Projet Java POO – Mini-application *Mini-Booking*

B1 Informatique

1 Objectif général

Réaliser en Java une mini-application de réservation de logements, inspirée de *Booking*, permettant de gérer des hébergements, des utilisateurs (clients et administrateurs) et des réservations, en appliquant les principaux concepts de la POO : classes, héritage, classes abstraites, interfaces, polymorphisme, collections et tri.

L'application pourra être en mode console (texte) et devra être suffisamment structurée pour pouvoir évoluer vers une interface graphique ou web par la suite.

2 Fonctionnalités à implémenter

2.1 Gestion des hébergements

Vous devez modéliser des hébergements de différents types (chambre d'hôtel, appartement, villa, etc.).

2.1.1 Classe de base `Hebergement`

Créez une classe de base `Hebergement` contenant au minimum :

- un identifiant unique ;
- un nom ;
- une adresse postale ;
- un type (hôtel, appartement, villa, ...) ;
- une capacité maximale de personnes ;
- un prix par nuit ;
- une description générale ;
- des informations sur les équipements (WiFi, TV, cuisine, etc.) ;
- une structure pour gérer les périodes disponibles (plages de dates) ;
- une structure pour stocker les notes des clients et la note moyenne.

Implémentez au minimum les méthodes suivantes :

- vérifier si une période (date d'arrivée, date de départ) est libre ;
- calculer le prix total d'un séjour sur une période donnée ;
- ajouter une période disponible ;
- supprimer une période disponible ;
- ajouter une note et mettre à jour la moyenne des notes.

La classe `Hebergement` implémente l'interface `Comparable<Hebergement>` afin de permettre le tri (par exemple par prix ou par note moyenne).

2.1.2 Sous-classes concrètes

Créez plusieurs sous-classes concrètes de `Hebergement` :

- `ChambreHotel` ;
- `Appartement` ;
- `Villa`.

Chaque sous-classe pourra ajouter des attributs spécifiques (par exemple : nombre d'étoiles pour un hôtel, présence de piscine pour une villa, etc.).

2.2 Interface `Reservable` et polymorphisme

Pour factoriser le comportement des objets réservables, définissez une interface `Reservable`. Cette interface contiendra au minimum :

- `boolean estDisponible(Date debut, Date fin);`
- `double calculerPrix(Date debut, Date fin, int nbPersonnes);`
- `void reserver(Client c, Date debut, Date fin);`
- `void annulerReservation(Client c, Date date);`
- `void afficherDetails();`
- `boolean estReservee(Date date);`
- des getters pour les propriétés principales (identifiant, type, capacité, etc.).

Les différentes classes d'hébergements (`ChambreHotel`, `Appartement`, `Villa`) implémentent `Reservable` et redéfinissent ces méthodes avec leur logique propre.

L'application devra montrer l'utilisation du polymorphisme : on doit pouvoir manipuler une `ChambreHotel`, un `Appartement` ou une `Villa` via une même référence de type `Reservable`.

2.3 Gestion des utilisateurs

Vous devez modéliser les personnes qui interagissent avec le système : clients et administrateurs.

2.3.1 Classe abstraite `Personne`

Définissez une classe abstraite `Personne` :

- Attributs (au moins, en `protected`) : `nom`, `prenom`, `email`.
- Getters correspondants.
- Méthode abstraite `String getTypePersonne();`.

La classe est abstraite car elle ne représente pas un type d'utilisateur concret, mais un concept général dont héritent les différents types d'utilisateurs (clients, administrateurs).

2.3.2 Classe `Client` et spécialisations

La classe `Client` hérite de `Personne`.

- Attributs possibles : `adresse`, `dateInscription`, une collection de réservations, etc.
- Méthodes possibles :
  - rechercher des hébergements (par type, destination, fourchette de prix, dates, capacité, etc.) ;
  - réserver un hébergement (en fournissant dates, nombre de personnes, etc.) ;

    * annuler une réservation ;
    * vérifier s'il bénéficie d'une réduction ;
    * afficher la facture d'une réservation (avec ou sans réduction).

Définissez deux sous-classes de `Client` :

- `NouveauClient` : doit s'inscrire (email + mot de passe, etc.), ne bénéficie d'aucune réduction.
- `AncienClient` : peut se connecter pour consulter son historique et bénéficie éventuellement d'une réduction en fonction de son nombre de réservations.

Certaines méthodes de `Client` devront être redéfinies dans `NouveauClient` et `AncienClient` pour gérer la logique d'inscription, de connexion et de réduction.

2.3.3 Classe `Administrateur`

La classe `Administrateur` hérite de `Personne` et propose des méthodes pour :

- ajouter un hébergement ;
- supprimer un hébergement ;
- modifier les informations d'un hébergement ;
- gérer les offres de réduction pour les anciens clients ;
- consulter les dossiers des clients et leurs réservations.

2.4 Gestion des réservations

Créez une classe `Reservation` pour relier un client, un hébergement et une période.

2.4.1 Attributs possibles

- identifiant unique ;
- statut (en attente, confirmée, annulée) ;
- référence vers un `Client` ;
- référence vers un `Hebergement` ou un `Reservable` ;
- dates d'arrivée et de départ ;
- prix total ;
- taux de réduction ;
- date de création.

2.4.2 Méthodes au minimum

- calculer le prix total (avec ou sans réduction, selon le nombre de nuits) ;
- appliquer un pourcentage de réduction ;
- vérifier si la réservation est en cours en fonction du statut et des dates ;
- annuler la réservation (mise à jour du statut et des disponibilités de l'hébergement).

2.5 Collections et recherche d'hébergements

Créez une classe `CollectionHebergements` qui encapsule la gestion d'un ensemble dynamique d'hébergements.

- Attribut : `ArrayList<Hebergement> hebergements;`
- Méthodes :
  - `void ajouter(Hebergement h);`
  - méthodes de recherche (par prix maximum, capacité minimale, type d'hébergement, note minimale, etc.) ;
  - éventuellement, méthodes de suppression ou de mise à jour.

Le tri des hébergements passera par l'interface `Comparable` de `Hebergement` (par exemple : tri par prix ou par note moyenne).

3 Diagramme de cas d'utilisation attendu

Le projet doit inclure un diagramme de cas d'utilisation pour l'application *Mini-Booking*, avec au minimum :

Acteurs

- Client (général) ;
- NouveauClient (spécialisation de Client) ;
- AncienClient (spécialisation de Client) ;
- Administrateur.

Cas d'utilisation principaux

- S'inscrire (`NouveauClient`) ;
- Se connecter (`AncienClient`, `Administrateur`) ;
- Rechercher des hébergements ;
- Consulter les détails d'un hébergement ;
- Vérifier les disponibilités ;
- Réserver un hébergement ;
- Consulter ses réservations ;
- Annuler une réservation ;
- Ajouter / modifier / supprimer un hébergement (`Administrateur`) ;
- Gérer les réductions (`Administrateur`) ;
- Consulter les réservations des clients (`Administrateur`).

Le diagramme doit montrer les relations entre acteurs et cas d'utilisation, ainsi que la généralisation `Client` → `NouveauClient` / `AncienClient`.

4 Diagramme de classes attendu

Un diagramme de classes UML doit être fourni, représentant au minimum :

- `Personne` (abstraite) ;
- `Client`, `NouveauClient`, `AncienClient` ;

- `Administrateur` ;
- `Reservable` (interface) ;
- `Hebergement` (implémente `Reservable`, `Comparable<Hebergement>`) ;
- `ChambreHotel`, `Appartement`, `Villa` (héritent de `Hebergement`) ;
- `Reservation` ;
- `CollectionHebergements` ;
- `MainBooking` (classe utilitaire contenant `main`).

Les associations (par exemple `Client` –*– `Reservation`, `Reservation` –1– `Hebergement`, `CollectionHebergements` –1..*– `Hebergement`) doivent être visibles, ainsi que les relations d'héritage et d'implémentation d'interface.

5 Programme de test (`MainBooking`)

Une classe `MainBooking` doit contenir la méthode `public static void main(String[] args)` et simuler plusieurs scénarios typiques.

Scénario 1

- Création d'un `NouveauClient` qui s'inscrit.
- Recherche d'hébergements via `CollectionHebergements`.
- Consultation du détail d'un hébergement et de ses disponibilités.
- Réservation d'un hébergement pour une période donnée.

Scénario 2

- Création d'un `AncienClient` avec plusieurs réservations passées.
- Connexion de cet ancien client.
- Calcul et application d'une réduction sur une nouvelle réservation.
- Annulation d'une réservation et mise à jour de la disponibilité.

Scénario 3

- Création d'un `Administrateur`.
- Ajout d'un nouvel hébergement, modification d'un autre, suppression d'un troisième.
- Affichage de la liste des hébergements avant et après tri (par prix ou note).

Les sorties console doivent être lisibles et montrer clairement les objets et actions (utilisation de `toString()` et de `afficherDetails()`), ainsi que le polymorphisme via des références de type `Personne` et `Reservable`.

6 Livrables et organisation du travail

6.1 Livrables

- Code Java complet, compilable, avec une structure de packages claire.
- Diagramme de cas d'utilisation UML (image ou fichier d'outil UML).
- Diagramme de classes UML (image ou fichier d'outil UML).
- Court rapport (3–5 pages) comprenant :

* une description générale du projet ;
* les principaux choix de conception (héritage, interface, collections, tri) ;
* les diagrammes UML ;
* la description de quelques scénarios de test (captures de la sortie console possibles).

Ce projet doit permettre de mettre en pratique de manière cohérente et motivante l'ensemble des notions de POO, sur un cas concret de système de réservation.