**Шаблони за създаване**

Използват се при инициализиране и конфигуриране на обекти и класове

Creational Design Patterns

**Ползи:**

* Отговарят за начина, по който се създават обекти
* Спомагат, за да се създадат обекти, който са нужни за конкретната ситуация
* Енкапсулират детайли свързани с всеки конкретен обект от системата
* Скриват имплементационни детайли свързани с конкретното създаване и комбиниране на класов

**Шаблони, които служат за създаване на обекти**

* Singleton
* Simple Factory
* Factory Method
* Abstract Factory
* Builder
* Prototype
* Fluent Interface
* Lazy initialization
* Object Pool

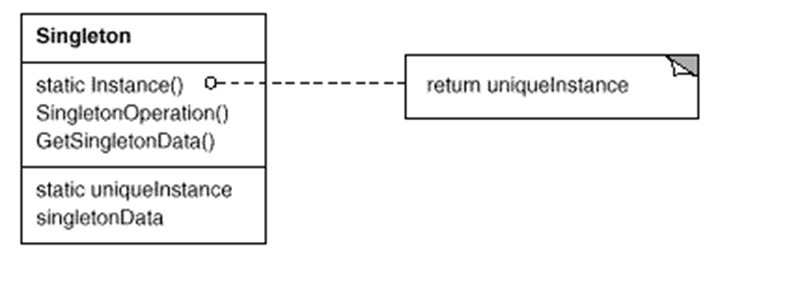
**Описание:**

**Singleton pattern**

Служи за създаването на един единствен обект от даден вид. За конкретния обект има само една единствена инстанция. Първоначално можем да се замислим каква е разликата между singleton pattern и обикновен статичен клас. Защо трябва да се имплементира singleton при положение, че може да се използва статичен клас? Истината е, че с инстанционен клас разработчика има повече възможности, защото класа е отворен за комуникация през интерфейси.

При имплементацията на този шаблон трябва да се внимава, защото първоначално няма гаранция за „thread-safe“ операции. Т.е. ако имаме имплементиран шаблон и той работи с една нишка, и ако се поиска от друга нишка в същото време, когато се прави инстанция от първата нишка, той ще се инициализира отново, за да се предпазим от това „явление“ трябва да се гарантира, че инстанцията на обекта ще е само една. Примерна имплементация може да се види в демото, където е създаден един Singleton клас и се стартират 20 извиквания към него на различни нишки.

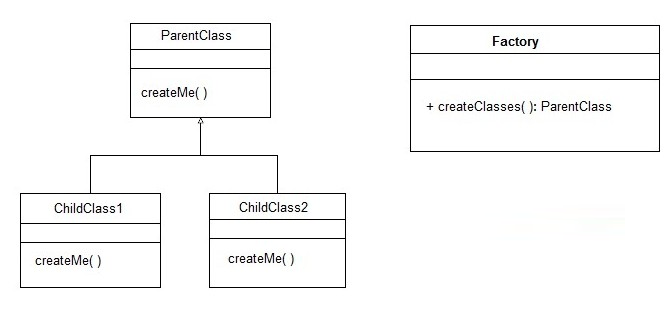
***UML диаграма Singleton pattern:***



**Simple Factory**

С негова помощ получаваме инстанция на един от няколко възможни класа в зависимост от предоставената информация. Това означава, че класовете, които се връщат като резултат имат общ родител и методи, но всеки от тях изпълнява задачите по различен начин.

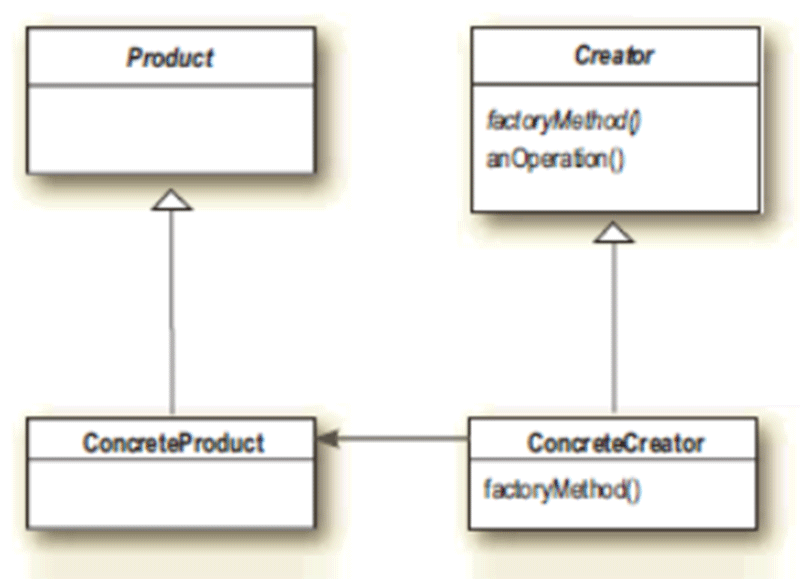
***UML диаграма Simple Factory***

****

**Factory method pattern**

Factory method pattern-a ни позволява да енкапсулираме инстанцирането на конкретни типове. Ако се погледне в диаграмата по-долу може да се види, че абстрактния клас „Creator“ ни предоставя интерфейс с метод за създаването на обект. Този метод е по-известен като “factory method”. Всички останали методи, които са имплементирани в абстрактния клас служат, за да оперират върху продукти, които са резултат от factory method. Като официална дефиниция често може да се чуе, че Factory Method-a позволява подкласовете да решат какъв клас да инстанцират. Казва се „решат“ не защото шаблона позволява подкласовете да решават, а защото „Creator“ класа е така написан, че не знае какъв продукт може да се получи. Това се случва според решението на подкласовете.

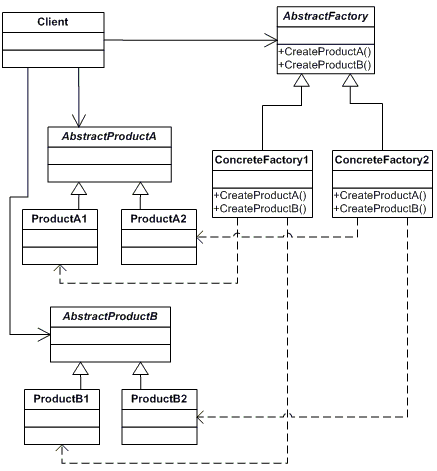
***UML диаграма Factory Method***

****

**ABSTRACT FACTORY**

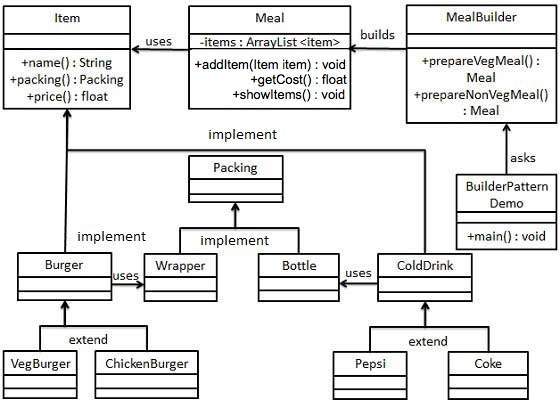
Предоставя интерфейс за създаване на семейства от свързани или зависими обекти без да определя техните конкретни класове.

***UML диаграма Abstract Factory***



**Builder pattern**

Този шаблон служи за построяването на сложни обекти като използва прости обекти и създава каквото е нужно стъпка по стъпка. Предоставя една от най-добрите възможности за създаването на обекти. Примерно ако имаме случай с ресторант за бързо хранене, където типичната храна е хамбургер и студена напитка. Хамбургера може да бъде вегетариански, пилешки и т.н. и е нужно да се опече. Студената напитка може да е кола или пепси и трябва да се сервира в бутилка. Детайли могат да се видят на следната **UML диаграма:**



Този шаблон много прилича на Abstract Factory. Каква е разликата? Abstract factory ще върне инстанция директно, а Builder ще има директор, който ще инструктира Builder класа да построи различни части/свойства от обекта и накрая ще получите обекта. Abstract factory не пази история във връзка със създаването, а Builder пази референция към създаденият обект.