

# Memento

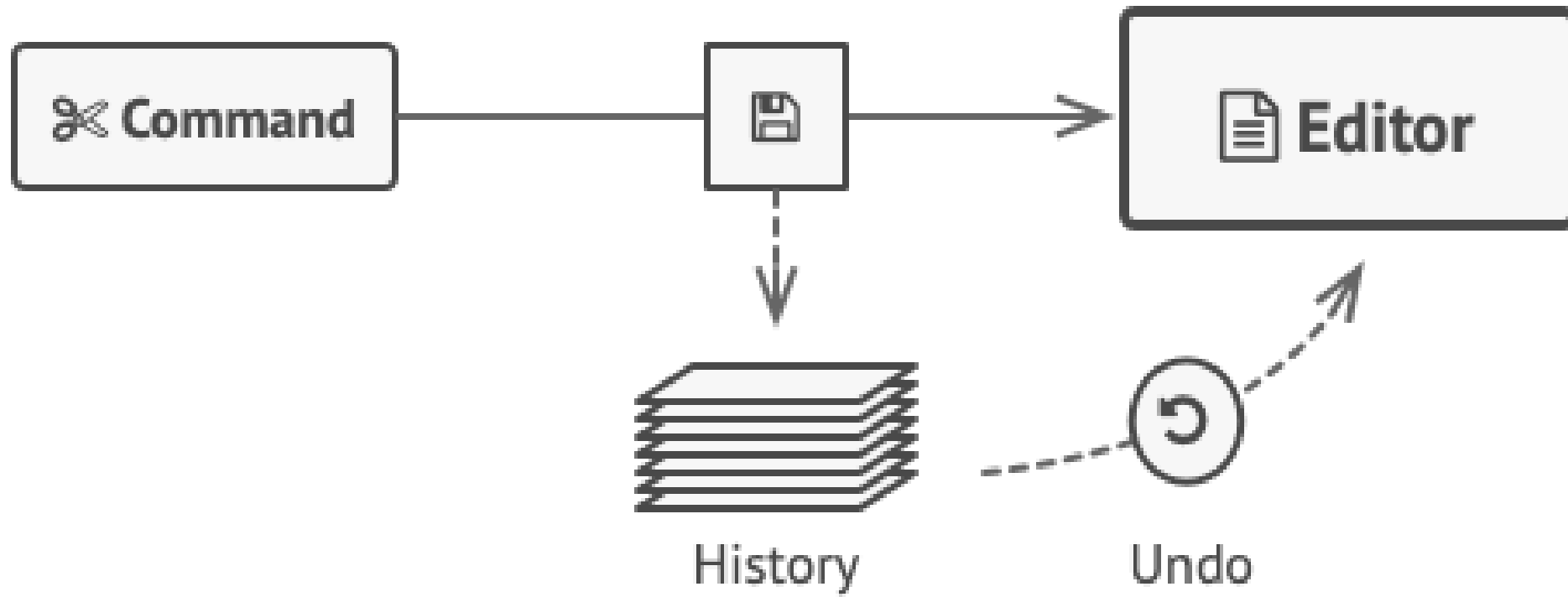
Daniel Lopata

---



# Motivace

*„Bez porušení zapouzdření zachytí a externalizuje vnitřní stav objektu tak, aby bylo možné objekt později obnovit do tohoto stavu.“*



# Motivace

```
public class Editor
{
    private StringBuilder text;
    public Font font;
    public int size;
    public Style style;

    public void Write(string s) { }
    public void SetFont(Font font) { }
    public void SetSize(int size) { }
    public void SetStyle(Style style) { }
}
```

```
static void Main(string[] args)
{
    Editor editor = new Editor();
    editor.Write("Hello");
    editor.SetSize(18);
    editor.SetStyle(Style.Bold);

    editor.Write(" world");
    editor.SetFont(Font.ComicSans);
}
```

Jak uložit stav?

# Zpřístupnění stavu veřejně

```
public class Editor
```

```
{
```

```
    private StringBuilder text;
```

```
    public Font font;
```

```
    public int size;
```

```
    public Style style;
```

```
    //...
```

```
}
```

```
public class State
```

```
{
```

```
    public StringBuilder text;
```

```
    public Font font;
```

```
    public int size;
```

```
    public Style style;
```

```
    public State(Editor editor) { }
```

```
    public void Undo(Editor editor) { }
```

```
}
```

Private členy změníme na public

```
static void Main(string[] args)
```

```
{
```

```
    Editor editor = new Editor();
```

```
    editor.Write("Hello");
```

```
    editor.SetSize(18);
```

```
    editor.SetStyle(Style.Bold);
```

```
    State snapshot = new State(editor);
```

```
    editor.Write(" world");
```

```
    editor.SetFont(Font.ComicSans);
```

```
    snapshot.Undo(editor);
```

```
}
```

Porušili jsme encapsulaci

# Úprava třídy Editor

```
public class Editor
{
    private StringBuilder text;
    public Font font;
    public int size;
    public Style style;

    private List<StringBuilder> texts;
    private List<Font> fonts;
    private List<int> sizes;
    private List<Style> styles;

    public void Save() { }
    public void Undo() { }
}
```

Editor dělá vše

```
static void Main(string[] args)
{
    Editor editor = new Editor();
    editor.Write("Hello");
    editor.SetSize(18);
    editor.SetStyle(Style.Bold);

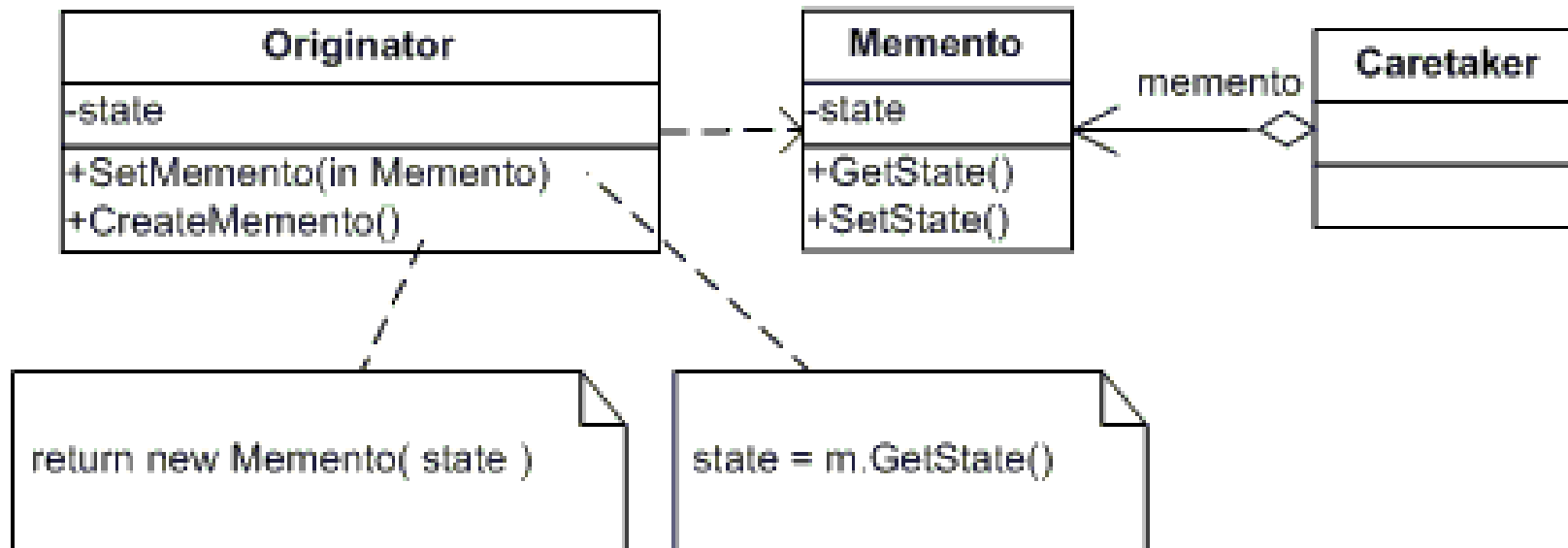
    editor.Save();

    editor.Write(" world");
    editor.SetFont(Font.ComicSans);

    editor.Undo();
}
```

Porušili jsme Single  
responsibility

# Struktura



# Účastníci

## Originator

- Objekt, jehož stav chceme uchovat
- Vytvoří memento s informacemi potřebnými pro obnovení aktuálního stavu

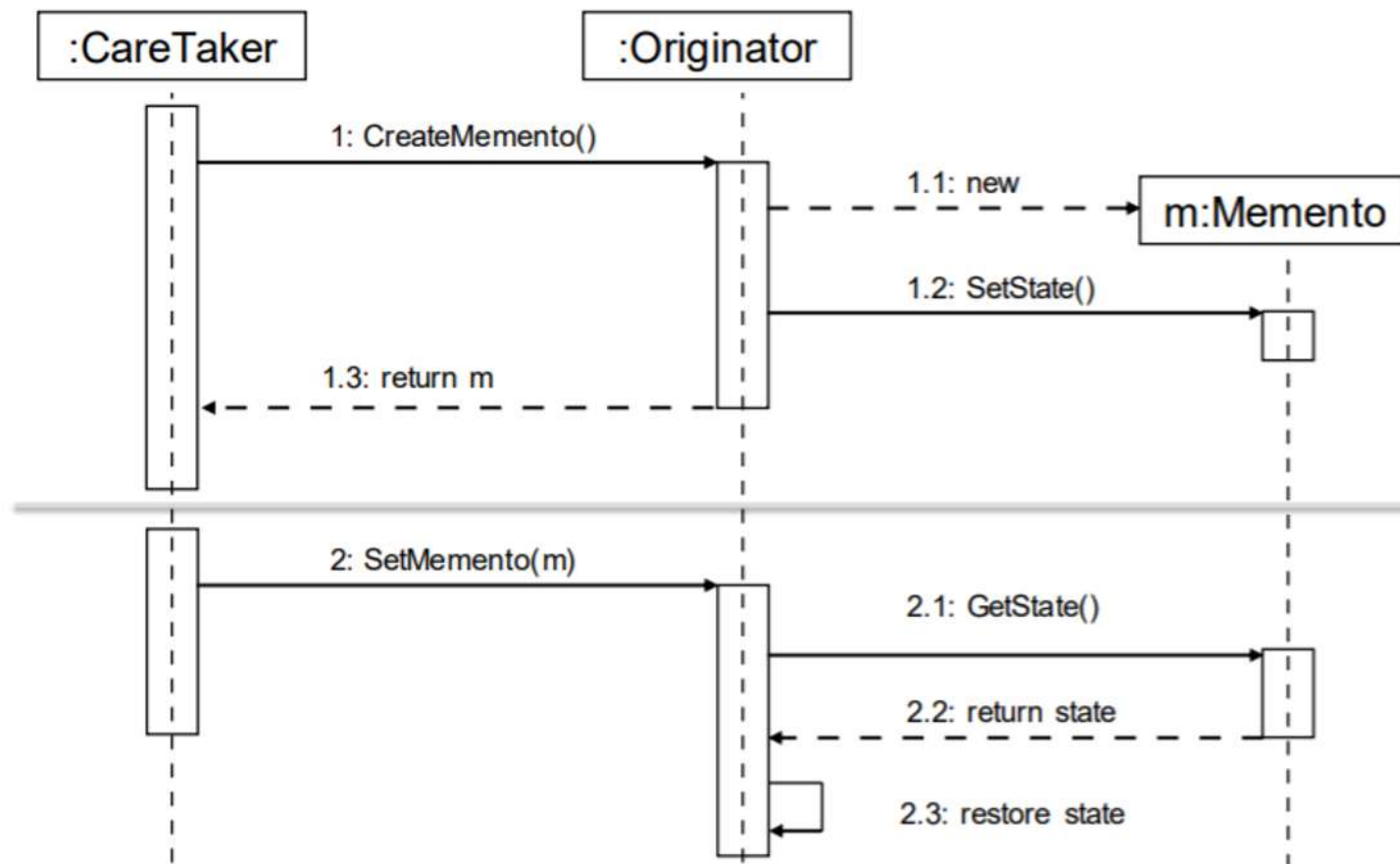
## Memento

- Objekt uchovávající informace o stavu Originator
- Chrání přístup k uloženým informacím
- Narrow/Wide interface

## Caretaker

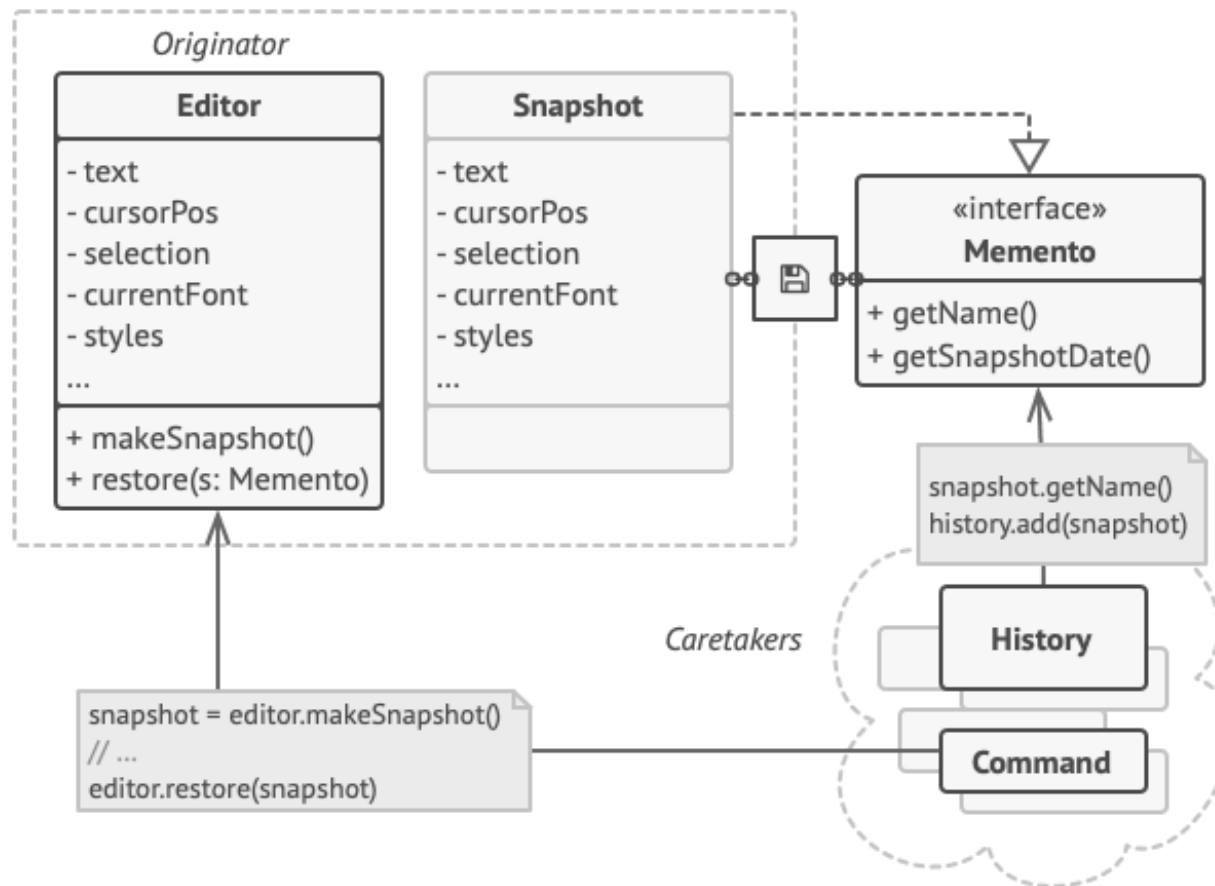
- Žádá o Memento/obnovení Originatoru z Mementa
- Zodpovědný za ukládání Mement
- Nijak nemění stav uložený v Mementu

# Interakce

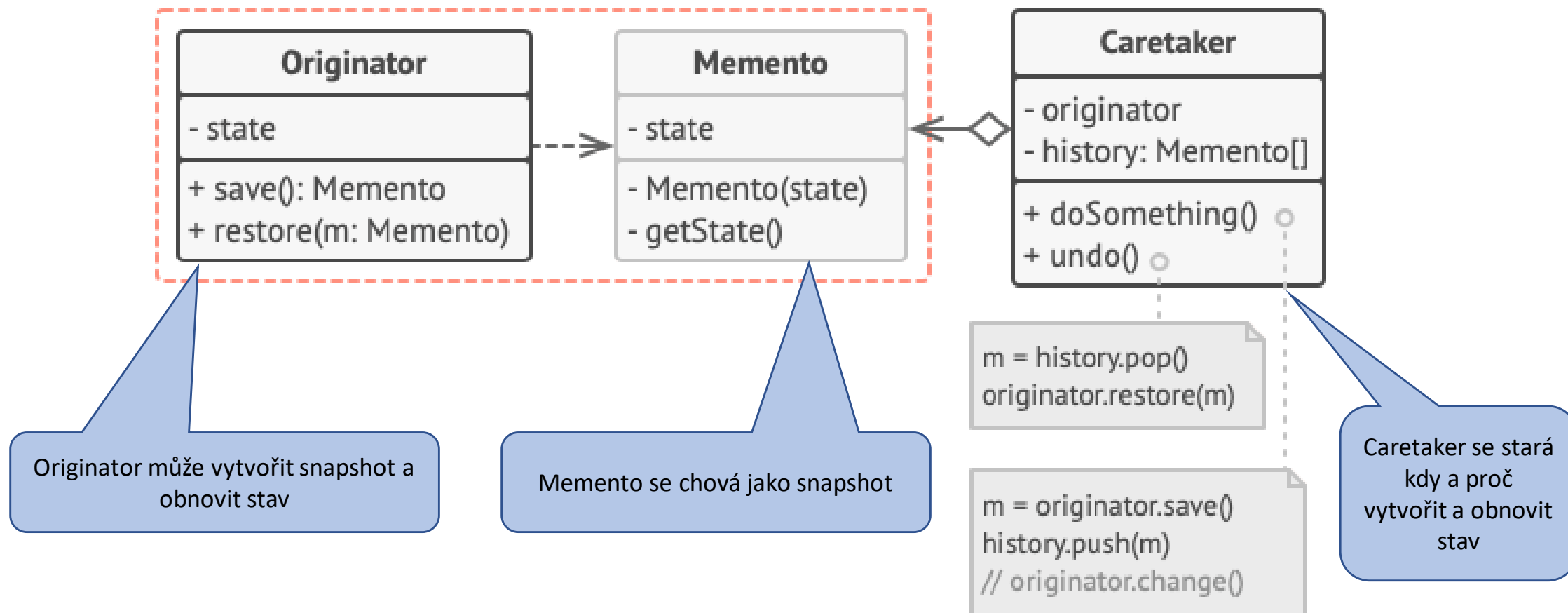




# Řešení motivačního příkladu



# Implementace použitím vnořených typů



# Implementace použitím vnořených typů

```
public class Memento {  
    private readonly StringBuilder text;  
    public readonly Font font;  
    public readonly int size;  
    public readonly Style style;  
    private Memento(StringBuilder text, Font font, int size, Style style) {  
        this.text = text;  
    }  
    public class Editor {  
        //...  
        public Memento CreateMemento() => new Memento(text, font, size,  
style);  
        public void Restore(Memento memento) {  
            text = memento.text;  
            //...  
        }  
    }  
}
```

Všechny členy Mementa chceme immutable

Editor je jediný, kdo může Memento vytvořit

Editor vidí na všechny členy Mementa

# Implementace použitím vnořených typů

```
public class Caretaker
{
    private List<Memento> mementos = new
List<Memento>();
    private Memento.Editor editor = new
Memento.Editor();
    public Caretaker(Memento.Editor editor)
    {
        this.editor = editor;
    }
    public void Backup() {
        mementos.Add(editor.CreateMemento());
    }
    public void Undo() { }
}
```

```
internal class Program
{
    static void Main(string[] args)
    {
        Memento.Editor editor = new Memento.Editor();
        Caretaker caretaker = new Caretaker(editor);

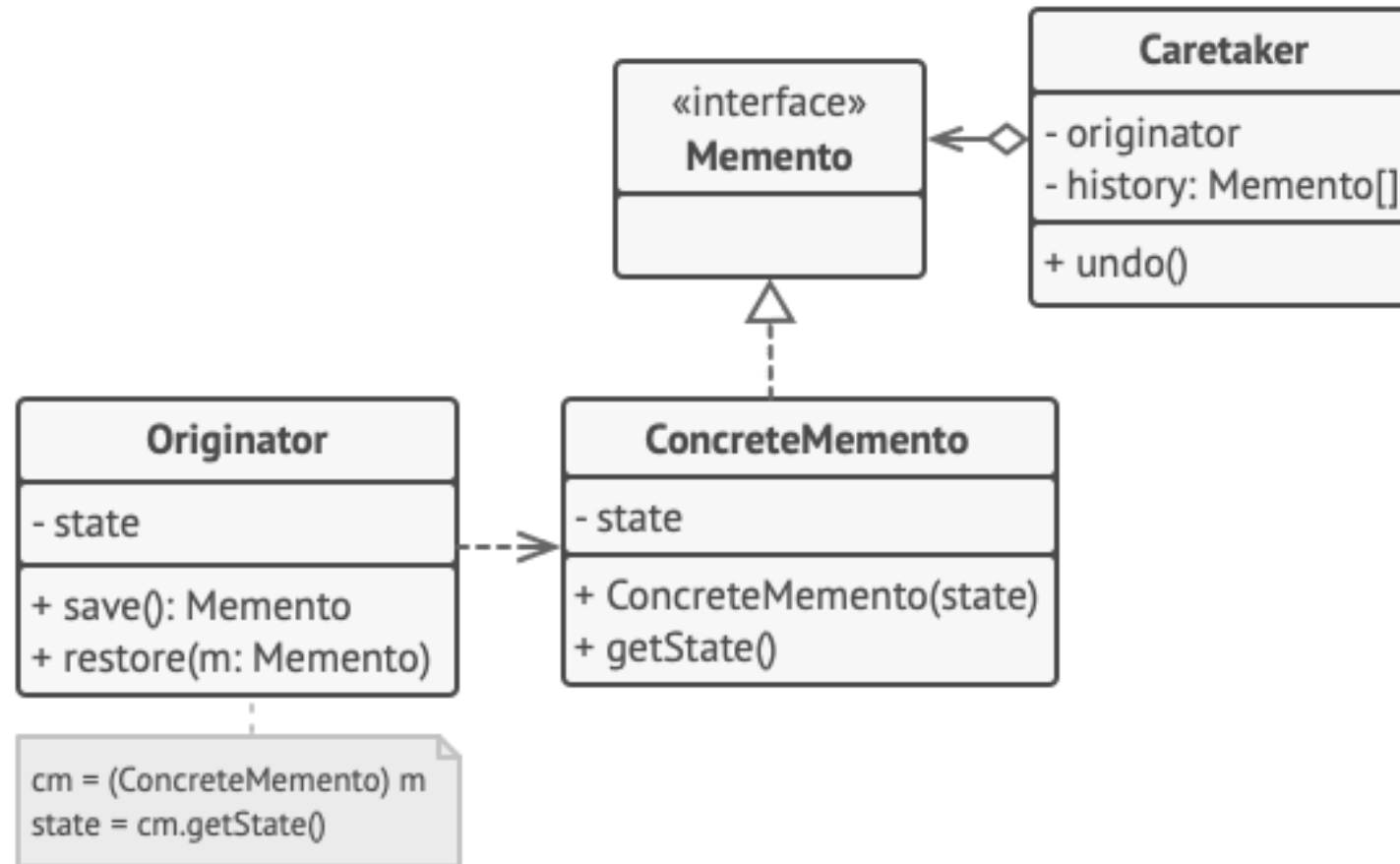
        editor.Write("Hello");
        editor.SetSize(18);
        editor.SetStyle(Style.Bold);
        caretaker.Backup();
        editor.SetFont(Font.ComicSans);

        caretaker.Backup();
        editor.Write(" world");
        caretaker.Undo();
        caretaker.Undo();
    }
}
```

Uložení stavu

Obnovení  
předchozích  
stavů

# Implementace přes rozhraní



# Implementace přes rozhraní

```
public interface IMemento
{
    DateTime GetDate();
}

public class ConcreteMemento : IMemento
{
    private StringBuilder text;
    public ConcreteMemento(StringBuilder
text)
    {
        this.text = text;
    }
    public StringBuilder GetState() => text;
}
```

Zpřístupní pouze  
metadata

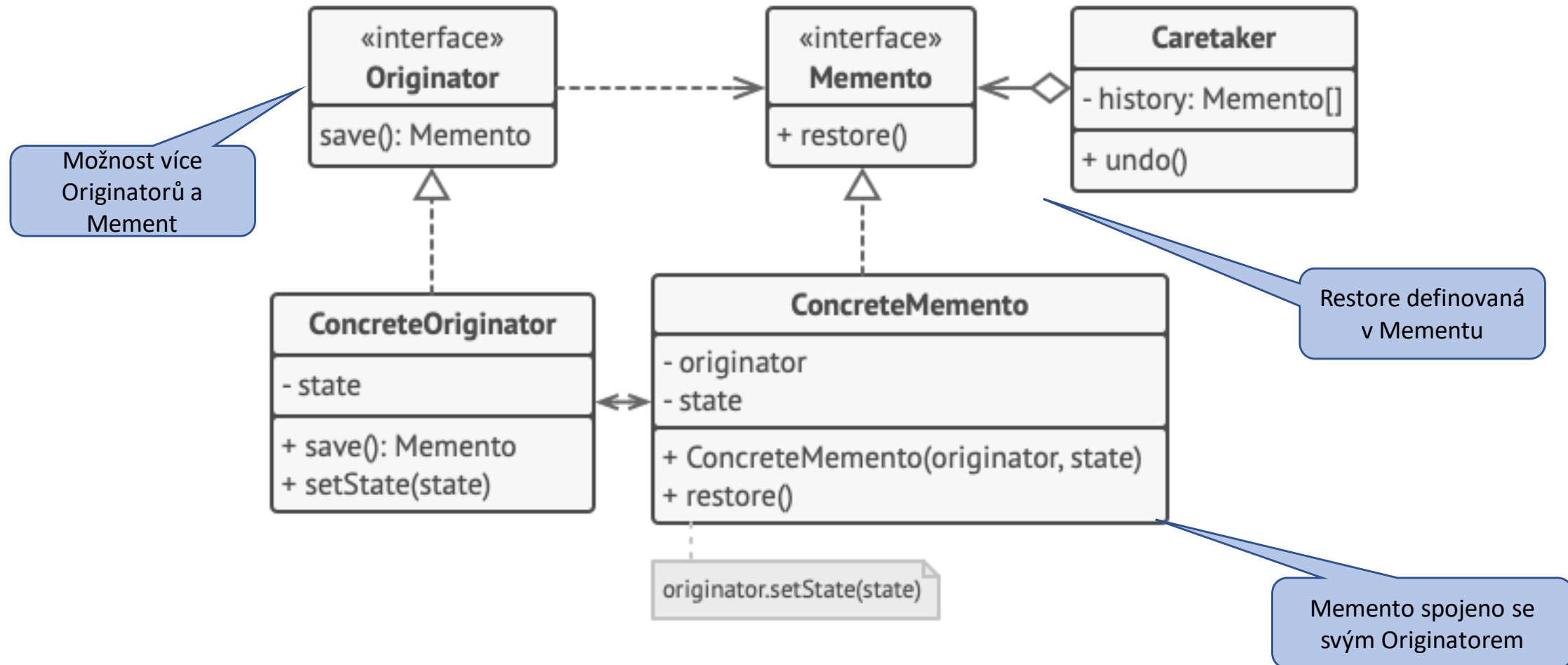
```
public class Editor {
    private StringBuilder text = new StringBuilder();
    public Editor() { }
    public void Write(string text) { }

    public IMemento Save() => new ConcreteMemento(this.text);
    public void Restore(IMemento memento)
    {
        if(!(memento is ConcreteMemento m)) { throw new
Exception();}
        this.text = m.GetState();
    }
}

public class Caretaker
{
    private List<IMemento> mementos = new List<IMemento>();
    private Editor editor = new Editor();
    //...
}
```

Caretaker pracuje pouze přes  
rozhraní, dovnitř mementa nevidí

# Implementace s ještě striktnější encapsulací



# Implementace v dalších jazycích

## C++

- Vnořené typy
- Explicitní rozhraní
- Friend class

```
class Editor {  
private:  
    string text;  
public:  
    Editor(){}  
    void Write(const string& text);  
    Memento CreateMemento();  
    void Restore(shared_ptr<const Memento> memento);  
};
```

```
class Memento {  
private:  
    Memento(const string& text);  
    friend class Editor;  
    string text_;  
};  
  
class Caretaker {  
public:  
    Caretaker();  
    void Backup();  
    void Undo();  
private:  
    vector<shared_ptr<const Memento>> mementos;  
    unique_ptr<Editor> editor_;  
};
```

Implementace  
pomocí friend class



# Implementace v dalších jazycích

## C++

- Vnořené typy
- Explicitní rozhraní
- Friend class

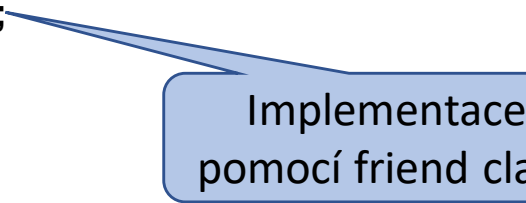
## Java

- Vnořené typy
- Explicitní rozhraní

## Jazyky bez vnořených typů

- Explicitní rozhraní

```
class Memento {  
private:  
    Memento(const string& text);  
    friend class Editor;  
    string text_;  
};  
  
class Caretaker {  
public:  
    Caretaker();  
    void Backup();  
    void Undo();  
private:  
    vector<shared_ptr<const Memento>> mementos;  
    unique_ptr<Editor> editor_;  
};
```



Implementace pomocí friend class

# Použití

- Kdy se hodí?
  - Undo/Redo
  - Zachování Single responsibility a Encapsulation
  - Uchování historie objektu
- Použití
  - Textové editory
  - Databázové transakce
  - Saving mechanika v počítačových hrách
  - UI

## Výhody vs. Nevýhody

- Zachování encapsulace
- Zjednodušení Originatora (uchování vnitřního stavu vně)
- Cena Mementa
- Definice úzkého a širokého rozhraní
- Skrytá cena uchování Mementa
- Sledování životního cyklu Mementa

## Vztahy s ostatními vzory

- Command a Memento můžeme společně použít při implementaci „undo“
- Spolu s Iterátorem můžeme zachytit aktuální iteraci a při nutnosti rollbacknout
- Někdy je jednodušší použít Prototype

# Shrnutí

- Zachycení vnitřního stavu objektu, aby ho bylo možné později tento stav obnovit, bez porušení encapsulace
- 3 hlavní komponenty: Originator, Memento a Caretaker
- Implementace pomocí nested tříd nebo interface

Děkuji za pozornost

# Zdroje

- *Design patterns: elements of reusable object-oriented software* (<http://www.javier8a.com/itc/bd1/articulo.pdf> )
- <https://refactoring.guru/design-patterns/memento>
- [https://www.tutorialspoint.com/design\\_pattern/memento\\_pattern.htm](https://www.tutorialspoint.com/design_pattern/memento_pattern.htm)
- [https://en.wikipedia.org/wiki/Memento\\_pattern](https://en.wikipedia.org/wiki/Memento_pattern)