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 №7

Технології розроблення програмного забезпечення

ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

Варіант №14

Виконав:

студент групи ІА-14,

Міщук Максим Дмитрович

Перевірив:

Мягкий М.Ю.

Київ 2023

Тема роботи: Шаблони «Mediator», «Facade», «Bridge», «Template method».

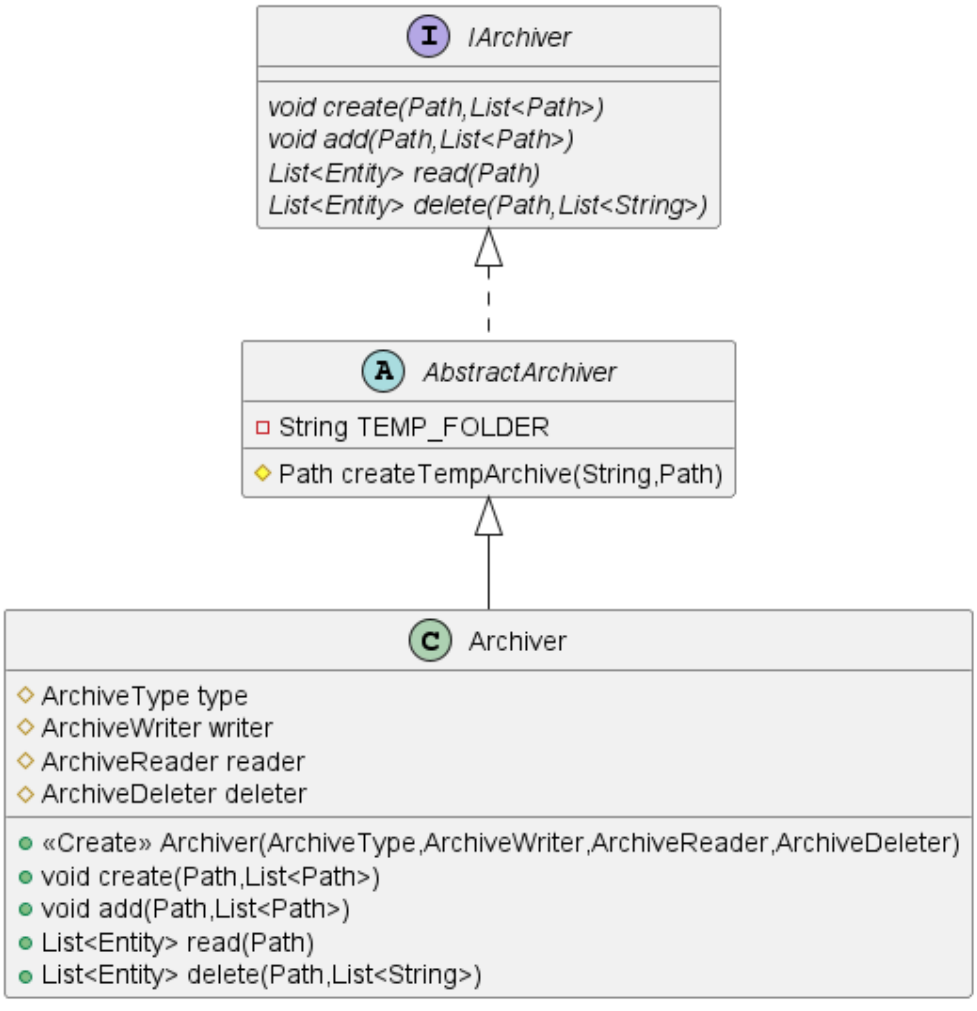
Вхідні дані:

..14 Архіватор (strategy, adapter, factory method, facade, visitor, p2p)

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) - додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

Хід роботи:

Суть шаблону фасаду полягає в тому, щоб дати змогу використовувати простий інтерфейс, котрий вже звертається до складної системи. Інтерфейсом взаємодії став клас архіватор, що приймає в себе тільки ту інформацію що саме потрібна від користувача, а подальші ініціалізації робить в середині без участі нього.



Діаграма класів

IArchive являє собою інтерфейс де задекларовано основні дії архіватора. *AbstractArchiver* як зрозуміло з назви є абстрактним. В ньому реалізована логіка ініціалізації тимчасової папки та архівів у ній. Таким чином усі видозміни відбуваються не напряму в архіві, а в її тимчасовій копії. Це зроблено в цілях уберегти оригінал від пошкоджень, у разі викидані виключень. Метод створення тимчасового архіву повертає відповідно шлях до цього архіву. Далі буде описано реалізацію класу *Archiver*, що наслідується від вище описаних класів.

В основі кожної дії класу лежить створення відповідний потоків на основі заданих параметрів. Самі потоки створюються у конструкції *try-with-resources*, що гарантується закриття потоків після їх виконання у незалежності від результату виконання.

У методів що виконують дії модифікації та видалення, напочатку створюється тимчасовий архів, з яким будуть відбуватися подальші дії. Після успішного завершення операції, відбувається перенесення нового архіву до оригінальної директорії із заміною минулого варіанту. Блок final відповідає за видалення тимчасового файлу, в незалежності від вихідного результату програми.

Код реалізації:

```
public interface IArchiver {
    void create(Path archivePath, List<Path> filePaths) throws IOException, ArchiveException;
    void add(Path archivePath, List<Path> filePaths) throws IOException, ArchiveException;
    List<Entity> read(Path archivePath, ArchiveReadingType archiveReadingType) throws IOException,
ArchiveException;
    List<Entity> delete(Path archivePath, List<String> fileNames) throws IOException, ArchiveException;
}
```

IArchiver.java

```
public abstract class AbstractArchiver implements IArchiver {
    private final String TEMP_FOLDER = "temp";

    private void initTempFolder() throws IOException{
        Path tempPath = Path.of(TEMP_FOLDER);
        if (Files.notExists(tempPath))
            Files.createDirectory(tempPath);
    }

    protected Path createTempArchive(String folder, Path archivePath) throws IOException {
        initTempFolder();
        Path folderPath = Path.of(TEMP_FOLDER, folder);
        if (Files.notExists(folderPath))
            Files.createDirectory(folderPath);

        Path tempArchivePath = folderPath.resolve(archivePath.getFileName().toString());
        tempArchivePath = FileUtils.getFreePath(tempArchivePath);

        return Files.createFile(tempArchivePath);
    }
}
```

AbstractArchiver.java

```java
public class Archiver extends AbstractArchiver {
    protected final ArchiveType archiveType;
    protected final ArchiveWriter writer;
    protected final ArchiveReader reader;
    protected final ArchiveDeleter deleter;

    public Archiver(ArchiveType archiveType, ArchiveWriter writer, ArchiveReader reader, ArchiveDeleter
deleter) {
        this.writer = writer;
        this.reader = reader;
        this.deleter = deleter;
        this.archiveType = archiveType;
    }

    @Override
    public void create(Path archivePath, List<Path> filePaths) throws IOException, ArchiveException {
        Path savedPath = FileUtils.getFreePath(archivePath);
        try (
                BufferedOutputStream bos = new BufferedOutputStream(Files.newOutputStream(savedPath));
                ArchiveOutputStream aos = new ArchiveStreamFactory()
                        .createArchiveOutputStream(archiveType.name(), bos)
        ) {
            writer.write(aos, filePaths);
        }
    }

    @Override
    public void add(Path archivePath, List<Path> filePaths) throws IOException, ArchiveException {
        Path tempArchivePath = createTempArchive("archives", archivePath);

        boolean isAdded = false;
        try (
                BufferedInputStream bis = new BufferedInputStream(Files.newInputStream(archivePath));
                BufferedOutputStream bos = new
BufferedOutputStream(Files.newOutputStream(tempArchivePath));
                ArchiveInputStream ais = new ArchiveStreamFactory()
                        .createArchiveInputStream(archiveType.name(), bis);
                ArchiveOutputStream aos = new ArchiveStreamFactory()
                        .createArchiveOutputStream(archiveType.name(), bos)

        ) {
            writer.write(ais, aos, filePaths);
            isAdded = true;
        } finally {
            if (isAdded) {
                Files.copy(tempArchivePath, archivePath, StandardCopyOption.REPLACE_EXISTING);
            }
            Files.delete(tempArchivePath);
        }
    }

    @Override
    public List<Entity> read(Path archivePath, ArchiveReadingType archiverReadingType) throws IOException,
ArchiveException {
        try (
                BufferedInputStream bis = new BufferedInputStream(Files.newInputStream(archivePath));
                ArchiveInputStream ais = new ArchiveStreamFactory().createArchiveInputStream(bis)
        ) {
            return switch (archiverReadingType) {
                case BASIC -> reader.readBasic(ais);
                case FULL -> reader.readFull(ais);
            };
        }
    }

    @Override
    public List<Entity> delete(Path archivePath, List<String> fileNames) throws IOException,
ArchiveException {
        Path tempArchivePath = createTempArchive("archives", archivePath);
```

```
        List<Entity> deletedEntities = null;
        try (
                BufferedInputStream bis = new BufferedInputStream(Files.newInputStream(archivePath));
                BufferedOutputStream bos = new
BufferedOutputStream(Files.newOutputStream(tempArchivePath));
                ArchiveInputStream ais = new ArchiveStreamFactory()
                        .createArchiveInputStream(archiveType.name(), bis);
                ArchiveOutputStream aos = new ArchiveStreamFactory()
                        .createArchiveOutputStream(archiveType.name(), bos)
        ) {
            deletedEntities = deleter.delete(ais, aos, fileNames);
            return deletedEntities;
        } finally {
            if (deletedEntities != null) {
                Files.copy(tempArchivePath, archivePath, StandardCopyOption.REPLACE_EXISTING);
            }
            Files.delete(tempArchivePath);
        }
    }
}
```

Archiver.java

Висновок:

За допомоги шаблону фасад, вдалося створити вхідний інтерфейс за для спрощення користування функціоналом. Так, наприклад, якщо треба створити архів певного типу, то користувач надсилає тільки посилання для збереження та список файлів, що буде стиснуто. А вже в середині будуть відбуватися необхідні налаштування перед виконанням дії.

Посилання на репозиторій: Maxim-Mishchuk/trpz-archiver at develop (github.com)