Module SOR
Examen session 1, 21 mars 2016
Systèmes à objets répartis
Durée : 1h20 minutes
Tous documents autorisés

Singhoff Frank
singhoff@univ-brest.fr
C-203

NB : ce document contient 2 feuilles recto/verso.

Exercice n^o 1 : Questions de cours sur .NET (5 points)

- Question 1 : Avec .NET, quelles sont les différences entre un objet de type MarshalByRefObject et un objet de type Serializable.
- Question 2 : Avec .NET, à quel moment est instancié puis détruit un objet de type MarshalByRefObject ?
- Question 3 : En C#, comment passe-t-on un argument par référence ?
- Question 4 : Contrairement aux objets Serializable .NET ou Java RMI, un objet CORBA ne peut être déplacé sur le réseau. Pourquoi ?

Exercice n^o 2 : Mise en oeuvre d'un service afin de gérer une centrale de réservation hôtelière (8 points)

Dans cet exercice on souhaite implanter un serveur permettant à des clients de gèrer l'occupation des chambres d'un ensemble d'hôtels. L'interface de ce serveur est donnée en annexe.

Les services offerts par le serveur sont décrits par deux interfaces :

1. L'interface *centrale_reservation* constitue le point d'entrée de l'application. On suppose que le serveur instancie une seule fois un objet conforme à cette interface. Cette interface offre un service d'usine à objets CORBA : en effet, pour chaque hôtel, l'interface *centrale_reservation* devra être préalablement utilisée pour allouer un objet CORBA de type *hotel*.

   Cette interface IDL ne comporte qu'une seule méthode : la méthode *ajouter_hotel*. Cette méthode permet d'ajouter un hôtel à la centrale de réservation. Chaque hôtel possède un nom unique. L'exception *dejaExistant* doit être levée par la méthode *ajouter_hotel* lorsqu'un client demande la création d'un hôtel déjà enregistré : le nom de l'hôtel peut être utilisé pour ce contrôle.

2. L'interface *hotel* permet à un client de réserver une chambre dans un hôtel donné. Un objet CORBA de type *hotel* est associé à chaque hôtel géré par la centrale de réservation.

   La méthode *ajouter_reservation* permet d'ajouter une réservation. Une réservation est décrite par la structure *reservation* et est composée du type de service demandé pour la chambre, du nom du client ainsi que du nombre de personne. Lors de son enregistrement, le serveur associe un numéro unique à la réservation. Ce numéro unique (argument *id* des méthodes *annuler_reservation* et *ajouter_reservation*) permet d'annuler ultérieuement une réservation avec la méthode *annuler_reservation*.

   Enfin, l'interface *hotel* permet de déterminer à tout instant le nombre de chambres disponibles avec la méthode *calculer_nombre_chambre_disponible* : l'argument *nb_disponible* permet de renvoyer au client cette information. Pour calculer cette information, il est nécessaire d'utiliser le nombre total de chambre de l'hôtel qui est mémorisé dans l'attribut *nombre_chambre* et d'y retrancher le nombre de réservation.

Question 1 :
Expliquez comment les types IDL *type_service*, *reservation* et *reservations* seront traduit en Java.

Question 2 :
Donnez les deux classes Java qui implantent les interfaces IDL *hotel* et *centrale_reservation*.

Annexe : l'interface IDL du serveur

```
module centrale {

      enum type_service {petit_dejeuner, pension_complete, demi_pension};

      struct reservation {
        type_service  repas;
        string   nom_client;
        long     nb_personne;
      };

      typedef sequence <reservation> reservations;

      interface hotel {
         readonly attribute string nom_hotel;
         attribute long nombre_chambre;
         readonly attribute reservations toutes_les_reservations;

         void ajouter_reservation(inout long id, in reservation r);
         void annuler_reservation(in long id);

         void calculer_nombre_chambre_disponible(inout long nb_disponible);
      };


      exception deja_existant{};

      interface centrale_reservation {
         hotel ajouter_hotel(in string nom_hotel) raises (deja_existant);
      };


};
```