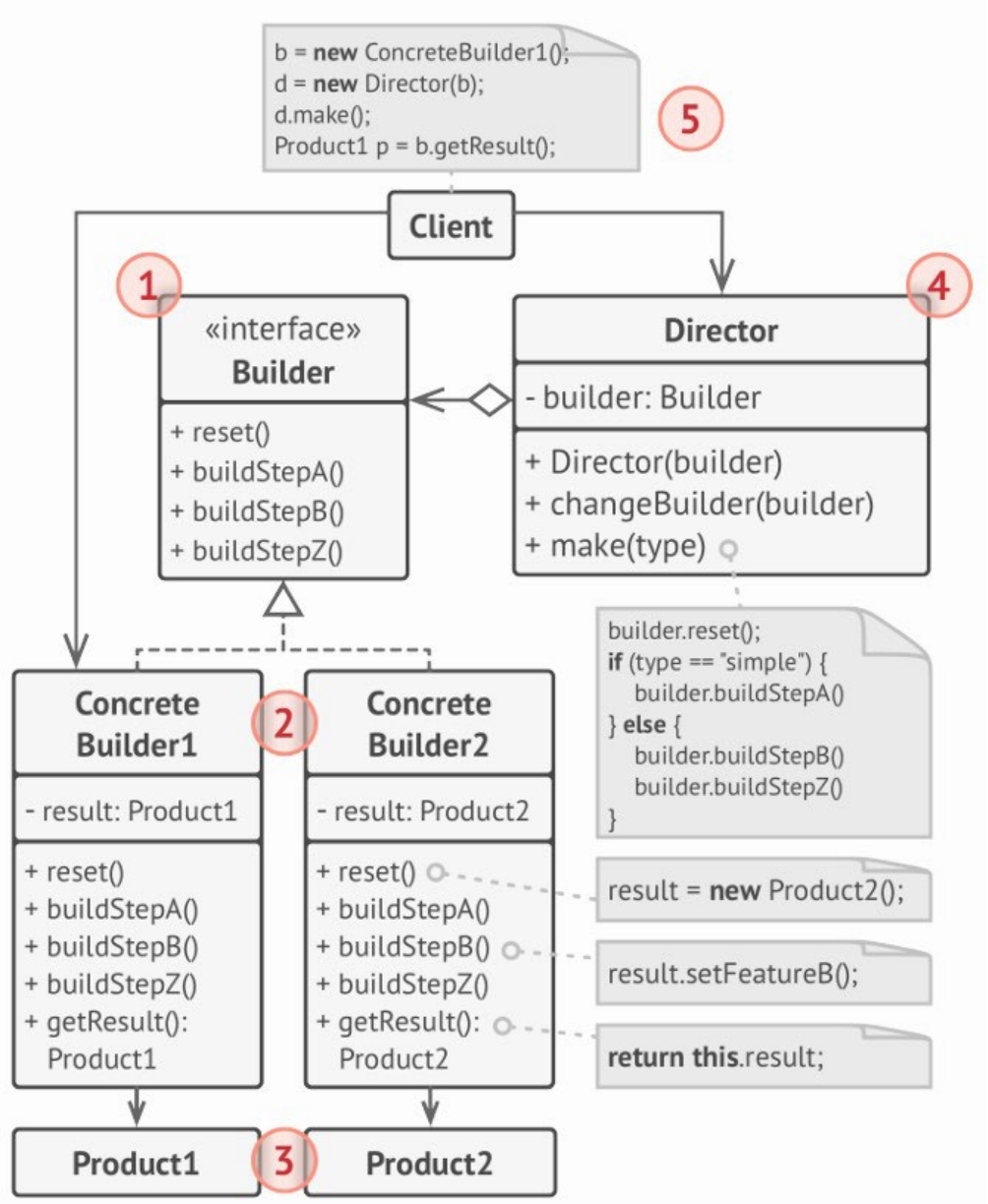
2)Невозможность пошагового создания - все параметры должны быть известны сразу.

3)Сложность чтения кода - непонятно, что означают числа 20, 2, true в конструкторе.

4)Отсутствие гибкости - нельзя менять процесс создания для разных вариантов объекта.

“Строитель”: структура

UML – общая структура



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

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

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

4.Директор определяет порядок вызова строительных шагов для производства той или иной конфигурации объектов.

5.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 различных представлений объектов.

“Строитель”: пример использования