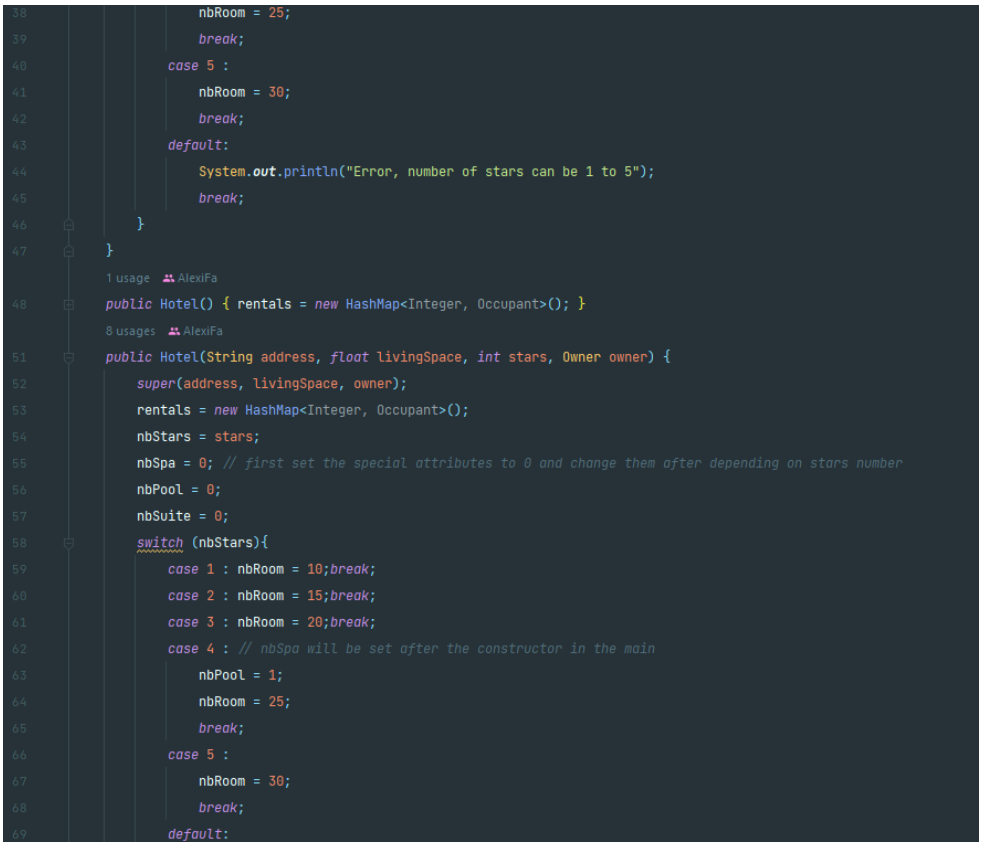
Alexis Faure
Marion Galfard
Rebecca Morel
Vinciane Loy
Romain Léoture

eFrei
PARIS PANTHÉON-ASSAS UNIVERSITÉ

Projet POO en Java

Promotion immobilière

Décembre 2022



```
                nbRoom = 25;
                break;
            case 5 :
                nbRoom = 30;
                break;
            default:
                System.out.println("Error, number of stars can be 1 to 5");
                break;
        }
    }
    1 usage  AlexiFa
    public Hotel() { rentals = new HashMap<Integer, Occupant>(); }
    8 usages  AlexiFa
    public Hotel(String address, float livingSpace, int stars, Owner owner) {
        super(address, livingSpace, owner);
        rentals = new HashMap<Integer, Occupant>();
        nbStars = stars;
        nbSpa = 0; // first set the special attributes to 0 and change them after depending on stars number
        nbPool = 0;
        nbSuite = 0;
        switch (nbStars){
            case 1 : nbRoom = 10;break;
            case 2 : nbRoom = 15;break;
            case 3 : nbRoom = 20;break;
            case 4 : // nbSpa will be set after the constructor in the main
                nbPool = 1;
                nbRoom = 25;
                break;
            case 5 :
                nbRoom = 30;
                break;
            default:
```

Groupe E
2022-2023

Table des matières

1. Introduction

2. Diagramme des classes

3. Présentation fonctionnelle du projet
   - 3.1. Fonctionnalités réalisées parmi celles proposées dans le projet
   - 3.2. Fonctionnalités ajoutées en plus de ce qui a été proposé

4. Présentation technique du projet
   - 4.1. Description des principaux algorithmes réalisés
   - 4.2. Justification des choix des structures de données employées
   - 4.3. Explication des difficultés (techniques)

5. Présentation des résultats
   - 5.1. Tests réalisés et résultats obtenus

6. Conclusion
   - 6.1. Apprentissages sur le plan technique
   - 6.2. Organisation du travail et gestion du temps

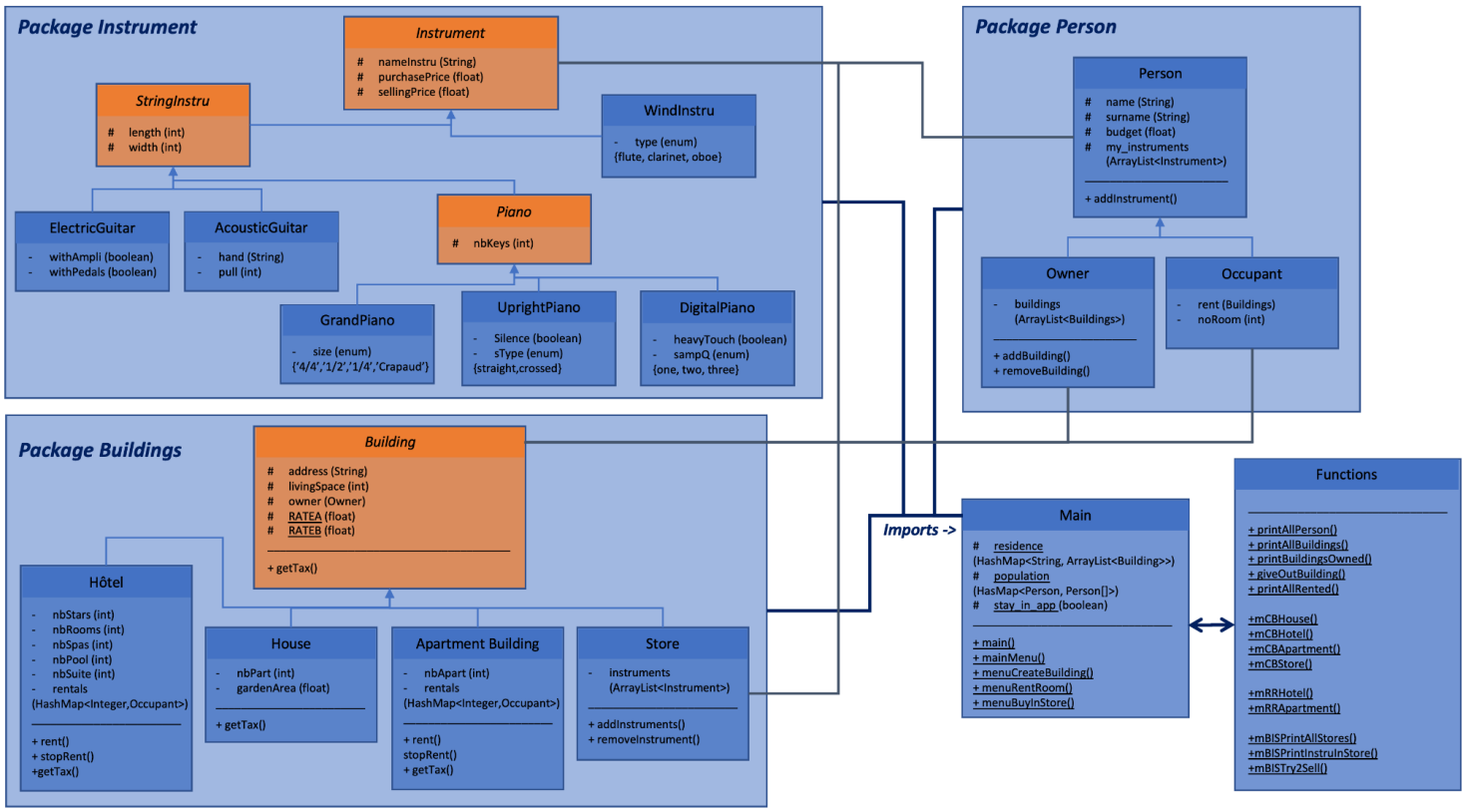
1. Introduction

Pour ce projet, nous devions concevoir l'architecture orientée objet d'une résidence dans une île. La résidence comportera plusieurs types de bâtiments dont des hôtels, des maisons, des immeubles et des magasins.

L'un des buts de ce projet est de mettre en avant nos compétences en codage acquis durant nos cours de Programmation Objet Orienté Java. De plus, il permet d'améliorer ces compétences techniques à partir d'un groupe de projet qui permet de réunir les capacités de chaque personne du groupe afin de mener à bien le projet.

Ce projet a également pour but de faire toutes les étapes d'un projet de A à Z afin de s'habituer pour quand nous serons dans le monde du travail. Il nous a permis de travailler en indépendance à partir de nos cours de Java mais également grâce à nos recherches personnelles.

2. Diagramme des classes



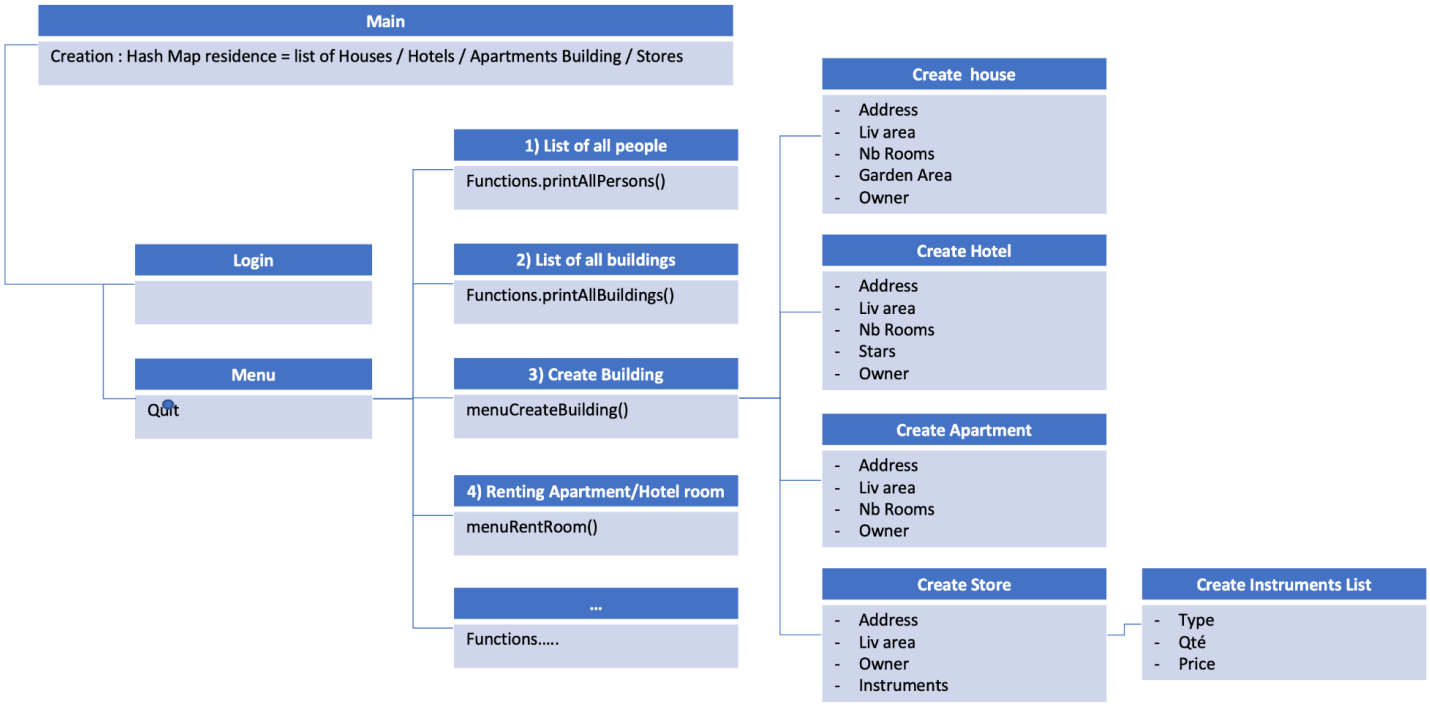
3. Présentation fonctionnelle du projet

3.1. Fonctionnalités réalisées parmi celles proposées dans le projet

Toutes les fonctionnalités demandées dans l'énoncé ont été réalisées, à savoir :

- L'affichage détaillé des descriptions d'infrastructure et d'instruments musicaux. Avec des toString(), les détails s'affiche dans le programme avec un menu permettant d'afficher tous les bâtiments avec, dans le cas d'un magasin, les fiches techniques des instruments.
- Les associations d'un propriétaire ou du locataire à un bâtiment, appartement, chambre. Lors de la création du bâtiment pour le propriétaire, et grâce à une fonction appelée dans un menu pour la location

- Le listage des biens d'un propriétaire, des appartements loués, des chambres et des suites disponibles. Des fonctions permettent cet affichage en allant chercher les informations dans les structures de données présentes dans les classes. Un menu permet aussi d'afficher tous les biens de la personne connectée.
- Le calcul de l'impôt local en fonction des bâtiments. Cela est fait tout simplement avec une fonction qui est héritée et complétée pour les classes filles au fur et à mesure.



Exemple de certaines fonctionnalités du main

3.2. Fonctionnalités ajoutées en plus de ce qui a été proposé

En plus des fonctionnalités proposées nous avons ajouté 4 autres fonctionnalités:

- telles que la possibilité de donner un bâtiment (si le propriétaire souhaite donner un logement une liste de personne enregistrée lui sera donnée).
- Il est possible de faire l'achat d'instruments. Avec un budget qui lui est demandé quand la personne décide d'acheter dans le magasin de son choix (s'il y en a plusieurs). L'instrument choisi est donc retiré des stocks du magasin et ajouté à l'inventaire de la personne.

- L'occupant à la possibilité de stopper la location d'un logement. Ce qui rend la chambre ou l'appartement de nouveau disponible.
- De plus, un système de logging simple à été fait. Lors du lancement du programme, l'utilisateur entre son nom et il est ajouté dans la liste des membres de la résidence. Il effectue des actions qui l'associent avec des bâtiments, des locations ou des instruments. Et lorsqu'il se log out, il y a la possibilité d'entrer un autre nom ce qui ajoutera une personne dans la liste. Si un nom déjà existant est entré, alors il récupère les associations qu'il avait créées plus tôt.

4. Présentation technique du projet

4.1. Description des principaux algorithmes réalisés

a. Menu divisées en différentes fonctions

Nous avons structuré l'application en plusieurs menus dans lesquels navigue l'utilisateur et qui le guide dans les actions qu'il peut faire et les champs qu'il doit remplir pour les effectuer. Ces menus sont affichés avec toutes les actions disponibles, des chiffres associés à chaque action et l'utilisateur entre dans la console les chiffres correspondants aux actions voulues. De ce fait, il y a beaucoup de sous menus qu'on a divisé en sous fonctions pour plus de lisibilité. Ainsi chaque fois qu'il sélectionne une action, un scanner prend ce qu'il a entré et avec des switch, on lance la sous fonction adéquate.

b. La location

La location est l'un des gros algorithmes que nous avons fait. Il consiste en une fonction rent qui associe, dans la classe du bâtiment la chambre à l'occupant. Mais qui modifie aussi la classe de l'occupant car il met dans ses attributs, la chambre qu'il loue et le bâtiment dans laquelle elle se trouve.

c. La création des bâtiments

La création de bâtiment à aussi pris beaucoup de temps, car il fallait faire beaucoup de vérifications. Notamment le fait qu'on ne puisse pas créer deux bâtiments avec la même adresse. Mais aussi le nombre d'informations à initialiser pour la création des bâtiments. Par

exemple, tous les instruments pour les magasins ou encore le nombre de suite, piscine, spa pour les hôtels. Nous avons décidé de faire un nombre de chambres par défaut pour chaque type d'hôtel et les autres informations étaient données par le créateur du bâtiment.

4.2. Justification des choix des structures de données employées

a. Hashmap et ArrayList

On a choisi d'utiliser des Hashmap :

- Pour stocker les bâtiments de la résidence. Avec des index de type String qui indique les types de bâtiment et l'objet qui est un tableau de chaque type de bâtiment. Ainsi on a accès aux tableaux des bâtiments avec leur type.
- Pour stocker la population de la résidence. En effet, l'index de type Person sera lié à un tableau de dimension 2 avec un Owner et un Occupent. Ces objets sont reliés à la même personne mais le type change en fonction de l'action qu'elle souhaite effectuer. (louer/créer un bâtiment)
- Pour stocker les locations dans les bâtiments. L'index est le numéro de la chambre ou de l'appartement et l'objet est de type Building qui correspond au bâtiment dans lequel se trouve la chambre/appartement.

Dans les autres cas, il nous semblait plus simple d'utiliser des ArrayList. Notamment pour les listes d'instruments disponibles dans les magasins ou appartenant à une personne ou encore la liste des bâtiments appartenant à un propriétaire.

b. Enum

On a également choisi d'utiliser des Enum pour certains attributs qui ne prenait que certaines valeurs. C'est le cas pour la taille du Grand Piano ou encore le type des instruments à vent.

c. Les classe rooms/flats

Nous n'avons pas créé de classes Room ou Flat pour les chambres ou les appartement car cela nous semblait plus simple de seulement stocker un nombre de chambre/appart et d'assigner des numéro de 1 au nombre de chambre. La location se fait donc sur les numéros de chambre et dans le hashmap qui stocke toutes les chambres louées.

4.3. Explication des difficultés (techniques)

a. Système de location

Nous avons mis un peu de temps pour choisir un système de location efficace. D'abord, nous avions fait une classe pour stocker le bâtiment, l'occupant et le numéro de la chambre. Mais ensuite, nous nous sommes rendu compte qu'il était plus simple de seulement stocker ces variables dans un hashmap dans le bâtiment ce qui nous permettait d'agir sur ces variables dans une fonction directement dans la classe bâtiment.

b. Les tests

Au départ, nous nous sommes mis à coder les différentes parties de l'algorithme en testant à la main dans la console ou dans l'outil de debug si les fonctions faisaient bien ce que nous voulions mais sans faire de test automatiques. Ce fût une erreur car lorsque nous modifions une fonction, cela pouvait modifier le comportement à certain endroit dans le code et provoquer des erreurs qui nous obligeait à retester les fonctions. Heureusement, nous nous en sommes rendu compte assez tôt ce qui nous a permis de faire tous ces tests automatique qui, en un run, nous permettait de tester toutes les fonctions et de voir lesquels ne fonctionnait pas.

5. Présentation des résultats

Dans la finalité, nous avons une application qui permet de faire toutes les fonctionnalités de l'énoncé avec certaines possibilités en plus. Nous pouvons faire une résidence avec autant de bâtiments que veut l'utilisateur. Les bâtiments ont tous un propriétaire qui peut être modifié (le propriétaire peut "léguer" son bâtiment). Les membres de la résidence peuvent tous louer une chambre ou un appartement parmis les appartements et hôtels disponibles, et tout cela avec des tests pour vérifier qu'une chambre ne peut pas être louée par deux personnes en même temps ou qu'un occupant ne peut pas louer deux location simultanément.

6. Conclusion

6.1. Apprentissages sur le plan technique

Ce projet nous a permis d'apprendre beaucoup sur l'organisation en groupe et la programmation sur de grands projets avec de nombreuses classes qui interagissent entre elles. Nous devions donc beaucoup communiquer entre nous pour que les fonctions ajoutées par quelqu'un fonctionne avec le travail d'un autre.
Cela nous a aussi permis d'apprendre à utiliser github et son gestionnaire de conflit qui nous a donné du fil à retordre.

6.2. Organisation du travail et gestion du temps

Nous avons, dans un premier temps, discuté du sujet afin de déterminer la structure du projet pour avancer de façon efficace. Nous nous sommes ensuite répartis les tâches afin de commencer de manière efficace. Au fil des semaines, nous avons tenu un trello afin d'avoir un suivi des tâches en cours tout en travaillant sur un github pour mettre en commun notre code.

Durant le projet, nous nous sommes beaucoup appuyés sur nos différents parcours académiques. En effet, venant de filières différentes, nous avons eu l'occasion d'être confrontés à différentes méthodes de travail, de gestion des difficultés et de communication. Pour conclure, nos qualités individuelles, notre communication et le travail approfondi de chacun a permis la réalisation d'un projet qui nous rend fiers.