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
ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»
Варіант 8

Виконав
студент групи ІА-13
Крутиус Владислав Віталійович

Перевірив:

Київ 2023р.

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

Патерни Decorator та Observer

Розглянемо клас Terminal

```
public class Terminal extends TerminalView {

    private PtyProcess process;
    private final ObjectProperty<Writer> outputWriterProperty;
    private final Path terminalPath;
    private String[] termCommand;
    private final LinkedBlockingQueue<String> commandQueue;

    public Terminal() {
        this(null, null);
    }

    public Terminal(TerminalConfig terminalConfig, Path
terminalPath) {

        setTerminalConfig(terminalConfig);
        this.terminalPath = terminalPath;
        outputWriterProperty = new SimpleObjectProperty<>();
        commandQueue = new LinkedBlockingQueue<>();
    }

    @WebkitCall
    public void command(String command) {
        try {
            commandQueue.put(command);
        } catch (final InterruptedException e) {
            throw new RuntimeException(e);
        }
        ThreadHelper.start(() -> {
            try {
                final String commandToExecute =
commandQueue.poll();
                getOutputWriter().write(commandToExecute);
                getOutputWriter().flush();
            } catch (final IOException e) {
                throw new RuntimeException(e);
            }
        });
    }

    @Override
    public void onTerminalReady() {
```

```
        ThreadHelper.start(() -> {
            try {
                initializeProcess();
            } catch (final Exception e) {
                throw new RuntimeException(e);
            }
        });
    }

    private void initializeProcess() throws Exception {
        final Path dataDir = getDataDir();
        if (SystemUtils.IS_OS_WINDOWS) {
            this.termCommand =
getTerminalConfig().getWindowsTerminalStarter().split("\\s+");
        } else {
            this.termCommand =
getTerminalConfig().getUnixTerminalStarter().split("\\s+");
        }

        final Map<String, String> envs = new
HashMap<>(System.getenv());
        envs.put("TERM", "xterm");

        System.setProperty("PTY_LIB_FOLDER",
dataDir.resolve("libpty").toString());

        if (Objects.nonNull(terminalPath) &&
Files.exists(terminalPath)) {
            this.process = PtyProcess.exec(termCommand, envs,
terminalPath.toString());
        } else {
            this.process = PtyProcess.exec(termCommand, envs);
        }

        columnsProperty().addListener(evt -> updateWinSize());
        rowsProperty().addListener(evt -> updateWinSize());
        updateWinSize();
        String
defaultCharEncoding=System.getProperty("file.encoding");
        setInputReader(new BufferedReader(new
InputStreamReader(process.getInputStream(),
defaultCharEncoding)));
        setErrorReader(new BufferedReader(new
InputStreamReader(process.getErrorStream(),
defaultCharEncoding)));
        setOutputWriter(new BufferedWriter(new
OutputStreamWriter(process.getOutputStream(),
defaultCharEncoding)));
```

```
        focusCursor();

        countDownLatch.countDown();

        process.waitFor();
    }

    private Path getDataDir() {
        final String userHome = System.getProperty("user.home");
        final Path dataDir =
Paths.get(userHome).resolve(".terminalfx");
        return dataDir;
    }

    public Path getTerminalPath() {
        return terminalPath;
    }

    private void updateWinSize() {
        try {
            process.setWinSize(new WinSize(getColumns(),
getRows()));
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    public ObjectProperty<Writer> outputWriterProperty() {
        return outputWriterProperty;
    }

    public Writer getOutputWriter() {
        return outputWriterProperty.get();
    }

    public void setOutputWriter(Writer writer) {
        outputWriterProperty.set(writer);
    }

    public PtyProcess getProcess() {
        return process;
    }

}
```

Використання такого патерну як Decorator можна побачити при створенні об’єктів BufferedReader та BufferedWriter.
Патерн Observer використовується при використанні eventListeners

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, обгортуючи його в інші класи, що реалізують однаковий інтерфейс.