



Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №6
**Технологія розроблення програмного
забезпечення**
«Використання шаблону Observer, тема: Download
manager»
Варіант 26

Виконала
студентка групи ІА-13
Танасієнко Анастасія Вячеславівна

Перевірив:

Київ 2023р.

Мета: дослідження та реалізація шаблонів проектування «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Шаблон «Observer»

Editor - це простий текстовий редактор на Java з можливістю додавання та видалення файлів. Він використовує EventManager для відстеження подій "add" і "delete". Метод addFile додає новий файл і сповіщає про це, а метод deleteFile видаляє файл, якщо він був доданий раніше, інакше генерує виключення.

```
1  import java.io.File;
2
3  public class Editor {
4
5      public EventManager events;
6
7      private File file;
8
9      public Editor() {
10         this.events = new EventManager(operations: "add", "delete");
11     }
12
13     public void addFile(String filePath) {
14         this.file = new File(filePath);
15         events.notify(eventType: "add", file);
16     }
17
18     public void deleteFile(String filePath) throws Exception {
19         if (this.file != null) {
20             events.notify(eventType: "delete", file);
21         } else {
22             throw new Exception("Please add a file first.");
23         }
24     }
25 }
```

Інтерфейс EventListener має метод update, який отримує тип події та об'єкт файлу для оновлення слухачів.

```
1  import java.io.File;
2  9 usages 2 implementations
3  public interface EventListener {
4      1 usage 2 implementations
5      void update(String eventType, File file);
6  }
7  |
```

EventManager - це клас, який забезпечує механізм підписки та сповіщення про події. Він має методи subscribe для додавання слухачів і notify для повідомлення їх про зміни. Слухачі визначаються за типом події, який передається разом із файлом.

```
1 import java.io.File;
2 import java.util.ArrayList;
3 import java.util.HashMap;
4 import java.util.List;
5 import java.util.Map;
6
7 public class EventManager {
8     Map<String, List<EventListener>> listeners = new HashMap<>();
9
10    @
11    public EventManager(String... operations) {
12        for (String operation : operations) {
13            this.listeners.put(operation, new ArrayList<>());
14        }
15    }
16
17    public void subscribe(String eventType, EventListener listener) {
18        List<EventListener> users = listeners.get(eventType);
19        users.add(listener);
20    }
21
22    public void notify(String eventType, File file) {
23        List<EventListener> users = listeners.get(eventType);
24        for (EventListener listener : users) {
25            listener.update(eventType, file);
26        }
27    }
28 }
```

```
1 public class EventManager {
2
3     Map<String, List<EventListener>> listeners = new HashMap<>();
4
5     public EventManager(String... operations) {
6         for (String operation : operations) {
7             this.listeners.put(operation, new ArrayList<>());
8         }
9     }
10
11    public void subscribe(String eventType, EventListener listener) {
12        List<EventListener> users = listeners.get(eventType);
13        users.add(listener);
14    }
15
16    public void notify(String eventType, File file) {
17        List<EventListener> users = listeners.get(eventType);
18        for (EventListener listener : users) {
19            listener.update(eventType, file);
20        }
21    }
22 }
```

LogAddListener - клас, який слухає події, реалізує інтерфейс EventListener. При оновленні виводить повідомлення до лог-файлу про виконану операцію та файл.

```
1  import java.io.File;
2
3  1 usage
4  public class LogAddListener implements EventListener {
5      2 usages
6      private File log;
7
8      1 usage
9      public LogAddListener(String fileName) { this.log = new File(fileName); }
10
11     1 usage
12     @Override
13     public void update(String eventType, File file) {
14         System.out.println("Add to log " + log + ": Someone has performed "
15             + eventType + " operation with the following file: " + file.getName());
16     }
17 }
```

LogDeleteListener - клас, який слухає події, реалізує інтерфейс EventListener. При оновленні виводить повідомлення до лог-файлу про виконану операцію видалення та файл.

```
1  import java.io.File;
2
3  1 usage
4  public class LogDeleteListener implements EventListener {
5      2 usages
6      private File log;
7
8      1 usage
9      public LogDeleteListener(String fileName) { this.log = new File(fileName); }
10
11     1 usage
12     @Override
13     public void update(String eventType, File file) {
14         System.out.println("Delete from log " + log + ": Someone has performed "
15             + eventType + " operation with the following file: " + file.getName());
16     }
17 }
```

Клас Main має метод main, який створює об'єкт редактора (Editor) та підписує два слухачі - LogAddListener та LogDeleteListener - на події "add" і "delete" відповідно. Потім викликає методи додавання та видалення файлу через редактор і виводить відповідні повідомлення до лог-файлу.



```
1 public class Main {
2     public static void main(String[] args) {
3         Editor editor = new Editor();
4         editor.events.subscribe( eventType: "add", new LogAddListener( fileName: "/path/to/log/file.txt"));
5         editor.events.subscribe( eventType: "delete", new LogDeleteListener( fileName: "/path/to/log/file.txt"));
6
7         try {
8             editor.addFile( filePath: "test.txt");
9             editor.deleteFile( filePath: "test.txt");
10        } catch (Exception e) {
11            e.printStackTrace();
12        }
13    }
14 }
15
```

Висновок:

У цій лабораторній роботі були досліджені та реалізовані п'ять важливих шаблонів проектування: «Abstract Factory», «Factory Method», «Memento», «Observer» та «Decorator». Кожен із них має свою унікальну функціональність та може бути використаний для вирішення різноманітних завдань у програмній розробці.

- Шаблон "Abstract Factory" дозволяє створювати сімейства взаємодіючих об'єктів без прив'язки до конкретних класів. Він надає абстрактний інтерфейс для створення сімейств пов'язаних або взаємозалежних об'єктів.
- Шаблон "Factory Method" визначає загальний інтерфейс для створення об'єктів, але залишає вибір конкретного класу-продукту до підкласів. Він дозволяє створювати об'єкти, не вказуючи конкретний клас.
- Шаблон "Memento" дозволяє зберігати стан об'єкта так, щоб його можна було відновити в майбутньому без розкриття деталей його реалізації.
- Шаблон "Observer" визначає залежність одного об'єкта від змін у іншому об'єкті, гарантуючи, що при зміні стану одного об'єкта всі його залежності будуть автоматично сповіщені та оновлені.
- Шаблон "Decorator" дозволяє динамічно надавати об'єкту нові функції, обгортаючи його в інші класи, що реалізують однаковий інтерфейс.

Робота над цими шаблонами дозволила отримати глибше розуміння принципів об'єктно-орієнтованого програмування та паттернів проектування. Кожен з цих шаблонів має свої конкретні випадки застосування та може бути важливим інструментом для розробників у різних сценаріях розробки програмного забезпечення.