Faculté des Sciences
Aix*Marseille Université

Année universitaire

Site : ☒ Luminy ☐ St-Charles ☐ St-Jérôme ☐ Cht-Gombert ☐ Aix-Montperrin ☐ Aubagne-SATIS
Sujet de : ☐ 1er semestre ☐ 2ème semestre ☒ Session 2 Durée de l'épreuve : 2h
Examen de : L2 Nom du diplôme : Licence d'informatique
Code du module : SIN3U02 Libellé du module : Programmation 2
Calculatrices autorisées : NON Documents autorisés : OUI

---

1 Consignes

- Créer un projet nommé `restaurant` avec comme *package* `fr.univ_amu.l2info` grâce à l'option `Create New Project`.
- Ne passez pas trop de temps sur une question. Si vous restez bloqué (code ne compilant pas ou erreur d'exécution que vous n'arrivez pas à compiler) et n'arrivez pas à corriger le problème, passez à la question suivante. Des points pourront être obtenus même pour du code ne compilant pas.
- Toutes les variables de type `List` seront à initialiser avec des objets de type `ArrayList`. Pour ajouter un élément à une liste, on peut utiliser la méthode `add`.
- Toutes les variables de type `Map` seront à initialiser avec des objets de type `HashMap`. Pour ajouter un élément à une association clé avec valeur, vous pouvez utiliser la méthode `put`. Pour récupérer une valeur à partir d'une clé, vous pouvez utiliser la méthode `get`.
- Pour chaque attribut, c'est à vous de choisir si cet attribut est `final` ou non, `private`, `protected` ou bien `public`.
- Lorsque vous avez fini de composer, trouvez votre répertoire contenant votre projet et compressez-le (clic droit puis compresser...) puis déposez l'archive sur le cours AMETICE programmation 2 au lien suivant https://ametice.univ-amu.fr/course/view.php?id=43051

2 Gestion d'un restaurant

Le but de ce sujet d'examen est de créer des classes permettant de représenter des réservations et des commandes d'un restaurant.

2.1 Classe `Client`

Cette classe permet de représenter des clients réservant des tables.

- Un attribut `name` de type `String` contenant le nom du client.
- Un constructeur évident avec un paramètre `name`.
- Une méthode `public String toString()` renvoyant le nom du client.

Créer la classe `Client` avec l'attribut, le constructeurs et la méthode demandée.

2.2 Interface `Item`

Cette interface représente les choix possibles dans le menu qui seront soit des plats qui seront représentés par la classe `Dish` ou bien des formules (entrée plus plat plus dessert) qui seront représentés par la classe `ThreeCourseMeal`.

- une méthode `getPrice` sans argument et renvoyant un prix représenté par un entier.
- une méthode `toString` sans argument et renvoyant une chaîne de caractère représentant l'item.

Créer l'interface `Item` avec les méthodes demandées.

2.3 Classe `Dish`

Cette classe permet de représenter des plats au menu. Elle devra implémenter l'interface `Item`.

- Un attribut `name` de type `String` contenant le nom du plat.
- Un attribut `price` de type `int` contenant le prix du plat.
- Un constructeur évident avec deux paramètres `name` et `price`.

— Une méthode publique `toString()` renvoyant une chaîne de caractère composée du nom du plat, suivi d'un espace, suivi du prix du plat, suivi d'un symbole €. Par exemple, pour un plat nommé `"Bouillabaisse"` avec un prix de `40`, la méthode devra renvoyer la chaîne de caractère `"Bouillabaisse : 40€"`
— Potentiellement une ou plusieurs méthodes supplémentaires qui vous semblent utiles.

Créer la classe `Dish` avec l'attribut, le constructeur et les méthodes demandées plus éventuellement des méthodes qui vous semblent utiles.

2.4 Classe `ThreeCourseMeal`

Cette classe permet de représenter une formule (une entrée, un plat et un dessert). Elle devra implémenter l'interface `Item`.

— Un attribut `name` de type `String` contenant le nom de la formule.
— Trois attributs `starter`, `mainDish` et `dessert` de type `String` qui représentent l'entrée, le plat principal et le dessert de la formule.
— Un attribut `price` de type `int` contenant le prix de la formule.
— Un constructeur évident avec 5 paramètres `name`, `price`, `starter`, `mainDish` et `dessert`.
— Une méthode publique `toString()` renvoyant une chaîne de caractère composé du nom de la formule, suivi d'un espace, suivis de l'entrée + plat principal + dessert entre parenthèse suivi du prix du plat, suivi d'un symbole €. Par exemple, le code suivant :

```
print(new ThreeCourseMeal("Formule déjeuner", 35,
                          "salade", "tartare de bœuf",
                          "crème brûlée"));
```

devra donner l'affichage suivant :

```
Formule déjeuner (salade + tartare de bœuf + crème brûlée) : 35€
```

— Potentiellement une ou plusieurs méthodes supplémentaires qui vous semble utiles.

Créer la classe `ThreeCourseMeal` avec les attributs, le constructeur et la méthode demandée plus éventuellement des méthodes qui vous semblent utiles.

2.5 Classe `Table`

La classe `Table` représentera les tables du restaurant.

— Un attribut de classe (ou champs statique) `totalNumberOfTables` qui est mis à jour pour toujours contenir le nombre de tables créées.
— Un attribut `id` de type `int` qui est l'identifiant de la table.
— Un attribut `capacity` de type `int` contenant le nombre de place de la table.
— Un constructeur prenant en argument un entier `capacity` et construisant une nouvelle table avec la capacité demandée et un identifiant différent de toutes les autres tables créées.
— Une méthode `public int getCapacity()` renvoyant le nombre de place de la table.
— Une méthode `public String toString()` renvoyant une chaîne de caractère représentant la table. La chaîne devra commencer par `"Table "` suivi de l'identifiant, suivi du nombre de places entre parenthèse. Par exemple, le code suivant :

```
print(new Table(4)));
```

devra donner l'affichage suivant :

```
Table 0 (4 places)
```

Créer la classe `Table` avec les attributs, le constructeur et la méthode demandée plus éventuellement des méthodes qui vous semblent utiles.

2.6 Classe `Booking`

Cette classe permet de représenter les réservations (*bookings* en anglais). Cette classe contiendra :

— un attribut `arrivalTime` de type `LocalTime` (du package `java.time`) qui représentera l'horaire de réservation.
— un attribut `client` de type `Client` qui représente le client réservant la table.
— un attribut `numberOfPeople` de type `int` qui représente le nombre de places réservées.
— un attribut `orderedItems` de type `List<Item>` qui représente les items commandés de cette réservation.

— un constructeur avec trois arguments `arrivalTime` de type `LocalTime`, `client` de type `Client` et `numberOfPeople` de type `int` qui construit une réservation avec les trois attributs correspondant initialisés aux bonnes valeurs et aucun item commandé (liste vide).
— une méthode `void addOrder(Item item)` ajoutant un `item` à liste des items commandés de cette réservation.

Créer la classe `Booking` avec les attributs, le constructeur et la méthode demandée.

2.7 Méthode du calcul du prix

Rajouter une méthode `int billAmount()` de la classe `Booking` calculant la somme due par les clients, c'est-à-dire le total des prix des items commandés.

2.8 Classe `Restaurant`

Cette classe va permettre la gestion des tables du restaurant. Cette classe contiendra :
— un attribut `tables` de type `List<Table>` qui contiendra toutes les tables du restaurant.
— un attribut `bookedTables` de type `Map<Table, Booking>` qui permettra d'associer des tables à des réservations.
— Un constructeur sans argument créant un restaurant sans table.
— Une méthode `public void addTable(int capacity)` ajoutant une table de la capacité demandée au restaurant.
— Une méthode `public boolean tableIsBooked(Table table)` renvoyant `true` si la table est réservée (associée à une réservation) et faux sinon.

Créer la classe `Restaurant` avec les attributs, le constructeur et les méthodes demandées.

2.9 Méthode `findAppropriateTable`

On souhaite ajouter une méthode à la classe `Restaurant`, qui étant donné un nombre de personnes, retourne une table disponible pouvant accueillir ces personnes (d'une capacité suffisante). Parmi toutes les tables qui conviennent, on en retourne une de capacité minimale. Si aucun table n'est suffisamment grande, on retourne `null`.

Rajouter la méthode `private Table findAppropriateTable(int numberOfPeople)` à la classe `Restaurant`.

2.10 Méthode `book`

On souhaite maintenant rajouter une méthode dans la classe restaurant qui crée une réservation pour un client, en trouvant une table disponible et la retournant. La table doit être enregistrée dans `bookedTables`. S'il n'y a pas de tables disponibles, la méthode retourne `null`.

Rajouter la méthode `public Table book(Client client, int numberOfSeats, LocalTime arrivalTime)` à la classe `Restaurant`.

2.11 Prendre une commande

Afin que les clients puissent commander, on souhaite ajouter les méthodes suivantes :
— une méthode `public void addOrder(Item item, Table table)` dans la classe `Restaurant`, permettant d'ajouter un `item` à la liste des commandes d'une table.
— une méthode `public void addOrder(Dish dish, int quantity)` dans la classe `Booking` permettant d'ajouter aux commandes le même `item` en plusieurs exemplaires.
— une méthode `public void addOrder(Dish dish, int quantity, Table table)` dans la classe `Restaurant` permettant d'ajouter aux commandes le même `item` en plusieurs exemplaires.

Rajouter les méthodes demandées aux classes `Booking` et `Restaurant`.

2.12 Affichage

On considère la méthode main suivante :

```
import java.time.LocalTime;

public class Main {
```

```
  public static void main(String[] args) {
    Restaurant room = new Restaurant();
    ThreeCourseMeal threeCourseMeal = new ThreeCourseMeal("Formule déjeuner", 35,
            "salade", "tartare de bœuf", "crème brûlée");
    Dish bouillabaisse = new Dish(40, "Bouillabaisse");
    room.addTable(4);
    room.addTable(2);
    room.addTable(2);
    Client marius = new Client("Marius");
    Client fanny = new Client("Fanny");
    Table fannyTable = room.book(fanny, 2, LocalTime.of(20,0));
    Table mariusTable = room.book(marius, 3, LocalTime.of(19,30));
    room.addOrder(bouillabaisse, 3, mariusTable);
    room.addOrder(threeCourseMeal, fannyTable);
    room.addOrder(bouillabaisse, fannyTable);
    System.out.println(room);
  }
}
```

Compléter des classes ***`Booking`*** *et* ***`Restaurant`*** *avec des méthodes* ***`public String toString()`****, de sorte que la méthode* ***`main`*** *produise l'affichage suivant :*

```
Table 0 (4 places)
Réservation de Marius (3, 19:30)
Bouillabaisse : 40€
Bouillabaisse : 40€
Bouillabaisse : 40€
Total : 120€

Table 1 (2 places)
Réservation de Fanny (2, 20:00)
Formule déjeuner (salade + tartare de bœuf + crème brûlée) : 35€
Bouillabaisse : 40€
Total : 75€

Table 2 (2 places)
vide
```

2.13 Factorisation du code

Les classes `Dish` et `ThreeCourseMeal` ont du code en commun.
Créez une classe abstraite ***`AbstractItem`*** *qui sera étendue par* ***`Dish`*** *et* ***`ThreeCourseMeal`*** *afin d'éviter la duplication de code.*