Gestion à distance de comptes bancaires

Version Client/Serveur Sockets

Objectifs : Mise en oeuvre des sockets en mode connecté et en mode non connecté.

Révision : Programmation multi-threadée - Gestion des exceptions - Manipulation des flux d'entrée-sortie.

Note : On pourra utiliser les supports de cours sur les Sockets et les Threads.

But du TD/TP : écrire une application répartie permettant de gérer des comptes bancaires.

Le serveur devra implanter les requêtes suivantes :

1. CREATION *id somme_initiale* : permet de demander la création d'un compte identifié par *id* sur lequel sera placé la *somme_initiale*.
2. POSITION *id* : permet d'obtenir la position courante du compte identifié par *id*.
3. AJOUT *id somme* : ajoute une somme sur le compte identifié par *id*.
4. RETRAIT *id somme* : retire une somme sur le compte identifié par *id*.

Le client recevra les réponses du serveur sous la forme suivante :

1. OK *commande* : informe le client que la commande s'est correctement déroulée.
2. ERREUR *raison* : la raison de l'échec de la commande sous forme de chaîne de caractères.
3. POS *solde date_dernière_opération* : envoi au client le solde actuel du compte et la date de la dernière opération.

   Les différents éléments constitutifs (commandes paramètres) seront séparés par des espaces.

   NB : la date de dernière opération sera envoyée sous la forme : Thu Nov 07 14:24:11 CET 2013 en utilisant la méthode toString() de la classe java.util.Date.

Travail demandé

1. A partir des classes fournies ci-dessous et de la description précédente du protocole applicatif, écrire une classe de gestion du protocole.
2. Implantation dans un environnement connecté :
   a. Écrire le serveur et le client de cette application en utilisant les sockets TCP.
   b. Modifier le serveur afin de prendre en compte le traitement simultané de plusieurs clients.
3. Implantation dans un environnement non-connecté :
   Écrire le serveur et le client de cette application en utilisant les sockets UDP.

4. Réalisation d'une application multi-protocoles. Créer un nouveau "Serveur" permettant de traiter les clients indifféremment en mode connecté ou en mode non-connecté.

```
import java.util.Date;
import java.util.HashMap;

public class BanqueSimple {
   class CompteEnBanque {
         private double solde;
         private Date derniereOperation;

         public CompteEnBanque(double solde) {
               this.solde = solde;
               derniereOperation = new Date(); // recupere la date courante
         }

         public double getSolde() {
               return solde;
         }

         public Date getDerniereOperation() {
               return derniereOperation;
         }

         public void ajouter(double somme) {
               solde += somme;
               derniereOperation = new Date(); // recupere la date courante
         }

         public void retirer(double somme) {
               solde -= somme;
               derniereOperation = new Date(); // recupere la date courante
         }
   }

   HashMap<String, CompteEnBanque> comptes;

   public BanqueSimple() {
         comptes = new HashMap<String, CompteEnBanque>();
   }

   public void creerCompte(String id, double somme) {
         comptes.put(id, new CompteEnBanque(somme));
   }

   public void ajouter(String id, double somme) {
         CompteEnBanque cpt = comptes.get(id);
         cpt.ajouter(somme);
   }

   public void retirer(String id, double somme) {
         CompteEnBanque cpt = comptes.get(id);
         cpt.retirer(somme);
```

```
    }

    public double getSolde(String id) {
           CompteEnBanque cpt = comptes.get(id);
           return cpt.getSolde();
    }

    public Date getDerniereOperation(String id) {
           CompteEnBanque cpt = comptes.get(id);
           return cpt.getDerniereOperation();
    }

    public boolean compteExiste(String id) {
           return comptes.containsKey(id);
    }

    public static void main(String[] args) {
           BanqueSimple s = new BanqueSimple();
           s.creerCompte("ABC1234", 1000);
           s.ajouter("ABC1234", 100);
           s.retirer("ABC1234", 30);
           double solde = s.getSolde("ABC1234");
           Date date = s.getDerniereOperation("ABC1234");
           System.out.println("ABC1234 -> " + solde + " " + date);
    }

}
```