Epreuve de Programmation Orientée Objet
DEUG MIAS 2 *

21 mars 2003

Durée : 2 heures.
Tous les documents sont interdits.

1 Programmation de classes en JAVA

1.1 Ce qui est évalué dans cette épreuve

Votre capacité à écrire un programme JAVA syntaxiquement correct et qui fonctionne sans erreurs. Plus précisément, il s'agit d'évaluer vos capacités :

- à définir et utiliser une classe, un constructeur, des méthodes, des champs ;
- à écrire des commentaires décrivant de manière claire et précise le rôle et le fonctionnement d'une classe et de ses caractéristiques ;
- à effectuer des choix corrects pour les types des champs, les arguments des méthodes et le type de retour des méthodes ;
- à effectuer des choix corrects pour la visibilité des champs et des méthodes ;
- à définir et utiliser des tableaux.

1.2 Présentation du problème

On souhaite réaliser une application permettant de gérer l'agenda d'une personne. Les deux classes `Agenda` et `RendezVous` sont destinées à gérer les données essentielles concernant l'ensemble des rendez-vous d'une personne. Tout ce qui concerne la gestion des interactions avec l'utilisateur (affichage, saisie des données...) est supposé être pris en charge par d'autres classes et ne sera pas traité ici.

Note : L'interface d'une classe décrit l'ensemble des caractéristiques de la classe dont la visibilité est `public`. Cette description se fait en donnant :

- un commentaire décrivant la classe de manière générale.
- les entêtes de toutes les méthodes de visibilité `public`.
- pour chaque méthode de visibilité `public`, un commentaire décrivant de manière précise et non ambiguë le rôle et le mode d'emploi de cette méthode.

Pour traité l'exercice 1.1, il est fortement recommandé de s'inspirer des interfaces des classes `Agenda` et `Date` qui figurent en annexe A et B.

Exercice 1.1 Interface de la classe `RendezVous`

Définissez l'interface de la classe `RendezVous`. L'interface de cette classe devra respecter les contraintes suivantes :

- un rendez-vous est défini par la donnée de la date du rendez-vous, d'un descriptif (sous la forme d'une chaîne de caractères) et d'une durée en heures ;
- une fois un rendez-vous créé sa date ne doit pas pouvoir être modifiée (si l'utilisateur veut changer la date d'un rendez-vous dans son agenda, l'application supprime le rendez-vous et insère un nouveau rendez-vous avec la nouvelle date dans l'agenda) ;
- le descriptif doit impérativement être défini à la création du rendez-vous, mais devra pouvoir être modifier par la suite ;

*M. Champesme – Département d'Informatique – Institut Galilée

– la durée doit, elle aussi, être fixée lors de la création, mais si cette durée n'est pas fixée par l'utilisateur lors de la création du rendez-vous, une valeur par défaut sera utilisée (par exemple 1 heure). Cette durée doit pouvoir être modifiée par la suite, mais devra toujours être strictement positive.

Vous utiliserez la classe `Date` dont l'interface est donnée à l'annexe B pour représenter la caractéristique date des rendez-vous.

Exercice 1.2 Implémentation des classes.

1. Donnez une implémentation complète (définitions de champs, ajout du code de chaque méthode) de la classe `RendezVous`. Cette implémentation devra être conforme à l'interface définie à la question précédente ;
2. Donnez une implémentation complète et conforme à son interface de la classe `Agenda` : le code écrit doit correspondre à l'interface de la classe donnée à l'annexe A.

A Interface de la classe `Agenda`

Classe `Agenda`

La classe `Agenda` définit les services de base nécessaire pour la gestion de l'agenda d'une personne, c'est-à-dire : création d'un agenda pour une personne donnée, possibilité d'ajouter ou de supprimer des rendez-vous. Par ailleurs, différentes méthodes permettent de consulter la liste des rendez-vous qui est triée par ordre croissant selon la date du rendez-vous :

– rendez-vous inscrits dans l'agenda à partir d'une date donnée
– rendez-vous inscrits dans l'agenda à partir de la date courante
– *n* derniers rendez-vous inscrits dans l'agenda, pour *n* choisit par l'utilisateur (i.e. les *n* derniers rendez-vous de la liste triée des rendez-vous)

La liste des rendez-vous est mémorisée dans un tableau dont la taille est fixée à la création de l'agenda. Lorsque le nombre de rendez-vous atteint la taille fixée initialement, la liste est recopiée dans un nouveau tableau de taille supérieure, qui se substitue ensuite au tableau précédent.

Agenda

```
public Agenda(String nom, String prenom)
```

Initialise un Agenda pour la personne dont le nom et le prénom sont transmis en paramètre.

Paramètres :

`nom` - Le nom de la personne pour laquelle l'agenda est créé.

`prenom` - Le prénom de la personne pour laquelle l'agenda est créé.

---

getNom

```
public String getNom()
```

Renvoie le nom du propriétaire de l'agenda.

---

getPrenom

```
public String getPrenom()
```

Renvoie le prénom du propriétaire de l'agenda.

---

getNbRendezVous

`public int` **getNbRendezVous**`()`

Renvoie le nombre de rendez-vous actuellement inscrits dans l'agenda.

---

ajouterRendezVous

`public void` **ajouterRendezVous**`(RendezVous nouveauRendezVous)`

Ajoute un rendez-vous dans l'agenda.

Paramètres :

`nouveauRendezVous` - Le rendez-vous à ajouter.

---

supprimerRendezVous

`public void` **supprimerRendezVous**`(Date uneDate)`

Supprime de l'agenda le (ou les) rendez-vous dont la date est la même que celle passée en paramètre. Si aucun rendez-vous présent dans l'agenda ne possède une date égale à `uneDate`, cette méthode n'a aucun effet sur l'agenda.

Paramètres :

`uneDate` - La date du (ou des) rendez-vous à supprimer.

---

getRendezVousApres

`public RendezVous[]` **getRendezVousApres**`(Date uneDate)`

Renvoie la liste des rendez-vous de l'agenda dont la date est postérieure à la date transmise en paramètre. La liste est renvoyée sous la forme d'un tableau dont les éléments sont triées par ordre croissant selon la date des rendez-vous. La taille du tableau renvoyé est égale au nombre de rendez-vous de cette liste.

Paramètres :

`uneDate` - La date qui détermine les rendez-vous sélectionnés.

Résultat : Un tableau de `RendezVous` trié par ordre croissant dont les éléments sont les rendez-vous de l'agenda postérieurs à `uneDate`.

---

getRendezVousFuturs

`public RendezVous[]` **getRendezVousFuturs**`()`

Renvoie la liste des rendez-vous de l'agenda dont la date est postérieure à la date courante. La liste est renvoyée sous la forme d'un tableau dont les éléments sont triées par ordre croissant selon la date des rendez-vous. La taille du tableau renvoyé est égale au nombre de rendez-vous de cette liste.

Résultat : Un tableau de `RendezVous` trié par ordre croissant dont les éléments sont les rendez-vous de l'agenda postérieurs à la date courante.

---

getNDerniersRendezVous

`public RendezVous[] getNDerniersRendezVous(int nbRendezVous)`

Renvoie les `nbRendezVous` derniers rendez-vous de l'agenda, c'est-à-dire les derniers rendez-vous dans l'ordre chronologique. La liste est renvoyée sous la forme d'un tableau dont les éléments sont triées par ordre croissant selon la date des rendez-vous. La taille du tableau renvoyé est égale à `nbRendezVous`.

Paramètres :

`nbRendezVous` - Le nombre de rendez-vous à renvoyer.

Résultat : Un tableau de `RendezVous` trié par ordre croissant dont les éléments sont les `nbRendezVous` derniers rendez-vous de l'agenda.

B Interface de la classe `Date`

Classe Date

La classe `Date` représente un instant spécifique dans le temps. Cet instant est mesuré avec une précision de l'ordre de la milliseconde.

Date

`public Date()`

Initialise une instance de la classe `Date` de sorte qu'elle représente l'instant dans le temps auquel elle a été allouée. Cet instant est mesuré avec une précision de l'ordre de la milliseconde.

Date

`public Date(long nbms)`

Initialise une instance de la classe `Date` de sorte qu'elle représente l'instant dans le temps situé `nbms` millisecondes après la base de temps standard, c'est-à-dire, le 1er janvier 1970 à 00 :00 :00 heure GMT.

Paramètres :

`nbms` - Le nombre de millisecondes depuis le 1er janvier 1970 à 00 :00 :00 heure GMT.

clone

`public Object clone()`

Renvoie une copie de l'instance courante représentant le même instant.

getTime

`public long getTime()`

Renvoie le nombre de millisecondes depuis le 1er janvier 1970 à 00 : 00 : 00 heure GMT, correspondant à l'instant représenté par cette instance de la classe `Date`.

---

setTime

`public void` **setTime**`(long nbms)`

Change la valeur de cette instance de la classe `Date`, afin qu'elle représente l'instant correspondant à `nbms` millisecondes depuis le 1er janvier 1970 à 00 :00 :00 heure GMT.

Paramètres :

`nbms` - le nombre de millisecondes.

---

before

`public boolean` **before**`(Date autreDate)`

Teste si la date courante est avant la date transmise en paramètre.

Paramètres :

`autreDate` - Une date.

Résultat : `true` si et seulement si l'instant représenté par l'instance courante est strictement antérieur à l'instant représenté par `autreDate` ; `false` sinon

---

after

`public boolean` **after**`(Date autreDate)`

Teste si la date courante est après la date transmise en paramètre.

Paramètres :

`autreDate` - Une date.

Résultat : `true` si et seulement si l'instant représenté par l'instance courante est strictement postérieur à l'instant représenté par `autreDate` ; `false` sinon

---

equals

`public boolean` **equals**`(Object obj)`

Teste si deux dates sont égales. Le résultat est `true` si et seulement si l'argument n'est pas `null` et est une instance de la classe `Date` représentant le même instant (à la milliseconde près) que l'instance courante. Deux instances de la classe `Date` sont *equals* si et seulement si la méthode `getTime` renvoie la même valeur pour les deux instances.

Paramètres :

`obj` - L'objet avec lequel comparer l'instance courante.

Résultat : `true` si les instances représentent le même instant ; `false` sinon.
