Université Lille 1 Juin 2012
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 4 pages.

1 Question de cours (2 points)

1.1 Dans le cadre de Java RMI, que permet de faire le programme `rmiregistry` ?
1.2 Les invocations de méthode en Java RMI sont-elles synchrones, semi-synchrones ou asynchrones ? Justifier votre réponse.

2 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`.

2.1 Donner la séquence d'octets transmis par la machine A à la machine B. (1 point)
2.2 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)
2.3 Donner le code Java du programme qui s'exécute sur la machine A. (1,5 point)
2.4 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`.

2.5 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)
2.6 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`.

2.7 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

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

3 CORBA – Gestion d'un parking (8 points)

On considère un parking dont on souhaite automatiser la gestion en l'administrant à distance avec un *middleware* CORBA. Le parking peut accueillir au maximum 50 véhicules. Une barrière gère les entrées et les sorties de véhicules : elle délivre un ticket à l'entrée d'un véhicule et elle contrôle le ticket fourni par l'automobiliste à la sortie. Un automate de paiement permet sur présentation du ticket, de régler le montant correspondant au temps de stationnement. L'automobiliste règle le montant du stationnement à l'automate avant de franchir la barrière pour sortir. La barrière, l'automate et le parking sont représentés chacun par un objet CORBA.

L'interface `BarriereItf` de la barrière fournit deux méthodes :

- `entrer` : correspond à l'entrée d'un véhicule dans le parking. Retourne vrai si le véhicule peut rentrer (si place libre), faux sinon (et dans ce cas le véhicule ne peut pas entrer). Cette méthode fournit en paramètre de sortie une structure `Ticket` comprenant un numéro de ticket de type entier et une heure d'entrée sous la forme d'une chaîne de caractères.

- `sortir` : correspond à la sortie d'un véhicule. Retourne vrai si l'automobiliste a réglé le montant du stationnement, faux sinon (et dans ce cas le véhicule ne peut pas sortir). Cette méthode prend en paramètre d'entrée une structure `Ticket`.

L'interface `AutomateItf` de l'automate fournit deux méthodes :
- `payer` : correspond au paiement en fonction de la durée de stationnement. Retourne vrai si le montant fourni par l'automobiliste est suffisant, faux sinon. Cette méthode prend en paramètre d'entrée une structure `Ticket` et un montant de type réel double. Cette méthode fournit également en sortie une valeur de type réel double représentant la monnaie à rendre à l'utilisateur.
- `cestpaye` : à partir d'une structure `Ticket` fournie en entrée, retourne vrai si le montant du ticket a été réglé, faux sinon.

L'interface `ParkingItf` du parking fournit trois méthodes :
- `nbPlacesLibres` : retourne un entier indiquant le nombre de places libres dans le parking.
- `entreeVehicule` : signale l'entrée d'un véhicule sur le parking. Ne prend aucun paramètre, ne retourne rien.
- `sortieVehicule` : signale la sortie d'un véhicule du parking. Ne prend aucun paramètre, ne retourne rien.

3.1 Définir la structure `Ticket` et les interfaces IDL correspondant à ces trois objets CORBA dans un module `ParkingPkg`. (2 points)

Arrivée d'un véhicule

3.2 Lorsqu'un automobiliste se présente à la `Barrière`, l'objet CORBA barrière interroge l'objet `Parking` avant de laisser entrer l'automobiliste. Pourquoi ? Quelle méthode invoque-t-il ? (0,5 point)

3.3 À l'issu de cela, il se peut que l'objet `Barrière` invoque la méthode `entreeVehicule` de l'objet `Parking`. Cette méthode n'a pas de paramètres et a pourtant un rôle essentiel. Lequel ? (0,5 point)

Sortie d'un véhicule

3.4 Décrire le scénario de la sortie d'un véhicule du parking : quel(s) objet(s) appelle(nt) quelle(s) méthode(s) de quel(s) autre(s) objet(s) et pourquoi ? En particulier, on s'attachera à fournir un scénario de fonctionnement permettant à l'objet `Barrière` d'autoriser ou non la sortie du véhicule. (1 point)

Implémentation du parking

3.5 Ecrire le code de la classe `ParkingImpl` implémentant l'interface `ParkingItf`. (1 point)

Ajout d'une barrière

3.6 On souhaite ajouter une deuxième barrière d'entrée/sortie au parking. Cela modifie-t-il les interfaces définies à la question 1 ? Si oui comment, si non pourquoi. (0,5 point)

3.7 La classe `ParkingImpl` de la question 5 est modifiée suite à l'ajout d'une barrière. Après avoir rappelé les méthodes invoquées lors de l'arrivée d'un véhicule, expliquez en quoi consiste cette modification (on ne vous demande pas de la programmer). (1 point)

3.8 Même question pour la sortie d'un véhicule. (1 point)

Evolution du parking

3.9 On souhaite maintenant connaître les heures d'entrée et de sortie des véhicules. Sans donner le code, expliquez comment vous pourriez faire cela. (0,5 point)