Examen

Durée : 3h. Aucun document autorisé. Le barême est donné à titre indicatif.

Exercice 1 : (programmation - 15 points)

Le but de cet exercice est de développer une nouvelle classe de collection semblable à celles de l'API. Cette classe permettra de stocker des tableaux en les maintenant triés.

1. Écrire une classe `SortedTab` stockant un tableau de chaînes de caractères. Le constructeur prendra en paramètre la taille du tableau ; cette taille ne changera pas ultérieurement. Tous les attributs de cette classe seront privés. Ajouter les méthodes de prototype suivant :

   - `public void add(String s)` permet d'ajouter une chaîne de caractères dans le tableau. Le tableau doit resté trié par ordre alphabétique (on rappelle que la méthode `compareTo(String s2)` appelée sur un objet `s1` de la classe `String` renvoie un entier négatif si `s1` est plus petit que `s2`, 0 si les chapines sont identiques, et un entier positif si `s1` est plus grand que `s2`).

     Si le tableau est plein, cette méthode ne fait rien.

   - `public void remove(String s)` supprime la chaîne de caractères `s` du tableau. Les éléments se trouvant après `s` dans le tableau doivent être décalés d'un cran afin que le tableau ne comporte pas de case vide.

     Si la chaîne n'apparaît pas dans le tableau, cette méthode ne fait rien.

   - `public boolean isMember(String s)` renvoie `true` si `s` est dans le tableau, et `false` sinon.

2. Écrire une classe `public class MonProg` comportant une unique méthode `main` qui permet de tester la classe `SortedTab`. Y tester les appels aux trois méthodes ci-dessus.

3. On souhaite gérer les problèmes des méthodes `add` et `remove` en utilisant des exceptions. Définir une exception vérifiée `NoSpaceLeftException extends Exception` et une exception non vérifiée `NoSuchElementException extends RuntimeException`. Modifier la méthode `add` afin qu'elle lève l'exception `NoSpaceLeftException` si le tableau est plein, et la méthode `remove` afin qu'elle lève l'exception `NoSuchElementException` si la chaîne de caractères n'appartient pas au tableau.

   Modifier la classe `MonProg` afin de rattraper les erreurs potentielles et le cas échéant d'afficher un message approprié.

4. On veut maintenant définir un itérateur pour parcourir notre ensemble. Définir une classe `MyIterator` qui implante l'interface `java.util.Iterator` dont voici la définition :

   ```
   interface Iterator {
   ```

```
  boolean hasNext(); // Returns true if the iteration has more elements.
  String next(); // Returns the next element in the iteration.
}
```

Pour notre itérateur, on souhaite que les éléments soient renvoyés en commençant par l'élément stocké en dernière position dans le tableau et en terminant par le premier élément du tableau. Le constructeur de la classe `MyIterator` prendra en paramètre le tableau à parcourir ainsi que le nombre d'éléments qui y sont stockés. la méthode `next` lèvera l'exception `NoSuchElementException` s'il ne reste plus d'élément à renvoyer.

5. Afin d'utiliser l'itérateur, modifier classe `SortedTab` afin qu'elle implante l'interface `java.util.Iterable` dont voici la définition :

```
interface Iterable {
  Iterator iterator(); // Returns an iterator over a set of elements.
}
```

Dans notre cas, il faudra évidemment utiliser un objet de la classe `MyIterator`.

Ajouter dans la classe `SortedTab` une méthode `public int getSize()` qui renvoie le nombre d'éléments stockées dans le tableau ; cette méthode devra utiliser l'itérateur pour calculer le nombre d'éléments du tableau.

6. **(Bonus)** Modifier la classe `SortedTab` afin qu'elle puisse stocker des éléments de types différents, en utilisant la généricité de Java. Modifier la classe `MyIterator` en conséquence. Voici les définitions correctes des interfaces `Iterator` et `Iterable` :

```
interface Iterator <E> {
  boolean hasNext(); // Returns true if the iteration has more elements.
  E next(); // Returns the next element in the iteration.
}
interface Iterable <E> {
  Iterator <E> iterator(); // Returns an iterator over a set of elements of
                           type E.
}
```

Exercice 2 : (analyse de code - 5 points)

Indiquer ce qu'affiche l'exécution du programme suivant.

```
class A {
   public int num;
   private String s = "";
   protected A(int x) {
      this();
      num = x;
      System.out.println("A");
   }
   A() {
      num = 7;
      s = "OK-A";
      System.out.println("AA");
   }
```

```
   void m() {
      System.out.println(s + " : " + num);
      f(num);
      num++;
   }
   void f(int n) {
      System.out.println("A.f -> " + n);
      num = n + 15;
      s = s + n;
   }
}
class B extends A {
   public int num = 13;
   B(int y) {
      super(y + 3);
      System.out.println("B");
   }
   void m(int x) {
      System.out.println("B.m -> " + x);
      m();
      m();
   }
}
class C extends B {
   private A x = new A();
   C(int v) {
      super(v);
      System.out.println("C");
   }
   void f(int x) {
      super.f(-1);
      System.out.println("C.f -> " + x + " ; num = " + num);
      num = num + x;
   }
}
class Test {
   public static void main(String[] args) {
      C c = new C(21);
      c.m(6);
   }
}
```