Examen

Classe : LFI3	**Année universitaire : 2020-2021**
Matière : Systèmes Répartis	**Date : Mardi 12 Janvier 2021 à 11h**
Enseignantes : Wafa BEN SLAMA &Mariem ZAOUALI	**Session : Principale, 1er semestre**
Documents autorisés : □Oui ☑Non	**Durée : 1h30 N.pages :3**

Exercice n°1 : QCM

1. RMI est l'acronyme de :
   a. Remote Method Invocation
   b. Remote Multiplatform Invoker
   c. Random Method Interceptor
2. Le protocole de transport utilisé par défaut par RMI est :
   a. JRMP (Java Remote Method Protocol) construit au-dessus de TCP.
   b. JRMP (Java Remote Multiplatform Protocol) construit au-dessus de UDP.
   c. IIOP (Interoperable Interface of Objects Protocol) construit au-dessus de TCP.
   d. IIOP (Interoperable Interface of Objects Protocol) construit au-dessus de UDP.
3. Les clients d'un même objet RMI distant ont l'impression de disposer de l'objet en local
   a. **Un stub** est un objet coté client qui gère l'encodage et le désencodage des données.
   b. **Un stub** est un objet coté serveur qui gère l'encodage et le désencodage des données lors de la réception d'un appel à un objet RMI.
   c. **Un skeleton** est un objet coté client qui gère l'encodage et le désencodage des données.
   d. **Un skeleton** est un objet coté serveur qui gère l'encodage et le désencodage des données lors de la réception d'un appel à un objet RMI.
4. Le rôle du Factory en JMS est
   a. Création d'un objet
   b. Création d'une connexion entre client et serveur
   c. Création d'une connexion entre producteur et consommateur
   d. Création d'un système de nommage
5. Pour lancer l'interface de gestion de activeMq, on a besoin d'écrire une commande qui est :
   a. **activemq.bat start**
   b. **start activemq.bat**
   c. **start activemq**
   d. **start activeMq**

Exercice n°2 : JMS

Le code suivant permet de créer un exemple de Queue entre un producteur et un consommateur. Remplir les lignes de codes qui manquent (total de lignes à remplir : 5 lignes).

```
public class MessageReceiver {

        private static String url = ActiveMQConnection.DEFAULT_BROKER_URL;

    private static String subject = "LFI_QUEUE";
```

```
    public static void main(String[] args) throws JMSException {

        ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(url);

        // Creating de la session
        Connection connection = connectionFactory.createConnection();
        connection.start();


        Session session = connection.createSession(false,
                Session.AUTO_ACKNOWLEDGE);


        Destination destination =

        // MessageConsumer is used for receiving (consuming) messages
        MessageConsumer consumer =

        // Here we receive the message.
        Message message = consumer.receive();

        // We will be using TestMessage in our example. MessageProducer sent us a
TextMessage
        // so we must cast to it to get access to its .getText() method.
        if (message instanceof TextMessage) {
            TextMessage textMessage = (TextMessage) message;
            System.out.println("Received message '" + textMessage.getText() + "'");
        }

    }
}
```

```
public class MessageSender {


    private static String url = ActiveMQConnection.DEFAULT_BROKER_URL;

    private static String subject = "LFI_QUEUE";

    public static void main(String[] args) throws JMSException {
        // Getting JMS connection from the server and starting it

        Connection connection = connectionFactory.createConnection();
        connection.start();


        Session session = connection.createSession(false,
                Session.AUTO_ACKNOWLEDGE);


        Destination destination = session.createQueue(subject);


        MessageProducer producer = session.createProducer(destination);
```

```
        TextMessage message = session
                .createTextMessage("Hello !!! Welcome to the world of ActiveMQ.");


        producer.send(message);

        System.out.println("LFI printing@@ '" + message.getText() + "'");
        connection.close();
    }
}
```

2. A quoi sert le paramètre du code suivant :

```
Session.AUTO_ACKNOWLEDGE);
```

4.Que fait ce programme ?

3.Quel est le mode d'envoi utilisé dans cet exemple ?

Exercice n°3 : RMI

Dans cet exercice, on se propose de réaliser une plateforme de films et de séries comme Netflix. Il s'agit d'une application répartie qui permet à un client de consulter la liste des films et de séries disponibles, de choisir l'item (film ou série) de son choix, puis de le visualiser. Pour ce faire, on a besoin d'un registry qui enregistrera un objet distribué (**GestionnaireFilmSerie)** que le client va consommer.

1. Ecrire une interface « distante » **InterfaceFilmSerie** qui définit deux méthodes et qui étend **java.rmi.Remote :**
    a. String consulter () : Cette méthode permet d'afficher la liste des films et de séries sur la plateforme.
    b. String visualiser(int entier) : Cette méthode permet de passer en paramètre le ID du film ou de série et de retourner un message « lancé » si la réservation est réussie, « échec » sinon.
2. Ecrire la classe **GestionnaireFilmSerie** qui réalise les tâches de l'interface précédente et qui étend la classe **java.rmi.server.UnicastRemoteObject.** Dans cette classe, ajoutez un ArrayList comme variable static et globale contenant tous les films/séries que le serveur propose.
3. Ecrire une classe **Enregistrement** qui possède une méthode main qui enregistre l'objet **GestionnaireFilmSerie** dans le registre.
4. Dans le client, créer une classe qui contient le main(). Obtenez une référence de l'objet distant et simulez la consultation et la visualisation d'un film en écrivant le code qui permet de réaliser ces étapes.

Bon travail ☺ !