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 №4

Технології розроблення програмного забезпечення

Шаблони «Singleton», «Iterator», «Proxy», «State», «Strategy»

Варіант №14

Виконав:

студент групи ІА-14,

Міщук Максим Дмитрович

Перевірив:

Мягкий М.Ю.

Київ 2023

Тема роботи: Шаблони «Singleton», «Iterator», «Proxy», «State», «Strategy».

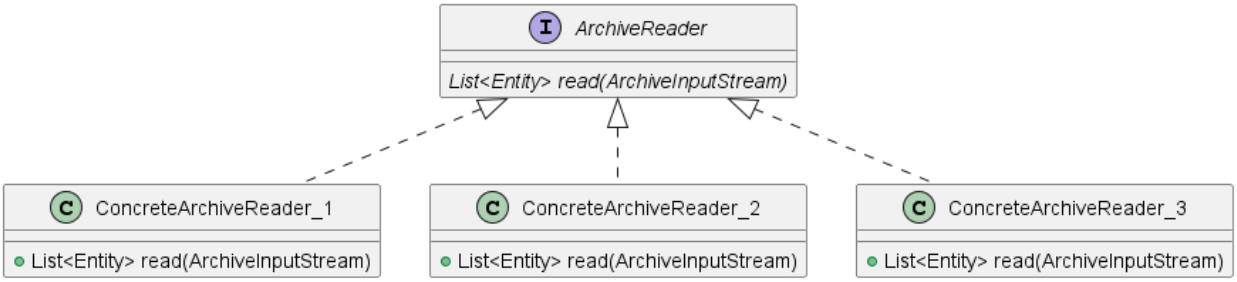
Вхідні дані:

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

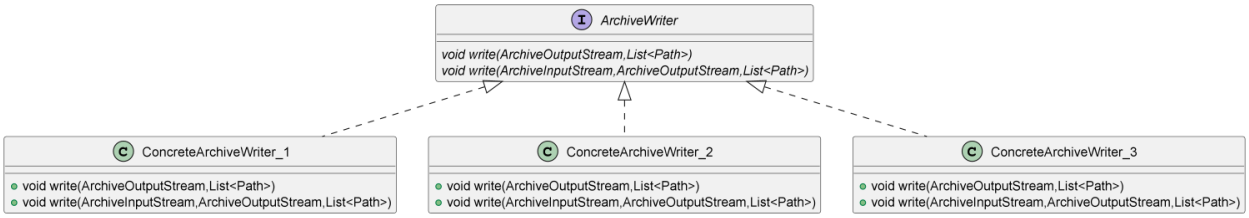
Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) - додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів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

Хід роботи:

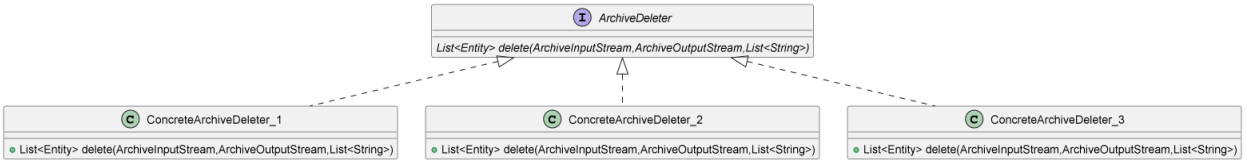
В застосунку є можливість створювати, редагувати, читати архіви. Але для кожного типу архівів треба свій підхід для роботу. Тому можна створити абстракції «Записник», «Читальник», «Видалятор», з відповідними абстрактними методами, а потім на основі них робити вже готові модулі, що будуть підставлятися відповідно до запиту користувача. Далі можна буде побачити приклад діаграми класів цих дій:



«Читальник»



«Записник»



«Видалятор»

Через використання бібліотеки Apache Commons Compress вдалося створити універсальні базові операції для роботи з архівами. Проте хоч зараз клас і один, через універсальний інтерфейс можна буде без проблем додавати реалізацію відповідних функцій, з використанням інших бібліотек.

Код реалізації:

```
public interface ArchiveWriter {
    void write(ArchiveOutputStream aos, List<Path> pathList) throws IOException;
    void write(ArchiveInputStream ais, ArchiveOutputStream aos, List<Path> pathList) throws
IOException;
}
```

ArchiveWriter.java

```
public interface ArchiveReader {
    List<Entity> readBasic(ArchiveInputStream ais) throws IOException;
    List<Entity> readFull(ArchiveInputStream ais) throws IOException;
}
```

ArchiveReader.java

```
public interface ArchiveDeleter {
    List<Entity> delete(ArchiveInputStream ais, ArchiveOutputStream aos, List<String>
entityNames) throws IOException;
}
```

ArchiveDeleter.java

```java
public class BasicArchiveWriter implements ArchiveWriter {
    @Override
    public void write(ArchiveOutputStream aos, List<Path> pathList) throws IOException {
        for (Path path : pathList) {
            if (Files.isDirectory(path)) {
                List<Path> localPathList = getAllFilePathsFromDirectory(path);
                for (Path localPath: localPathList) {
                    String cutName = cutPathName(path, localPath);
                    ArchiveEntry newEntry = aos.createArchiveEntry(localPath, cutName);

                    aos.putArchiveEntry(newEntry);
                    try (InputStream is = Files.newInputStream(localPath)) {
                        IOUtils.copy(is, aos);
                    }
                    aos.closeArchiveEntry();
                }
            } else {
                ArchiveEntry newEntry = aos.createArchiveEntry(path,
path.getFileName().toString());
                aos.putArchiveEntry(newEntry);
                try (InputStream is = Files.newInputStream(path)) {
                    IOUtils.copy(is, aos);
                }
                aos.closeArchiveEntry();
            }
        }
    }

    @Override
    public void write(ArchiveInputStream ais, ArchiveOutputStream aos, List<Path> pathList)
throws IOException {
        ArchiveEntry currentEntry;
        List<String> namesList = new LinkedList<>();
        for (Path path : pathList) {
            if (Files.isDirectory(path)) {
                List<Path> localPaths = getAllFilePathsFromDirectory(path);
                namesList.addAll(
                        localPaths.stream().map(localPath -> cutPathName(path,
localPath)).toList()
                );
            } else {
                namesList.add(path.getFileName().toString());
            }
        }
        while ((currentEntry = ais.getNextEntry()) != null) {
            String entryName = Path.of(currentEntry.getName()).toString();

            if (!namesList.contains(entryName)) {
                aos.putArchiveEntry(currentEntry);
                IOUtils.copy(ais, aos);
                aos.closeArchiveEntry();
            }
        }
        write(aos, pathList);
    }

    private String cutPathName(Path rootPath, Path filePath) {
        return filePath.toString()
                .replace(rootPath.getParent().toString(), "");
    }

    private List<Path> getAllFilePathsFromDirectory(Path dir) throws IOException {
        try (Stream<Path> walk = Files.walk(dir)) {
            return walk.filter(Files::isRegularFile).toList();
    }
}
```

BasicArchiveWriter.java

```java
public class BasicArchiveReader implements ArchiveReader {
    @Override
    public List<Entity> readBasic(ArchiveInputStream ais) throws IOException {
        List<Entity> entityList = new LinkedList<>();

        ArchiveEntry currentEntry;
        while ((currentEntry = ais.getNextEntry()) != null) {
            entityList.add(Entity.create(currentEntry.getName()));
        }

        return entityList;
    }

    @Override
    public List<Entity> readFull(ArchiveInputStream ais) throws IOException {
        List<Entity> entityList = new LinkedList<>();

        ArchiveEntry currentEntry;
        while ((currentEntry = ais.getNextEntry()) != null) {
            entityList.add(Entity.create(currentEntry.getName(),
currentEntry.getLastModifiedDate(), IOUtils.toByteArray(ais)));
        }

        return entityList;
    }
}
```

BasicArchiveReader.java

```java
public class BasicArchiveDeleter implements ArchiveDeleter {
    @Override
    public List<Entity> delete(ArchiveInputStream ais, ArchiveOutputStream aos, List<String>
entityNames) throws IOException {
        List<Entity> deletedEntities = new LinkedList<>();

        ArchiveEntry currentEntry;
        while ((currentEntry = ais.getNextEntry()) != null) {
            if (entityNames.contains(currentEntry.getName())) {
                deletedEntities.add(Entity.create(currentEntry.getName()));
            } else {
                aos.putArchiveEntry(currentEntry);
                IOUtils.copy(ais, aos);
                aos.closeArchiveEntry();
            }
        }

        return deletedEntities;
    }
}
```

BasicArchiveDeleter.java

```java
public record Entity(String name, Date lastModifiedDate, byte[] bytes) {
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Entity entity = (Entity) o;
        return Objects.equals(name, entity.name) && Arrays.equals(bytes, entity.bytes);
    }

    @Override
    public int hashCode() {
        int result = Objects.hash(name);
        result = 31 * result + Arrays.hashCode(bytes);
        return result;
    }

    @Override
    public String toString() {
        return "archiver_api.output.Entity{" +
                "name='" + name + '\'' +
                ", bytes=" + Arrays.toString(bytes) +
                '}';
    }

    public static Entity create(String name, Date lastModifiedDate, byte[] bytes) {
        return new Entity(name, lastModifiedDate, bytes);
    }

    public static Entity create(String name) {
        return new Entity(name, null, null);
    }

    public void accept(Visitor v) {
        v.visit(this);
    }
}
```

Entity.java

Примітка: клас Entity являє собою DTO’шку котра має в собі такі поля: назва сутності, масив байтів сутності та дату останньої модифікації. Приклади використання цього класу буде описано в наступних лабораторних роботах.

Додатково:

Задля привернення уваги користувачів з різних країн, було вирішено додати функціонал локалізації застосунку. В залежності від регіону де було здійснено запуск, буде вибиратися конкретний ресурс та на основі нього буде формуватися вивід користувачу. Реалізовано це на функціоналі влаштованих бібліотек java, а саме на *ResourceBundle* та *Locale*.

Так як *ResourceBundle* є *immutable*, то для полегшення реалізації зміни мови під час роботи системи, було розроблено клас за шаблоном Singleton через який буде відбуватися вся взаємодія з мовними файлами властивостей.

Клас має всередині константу шляху до місця збереження локалізації та під час створення встановлюється локалізація за замовчуванням системи. Для зміни локалізації присутній метод *changeResource()*, де в якості параметру приходить нова локалізація, після чого відбувається перезапис змінної *currentBundle*. Екземпляр цього класу існує лише в одному екземплярі задля унеможливлення виникнення ситуації коли кожен компонент програми використовує власну локалізацію.

```
public class ResourceManager {
    private final String root = "location/lang";
    private ResourceBundle currentBundle = ResourceBundle.getBundle(root);
    public static final ResourceManager INSTANCE = new ResourceManager();
    private ResourceManager() {}

    public Locale getLocale() {
        return currentBundle.getLocale();
    }
    public void changeResource(Locale locale) {
        currentBundle = ResourceBundle.getBundle(root, locale);
    }

    public String getString(String key) {
        return currentBundle.getString(key);
    }
}
```

ResourceManager.java

Висновок:

Використання шаблону стратегії дало змогу розбити реалізації конкретних операцій на «модулі» зі спільним інтерфейсом, що дає змогу міняти робити заміну цих «модулів» без зміни кінцевого результату.

Додатково було реалізовано шаблон одинак. За допомоги нього під вдалося створити єдиний менеджер вибору мову, зміни якого призводять до зміни мову в усьому застосунку, а не тільки в окремих частинах

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