Sup Galilée

Troisième année
Motifs et conceptions
Rapport

Projet de modélisation - Patron MVC

Élèves :
Sidane Alp
Hamza Salah
Wissal Bendidi

Enseignant :
G. Santini

23 janvier 2025

Table des matières

1 Introduction

Nous avons développé, dans le cadre de notre projet de programmation orientée objet, une application de gestion de zoo en Java. Pour mener à bien ce projet, nous nous sommes répartis les tâches au sein de notre équipe de trois étudiants :

Pour ma part, Sidane ALP, je me suis occupé de l'architecture MVC et du développement du contrôleur. Wissal BENDIDI s'est chargée de la modélisation et de l'implémentation des classes d'animaux et d'enclos. Enfin, Hamza SALAH a pris en charge la gestion des employés et le système financier.

2 Présentation du projet

2.1 Objectifs et contraintes

Pour ce projet, nous avons dû répondre à plusieurs contraintes tout en développant une application complète. Nous avons choisi de créer une interface en ligne de commande permettant de gérer tous les aspects d'un parc zoologique. Dans notre développement, nous avons veillé à :

- N'utiliser que les librairies standards de Java
- Créer une interface utilisateur en mode console
- Appliquer rigoureusement les principes SOLID et le pattern MVC
- Documenter entièrement notre code avec Javadoc

2.2 Choix du sujet et originalité

Nous avons choisi de développer un gestionnaire de zoo car ce sujet nous permettait d'explorer une modélisation riche et variée. Nous voulions travailler sur un projet qui combine différents types d'acteurs et de nombreuses interactions. Dans notre application, nous avons mis l'accent sur :

- Une hiérarchie d'animaux que nous avons voulue complexe avec des comportements propres à chaque espèce
- Un système de gestion des employés que nous avons enrichi avec différentes spécialisations
- Une gestion financière que nous avons détaillée avec un suivi précis des transactions
- Un système d'enclos pour lequel nous avons implémenté la gestion de la propreté et de la capacité

2.3 Fonctionnalités principales

Nous avons développé trois profils utilisateurs différents (Directeur, Soigneur, Agent d'entretien). Pour chacun, nous avons implémenté des fonctionnalités spécifiques :

- Pour les animaux : nous avons mis en place un système complet de gestion incluant l'ajout, le nourrissage et les soins
- Pour les enclos : nous avons développé des fonctionnalités de création, nettoyage et assignation
- Pour le personnel : nous avons implémenté un système de recrutement et d'attribution des tâches
- Pour les finances : nous avons créé un suivi détaillé avec gestion des revenus, dépenses et génération de rapports

3 Architecture globale

3.1 Pattern MVC

Nous avons structuré notre application selon le patron de conception MVC pour assurer une séparation claire des responsabilités. L'architecture se compose de :

- **Modèle** : Les classes du package model représentant les entités métier (animaux, employés, enclos)

— **Vue** : La classe ConsoleView gérant l'interface utilisateur en ligne de commande
— **Contrôleur** : Le ZooController orchestrant les interactions entre la vue et le modèle

3.2 Organisation des packages

L'architecture du projet est organisée en packages distincts, reflétant la séparation des préoccupations :

```
fr.zoomanager/
              controller/
                    ZooController.java
              model/
                    animal/
                    employee/
                    enclos/
                    financial/
                    user/
              view/
                    ConsoleView.java
              Main.java
```

Cette organisation permet une maintenance facilitée et une évolution indépendante de chaque composant.

3.3 Interactions entre les composants

Le flux des interactions dans notre application suit le schéma classique du MVC :
— La vue capture les entrées utilisateur et les transmet au contrôleur
— Le contrôleur interprète ces actions et manipule le modèle
— Le modèle notifie ses changements
— La vue est mise à jour pour refléter l'état actuel du modèle

Ce découpage nous a permis de développer chaque partie indépendamment tout en maintenant une forte cohésion dans l'application. La communication entre les composants se fait uniquement via des interfaces bien définies, ce qui facilite les modifications et les tests.

Le diagramme de classe mis dans le dossier rap/ illustre la structure générale de notre application et les relations entre les différentes classes. Cette visualisation met en évidence la séparation claire entre les responsabilités de chaque composant.

4 Modélisation du domaine

4.1 Hiérarchie des animaux

La modélisation des animaux repose sur une hiérarchie de classes permettant une extension facile pour de nouvelles espèces :

```
Animal (abstract)
              Mammifere (abstract)
                    Lion
                    Zebre
              Oiseau (abstract)
```

```
                    Pelican
                    FlamantRose
```

Chaque animal possède des caractéristiques communes (nom, santé, faim) et des comportements spécifiques. Par exemple, les lions peuvent chasser tandis que les pélicans peuvent pêcher. Cette hiérarchie nous permet d'appliquer efficacement le principe de substitution de Liskov.

4.2 Gestion des employés

La gestion du personnel s'articule autour d'une hiérarchie de classes employés :

```
Employee (abstract)
                Soigneur
                AgentMenage
```

Les spécificités de chaque type d'employé ont été modélisées :
— Les soigneurs possèdent des spécialités par espèces d'animaux
— Les agents de ménage sont assignés à des enclos spécifiques

4.3 Système d'enclos

La classe Enclos gère les aspects suivants :
— Une capacité maximale d'animaux
— Un niveau de propreté évoluant selon le nombre d'animaux
— Une liste d'animaux présents

```
public class Enclos {
   private String name;
   private int capacity;
   private List<Animal> animals;
   private double cleanliness;
   // ...
}
```

4.4 Gestion financière

Le système financier s'articule autour de trois classes principales :

```
Budget
                Transaction
                TransactionType (enum)
```

Les transactions sont enregistrées pour chaque action ayant un impact financier :
— Recrutement d'employés
— Construction d'enclos
— Achat de nourriture
— Soins médicaux

Cette modélisation nous a permis de créer une structure robuste et évolutive, capable d'accueillir facilement de nouvelles fonctionnalités tout en maintenant une cohérence globale du système.

5 Application des principes SOLID

5.1 Single Responsibility Principle

Dans notre implémentation, nous avons veillé à ce que chaque classe ait une responsabilité unique :

```
// La classe Budget ne g re que les aspects financiers
public class Budget {
    private double balance;
    private List<Transaction> transactions;
    // ...
}

// La classe Enclos ne g re que son  tat  et ses animaux
public class Enclos {
    private List<Animal> animals;
    private double cleanliness;
    // ...
}
```

5.2 Open/Closed Principle

L'architecture permet l'extension sans modification du code existant :
- La hiérarchie des animaux peut accueillir de nouvelles espèces
- Le système d'employés peut intégrer de nouveaux rôles
- Les types de transactions sont extensibles via l'énumération

5.3 Liskov Substitution Principle

Les classes dérivées peuvent se substituer à leurs classes de base :

```
// Exemple avec la hi rarchie des animaux
public abstract class Animal {
    public abstract String makeSound();
    // ...
}

public class Lion extends Mammifere {
    @Override
    public String makeSound() {
        return "ROAAAR!";
    }
}
```

5.4 Interface Segregation Principle

Les interfaces sont spécifiques aux besoins des clients :

```
// Les soigneurs ont leurs propres m thodes sp cialis es
public class Soigneur extends Employee {
    public void feedAnimal(Animal animal) {
        // ...
    }
```

```
6
7       public void healAnimal(Animal animal) {
8           // ...
9       }
10  }
```

5.5 Dependency Inversion Principle

Les modules de haut niveau ne dépendent pas des modules de bas niveau :

```
1  // Le contr leur  d pend  des abstractions
2  public class ZooController {
3      private Zoo zoo;
4      private ConsoleView view;
5
6      public ZooController(Zoo zoo, ConsoleView view) {
7          this.zoo = zoo;
8          this.view = view;
9      }
10 }
```

6 Patterns de conception utilisés

6.1 Pattern Observer

Utilisé pour la mise à jour des états :

```
1  // La mise    jour de l' tat  du zoo propage les changements
2  public void updateState() {
3      for (Enclos enclos : this.enclos) {
4          enclos.updateState();
5      }
6  }
```

6.2 Pattern Strategy

Appliqué pour les comportements des animaux :

```
1  // Diff rentes  strat gies  de  d placement
2  public abstract class Animal {
3      protected MovementBehavior movementBehavior;
4
5      public void move() {
6          movementBehavior.execute();
7      }
8  }
```

6.3 Autres patterns pertinents

Nous avons également utilisé :
- Le pattern Factory pour la création des animaux
- Le pattern Singleton pour la gestion du budget
- Le pattern Command pour les actions utilisateur

7 Interface utilisateur

7.1 Architecture de la vue console

La vue console a été implémentée pour offrir une expérience utilisateur claire et intuitive :

```
public class ConsoleView {
    public void displayMainMenu(Role userRole) {
        System.out.println("=== Menu Principal ===");
        switch (userRole) {
            case DIRECTEUR:
                // Options du directeur
                break;
            case SOIGNEUR:
                // Options du soigneur
                break;
            // ...
        }
    }
}
```

7.2 Gestion des interactions utilisateur

L'interface propose des menus adaptés selon le rôle de l'utilisateur :
— Le directeur accède à toutes les fonctionnalités
— Le soigneur peut gérer les animaux dont il est responsable
— L'agent d'entretien gère la propreté des enclos

7.3 Robustesse et gestion des erreurs

Nous avons mis en place une gestion d'erreurs complète :

```
try {
    int choice = Integer.parseInt(view.getUserInput());
    // Traitement du choix
} catch (NumberFormatException e) {
    view.displayError("Entr e invalide");
}
```

8 Difficultés rencontrées et solutions

8.1 Challenges techniques

Durant le développement, nous avons fait face à plusieurs défis :
— La gestion des dépendances entre les différents composants
— La synchronisation des états (animaux, enclos)
— L'implémentation d'une interface utilisateur robuste

8.2 Solutions apportées

Pour résoudre ces difficultés, nous avons :
— Appliqué rigoureusement le pattern MVC
— Mis en place un système de mise à jour périodique
— Développé une gestion d'erreurs exhaustive

8.3 Perspectives d'amélioration

Plusieurs pistes d'amélioration ont été identifiées :
— Ajout d'une interface graphique
— Persistance des données
— Système de statistiques avancées

9 Conclusion

9.1 Bilan technique

Ce projet nous a permis de mettre en pratique les concepts théoriques vus en cours :
— Application concrète des principes SOLID
— Utilisation pertinente des design patterns
— Architecture logicielle structurée

9.2 Apprentissages

Nous avons particulièrement progressé sur :
— La conception orientée objet
— La gestion de projet en équipe
— L'implémentation de patterns de conception

9.3 Perspectives d'évolution

Pour une version future, nous envisageons :
— L'ajout de nouvelles espèces d'animaux
— L'implémentation d'un système de sauvegarde
— Le développement d'une API REST

10 Annexes

10.1 A. Diagrammes UML détaillés

Voir le fichier annexe pour le diagramme de classe.

10.2 B. Captures d'écran de l'application

10.2.1 B.1 Menu principal

```
=== Zoo Manager ===
1. G rer les enclos
2. G rer les animaux
3. G rer les employ s
4. Voir le rapport financier
5. Quitter
Votre choix :
```

10.2.2 B.2 Gestion des animaux

```
=== Gestion des animaux ===
1. Voir tous les animaux
2. Ajouter un animal
3. Nourrir un animal
4. Soigner un animal
5. Retour
Votre choix :
```

10.3 C. Guide d'utilisation

10.3.1 C.1 Installation

Pour installer et exécuter l'application :

```
# Compilation
javac -d bin src/fr/zoomanager/**/*.java

# Ex cution
java -cp bin fr.zoomanager.Main
```

10.3.2 C.2 Utilisation des rôles

- **Directeur** : Accès complet à toutes les fonctionnalités
- **Soigneur** : Gestion des animaux selon les spécialités
- **Agent d'entretien** : Nettoyage et maintenance des enclos

10.3.3 C.3 Commandes principales

Les principales commandes disponibles selon le rôle :
- Connexion : Choix du rôle au démarrage
- Navigation : Utilisation des menus numérotés
- Validation : Touche Entrée après chaque choix
- Déconnexion : Option "Quitter" dans le menu principal