Strategy Pattern

Дефинира семейство от алгоритми, енкапсулира всеки един и след това ги прави взаимнозаменяеми. Този патерн позволява на алгоритъма да се променя независимо от клиента който го използва.

Описание:

Енкапсулираме алгоритми в клас и правим всеки алгоритъм заменим. Добавяме интерфейс strategy и алгоритмите наследяват общия интерфейс, а всеки алгоритъм прави нещата си по различен начин. Клиента работи само с интерфейса и му се подпъхват различни алгоритми.

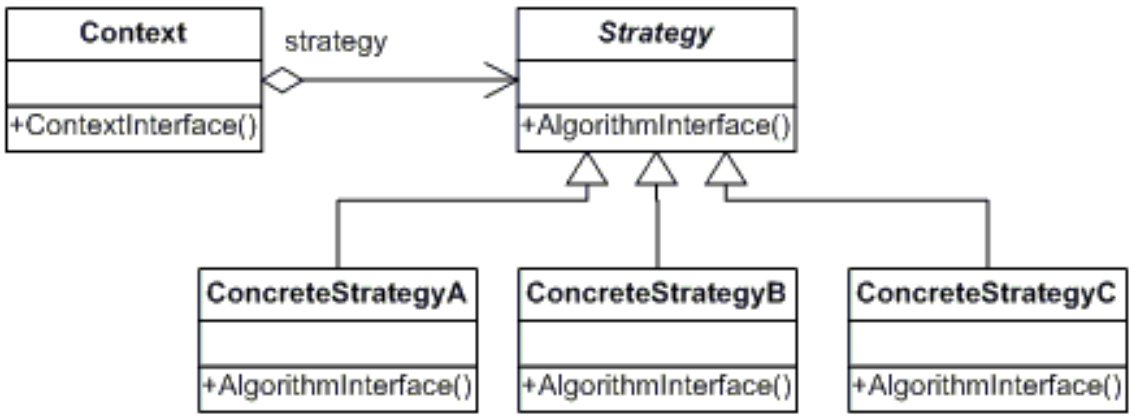
- Подобно на Command design pattern енкапсулира някакъв вид държание отделено в клас.
- Работи с различни имплементации на алгоритъма, но всички вършат една и съща работа.
- Включва и използва медоти, които трябва да бъдат имплементирани в наследниците.
- Ексапсулира алгоритъм в клас
- Прави всеки алгоритъм заменим
- Всички алгоритми могат да работят с едни и същи данни по прозрачен начин.
- Клиентът може да работи по прозрачен начин с всеки алгоритъм.

Примери:

Checker системата в BGCoder: Check(ICheckerStrategy checker) - TrimChecker, SortChecker, ExactChecker

Избор на подходящи алгоритми за търсене и сортирани, които прилагат като класа наследява интерфейс (ISortStrategy например)

ULM Диаграма:



Façade Pattern

"Осигуряване на единен интерфейс за набор от интерфейси в подсистема, Фасада дефинира интерфейс на по-високо ниво, такъв че прави подсистемата по-лесна за използване. "

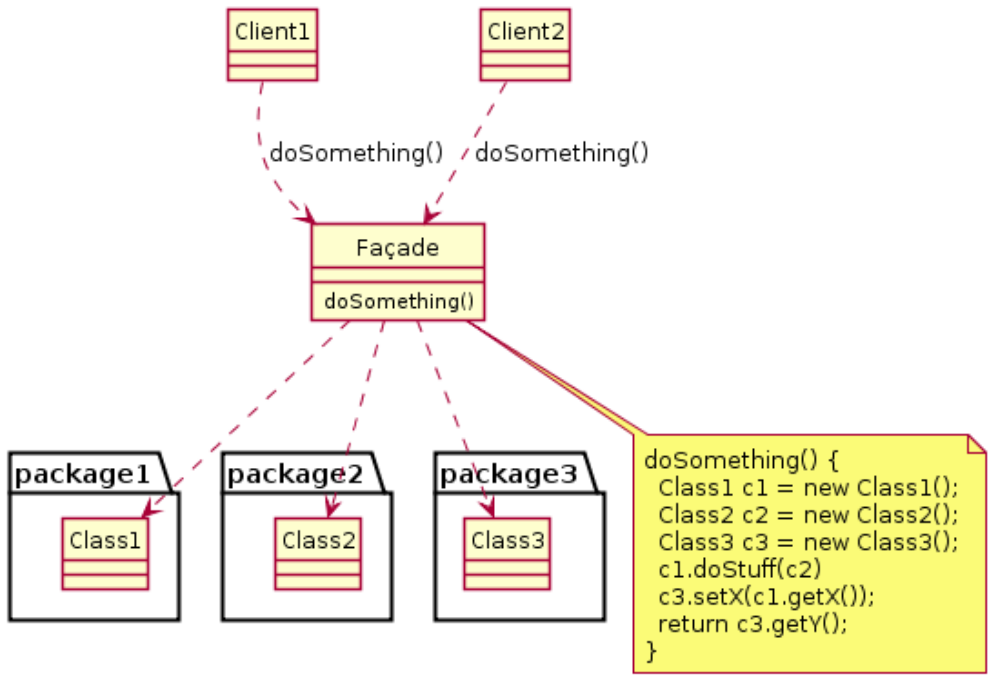
Описание:

- Създава изкуствен клас или интерфейс, който скрива сложните неща и всички външни неща комуникират само с фасадата
- Използва се за да отдели и опрости дадена функционалност от набор от сложни класове.
- Използвайки този патерн клиента може да използва по-прости интерфейси, но ако все пак му е нужно той пак може да използва цялата функционалност която подсистемата му дава, тоест може да я достъпва свободно ако пожелае. За разлика от Adapter pattern, където можем да комуникираме само и единствено с адаптера.

Примери:

- Façade pattern е използван в много Win32 API базирани класове за да скрие сложността на Win32.
- В C#: File.ReadAllText() метода крие сложността (отваряне на потока(stream), четене на данните от файла, конвертирането им в string …)

UML Диаграма:



Builder pattern

Разграничава създаването на сложен обект от представянето му, така че един и същ процес да може да създава обекти с различни представяния. Позволява пълен контрол върху процеса на създаване на обекта.

Описание:

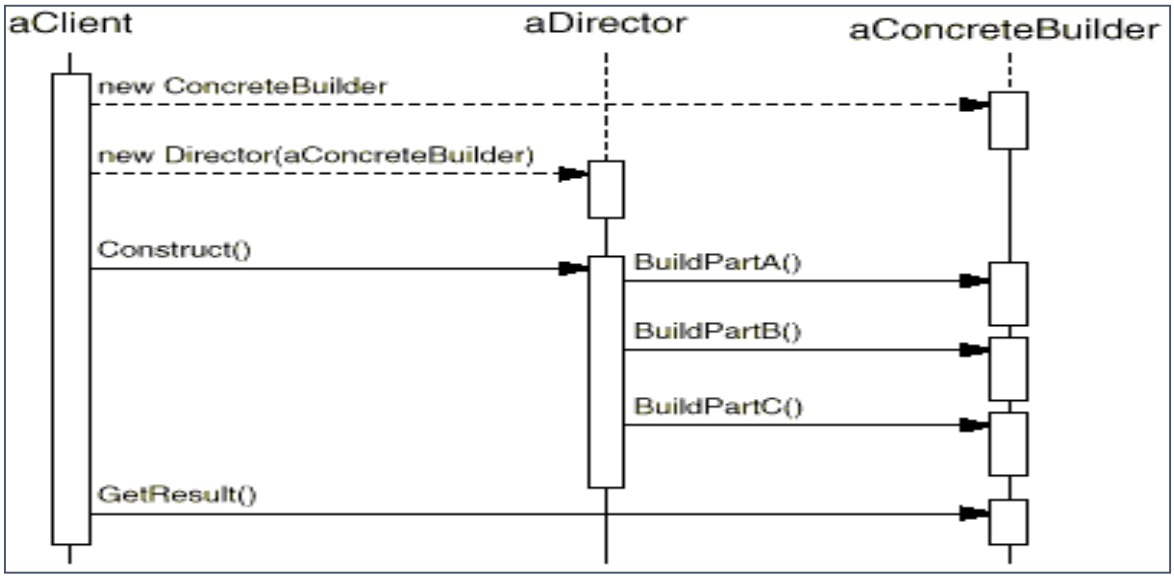
- Създаване на сложен обект
  - Твърде много параметри
  - Специфичен ред на създаване
  - Различни конструктори
- Преобразуване на обект от едно представяне към друго
- Създава обект по стъпки в специфичен ред, които зависят на много неща

Пример:

- Director -> Конструира обект посредством Builder
- Builder -> Абстрактен клас или интерфейс за създаване на части от обект
- Concrete Builder ->
  - Конструира и сглобява части от продукта чрез имплементация на Builder
  - Дефинира и следи създаваните от него инстанции
  - Представя интерфайс за извличане на крайния продукт

Взаимодействия между класовете:

1. Клиентът създава обекта Director и го конфигурира с желания обект Builder
2. Директорът извиква метод на строителя за да изгради някоя част от продукта
3. Builder обработва заявките от директора и добавя части към продукта
4. Клиентът получава продукта от строителя



UML Диаграма: