

# Intergiciels

## Examen

Daniel Hagimont

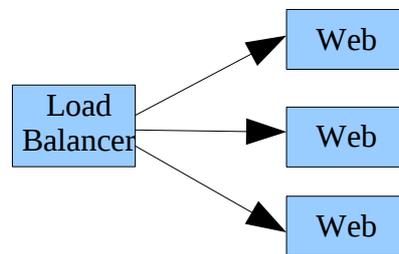
Durée: 1 heures 45 minutes, documents autorisés

*Lire l'ensemble des énoncés avant de commencer à répondre. La **clarté**, la **précision** et la **concision** des réponses, ainsi que leur **présentation matérielle**, seront des éléments importants d'appréciation. Donnez essentiellement les programmes Java demandés. Dans vos programmes, vous n'avez pas à programmer les imports et le traitement des exceptions.*

### PROBLÈME I (sockets)

Vous avez à réaliser avec des sockets en Java un répartiteur de charge (une classe *LoadBalancer*) qui reçoit des requêtes HTTP et les distribue de façon aléatoire vers un ensemble de serveurs Web. Le début de la classe *LoadBalancer* est le suivant :

```
public class LoadBalancer {
    static String hosts[] = {"host1", "host2"};
    static int ports[] = {8081,8082};
    static int nbHosts = 2;
    static Random rand = new Random();
    ...
}
```



A chaque réception d'une requête HTTP, *LoadBalancer* transfère la requête à un des serveurs web (les adresses de ces serveurs sont données par les tables *hosts* et *ports*) et *LoadBalancer* transfère le résultat de la requête à l'émetteur. Le choix du serveur Web est aléatoire (*rand.nextInt(nbHosts)* retourne un entier entre 0 et *nbHosts-1*). Pour être efficace, *LoadBalancer* est évidemment multi-threadé.

Pour la programmation des entrées/sorties, on utilisera :

- *InputStream*
  - *public int read(byte[] b); // bloquante, retourne le nombre de bytes lus*
- *OutputStream*
  - *public void write(byte[] b, int off, int len); // écrit les len bytes à la position off*

et on suppose que les requêtes et réponses sont lues ou écrites en un seul appel de méthode avec un buffer de 1024 bytes.

### PROBLÈME II (RMI)

Vous devez réaliser avec Java RMI une implantation d'un intergiciel de type JMS. Ce service est implémenté sous la forme d'un serveur RMI (sur une machine serveur) qui implante l'interface *MOM*, qui fournit 2 méthodes ayant les prototypes suivants :

```
public void publish(String topic, Message message) // publie un message sur un topic
public void subscribe(String topic, Callback cb) // s'abonne à un topic
```

L'abonnement (*subscribe*) à un topic prend en paramètre une référence à un objet d'interface *Callback* permettant le rappel du client pour lui transmettre un message publié sur le topic. L'interface *Callback* fournit la méthode de prototype :

```
public void onMessage(Message message)
```

Vous devez programmer ce service (interface *MOM* et classe *MOMImpl*) ainsi qu'un exemple de programme client démontrant l'utilisation du service lors d'une publication et d'un abonnement. Pour simplifier, je vous propose d'implanter une classe *MOMImpl* qui ne prend en compte qu'un unique topic de nom "topic".