Université Lille 1 Juin 2013
Master mention Informatique – M1
Module : Conception d'applications réparties (CAR)

# Examen – 2nde session

# Conception d'applications réparties (CAR)

Tous documents papier autorisés. Téléphones, calculatrices et ordinateurs interdits.
Le barème est donné à titre indicatif. 3 heures.
Ce sujet comporte 3 pages.

## 1 Questions de cours (3 points)

1.1 Pourquoi un appel de méthode semi-synchrone permet d'introduire de la concurrence de traitement entre un client et un serveur ?

1.2 Pourquoi dans la politique de synchronisation lecteurs/écrivain vue en cours, l'appel à la méthode Java `notify()` n'est pas suffisant à la fin d'une demande de lecture et pourquoi faut-il un appel à la méthode `notifyAll()` ?

1.3 Quel est le point commun entre SOAP et Java RMI ?

## 2 Méthode de rappel en RMI (4 points)

On considère deux objets RMI A et B. B fournit une méthode `mult` permettant de faire le produit de deux matrices X et Y de taille identique. Chaque matrice est un tableau bidimensionnel de réels double. On considère que la méthode `mult` effectue une itération sur le nombre de lignes de la matrice X (on ne demande pas de fournir plus de détails sur le code de la méthode `mult`). On souhaite qu'à chaque pas de l'itération, l'objet B fournisse à l'objet A l'indice de l'itération en invoquant une méthode `rappel` que vous définirez. A affichera l'indice reçu. Par ailleurs, A fournit une méthode `main` qui invoque la méthode `mult` en transmettant deux matrices X et Y que l'on suppose connues.

2.1 Proposer le code Java de l'interface `AItf` de l'objet RMI A ainsi que de la classe `AImpl` implantant cette interface. Même question pour l'interface `BItf` de l'objet RMI B et de sa classe d'implantation `BImpl`. (2 points)

2.2 Que se passerait-il si l'objet A n'était pas *multi-threadé* ? (1 point)

2.3 On souhaite que l'invocation de la méthode `rappel` soit asynchrone. Java RMI propose-t-il cette notion ? Si oui, comment ? Si non, proposez une solution de remplacement ? (1 point)

## 3 RMI et Multicast-IP (3 points)

3.1 Quel est le protocole de transport utilisé de façon standard par Java RMI pour les invocations de méthode ? (0,5 point)

On souhaite remplacer ce protocole par Multicast IP afin de faire en sorte que chaque invocation de méthode puisse atteindre, non pas un seul objet serveur RMI, mais plusieurs objets serveurs RMI. Pour cela, on propose de remplacer la classe `UnicastRemoteObject` par une nouvelle classe appelée `MulticastRemoteObject`.

3.2 Quel est le rôle de cette nouvelle classe `MulticastRemoteObject` ? Décrire en français son comportement. (1 point)

3.3 Quel est le rôle de l'outil `rmic` ? Quel est l'impact de classe `MulticastRemoteObject` sur `rmic` ? (1 point)

3.4 Quel est impact de cette classe `MulticastRemoteObject` sur l'annuaire `rmiregistry` ? (0,5 point)

## 4 Ping-pong (10 points)

Soient deux objets A et B. L'objet A envoie un message `PING` à l'objet B qui lui répond par un message `PONG`. La 1ère partie de l'exercice met en œuvre cet échange avec des *sockets* Java, la 2ème avec Java RMI et la 3ème avec REST.

### *Sockets* Java

Les messages échangés via les *sockets* ont la structure suivante :
- 1 octet pour identifier le message. L'octet vaut 0 pour `PING` et 1 pour `PONG`.
- 1 octet pour le nombre de paramètres associés au message.
- Les octets suivants contiennent les paramètres associés au message.
  Pour chaque paramètre, on trouve :
  - 1 octet pour son type. L'octet vaut 0 si le paramètre est un entier et 1 si le paramètre est une chaîne de caractères.
  - La valeur du paramètre.
    - Les entiers sont codés sur 2 octets. L'octet de poids fort est placé en tête.
    - Les chaînes de caractères sont précédées d'un octet qui fournit la taille de la chaîne. Chaque caractère occupe 1 octet.

Le message `PING` est associé à 5 paramètres. Chaque paramètre est un entier. Les 4 1ers correspondent à l'adresse IP de la machine qui envoie le message `PING` (par exemple 163.172.139.14). Le 5ème paramètre correspond au numéro de port TCP sur lequel la machine attend la réception d'un message `PONG`. Le message `PONG` est associé à un paramètre de type chaîne de caractères.

On suppose que la machine A connaît l'adresse IP et le port TCP sur laquelle la machine B reçoit les messages `PONG`. Soit l'échange suivant :
- la machine A envoie PING 163.172.139.14 2048 à la machine B,
- la machine B reçoit ce message, en extrait l'adresse IP et le port TCP et utilise ces informations pour envoyer à la machine A un message `PONG` avec la chaîne de caractères « Ok ».
- la machine A reçoit le message `PONG`.

4.1 Donner la séquence d'octets transmis par la machine A à la machine B. (1 point)

4.2 Donner la séquence d'octets transmis par la machine B à la machine A. À titre d'information, le code ASCII du caractère O est 79 et celui du caractère k est 107. (1 point)
4.3 Donner le code Java du programme qui s'exécute sur la machine A. (1,5 point)
4.4 Donner le code Java du programme qui s'exécute sur la machine B. (1,5 point)

## Java RMI

Les machines A et B communiquent maintenant via Java RMI. La machine A implémente l'interface `AItf` et la machine B implémente l'interface `BItf`.

4.5 Précédemment le message `PING` était associé à 5 paramètres (adresse IP + numéro de port TCP). Dans le contexte de Java RMI, par quoi suggérez-vous de remplacer ces paramètres ? (1 point)
4.6 Donner le code Java des interfaces `AItf` et `BItf`. (1 point)

On suppose que l'objet RMI s'exécutant sur la machine B est enregistré dans l'annuaire `rmiregistry` sous le nom « machineB ». On suppose que la machine A possède une méthode `main` qui démarre l'échange en invoquant la méthode `PING`.

4.7 Donner le code Java des classes `AImpl` et `BImpl` implémentant respectivement les interfaces `AItf` et `BItf`. La méthode `PONG` affiche à l'écran le message qu'elle reçoit. (2 points)

## REST

4.8 On suppose maintenant que les communications utilisent REST. Que proposez-vous pour le code des objets A et B ? (1 point)