TraderServer.java • Classe principale du serveur et fonction *main*

```
import java.lang.*;
import java.io.*;
import java.net.Socket;
import java.net.ServerSocket;


class TraderServer
{
  private Market marche;
  private Holders utilisateurs;
  ServerSocket socketEcoute;
  int numport;

  public static final String FICHIER_ACTIONS="actions.cfg";
  public static final String FICHIER_HOLDERS="users.cfg";

  public TraderServer(int port)
  {
    marche=new Market(FICHIER_ACTIONS);
    utilisateurs=new Holders(FICHIER_HOLDERS, marche);
    numport=port;

    try
    {
      socketEcoute=new ServerSocket(numport);
    }
    catch(IOException e)
    {
      System.out.println("Impossible de creer la socket d'ecoute...");
    }
  }

  // Boucle d'ecoute du serveur :
  public void communication()
  {
    Socket socket;
    TransactionThread transac;
    boolean stopServeur=false;

    while(!stopServeur)
    {
      socket=null;
      try
      {
        socket=socketEcoute.accept();
      }
      catch(IOException e)
      {
        System.out.println("Erreur de jonction du serveur");
      }

      // Gestion du nouveau client :
      transac=new TransactionThread(this, socket);
      transac.start();
    }
  }

  // Transactions :
  public boolean achat(String user, String action, int quantite)
  {
    // Un utilisateur doit etre logge :
    if(user==null) return false;

    try
    {
      utilisateurs.doTransaction(user, action, quantite);
      sauvegarde();
      return true;
    }
    catch(Holders.ActionNotFoundException e)
    { // System.out.println("action non trouvee !");
    }
    catch(Holders.NotEnoughStocksException e)
    { // System.out.println("pas assez de stocks !");
    }
    catch(Holders.NotATransactionException e)
    { // System.out.println("transaction erronee !");
    }
    catch(Holders.HolderException e)
    { // System.out.println("autre erreur !");
    }
    return false;
  }

  // Creation d'un nouvel utilisateur : attention, n'est
  // pas loggé automatiquement.
  public boolean addUser(String nom)
  {
    if(nom==null) return false;

    try
    {
      utilisateurs.addShareHolder(nom);
      sauvegarde();
      return true;
    }
    catch(Holders.UserAlreadyExistsException e)
    {}
    return false;
  }

  // Sauvegarde toutes les modifications :
  public void sauvegarde()
  {
    marche.save(FICHIER_ACTIONS);
    utilisateurs.save(FICHIER_HOLDERS);
  }



  // Login d'un utilisateur :
  public boolean logUser(String nom)
  {
    if(nom==null) return false;

    try
    {
      utilisateurs.getShareHolder(nom);
      return true;
    }
    catch(Holders.HolderNotFoundException e)
    {

    }
    return false;
  }


  // L'utilisateur est-il connecte en ce moment ?
  public boolean isRegisteredUser(String nom)
  {
    if(nom==null) return false;

    try
    {
      return utilisateurs.getShareHolder(nom).estConnecte();
    }
    catch(Holders.HolderNotFoundException e)
    {}
    return false;
  }


  // Indique que l'utilisateur se connecte :
  public void registerUser(String nom)
  {
    if(nom==null) return;
    try
    {utilisateurs.getShareHolder(nom).setConnecte(true);}
    catch(Holders.HolderNotFoundException e) {}
  }


  // Precise que l'utilisateur se deconnecte :
  public void unregisterUser(String nom)
  {
    if(nom==null) return;
    try
    {utilisateurs.getShareHolder(nom).setConnecte(false);}
    catch(Holders.HolderNotFoundException e) {}
  }


  // Liste le portefeuille du marche ou d'un utilisateur :
  public String liste(String nom, boolean user)
  {
    // Un utilisateur doit etre logge :
    if(nom==null) return null;

    try
    {
      if(user==false) return marche.toString();
      else return utilisateurs.getShareHolder(nom).toString();
    }
    catch(Holders.HolderNotFoundException e) {}
    return null;
  }



  // Demarrage du serveur :
  public static void main(String args[])
  {
    TraderServer serveur=new TraderServer(6681);

    // Pour tests :
    //serveur.marche.afficheActions();
    //serveur.utilisateurs.afficheHolders();

    // Entre en phase d'attente des clients :
    System.out.println("Serveur pret.");
    serveur.communication();
  }

};
```

ActionValue.java • Une action élémentaire : son nom et sa quantité

```
import java.lang.*;
import java.io.*;


class ActionValue implements Cloneable, Serializable
{
  private String intitule;
  private int valeur;

  public ActionValue()
  {
    intitule=new String("");
    valeur=0;
  }

  public ActionValue(String intitule, int valeur)
  {
    this.intitule=intitule;
    this.valeur=valeur;
  }

  public synchronized String getIntitule() {return intitule;}
  public synchronized int getValeur() {return valeur;}

  public synchronized void setIntitule(String intitule) {this.intitule=intitule;}
  public synchronized void setValeur(int valeur) {this.valeur=valeur;}

  public synchronized int addValeur(int valeur) {return this.valeur+=valeur;}

  public synchronized String toString()
  {
    String res;
    res=intitule.concat(" ");
    res=res.concat(String.valueOf(valeur));
    //res=res.concat(" actions");
    return res;
  }

  // Surdefinition des fonctions classiques de Object
  public synchronized Object clone()
  {
    return new ActionValue(intitule.substring(0), valeur);
  }
};
```

Market.java • Classe du marché (ensemble d'actions associées à leur quantité)

```
import java.lang.*;
import java.io.*;
import java.util.Vector;
import java.util.Iterator;
import java.util.StringTokenizer;


class Market
{
  protected int nbActions;
  protected Vector tActions;


  // Constructeur par defaut - aucune action :
  public Market()
  {
    nbActions=0;
    tActions=new Vector();
  }

  // Constructeur a partir d'un fichier d'initialisation :
  public Market(String nomFichier)
  {
    String ligne=new String();
    String intitule;
    int quantite;
    StringTokenizer st;
    tActions=new Vector();

    // Ouvre le fichier :
    FileReader lecteur;
    BufferedReader fichier;
    try
    {
      lecteur=new FileReader(nomFichier);
      fichier=new BufferedReader(lecteur);

      // Lecture du nombre d'actions :
      ligne=fichier.readLine();
      if(ligne==null) nbActions=0;
      else nbActions=Integer.valueOf(ligne).intValue();

      int i=0;
      while((ligne=fichier.readLine())!=null && i<nbActions)
      {
        // Recupere l'intitule et la quantite de l'action consideree :
        st=new StringTokenizer(ligne);
        if(st.hasMoreTokens())
        {
          intitule=st.nextToken();
          if(!st.hasMoreTokens()) quantite=0;
          else quantite=Integer.valueOf(st.nextToken()).intValue();

          // Ajoute l'action dans le portefeuille...
          tActions.addElement(new ActionValue(intitule, quantite));
        }
        i++;
      }
    }
    catch(FileNotFoundException except)
    {
      nbActions=0;
    }
    catch(IOException except)
    {
      nbActions=0;
    }
  }


  // Construit un Market a partir d'un autre Market,
  // mais reinitialise les
  // nombres d'action :
  public void derive(Market reference)
  {
    // Clonage :
    nbActions=reference.nbActions;

    // CLONAGE A LA MAIN CAR M. JAVA NE VEUT PAS CLONER
    // MES ACTIONVALUE !!!
    // AU LIEU DE CA QUI DEVRAIT MARCHER !!! :
    // tActions=(Vector)(reference.tActions.clone());
    tActions=new Vector();
    Iterator itclone=reference.tActions.iterator();
    while(itclone.hasNext())
    {
      ActionValue av=(ActionValue)itclone.next();
      tActions.addElement(av.clone());
    }

    // Initialise a zero les quantites :
    Iterator it=tActions.iterator();
    ActionValue action;
    while(it.hasNext())
    {
      action=(ActionValue)it.next();
      action.setValeur(0);
    }
  }


  // Renvoit le nombre d'actions enregistrees :
  public int getNbActions()
  {
    return nbActions;
  }


  // Renvoit le nombre d'actions non nulles :
  public int getNbNonNullActions()
  {
    Iterator it=tActions.iterator();
    String res=new String("");
    ActionValue action;
    int i=0;
    while(it.hasNext())
    {
      action=(ActionValue)it.next();
      if(action.getValeur()>0) i++;
    }
    return i;

  }
```

```
  // Deboguage : affiche les actions stockees :
  public void afficheActions()
  {
    if(nbActions==0) System.out.println("Aucune action disponible.");

    synchronized(tActions)
    {
      Iterator it=tActions.iterator();
      ActionValue action;
      int i=0;
      while(it.hasNext())
      {
        action=(ActionValue)it.next();
        i++;
        System.out.println(i + ". " + action);
      }
    }
  }


  public String toString()
  {
    if(nbActions==0) return new String("\n");

    synchronized(tActions)
    {
      Iterator it=tActions.iterator();
      String res=new String("");
      ActionValue action;
      int i=0;
      while(it.hasNext())
      {
        action=(ActionValue)it.next();
        i++;
        if(action.getValeur()>0)
        {
          res=res+action.toString();
          res=res+"\n";
        }
      }
      return res;
    }
  }



  public String toString2()
  {
    if(nbActions==0) return new String("\n");

    synchronized(tActions)
    {
      Iterator it=tActions.iterator();
      String res=new String("");
      ActionValue action;
      int i=0;
      while(it.hasNext())
      {
        action=(ActionValue)it.next();
        i++;
        // On renvoie meme les actions dont la quantite est nulle :
        res=res+action.toString();
        res=res+"\n";
      }
      return res;
    }
  }


  // Renvoie l'action a partir de son nom :
  public ActionValue getActionValue(String intitule)
  {
    synchronized(tActions)
    {
      Iterator it=tActions.iterator();
      ActionValue action;
      while(it.hasNext())
      {
        action=(ActionValue)it.next();
        if(intitule.equalsIgnoreCase(action.getIntitule()))
          return action;
      }
      return null;
    }
  }

  // Definit la valeur de l'action demandee, renvoie faux si impossible
  public boolean setValeur(String intitule, int valeur)
  {
    ActionValue action=getActionValue(intitule);
    if(action==null) return false;

    action.setValeur(valeur);
    return true;
  }


  // Sauvegarde sur le disque du marche :
  public void save(String nomFichier)
  {
    try
    {
      FileWriter sauveur=new FileWriter(nomFichier);
      BufferedWriter fichier=new BufferedWriter(sauveur);
      PrintWriter sortie=new PrintWriter(fichier);

      sortie.println(String.valueOf(nbActions));
      sortie.print(toString2());
      sortie.println("");
      sortie.flush();
      fichier.close();
    }
    catch(IOException e)
    {}
  }
};
```

ShareHolder.java • Utilisateur possédant un ensemble d'actions

```
import java.lang.*;
import java.io.*;
import java.util.Vector;
import java.util.Iterator;
import java.util.StringTokenizer;

class ShareHolder extends Market
{
  private String nom;
  private boolean connecte;

  // Constructeur a partir d'un marche de reference :
  public ShareHolder(String nom, Market base)
  {
    super();
    derive(base);
    this.nom=nom;
    connecte=false;
  }

  // Constructeur par defaut :
  public ShareHolder()
  {
    super();
    nom=new String("");
    connecte=false;
  }


  // Renvoie le nom de la personne :
  public String getNom() {return nom;}


  // Gestion multi-connexion :
  public boolean estConnecte() {return connecte;}
  public void setConnecte(boolean flag) {connecte=flag;}
}
```

Holders.java • Ensemble des utilisateurs

```
import java.lang.*;
import java.io.*;
import java.util.Vector;
import java.util.Iterator;
import java.util.StringTokenizer;


class Holders
{
  public class HolderException extends Exception {};
  public class HolderNotFoundException extends HolderException {};
  public class ActionNotFoundException extends HolderException {};
  public class NotEnoughStocksException extends HolderException {};
  public class NotATransactionException extends HolderException {};
  public class UserAlreadyExistsException extends HolderException {};


  private int nbHolders;
  private Vector tHolders;
  Market marche;

  public Holders(String nomFichier, Market marche)
  {
    this.marche=marche;
    String ligne=new String();
    String intitule;
    int i,j,nactions,quantite;
    StringTokenizer st, st2;
    ShareHolder holder;
```

```
  tHolders=new Vector();

  // Ouvre le fichier :
  FileReader lecteur;
  BufferedReader fichier;
  try
  {
    lecteur=new FileReader(nomFichier);
    fichier=new BufferedReader(lecteur);

    // Lecture du nombre d'actions :
    ligne=fichier.readLine();
    if(ligne==null) nbHolders=0;
    else nbHolders=Integer.valueOf(ligne).intValue();

    i=0;
    while((ligne=fichier.readLine())!=null && i<nbHolders)
    {
      //System.out.println(ligne);
      // Recupere le nom du ShareHolder ainsi que son nombre d'actions :
      st=new StringTokenizer(ligne);
      intitule=st.nextToken();
      i++;
      if(!st.hasMoreTokens()) nactions=-1;
      else nactions=Integer.valueOf(st.nextToken()).intValue();

      if(nactions>=0)
      {
        // Cree un nouvel utilisateur :
        holder=new ShareHolder(intitule, marche);

        j=0;
        while(j<nactions && (ligne=fichier.readLine())!=null)
        {
          // Parcourt et ajoute les actions possedees par ce ShareHolder :
          st2=new StringTokenizer(ligne);
          intitule=st2.nextToken();
          j++;
          if(!st2.hasMoreTokens()) quantite=0;
          else quantite=Integer.valueOf(st2.nextToken()).intValue();
          holder.setValeur(intitule, quantite);
        }

        // Ajoute le ShareHolder dans la liste des utilisateurs :
        tHolders.addElement(holder);
      }
    }
  }
  catch(FileNotFoundException except)
  {
    nbHolders=0;
  }
  catch(IOException except)
  {
    nbHolders=0;
  }
}


// Pour tests :
public void afficheHolders()
{
  synchronized(tHolders)
  {
    if(nbHolders==0) System.out.println("Aucun utilisateur trouve.");

    Iterator it=tHolders.iterator();
    ShareHolder mister;
    int i=0;
    while(it.hasNext())
    {
      mister=(ShareHolder)it.next();
      i++;
      System.out.println(i + ". " + mister.getNom() + " (" + mister + ")");
    }
  }
}


// Teste si la transaction est possible et l'effectue :
public void doTransaction(String holder, String action, int quantite) throws HolderException
{
  // Recherche le holder :
  ShareHolder mister=getShareHolder(holder);

  synchronized(mister)
  {
    // Cette transaction est toujours possible :
    if(quantite==0) throw new NotATransactionException();

    // Recupere l'action demandee :
    ActionValue holderAction=mister.getActionValue(action);
    ActionValue marketAction=marche.getActionValue(action);
    if(marketAction==null || holderAction==null) throw new ActionNotFoundException();

    if(quantite>0)
    {
      // C'est un achat :
      if(quantite>marketAction.getValeur()) throw new NotEnoughStocksException();
    }
    else
    {
      // C'est une vente :
      if(-quantite>holderAction.getValeur()) throw new NotEnoughStocksException();
    }

    // Effectue la transaction securisee :
    holderAction.addValeur(quantite);
    marketAction.addValeur(-quantite);
  }
}
```

```
 // Recherche un ShareHolder par son nom :
  public ShareHolder getShareHolder(String nom) throws HolderNotFoundException
  {
    synchronized(tHolders)
    {
      Iterator it=tHolders.iterator();
      ShareHolder mister;
      while(it.hasNext())
      {
        mister=(ShareHolder)it.next();
        if(nom.equalsIgnoreCase(mister.getNom())) return mister;
      }
      throw new HolderNotFoundException();
    }
  }


  // Ajout d'un nouveau ShareHolder :
  public void addShareHolder(String nom) throws UserAlreadyExistsException
  {
    // Verification que l'utilisateur n'est pas deja present :
    try
    {
      getShareHolder(nom);
      throw new UserAlreadyExistsException();
    }
    catch(HolderNotFoundException e) {}

    // Ajout proprement dit :
    synchronized(tHolders)
    {
      tHolders.addElement(new ShareHolder(nom, marche));
      nbHolders++;
    }
  }


  // Sauvegarde sur le disque des utilisateurs :
  public void save(String nomFichier)
  {
    try
    {
      FileWriter sauveur=new FileWriter(nomFichier);
      BufferedWriter fichier=new BufferedWriter(sauveur);
      PrintWriter sortie=new PrintWriter(fichier);

      synchronized(tHolders)
      {
        sortie.println(String.valueOf(nbHolders));
        for(int i=0; i<nbHolders; i++)
        {
          ShareHolder holder=(ShareHolder)tHolders.elementAt(i);
          sortie.println(holder.getNom() + " " + holder.getNbNonNullActions());
          sortie.print(holder.toString());
        }
      }
      sortie.println("");
      sortie.flush();
      fichier.close();
    }
    catch(IOException e)
    {}
  }
};
```

TransactionThread.java • Thread du serveur prenant en charge un client individuellement

```
import java.lang.*;
import java.io.*;
import java.net.Socket;
import java.util.StringTokenizer;


class TransactionThread extends Thread
{
  private TraderServer serveur;
  private Socket socket;
  private boolean doitSortir;
  private final String STR_OK="OK ";
  private final String STR_ERR="ERR ";

  String loggedUser;


  public TransactionThread(TraderServer serveur, Socket socket)
  {
    super();
    loggedUser=null;
    this.serveur=serveur;
    this.socket=socket;
    doitSortir=false;
  }


  public void run()
  {
    BufferedReader reader=null;
    PrintWriter writer=null;

    System.out.println("Session initiee.");

    try
    {
      reader=new BufferedReader(new InputStreamReader(socket.getInputStream()));
      writer = new PrintWriter( socket.getOutputStream(),true);
      while(!doitSortir)
      {
        String line = reader.readLine();
        if(line!=null)
        {
          // Analyse de la commande et reponse au client :
          writer.println(requete(line));
          writer.flush();
        }
        else
```

```
        {
          System.out.println("Session interrompue.");
          doitSortir=true;
        }
      }
    }
    catch (IOException e)
    {
      System.out.println("Erreur de transmission.");
    }

    if(loggedUser!=null) serveur.unregisterUser(loggedUser);
  }


  // Traite une commande et renvoie la reponse :
  public String requete(String commande)
  {
    String demande=commande.toLowerCase();

    // COMMANDE : Ajout d'un utilisateur :
    if(demande.startsWith("add user "))
    {
      String nom=demande.substring(9);
      if(serveur.addUser(nom))
      {
        return STR_OK + "L'utilisateur " + nom + " vient d'etre cree avec un portefeuille vide.";
      }
      else
      {
        return STR_ERR + "Impossible de cree l'utilisateur " + nom + ".";
      }
    }

    // COMMANDE : Login d'un utilisateur :
    if(demande.startsWith("login "))
    {
      if(loggedUser!=null) {serveur.unregisterUser(loggedUser); loggedUser=null;}
      String nom=demande.substring(6);
      if(serveur.logUser(nom))
      {
        if(serveur.isRegisteredUser(nom)) return STR_ERR + "Utilisateur " + nom + " deja connecte...";
        loggedUser=nom;
        serveur.registerUser(nom);
        System.out.println(nom + " authentifie.");
        return STR_OK + "Utilisateur " + nom + " authentifie.";
      }
      else
      {
        return STR_ERR + "Utilisateur " + nom + " inconnu !";
      }
    }

    // COMMANDE : Liste le portefeuille du marche :
    if(demande.equals("list market"))
    {
      String rep=serveur.liste(loggedUser,false);
      if(rep==null) return STR_ERR + "Vous devez etre connecte !";
      else return rep;
    }


    // COMMANDE : Liste le portefeuille de l'utilisateur courant :
    if(demande.equals("list"))
    {
      String rep=serveur.liste(loggedUser,true);
      if(rep==null) return STR_ERR + "Vous devez etre connecte !";
      else return rep;
    }


    // COMMANDE : Achat et vente d'une action :
    if(demande.startsWith("buy ") || demande.startsWith("sell "))
    {
      StringTokenizer st=new StringTokenizer(demande.substring(4));
      String stockid=null;
      int qte=0;

      // Recupere les arguments :
      if(st.hasMoreTokens()) stockid=st.nextToken();
      if(st.hasMoreTokens()) qte=Integer.valueOf(st.nextToken()).intValue();

      if(demande.startsWith("sell ")) qte=-qte;

      if(serveur.achat(loggedUser, stockid, qte))
      {
        return STR_OK + "Transaction effectuee.";
      }
      else
      {
        return STR_ERR + "Transaction impossible.";
      }
    }


    // COMMANDE : Fin de session :
    if(demande.equals("logout"))
    {
      doitSortir=true;
      if(loggedUser!=null) serveur.unregisterUser(loggedUser);
      loggedUser=null;
      System.out.println("Session terminee.");
      return STR_OK + "Au revoir.";
    }


    // Si on est ici, la commande n'a pas ete comprise :
    return STR_ERR + "Commande inconnue.";
  }
}
```