Les seuls documents autorisés sont les documents « sur papier » ;
à l'exclusion de la copie ou les brouillons des voisins.

Consignes

- Lire très attentivement tout le sujet.
- Il est tout à fait possible de répondre à une question sans avoir répondu à toutes les questions précédentes et aussi d'utiliser les méthodes des questions précédentes même si on n'a pas donné leur code.
- À la fin du document, on rappelle quelques méthodes Java qui pourraient être utiles.
- Quand des consignes manquent de précision (par exemple le type de retour des méthodes), libre à vous de proposer la solution qui vous semble le mieux.
- On supposera que tous les messages circulant sur le réseau sont corrects (on ne vous demande pas de traiter les cas où un message ne respecte pas la spécification).

Contexte

On souhaite développer un système de multi-diffusion de messages où chaque entité recevant un message devra le transmettre à d'autres entités qui se seront enregistrées au préalable.

Toutes les communications seront en mode par paquet (il n'y aura donc pas de connexion).

Chaque entité, appelée émetteur, aura une identité (un entier positif strictement inférieur à 10000), un numéro de port et une liste d'émetteurs contenant pour chaque émetteur son adresse IP et un numéro de port. On supposera que chaque identité d'émetteur est unique.

Un émetteur qui souhaite s'enregistrer chez un autre émetteur lui enverra un message de la forme : R id num_port où id est une chaîne de quatre caractères contenant l'identité de l'émetteur souhaitant s'enregistrer et num_port est une chaîne de caractères contenant le port (un entier d'exactly 4 chiffres, par exemple l'entier 12 sera codé par 0012). Si un émetteur souhaite se désenregistrer il enverra un message de la forme D id (ici encore id est une chaîne de 4 caractères) et dans ce cas il sera désenregistrer pour tous ses ports (c'est à dire que si l'émetteur avec identité 1234 envoie les messages R 1234 4242 et R 1234 4243 à un autre émetteur, et si il envoie ensuite le message D 1234, alors il sera retiré de la liste pour les ports 4242 et 4243).

Les émetteurs peuvent recevoir des messages de la forme M count message où count est une chaîne de 2 caractères représentant un compteur de ré-émission de messages (pour éviter qu'un message tourne trop longtemps dans le système) et message est un message d'au plus 100 caractères. Lorsqu'un émetteur reçoit un message, il renvoie à tous les émetteurs qu'il a dans sa liste le message M countbis message où countbis est une chaîne de deux caractères représentant un entier dont la valeur correspond à celle de count diminuée de un. Si jamais la valeur de count est égale à 0, l'émetteur ne renvoie pas le message.

Questions

On considère que les informations d'un émetteur seront encodées dans une classe `EmetteurItem` qui aura les champs suivants :

```
class EmetteurItem{
    public int id;
    public int port;
```

```

public String host; //correspond a l'adresse IP ou au nom de la machine
                    //ou se trouve l'emetteur
}

```

1. Écrire un constructeur de la classe `EmetteurItem` qui prendra comme arguments deux entiers et une chaîne de caractères. On supposera ici que les entiers correspondant à `id` et `port` respectent la spécification (c'est à dire qu'ils sont tous deux strictement inférieurs à 10000).

On aura ensuite une classe `EmetteurListe` qui sera une liste d'objets de la classe `EmetteurListe`. Cette classe aura donc un champ un `public LinkedList<EmetteurItem> liste;`

2. Écrire le constructeur de la classe `EmetteurListe` en supposant qu'au début la liste est vide.
3. Écrire une méthode `ajoute` de la classe `EmetteurListe` qui ajoute à la liste un objet `EmetteurItem` donné en argument.
4. Écrire une méthode `retire` de la classe `EmetteurListe` qui prend en argument un entier et retire de la liste tous les objets `EmetteurItem` dont l'`id` correspond à l'entier donné argument.
5. Écrire une méthode `diffuse` de la classe `EmetteurListe` qui prend en arguments une chaîne de caractères message d'au plus 100 caractères et un entier `count` et envoie le message `M count message` à tous les émetteurs présents dans la liste.
6. Que faudrait-il faire si jamais on voulait rendre cette liste manipulable par des accès concurrents multi-threadés sans que cela ne pose problème?

On s'intéresse maintenant à la classe `Emetteur` qui contiendra le code principal de l'émetteur. Cette classe aura les champs suivants :

```

class Emetteur{
    public int id;
    public int port;
    public EmetteurListe eListe;
}

```

7. Écrire le constructeur de la classe `Emetteur` qui prendra comme argument un numéro de port et une identité. On supposera ici que les entiers correspondant à `id` et `port` respectent la spécification (c'est à dire qu'ils sont tous deux strictement inférieurs à 10000).
8. Écrire une méthode `nouveauPaquet` de la classe `Emetteur` qui prend en argument un `DatagramPacket` et selon le message qu'il contient (`R id num_port` ou `D id` ou `M countbis message`) ajoute l'émetteur du message à la liste `eListe` ou le retire ou envoie le message à tous les membres de la liste. Il faudra bien entendu utiliser les méthodes de la classe `EmetteurListe`.
9. Écrire une méthode `ecoute` de la classe `Emetteur` qui en boucle écoute sur son port et traite les messages reçus.
10. Écrire une méthode `encode` qui prend en argument un entier supposé strictement inférieur à 1000 et retourne la chaîne d'exactly 4 caractères correspondant à l'encodage de cet entier (pour 10 la chaîne retournée sera 0010).
11. Écrire une méthode `enregistre` de la classe `Emetteur` qui prend en arguments une chaîne de caractère correspondant à un nom de machine sur le réseau local et un entier correspondant à un numéro de port et qui envoie sur cette machine et sur ce port un message pour enregistrer l'émetteur appelant (c'est-à-dire un message de la forme `R id num_port` où `id` et `num_port` sont l'identité et le port de l'émetteur appelant la méthode `enregistre`).
12. Écrire dans une classe `Prog1` un programme principale qui crée un objet `Emetteur` (on pourra choisir le port et l'identifiant que l'on souhaite) et le lance pour qu'il traite les messages arrivant.

13. On suppose que le programme précédent s'exécute sur la machine `machine1` (du réseau local), écrire alors dans une classe `Prog2` un programme principal qui crée un objet `Emetteur` (on pourra choisir le port et l'identifiant que l'on souhaite), fait enregistrer cet émetteur auprès de l'émetteur tournant sur `machine1` et le lance ensuite pour qu'il traite les messages.
14. On suppose que le programme précédent tourne sur une machine qui s'appelle `machine2`. Écrire un programme `C` qui envoie à l'émetteur tournant sur `machine1` un message de la forme `M-10 Hello-World!`.
15. On souhaite observer le comportement des programmes précédents. Pour cela, on va écrire un programme `C` qui s'enregistre avec un port sur l'émetteur tournant sur `machine1` et avec un autre numéro de port sur l'émetteur tournant sur `machine2` et qui ensuite va écouter et afficher sur le terminal les messages envoyés par ces deux émetteurs. On demande de donner ce programme.

Quelques fonctions de Java pouvant être utiles

- Méthodes de la classe `LinkedList<E>`

```
boolean add(E e)

boolean remove(Object o)

ListIterator<E> listIterator()
```

- Méthodes de la classe `ListIterator<E>`

```
boolean hasNext()

E next()

void remove()
/*Removes from the list the last element that was returned by next()
or previous() (optional operation). This call can only be made once
per call to next or previous. It can be made only if add(E) has not
been called after the last call to next or previous.*/
```

- Méthodes de la classe `String`

```
int length()

String substring(int beginIndex, int endIndex)
```

- Méthodes de la classe `Integer`

```
Integer(Integer s)
/*Constructs a newly allocated Integer object that represents the int
value indicated by the String parameter.*/

int intValue()
```