Ministère de l'Enseignement Supérieur et
de la Recherche Scientifique
Institut International de Technologie à Sfax

IIT
INSTITUT INTERNATIONAL TECHNOLOGIE
Université Nord Américaine privée

وزارة التعليم العالي والبحث العلمي
المدرسة العليا الدولية الخاصة للتكنولوجيا بصفاقس

Validation du Projet de la Matière Modélisation avancée des Systèmes D'Information

Rapport de Conception du Projet

Effectué à

Institut International de Technologie
(IIT)

Préparé par

Heni Kenoun
Basma Gouiaa
Achraf Khemiri

Année Universitaire 2024 / 2025

I. Introduction

Ce projet est une application de visualisation de graphes et de formes, implémentant divers algorithmes de recherche de chemin. Le projet suit une architecture en couches (MVC) et applique plusieurs Design Patterns pour assurer la maintenabilité et l'extensibilité du code.

II. Structure des Packages

1. Controller

Contient les contrôleurs qui gèrent les interactions utilisateur et appellent les services appropriés.

2. Service

Contient la logique métier, y compris les algorithmes de recherche de chemin et la gestion des formes.

3. Model

Définit les entités de base du projet (formes, nœuds, arêtes, types).

4. Factory

Implémente la création dynamique des formes selon leur type (Factory Pattern).

5. Repository

Gestion de la persistance des formes, que ce soit via console, fichiers ou base de données.

6. Presenter

Formate les résultats pour l'affichage, séparant logique métier et présentation.

7. Logger

Gère la journalisation et les logs du projet.

8. config

Contient les configurations globales comme la connexion à la base de données.

9. util

Classes utilitaires générales.

10. domaine

Gère les aspects liés au domaine métier, notamment la connexion à la base de données.

III. Application des Principes SOLID

1. Single Responsibility Principle (SRP)

Chaque classe a une seule responsabilité :

- **AlgorithmService :** Gère uniquement la logique des algorithmes (Dijkstra, A*, BFS)
- **ShapeService :** Responsable uniquement de la gestion des formes (Shape)
- **ShapePresenter.java :** Séparé pour uniquement présenter les formes sans logique métier

2. Open/Closed Principle (OCP) :

Le code doit être ouvert à l'extension mais fermé à la modification

ShapeFactory: Ajout facile de nouvelles formes sans modifier l'ancienne logique.

ShapeRepository+ DatabaseShapeRepository, FileShapeRepository,

ConsoleShapeRepository : Extension facile de nouveaux types de stockage sans changer le repository principal

3. Liskov Substitution Principle (LSP)

Une classe dérivée doit pouvoir être utilisée à la place de sa classe parente sans modifier le comportement attendu

Les sous-classes de ShapeEntity (RectangleEntity, CircleEntity, SquareEntity) peuvent être utilisées partout où ShapeEntity est attendu.

exp:ShapeEntity shape = new CircleEntity();

shape.draw(); // fonctionnera sans problème, même si c'est un cercle

4. Interface Segregation Principle (ISP)

Une interface ne doit pas forcer une classe à implémenter des méthodes dont elle n'a pas besoin

ShapeRepository Interface très fine, seulement avec les méthodes nécessaires (saveShape)

5. Dependency Inversion Principle (DIP)

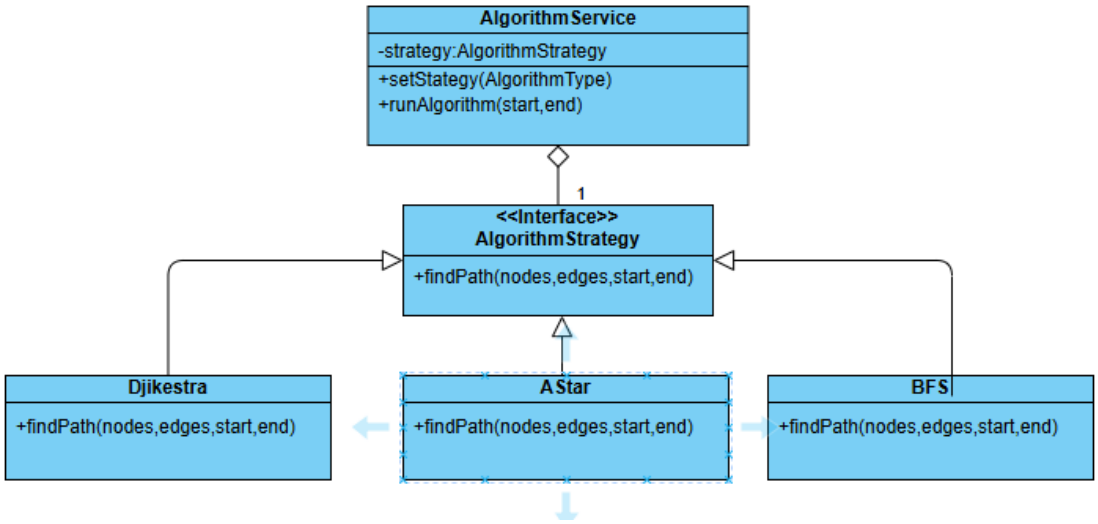
Les classes haut niveau ne doivent pas dépendre des classes bas niveau. Toutes deux doivent dépendre d'abstractions.

Dans notre cas , Les classes ne dépendent jamais directement de FileShapeRepository ou DatabaseShapeRepository, mais plutôt de l'abstraction ShapeRepository.

IV. Design Patterns Utilisés

1. Strategy

Choix dynamique d'un algorithme de recherche dans AlgorithmService.

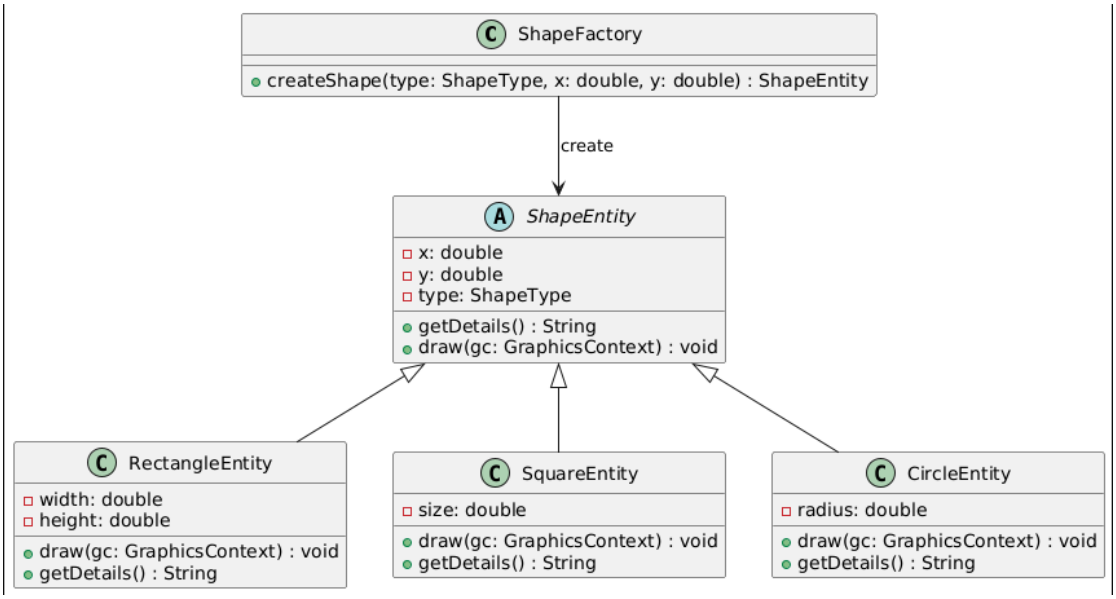


2. Factory Method

Création dynamique des formes dans ShapeFactory.

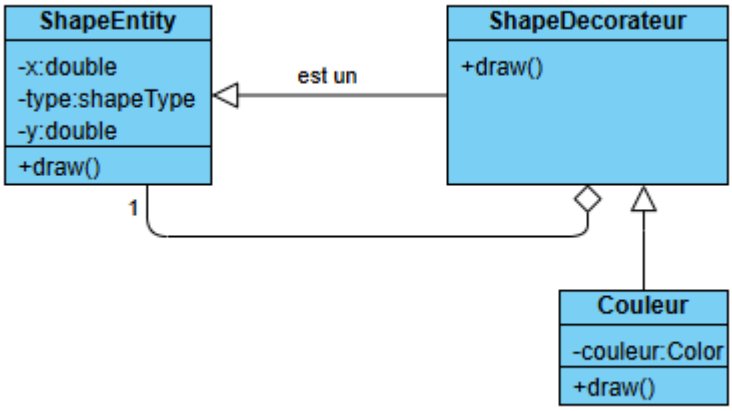
La figure ci-dessous montre Le pattern Factory qui est implémenté via la classe ShapeFactory, qui crée des objets ShapeEntity selon leur type.

Chaque forme (RectangleEntity, SquareEntity, CircleEntity) hérite de ShapeEntity et implémente ses propres comportements. Cela centralise la création d'objets et facilite l'extension du système.



3. Decorator

Ajout de fonctionnalités aux formes via ColoredShapeDecorator.



4. Singleton

Connexion unique à la base de données via DatabaseConnection.

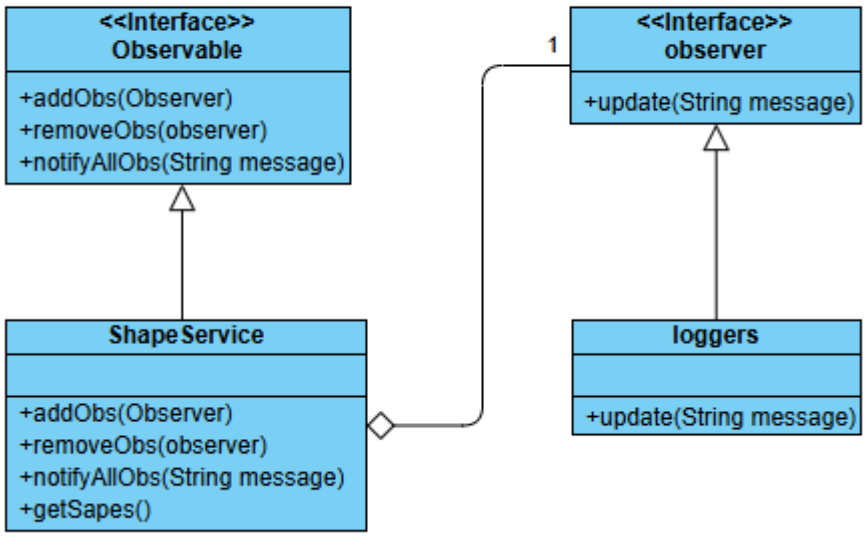
```
public static DatabaseConnection getInstance() {
        if (instance == null) {
            instance = new DatabaseConnection();
        }
        return instance;}
```

5. Observer

Observable est un objet qui gère une liste d'Observers et les notifie en cas de changement d'état.

Observer est un objet qui s'abonne à un Observable pour recevoir automatiquement les mises à jour.

Ce mécanisme permet une communication automatique entre objets sans les coupler fortement.



V. Résumé

Le projet est structuré autour de principes de conception solides, appliquant plusieurs design patterns pour assurer la modularité, la lisibilité et la facilité d'extension. Cette organisation permettra de facilement ajouter de nouvelles fonctionnalités ou modifier des comportements existants sans impact majeur sur l'ensemble du code.