Examen Ecrit

Tous documents sur support papier autorisés. Durée 2h.

Question 1. Savoir déclarer et signaler des exceptions ; savoir écrire une assertion.

Dans ce sujet, nous aurons besoin de manipuler des heures et des intervalles horaires sur une journée (entre 0h et 23h59). Une heure sera représentée par une chaîne de 4 caractères au format HHMM, par exemple "1300" représente l'heure de début de cet examen (13h), "0559" représente l'heure 5h59, et les heures s'échelonnent entre "0000" et "2359". Les heures sont ordonnées par ordre lexicographique. Une heure qui n'est pas formée sur 4 caractères ou bien dont les deux derniers caractères ne sont pas compris par ordre lexicographique entre "00" et "59" est mal formée. Par exemple "13" et "1378" sont des heures mal formées.

On vous donne la classe **Intervalle** ci-dessous. Elle comprend deux attributs représentant respectivement l'heure de début et l'heure de fin de l'intervalle, un constructeur pour les initialiser, une méthode d'instance permettant de savoir si un intervalle en précède un autre, une méthode d'instance permettant de savoir si un intervalle en chevauche un autre, une méthode `toString` et enfin trois méthodes statiques utilitaires permettant de savoir si une heure en suit ou en précède une autre et si une chaîne de caractères est bien formée pour représenter la partie minutes d'une heure.

```
public class Intervalle{

   private String heureDebut, heureFin;

   public Intervalle(String d, String f){
      this.heureDebut = d; this.heureFin=f;
   }

   public boolean precede(Intervalle v2){return precede(this.heureFin, v2.heureDebut);}

   public boolean chevauche(Intervalle v){return !(this.precede(v) || v.precede(this));}

   public String toString(){return this.heureDebut+"-"+this.heureFin;}

   public static boolean precede(String heure1, String heure2)
   {return heure1.compareTo(heure2)<=0;}

   public static boolean suit(String heure1, String heure2)
   {return heure1.compareTo(heure2)>=0;}

   public static boolean minutesCorrectes(String h)
   {return precede(h.substring(2, 4),"59")&& suit(h.substring(2, 4),"00");}
}
```

a- Ecrivez dans la classe `Intervalle` une méthode statique permettant de savoir si une heure est correctement formée.

b- On veut compléter le constructeur afin que celui-ci soit protégé sur les aspects suivants : l'heure de début et l'heure de fin doivent être correctement formées ; l'heure de début doit précéder l'heure de fin. Cela vous amènera : à écrire des classes exception pour représenter les erreurs indiquées ; à réécrire le constructeur.

c- On veut compléter le constructeur pour vérifier la post-condition selon laquelle l'affectation de valeurs aux attributs a été correctement réalisée. Indiquez quelle assertion vous écririez et où vous la placeriez.

On dispose à présent aussi de l'énumération `public enum JourSemaine{Lu,Ma,Me,Je,Ve,Sa,Di}`

Question 2. Ecrire une interface `IEmissionRadio`.
On souhaite à présent représenter des émissions de radio. Une émission de radio a un titre, elle est diffusée un ou plusieurs jours de la semaine au même intervalle horaire. Ecrivez le code Java d'une interface pour représenter une émission de radio munie : (1) d'une méthode `getTitre` permettant de connaître le nom sous forme d'une chaîne de caractères, (2) d'une méthode `getChaine` permettant de connaître le nom de la chaîne de radio qui la diffuse sous forme d'une chaîne de caractères, (3) d'une méthode `getJoursSemaines` permettant de connaître la liste des jours de la semaine où l'émission est diffusée, (4) d'une méthode `getHoraire` permettant de connaître l'intervalle horaire de diffusion, (5) d'une méthode `boolean precede(EmissionRadio autreEmission)` permettant de savoir si l'émission (this, receveur du message) se termine avant `autreEmission`, (6) d'une méthode `boolean chevauche(EmissionRadio autreEmission)` permettant de savoir si l'émission (this, receveur du message) a une partie qui se déroule en même temps que `autreEmission`. Lorsque c'est possible, écrivez le code des méthodes.

Question 3. Compléter l'interface `IEmissionRadio`.
On rappelle l'interface générique existant en Java pour représenter les objets comparables : `public Interface Comparable<T> {int compareTo(T o)}`. Comme l'indique la documentation, dans une implémentation, la méthode `compareTo` doit retourner *a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.* Compléter le code de l'interface `IEmissionRadio` (indiquez les modifications sans tout recopier) de manière à ce que :

- elle spécialise l'interface `Comparable`, pour représenter le fait que les émissions de radio sont comparables entre elles (ajustez le paramètre `T` au problème) ;
- elle contienne une méthode `default compareTo`. Une émission de radio est égale à une autre si leurs intervalles horaires se chevauchent ; une émission de radio est plus petite (resp. grande) qu'une autre si elle la précède (resp. suit).

Question 4. Implémenter une interface.

a- Ecrivez une classe `EmissionFranceCulture` qui implémente l'interface `IEmissionRadio`. N'écrivez que les attributs et les méthodes nécessaires pour implémenter l'interface.

b- Ajoutez à la classe `EmissionFranceCulture` une méthode `equals`, redéfinissant celle de la classe `Object`, qui retourne vrai lorsque deux émissions se chevauchent.

c- Puis écrivez l'en-tête d'une classe `EmissionFranceCultureSciences` qui est sous-classe de `EmissionFranceCulture` (uniquement l'en-tête). Pour la suite, leurs constructeurs et accesseurs sont tous supposés exister avec les conventions classiques.

Question 5. Utiliser une classe de l'API et écrire une structure de données
Nous introduisons la classe générique `TreeSet<T>` de l'API standard de Java. Cette classe permet de représenter des ensembles à l'aide d'une structure interne arborescente. Un ensemble est une collection d'objets sans doublons (pour savoir si des objets sont identiques, les méthodes `compareTo` et `equals` de la classe des objets stockés dans l'ensemble sont appelées). `TreeSet` dispose entre autres des méthodes suivantes (s étant un `TreeSet`) :

- un constructeur sans paramètre pour créer un ensemble vide,
- $s.size()$ retourne le nombre d'éléments de s,
- $s.add(e)$ — ajoute e à s si e n'est pas déjà présent (vérifié avec `equals`),
- $s.contains(e)$ — retourne true si s contient e (vérifié avec `equals`).

Utilisez la classe `TreeSet<T>` pour mettre en place une classe **générique** `ProgrammeEcouteJournalier`. Un programme d'écoute journalier est prévu pour une journée particulière de la semaine. Il contient un ensemble d'émissions de radio que l'on désire écouter ce jour-là. Cela demande que ces émissions ne se chevauchent pas et qu'elles soient bien diffusées le jour en question.

De plus on désire représenter des programmes d'écoute journaliers contenant un ensemble d'émissions hétérogènes (instances de plusieurs classes différentes, comme des émissions de France Culture ou des émissions de France Info) ou homogènes (instances d'une même classe, par exemple uniquement des émissions de France Culture instances de la classe `EmissionFranceCulture`, la classe en question devant être obligatoirement une classe d'émissions de radio). Dans cette question, écrivez :

- l'entête de la classe générique `ProgrammeEcouteJournalier`,
- ses attributs en les initialisant,
- une méthode d'ajout vérifiant les contraintes indiquées plus haut (pas de chevauchement, journée de diffusion correcte) ou signalant sous forme d'affichage d'un message une possible erreur.

Question 6. Ecrire un main
Complétez le `main` suivant dans lequel vous capturerez toutes les exceptions identifiées à la question plus haut qui pourraient potentiellement y surgir. Ce programme comprend (1) la création de deux émissions (2) la création d'un programme d'écoute journalier et l'ajout des émissions dans ce programme d'écoute journalier.

```
public static void main(String[] args) {
   ......  à compléter ......
   EmissionFranceCulture
      e1 = new EmissionFranceCultureSciences("Continent Sciences","Stéphane Deligeorges",
                  liste1, new Intervalle("1600","1678")),
      e2 = new EmissionFranceCultureSciences("La bibliothèque des Sciences","Victor Sheriko",
                  liste2, new Intervalle("1830","1800"));

   ProgrammeEcouteJournalier<  ......  à compléter ...... > pj1
        = new ProgrammeEcouteJournalier< ......  à compléter ......>(JourSemaine.Je);
   pj1.ajoute(e1);
   pj1.ajoute(e2);
      ......  à compléter ......
}
```

Question 7. Comprendre les streams et ajouter une méthode manipulant la structure (question optionnelle, à faire seulement si vous avez fini à temps le reste)
En supposant que l'ensemble des émissions d'un programme journalier se nomme `listeEmissions`. On peut écrire la méthode suivante basée sur les `streams`.

```
  public void m(){
      listeEmissions
        .stream()
        .filter(e -> e.getJoursSemaines().size()==7)
        .map(e -> e.getTitre()+" "+e.getHoraire())
        .forEach(e->System.out.println(e));
  }
```

Expliquez ce qu'elle fait et réécrivez cette méthode avec une itération classique (`for` ou `while`).