SEMAINE 4 : TP1 - ARRAYLIST

La classe **`ArrayList`** , du package **`java.util`**, représente un tableau dynamique d'objets.

- On ne peut placer dans une **`ArrayList`** que des objets (*pas de types primitifs)* .
- Pour une liste de valeurs entières, on utilise une **`ArrayList<Integer>`** .
  - *AutoBoxing : depuis java 5 les types primitifs sont convertis en leur Wrapper dès que nécessaire.*
- Une syntaxe particulière permet le parcours intégral d'une liste (entre autres) sans avoir à gérer d'indice :
  `for (Type objet : ArrayList<Type>) {…}`
  `objet` prend successivement toutes les valeurs de l'**`ArrayList`** spécifiée.
  - Exemple :
    - supposons une liste de chaînes stockées dans une **`ArrayList<String>`** représentant un dictionnaire, la boucle suivante affichera le contenu du dictionnaire :

```
for (String mot :  dictionnaire ){
     System.out.println(mot);
}
```

Consulter la Javadoc pour pouvoir utiliser correctement la classe **`java.util.ArrayList`**.

ALPHABET

Créer une classe **`Alphabet`** dont la méthode **`main()`** :

1. Crée une liste contenant dans l'ordre les 26 lettres de l'alphabet en majuscule.
   A l'aide de la classe **`ArrayList`** *et de sa méthode* **`add()`** *,*
2. Affiche la taille de la liste,
3. Affiche le 5ème élément de la liste,
4. Affiche la position du caractère 'S' dans la liste,
5. Supprime tous les éléments entre la 4ème et la 6ème position,
6. Affiche la liste au complet
   a) sans utiliser de boucle,
   b) en utilisant une boucle.
7. Ajoute l'élément '?' en 100ème position, *Que se passe-t-il ?*
8. Efface tous les éléments de la liste.

EVÉNEMENT

Un événement est un fait qui survient à un moment donné, et pour une certaine durée.

La classe `Evenement` doit contenir les attributs nécessaires à son fonctionnement :

- un `intitulé` (chaîne de caractères décrivant l'événement),
- un `lieu` (chaîne de caractères désignant le lieu),
- une date de `début` (objet de type `GregorianCalendar`),
- une date de `fin` (idem).

1. **Ecrire la classe** `Evenement` complète :
   a. Constructeurs qui contrôlent l'initialisation : date de début doit précéder la date de fin (dans le cas contraire, on considérera que la date de début est aussi celle de fin)
   b. Méthode `String toString()` retournant une chaîne avec la valeur des attributs,
   c. Méthode `boolean equals(Evenement evenement)` comparant un événement donné à l'événement courant.

2. **Ajouter à cette classe une méthode** `boolean chevauche(Evenement evenement)` qui retourne vrai si une partie de l'événement courant se déroule en même temps que celui entré en paramètre *(faux sinon)*.

Note :
On notera que si
1. la fin de l'événement courant est avant le début celui entré en paramètre,
2. ou l'inverse (la fin de l'événement passé en paramètre est avant le début de l'événement courant),

alors il n'y a pas chevauchement (*les autres cas induisent un chevauchement*).

2 AGENDA

Nous allons nous servir de la classe `Evenement` pour créer un type `Agenda`.

L'agenda sera une liste d'événements (*liste non ordonnée dans un premier temps*).

Il permettra de :

1. tester si un événement peut être entré dans l'agenda
   - i.e. s'il n'y a pas de chevauchement ⇔ ne chevauche aucun des événements déjà présents,
2. entrer un événement dans l'agenda,
3. supprimer un événement de l'agenda,
   - en spécifiant un événement, celui-ci doit exister dans l'agenda, rien ne se passe sinon
   - en spécifiant un indice, l'indice doit être un indice valide, rien ne se passe sinon
4. afficher l'ensemble des événements.

En plus de la méthode `toString()`, **écrire les méthodes suivantes** :

```
boolean entrable(Evenement evenement){...}

void entrerEvenement(Evenement evenement){...}

void supprimerEvenement(Evenement evenement){...}

void supprimerEvenement(int indice){…}
```

Travail à réaliser :

1. **Ecrire la classe `Agenda`** avec les méthodes spécifiées.
2. **Ecrire une classe `EssaiAgenda`** avec une méthode `main()` qui crée un agenda, des événements de votre choix, et les entre dans cet agenda.
3. **Ajouter à la classe `Agenda`** la méthode de signature :
   `void supprimerChevauchants(Evenement evenement)`
   qui supprime tous les événements qui chevauchent l'événement fourni comme paramètre.