**Command Design Pattern**

-**Причина**:

* Нужда от връщане на състояние на обект
* Енкапсулиране на заявка като обект, чрез което е възможно параметризирането на клиенти с различни заявки и поддържането на неизпълними операции

-Приложение: При нужда на заявки към обекти без каквото и да е знание за исканата операция или за получаващия заявката

-**На кратко**:

Командата разделя обекта, който призовава операцията от този, който знае как да я изпълни.За да се получи това разделение, дизайнерът създава абстрактен базов клас, който посочва получател(обект) с действие(указател към член-функция).Базовият клас съдържа Execute() метод, който извиква медота на получателя.

Всички клиенти на командни обекти разглеждат всеки обект като черна кутия като просто призовават Execute() виртуалния метод на обекта, при извикване от страна на клиента на услугата.

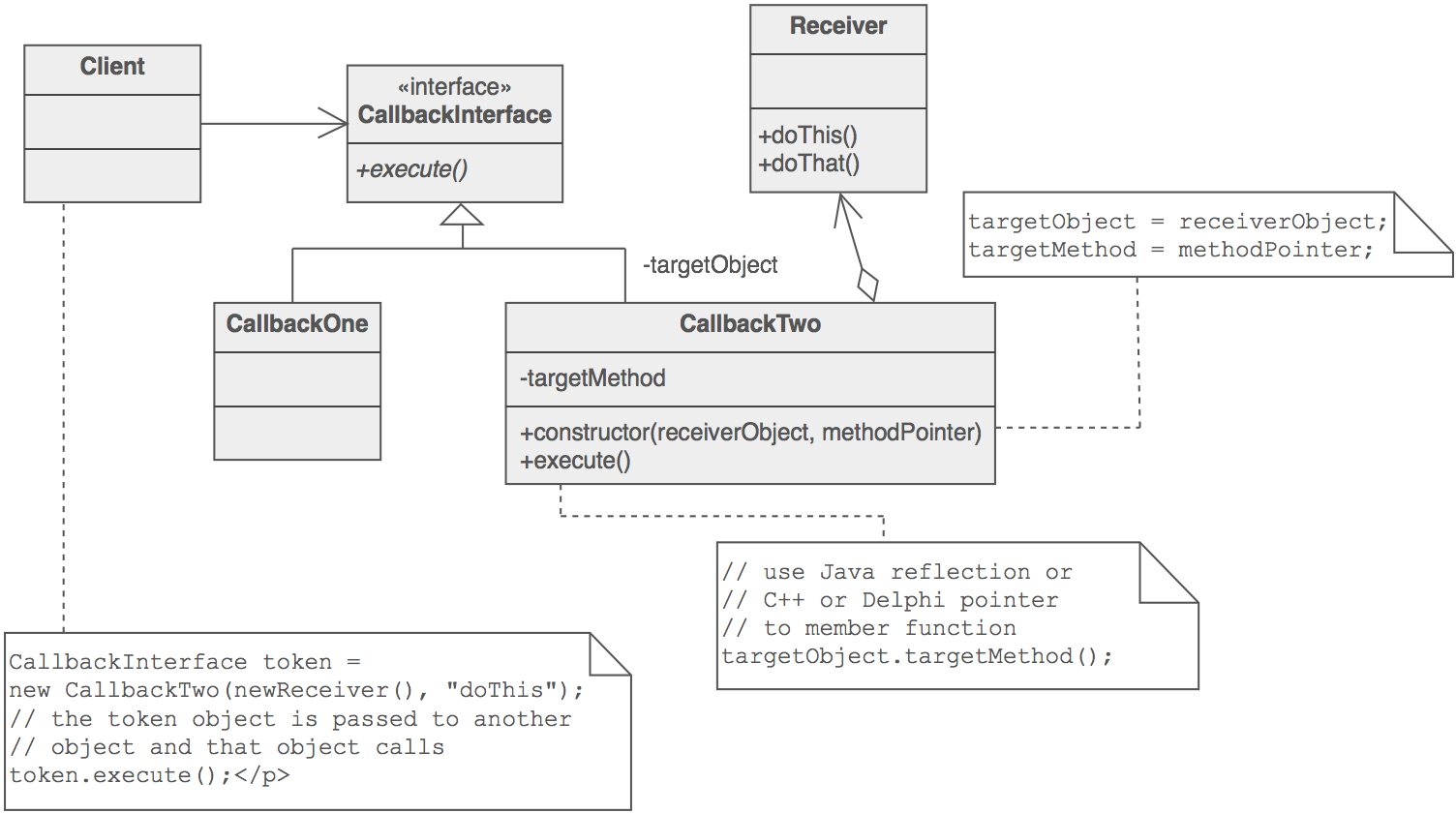
Командният клас съдържа някакво подмножество от следните: обект, метод,който да се изпълни към обекта, и аргументите нужни на метода за изпълнението му.След това командния Execute() метод напасва парчетата.

Последователности от командни обекти могат да бъдат свързани в макро команди.

-**Структура**:

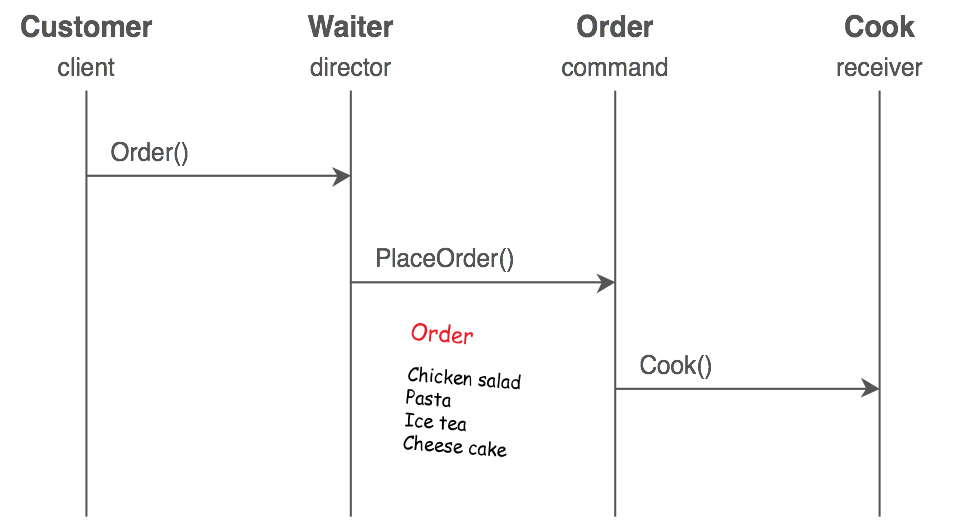
Клиента, който създава команда не е клиента, който я изпълнява.Разделянето предоставя гъвкавост в използването на командата в точното време, и изпълняването й в поредица.Материализирането на команди като обектис означава че могат да бъдат подавани, нагласяни, споделяни, зареждани в таблица и манипулирани като всеки друг обект.

Команднитте обекти могат да се считат като символ, който е създаден от един клиент, който знае какво трябва да се направи и подаден на друг клиент, който има ресурсите за изпълнението му.



-**Пример**:

Командния модел позволява заявките да се енкапсулират като обекти, следователно позволяват параметризирането на клиенти с различни заявки.Сметка на вечеря в ресторант е пример за команден модел.Сервитьора/ката взима поръчката или команда от клиент и енкапсулира поръчката като я записва на сметката.След това поръчката се дава за приготвянето й.



**Factory Method Design Pattern:**

- **Причина**:

* new оператора се смята за вреден
* Дефиниране на виртуален конструктор
* Дефиниране на интерфейс за създаване на обект, но подкласовете решават кой клас да инстанцират.

-**Проблем**:

Фреймуърка трябва да стандартизира архитектурния модел за определен брой приложения, но да позволява на индивидуални приложения да дефинират тяхни обекти и да им осигури инстанциране.

-**На кратко**:

Суперклас спрецифицира цялото поведение и след това делегира детайлите по създването на под класовете, които са предоставени от клиента.

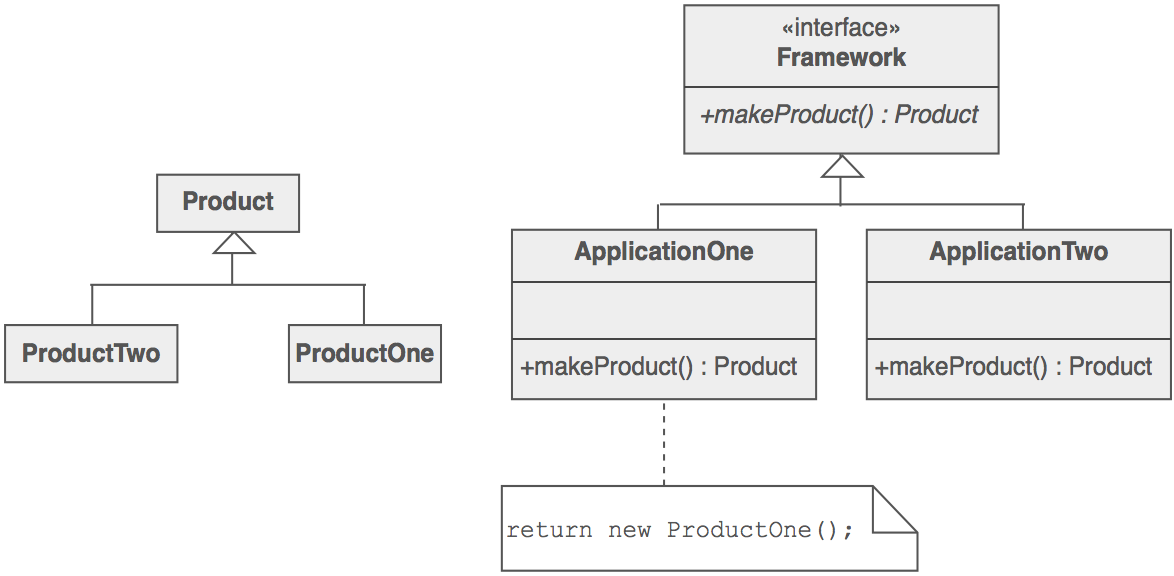
Factory модела прави дизайна персонализируем като го усложнява малко.Другите модели изискват създаване на нови класове, докато този модел изисква само нова операция.

Хората обикновено използват Factory модела като стандартен начин за създаване на обекти.

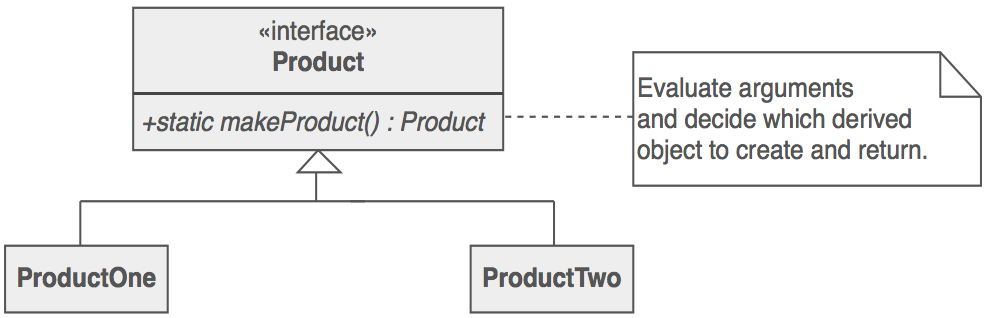
-**Наподобява** Abstract Factory

-**Структура:**

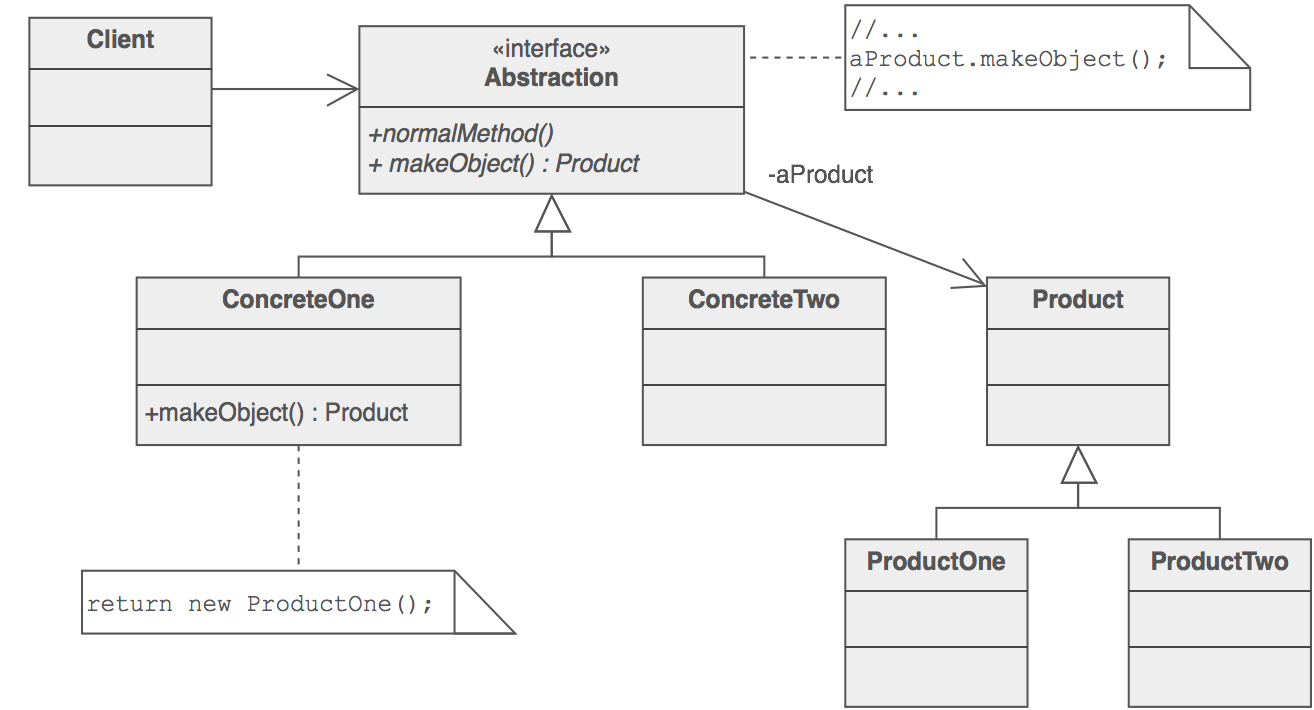
Имплементацията на Factory модела се припокрива с тази на Abstract Factory.



Все по-известна дефиниция на Factory модела е статичен метод на клас, който връща обект от типа на класа.Но за разлика от конструктор, реалния обект, който връща може да бъде инстанция на подклас.Съществуващият обект може да се преизползва, вместо да се създава нов обект.



Клиента е отделен изцяло от имплементационните детайли на производния клас.Създаването на полиморфен обект е възможно.



-**Пример:**

Factory модела дефинира интерфейс за създаване на обекти, но оставя подкласовете да изберат кои класове да инстанцират.Производителите на пластмасови играчки обработват пластмаса, след което избират формата в която да бъде излята и да бъде получена определена играчка.

