Flemming Lundahl

Group 1

Jacob Kurtzhals [au]

Tobias

Mikkel

Flemming Lundahl [au]

Software Design – Mandatory Assignment 3

Indholdsfortegnelse

[1 Chosen design pattern 2](#_Toc500154432)

[1.1 Purpose 2](#_Toc500154433)

[1.2 Type 2](#_Toc500154434)

[1.3 Structure 2](#_Toc500154435)

[1.4 Dynamics 2](#_Toc500154436)

[1.5 Consequenses 2](#_Toc500154437)

[2 Related patterns 2](#_Toc500154438)

[3 Implementation 2](#_Toc500154439)

[4 Usefulness 2](#_Toc500154440)

[4.1 Pros 2](#_Toc500154441)

[4.2 Cons 2](#_Toc500154442)

# 1 Memento Pattern

We have chosen to explore the Memento pattern. Through out this rapport, will we explain what the pattern is, how it is used, how to implement it and show an example of how to implement it. The example is written in C#.

## 1.1 Purpose

The purpose with the memento pattern, is to give an object the ability to a former state. In our implementation, are we making a wish list which consists of multiple wishes you can add dynamicly. The use of the memento pattern comes to use, if you add a wish you can undo the wish list to a previous state, if you are not satisfied with the wish you just added.

Other real world application of this pattern could be, used in database transactions, so you could undo a transaction. But it was hard to find a specific use for the pattern.

## 1.2 Type

Memento is a behavioral pattern. It doesn’t make new objects only changes them, in this instance is it the objects state. The pattern allows the user to go back a state and undo an action that have been made on a object.

## 1.3 Structure

The base structure of the memento pattern consists of three different classes, the memento class, the originator class and the caretaker class. The pattern is shown in figure 1 and is marked with a red box.

### 1.3.1 Memento

The memento object store the internal state of the originator object. It has two interfaces, an interface to the caretaker and an interface to the originator. The interface to the caretaker must not allow any access and any operation on the internal state. The interface to the originator and allows it to access any state variables if necessary, so it can restore it. In our implementation is it realized as WishlistMemento.

### 1.3.2 Originator

The originator class shall make the Memento object, which contains the originators internal state and uses the memento to restore it is the previous state. In our implementation is it realized as Wishlist.

### 1.3.3 Caretaker

The caretaker is responsible for keeping the memento. The caretaker must not operate on the memento. In our implementation is it realized as IWishlistCaretaker.

## 1.4 Consequenses

The memento pattern protects encapsulation and won’t expose the originators implementation and internal state. The pattern won’t affect your classes, but if the class is too big, the program can use too many resources, especially if you want to keep multiple states.

# 2 Related pattern

The memento pattern is similar to the command pattern. The command pattern doesn’t necessarily have anything to do with the undo nature of the memento pattern. But the use of the command pattern often include the make of a undo command, but they work a bit differently. The undo in the memento pattern works by restoring a saved state of an object. Meanwhile the undo with command pattern is by executing compensating actions.

# 3 Implementation

In our implementation of the memento pattern is realized as a wish list. The user will have the option to make his/her own wish list, and add wishes to it. The pattern is used to undo a recently added wish to the wish list and undo all the wishes that have been added during the session. The list will update dynamically. In the next part there will be a show of our class diagram and a sequence diagram that shows how the memento handles the AddWish and UndoWish functions.

## 3.1 Class diagram

On figure 1 below is our class diagram. The memento pattern is marked with red. If we were to remove the pattern the program will still run, but it would the ability to undo recent actions. As it is seen the caretaker stores our wish list memento and our wish list menu makes sure to set or get the state from the caretaker, if the originator (Wishlist) calls restoreToCheckpoint it will undo the last wish added.

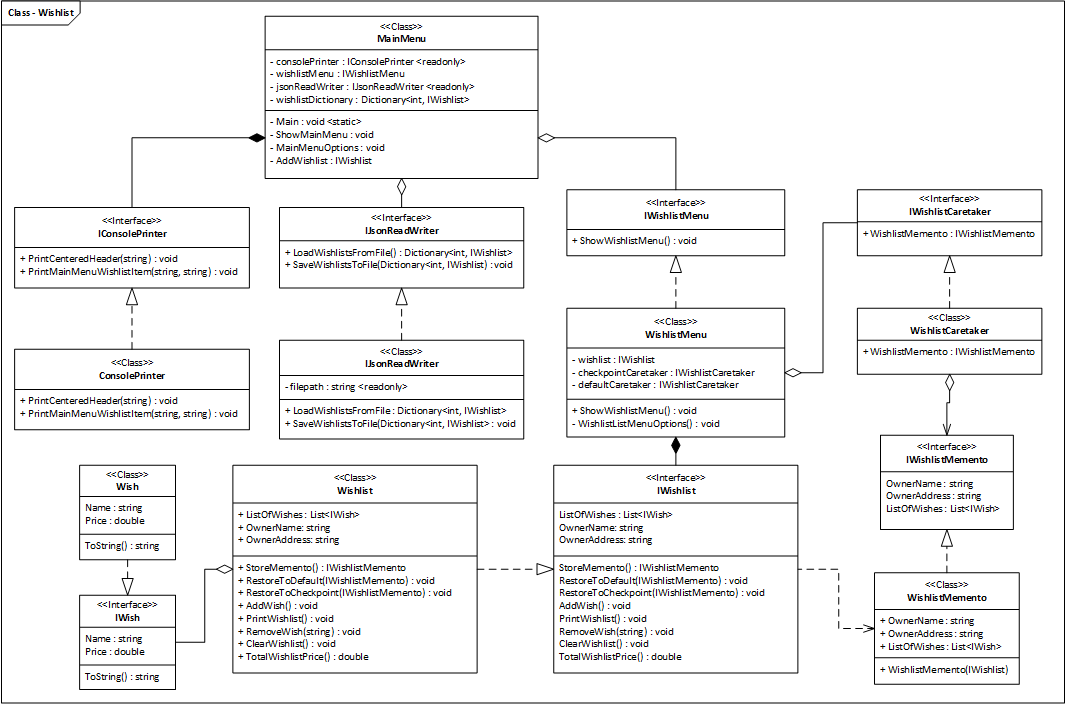


Figure , Shows the class diagram of our implementation of the memento pattern.

## 3.2 Sequence diagrams

On figure 2 it shows one run through of our program. The user will be presented a menu where the user can edit a existing wish list or add a new one. In this diagram the user adds a new list and it shows who and when the memento and caretakers are instantiated. After the call ShowWishlistMenu the user is presented with options to Add a new wish, Undo last added wish, save and exit, and exit without saving. In the exit without saving it uses the defaultCaretaker and will undo all the wishes added during your session. The next bits are a bit chopped up, but the full sequence diagram are attached in the folder.

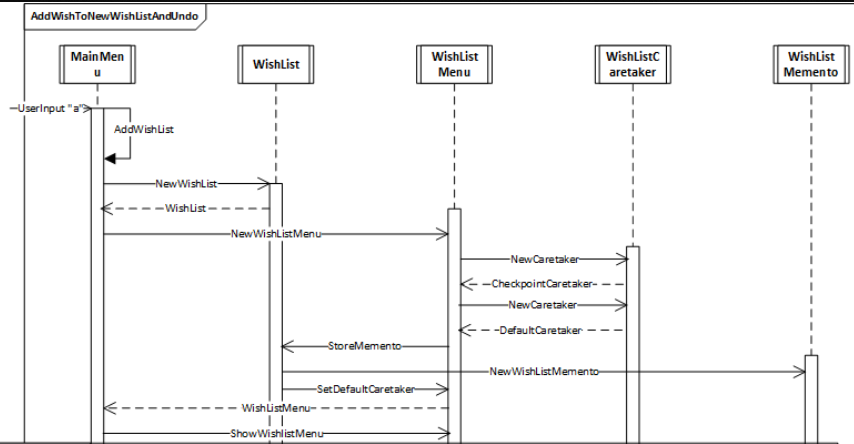


Figure , shows the sequence diagram for one run through of the program

### 3.2.1 Add command

This is after user are presented with the wish list menu. It shows how the add works and who calls the storeMemento and how it Sets the caretaker.

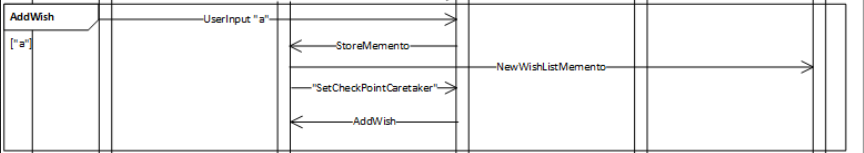


Figure , shows the sequence diagram for AddWish

### 3.2.2 Undo command

This is after user are presented with the wish list menu. It shows how the undo works who calls the restore and how it gets the caretaker.

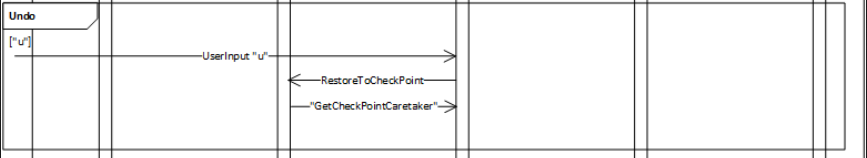


Figure , shows the sequence diagram for Undo

## 3.3 Originator klasse

In our implementation is the originator class, wishlist. It has responsibility for all its functions aswell as the memento functions. In the code snippet below it shows how the method RestoreToCheckpoint works. It changes the state of the wish list and restores last saved and it saves to checkpoint.

public void RestoreToCheckpoint(IWishlistMemento checkpointWishlistMemento)

{

if (checkpointWishlistMemento != null)

{

OwnerName = checkpointWishlistMemento.OwnerName;

OwnerAddress = checkpointWishlistMemento.OwnerAddress;

ListOfWishes = checkpointWishlistMemento.ListOfWishes;

}

}

## 3.4 Memento class

The memento class is similar to the originator, but it only contains its properties. Because it has to save all the information of the originator.

Memento klassen ligner meget Originator klassen, dog uden alle funktionerne. Dette skyldes at Memento objektet gemmer det state som Originator klassen er i og derfor bliver nød til at have de samme attributter som Originator klassen.

public WishlistMemento(IWishlist wishlist)

{

OwnerAddress = wishlist.OwnerAddress;

OwnerName = wishlist.OwnerName;

if(wishlist.ListOfWishes != null)

ListOfWishes = new List<IWish>(wishlist.ListOfWishes);

}

## 3.5 Caretaker klasse

Caretaker klassen skal blot holde på de gemte Memento klasser, har derfor blot en liste af disse i sig.

public class WishlistCaretaker : IWishlistCaretaker

{

public IWishlistMemento WishlistMemento { get; set; }

}

# 4 Usefulness

Transaktions bruger Memento pattern.

## 4.1 Pros

## 4.2 Cons