**Projet Zoo**

Réaliser par

Algassimou Barry

Client

Vincencini Jacques

Sommaire

1. **Introduction**
2. **Méthode 2TUP**
3. **Apport scientifique**
4. **Conclusion**
5. **Introduction**

Dans le cadre de mon apprentissage en conception orientée objet, j’ai réalisé ce projet du début à la fin en mode agile.

J’ai modélisé et j’ai implémenté le zoo en appliquant la méthode agile(2TUP). Le but est de mettre en pratique mes connaissances acquises en conception orientée objet, créer un projet Maven, passer les tests unitaires, générer la documentation du projet et enfin ajouter le projet sur GitLab.

1. Méthode 2TUP

Cette méthode permettra de faire la conception du projet et de revenir sur la modélisation si besoin.

**Etude du cahier des charges**

1. **Acteurs :** l’idée est de définir tous les acteurs qui interagissent avec le système (Zoo). Les acteurs doivent être externes au système.
   1. **Utilisateur :** Il peut lancer le système et exécuter certaines actions, comme ajouter, supprimer, afficher la liste d’animaux ou de personnes…
   2. **Administrateur :** il gère l’application, à savoir, les mises à jour, l’ajout des nouvelles fonctionnalités comme les règles de gestion d’ajout, de suppression, d’action des animaux.
2. **Identifications des messages :** cette partie permet d’identifier tous les messages entre le système et les acteurs.

**De l’utilisateur (gérant) vers le système :** Il peut demander le système d’ajouter ou de

Supprimer des animaux ou des personnes dans les listes et afficher leurs caractéristiques.

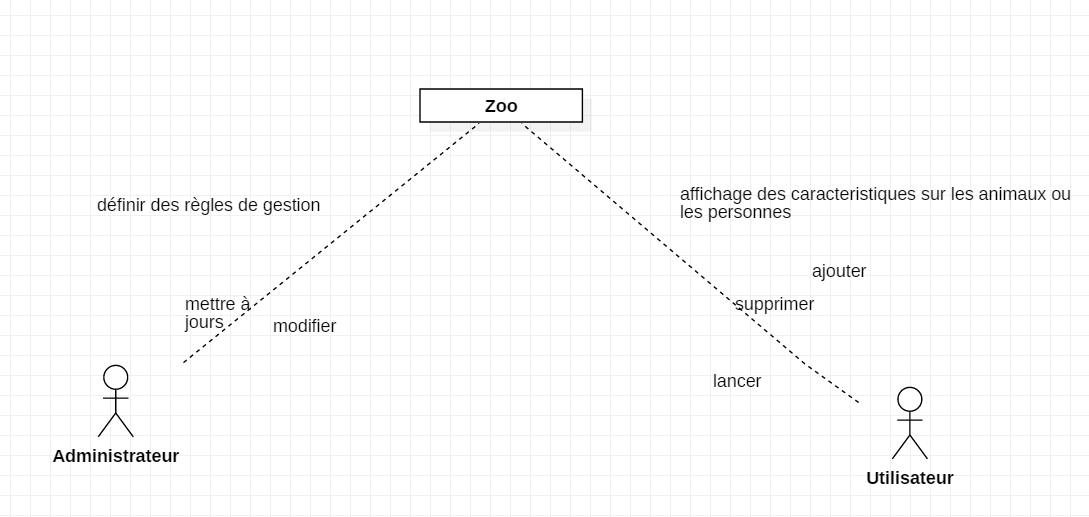
Il peut demander aussi le système de faire exécuter des actions aux animaux, comme manger, courir, crier, sauter et dormir.

**Du système vers l’utilisateur :** Le système confirme l’exécution des tâches par des affichages sur la console en affichant des messages ou des exceptions.

**De l’administrateur vers le système :** Il peut demander le système de d’effectuer des mises à jour ou des modifications : créer d’autres animaux, définir des règles de gestion d’ajout, de suppression, de sécurité …

**Du système vers l’administrateur :** Réponse du système à l’administrateur par des affichages sur la console. Succès ou échec de l’ajout, succès ou échec de la suppression. Idem pour les autres.

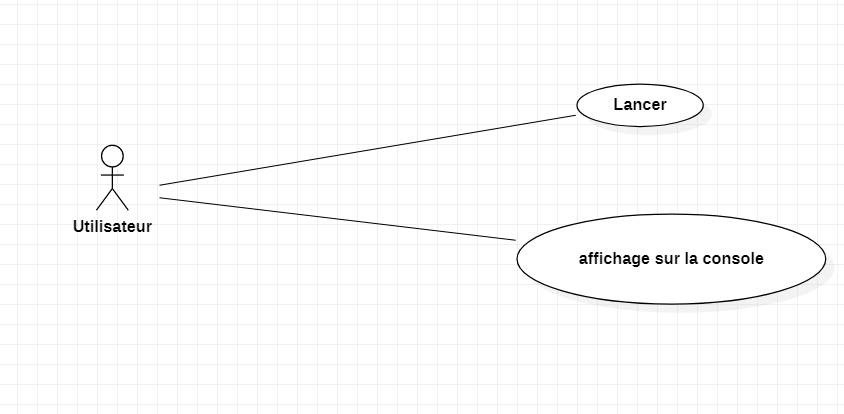
1. **Diagramme de contexte**



1. **Capture des besoins fonctionnels :** elle permettra d’avoir une meilleure approche des utilisateurs, des fonctionnalités et de la relation entre les deux.
   1. **Liste des cas d’utilisation**

|  |  |  |
| --- | --- | --- |
| Nom Use Case | Action | Message |
| Lancer | Utilisateur et système | Utilisateur🡪Système = afficher, Affichage des messages sur la console,  Système 🡪 Utilisateur = message, |
| Ouvrir | Utilisateur et système | Utilisateur->système = ouverture de la cage,  Système -> Utilisateur = ouvert, si la cage est fermée elle est ouverte sinon, affiche elle est déjà ouverte |
| Entrer | Utilisateur et système | Utilisateur🡪Système = entrer,  Système 🡪 Utilisateur réponse, sinon message d’échec |
| Sortir | Utilisateur et système | Utilisateur->Système = sortir  Système-> Utilisateur = réponse, sinon message d’échec |
| Affichage | Système et utilisateur | Utilisateur🡪Système = affiche les caractéristiques  Des animaux,  Système 🡪 Utilisateur = affichage des informations sur la console |

1. **Digramme de use case :**



1. **Scénario**

**UC lancer**

**Précondition :** l’animal et la cage doivent exister

**Cas minimal**

1. Affiche l’exécution du programme
2. L’utilisateur ferme l’application
3. Fin

**Variantes**

**Post-condition**

**UC ouvrir**

**Précondition :** la cage doit exister avant

**Cas minimal**

1. L’utilisateur demande l’ouverture de la cage
2. Confirmation de l’ouverture
3. Fin

**Variantes**

**Post-condition**

**UC entrer**

**Précondition :** la cage doit être ouverte

**Cas minimal**

1. L’utilisateur demande de faire entrer un animal dans la cage
2. Confirmation par un message

2-A) sinon lance une Exception PorteException

2-B)S’il contient un autre animal, le système lance une Exception CagePleineException

1. Fin

**Post-condition**

**UC sortir**

**Précondition**

L’animal doit exister à l’intérieur de la cage

**Cas minimal**

1. L’utilisateur demande de sortir un animal
2. Le système effectue une confirmation par message
3. Confirmation

**Variante**

2-A) Si la cage est vide, affichage d’un message la cage est vide

2-B) Si la cage est fermée, lance une Exception PorteException

**Post-condition**

**UC fermer**

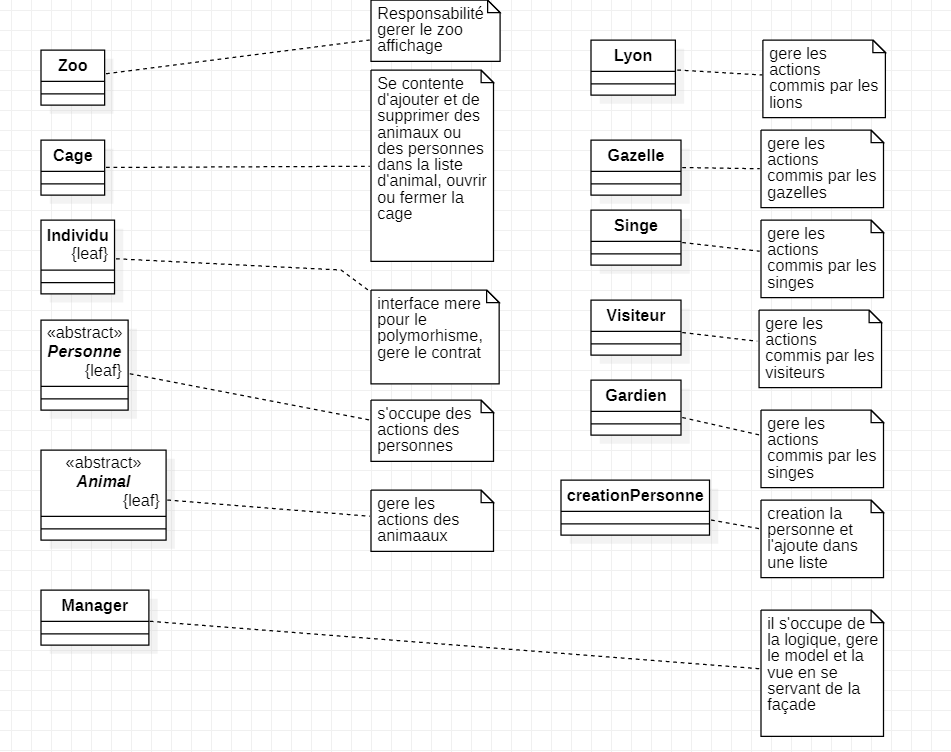
**Précondition :** La Cage doit être ouverte

**Cas minimal**

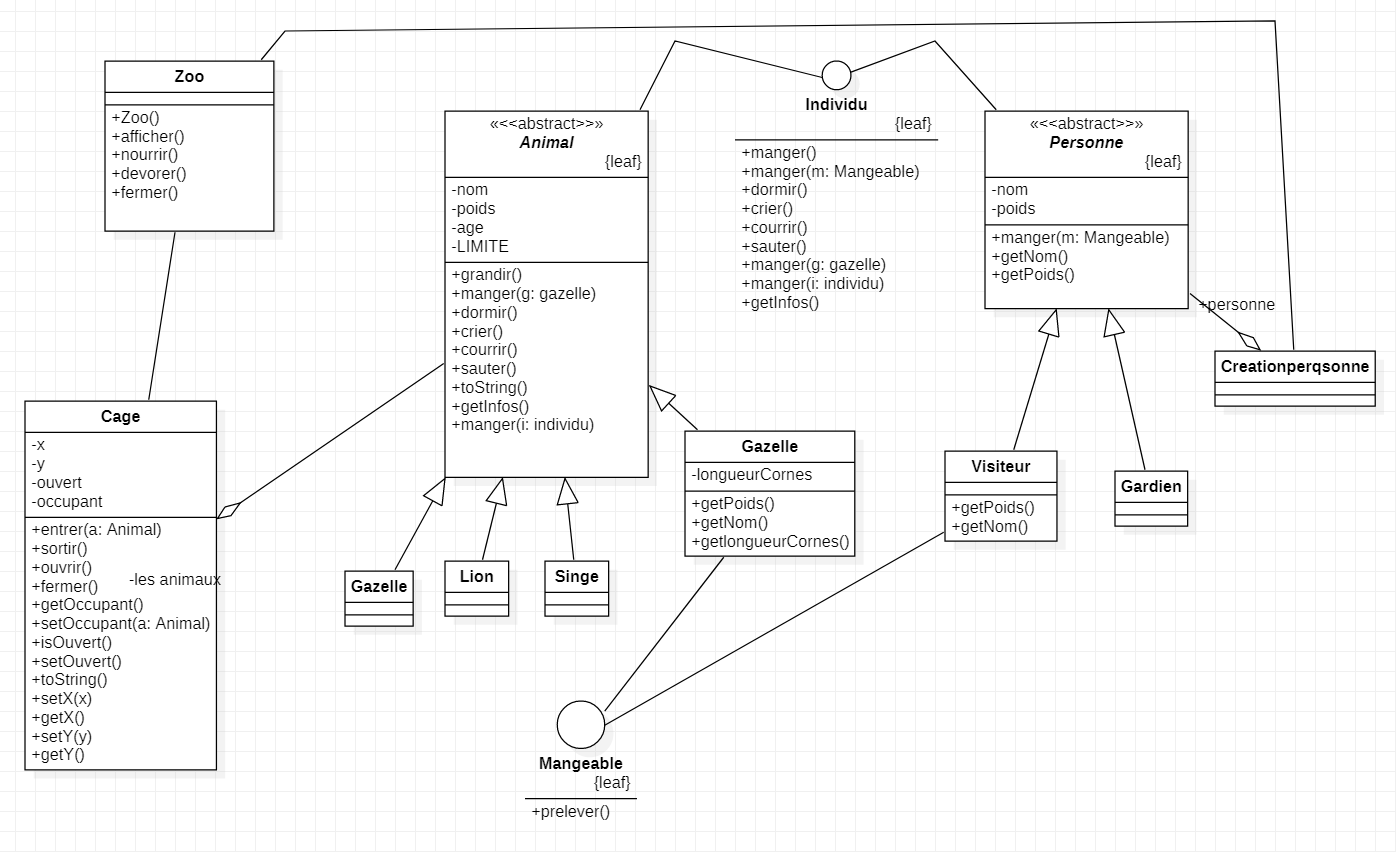
1. L’utilisateur demande de fermer
2. Confirmation du système
3. Fin

**Variantes**

**Post-condition**



1. **Digramme de classes**

**.**

1. **Apport scientifique**

Ce projet m’a permis de mettre en compétence les connaissances théoriques que j’ai

Appris durant ce semestre. J’ai appris à utiliser des logiciels de modélisation de l’UML, à travailler avec un projet Maven ensuite faire des tests unitaires et enfin utiliser un logiciel de versioning qui est git.

1. **Conclusion**

Le codage est une étape cruciale dans la réalisation d’un projet informatique, mais sans une méthode de gestion, il y aura un risque de perte de temps énorme lors du développement. Cela m’a appris de se focaliser d’abord dans la méthode de gestion qui donnera une vision globale du projet, comprendre le contexte, le besoin du client avant de se lancer au codage.