**Creational Patterns**

**Author:** Gîncu Olivia

**Objectives:**

- Get familiar with the Creational Design Patterns (DPs).

- Choose a specific domain for the project.

- Implement at least three Creational Design Patterns within the chosen domain.

**Domain:**

For this project, the chosen domain is a library system. The system manages books, authors, and genres, and allows users to add and list books in a library.

**Used Design Patterns:**

- **Singleton:** Ensures only one instance of the `Library` class.

- **Factory Method:** Creates `Genre` objects dynamically using a factory class.

- **Builder:** Constructs `Book` objects with customizable properties such as title, author, genre, and ID.

**Implementation:**

This project is implemented in Python and follows object-oriented principles to demonstrate three creational design patterns in a library management system.

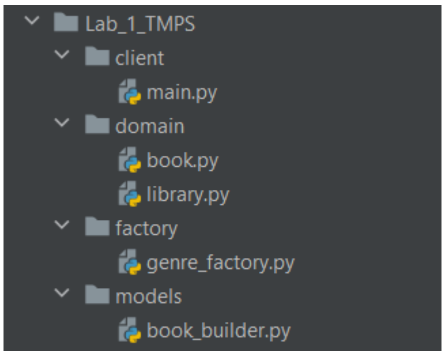
1. **Singleton Pattern:** The `Library` class is implemented as a singleton to ensure only one library instance exists. This pattern is ideal here, as it restricts the library to a single instance for consistency across the application.
   **domain/library.py**
```
class Library:
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super(Library, cls).__new__(cls)
            cls._instance.books = []
        return cls._instance

    def add_book(self, book):
        self.books.append(book)

    def list_books(self):
        return self.books
```
2. **Factory Method Pattern:** The `GenreFactory` class uses the Factory Method pattern to create `Genre` objects based on a genre name. This design pattern simplifies object creation and allows for easy extension if more genres are needed.
   **factory/genre_factory.py**
```
from Lab_1_TMPS.domain.book import Genre

class GenreFactory:
    @staticmethod
    def create_genre(genre_name):
        return Genre(genre_name)
```

3. **Builder Pattern:** The `BookBuilder` class employs the Builder pattern to construct `Book` objects. This pattern is particularly useful here as it allows flexible construction of `Book` instances by setting properties step-by-step, such as title, author, genre, and ID.
   **models/book_builder.py**

```
from Lab_1_TMPS.domain.book import Book, Author

class BookBuilder:
    def __init__(self):
        self.title = None
        self.author = None
        self.genre = None
        self.id = None

    def set_title(self, title):
        self.title = title
        return self

    def set_author(self, name, bio=""):
        self.author = Author(name, bio)
        return self

    def set_genre(self, genre):
        self.genre = genre
        return self

    def set_id(self, id):
        self.id = id
        return self

    def build(self):
        return Book(self.title, self.author, self.genre,
self.id)
```

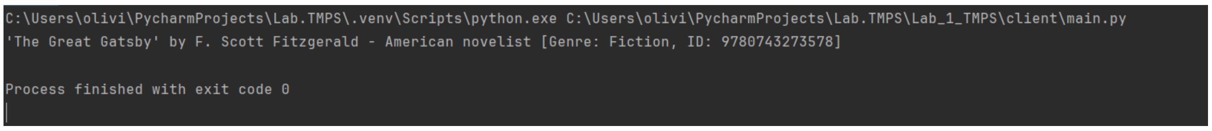
**Screenshots / Results:**

***Project structure:***



```
Lab_1_TMPS
    client
        main.py
    domain
        book.py
        library.py
    factory
        genre_factory.py
    models
        book_builder.py
```

**Output:**



```
C:\Users\olivi\PycharmProjects\Lab.TMPS\.venv\Scripts\python.exe C:\Users\olivi\PycharmProjects\Lab.TMPS\Lab_1_TMPS\client\main.py
'The Great Gatsby' by F. Scott Fitzgerald - American novelist [Genre: Fiction, ID: 9780743273578]

Process finished with exit code 0
```

This line represents a `Book` object created using the `BookBuilder`. The details include:

- **Title**: `'The Great Gatsby'`
- **Author**: `F. Scott Fitzgerald` with a bio, `American novelist`
- **Genre**: `Fiction` (created using the `GenreFactory`)
- **ID**: `9780743273578`

This output confirms that the `BookBuilder` successfully set all the specified attributes and created the `Book` object with the correct values.

**Conclusions:**

This project provided hands-on experience with three essential creational design patterns: Singleton, Factory Method, and Builder. Each pattern was applied to solve specific object creation challenges within the library domain. The Singleton pattern ensured a single instance of the `Library` class, the Factory Method enabled flexible creation of genres, and the Builder pattern allowed for customizable book creation. These design patterns helped to improve the modularity, scalability, and readability of the code, making the project easier to maintain and extend.