using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

namespace Memento

{

class Originator

{

private string \_state;

public Originator(string state)

{

this.\_state = state;

Console.WriteLine("Originator: My initial state is: " + state);

}

// The Originator's business logic may affect its internal state.

// Therefore, the client should backup the state before launching

// methods of the business logic via the save() method.

public void DoSomething()

{

Console.WriteLine("Originator: I'm doing something important.");

this.\_state = this.GenerateRandomString(30);

Console.WriteLine($"Originator: and my state has changed to: {\_state}");

}

private string GenerateRandomString(int length = 10)

{

string allowedSymbols = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

string result = string.Empty;

while (length > 0)

{

result += allowedSymbols[new Random().Next(0, allowedSymbols.Length)];

Thread.Sleep(12);

length--;

}

return result;

}

// Saves the current state inside a memento.

public IMemento Save()

{

return new ConcreteMemento(this.\_state);

}

// Restores the Originator's state from a memento object.

public void Restore(IMemento memento)

{

if (!(memento is ConcreteMemento))

{

throw new Exception("Unknown memento class " + memento.ToString());

}

this.\_state = memento.GetState();

Console.Write($"Originator: My state has changed to: {\_state}");

}

}

// The Memento interface provides a way to retrieve the memento's metadata,

// such as creation date or name. However, it doesn't expose the

// Originator's state.

public interface IMemento

{

string GetName();

string GetState();

DateTime GetDate();

}

// The Concrete Memento contains the infrastructure for storing the

// Originator's state.

class ConcreteMemento : IMemento

{

private string \_state;

private DateTime \_date;

public ConcreteMemento(string state)

{

this.\_state = state;

this.\_date = DateTime.Now;

}

// The Originator uses this method when restoring its state.

public string GetState()

{

return this.\_state;

}

// The rest of the methods are used by the Caretaker to display

// metadata.

public string GetName()

{

return $"{this.\_date} / ({this.\_state.Substring(0, 9)})...";

}

public DateTime GetDate()

{

return this.\_date;

}

}

// The Caretaker doesn't depend on the Concrete Memento class. Therefore, it

// doesn't have access to the originator's state, stored inside the memento.

// It works with all mementos via the base Memento interface.

class Caretaker

{

private List<IMemento> \_mementos = new List<IMemento>();

private Originator \_originator = null;

public Caretaker(Originator originator)

{

this.\_originator = originator;

}

public void Backup()

{

Console.WriteLine("\nCaretaker: Saving Originator's state...");

this.\_mementos.Add(this.\_originator.Save());

}

public void Undo()

{

if (this.\_mementos.Count == 0)

{

return;

}

var memento = this.\_mementos.Last();

this.\_mementos.Remove(memento);

Console.WriteLine("Caretaker: Restoring state to: " + memento.GetName());

try

{

this.\_originator.Restore(memento);

}

catch (Exception)

{

this.Undo();

}

}

public void ShowHistory()

{

Console.WriteLine("Caretaker: Here's the list of mementos:");

foreach (var memento in this.\_mementos)

{

Console.WriteLine(memento.GetName());

}

}

}

class Program

{

static void Main(string[] args)

{

// Client code.

Originator originator = new Originator("Super-duper-super-puper-super.");

Caretaker caretaker = new Caretaker(originator);

caretaker.Backup();

originator.DoSomething();

caretaker.Backup();

originator.DoSomething();

caretaker.Backup();

originator.DoSomething();

Console.WriteLine();

caretaker.ShowHistory();

Console.WriteLine("\nClient: Now, let's rollback!\n");

caretaker.Undo();

Console.WriteLine("\n\nClient: Once more!\n");

caretaker.Undo();

Console.WriteLine();

Console.WriteLine("\n\nClient: Once more!\n");

caretaker.Undo();

}

}

}