**Singleton:**

- **Причина**:

* Осигурява, че даденият клас ще има само една инстанция и достъп на всички до него
* Енкапсулирана инициализация при първа употреба

-**Проблем**:

На приложението му трябва само една единствена инстанция на класа. Допълнително късно инициализиране и глобален достъп са необходими.

-**На кратко**:

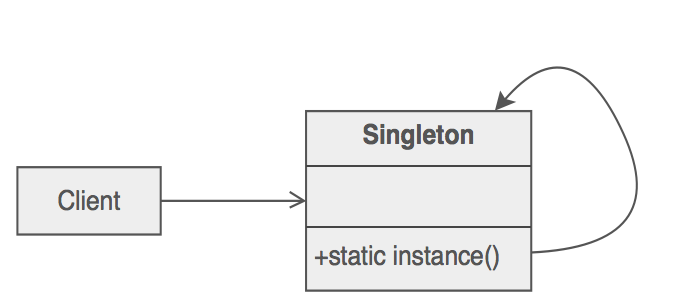
Направете класа на единичната инстанция отговорна за създаването, инициализацията, достъпа и изпълнението. Декларирайте инстанцията като private статичен дата член. Дайте достъп до public статична функция, която енкапсулира цялата част по инициализирането. Клиента извиква достъпващата функция, при нужда от достъп до единичната инстанция.

Singleton може да се смята като възможна само ако и три от следните изисквания са удовлетворени:

* Късно инициализиране
* Глобален достъп до класа
* Единичната инстанция не може да бъде собственост на друг – сама трябва да се инстанцира

Кога Singleton не е нужен? Краткият отговор: В повечето случай. По-подробният отговор: Когато е по-просто да подадеш обект като референция към обекта, на който му е нужен, вместо да се достъпва глобално. Истинският проблем със Singleton е, че ти дава добро извинение да не мислиш внимателно за енкапсулацията на обект. Намирането на правилен баланс на показ и защита за обект е критично за поддържането на гъвкавост.

-**Структура:**



Направете класа на единичната инстанция отговорен за инициализацията при първо използване. Единичната инстанция е private статичен атрибут. Функцията, чрез която се достъпва въпросната инстанция е public статичен метод.



**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.

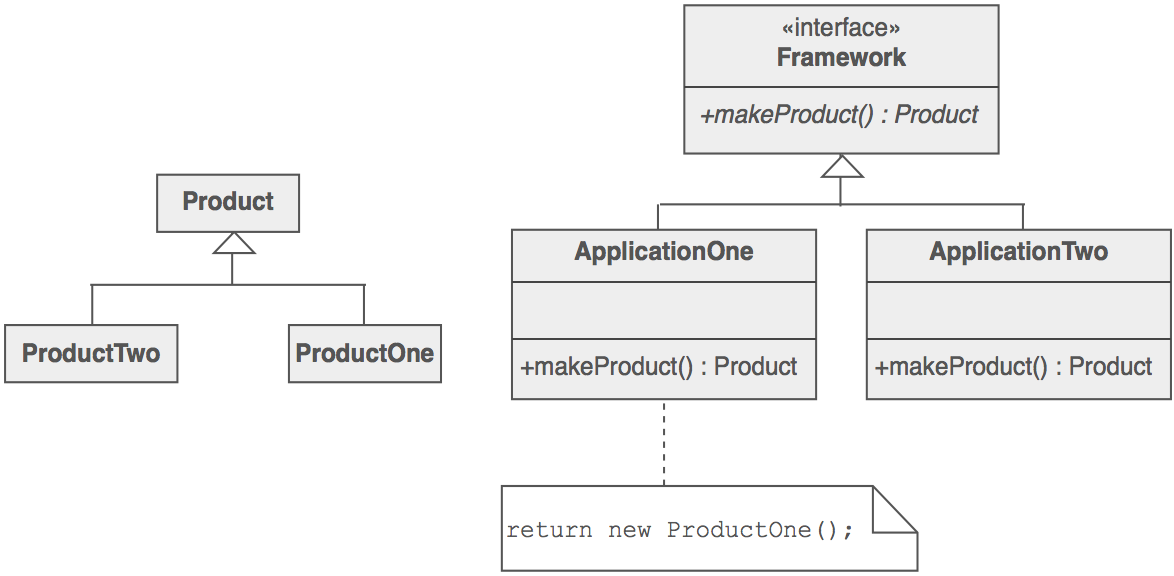
Factory модела прави дизайна персонализируем като го усложнява малко.Другите модели изискват създаване на нови класове, докато този модел изисква само нова операция.

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

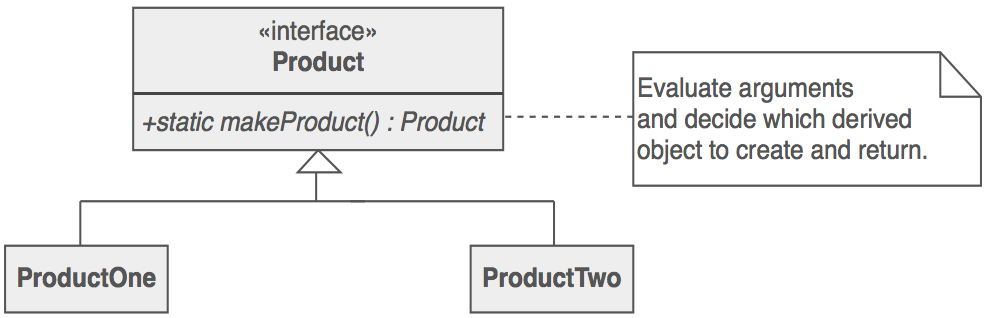
-**Наподобява** Abstract Factory

-**Структура:**

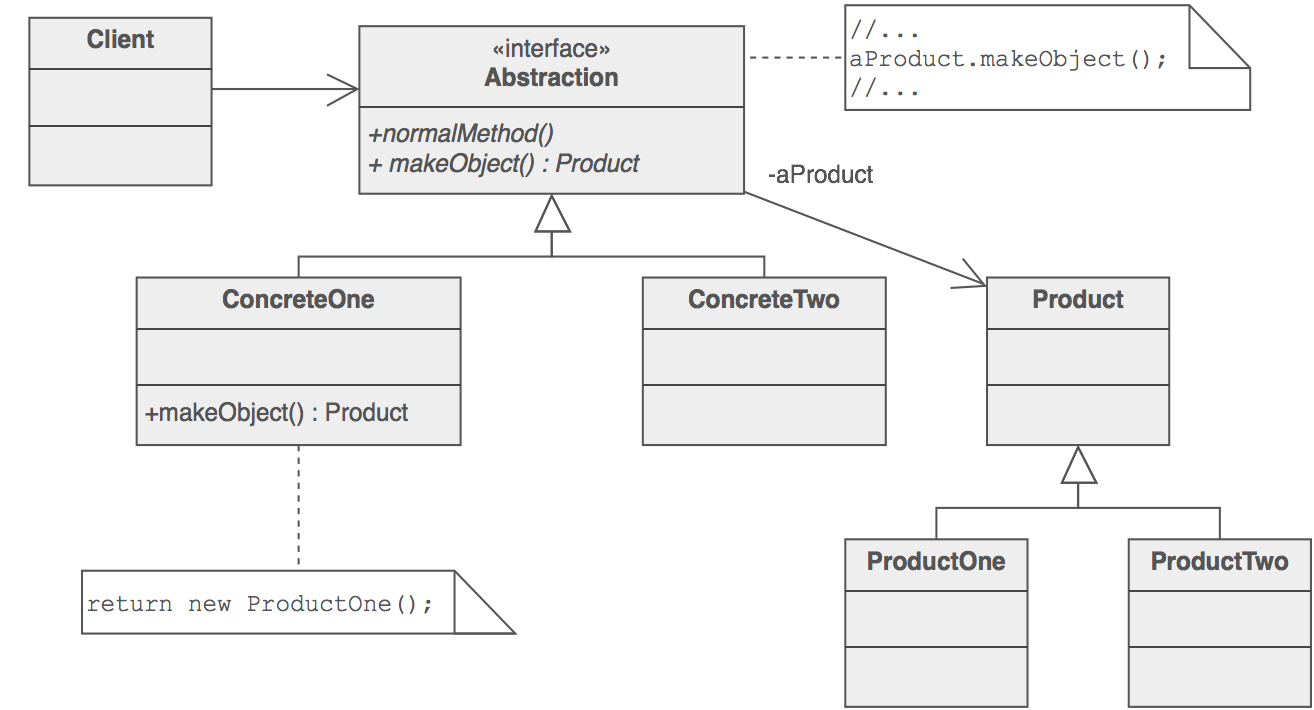
Имплементацията на Factory модела се припокрива с тази на Abstract Factory.



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.

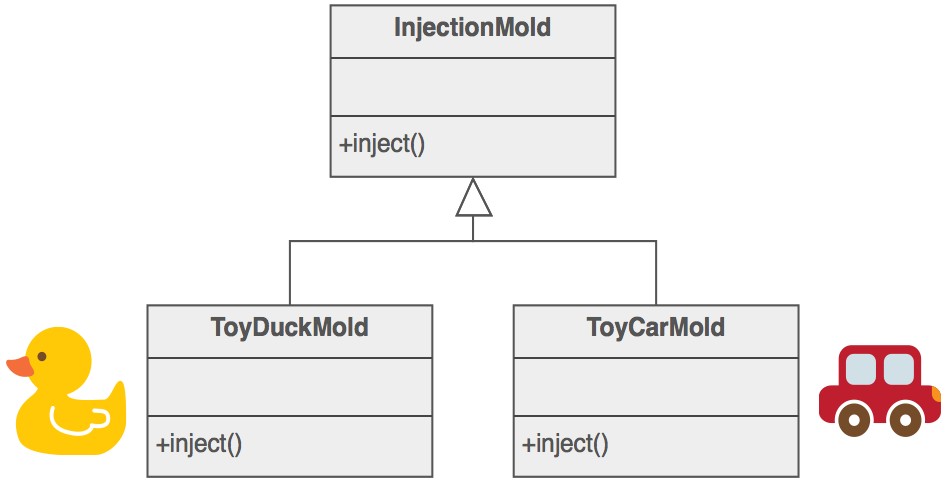


Клиента е отделен изцяло от имплементационните детайли на производния клас.Създаването на полиморфен обект е възможно.



-**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.



**Abstract Factory Design Pattern:**

- **Причина**:

* new оператора се смята за вреден
* Йерархия, която енкапсулира много възможни платформи и създаването на комплект от продукти
* Дефиниране на интерфейс за създаване на семейства от сродни или зависими обекти,без специфициране на тяхните конкретни класове

-**Проблем**:

Ако приложението се смята да е портативно, е необходимо да енкапсулира платформените зависимости. Тези платформи могат да бъдат: операционна система, база данни и др. В много случай тази енкапсулация не е създадена предварително.

-**Структура:**

Abstract Factory дефинира Factory Метод за продукт. Всеки Factory Метод енкапсулира new оператора и конкретния платформено зависим продуктов клас. Всяка платформа е моделирана със извлечен от Factory клас.

