**Master Informatique – 1re année**
**INF 2165 – Introduction aux systèmes distribués**
**2nde session – 9 juin 2021 – 10 h 00**
**Responsable : Y. Mahéo**

**3 pages (+ Annexes 4 pages) – Durée : 2 h 00**

**Avertissements :** *Le barème de notation indiqué ci-dessous est purement indicatif et pourrait être révisé lors de la correction. La notation tiendra compte de l'esprit de pédagogie et de synthèse dont vous ferez preuve (soyez clairs et concis tout en faisant des phrases !), mais aussi de la présentation générale de la copie. Il est évident qu'une simple restitution du cours ne rapportera que très peu de points.*

# 1 Java Networking (6 points)

**a)** Nous avons écrit un programme Java client/serveur TCP : le programme serveur attend la connexion d'un client sur un port dont le numéro lui est passé en paramètre, et le programme client ouvre une connexion avec un serveur dont l'adresse IP et le numéro de port lui sont passés en paramètres **(important : pour les questions qui suivent, identifiez précisément le problème rencontré. Une réponse listant toutes les causes possibles à tout type de problème sera considérée comme fausse)**.

1. Au démarrage du serveur, une exception est levée, nous indiquant que l'ouverture de la socket est impossible. Expliquez quelles peuvent en être les causes.
2. Après avoir corrigé le problème précédent et avoir lancé le serveur sans problème, on lance le client. Cette fois, c'est le client qui lève une exception nous indiquant que la connexion est impossible. Expliquez quelles peuvent en être les causes.

**b)** On veut écrire une application pair à pair permettant d'échanger des banques d'images entre utilisateurs : chaque utilisateur possède des images, qu'il veut partager avec d'autres. À chaque image est associée une description : catégorie de l'image, date de l'image, taille du fichier, empreinte MD5 de l'image (séquence de 32 octets calculée à partir du contenu du fichier, qu'on peut considérer comme unique, c'est-à-dire que la probabilité que deux fichiers différents donnent deux empreintes différentes est très forte). Proposez une architecture et un protocole permettant d'implémenter cette application : entités mises en œuvre, séquence des messages échangés et format des messages, protocoles utilisés. Vous pouvez vous aider de schémas pour décrire votre solution.

# 2 Intergiciels à messages (7 points)

La société « Labbeli » fabrique des chaussures et possède un certain nombre de magasins franchisés assurant la distribution de ses produits. Un logiciel est développé en Java pour fournir aux magasins revendeurs des informations relatives à la stratégie de vente qu'ils doivent appliquer. Ce logiciel s'appuie sur JORAM, une implantation de JMS, pour le transfert des informations : lorsqu'il a choisi une stratégie, le fabricant communique à chaque revendeur une liste de produits à mettre en vitrine. Chaque produit est identifié par son code article (connu du fabricant et des revendeurs).

Le développeur du logiciel a prévu de mettre en place un ensemble de files JMS (une file entre le fabricant et chacun des revendeurs, file qui porte le nom du revendeur). L'ensemble consitué par les revendeurs n'est pas très stable (il est relativement fréquent de voir un revendeur disparaître ou de voir un nouveau revendeur adopter la franchise). Un module est donc présent dans le logiciel pour que le fabricant puisse créer ou détruire facilement des files.

Le code Java suivant décrit le code exécuté par le fabricant pour un envoi au revendeur de Vannes d'une liste de produits à mettre en vitrine. Notez que le code qui est donné ici n'est pas le code réel mais une version simplifiée (moins générique), ceci afin de faire apparaître ce qui est effectivement exécuté.

```
Context ctx = new InitialContext();
ConnectionFactory connectionFactory = (QueueConnectionFactory) ctx.lookup("gyar");
Queue q = (Queue) ctx.lookup("Vannes");
Connection connexion = connectionFactory.createQueueConnection();
Session session = connexion.createQueueSession(false, session.AUTO_ACKNOWLEDGE);
QueueSender qs = session.createSender(q);
HashSet<Integer> vitrine = getVitrine("Vannes");
ObjectMessage m = session.createObjectMessage();
m.setObject(vitrine);
qs.send(m);
```

**a)** Expliquez à quoi servent les deux premières lignes de ce code et les traitements qu'elles recouvrent.

**b)** Donnez le code Java exécuté par le revendeur de Vannes pour recevoir les informations données dans cet extrait de code. On appliquera un mode de réception asynchrone (mode PUSH). Le code affichera à l'écran un texte selon le modèle suivant :

```
Liste des articles à mettre en vitrine :
Sandale Tolan
Botte Tamasi
Botte Arad
```

On pourra utiliser la fonction `String nomArticle(int code)` qui rend le nom d'un article en fonction de son code.

**c)** Expliquez précisement le sens du paramètre `session.AUTO_ACKNOWLEDGE` dans le code ci-dessus.

**d)** On étend le logiciel pour que la stratégie ne se limite pas à une simple liste des produits à mettre en vitrine. Dorénavant, le fabricant est susceptible d'envoyer à tout moment soit une liste de produits à mettre en vitrine, soit une liste de produits qu'il faut solder, soit une liste de produits pour lesquels il faut installer un présentoir particulier en magasin. Expliquez comment vous aménagez le code exécuté chez le fabricant et le code exécuté chez le revendeur pour prendre en compte cette nouvelle façon de faire ?

**e)** Au bout d'un certain temps, on s'aperçoit que les stratégies que l'on indique aux revendeurs sont quasiment identiques, même si le code avait été initialement prévu pour spécialiser la stratégie à la situation de chaque revendeur (i.e. une stratégie propre était communiquée à chaque revendeur). Le fabricant décide donc de simplifier la procédure : une seule stratégie sera fournie à l'ensemble des revendeurs. Expliquez (sans donner de code) comment réviser l'application pour tenir compte de cette nouvelle approche ?

**f)** La société « Oltozet », qui fabrique des vêtements, appartient au même groupe que Labbeli. Oltozet a la même organisation que Labelli, avec un ensemble de magasins franchisés. Oltozet acquiert le logiciel développé pour Labelli afin de mettre en œuvre sa stratégie de vente, qu'elle a l'intention de déployer côté fabricant et côté magasin. Mais à la différence de Labelli, elle souhaite s'appuyer sur OpenJMS. Dans le cadre de ce déploiement par Oltozet, le logiciel va-t-il fonctionner sans modifications ? Justifiez votre réponse.

## 3 Mise en œuvre d'une politique at-most-once (2 points)

Vous avez à réaliser un serveur qui reçoit des requêtes identifiées par un numéro unique et qui contiennent des opérations à exécuter qui ne sont pas idempotentes. Expliquez comment mettre en œuvre une politique « at-most-once » dans ce serveur.

## 4 Ferme de calcul en Java RMI (5 points)

On cherche à mettre en place un service de calcul sur une machine très puissante qui soit accessible à un nombre quelconque de postes de travail. Le principe de réalisation retenu est une architecture de type client-serveur avec exécution à distance de tâches programmées en Java sur le serveur à l'aide de JAVA RMI (voir figure 1).

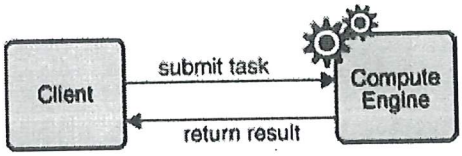


FIGURE 1 – Architecture client-serveur de la ferme de calcul

Les programmes sources Java de la première version fournie en annexe constitue la base de l'implémentation que vous êtes chargés de compléter progressivement en répondant aux questions suivantes. À noter que le code fourni contient une application de calcul de Pi en précision multiple pour tester le service ; le contenu exact de ce calcul n'est pas explicité (méthode `PiTask.computePi()`) car sans importance pour répondre aux questions.

1. Identifiez parmi les fichiers de la première version fournie :
   (a) quelle classe contient le code du client ?
   (b) quelle classe contient le code du serveur ?
   (c) quelle classe contient le code du servant ?
2. Complétez le code de la méthode `ComputeTask.executeTask()`
3. On désire maintenant que le client n'ait pas à attendre le résultat d'un calcul à distance avant de poursuivre son exécution. On modifie l'interface d'un calcul distant en ajoutant à la méthode `Compute.executeTask()` le paramètre de type `Result` qui permettra au serveur d'invoquer à distance la méthode de callback `Result.deliver()` chez le client. Le programme source Java de la seconde version fournie en annexe contient cette nouvelle version de la classe `Compute`.
   Quelles sont les autres modifications à apporter aux classes ou interfaces pour pouvoir exécuter le calcul de Pi à distance et afficher (par un `System.out.println`) son résultat chez le client ? Récrivez uniquement les parties de code en Java qui sont nécessaires en précisant où elles doivent être insérées.
4. Il reste maintenant à modifier le service de calcul pour se donner la possibilité de lancer plusieurs calculs en parallèle sur le serveur, c-a-d. avoir plusieurs instances simultanées du servant susceptibles d'exécuter une tâche. Décrivez les modifications qu'il va falloir apporter au code existant pour finaliser le service de calcul ; n'écrivez que le code des nouvelles interfaces, pas leur implémentation.

Master M1 INFO

—

# UE INF2165

Introduction aux systèmes distribués

—

# Annexes du sujet d'examen sur Java RMI

## Compute Farm Version 1

```
1  // Compute Farm version 1
2  // fichier  Compute.java
3
4  package compute;
5
6  import java.rmi.Remote;
7  import java.rmi.RemoteException;
8
9  public interface Compute extends Remote {
10
11     static String rmi_name= "compute";
12     static int rmi_port = 1099;
13     static String rmi_host = "localhost";
14
15     Object executeTask(Task t) throws RemoteException;
16 }
```

```
1  // Compute Farm version 1
2  // fichier  Task.java
3  package compute;
4
5  public interface Task {
6      Object execute();
7  }
```

```
1  // Compute Farm version 1
2  // fichier  ComputeEngine.java
3
4  package compute;
```

```
5
6  import java.rmi.registry.LocateRegistry;
7  import java.rmi.registry.Registry;
8
9  public class ComputeEngine {
10
11     public static void main(String[] args) {
12         if (System.getSecurityManager() == null) {
13           System.setSecurityManager(new SecurityManager());
14         }
15         try {
16           LocateRegistry.createRegistry(Compute.rmi_port);
17         } catch(Exception e) {
18           System.err.println("RMIregistry already started");
19         }
20         try {
21             Compute stub = new ComputeTask();
22             Registry registry = LocateRegistry.getRegistry();
23             registry.rebind(Compute.rmi_name, stub);
24         } catch (Exception e) {
25             System.err.println("ComputeEngine exception:");
26             e.printStackTrace();
27         }
28     }
29 }
```

---

```
1  // Compute Farm version 1
2  // fichier ComputePi.java
3
4  package compute;
5
6  import java.math.BigDecimal;
7  import java.rmi.Naming;
8
9  public class ComputePi {
10
11     public static void main(String args[]) {
12
13         try {
14           Compute compute = (Compute) Naming.lookup(
15             "rmi://"+Compute.rmi_host+":"+Compute.rmi_port+"/"+Compute.rmi_name);
16           PiTask task = new PiTask(1000);
17           BigDecimal pi = (BigDecimal) compute.executeTask(task);
18           System.out.println(pi);
19         } catch (Exception e) {
20             System.err.println("ComputePi exception:");
21             e.printStackTrace();
22         }
23     }
```

```
24 }
```

```
1  // Compute Farm version 1
2  // fichier ComputeTask.java
3
4  package compute;
5
6  import java.rmi.RemoteException;
7  import java.rmi.server.UnicastRemoteObject;
8
9  public class ComputeTask extends UnicastRemoteObject implements Compute {
10
11     protected ComputeTask() throws RemoteException {
12       super();
13   }
14
15     public Object executeTask(Task t) {
16         // a completer (question 2.)
17     }
18
19 }
```

```
1  // Compute Farm version 1
2  // fichier PiTask.java
3  package compute;
4
5  import java.io.Serializable ;
6  import java.math.BigDecimal;
7
8  public class PiTask implements Task, Serializable {
9
10     /** digits of precision after the decimal point */
11     private final int digits ;
12
13     /**
14      * Construct a task to calculate pi to the specified
15      * precision.
16      */
17     public PiTask(int digits) {
18         this.digits = digits;
19     }
20
21     /**
22      * Calculate pi.
23      */
24     public BigDecimal execute() {
25         return computePi(digits);
26     }
27
```

```
28      /**
29       * Compute the value of pi to the specified number of
30       * digits after the decimal point.
31       */
32      public static BigDecimal computePi(int digits) {
33          // code non explicite pour reduire la taille du code
34      }
35  }
```

## Compute Farm Version 2

```
1   // Compute Farm version 2
2   // fichier Compute.java
3
4   package compute;
5
6   import java.rmi.Remote;
7   import java.rmi.RemoteException;
8
9   public interface Compute extends Remote {
10
11      static String rmi_name= "compute";
12      static int rmi_port = 1099;
13      static String rmi_host = "localhost";
14
15      void executeTask(Task t, Result r) throws RemoteException;
16  }
```