# Singleton (Единствен)

## Цел.

Осигурява наличието на само един представител на даден клас и предоставя глобална точка за достъп до него.

## Мотивация.

За някои класове е важно да имат точно по една инстанция. Глобалната променлива прави обекта достъпен, но това не ви предпазва от инстанциране на множество обекти. По-доброто решение е да направите самия клас отговорен за следенето на единствената си инстанция. Класът може да гарантира, че не може да се създаде друга инстанция и може да предоставя начин за достъп до инстанцията.

## Приложимост.

Използва се когато:

- трябва да има точно една инстанция на даден клас и тя трябва да бъде достъпна за клиентите чрез добре известна точка.

- когато единствената инстанция трябва да бъде разширяема чрез създаване на подкласове и клиентите трябва да могат да използват разширена инстанция, без да променят своя код.

## Структура.

|  |
| --- |
| Singleton |
| static Instance()  SingletonOperation()  GetSingletonData() |
| static uniqueInstance  singletonData |

|  |
| --- |
| return uniqueInstance |

## Участници.

### Singleton

- дефинира операция Instance, даваща на клиентите възможност да достигнат до уникалната инстанция. Instance е операция в класа.

- може да бъде отговорен за създаването на своя та уникална инстанция.

## Взаимодействия.

Клиентите достигат до инстанцията на Singleton единствено през операцията Instance на Singleton.

## Следствия.

1. Контролиран достъп до единствената инстанция.
2. Намалено пространство от имена.
3. Позволява усъвършенстването на операции и представяне.
4. Позволява променлив брой инстанции.
5. Повече гъвкавост от класовите операции.

## Примерен код.

namespace Singleton

{

public sealed class Singleton

{

private Singleton() { }

private static readonly Singleton instance = new Singleton();

public static Singleton Instance

{

get

{

return instance;

}

}

}

}

## Известни употреби.

Пример за шаблона в Smalltalk-80 е набора от промени в кода, който е ChangeSet current. По-завоалиран пример е връзката между класове и техните метакласове. Метаклас е класа на нададен клас и всеки метаклас е с една инстанция. Метакласовете нямат имена, но те следят своите единствени инстанции и обикновено не създават други.

# Memento (Спомен)

## Цел.

Без да нарушава капсулирането, запазва вътрешното състояние на даден обект на външен носител, така че той да може да бъде възстановен в това състояние по-късно.

## Мотивация.

Понякога се налага вътрешното състояние на някой обект да се запомни временно. Това е необходимо при имплементиране на контролни точки и механизми за отмяна на действия, за да могат потребителите да се отказват от извършените операции или направените грешки. Информацията за състоянието трябва да се запише някъде, така че да може да възстановявате обектите в предишните им състояния. Обикновено обаче обектите капсулират част или цялото си вътрешно състояние, правейки го недостъпно за записване вън от тях. Излагането на това състояние на показ би нарушило капсулирането, което може да компрометира надеждността и разширяемостта на приложението.

## Приложимост.

Използва се когато:

- трябва да се запише снимка на (цялото или част от) състоянието на обекта, така че той да може да бъде възстановен в това състояние на по-късен етап.

- директен интерфейс за получаване на състоянието би разкрил подробности от имплементацията и би нарушил капсулирането на обекта.

|  |
| --- |
| Memento |
| GetState()  SetState() |
| state |

## Структура.

|  |
| --- |
| Memento |
| SetMemento(Memento m)  CreateMemento() |
| state |

|  |
| --- |
| Caretaker |

|  |
| --- |
| Return new Memento(state) |

|  |
| --- |
| State = m -> GetState() |

## Участници.

### Memento(SolverState)

- съхранява вътрешното състояние на обекта Originator.

- предотвратява достъп на други обекти освен източника.

### Originator(ConstraintSolver)

- създава спомен, съдържащ снимка на текущото си вътрешно състояние.

- използва спомена, за да възстанови вътрешното си състояние.

### Caretaker(механизъм за отмяна)

- отговорен е за съхранението на спомена.

- никога не манипулира или изследва съдържанието на спомена.

## Взаимодействия.

## Следствия.

## Примерен код.

namespace Memento

{

internal class Memento

{

public Memento(string name, string phone, double budget)

{

this.Name = name;

this.Phone = phone;

this.Budget = budget;

}

public string Name { get; set; }

public string Phone { get; set; }

public double Budget { get; set; }

}

}

## Известни употреби.