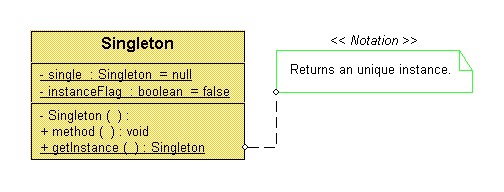
Singleton

Singleton pattern-а е от типа creational design patterns. Предназначен е да инициализира една единствена инстанция от своя клас, която да координира действията на останалите обекти в проекта. Ако не съществува инстанция от класа, тя трябва да се създаде, в противен случай – винаги да се праща референция към вече съществуващата инстанция, вместо да се инициализира друга. Известен начин това да се направи е конструкторът да има частно определение за достъп (private access modifier). Разликата между обикновен статичен клас и Singleton е, че при втория може да се осъществи така нареченото „мързеливо“ инициализиране, при което не се изразходват памет и ресурси до момента, в който това не стане наистина наложително. Сред недостатъците на този design pattern е фактът, че имплементирането на unit тестове е много по-трудно.



Пример в C#:

using System;

public class Singleton

{

private static Singleton instance;

private Singleton() {}

public static Singleton Instance

{

get

{

if (instance == null)

{

instance = new Singleton();

}

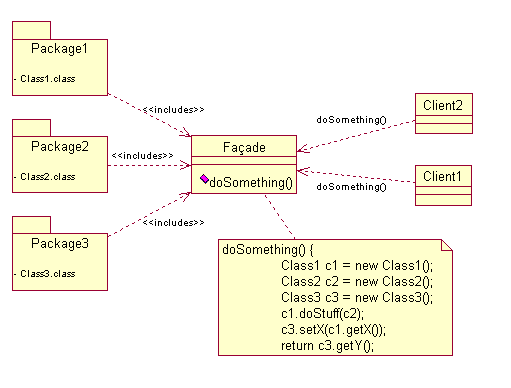
return instance;

}

}

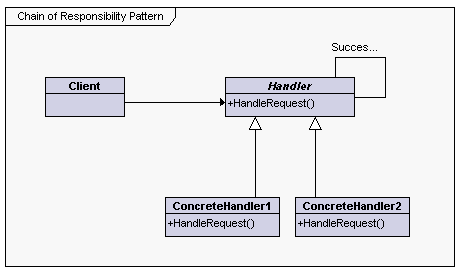
}

Façade

Това е design pattern от типа Structural design patterns, имащи за цел да имплементират нова функционалност на обектите и опростяване на връзките между различните обекти. „Фасадата“ може да направи дадена библиотека по-лесна за упралвние и за разбиране чрез имплементирането на достъпен програмен интерфейс като освен това намали зависимостта между отделните обекти и увеличи гъвкавостта на дадената система. Освен това всичката тази функционалност, която ни предоставя Façade design pattern-a, не ни „коства“ нищо – т.е. този шаблон не притежава големи или много на брой недостатъци.

Пример за този шаблон е приложен като отделен текстов документ.

Chain of responsibility

Chain-of-responsibility design pattern е от тип behavioral или иначе казано – занимава се с взаимодействието и „отговорностите“ между различните обекти. Той ни дава възможност да предаваме заявка от един обект на друг, докато тя не се изпълни – това е „верижното“ в този шаблон. Състои се от команди обекти и процесни такива. Всеки процесен обект притежава логика, определяща кои команди може да изпълни. Когато даден процесен обект не може да изпълни определена команда той я препраща към следващия процесен обект и т.н. Предимството на този модел е, че ни предоставя възможност за loose coupling между отделните класове – нещо много важно в обектно-ориентираното програмиране.

Пример в C# за този модел е прикачен като отделен текстов документ.