Строитель

Назначение

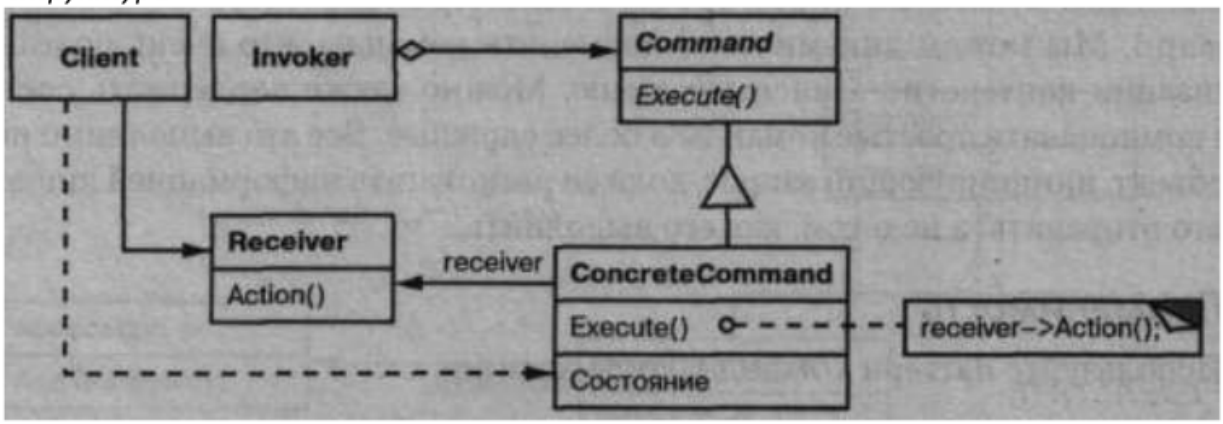
Инкапсулирует запрос как объект, позволяя тем самым задавать параметры клиентов для обработки
соответствующих запросов, ставить запросы в очередь или протоколировать их, а также поддерживать отмену
операций.

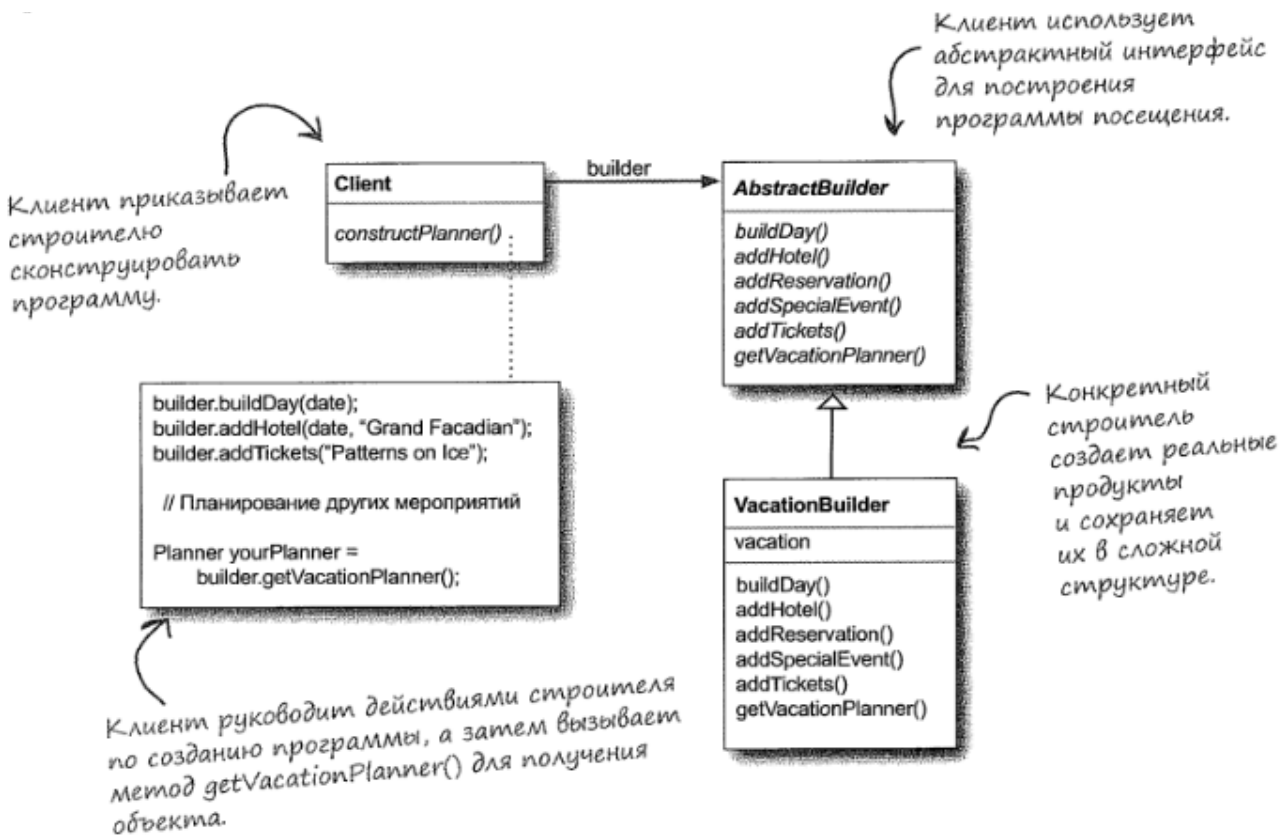
Применимость

Используйте паттерн команда, когда хотите:

- Параметризовать объекты выполняемым действием, как в случае с пунктами меню.
- Определять, ставить в очередь и выполнять запросы в разное время.
- Поддержать отмену операций. Операция Execute объекта Command может сохранить состояние, необходимое для отката действий, выполненных командой.
- Поддержать протоколирование изменений, чтобы их можно было выполнить повторно после аварийной остановки системы.
- Структурировать систему на основе высокоуровневых операций, построенных из примитивных.

Структура





Преимущества Строителя

- •Инкапсуляция процесса создания сложного объекта.
- •Возможность поэтапного конструирования объекта с переменным набором этапов (в отличие от «одноэтапных» фабрик).
- •Сокрытие внутреннего представления продукта от клиента.
- •Реализации продуктов могут свободно изменяться, потому что клиент имеет дело только с абстрактным интерфейсом.

г- Область применения и недостатки –

- •Часто используется для создания составных структур.
- Конструирование объектов с использованием паттерна Фабрика требует от клиента дополнительных знаний предметной области.

Результаты применения паттерна команда таковы:

- □ Команда разрывает связь между объектом, инициирующим операцию, и объектом, имеющим информацию о том, как ее выполнить;
- □ Команды - это самые настоящие объекты. Допускается манипулировать ими и расширять их точно так же, как в случае с любыми другими объектами;
- □ Из простых команд можно собирать составные. В общем случае составные команды описываются паттерном компоновщик;
- □ Добавлять новые команды легко, поскольку никакие существующие классы изменять не нужно.

```cpp
#include <iostream>
class Pizza
{
private:
    std::string dough;
    std::string sauce;
    std::string topping;
public:
    Pizza() { }
    ~Pizza() { }
    void SetDough(const std::string& d) { dough = d; }
    void SetSauce(const std::string& s) { sauce = s; }
    void SetTopping(const std::string& t) { topping = t; }
    void ShowPizza(){
        std::cout << " Yummy !!!" << std::endl
                  << "Pizza with Dough as " << dough
                  << ", Sauce as " << sauce
                  << " and Topping as " << topping
                  << " !!! " << std::endl;
    }
};
// Abstract Builder
class PizzaBuilder
{
protected:
    Pizza* pizza;
public:
    PizzaBuilder() {}
    virtual ~PizzaBuilder() {}
    Pizza* GetPizza() { return pizza; }
    void createNewPizzaProduct() { pizza=new Pizza; }
    virtual void buildDough()=0;
    virtual void buildSauce()=0;
    virtual void buildTopping()=0;
};
// ConcreteBuilder
class HawaiianPizzaBuilder : public PizzaBuilder
{
public:
    HawaiianPizzaBuilder() : PizzaBuilder() {}
    ~HawaiianPizzaBuilder(){}
    void buildDough() { pizza->SetDough("cross"); }
    void buildSauce() { pizza->SetSauce("mild"); }
    void buildTopping() { pizza->SetTopping("ham and pineapple"); }
};
// ConcreteBuilder
class SpicyPizzaBuilder : public PizzaBuilder
{
public:
    SpicyPizzaBuilder() : PizzaBuilder() {}
    ~SpicyPizzaBuilder() {}
    void buildDough() { pizza->SetDough("pan baked"); }
    void buildSauce() { pizza->SetSauce("hot"); }
    void buildTopping() { pizza->SetTopping("pepperoni and salami"); }
```

```
};
class Waiter
{
private:
    PizzaBuilder* pizzaBuilder;
public:
    Waiter() : pizzaBuilder(NULL) {}
    ~Waiter() { }
    void SetPizzaBuilder(PizzaBuilder* b) { pizzaBuilder = b; }
    Pizza* GetPizza() {  return pizzaBuilder->GetPizza(); }
    void ConstructPizza()
    {
        pizzaBuilder->createNewPizzaProduct();
        pizzaBuilder->buildDough();
        pizzaBuilder->buildSauce();
        pizzaBuilder->buildTopping();
    }
};
int main() {
    Waiter waiter;
    HawaiianPizzaBuilder hawaiianPizzaBuilder;
    waiter.SetPizzaBuilder (&hawaiianPizzaBuilder);
    waiter.ConstructPizza();
    Pizza* pizza = waiter.GetPizza();
    pizza->ShowPizza();
    SpicyPizzaBuilder spicyPizzaBuilder;
    waiter.SetPizzaBuilder(&spicyPizzaBuilder);
    waiter.ConstructPizza();
    pizza = waiter.GetPizza();
    pizza->ShowPizza();
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```