Some models/ideas of former Exams

Here you find some questions that were given in previous exams of this course, which aim to evaluate your understanding of the learnt concepts. Consider that documents are not allowed ! You also find a pdf file, with one example of an RMI application that you need to run on paper !

1. RMI impose que toute méthode d'une interface Remote (et donc son code d'implémentation) déclare jeter éventuellement une exception « java.rmi.RemoteException ». Quelle implication cela a-t-il sur le code qui invoque des méthodes RMI ? Il y a surement de bonnes raisons à imposer ceci, pourquoi à votre avis ?
2. RMIRegistry est lui-même une application Java RMI ; détailler ce qui en fait un serveur RMI (cad. quels sont les programmes qui sont clients de ce serveur ?). Argumentez pourquoi RMIRegistry n'est pas lui-même client d'autres objets RMI
3. Le téléchargement dynamique de .class est un mécanisme puissant de la plateforme Java RMI. Rappeler brièvement quels en sont les principes
4. *Suite de 3.* Rappeler dans quelle circonstance RMIRegistry peut être amené à utiliser ce mécanisme de téléchargement de classes
5. Expliquer brièvement la différence entre utiliser une queue JMS en Point-à-Point ou en Publish/Subscribe. Est-ce que cela change quelque chose du côté de celui qui produit les messages.
6. Expliquer comment en JMS réaliser une application distribuée et concurrente, qui se base sur un modèle Producteur-Consommateur, avec la possibilité qu'il y ait P producteurs et C consommateurs, P et C >=1
7. Comment un Message-Oriented Middleware tel ActiveMQ (basé donc sur un principe de queue) fait-il pour assurer que tout message produit sera effectivement bien traité par ce(ux) qui le doivent. Et ce même dans les pires situations où n'importe quel élément participant pourrait tomber en panne.

Un exercice plus long

L'objectif de cet exercice est de spécifier (mais pas de programmer en détail, juste de donner les principes d'architecture, puis de « coder » dans les grandes lignes) une application permettant à des producteurs de produire un message, de sorte qu'ensuite chacun de ces messages soit récupéré de manière explicite (puis ultérieurement traité) par exactement un consommateur (n'importe lequel). Tout en permettant que plusieurs consommateurs puissent se partager ainsi la tâche de consommer l'ensemble des messages (il n'est pas question ici de faire en sorte qu'un même message soit consommé par tous les consommateurs). Afin de gérer le cas où il n'y a rien à consommer, vous supposerez que la demande de consommation est bloquante : elle ne renvoie le message à traiter que lorsque il y a effectivement un message, et bloque la demande du consommateur tant qu'il n'y en a pas.

1. Spécifier dans les grandes lignes le principe que vous allez adopter (en y incluant le choix pour gérer les cas où il n'y a rien à consommer). Attention, cette description doit être indépendante du type de technologie utilisable (ce sera abordé aux questions 2 et 3)

2. Puis décrire dans les grandes lignes comment réaliser ceci en JMS. Vous aurez deux types de clients JMS : un type de client capable de jouer le rôle de producteur. Un autre type de client capable de jouer le rôle de consommateur.

3. Ensuite, décrire dans les grandes lignes comment réaliser ceci en Java RMI. Comme pour la version basée sur JMS, vous aurez deux types de programmes RMI. Qu'en est-il de la partie serveur RMI ? A quoi sert-elle ? Comment la spécifier ?

4. Conclure et synthétiser afin de faire ressortir les grandes différences ou similarités entre la solution exhibée en 2 et celle donnée en 3