Autor: Philipp Khachik Version: 1.0 Datum: 2025-01-14

Memento Pattern

Memento Pattern is a behavioral design pattern that allows saving and restoring the previous state of an object without revealing the details of its implementation.

The Memento pattern delegates creating the state snapshots to the actual owner of that state, the originator object.

Project Memento Pattern

This project demonstrates the usage of the Memento pattern for saving and restoring the state of a `Document` object, enabling undo/redo functionality.

Problem

Komplexe Zustandsverwaltung: Eine Textverarbeitungs-App benötigt die Fähigkeit, den Zustand von Dokumenten zu speichern und wiederherzustellen, um "Undo"- und "Redo"-Funktionen zu ermöglichen.

Lösung: Memento Pattern

Originator: Das `Document`-Objekt, das seinen Zustand selbst verwaltet. Memento: Ein Snapshot des `Document`-Zustands, z.B. `DocumentMemento`. Caretaker: Die `DocumentHistory`-Klasse, die die Mementos verwaltet.

Code-Ausschnitte

```
    //public class Document
    //{
    //    private string content;
    //    private Dictionary<string, string> formatting;
    //    private Dictionary<string, string> metadata;
    //    private readonly string documentId;
```

```
//    public Document(string id);
```

```
//    public void UpdateContent(string newContent);
```

```
//    public void ApplyFormatting(string element, string style);
```

```
//    public void UpdateMetadata(string key, string value);
```

```
//    public DocumentMemento CreateMemento(string description);
```

```
//    public void RestoreFromMemento(DocumentMemento memento);
```

```
//    public override string ToString();
//}
```

```
    //public class DocumentMemento : IDocumentMemento
    //{
    //    private readonly string content;
    //    private readonly Dictionary<string, string> formatting;
    //    private readonly Dictionary<string, string> metadata;
    //    private readonly DateTime savedAt;
    //    private readonly string description;
```

```
//    public DateTime SavedAt => savedAt;
//    public string Description => description;
```

```
//    public DocumentMemento(string content, Dictionary<string, string> formatting)
```

```
//    internal string GetContent() => content;
```

```
//    internal Dictionary<string, string> GetFormatting()
```

```
//    internal Dictionary<string, string> GetMetadata()

//}
```

```
    //public class DocumentHistory
    //{
    //    private readonly Stack<DocumentMemento> history;
    //    private readonly Document document;
    //    private readonly int maxHistorySize;
```

```
//    public DocumentHistory(Document document, int maxHistorySize = 10)
```

```
//    public void SaveState(string description)
```

```
//    public bool Undo()
```

```
//    public List<IDocumentMemento> GetHistory()

//}
```

Key Points

Originator-Implementierung: Das `Document`-Objekt erstellt und restauriert seinen Zustand mithilfe von Mementos. Memento-Implementierung: `DocumentMemento` kapselt den Zustand des `Document` -Objekts. Caretaker-Verantwortung: `DocumentHistory` verwaltet die Mementos, um den Zustand des `Document`-Objekts zu speichern und wiederherzustellen.

Vorteile und Nachteile

Vorteile

Encapsulation: Der interne Zustand des Objekts bleibt geschützt.

Flexibilität: Leichtes Hinzufügen von Undo/Redo-Funktionen.

Nachteile

Erhöhter Speicherbedarf: Viele Mementos können viel Speicher verbrauchen.

Erhöhte Komplexität: Zusätzliche Klassen und Logik erforderlich.

Beziehungen zu anderen Mustern

Command Pattern: Kombinierbar für umfassende Undo/Redo-Funktionalität. Iterator Pattern: Verwendbar für die Iteration über Zustandsänderungen. Prototype Pattern: Alternativ zur Vereinfachung des Zustandsspeichers in einigen Fällen.