Rapport final - Projet Booking

1. Description générale du projet

Le projet **Booking** a pour objectif de concevoir une application Java orientée objet simulant une plateforme simplifiée de réservation d'hébergements. Il s'agit d'un projet pédagogique permettant de mettre en pratique plusieurs concepts fondamentaux de la programmation orientée objet, tels que l'héritage, le polymorphisme, les interfaces, la gestion de collections, le tri et la manipulation d'objets complexes.

Cette application modélise un système de réservation d'hébergements qui peut être comparé à une version très simplifiée de plateformes réelles comme Booking.com ou Airbnb. Elle prend en compte les utilisateurs, les différents types d'hébergements, le processus de réservation et d'annulation, ainsi que la gestion de réductions pour certains clients.

Les utilisateurs sont classés selon leur rôle et leur ancienneté :

- **Nouveau Client** : client récent sans historique de réservation, n'a pas de réduction.
- **Ancien Client** : client fidèle, pouvant bénéficier de réductions en fonction de son historique de réservations.
- **Administrateur** : utilisateur privilégié pouvant gérer les hébergements, appliquer des réductions, et consulter les informations clients.

Les hébergements sont également diversifiés pour refléter différents types d'offres :

- **Appartement** : logement complet, souvent destiné à plusieurs occupants.
- **Chambre d'hôtel** : offre standard, généralement pour un ou deux occupants.
- **Villa** : logement de standing, pouvant inclure des fonctionnalités supplémentaires comme piscine ou jardin.

Le projet est organisé en plusieurs packages afin de séparer les responsabilités et rendre le code plus lisible et maintenable :

- `hebergements` : classes liées aux hébergements (Appartement, ChambreHotel, Villa) et à l'interface `Reservable`.
- `personnes` : classes représentant les utilisateurs et leurs différents rôles.
- `reservations` : gestion des réservations, avec calcul automatique des prix et application des réductions.
- `collections` : gestion centralisée des hébergements disponibles sur la plateforme.

Chaque package est indépendant et clairement structuré, ce qui facilite l'évolution future du projet (ajout de nouveaux types d'hébergements ou de clients, extension des fonctionnalités).

---

2. Principaux choix de conception

2.1 Héritage et polymorphisme

L'héritage est un choix central dans ce projet, permettant de factoriser le code et de réduire la duplication.

- La classe abstraite `Personne` contient les attributs communs à toutes les personnes : `nom`, `prenom` et `email`. Elle définit également une méthode abstraite `getTypePersonne()` que toutes les classes filles doivent implémenter.
- La classe `Client` hérite de `Personne` et inclut les attributs et comportements spécifiques aux clients, comme les réservations, l'adresse et la date d'inscription.
- Les classes `NouveauClient` et `AncienClient` spécialisent `Client` en fonction du type de client. Par exemple, `AncienClient` peut bénéficier d'une réduction automatique lorsqu'il a effectué un certain nombre de réservations, tandis que `NouveauClient` ne reçoit pas de réduction.
- `Administrateur` hérite de `Personne` et dispose de méthodes spécifiques pour gérer les hébergements, comme l'ajout, la modification et la suppression, ainsi que l'application de réductions aux anciens clients.

Pour les hébergements :

- `Hebergement` regroupe les attributs et comportements communs à tous les types d'hébergements : prix par nuit, capacité, nom, description, adresse, notes et périodes de disponibilité.
- `Appartement`, `ChambreHotel` et `Villa` héritent de `Hebergement` et redéfinissent certaines méthodes si nécessaire, par exemple pour calculer le prix ou gérer les périodes réservées.

Le polymorphisme permet de manipuler les objets de manière générique. Par exemple, une méthode peut accepter un paramètre de type `Hebergement` et fonctionner aussi bien pour un `Appartement`, une `Villa` ou une `ChambreHotel`, sans connaître le type exact au moment de l'écriture du code.

Cette approche améliore la réutilisabilité et la maintenabilité du code, car les modifications apportées à la classe parente se propagent automatiquement aux classes filles.

2.2 Interface `Reservable`

L'interface `Reservable` définit un contrat commun pour tous les hébergements qui peuvent être réservés. Elle inclut les méthodes suivantes :

- `afficherDetails()` : afficher les informations détaillées de l'hébergement.
- `estDisponible(LocalDate, LocalDate)` : vérifier la disponibilité pour une période donnée.
- `calculerPrix(LocalDate, LocalDate, int)` : calculer le prix total pour un séjour en fonction du nombre de nuits et des occupants.
- `reserver(Client, LocalDate, LocalDate)` : effectuer une réservation pour un client.
- `estReservee(LocalDate)` : vérifier si une date spécifique est déjà réservée.
- `annulerReservation(Client, LocalDate)` : annuler une réservation existante.

Ce choix garantit que tous les types d'hébergements respectent le même contrat et que les méthodes principales peuvent être appelées sur n'importe quel objet `Reservable`. Cela permet aussi d'ajouter facilement de nouveaux types d'hébergements sans modifier le reste du code.

2.3 Gestion des collections

La classe `CollectionHebergements` centralise la gestion des hébergements disponibles. Elle utilise une liste dynamique (`List`) pour stocker les hébergements et fournit plusieurs fonctionnalités :

- ajout et suppression d'hébergements,
- recherche selon différents critères : capacité minimale, prix maximum, type ou note minimale,
- tri par prix ou par note moyenne,
- affichage de tous les hébergements présents dans la collection.

Chaque client possède également sa propre collection de réservations, ce qui permet de suivre facilement l'historique des séjours et de calculer d'éventuelles réductions. La structure globale assure la cohérence entre les clients et les hébergements : chaque réservation est à la fois enregistrée dans le client et dans l'hébergement correspondant.

2.4 Tri et comparaison

La classe `Hebergement` implémente `Comparable<Hebergement>` pour permettre le tri naturel des hébergements par prix par nuit. Cette fonctionnalité facilite l'utilisation de méthodes standard de Java comme `Collections.sort()`.

Pour un tri plus avancé, par exemple par note moyenne, un comparateur peut être utilisé, offrant une flexibilité dans l'affichage et la sélection des hébergements pour le client.

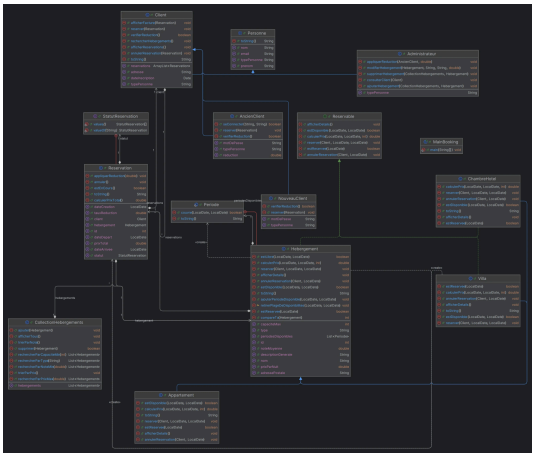
---

3. Diagrammes UML

3.1 Diagramme de classes

Le diagramme de classes permet de visualiser la structure complète du projet, les relations entre les classes, les attributs, méthodes et héritages. Il met en évidence :

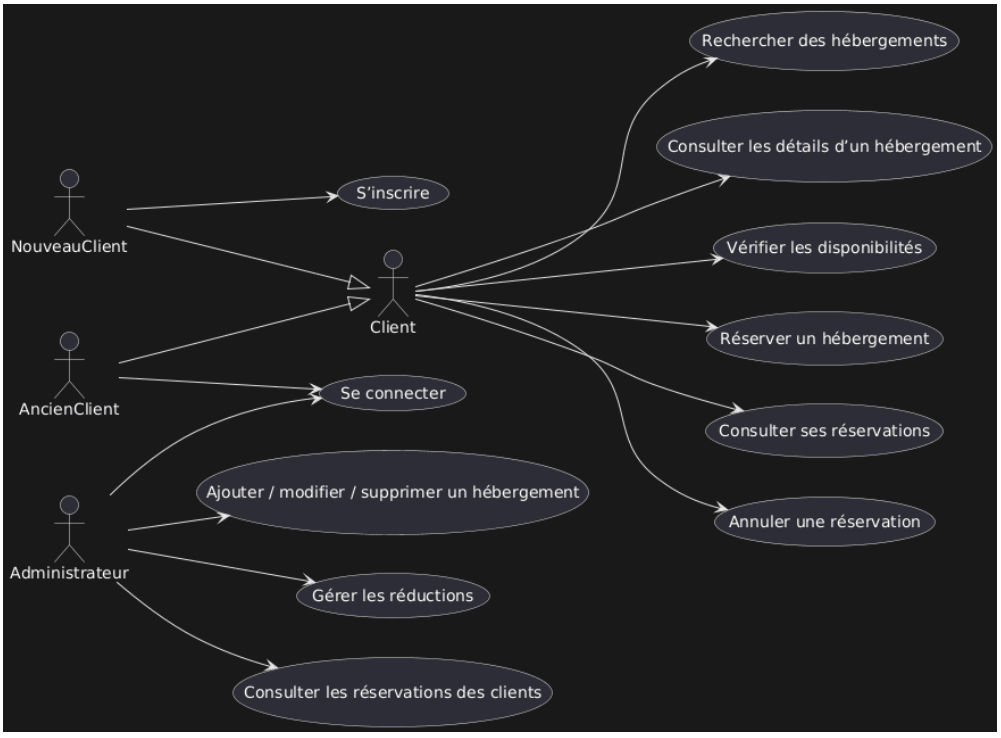
- l'héritage entre `Personne`, `Client`, `NouveauClient`, `AncienClient` et `Administrateur`,
- l'héritage entre `Hebergement` et ses classes filles (`Appartement`, `ChambreHotel`, `Villa`),
- l'utilisation de l'interface `Reservable`,
- les associations entre `Reservation`, `Client` et `Hebergement`.



3.2 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation illustre les interactions principales entre les acteurs et le système :

- Le client peut rechercher des hébergements, réserver ou annuler une réservation et consulter ses factures.
- L'ancien client peut bénéficier d'une réduction automatique.
- L'administrateur peut gérer la collection d'hébergements, appliquer des réductions et consulter les informations des clients.



4. Scénarios de test

Plusieurs scénarios de test ont été réalisés pour vérifier le bon fonctionnement de l'application.

4.1 Scénario 1 : Réservation par un nouveau client

Le nouveau client est créé, consulte les détails d'une chambre d'hôtel, vérifie sa disponibilité puis effectue une réservation. La réservation est ajoutée à la fois dans la liste des réservations du client et celle de l'hébergement, et le prix total est calculé automatiquement.

```
=== SCÉNARIO 1 : NouveauClient ===
Hébergement ajouté : Chambre Standard
Hébergement ajouté : Appartement Cosy
Détails chambre :
ChambreHotel{id=1, nom='Chambre Standard', etoiles=3, prixParNuit=80.0}
Est disponibledu 26/01 au 30/01 ?true
Réservation par un nouveau client : Clara Leroy
Réservation simulée pour Clara Leroy :
- Hébergement : Chambre Standard
- Date d'arrivée : 2026-01-26
- Date de départ : 2026-01-30
Réservation effectuée : Reservation{id=1, statut=EN_ATTENTE, client=Leroy, hebergement
=Chambre Standard, dateArrivee=2026-01-26, dateDepart=2026-01-30, prixTotal=320.0, tau
xReduction=0.0, dateCreation=2026-01-30}
```

4.2 Scénario 2 : Ancien client avec réduction et annulation

Un ancien client réserve un appartement. L'administrateur applique une réduction conformément à la politique de fidélité. Le client peut ensuite annuler la réservation, ce qui met automatiquement la période correspondante à nouveau disponible.

```
=== SCÉNARIO 2 : AncienClient ===
Ancien client sans réduction : Paul Durand
Réservation simulée pour Paul Durand :
- Hébergement : Appartement Cosy
- Date d'arrivée : 2026-01-29
- Date de départ : 2026-02-03
Réservation initiale : Reservation{id=2, statut=EN_ATTENTE, client=Durand, hebergement
=Appartement Cosy, dateArrivee=2026-01-29, dateDepart=2026-02-03, prixTotal=600.0, tau
xReduction=0.0, dateCreation=2026-01-30}
Réduction de 15.0% appliquée pour : Durand
Après réduction : Reservation{id=2, statut=EN_ATTENTE, client=Durand, hebergement=Appa
rtement Cosy, dateArrivee=2026-01-29, dateDepart=2026-02-03, prixTotal=510.0, tauxRedu
ction=15.0, dateCreation=2026-01-30}
Annulation simulée pour Paul Durand :
- Hébergement : Appartement Cosy
- Date d'arrivée : 2026-01-29
- Date de départ : 2026-02-03
Après annulation : Reservation{id=2, statut=ANNULEE, client=Durand, hebergement=Appart
ement Cosy, dateArrivee=2026-01-29, dateDepart=2026-02-03, prixTotal=510.0, tauxReduct
ion=15.0, dateCreation=2026-01-30}
Disponibilités de l'appartement : [[2026-01-28 -> 2026-02-10], [2026-01-29 -> 2026-02-
03]]
```

4.3 Menu interactif

Le projet intègre un menu interactif permettant de centraliser toutes les fonctionnalités de l'application. Depuis ce menu, l'utilisateur peut :

- créer de nouveaux clients (nouveau ou ancien),
- ajouter différents types d'hébergements (Chambre, Appartement, Villa),
- consulter la liste des hébergements disponibles,
- créer ou annuler des réservations,
- appliquer des réductions aux anciens clients,
- exécuter les scénarios prédéfinis.

Cette approche permet de tester l'ensemble des fonctionnalités de manière flexible et interactive, tout en conservant l'historique des réservations et en gérant correctement les disponibilités des hébergements.

---

5. Conclusion

Le projet **Booking** a permis d'illustrer de manière pratique l'ensemble des concepts abordés en cours : héritage, polymorphisme, interfaces, collections et tri. La structuration en packages et la séparation claire des responsabilités ont contribué à rendre le code lisible, maintenable et évolutif.

Des améliorations futures pourraient inclure :

- une interface graphique pour remplacer la console,
- la persistance des données dans une base de données ou un fichier,
- l'ajout de nouvelles fonctionnalités comme des promotions, des avis clients ou des notifications,
- une gestion plus complexe des tarifs selon la saison ou les options supplémentaires.

Malgré ces extensions possibles, les objectifs pédagogiques principaux ont été atteints et le projet constitue une base solide pour toute application de réservation orientée objet.