TP 4 : Diagramme de classes – Interface - Classes abstraites – Polymorphisme – Collections

Exercice 1

Faire le **test 4** sur BoostCamp du QCM sur le Cours 4 et exercices Semaine 3 Polymorphisme-Classe abstraite-Interface-Collections avec le navigateur sécurisé **Safe Exam Browser**.

- QCM de 20 questions à réponse unique par question, d'une durée max de 20 minutes, extrait aléatoirement d'une banque de questions.
- Les questions et les réponses proposées sont mélangées aléatoirement.
- Pas de possibilité de retour à la question précédente, une fois celle-ci répondue et passée à la question suivante.
- Une seule tentative possible.
- Les réponses avec feedback seront visibles après la fermeture du test à la fin de la semaine.

Exercice 2 Diagramme de classes et implémentation Java de l'application Booking (suite du **TP2** et **TP3**) avec interface, classe abstraite, polymorphisme et Collection

D'après votre diagramme de classes du TP2 Cas d'utilisation et Diagramme de classes : Semaine 2 de l'application Booking et sa suite de l'**exercice 2** du TP3 Diagramme de classes et implémentation en Java de concepts de base, complétez votre diagramme de classes en introduisant les concepts suivants. **[source : aides de chatGPT]** puis les implémenter en Java.

Pour rappel, vous pouvez dessiner le diagramme de classes de l'application Booking avec le logiciel dont le lien est **draw.io (diagrams.net)** ou équivalent : voir les instructions d'utilisation dans le **TP2**.

➢ **Question 1 :** [sources : chatGPT]

a) Modifier votre diagramme de classes du **TP2** pour y définir une interface *Reservable* représentant tout élément (objet) d'hébergement réservable dans l'application : chambre d'hôtel, appartement, villa, etc.

Les exemples suivants de méthodes couvrent les principales actions qu'un utilisateur ou le système peut effectuer sur un hébergement, une activité ou autre élément réservable de cette interface :

- *estDisponible(...)* Retourne vrai si l'élément réservable est disponible, en fonction de la période donnée.
- *calculerPrix(...)* Retourne le prix total de l'élément réservable, en fonction de la période donnée et du nombre de personnes qui réservent.
- *reserver(...)* Réalise la réservation si disponibilité confirmée, en fonction de la période donnée et du client.
- *afficherDetails()* Affiche les informations de l'élément réservable.
- *annulerReservation(...)* Annule la réservation d'un client.
- *estReservee()* Vérifie si l'élément a été réservé à une date donnée
- Les méthodes « getters » nécessaires pour l'élément réservable, avec un getter par attribut défini par les classes qui implémentent cette interface (voir **question 2** suivante) : par exemple pour l'identifiant, le type, la capacité, les options incluses (le Wifi, le service ménage, la climatisation, …) de l'élément réservé.
- Etc.


Rappeler l'avantage de l'interface comme *Reservable* dans une application Booking.

b) Modifier votre diagramme de classes du **TP2** pour des classes qui héritent de la classe *Hébergement* (par exemple : *ChambreHotel*, *Appartement*, *Villa…)* implémentent l'interface *Reservable*, … avec les attributs nécessaires, la redéfinition des méthodes de l'interface et si besoin d'autres méthodes pour chacune de ces classes.

Pourquoi chacune des méthodes définies dans l'interface *Reservable* doivent être redéfinies dans chacune des classes qui l'implémentent ? Pra rapport à la classe Hébergement Quels sont les attributs est abstraite ? En quoi sa méthode ci-dessus est-elle abstraite ?

➢ **Question 2 :** Modifier votre diagramme de classes du **TP2** pour y ajouter la classe abstraite *Personne* avec les attributs et méthodes suivants :

- Attributs **protected** : nom, prénom, email, etc. ;
- Méthodes :
  - « getter » pour chacun de ses attributs **protected** ;
  - Méthode abstraite qui retourne le type de personne qui utilise l'application Booking (client ou administrateur).

Pourquoi la classe *Personne* est abstraite ? En quoi sa méthode ci-dessus est-elle abstraite ?

Chaque objet de la classe *Personne* doit être spécialisée dans les 2 classes suivantes avec leurs propriétés :

a) *Client* : en plus des méthodes de cette classe définies au **TP2**, ajouter d'autres méthodes comme calculer et visualiser la facture des hébergements réservés avec ou sans réductions, naviguer dans la disponibilité des hébergements, etc.

b) *Administrateur* : mettre à jour les hébergements actuellement disponibles, ajouter ou supprimer des hébergements, introduire diverses offres de réduction pour les anciens clients, maintenir les dossiers des clients, etc.

Pourquoi les classes *Client* et *Administrateur* sont-elles spécialisées de la classe abstraite Personne ? Qu'est-ce que cette spécialisation implique ?

➢ **Question 3 :** Dans votre diagramme de classes, intégrer une nouvelle classe *CollectionHebergements*, constituée d'une liste d'objets de la classe *Hebergement*., à ajouter dans votre diagramme de classes.

Cette nouvelle classe doit disposer de la méthode *ajouter* pour ajouter un hébergement dans la liste des objets des hébergements. Dans ce diagramme, la classe *Hebergement* implémente l'interface **java.lang.Comparable**<*Hebergement*> et redéfinit la méthode *compareTo* de cette interface pour comparer l''un des critères suivants entre hébergements : par exemple, leur prix.

Pour l'utilisation de la classe **ArrayList** (collection dynamique), voir le tutoriel Exemples avec la classe ArrayList et toutes les caractéristiques de la classe **ArrayList** sur le site suivant : **https://cs.oracle.com/javase/8/docs/api/java/util/ArrayList.html**

Quel est l'intérêt principal d'une ArrayList par rapport à un tableau ?

Exercice 3 : implémentation Java du diagramme des classes des **TP2**, **TP3** et **TP4**

Dans votre implémentation en Java des différentes classes définies de votre diagramme de classes successifs des **TP2**, **TP3** précédents et ce **TP4**, introduisez qu'il y plusieurs sortes de clients spécifiques (en différenciant les nouveaux et anciens clients) et les administrateurs, avec chacun leurs propres attributs et méthode, sans oublier les classes mentionnées du **TP2** au **TP4**, ainsi que leurs attributs et méthodes.

PS : comme méthodes à tout client de la classe *Client*, il doit être possible au minimum de réserver et payer un hébergement.

- **Question 1 :**

Pour chacune des classes spécialisées de personnes utilisant cette application Booking (classes *Client* et *Administrateur*), redéfinir la.les méthode.s abstraite.s de la classe abstraite *Personne*.

- **Question 2 :**

Vous devez au minimum définir des méthodes comme saisir et afficher les attributs de toute personne utilisant cette application Booking.

Remarque : un objet contient par défaut plusieurs méthodes de base dont la méthode redéfinie ***toString()*** héritée de la classe générale **Object**. Réécrivez toujours la méthode ***toString()*** ne serait-ce que pour déjà faciliter le débogage

Exemple :
@Override // surcharge d'une méthode héritée et redéfinie
public String toString()
{ return … ; // Une description synthétique de votre texte **}**

- **Question 3 :**

a) Dans votre diagramme de classes, intégrer une nouvelle classe *CollectionHebergements*, constituée d'une liste d'objets de la classe *Hebergement*., à ajouter dans votre diagramme de classes. Cette nouvelle classe doit disposer de la méthode *ajouter* pour ajouter un hébergement dans la liste des objets des hébergements. Dans ce diagramme, la classe *Hebergement* implémente l'interface **java.lang.Comparable**<*Hebergement*> et redéfinit la méthode *compareTo* de cette interface pour comparer l"un des critères suivants entre hébergements : par exemple, leur prix.

Pour vous aider, servez-vous des 2 liens suivants : **cours Interfaces comparable et comparator** et de l'API Comparable (Java Platform SE 8)

b) En fonction de votre diagramme de classes amélioré, implémentez les classes avec les attributs, constructeurs et méthodes nécessaires, sans oublier de les tester avec le **main** dans une autre classe.

Vous pouvez vous aider de chatGPT, Copilot ou équivalent pour des conseils, mais sans génération de code sans réflexion ni compréhension de votre part.

N'oubliez pas de citer vos sources : auteurs, liens web, etc.

JP Segado