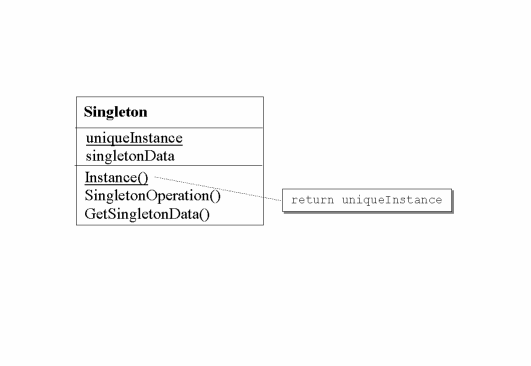
# Singleton

**Сек (Singleton) е създаващ шаблон за дизайн, който се използва в обектно-ориентираното програмиране. Този шаблон се използва обикновено в моделирането на обекти, които трябва да бъдат глобално достъпни за обектите на приложението (например обекта съдрържайки структурите с настройките на програмата ) или обекти, които се нуждаят от максимално късна инициализация за пестенето на ресурси от паметта.**



// Пример за Сек

using System;

namespace SingletonDesignPattern

{

/// <summary>

/// главния клас за стартиране на примерното приложение

/// </summary>

class MainApp

{

/// <summary>

/// Главния метод за стартиране

/// </summary>

static void Main()

{

// Конструктора (Constructor) е защитен (protected) -- операторът new не може да бъде извикан

Singleton s1 = Singleton.Instance();

Singleton s2 = Singleton.Instance();

// Test for same instance

if (s1 == s2)

{

Console.WriteLine("Objects are the same instance"); //и двата обекта са една и съща инстанция (оплътворение) на този клас

}

// Изчакай удар по конзолата от потребителя

Console.ReadKey();

}

}

/// <summary>

/// The 'Singleton' class - класът Сек

/// </summary>

class Singleton

{

private static Singleton \_instance; //променлива за единствената инстанция на този клас

// Constructor is 'protected' - конструтора е защитен и не може да бъде извикан

protected Singleton()

{

}

//единтвения начин за инстанцииране е от тук

public static Singleton Instance()

{

// Uses lazy initialization. - използва късна инициализация

// Note: this is not thread safe. - Да не се използва в многонишкови приложения

if (\_instance == null)

{

\_instance = new Singleton();

}

return \_instance;

}

}

}

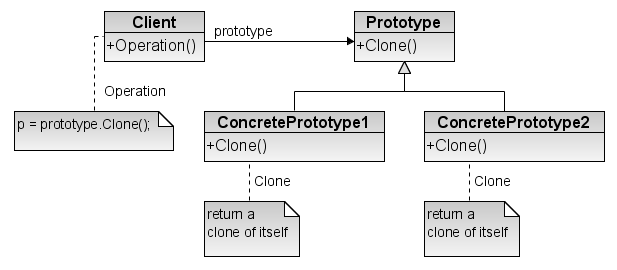
# Декоратор (Decorator) е структурен шаблон за дизайн, който се използва в обектно-ориентираното програмиране.



Този шаблон може да бъде използван за разширяването на функционалността на опеределен клас по времето на изпълнение на програмата, като запазва интерфейса му. Това разширение може да бъде постигнато чрез прибавянето на нов клас които "обвива" оригиналния клас. Декораторите предоставят гъвкава алтернатива на наследяването за разширяване на функционалността.

# Prototype

Прототип (Prototype) е създаващ шаблон за дизайн, който се използва в обектно-ориентираното програмиране. Създава обекти с помощта на обект-прототип. Новите обекти се създават чрез клониране на прототипа, вместо с използване на конструктор.



// Шаблон за прототип

using System;

using System.Collections.Generic;

namespace PrototypePattern

{

/// <summary>

/// </summary>

class MainApp

{

/// <summary>

/// Главния стартиращ метод

/// </summary>

static void Main ()

{

ColorManager colormanager = new ColorManager ();

// Инициализирай със стандартните цветове

colormanager["red"] = new Color ( 255, 0, 0 );

colormanager["green"] = new Color ( 0, 255, 0 );

colormanager["blue"] = new Color ( 0, 0, 255 );

// Потребителят избира цветове според предпочитаниятаси

colormanager["angry"] = new Color ( 255, 54, 0 );

colormanager["peace"] = new Color ( 128, 211, 128 );

colormanager["flame"] = new Color ( 211, 34, 20 );

// Потребителя клонира избраните цветове

Color color1 = colormanager["red"].Clone () as Color;

Color color2 = colormanager["peace"].Clone () as Color;

Color color3 = colormanager["flame"].Clone () as Color;

// Изчакай за удар от потребителя

Console.ReadKey ();

}

}

/// <summary>

/// Абстрактния клас от тип 'Prototype'

/// </summary>

abstract class ColorPrototype

{

public abstract ColorPrototype Clone ();

}

/// <summary>

/// The 'ConcretePrototype' class

/// </summary>

class Color : ColorPrototype

{

private int \_red;

private int \_green;

private int \_blue;

// Constructor

public Color ( int red, int green, int blue )

{

this.\_red = red;

this.\_green = green;

this.\_blue = blue;

}

// Create a shallow copy

public override ColorPrototype Clone ()

{

Console.WriteLine (

"Cloning color RGB: {0,3},{1,3},{2,3}",

\_red, \_green, \_blue );

return this.MemberwiseClone () as ColorPrototype;

}

}

/// <summary>

/// Prototype manager

/// </summary>

class ColorManager

{

private Dictionary<string, ColorPrototype> \_colors =

new Dictionary<string, ColorPrototype> ();

// Indexer

public ColorPrototype this[string key]

{

get { return \_colors[key]; }

set { \_colors.Add ( key, value ); }

}

}

}