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 №8
# Технологія розроблення програмного забезпечення

ШАБЛОНИ «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR»

Варіант 24

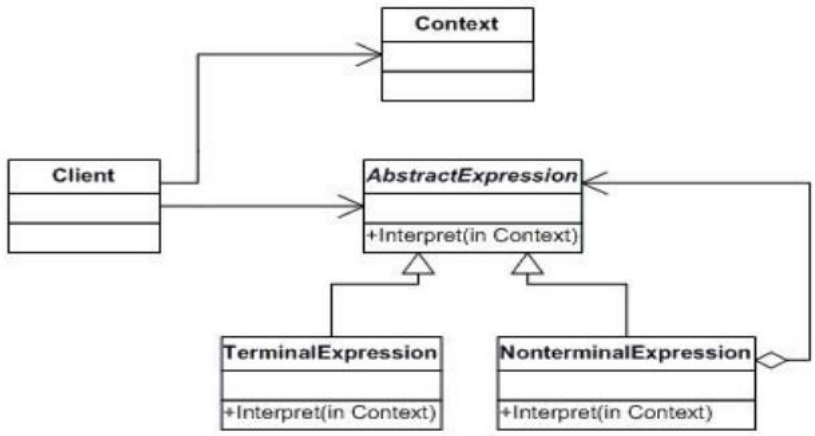
Виконала
студентка групи ІА-13
Сидорук Аліна Костянтинівна

Перевірив:
Мягкий Михайло Юрійович

Київ 2023р.

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

Даний шаблон використовується для подання граматики і інтерпретатора для вибраної мови (наприклад, скриптової). Граматика мови представлена термінальними і нетермінальними символами, кожен з яких інтерпретується в контексті використання. Клієнт передає контекст і сформовану пропозицію в використовувану мову в термінах абстрактного синтаксичного дерева (деревоподібна структура, яка однозначно визначає ієрархію виклику підвиразів), кожен вираз інтерпретується окремо з використанням контексту. У разі наявності дочірніх виразів, батьківський вираз інтерпретує спочатку дочірні (рекурсивно), а потім обчислює результат власної операції. Шаблон зручно використовувати в разі невеликої граматики (інакше розростеться кількість використовуваних класів) і відносно простого контексту (без взаємозалежностей і т.п.). Даний шаблон визначає базовий каркас інтерпретатора, який за допомогою рекурсії повертає результат обчислення пропозиції на основі результатів окремих елементів. При використанні даного шаблону дуже легко реалізовується і розширюється граматика, а також додаються нові способи інтерпретації виразів




код реалізує систему інтерпретації JSON-даних для створення скриптів у програмі, яка  працює з автоматизацією та макросами. Основна ідея полягає в тому, щоб перетворити JSON-структуру на внутрішні об'єкти доменної моделі, такі як ScriptActions, ScriptDelay, ScriptRepeat тощо.

Ключові Компоненти

JsonInterpreter<T>: Це інтерфейс, який визначає метод interpret, призначений для перетворення JSONObject в об'єкт типу T. Цей інтерфейс є основою для патерну "Інтерпретатор" у вашому коді.

```
package com.example.trpzmacrosproject.interpreters.components;

import org.json.JSONObject;

public interface JsonInterpreter<T> {

    T interpret(JSONObject json);

}
```

Конкретні Інтерпретатори: Класи, такі як RepeatInterpreter, DelayInterpreter тощо, імплементують JsonInterpreter<T>. Кожен інтерпретатор обробляє певну частину JSON-даних і перетворює їх у специфічні об'єкти доменної моделі (наприклад, ScriptRepeat для повторень).

```
package com.example.trpzmacrosproject.interpreters.components;

import
com.example.trpzmacrosproject.interpreters.exceptions.DelayParsingException;
import com.example.trpzmacrosproject.scripts.ScriptRepeat;
import org.json.JSONObject;
import org.springframework.stereotype.Component;

import java.util.function.Consumer;

@Component
public class RepeatInterpreter implements JsonInterpreter<ScriptRepeat> {
    /**
     * Отримання об'єкта ScriptRepeat із JSON-об'єкта
     * @param script JSON-об'єкт, що представляє повторення
     * @return ScriptRepeat об'єкт, що містить інформацію про повторення
     */
    @Override
    public ScriptRepeat interpret(JSONObject script) {
        // Створення нового об'єкта ScriptRepeat
        ScriptRepeat result = new ScriptRepeat();

        // Встановлення значень, якщо вони присутні в JSON-об'єкті
        setIfPresent(script, "second", result::setExactSecond);
        setIfPresent(script, "minutes", result::setExactMinute);
        setIfPresent(script, "hours", result::setExactHour);
        setIfPresent(script, "dayOfMonth", result::setExactDayOfMonth);
        setIfPresent(script, "month", result::setExactMonth);
        setIfPresent(script, "dayOfWeek", result::setExactDayOfWeek);
        return result;
    }

    /**
     * Встановлення значення, якщо воно присутнє в JSON-об'єкті
     *
     * @param script JSON-об'єкт
     * @param key    ключ поля
     * @param setter функція-встановлювач для відповідного поля в
ScriptRepeat
     */
    private void setIfPresent(JSONObject script, String key,
Consumer<Integer> setter) {

        try {
            // Перевірка, чи присутнє значення за ключем у JSON-об'єкті
            if (script.has(key)) {
                // Якщо так, викликається функція-встановлювач для
встановлення значення
                setter.accept(script.getInt(key));
            }
        } catch (Exception e) {
            throw new DelayParsingException("Can't parse ScriptRepeat: " +
e.getMessage(), e);
        }
    }
```

```
/*public ScriptRepeat getScriptRepeat(JSONObject script) {
        // Створення нового об'єкта ScriptRepeat
        ScriptRepeat result = new ScriptRepeat();

        // Встановлення значень, якщо вони присутні в JSON-об'єкті
        setIfPresent(script, "second", result::setExactSecond);
        setIfPresent(script, "minutes", result::setExactMinute);
        setIfPresent(script, "hours", result::setExactHour);
        setIfPresent(script, "dayOfMonth", result::setExactDayOfMonth);
        setIfPresent(script, "month", result::setExactMonth);
        setIfPresent(script, "dayOfWeek", result::setExactDayOfWeek);
        return result;
    }
*/
}
```

```
package com.example.trpzmacrosproject.interpreters.components;

import
com.example.trpzmacrosproject.interpreters.exceptions.DelayParsingException;
import com.example.trpzmacrosproject.scripts.ScriptDelay;
import org.json.JSONObject;
import org.springframework.stereotype.Component;

@Component
public class DelayInterpreter implements JsonInterpreter<ScriptDelay> {
    /**
     * Отримання об'єкта ScriptDelay із JSON-об'єкта
     * @param script JSON-об'єкт, що представляє затримку
     * @return ScriptDelay об'єкт, що містить інформацію про затримку
     */
    @Override
    public ScriptDelay interpret(JSONObject script) {
        // Створення нового об'єкта ScriptDelay
        ScriptDelay result = new ScriptDelay();

        // Встановлення годин, якщо вони присутні в JSON-об'єкті
        if (script.has("hours")) {
            result.setHours(getInt(script, "hours"));
        }
        // Встановлення хвилин, якщо вони присутні в JSON-об'єкт
        if (script.has("minutes")) {
            result.setMinutes(getInt(script, "minutes"));
        }
        // Встановлення секунд, якщо вони присутні в JSON-об'єкті
        if (script.has("seconds")) {
            result.setSeconds(getInt(script, "seconds"));
        }
        return result;
    }


  /*  public ScriptDelay getScriptDelay(JSONObject script) {

        // Створення нового об'єкта ScriptDelay
        ScriptDelay result = new ScriptDelay();

        // Встановлення годин, якщо вони присутні в JSON-об'єкті
        if (script.has("hours")) {
            result.setHours(getInt(script, "hours"));
        }
```

```
        // Встановлення хвилин, якщо вони присутні в JSON-об'єкт
        if (script.has("minutes")) {
            result.setMinutes(getInt(script, "minutes"));
        }
        // Встановлення секунд, якщо вони присутні в JSON-об'єкті
        if (script.has("seconds")) {
            result.setSeconds(getInt(script, "seconds"));
        }
        return result;
    }*/

    /**
     * Отримання цілочисельного значення з JSON-об'єкта за заданим ім'ям
     * @param script JSON-об'єкт
     * @param name ім'я поля
     * @return int цілочисельне значення поля
     * @throws DelayParsingException виняток, якщо не вдається здійснити
парсинг значення
     */
    private int getInt(JSONObject script, String name) throws
DelayParsingException {
        try {
            // Отримання цілочисельного значення за заданим ім'ям
            return script.getInt(name);
        } catch (Exception e) {
            throw new DelayParsingException("Can't parse delay: " +
e.getMessage(), e);
        }
    }



}
```

ScriptCreator: Цей клас використовує інтерпретатори для обробки входу у вигляді JSON-файлу і створення з нього складнішого об'єкта Script. ScriptCreator служить як координатор, який делегує обробку різних частин JSON відповідним інтерпретаторам.

```
package com.example.trpzmacrosproject.interpreters;

import com.example.trpzmacrosproject.events.Event;
import
com.example.trpzmacrosproject.interpreters.components.ActionsInterpreter;
import
com.example.trpzmacrosproject.interpreters.components.DelayInterpreter;
import
com.example.trpzmacrosproject.interpreters.components.EventInterpreter;
import
com.example.trpzmacrosproject.interpreters.components.RepeatInterpreter;
import
com.example.trpzmacrosproject.interpreters.exceptions.ParsingJsonException;
import com.example.trpzmacrosproject.scripts.Script;
import com.example.trpzmacrosproject.scripts.ScriptActions;
import com.example.trpzmacrosproject.scripts.ScriptDelay;
import com.example.trpzmacrosproject.scripts.ScriptRepeat;
import lombok.AllArgsConstructor;
import org.json.JSONArray;
import org.json.JSONObject;
```

```
import org.json.JSONTokener;
import org.springframework.stereotype.Component;

import java.io.File;
import java.io.FileReader;
import java.util.List;

@Component
@AllArgsConstructor
public class ScriptCreator {
    EventInterpreter eventInterpreter;
    DelayInterpreter delayInterpreter;
    RepeatInterpreter repeatInterpreter;
    ActionsInterpreter actionsInterpreter;

    public Script getScript(File scriptFile) {
        try {
            JSONObject root = new JSONObject(new JSONTokener(new
FileReader(scriptFile)));

            //Обробка дій
            Script result = new Script(getScriptActions(root));

            // Обробка повторень
            boolean hasRepeat = false;
            if (root.has("repeat")) {
                result.setRepeat(getScriptRepeat(root));
                hasRepeat = true;
            }

            // Обробка евентів
            if (!hasRepeat) {
                result.addAllEvents(getEvents(root));
            }

            // Обробка кількості повторень
            if (root.has("repeatAmount")) {
                int repeatAmount = root.getInt("repeatAmount");
                result.setRepeatAmount(repeatAmount);
            }

            // Обробка затримки
            if (root.has("delay")) {
                result.setDelay(getScriptDelay(root));
            }

            return result;
        } catch (Exception e) {
            throw new ParsingJsonException(String.format("Exception in
parsing file \"%s\": %s",
                    scriptFile.getName(), e.getMessage()), e);
        }
    }

    private ScriptDelay getScriptDelay(JSONObject root) {
        JSONObject jsonObject = root.getJSONObject("delay");
        return delayInterpreter.interpret(jsonObject);
    }

    private List<Event> getEvents(JSONObject root) {
        JSONArray eventsJson = root.getJSONArray("conditions");
        return eventInterpreter.getEvents(eventsJson);
    }
```

```
    private ScriptRepeat getScriptRepeat(JSONObject root) {
        JSONObject jsonObject = root.getJSONObject("repeat");
        return repeatInterpreter.interpret(jsonObject);
    }

    private ScriptActions getScriptActions(JSONObject root) {
        JSONArray actionsJson = root.getJSONArray("actions");
        return actionsInterpreter.getScriptActions(actionsJson);
    }


}
```

Реалізація Патерну "Інтерпретатор"

Абстракція Інтерпретації: Інтерфейс JsonInterpreter<T> абстрагує процес інтерпретації, дозволяючи різним інтерпретаторам мати специфічну для себе реалізацію. Це забезпечує гнучкість, оскільки можна легко додавати нові інтерпретатори для різних типів даних.

Специфічна Інтерпретація: Кожен конкретний інтерпретатор, як RepeatInterpreter, фокусується на певній частині JSON-структури і відповідає за її перетворення у відповідний об'єкт доменної моделі.

Делегування та Композиція: ScriptCreator використовує делегування для обробки JSON-файлу, передаючи різні частини JSON відповідним інтерпретаторам. Це забезпечує чітке відокремлення відповідальності і сприяє композиції.

Висновок : У цій лабораторній роботі були досліджені ШАБЛОНИ «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR» та реалізован Шаблон «Interpreter»