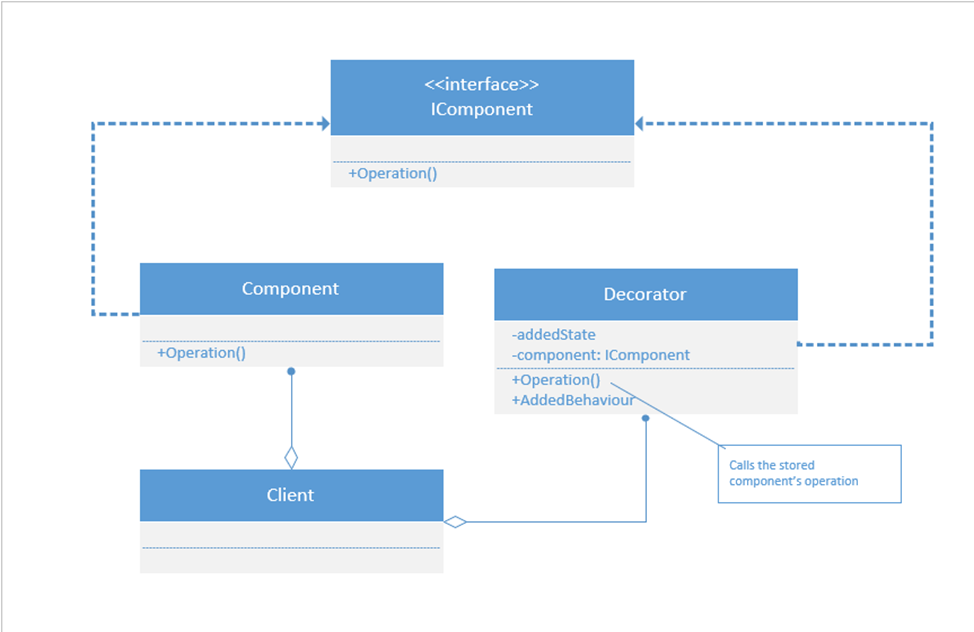
|  |
| --- |
| Name: សោ វិសាលស៊ីង  SO VISALSING  Class M1 Y4 |

HW1: Decorator Pattern

Professor: Dan Vournpheng

1. Intent/Purpose: The purpose of the Decorator pattern is to provide a flexible and dynamic way to extend or augment the behavior of objects at runtime. It allows you to attach additional responsibilities to an object by wrapping it in a series of decorator classes. These decorators add new state or behavior without altering the structure of the original object.
2. Draw a Basic UML Class Diagram & describe all the Players in this diagram:



* Component: An original class of objects that can have operations added or modified (there may be more than one such class)
* Operation: An operation in IComponent objects that can be replaced (there may be several operations)
* IComponent: The interface that identifies the classes of objects that can be decorated (Component is one of these)
* Decorator: A class that conforms to the IComponent interface and adds state and/or behavior (there may be more than one such class).

1. Write code of UML Class Diagram (Basic) of Design Pattern:
2. using System;
3. using System.Security.Cryptography.X509Certificates;
4. interface IComponent { string Operation(); }
5. class Component : IComponent {
6. public string Operation() {
7. return "I am walking";
8. }
9. }
10. class DecoratorA : IComponent
11. {
12. IComponent component;
13. public DecoratorA (IComponent c)
14. {
15. component = c;
16. }
17. public string Operation()
18. {
19. string s = component.Operation();
20. s += "and listening to Classic FM";
21. return s;
22. }
23. }
24. class DecoratorB : IComponent
25. {
26. IComponent component;
27. public string addedState = "past the Coffee Shop";
28. public DecoratorB (IComponent c)
29. {
30. component = c;
31. }
32. public string Operation()
33. {
34. string s = component.Operation();
35. s += "to school";
36. return s;
37. }
38. public string AddedBehavior()
39. {
40. return "and I bought a cappuccino";
41. }
42. }
43. class DecoratorPattern
44. {
45. static void Display (string s, IComponent c)
46. {
47. Console.WriteLine(s + c.Operation());
48. }
49. static void Main()
50. {
51. Console.WriteLine("Decorator Pattern\n");
52. IComponent component = new Component();
53. Display("1. Basic component: ", new DecoratorA(component));
54. Display("2. A-decorated : ", new DecoratorB(component));
55. Display("3. B-decorated : ", new DecoratorB(component));
56. Display("4. B-A-decorated : ", new DecoratorB(new DecoratorA(component)));
57. DecoratorB b = new DecoratorB(new Component());
58. Display("5. A-B-decorated : ", new DecoratorA(b));
59. Console.WriteLine("\t\t\t" + b.addedState + b.AddedBehavior());
60. }
61. }

OUTPUT:

Decorator Pattern

1. Basic component: I am walkingand listening to Classic FM

2. A-decorated : I am walkingto school

3. B-decorated : I am walkingto school

4. B-A-decorated : I am walkingand listening to Classic FMto school

5. A-B-decorated : I am walkingto schooland listening to Classic FM

past the Coffee Shopand I bought a cappuccino

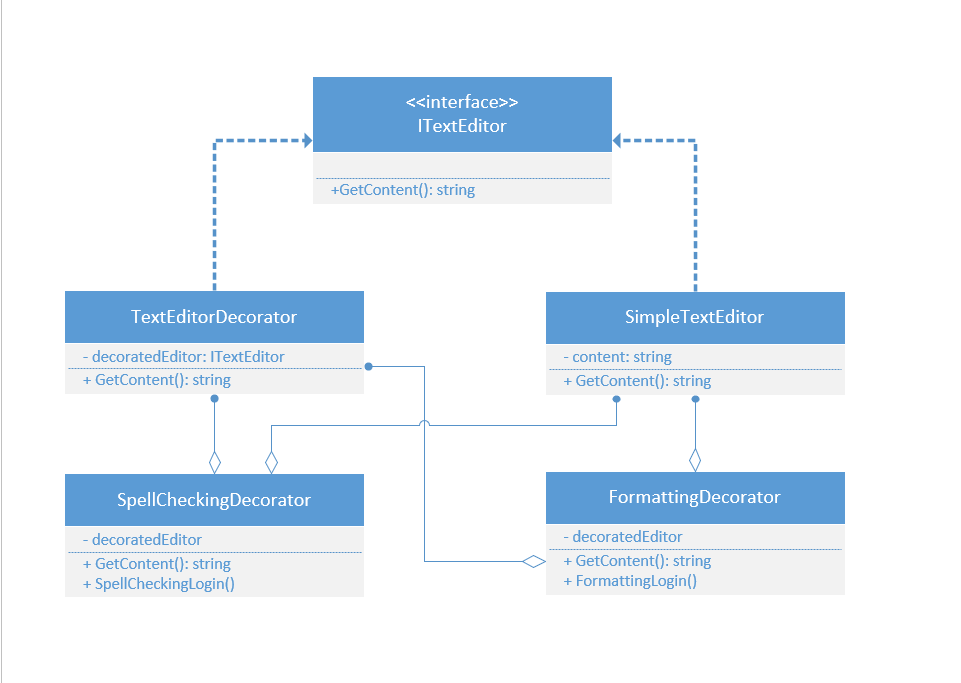
4. Real world example:

Enhance a basic text editor by dynamically adding features like spell checking and formatting.

In this example:

* ITextEditor is the component interface representing the basic text editor.
* SimpleTextEditor is the concrete component representing a basic text editor.
* TextEditorDecorator is the decorator interface for additional features.
* SpellCheckingDecorator and FormattingDecorator are concrete decorators adding spell checking and formatting features, respectively.
* The Program class demonstrates creating a basic text editor and dynamically decorating it with spell checking and formatting features.

5. UML Class Diagram Practice



6. Write code of that UML Diagram:

using System;

using System.Security.Cryptography.X509Certificates;

using System;

// Component interface representing the basic text editor

public interface ITextEditor

{

string GetContent();

}

// Concrete component representing a basic text editor

public class SimpleTextEditor : ITextEditor

{

private string content;

public SimpleTextEditor(string content)

{

this.content = content;

}

public string GetContent()

{

return content;

}

}

// Decorator interface for additional features

public abstract class TextEditorDecorator : ITextEditor

{

protected ITextEditor decoratedEditor;

public TextEditorDecorator(ITextEditor editor)

{

decoratedEditor = editor;

}

public virtual string GetContent()

{

return decoratedEditor.GetContent();

}

}

// Concrete decorator adding spell checking feature

public class SpellCheckingDecorator : TextEditorDecorator

{

public SpellCheckingDecorator(ITextEditor editor) : base(editor) { }

public override string GetContent()

{

string originalContent = base.GetContent();

// Simulate spell checking logic

return originalContent + " (Spell Checked)";

}

}

// Concrete decorator adding formatting feature

public class FormattingDecorator : TextEditorDecorator

{

public FormattingDecorator(ITextEditor editor) : base(editor) { }

public override string GetContent()

{

string originalContent = base.GetContent();

// Simulate formatting logic

return originalContent + " (Formatted)";

}

}

class Program

{

static void Main()

{

// Create a basic text editor

ITextEditor simpleEditor = new SimpleTextEditor("Hello, World!");

// Decorate the editor with spell checking

ITextEditor spellCheckingEditor = new SpellCheckingDecorator(simpleEditor);

// Decorate the editor with formatting

ITextEditor formattedEditor = new FormattingDecorator(spellCheckingEditor);

// Display content

Console.WriteLine("Original Content: " + simpleEditor.GetContent());

Console.WriteLine("Spell Checked Content: " + spellCheckingEditor.GetContent());

Console.WriteLine("Formatted Content: " + formattedEditor.GetContent());

}

}

OUTPUT:

Original Content: Hello, World!

Spell Checked Content: Hello, World! (Spell Checked)

Formatted Content: Hello, World! (Spell Checked) (Formatted)