Interrogation d'informatique
2ème année - 2018-2019

Durée totale : *1h*
Documents autorisés : *1 feuille recto/verso - téléphones portables interdits.*

Informations générales :
- Le barème est indicatif.
- Une présentation de mauvaise qualité sera pénalisée (jusqu'à -1 pt).
- Le sujet est sur 6 pages.

Exercice 1 Utilisation d'une liste Java (5 pts)

Dans une entreprise, chaque bureau de chaque bâtiment est équipé d'un capteur de température. La classe `Bureau` possède deux attributs "`public`" : le numéro du bureau et le numéro du bâtiment dans lequel le bureau se trouve. La signature du constructeur est : `public Bureau(int numBur, int numBat)`.

Cette classe comporte aussi les méthodes :
- `public String toString()`
- `public boolean enVie()` qui retourne `true` si le capteur de température du bureau fonctionne, et `false` sinon.

Important : on considère que cette classe existe déjà, vous n'avez pas à fournir son code. De plus, tous les `import` nécessaires ont été faits.

(Q1.1) Création (1 pt)

En utilisant la classe `LinkedList`, donnez le code permettant de :
- déclarer et créer une variable `lesBureaux` qui est une liste d'objets `Bureau` et
- d'ajouter à `lesBureaux` le bureau numéro 12 du bâtiment 2.

```
1  LinkedList<Bureau> lesBureaux = new LinkedList<Bureau>();
2  lesBureaux.add(new Bureau(12, 2));
```

(Q1.2) Parcours (2 pts)

On suppose que la liste `lesBureaux` contient au moins un bureau. Ecrire le code Java permettant d'afficher les bureaux où le capteur ne fonctionne pas, en utilisant une boucle de type "for-each".

```
1  for(Bureau b : lesBureaux)
2  {
3    if (!b.enVie())
4      System.out.println("Le capteur du bureau " + b.numero + " dans le batiment " +
           b.batiment + " n'est pas en vie.");
5  }
```

(Q1.3) Et avec `Arraylist` ? (2 pts)

Indiquer les modifications à apporter aux questions précédentes, si on souhaite maintenant utiliser la classe `ArrayList` pour la liste de bureaux. Justifiez rapidement.

Juste remplacer LinkedList par ArrayList. Ces classes relèvent de la même interface (la même classe mère).

Exercice 2 Liste chainée dynamique (4 pts)

(Q2.1) Programmation directe en Java (2 pts)

On considère maintenant le code suivant, qui programme une liste simplement chainée en Java.

```
1
2  public class BureauElement{
3    public int numero;
4    public int batiment;
5    public BureauElement next;
6
7    public BureauElement(int num, int bat) {
8      numero = num;
9      batiment = bat;
10     next = null;
11   }
12   public BureauElement getNext(){...}
13   public void setNext(BureauElement b){...}
14   //... tous les getters et setters voulus sont presents
15
16 }
17
18 public class ListeBureaux {
19   private BureauElement root;
20
21   public ListeBureaux(){
22     root = null;
23   }
24   ...
25 }
```

Travail à faire : Donnez le code la méthode `public void insererTete(BureauElement b)` de la classe `ListeBureaux` qui insère l'élément `b` au début de la liste.

```
1 public void insererTete(BureauElement b)
2 {
3   b.next = root;
4   root = b;
5 }
```

(Q2.2) Complexité (2 pt)

Donner la complexité de l'insertion en tête de liste pour les trois cas suivants :
- La liste est représentée à l'aide d'un tableau.
- La liste est représentée comme celle de la classe `ListeBureaux`.
- La liste est représentée par une `ArrayList`.

Expliquez rapidement.

tableau : O(n) LinkedList : O(1) ArrayList : O(n)

Exercice 3 Affichage dans les stations de métro (11 pts)

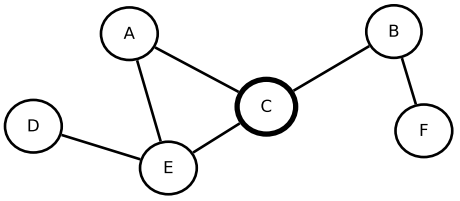
Nous modélisons le réseau de métro lyonnais de la manière suivante : une station est représentée par un nom (une chaine de caractères) et la liste de toutes les stations accessibles **directement** à partir d'elle (c'est-à-dire sans arrêt intermédiaire), toutes lignes confondues. Le réseau est alors représenté par la liste des stations. La classe `Station` est codée par :

```
1  public class Station {
2      public String nom;
3      public ArrayList<Station> listeStations;
4
5      public Station(String nom){
6          this.nom = nom;
7          listeStations = new ArrayList<Station>();
8      }
9
10     public String toString(){
11         return nom;
12     }
13     ...
14 }
```

On suppose que les méthodes `equals` et `compareTo` de la classe `Station` ont été écrites. De plus, pour toutes les questions, tous les `import` nécessaires ont été faits.

Pour informer les usagers lyonnais, Max doit placer une affiche dans chaque station de métro à partir de la station "Vieux Lyon". Il décide de réaliser un parcours dit "en largeur d'abord" en tenant compte des connexions directes entre stations. L'objectif de cet exercice est d'afficher le nom des stations dans l'ordre de ce parcours.

Un parcours en largeur à partir d'un noeud source, selon wikipedia, s'effectue ainsi : "on commence par explorer un nœud source, puis ses successeurs, puis les successeurs non explorés des successeurs, etc." Un parcours du graphe suivant en commençant par C serait par exemple : C - A - B - E - D - F ou encore C - E - B - A - D - F.



Dans le cas du réseau de métro, on désigne par S l'ensemble des stations de métro, par NT la liste des stations non encore traitées, T la liste des stations traitées et par *source* la station de laquelle on part. L'algorithme peut alors être décrit ainsi :

```
1 Initialiser NT avec {source};
2 Initialiser T avec ∅ ;
3 Tant que NT est non vide faire :
4   (1) Afficher le premier element s de la liste NT ;
5   (2) Supprimer s de la liste NT et le placer dans T;
6   (3) Calculer les stations directement accessibles depuis s qui n'appartiennent ni a
        NT ni a T et les ajouter a la fin de NT;
7 Fin du tant que
```

(Q3.1) Programmation de l'algorithme (8 pts)

Max a déjà écrit le code suivant (dans sa méthode `main`) :

```
1  ArrayList<Station> reseau = new ArrayList<Station>();
2  // ici le code initialisant le reseau (fait - ne pas le faire)
3  Station source = ... ; //init de la variable avec "Vieux Lyon" - ne pas le faire
4
5  LinkedList<Station> nonTraitees = new LinkedList<Station>();
6  LinkedList<Station> traitees = new LinkedList<Station>();
7
8  // initialiser nonTraitees - A COMPLETER
9
10 // boucle - A COMPLETER
```

Travail à faire : Compléter le code ci-dessus de façon à afficher les stations selon l'ordre de parcours en largeur.

Indication : pour une liste d'éléments de type `Station`, que ce soit une `ArrayList` ou une `LinkedList` :

- `Station remove(int pos)` : enlève de la liste la station située en position `pos` et la retourne.
- `boolean add(Station s)` : ajoute la station `s` à la fin de la liste.

```
1  nonTraitees.add(source);
2  while(!nonTraitees.isEmpty()){
3      Station s = nonTraitees.remove(0);
4      traitees.add(s);
5      System.out.print(s+ " ");
6      for(Station e : s.listeStation)
7           if(!nonTraitees.contains(e) && !traitees.contains(e))
8               nonTraitees.add(e);
9  }
```

(Q3.2) Efficacité (2 pts)

Le programme de la question (Q3.1) serait-il plus ou moins efficace si on changeait le type des variables `nonTraitees` et `traitees` par : `ArrayList<Station>` ? Justifiez.

moins efficace avec ArrayList car remove(0) est en O(1) pour une LinkedList et en O(n) pour une ArrayList. Mais d'autres réponses basées sur des connaissances plus approfondies sont possibles (notamment "LinkedList est doublement chainée et le add est en O(1)", ou, pour une réponse plus mitigée, la notion de complexité "amortie" pour le "add", ...)

(Q3.3) Et sans `equals` ni `compareTo` ? (1 pt)

Supposons que les méthodes `equals` ni `compareTo` n'aient pas été écrites. Votre programme fonctionnerait-il toujours ? Justifiez.

Noter surtout la cohérence entre la réponse et la justification. Accepter les deux types de réponses suivants car peu/pas de trace dans les diapos/tds de l'utilisation par défaut de l'égalité des pointeurs en cas d'absence de `equals` :
NON, car la méthode `contains` a besoin de `equals` pour fonctionner.
OUI, car les références /pointeurs/adresses des objets `Station` seraient utilisées lors de l'appel à `contains`.
(non demandé : à condition de ne pas avoir créé dans le programme deux fois avec `New Station(...)` une même station physique.)