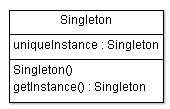
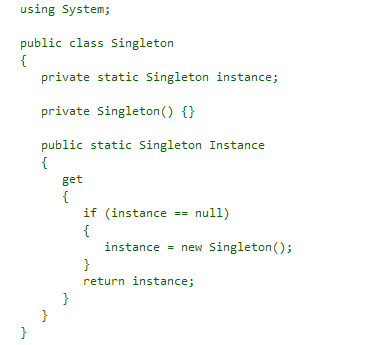
# Design Patterns

## Singelton

Целта на singleton е да гарантира, че ще се създаде само една инстанция на даден клас.

Има много неща в реалния свят, които са уникални. Не може да има две едни и същи създания. Не може да има двама президента на Америка или на България. Това са случаи или по скоро проблеми, които можем да разрешим чрез design pattern-a Singelton.

Кога да използваме Singelton? Винаги когато имаме нужда да съществува само една инстанция от определен клас, която да може да се достъпва от определено място. Достъпа до тази инстанция, трябва да може само да се използва, без да се променя и да се модифицира кода. 

Какво още можем да кажем за Singelton? Той ни помага да избегнем многократното създаване и премахване на обекти. 

## Factory Method

Factory method ни предоставя възможноста да енкапсулираме създаването на обекти, като същевременно го улесни и предостави по-добра абстракция на потребителите на фабриката. Ще разгледаме в детайли както имплентацията на шаблона, така и различните му приложения.

Factory Method е шаблон, който дефинира интерфейс за създаване на обекти, но точния тип на създадения обект се решава от наследниците на фабриката.

### Кога да го използваме?

Когато не искаме да използваме обикновен конструктор и искаме да енкапсулираме създаването на конкретен обект.

Когато искаме да предоставим по-добро и по-говорящо име на конструктора на даден клас.

Когато искаме да контролираме броя на създадените обекти:

Когато искаме да избегнем оператора new, който е опасен и може да има странични ефекти.

Понякога създаването на обект не е сигурно, че ще е успешно - може да хвърли изключение, което не е добре да се случва в конструктор.

Използва се за създаване на обекти зависещи от интернет услуги, файлова система, потоци или други обекти на операционната система.

### Предимства

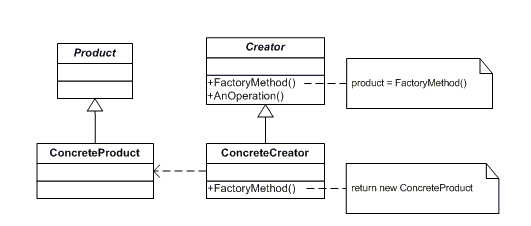
Шаблонът „Метод Фабрика“ дава предимство пред простото използване на контруктори, енкапсулирайки създаването на конкретен обект. Много често се налага, когато се създаде един обект да се конфигурира по специален начин. Това конфигуриране заедно със самото създаване може много лесно да се скрие (енкапсулира) в един метод, вместо да бъде копирано навсякъде, където създаваме обекти.

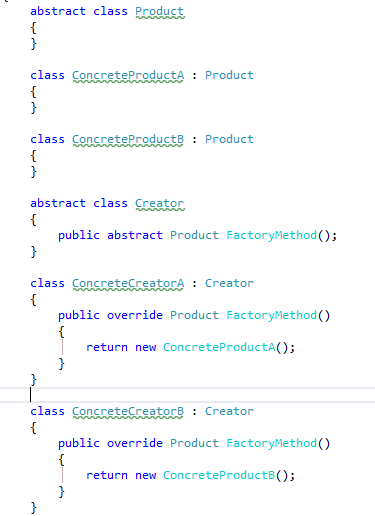
Използването на шаблона води до подобряване качеството на кода, защото се програмира срещу общ интерфейс на продуктите, а не конкретни техни имплементации.

Предоставя се точка на разширение за наследниците на даден клас при създаването на даден продукт. Наследниците на обекта създадел могат да предефинирират метода фабрика и по-този начин да променят типа на връщания обект (конкретен продукт), запазвайки базовата функционалност. (Виртуален конструктор).

Може да служи като средство за по-добра абстракция, когато работим с различни йерархии от класове.

Един от основните недостатъци е, че Фабриките работят с йерархии от класове и ако продуктите не споделят общ интерфейс (базов клас), то не може да се използва шаблона „Метод Фабрика“.





## Builder Pattern

С помощта на шаблона „Строител” (Builder) ще превърнем сложния процес на конструиране в интуитивен и лесно преизползваем алгоритъм. Ще разгледаме и някои възможни подобрения на класическия шаблон. Накрая ще се спрем върху няколко примера, в които шаблонът е приложим и ще поговорим за допирните му точки с някои други шаблони.

Цел на шаблона

Шаблонът „Строител” (Builder) се използва за конструиране на обекти със сложна структура. Това са обекти, които съдържат множество член данни, включително и такива които сами по себе си са обекти със сложна структура. Целта му е да разделя конструирането на обекта от неговото вътрешно представяне. Разделянето на отговорностите се прави и с цел същите стъпки от процеса на конструиране да се преизползват за създаване на сходни обекти с различно вътрешно представяне. Тези обекти се наричат продукти и често имплементират единен интерфейс.

Проблем

Всяко съвременно приложение обработва и представя данни, които се предоставят от различен брой източници. В зависимост от предназначението си, източници на данни се различават, както по вътрешното представяне, така и по начина на достъп до данните, които съдържат в себе си. Един често използван механизъм за достъп до данни е конструирането на заявки. Самите те също могат да варират по синтаксис и семантика и в резултат от това, възниква едно сериозно предизвикателство за съвременните софтуерни приложения, а именно - да осигури унифициран начин за достъп до хетерогенни източници от данни.

Нека допуснем, че трябва да проектираме приложение, което конструира заявка, като се абстрахира от конкретния източник от данни. Заявките към него трябва да поддържат следните основни операции:

* проекция – позволява достъп само до едно или няколко от свойствата на даден тип;
* селекция – филтрира данните от даден тип по зададено условие за едно или няколко от свойствата на типа;
* сортиране – подрежда резултатите в ред определен от стойностите на едно или няколко от свойствата на сортирания тип;

Също така приложението трябва да може да конструира три различни типа заявки – SQL заявки към релационна база данни, XQuery заявки към xml документна база данни и HTTP заявки към RESTful уеб-услуга.

За целите на примера нека работим с данни за множество от продукти, като всеки продукт има име и цена.

След като вече имаме ясна дефиниция за набора от данни, с които работим, очаквания резултат, поведение и начин на взаимодействие с нашата система, може да пристъпим към проектиране на йерархията от класове.

