Projet

Station de lavage

Travail à rendre pour la première partie :

au plus tard le 08 janvier 2026 avant 20h

Dans le devoir teams intitulé projet - Partie 1 :

Ne pas oublier d'écrire un cartouche d'identification (avec vos deux noms) dans tous les fichiers de votre programme.

Travail à rendre après la dernière séance :

au plus tard le jeudi 22 janvier 2026 avant 20h

Rapport en pdf:

- une page de garde, une introduction, une conclusion,
- un jeu d'essais commenté,
- le programme commenté et indenté.

Dans le devoir teams intitulé Projet - version finale :

- le dossier, correspondant à la **version finale** du projet (première et seconde partie),
- le rapport.

Présentation Orale le 23 Janvier pour chaque groupe montrant le jeu d'essai du projet.
Questions seront demandés à chaque membre du groupe.

Avertissement : en cas de non-respect des consignes, les pénalités suivantes seront appliquées sur la note de TP

- en cas de retard dans la remise de la première partie ou dans la remise de la version finale, une pénalité de -1 point par jour de retard,
- en cas de copie entre binômes, les notes seront divisées par 2 pour 2 programmes similaires, par 3 pour 3 programmes similaires, etc.

1. Objectifs du TP :

- Concepts de base de la programmation orientée objet.
- Environnement Netbeans en mode console.

2. Description générale :

L'objectif du TP est d'écrire un programme permettant de gérer les activités d'une station de nettoyage manuel de véhicules.

L'établissement propose plusieurs types de prestations : une prestation express, une prestation pour véhicule sale et une prestation pour véhicule très sale.

Dans toutes les prestations, le nettoyage du véhicule comprend au minimum une phase de lavage et une phase de séchage.

Dans la prestation express le client peut choisir ou non de faire nettoyer l'intérieur de son véhicule (sièges et moquette), dans les deux autres ce nettoyage est toujours inclus dans la prestation.

Dans la prestation pour véhicule sale, le nettoyage comprend en plus, avant les deux phases précédentes, une phase de prélavage.

Dans la prestation pour véhicule très sale, le nettoyage comprend les mêmes trois phases que pour un véhicule sale, mais les phases de prélavage et de lavage dépendent aussi du type de salissure à nettoyer (taches de nourriture sur les sièges du véhicule, taches de boue sur la carrosserie, etc.).

Le programme doit permettre de gérer les rendez-vous des clients et de calculer le prix des prestations.

L'établissement est ouvert tous les jours de la semaine sauf le lundi (y compris les jours fériés), entre 10h et 18h, avec des créneaux disponibles pour des rendez-vous toutes les demi-heures.

Le prix total d'un nettoyage s'obtient en faisant la somme des prix des différentes phases incluses dans la prestation. Ceux-ci dépendent du type de prestation et de la catégorie du véhicule (A : citadines, B : berlines, C : monospaces et 4x4).

Les tarifs pratiqués par l'établissement sont les suivants :

- le prix du lavage d'un véhicule de catégorie A est de 20 €. Il est majoré de 50% pour les véhicules de catégorie B et de 75% pour les véhicules de catégorie C. Si le véhicule est très sale, selon le type de salissure, on applique un surcoût correspondant au produit à utiliser pour enlever les taches,

- le prix du séchage d'un véhicule de catégorie A est de 10 €. Il est majoré de 5% pour les véhicules de catégorie B et de 10% pour ceux de catégorie C,

- le prix du prélavage d'un véhicule de catégorie A est de 5 €. Comme pour le lavage, il est majoré de 50% pour les véhicules de catégorie B et de 75% pour les véhicules de catégorie C. Si le véhicule est très sale, on applique un surcoût correspondant au produit à utiliser pour enlever les taches,

- le prix du nettoyage de l'intérieur d'un véhicule de catégorie A ou B est de 30 €, il est de 40 € pour les véhicules de catégorie C.

3. Etapes de réalisation :

1ère partie :

Il est demandé **de respecter les noms de classes, attributs et méthodes indiqués ci-après**.
Cela étant, vous pouvez bien entendu, si nécessaire, ajouter des attributs ou des méthodes supplémentaires dans votre programme.

Eviter de mélanger saisies, affichages et traitements dans les méthodes.

Tester les méthodes au fur et à mesure de leur écriture, en créant dans le programme principal des instances de vos classes.

1. Créer les différentes classes de base du programme : définir les attributs, les constructeurs et des méthodes *toString*.

(a) Créer une classe *Client*. Chaque client de l'établissement est identifié par un numéro de client, son nom, son numéro de téléphone et, éventuellement, son adresse électronique.

On doit pouvoir créer un client de deux façons différentes :
- soit en spécifiant seulement un numéro de client, un nom et un numéro de téléphone,
- soit en spécifiant en plus son adresse électronique, lorsqu'elle est renseignée.

(b) Créer quatre classes de prestations :
- *Prestation.* Quelle que soit la prestation, on doit connaitre la catégorie du véhicule (A : citadines, B : berlines, C : monospaces et 4x4),
- *PrestationExpress*. Quelle que soit la prestation express, on doit savoir s'il faut nettoyer ou non l'intérieur du véhicule.
- *PrestationSale.*
- *PrestationTresSale.* Quelle que soit la prestation pour véhicule très sale, on doit connaitre le type de salissure à nettoyer (1 : taches de nourriture, 2 : taches de boue, 3 : taches de transpiration, 4 : taches de graisse, etc.).

(c) Créer une classe *RendezVous.* Pour chaque rendez-vous, on connait le client concerné, la prestation choisie et son prix.

2. Créer une classe *Etablissement.* Dans cette classe seront définis le nom de l'établissement, le nombre et l'ensemble des clients de l'établissement, et le planning de rendez-vous des 7 jours suivants.
On limitera le nombre de clients et on stockera les clients dans un tableau de dimension 1.
On stockera les rendez-vous dans un tableau de dimension 2, chaque ligne correspondant à un créneau horaire et chaque colonne à un jour.

Ecrire un constructeur et une méthode pour afficher les instances de la classe.

3. Ecrire les méthodes permettant de gérer les clients.

Le tableau contenant les clients de l'établissement sera classé par ordre lexicographique.

(a) Dans la classe *Etablissement*, écrire une méthode *rechercher* permettant de rechercher, dans le tableau de clients, le client correspondant à un nom et un numéro de téléphone donnés. La méthode doit retourner le client qu'elle a trouvé ou *null* s'il n'y en a pas.

(b) Dans la classe *Client*, écrire une méthode *placerApres* permettant de comparer deux clients. La méthode doit retourner *true* si le client correspondant à l'objet appelant doit être placé après l'autre dans le tableau des clients, *false* sinon.

(c) Dans la classe *Etablissement*, écrire deux méthodes *ajouter* permettant :
- d'ajouter un client en donnant son nom et son numéro de téléphone,
- d'ajouter un client en donnant, en plus, son adresse électronique.

Ces deux méthodes doivent retourner le client ajouté au tableau des clients de l'établissement. Les clients doivent être numérotés automatiquement par le programme, de manière continue.

4. Ecrire les méthodes permettant de gérer les prestations et les rendez-vous.

(a) Dans la hiérarchie de prestations, écrire les méthodes permettant de calculer le prix d'une prestation :
- *lavage*, *sechage* et *prelavage* permettant de calculer, respectivement, le prix du lavage, du séchage et du prélavage d'un véhicule,
- *nettoyage* permettant de calculer le prix total de la prestation.

(b) Dans la classe *Etablissement*, écrire deux nouvelles méthodes *rechercher* permettant :
- de rechercher un créneau pour un jour donné. La méthode doit afficher toutes les heures disponibles pour le jour souhaité, puis en faire choisir une au client et retourner le créneau (date et heure) correspondant,
- de rechercher un créneau pour une heure donnée. La méthode doit afficher tous les jours disponibles pour l'heure souhaitée, puis en faire choisir un au client et retourner le créneau (date et heure) correspondant.

Pour définir les jours, heures et créneaux, utiliser les classes *LocalDate*, *LocalTime* et *LocalDateTime* (cf. annexe).

(c) Dans la classe *Etablissement*, écrire trois nouvelles méthodes *ajouter* permettant :
- d'ajouter un rendez-vous pour une prestation express en donnant le client, le créneau choisi (date et heure), la catégorie du véhicule et en indiquant si l'intérieur du véhicule doit être nettoyé ou non,
- d'ajouter un rendez-vous pour une prestation pour un véhicule sale en donnant le client, le créneau choisi (date et heure) et la catégorie du véhicule,
- d'ajouter un rendez-vous pour une prestation pour un véhicule très sale en donnant le client, le créneau choisi (date et heure), la catégorie du véhicule et le type de salissure à nettoyer.

Ces trois méthodes doivent retourner le rendez-vous ajouté au tableau des rendez-vous de l'établissement.

2ème partie :

1. Dans la classe *Etablissement*, écrire une méthode *planifier* permettant de planifier un rendez-vous pour le nettoyage d'un véhicule. Le programme doit :
   - demander le nom et le numéro de téléphone du client et, s'il s'agit d'un nouveau client, l'ajouter dans le tableau de clients de l'établissement,
   - lui faire choisir un créneau dans les 7 jours suivants,
   - lui faire choisir le type de prestation et lui demander les informations correspondantes,
   - ajouter le rendez-vous dans le tableau de rendez-vous de l'établissement et indiquer le prix de la prestation au client.

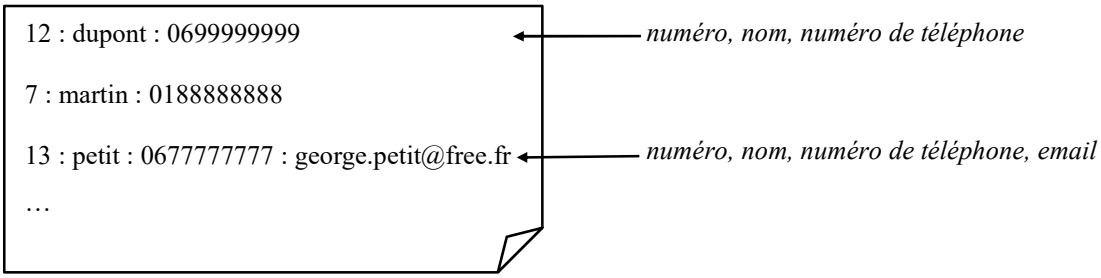
   Utiliser les méthodes précédemment écrites.

2. Dans la classe *Etablissement*, ajouter des méthodes *afficher* permettant :
   - d'afficher le planning des rendez-vous pour un jour donné.
   - d'afficher le(s) client(s) de l'établissement correspondant à un nom ou à un numéro de téléphone donné,
   - d'afficher le(s) rendez-vous pris par le client correspondant à un numéro de client donné,

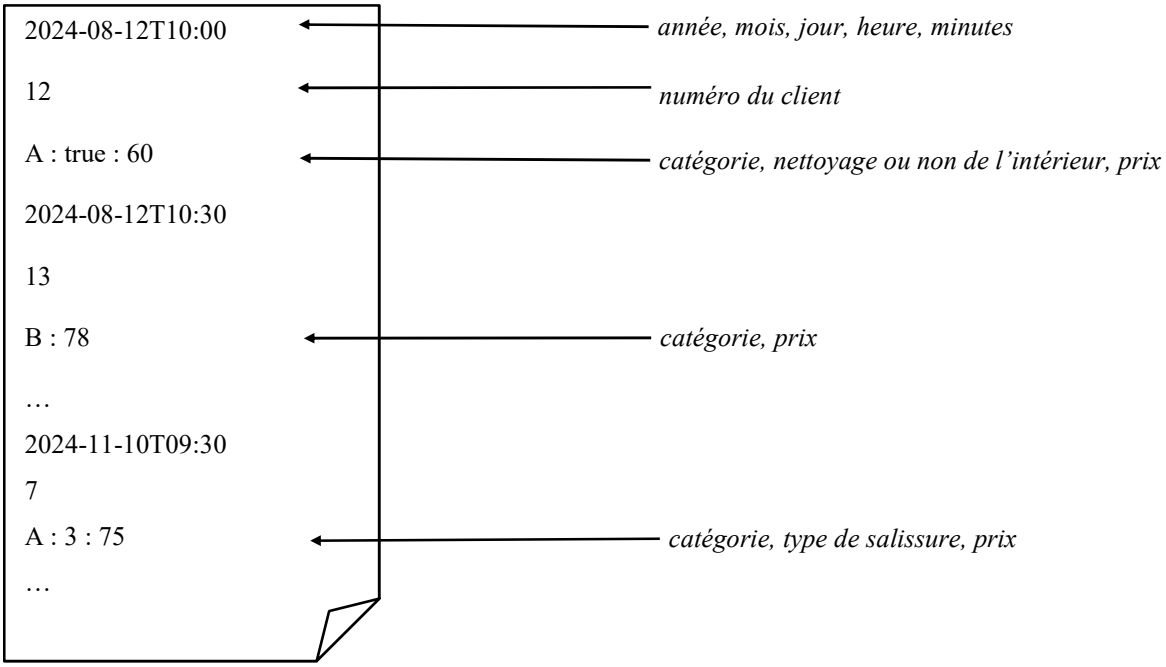
3. On veut que l'ensemble des clients et des rendez-vous de l'établissement soient sauvegardés dans des fichiers texte.

   Les deux fichiers doivent respecter le format suivant (sans aucune ligne vide) :

   - le fichier des clients :



   - le fichier des rendez-vous :



(a) Dans la classe *Client*, dans la hiérarchie de prestations et dans la classe *RendezVous*, écrire une méthode *versFichier* retournant sous forme d'une chaine de caractères les informations à écrire dans le fichier.

(b) Dans la classe *Etablissement*, écrire les méthodes pour gérer le fichier des clients :

- la méthode *versFichierClients* permettant de sauvegarder dans un fichier texte les informations contenues dans le tableau des clients,
- la méthode *depuisFichierClients* permettant de charger l'ensemble des clients à partir d'un fichier texte.

Pour écrire dans un fichier texte, utiliser la classe *FileWriter* (cf. annexe).
Pour lire dans un fichier texte, utiliser les classes *FileReader et BufferedReader* (cf. annexe).

(c) Dans la classe *Etablissement*, écrire les méthodes pour gérer le fichier des rendez-vous :

- la méthode *versFichierRDV* permettant d'ajouter les nouveaux rendez-vous dans un fichier texte contenant tous les rendez-vous de l'établissement,
- la méthode *depuisFichierRDV* permettant de charger les rendez-vous des 7 jours suivants à partir d'un fichier texte contenant tous les rendez-vous de l'établissement.

4. Dans la classe principale du projet :

- créer une instance de la classe *Etablissement*,
- appeler les méthodes de chargement des fichiers texte,
- ajouter un menu comportant les différentes fonctionnalités du programme (cf. méthodes de la classe *Etablissement*),
- appeler les méthodes de sauvegarde dans des fichiers texte, avant de quitter le programme.

Annexe

Classe LocalTime :

- *static LocalTime of (int h, int m)* : retourne une instance de la classe LocalTime correspondant aux heure et minutes *h* et *m* donnés.
- *static LocalTime now ( )* : retourne une instance de la classe LocalTime correspondant à l'heure actuelle du système.
- *static LocalTime parse (String ch) :* retourne une instance de la classe LocalTime correspondant à la conversion de la chaîne de caractères *ch* donnée. Par défaut, le format utilisé est : " hh:mm ".
- *long until (LocalTime t2, MINUTES) :* retourne le nombre de minutes écoulées entre l'heure correspondant à l'objet appelant et l'heure *t2* donnée (accepter d'importer « java.time.temporal.ChronoUnit.MINUTES »).

Classe LocalDate :

- *static LocalDate of (int y, int m, int d)* : retourne une instance de la classe LocalDate correspondant à l'année, mois et jour du mois *y, m* et *d* donnés.
- *static LocalDate now ( )* : retourne une instance de la classe LocalDate correspondant à la date du jour.
- *static LocalDate parse (String ch) :* retourne une instance de la classe LocalDate correspondant à la conversion de la chaîne de caractères *ch* donnée. Par défaut, le format utilisé est : " yyyy-MM-dd ".
- *long until (LocalDate d2, DAYS) :* retourne le nombre de jours écoulés entre la date correspondant à l'objet appelant et la date *d2* donnée (accepter d'importer « java.time.temporal.ChronoUnit.DAYS»).

Classe LocalDateTime :

- *static LocalDateTime of (int y, int M, int j, int h, int m)* : retourne une instance de la classe LocalDateTime correspondant à l'année, mois, jour du mois, heure et minutes *y*, *M*, *j, h* et *m* donnés.
- *static LocalDateTime now ( )* : retourne une instance de la classe LocalDateTime correspondant à la date et heure actuelle du système.
- *static LocalDateTime parse (String ch) :* retourne une instance de la classe LocalDateTime correspondant à la conversion de la chaîne de caractères *ch* donnée. Par défaut, le format utilisé est : " yyyy-MM-dd**T**hh :mm ".
- *boolean isAfter (LocalDateTime d),* resp. *isBefore* : vérifie si la date correspondant à l'objet appelant est postérieure, resp. antérieure, à la date *d* donnée.
- *LocalDateTime plusDays (long n)* : retourne la date correspondant à l'ajout du nombre de jours *n* donné à la date correspondant à l'objet appelant.

Classe FileWriter :

- *FileWriter* (String *nomF*) : crée une instance de la classe *FileWriter*, *nomF* est le nom du fichier dans lequel on veut écrire.
- *FileWriter* (String *nomF,* boolean *ajout*) : crée une instance de la classe *FileWriter*, *nomF* est le nom du fichier dans lequel on veut écrire et *ajout* indique s'il faut écrire à la fin du fichier (cas où *ajout* est égal à *true*) ou s'il faut écraser le fichier existant (cas où *ajout* est égal à *false*).
- *void write (*String *lig)* : écrit la ligne *lig* donnée dans le fichier texte.
- *void close ( )* : ferme le fichier.

Classe System :

- *static String lineSeparator ( )* : retourne la chaine de caractères correspondant au passage à la ligne (ce caractère dépend du système d'exploitation).

Classe FileReader :

- *FileReader* (String *nomF*) : crée une instance de la classe *FileReader*, *nomF* est le nom du fichier dans lequel on veut lire.
- *void close ( )* : ferme le fichier.

Classe BufferedReader :

- *BufferedReader (FileReader fich)* : crée une instance de la classe BufferedReader, *fich* est l'instance de la classe FileReader correspondant au fichier texte que l'on veut lire.
- *String readLine ( )* : lit et retourne une ligne du fichier.

Classe String :

- *String [] split (String exp)* : retourne un tableau contenant les sous-chaînes de la chaîne de caractères correspondant à l'objet appelant, en la découpant selon l'expression *exp* donnée.