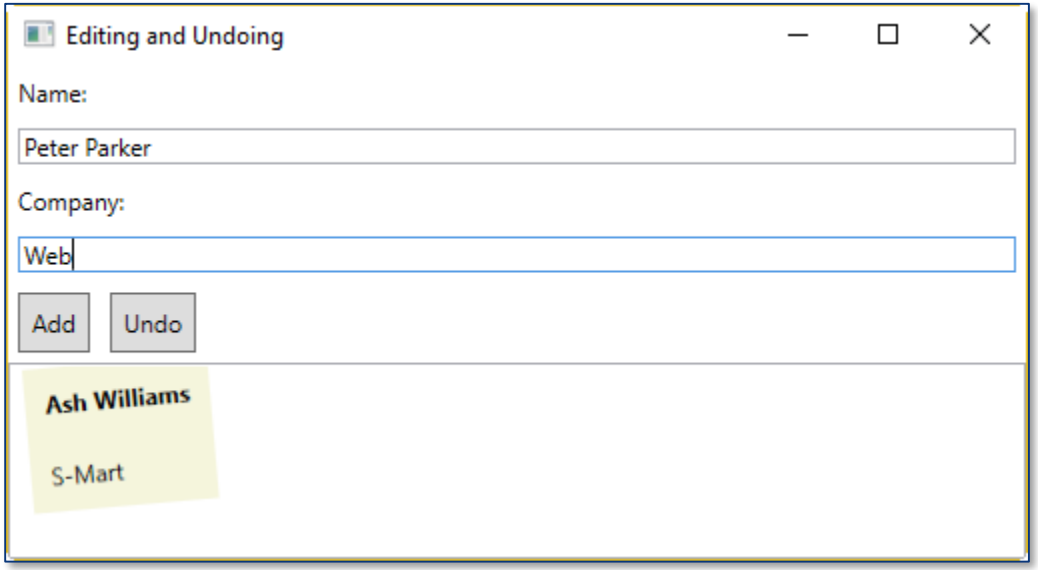
Module 18: "Memento"

Agenda

- Introductory Example: Editing and Undoing
- Challenges
- Implementing the Memento Pattern
- Pattern: Memento
- Overview of Memento Pattern
- Reusability vs. Encapsulation

Introductory Example: Editing and Undoing




```
readonly struct Guest
{
    public string Name { get; }
    public string Company { get; }
}
```

Challenges

- How do we implement undo?
- How do we externally save the internal state?
- Is it possible to do so without breaking the encapsulation of the object, i.e. exposing internal state?
- And how do we ensure only the object can access the state when externalized…?

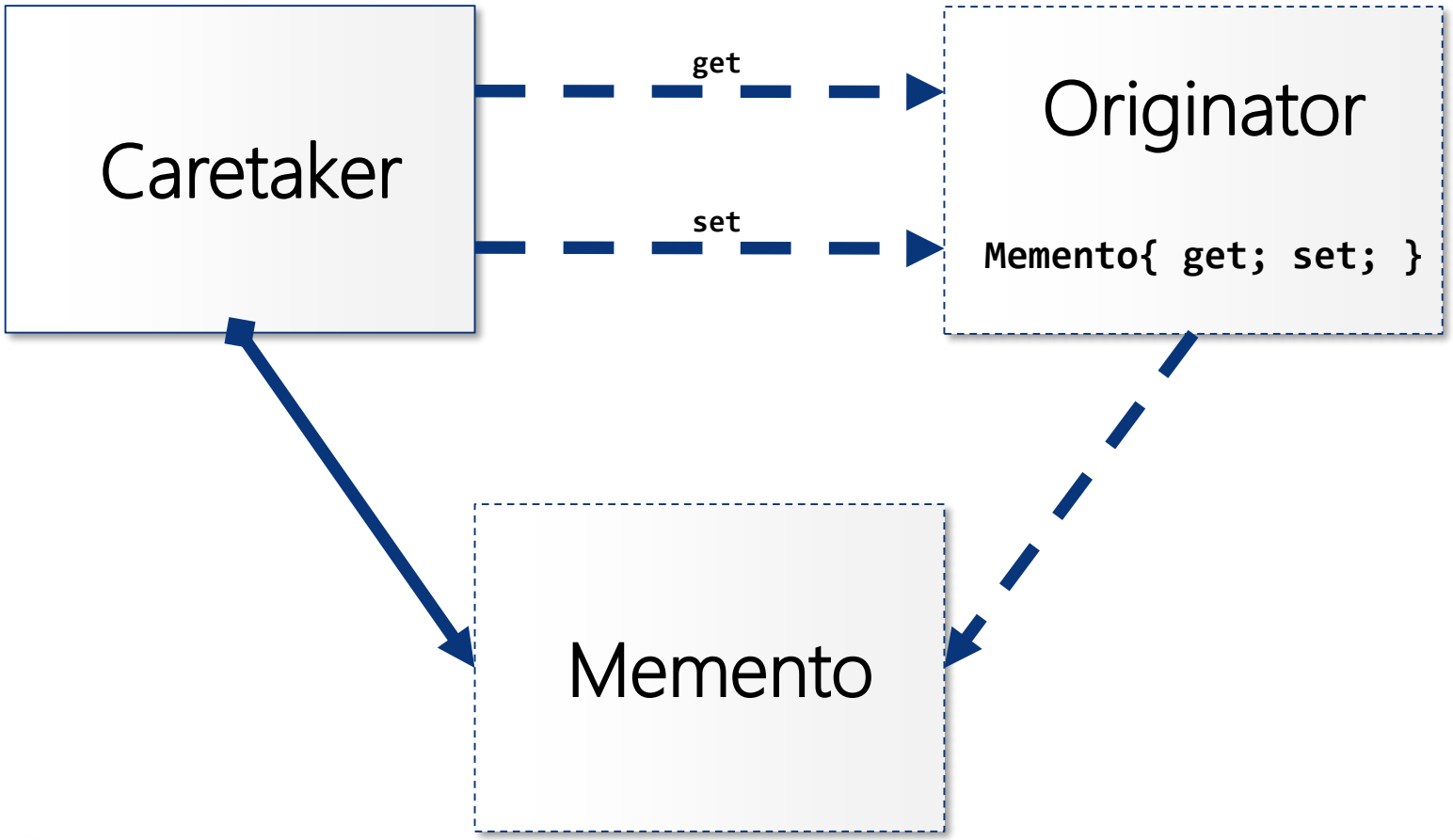
Pattern: Memento

- *Without violating encapsulation, capture and externalize an object's internal state so that it can be restored to this state later.*

- Outline
  - Make object itself responsible for saving its internal state to a memento object.
  - Make object itself responsible for restoring its internal state from a memento object

- Origin: Gang of Four

Overview of Memento Pattern



Overview of Memento Pattern

- Caretaker
  - Concrete class controlling the create/restore of Originator state
  - Retrieves and sets Memento on Originator
- Originator
  - Concrete class containing state to be externalized
  - Implements a property exposing Memento object
- Memento
  - Interface (or occasionally concrete class) containing externalized version of Originator state

Reusability vs. Encapsulation

- It is not hard to create a reusable setup for Memento
  - `IMemento<T>`
  - `Originator<T>`
  - …
- But this allows everyone to "see" externalized state

- Consider creating an "anonymous" interface implemented by class internal to Originator

```
public interface IMemento
{
    object State { get; }
}
```


WINCUBATE
Jesper Gulmann Henriksen
PhD, MCT, MCSD, MCPD
Phone : +45 22 12 36 31
Email : jgh@wincubate.net
WWW : http://www.wincubate.net
Hasselvangen 243
8355 Solbjerg
Denmark