

Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки  
Кафедра інформаційних систем та технологій

**Лабораторна робота № 6**

з дисципліни «Теорія розробки ПЗ»

Тема: ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento»,  
«Observer», «Decorator»

Варіант №3

Виконав:

студент групи ІА-13

Губенко Єгор Олександрович

Київ, 2023

Тема: ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator».

### **Завдання.**

- Ознайомитися з короткими теоретичними відомостями.
- Реалізувати частину функціонала робочої програми у вигляді класів і їх взаємодій для досягнення конкретних функціональних можливостей.
- Застосування одного з даних шаблонів при реалізації програми.

### **Варіант.**

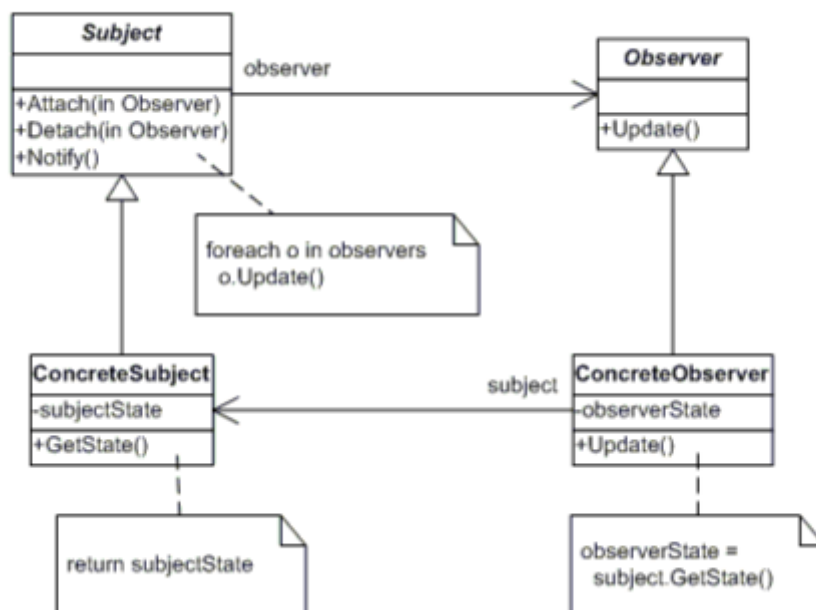
3. Текстовий редактор (strategy, command, observer, template method, flyweight, SOA)

Текстовий редактор повинен вміти розпізнавати текстові файли в будь-якій кодуванні, мати розширені функції редагування: макроси, сніппети, підказки, закладки, перехід на рядок / сторінку, підсвічування синтаксису (для однієї мови програмування або розмітки на розсуд студента).

### **Хід роботи**

#### **Шаблон «Observer»**

Шаблон визначає залежність "один-до-багатьох" таким чином, що коли один об'єкт змінює власний стан, усі інші об'єкти отримують про це сповіщення і мають можливість змінити власний стан також.



```
observer.py x
1 import os
2 from abc import abstractmethod, ABC
3
4
5 class Observer(ABC):
6     @abstractmethod
7     def update(self, hint_text):
8         pass
9
10
11 class PathSubject:
12     def __init__(self):
13         self.observers = []
14
15     def add_observer(self, observer):
16         self.observers.append(observer)
17
18     def remove_observer(self, observer):
19         self.observers.remove(observer)
20
21     def notify_observers(self, file_path):
22         for observer in self.observers:
23             observer.update(file_path)
24
25
26 class EditorObserver(Observer):
27     def update(self, changes):
28         print(f"Editor received an update: {changes}")
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```

30 def start_edit_mode(self):
31     print("Entering edit mode. Type '/finish' to exit editing.")
32
33 def end_edit_mode(self):
34     print("Exiting edit mode.")
35
36 def open_file_by_path(self, file_path):
37     print(f"Opening file by path: {file_path}")
38
39
40 class PathObserver(Observer):
41     def __init__(self, editor):
42         self.editor = editor
43
44     def update(self, file_path):
45         if file_path and os.path.exists(file_path):
46             self.editor.open_file_by_path(file_path)
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
observer.py x hint_strategy.py x
1 from abc import abstractmethod
2 from observer import Observer
3
4
5 class HintStrategy(Observer):
6     @abstractmethod
7     def get_hints(self, hints):
8         pass
9
10
11 class SimpleHintStrategy(HintStrategy):
12     def __init__(self, editor):
13         self.editor = editor
14         self.hint_text = ""
15
16     def get_hints(self, hints):
17         return [hint.hint_text for hint in hints]
18
19     def update(self, hint_text):
20         if hint_text and hint_text.strip().lower() == '/hint':
21             self.hint_text = hint_text
22             print(f"Підказка додана: {self.hint_text}")
23
24
25 class AdvancedHintStrategy(HintStrategy):
26     def get_hints(self, hints):
27         pass
```

**Висновок:** у даній лабораторній я додав декілька класів до свого проєкту, використовуючи паттерн Observer.