Projet Programmation Java Avancée

Université Paris-Dauphine

Michael GOLETTO
Salim GUENNOUNI

Département MIDO
Master 1 Informatiques des Organisations
Parcours MIAGE 2011/2012
UE : Programmation Java Avancée
Enseignant : Flavien Balbo

Sommaire :

Introduction

Nous étudions dans ce projet la réalisation de threads, la communication par socket et les mécanismes de synchronisation à travers un modèle Client/Serveur.

Le but est de réaliser un service communautaire de partage d'informations pour une application touristique « Tourisme 2.0 » qui propose un ensemble d'activités dans les agglomérations. Ces dernières peuvent être notées et commentées par les utilisateurs. Il s'agit donc de gérer la cohérence des informations fournies.

L'application serveur devra permettre l'authentification de plusieurs utilisateurs simultanément, ou chaque utilisateur est un thread qui interagit avec le service à travers le protocole TCP. Ce dernier peut aussi s'abonner à un ensemble d'activités dont il choisit de recevoir des alertes à tout moment durant sa session de connexion, ceci est effectué avec le protocole UDP.

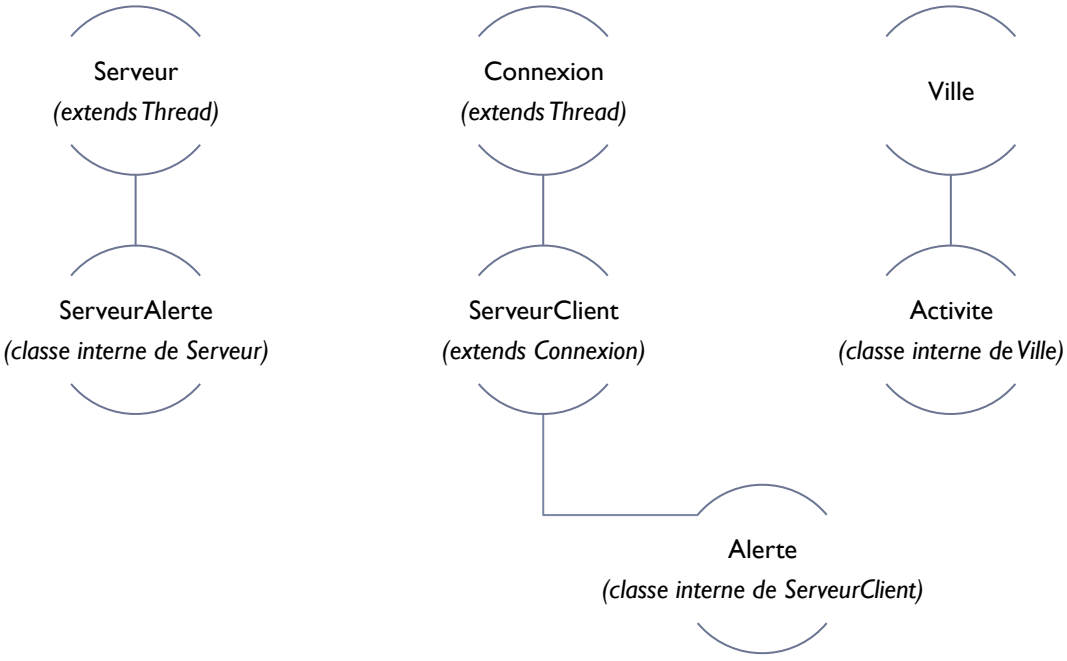
« Tourisme 2.0 » est un très bon exemple de réseautage social, puisque le projet existe réellement à travers une entreprise américaine Yelp, celle-ci édite le site internet Yelp.com (et application iPhone, Android, BlackBerry, etc.). Ce dernier est un outil simple pour lire, publier des avis et discuter des tendances dans votre ville. En juin 2008, plus de 10,8 millions de personnes ont consulté ce site web.

Modélisation

Serveur *(Package Serveur)*

L'application serveur a été répartie en plusieurs classes :

- Serveur *(Serveur.java)*, qui représente la classe principale. Elle permet d'instancier un socket d'écoute qui répond à chaque demande de connexion Client, puis elle délocalise le traitement de chaque client dans un thread spécifique. Elle prend en charge aussi le démarrage d'un Serveur d'Alertes, celui-ci crée un thread Alerte pour chaque client dont la connexion a été établie.
  → ServeurAlerte *(Serveur.ServeurAlerte.java)* est une classe interne de Serveur
- Connexion *(Connexion.java),* est la classe ou le traitement est externalisé une fois que chaque client se connecte. Elle permet de gérer et centraliser toute instruction relative à la communication avec le client, essentiellement les différentes IHM qu'il devra suivre pour profiter du service. Elle implémente les différents mécanismes de dialogue qui concernent la sélection d'une ville, d'une activité, etc. Elle gère les échanges TCP/UDP.
- ServeurClient *(ServeurClient.java)*, elle représente l'objet Client du côté serveur dont quelques informations sont retenues comme son adresse IP ou les différentes activités dont il est abonné (Alertes). En ce qui concerne ces dernières, les alertes, elles sont représentées par une classe interne Alerte *(ServeurClient.Alerte.java)*.
- Ville *(Ville.java),* est une classe qui modélise simplement une ville. Elle contient une classe interne Activite *(Ville.Activite.java)* qui représente l'atome principal du service, elle sera en effet notée et commentée par les utilisateurs.

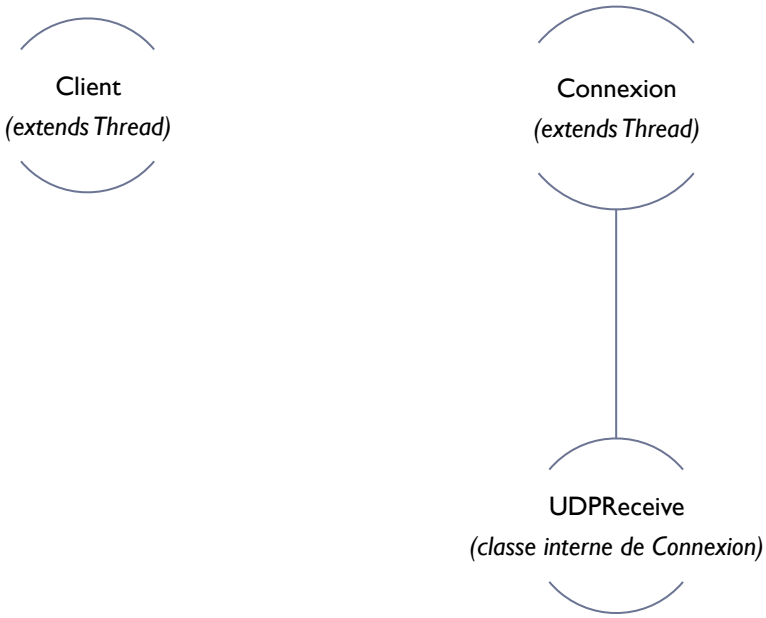


Client *(Package Client)*

L'application client a été elle aussi répartie en différentes classes afin d'assurer un bon dialogue avec le serveur :

- Client *(Client.java),* est une classe qui crée un seul et unique client (thread) par exécution du main, elle permet de donner un comportement aléatoire au client, c'est-à-dire un automate capable de répondre automatiquement aux messages TCP du serveur. Plusieurs clients peuvent être instanciés à partir de cette classe, ou chacun réagit comme un électron libre. Les instructions spécifiques relatives aux échanges TCP/UDP sont gérées dans une classe externe Connexion.
- Connexion *(Connexion.java)*, gère simplement les envois et réceptions TCP, ainsi que la lecture des alertes UDP à travers une classe interne UDPReceive *(Connexion.UDPReceive.java)*. Cette dernière

crée un thread qui permet de surveiller les messages UDP reçus sur serveur sur port spécifique communiqué par le serveur.



Explication technique

Gestion des clients

La gestion de clients commence dans la classe Serveur, puisque celle-ci mémorise dans une liste FIFO *(ArrayList)* l'ensemble des clients connectés :

```
public class Serveur extends Thread {
  protected static ArrayList<Ville> villes;
  private static ArrayList<ServeurClient> clients;
  private static final int port = 8080;
  private ServerSocket srv;
```

Quelques vérifications sont faites au niveau de la méthode *communication()* de la classe Connexion, elles permettent de savoir si le client a été connecté auparavant, dans ce cas la son instance est récupérée, sinon un nouvel objet *ServeurClient* est créé et ajouté à la liste *clients*.

D'autres informations Client sont aussi mémorisées :

```
public class ServeurClient extends Connexion {
  private String nomClient;
  private InetAddress adresseIP;
  private int portClient;
  private ArrayList<Alerte> alertes = new ArrayList<Alerte>();
```

- Le nom du client
- Son adresse IP
- Le dernier port qui lui a été octroyé par le serveur
- Une liste qui tient ses abonnements (les alertes)

La liste *clients* mémorise tout client connecté, même ci se dernier se déconnecte du service. En effet une deuxième liste est gérée au niveau du serveur d'alertes qui permet de retenir seulement les clients en ligne.

Gestion du dialogue avec le client

La classe *Connexion* du *package Serveur* se charge de tout échange TCP/UDP. A chaque demande de connexion au serveur, une instance de cette classe est crée pour proposer le service au client :

```
Serveur :
  public void run() {
        try {
              Connexion c;
              ServeurAlerte srvAlerte = new ServeurAlerte();
              srvAlerte.start();
              while(true) {
                    c = new Connexion(srv.accept());
                    c.start();
                    srvAlerte.add(c);
              }
        } catch (IOException e) {
              e.printStackTrace();
        }
  }
```

Elle tient un ensemble de propriétés pour assurer son fonctionnement :

```
public class Connexion extends Thread {
  private Socket s;
  private DatagramSocket socketUDP;
  private PrintWriter out;
  private BufferedReader in;
  private DatagramPacket bufferUDP;
  private static int port = 8181;
  private String userName;
  protected ServeurClient clientCourant;
```

- Un numéro de port spécifique est attribué à chaque client qui se connecte, il lui est envoyé en utilisant la méthode *sendPort(int port)*. Le client doit configurer son socket d'écoute UDP sur ce port afin de recevoir ses alertes. Ce numéro est unique par client, il commence à 8181, et est incrémenté à chaque nouvelle connexion d'un client.

La méthode principale de dialogue avec client est *communication() :*

```
Connexion :
  public void run() {
        openConnection();
        communication();
        closeConnection();
  }
```

Elle commence par récupérer le nom du client, ensuite lui propose les services du serveur :

```
Saisissez votre nom d'utilisateur :
Salim
Le port est : 8181
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
```

En fonction de la réponse du client, elle appelle la bonne méthode pour donner son résultat :

```
Saisissez votre nom d'utilisateur :
Salim
Le port est : 8181
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
```

```
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
1
Selectionnez la ville parmi :
1. Paris
2. Nantes
1
Selectionnez votre activite parmi :
1. Nom : Natation    Debut  : Thu Jan 05 20:00:00 CET 2012    Fin : Sun
Feb 05 22:00:00 CET 2012    Type : Sport    Note : -1.0
2. Nom : Coiffure    Debut  : Sun Feb 05 10:00:00 CET 2012    Fin : Sun
Feb 05 12:00:00 CET 2012    Type : Bien-etre    Note : -1.0
3. Nom : Restaurant    Debut  : Mon Mar 05 20:00:00 CET 2012    Fin : Sun
Feb 05 22:00:00 CET 2012    Type : Restauration    Note : -1.0
4. Nom : Soin visage    Debut  : Thu Apr 05 18:00:00 CEST 2012    Fin :
Sun Feb 05 20:00:00 CET 2012Type : Bien-etre    Note : -1.0
1
Que souhaitez vous faire.
1. Ajouter ou modifier une note.
2. Ajouter ou modifier un commentaire.
3. Ajouter ou modifier une note et un commentaire.
4. Ajouter une alerte.
5. Voir toutes les appréciations.
```

Le client répond simplement par un chiffre correspondant au choix désiré. Ce mécanisme nous a permit de simplifier l'implémentation du « *client automatique » ; automate de réponse automatique ;* puisqu'on peut générer des chiffres aléatoirement compris dans le champ de réponse.

Cette classe implémente aussi diverses méthodes pour assurer la communication avec le client, elles permettent de récupérer le nom d'utilisateur du client, son adresse IP, son choix de ville ou d'activité ou encore ses commentaires ou ses notes saisies :

- nomUtilisateur()
- getIP()
- selectionnerVille()
- selectionnerActivite(Ville)
- listeVille()
- listeActivite(Ville)
- verifierNote(float)
- ajouterNote(Ville, Activite, Float)
- ajouterCommentaire(Ville, Activite, String)
- dejaCommente(Ville, Activite)
- dejaNote(Ville, Activite)
- noterCommenter(Ville, Activite)
- commenter(Ville, Activite)
- noter(Ville, Activite)
- saisieNote(String)
- ajouterAlerte(Activite)

Toutes ses méthodes sont indépendantes pour chaque client, leur cohérence est garantie grâce à des *synchronized* à des points spécifiques.

Gestion des villes et des activités

Une ville est représentée par une classe Ville, elle contient un ensemble d'activités qui lui sont affectées :

```
public class Ville {
  private String nom;
  private ArrayList<Activite> activites;
```

Chaque activité est modélisée comme ci-dessous :

```
public class Activite {

	private String nom;
	private Calendar debut;
	private Calendar fin;
	private String type;
	private String informations;
	private HashMap <String, String> commentaires;
	private HashMap <String, Float> notes;
```

Les commentaires relatifs à une activité sont représentés dans une *HashMap* ou la clé est le nom d'utilisateur du client, et la valeur son commentaire. Il en est de même pour les notes. Cette structure nous permet de faciliter la modification d'une note ou d'un commentaire on y accédant directement par sa clé.

Gestion des alertes

Une alerte est représentée par la classe interne *ServeurClient.Alerte.java* :

```
public class Alerte {

	private Ville.Activite activite;
	private float noteMin = 0;
	private float noteMax = 20;
	private boolean notifiee = false;
```

Chaque alerte fait référence à un objet Activite spécifique, il est utilisé dans la méthode *alerteActivee()* qui permet de savoir si l'instance en cours Alerte est activé :

```
	public synchronized boolean alerteActivee() {
		float moy = activite.getMoyenne();
		if (moy > noteMin && moy < noteMax ){
			return true;
		}
		return false;
	}
```

L'objet Activite nous sert donc à récupérer la moyenne des notes données et à la comparer à la note minimale ou maximale définie par l'alerte. Dans le cas d'une véracité vraie, l'alerte est activée et ensuite notifié à l'utilisateur une seule et unique fois. Nous allons détailler ce processus.

Pour un client *ServeurClient,* la méthode *alerte**s**Activee**s**()* permet de récupérer une liste des alertes activées en utilisant la méthode ci-dessus, elle parcourt l'ensemble de ses abonnements *alertes : ArrayList<Alerte>* et vérifie si chacune est vraie.

```
public ArrayList<Alerte> alertesActivees() {
	// retourne la liste des alertes activées
	ArrayList<Alerte> list = new ArrayList<Alerte>();
	synchronized (alertes) {
		for(Alerte e : alertes) {
			if(e.alerteActivee() && !e.isNotifiee())
list.add(e);
		}
	}
	return list;
 }
```

Cette méthode est ensuite appelée au niveau du thread *ServeurAlerte* qui vérifie pour chaque client connecté si une ou plusieurs de ses alertes ont été activées :

```
public class ServeurAlerte extends Thread {
	private ArrayList<Connexion> clientsConnectes;
```

Cette liste de clients connectés est tenue à jour dans le *run()* du *ServeurAlerte :*

```
        public void run() {
                ArrayList<Connexion> clientsDeconnectes = new
ArrayList<Connexion>();
                while(true) {
                        synchronized(clientsConnectes) {
                                for(Connexion e : clientsConnectes){
                                        if(e.isClosed()) {
                                                clientsDeconnectes.add(e);
                                        } else {
                                                ServeurClient cl =
e.clientCourant;
                                                if(cl != null &&
!cl.alertesActivees().isEmpty()) {

 e.writeUDP(cl.alertesActiveesToString());
                                                }
                                        }
                                }
                                for(Connexion e : clientsDeconnectes){
                                        remove(e);
                                }
                                clientsDeconnectes.clear();
                        }
                }
        }
```

On vérifie à chaque fois si le client est connecté, si c'est le cas, il est placé dans une liste temporaire afin de le supprimer une fois sortie de la boucle. Ceci a été fait pour éviter les problèmes de concurrence d'accès à la liste des clients connectés *clientsConnectes*.

Il est très important de souligner qu'un seul et unique *ServeurAlerte (un thread)* est actif et permet de surveiller l'ensemble des alertes.

Une fois une alerte activée, il appelle la méthode *writeUDP(String)* de la classe Connexion pour envoyer une alerte à l'utilisateur représenté par son objet *Connexion*.

Les points de synchronisation

Afin de garantir une cohérence d'accès aux ressources, plusieurs points de synchronisation ont été définis :

- Au niveau de la classe interne *ServeurAlerte,* tout accès aux clients en ligne *ArrayList<Connexion> clientsConnectes* est synchronisé
- Au niveau de la classe *ServeurClient,* tout accès à la liste des alertes d'un client est synchronisé, *ArrayList<Alerte> alertes*
- Au niveau de la classe Ville, l'ajout d'une note ou d'un commentaire est synchronisé :

```
        public synchronized Float ajouterNote(String nomClient, Float
value) {
                return notes.put(nomClient, value);
        }

        public synchronized String ajouterCommentaire(String nomClient,
String value) {
                return commentaires.put(nomClient, value);
        }
```

Scénario automatique et aléatoire

Le scénario de réponse automatique est implémenté dans le *run()* de la classe *Client* du *package Client*. L'automate répond en fonction des messages Serveur, il implémente un (*switch, case)* qui permet de traiter le cas par cas, et y répondre par un entier généré aléatoirement compris dans le champ de réponse.

```
switch(etat){
                                        case 0 :
                                               // Login
                                               login = "Login : " +
System.currentTimeMillis();
                                               System.out.println(login);
                                               co.write(login);
                                               etat = 1;
                                               break;

                                        case 1 :
                                               // Menu Principal
                                               // Compris entre 1 et 4
                                               aleatoire = (int) ((int) 1 +
Math.random() * 4);
                                               System.out.println(aleatoire);
                                               co.write(aleatoire.toString());
                                               if(aleatoire == 1){
                                                      etat = 2;
                                               } else if (aleatoire == 2){
                                                      // Retour sur le menu
initial
                                               } else if (aleatoire == 3){
                                                      etat = 15;
                                               } else if (aleatoire == 4){
                                                      // Quitter
                                                      etat = 16;
                                               }
                                               break;
```

Gestion des messages UDP

Le traitement des alertes UDP est géré dans une classe interne *UDPReceive* de la classe *Connexion*. Cette classe crée un thread spécifique pour cette fonction qui surveille le buffer UDP et affiche son contenu à l'écran :

```
 protected class UDPReceive extends Thread {
        private DatagramSocket socketUDP;
        private DatagramPacket bufferUDP;

        public UDPReceive(int port) {
               try {
                      socketUDP = new DatagramSocket(port);
                      bufferUDP = new DatagramPacket(new byte[1024],
1024);
               } catch (SocketException e) {
                      e.printStackTrace();
               }
        }

        private String receive() {
               try {
                      socketUDP.receive(bufferUDP);
               } catch (IOException e) {
                      e.printStackTrace();
               }
               String str = new String(bufferUDP.getData(),
bufferUDP.getOffset(), bufferUDP.getLength());
               return str;

        }
```

Imprimes écran

Le scénario automatique permet de détecter les problèmes de conflit, ci-dessous quelques imprimes écran.

Exemple 1 : le client s'abonne à une activité

```
Saisissez votre nom d'utilisateur :
Login : 1326046798952
Le port est : 8187
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
2
La liste de villes est :
1. Paris
2. Nantes
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
1
Selectionnez la ville parmi :
1. Paris
2. Nantes
2
Selectionnez votre activite parmi :
1. Nom : Modelage    Debut  : Sun Feb 05 18:00:00 CET 2012    Fin : Sun
Feb 05 20:00:00 CET 2012    Type : Sport    Note : -1.0
2. Nom : Soin visage    Debut  : Sat May 05 16:00:00 CEST 2012    Fin :
Sun Feb 05 18:00:00 CET 2012Type : Bien-etre    Note : -1.0
3. Nom : Restaurant    Debut  : Thu Apr 05 20:00:00 CEST 2012    Fin :
Sun Feb 05 22:00:00 CET 2012Type : Restauration    Note : -1.0
4. Nom : Soin visage    Debut  : Mon Mar 05 18:00:00 CET 2012    Fin :
Sun Feb 05 20:00:00 CET 2012Type : Bien-etre    Note : -1.0
1
Que souhaitez vous faire.
1. Ajouter ou modifier une note.
2. Ajouter ou modifier un commentaire.
3. Ajouter ou modifier une note et un commentaire.
4. Ajouter une alerte.
5. Voir toutes les appréciations.
4
Souhaitez vous rentrer :
1. Note minimum.
2. Note minimum et maximum.
1
Saisissez la note minimum :
10
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
4
```

Exemple 2 : Le client s'abonne et note une activité

```
Saisissez votre nom d'utilisateur :
Login : 1326046967965
Le port est : 8195
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
```

```
3. Affiche les activités d'une ville.
4. Quitter.
1
Selectionnez la ville parmi :
1. Paris
2. Nantes
2
Selectionnez votre activite parmi :
1. Nom : Modelage    Debut  : Sun Feb 05 18:00:00 CET 2012    Fin : Sun
Feb 05 20:00:00 CET 2012    Type : Sport    Note : -1.0
2. Nom : Soin visage    Debut  : Sat May 05 16:00:00 CEST 2012    Fin :
Sun Feb 05 18:00:00 CET 2012Type : Bien-etre    Note : -1.0
3. Nom : Restaurant    Debut  : Thu Apr 05 20:00:00 CEST 2012    Fin :
Sun Feb 05 22:00:00 CET 2012Type : Restauration    Note : -1.0
4. Nom : Soin visage    Debut  : Mon Mar 05 18:00:00 CET 2012    Fin :
Sun Feb 05 20:00:00 CET 2012Type : Bien-etre    Note : -1.0
3
Que souhaitez vous faire.
1. Ajouter ou modifier une note.
2. Ajouter ou modifier un commentaire.
3. Ajouter ou modifier une note et un commentaire.
4. Ajouter une alerte.
5. Voir toutes les appréciations.
1
Saississez la note saisie :
11
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
1
Selectionnez la ville parmi :
1. Paris
2. Nantes
1
Selectionnez votre activite parmi :
1. Nom : Natation    Debut  : Thu Jan 05 20:00:00 CET 2012    Fin : Sun
Feb 05 22:00:00 CET 2012    Type : Sport    Note : -1.0
2. Nom : Coiffure    Debut  : Sun Feb 05 10:00:00 CET 2012    Fin : Sun
Feb 05 12:00:00 CET 2012    Type : Bien-etre    Note : -1.0
3. Nom : Restaurant    Debut  : Mon Mar 05 20:00:00 CET 2012    Fin : Sun
Feb 05 22:00:00 CET 2012    Type : Restauration    Note : -1.0
4. Nom : Soin visage    Debut  : Thu Apr 05 18:00:00 CEST 2012    Fin :
Sun Feb 05 20:00:00 CET 2012Type : Bien-etre    Note : -1.0
4
Que souhaitez vous faire.
1. Ajouter ou modifier une note.
2. Ajouter ou modifier un commentaire.
3. Ajouter ou modifier une note et un commentaire.
4. Ajouter une alerte.
5. Voir toutes les appréciations.
4
Souhaitez vous rentrer :
1. Note minimum.
2. Note minimum et maximum.
1
Saississez la note minimum :
15
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
2
La liste de villes est :
1. Paris
```

```
2. Nantes
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
4
```

Exemple 3 : Le client s'abonne et commente une activité

```
Saisissez votre nom d'utilisateur :
Login : 1326047071351
Le port est : 8199
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
1
Selectionnez la ville parmi :
1. Paris
2. Nantes
2
Selectionnez votre activite parmi :
1. Nom : Modelage    Debut  : Sun Feb 05 18:00:00 CET 2012    Fin : Sun
Feb 05 20:00:00 CET 2012    Type : Sport    Note : -1.0
2. Nom : Soin visage    Debut  : Sat May 05 16:00:00 CEST 2012    Fin :
Sun Feb 05 18:00:00 CET 2012Type : Bien-etre    Note : -1.0
3. Nom : Restaurant    Debut  : Thu Apr 05 20:00:00 CEST 2012    Fin :
Sun Feb 05 22:00:00 CET 2012Type : Restauration    Note : 11.0
4. Nom : Soin visage    Debut  : Mon Mar 05 18:00:00 CET 2012    Fin :
Sun Feb 05 20:00:00 CET 2012Type : Bien-etre    Note : -1.0
3
Que souhaitez vous faire.
1. Ajouter ou modifier une note.
2. Ajouter ou modifier un commentaire.
3. Ajouter ou modifier une note et un commentaire.
4. Ajouter une alerte.
5. Voir toutes les appréciations.
3
Saisissez la note saisie :
10
Saisissez votre commentaire :
Commentaire : 1326047071414
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
3
Selectionnez la ville parmi :
1. Paris
2. Nantes
2
Voici la liste des activites :
0. Nom : Modelage    Debut  : Sun Feb 05 18:00:00 CET 2012    Fin : Sun
Feb 05 20:00:00 CET 2012    Type : Sport    Note : -1.0
1. Nom : Soin visage    Debut  : Sat May 05 16:00:00 CEST 2012    Fin :
Sun Feb 05 18:00:00 CET 2012Type : Bien-etre    Note : -1.0
2. Nom : Restaurant    Debut  : Thu Apr 05 20:00:00 CEST 2012    Fin :
Sun Feb 05 22:00:00 CET 2012Type : Restauration    Note : 10.5
3. Nom : Soin visage    Debut  : Mon Mar 05 18:00:00 CET 2012    Fin :
Sun Feb 05 20:00:00 CET 2012Type : Bien-etre    Note : -1.0
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
```

```
3. Affiche les activités d'une ville.
4. Quitter.
```

Exemple 4 : Le client note, commente et s'abonne à une activité

```
Saisissez votre nom d'utilisateur :
Login : 1326047129901
Le port est : 8200
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
1
Selectionnez la ville parmi :
1. Paris
2. Nantes
1
Selectionnez votre activite parmi :
1. Nom : Natation    Debut  : Thu Jan 05 20:00:00 CET 2012    Fin : Sun
Feb 05 22:00:00 CET 2012    Type : Sport    Note : -1.0
2. Nom : Coiffure    Debut  : Sun Feb 05 10:00:00 CET 2012    Fin : Sun
Feb 05 12:00:00 CET 2012    Type : Bien-etre    Note : -1.0
3. Nom : Restaurant    Debut  : Mon Mar 05 20:00:00 CET 2012    Fin : Sun
Feb 05 22:00:00 CET 2012    Type : Restauration    Note : -1.0
4. Nom : Soin visage    Debut  : Thu Apr 05 18:00:00 CEST 2012    Fin :
Sun Feb 05 20:00:00 CET 2012Type : Bien-etre    Note : -1.0
2
Que souhaitez vous faire.
1. Ajouter ou modifier une note.
2. Ajouter ou modifier un commentaire.
3. Ajouter ou modifier une note et un commentaire.
4. Ajouter une alerte.
5. Voir toutes les appréciations.
3
Saississez la note saisie :
10
Saississez votre commentaire :
Commentaire : 1326047129909
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
1
Selectionnez la ville parmi :
1. Paris
2. Nantes
1
Selectionnez votre activite parmi :
1. Nom : Natation    Debut  : Thu Jan 05 20:00:00 CET 2012    Fin : Sun
Feb 05 22:00:00 CET 2012    Type : Sport    Note : -1.0
2. Nom : Coiffure    Debut  : Sun Feb 05 10:00:00 CET 2012    Fin : Sun
Feb 05 12:00:00 CET 2012    Type : Bien-etre    Note : 10.0
3. Nom : Restaurant    Debut  : Mon Mar 05 20:00:00 CET 2012    Fin : Sun
Feb 05 22:00:00 CET 2012    Type : Restauration    Note : -1.0
4. Nom : Soin visage    Debut  : Thu Apr 05 18:00:00 CEST 2012    Fin :
Sun Feb 05 20:00:00 CET 2012Type : Bien-etre    Note : -1.0
1
Que souhaitez vous faire.
1. Ajouter ou modifier une note.
2. Ajouter ou modifier un commentaire.
3. Ajouter ou modifier une note et un commentaire.
4. Ajouter une alerte.
5. Voir toutes les appréciations.
1
```

```
Saisissez la note saisie :
18
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
2
La liste de villes est :
1. Paris
2. Nantes
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
4
```

Exemple 5 : Réception d'une alerte

```
Saisissez la note saisie :
10
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
1
Selectionnez la ville parmi :
1. Paris
2. Nantes
1
Selectionnez votre activite parmi :
1. Nom : Natation    Debut  : Thu Jan 05 20:00:00 CET 2012    Fin : Sun
Feb 05 22:00:00 CET 2012    Type : Sport    Note : 10.0
2. Nom : Coiffure    Debut  : Sun Feb 05 10:00:00 CET 2012    Fin : Sun
Feb 05 12:00:00 CET 2012    Type : Bien-etre    Note : 10.0
3. Nom : Restaurant    Debut  : Mon Mar 05 20:00:00 CET 2012    Fin : Sun
Feb 05 22:00:00 CET 2012    Type : Restauration    Note : -1.0
4. Nom : Soin visage    Debut  : Thu Apr 05 18:00:00 CEST 2012    Fin :
Sun Feb 05 20:00:00 CET 2012Type : Bien-etre    Note : -1.0
1
Que souhaitez vous faire.
1. Ajouter ou modifier une note.
2. Ajouter ou modifier un commentaire.
3. Ajouter ou modifier une note et un commentaire.
4. Ajouter une alerte.
5. Voir toutes les appréciations.
4
Souhaitez vous rentrer :
1. Note minimum.
2. Note minimum et maximum.
1
Saisissez la note minimum :
12
Que souhaitez vous faire :
1. Ajouter une note et/ou une activité et/ou une alerte.
2. Affiche la liste des villes.
3. Affiche les activités d'une ville.
4. Quitter.
>>> ALERTE : noteMin : 12.0    noteMax : 20.0    Nom : Natation    Debut
: Thu Jan 05 20:00:00 CET 2012    Fin : Sun Feb 05 22:00:00 CET 2012Type
: Sport    Note : 10.0
```

Listing du code

Package Serveur

Serveur.java

```
package Serveur;

import java.io.IOException;
import java.net.ServerSocket;
import java.util.ArrayList;
import java.util.Iterator;


public class Serveur extends Thread {
  protected static ArrayList<Ville> villes;
  private static ArrayList<ServeurClient> clients;
  private static final int port = 8080;
  private ServerSocket srv;
  private static boolean edited = false;

  public Serveur() {
        try {
              srv = new ServerSocket(port);
              villes = new ArrayList<Ville>();
              clients = new ArrayList<ServeurClient>();
        } catch (IOException e) {
              e.printStackTrace();
        }
  }

  public Serveur(int portNumber) {
        try {
              srv = new ServerSocket(portNumber);
              villes = new ArrayList<Ville>();
              clients = new ArrayList<ServeurClient>();
        } catch (IOException e) {
              e.printStackTrace();
        }
  }

  public boolean add(Ville e) {
        return villes.add(e);
  }

  static public String villesToString() {
        String res = "";
        for(Ville villeCourante : villes) {
              res += villeCourante.toString() + "\n";
        }
        return res;
  }

  public static boolean add(ServeurClient e) {
        return clients.add(e);
  }

  public static ServeurClient isClient(String userName) {

        ServeurClient clientCourant;
        Iterator<ServeurClient> it = clients.iterator();
        while(it.hasNext()){
              clientCourant = it.next();
              if (clientCourant.getNomClient().equals(userName)){
                    return clientCourant;
```

```
                }
        }
        return null;
  }

  public class ServeurAlerte extends Thread {
        private ArrayList<Connexion> clientsConnectes;

        public ServeurAlerte() {
                clientsConnectes = new ArrayList<Connexion>();
        }

        public void add(Connexion c){
                synchronized(clientsConnectes) {
                        clientsConnectes.add(c);
                }
        }

        public boolean remove(Connexion o) {
                return clientsConnectes.remove(o);

        }

        public void run() {
                ArrayList<Connexion> clientsDeconnectes = new
ArrayList<Connexion>();
                while(true) {
                        synchronized(clientsConnectes) {
                                for(Connexion e : clientsConnectes){
                                        if(e.isClosed()) {
                                                clientsDeconnectes.add(e);
                                        } else {
                                                ServeurClient cl =
e.clientCourant;
                                                if(cl != null &&
!cl.alertesActivees().isEmpty()) {

  e.writeUDP(cl.alertesActiveesToString());
                                                }
                                        }
                                }
                                for(Connexion e : clientsDeconnectes){
                                        remove(e);
                                }
                                clientsDeconnectes.clear();
                        }
                }
        }
  }

  public void run() {
        try {
                Connexion c;
                ServeurAlerte srvAlerte = new ServeurAlerte();
                srvAlerte.start();
                while(true) {
                        c = new Connexion(srv.accept());
                        c.start();
                        srvAlerte.add(c);
                }
        } catch (IOException e) {
                e.printStackTrace();
        }
  }

  public static void main(String[] args) {
        Ville paris = new Ville("Paris");
```

```
        paris.addActivite("Natation", 2012, 0, 5, 20, 00, 00, 2012, 1, 5,
22, 00, 00, "Sport", "Info.");
        paris.addActivite("Coiffure", 2012, 1, 5, 10, 00, 00, 2012, 1, 5,
12, 00, 00, "Bien-etre", "Info.");
        paris.addActivite("Restaurant", 2012, 2, 5, 20, 00, 00, 2012, 1,
5, 22, 00, 00, "Restauration", "Info.");
        paris.addActivite("Soin visage", 2012, 3, 5, 18, 00, 00, 2012, 1,
5, 20, 00, 00, "Bien-etre", "Info.");
        Ville nantes = new Ville("Nantes");
        nantes.addActivite("Modelage", 2012, 1, 5, 18, 00, 00, 2012, 1,
5, 20, 00, 00, "Sport", "Info.");
        nantes.addActivite("Soin visage", 2012, 4, 5, 16, 00, 00, 2012,
1, 5, 18, 00, 00, "Bien-etre", "Info.");
        nantes.addActivite("Restaurant", 2012, 3, 5, 20, 00, 00, 2012, 1,
5, 22, 00, 00, "Restauration", "Info.");
        nantes.addActivite("Soin visage", 2012, 2, 5, 18, 00, 00, 2012,
1, 5, 20, 00, 00, "Bien-etre", "Info.");
        Serveur srv = new Serveur();
        srv.add(paris);
        srv.add(nantes);
        srv.run();
  }
}
```

Connexion.java

```
package Serveur;

import java.io.IOException;
import java.net.ServerSocket;
import java.util.ArrayList;
import java.util.Iterator;


public class Serveur extends Thread {
  protected static ArrayList<Ville> villes;
  private static ArrayList<ServeurClient> clients;
  private static final int port = 8080;
  private ServerSocket srv;
  private static boolean edited = false;

  public Serveur() {
        try {
              srv = new ServerSocket(port);
              villes = new ArrayList<Ville>();
              clients = new ArrayList<ServeurClient>();
        } catch (IOException e) {
              e.printStackTrace();
        }
  }

  public Serveur(int portNumber) {
        try {
              srv = new ServerSocket(portNumber);
              villes = new ArrayList<Ville>();
              clients = new ArrayList<ServeurClient>();
        } catch (IOException e) {
              e.printStackTrace();
        }
  }

  public boolean add(Ville e) {
        return villes.add(e);
  }
```

```
  static public String villesToString() {
        String res = "";
        for(Ville villeCourante : villes) {
              res += villeCourante.toString() + "\n";
        }
        return res;
  }

  public static boolean add(ServeurClient e) {
        return clients.add(e);
  }

  public static ServeurClient isClient(String userName) {

        ServeurClient clientCourant;
        Iterator<ServeurClient> it = clients.iterator();
        while(it.hasNext()){
              clientCourant = it.next();
              if (clientCourant.getNomClient().equals(userName)){
                    return clientCourant;
              }
        }
        return null;
  }

  public class ServeurAlerte extends Thread {
        private ArrayList<Connexion> clientsConnectes;

        public ServeurAlerte() {
              clientsConnectes = new ArrayList<Connexion>();
        }

        public void add(Connexion c){
              synchronized(clientsConnectes) {
                    clientsConnectes.add(c);
              }
        }

        public boolean remove(Connexion o) {
              return clientsConnectes.remove(o);

        }

        public void run() {
              ArrayList<Connexion> clientsDeconnectes = new
ArrayList<Connexion>();
              while(true) {
                    synchronized(clientsConnectes) {
                          for(Connexion e : clientsConnectes){
                                if(e.isClosed()) {
                                      clientsDeconnectes.add(e);
                                } else {
                                      ServeurClient cl =
e.clientCourant;
                                      if(cl != null &&
!cl.alertesActivees().isEmpty()) {

 e.writeUDP(cl.alertesActiveesToString());
                                      }
                                }
                          }
                          for(Connexion e : clientsDeconnectes){
                                remove(e);
                          }
                          clientsDeconnectes.clear();
                    }
              }
```

```
        }
  }

  public void run() {
        try {
              Connexion c;
              ServeurAlerte srvAlerte = new ServeurAlerte();
              srvAlerte.start();
              while(true) {
                    c = new Connexion(srv.accept());
                    c.start();
                    srvAlerte.add(c);
              }
        } catch (IOException e) {
              e.printStackTrace();
        }
  }

  public static void main(String[] args) {
        Ville paris = new Ville("Paris");
        paris.addActivite("Natation", 2012, 0, 5, 20, 00, 00, 2012, 1, 5,
22, 00, 00, "Sport", "Info.");
        paris.addActivite("Coiffure", 2012, 1, 5, 10, 00, 00, 2012, 1, 5,
12, 00, 00, "Bien-etre", "Info.");
        paris.addActivite("Restaurant", 2012, 2, 5, 20, 00, 00, 2012, 1,
5, 22, 00, 00, "Restauration", "Info.");
        paris.addActivite("Soin visage", 2012, 3, 5, 18, 00, 00, 2012, 1,
5, 20, 00, 00, "Bien-etre", "Info.");
        Ville nantes = new Ville("Nantes");
        nantes.addActivite("Modelage", 2012, 1, 5, 18, 00, 00, 2012, 1,
5, 20, 00, 00, "Sport", "Info.");
        nantes.addActivite("Soin visage", 2012, 4, 5, 16, 00, 00, 2012,
1, 5, 18, 00, 00, "Bien-etre", "Info.");
        nantes.addActivite("Restaurant", 2012, 3, 5, 20, 00, 00, 2012, 1,
5, 22, 00, 00, "Restauration", "Info.");
        nantes.addActivite("Soin visage", 2012, 2, 5, 18, 00, 00, 2012,
1, 5, 20, 00, 00, "Bien-etre", "Info.");
        Serveur srv = new Serveur();
        srv.add(paris);
        srv.add(nantes);
        srv.run();
  }
}
```

ServeurClient.java

```
package Serveur;

import java.net.InetAddress;
import java.util.ArrayList;

public class ServeurClient extends Connexion {
  private String nomClient;
  private InetAddress adresseIP;
  private int portClient;
  private ArrayList<Alerte> alertes = new ArrayList<Alerte>();

  public ServeurClient(String nomClient, InetAddress adresseIP, int port)
{
        super();
        this.nomClient = nomClient;
        this.adresseIP = adresseIP;
        this.portClient = port++;
  }

  public String getNomClient() {
```

```
		return nomClient;
	}

	public void setNomClient(String nomClient) {
		this.nomClient = nomClient;
	}

	public InetAddress getAdresseIP() {
		return adresseIP;
	}

	public void setAdresseIP(InetAddress adresseIP) {
		this.adresseIP = adresseIP;
	}

	protected int getPortClient() {
		return portClient;
	}

	protected void setPortClient(int portClient) {
		this.portClient = portClient;
	}

	public void abonnerAlerte(Alerte e) {
		synchronized (alertes) {
			alertes.add(e);
		}
	}

	public ArrayList<Alerte> alertesActivees() {
		// retourne la liste des alertes activées
		ArrayList<Alerte> list = new ArrayList<Alerte>();
		synchronized (alertes) {
			for(Alerte e : alertes) {
				if(e.alerteActivee() && !e.isNotifiee())
list.add(e);
			}
		}
		return list;
	}

	public String alertesActiveesToString() {
		ArrayList<Alerte> list = alertesActivees();
		String res = "";
		for(Alerte e : list) {
			res += e.toString();
			e.setNotifiee(true);
		}
		return res;
	}

	public class Alerte {

		private Ville.Activite activite;
		private float noteMin = 0;
		private float noteMax = 20;
		private boolean notifiee = false;

		public Alerte(Ville.Activite e, float noteMin){
			this.activite = e;
			this.noteMin = noteMin;

		}

		public Alerte(Ville.Activite e, float noteMin, float noteMax){
			this.activite = e;
			this.noteMin = noteMin;
```

```
		this.noteMax = noteMax;
	}

	public float getNoteMin() {
		return noteMin;
	}

	public void setNoteMin(float noteMin) {
		this.noteMin = noteMin;
	}

	public float getNoteMax() {
		return noteMax;
	}

	public void setNoteMax(float noteMax) {
		this.noteMax = noteMax;
	}

	protected boolean isNotifiee() {
		return notifiee;
	}

	protected void setNotifiee(boolean notifiee) {
		this.notifiee = notifiee;
	}

	public synchronized boolean alerteActivee() {
		float moy = activite.getMoyenne();
		if (moy < noteMin && moy < noteMax ){
			return true;
		}
		return false;
	}

	public synchronized String toString() {
		String res = ">>> ALERTE : noteMin : " + this.noteMin + "
noteMax : " + this.noteMax + "    " + this.activite.toString() ;
		return res;
	}
  }

}
```

Ville.java

```
package Serveur;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Locale;
import java.util.Set;


public class Ville {
  private String nom;
  private ArrayList<Activite> activites;

  public Ville (String nom){
	this.setNom(nom);
	activites = new ArrayList<Activite>();
  }
```

```
  public void addActivite(String nom, int yearDebut, int monthDebut, int
dateDebut, int hourOfDayDebut,
                int minuteDebut, int secondDebut, int yearFin, int
monthFin, int dateFin, int hourOfDayFin,
                int minuteFin, int secondFin, String type, String
informations){
        activites.add(new Activite(nom, yearDebut, monthDebut, dateDebut,
hourOfDayDebut, minuteDebut, secondDebut,
                        yearFin, monthFin, dateFin, hourOfDayFin, minuteFin,
secondFin, type, informations));
  }

  public String getNom() {
        return nom;
  }

  public void setNom(String nom) {
        this.nom = nom;
  }

  public Activite getActiviteByName(String s) {
        for(Activite a : activites){
                s.equals(a.getNom());
                return a;
        }
        return null;
  }

  public String toString() {
        return nom;
  }

  public String activitesToString() {
        String res = "\nActivités :\n";
        for(Activite activiteCourante : activites) {
                res += activiteCourante.toString();
        }
        return res;
  }

  public Iterator<Activite> activitesToIterator() {
        return activites.iterator();
  }

  public boolean addActiviteNote(Activite activite, String nomClient,
Float note){
        Iterator<Activite> it = activites.iterator();
        while(it.hasNext()){
                Activite activiteCourante = it.next();
                if (activiteCourante.equals(activite)){
                        activiteCourante.ajouterNote(nomClient, note);
                        return true;
                }
        }
        return false;
  }

  public boolean addActiviteCommentaire(Activite activite, String
nomClient, String commentaire){
        Iterator<Activite> it = activites.iterator();
        while(it.hasNext()){
                Activite activiteCourante = it.next();
                if (activiteCourante.equals(activite)){
                        activiteCourante.ajouterCommentaire(nomClient,
commentaire);
                        return true;
                }
```

```
		}
		return false;
	}

	public String activiteDejaCommente(Activite activite, String nomClient){
		Iterator<Activite> it = activites.iterator();
		while(it.hasNext()){
			Activite activiteCourante = it.next();
			if (activiteCourante.equals(activite)){
				if
(activiteCourante.commentaires.containsKey(nomClient)){
					return
activiteCourante.commentaires.get(nomClient);
				}
				else {
					return null;
				}
			}
		}
		return null;
	}

	public Float activiteDejaNote(Activite activite, String nomClient){
		Iterator<Activite> it = activites.iterator();
		while(it.hasNext()){
			Activite activiteCourante = it.next();
			if (activiteCourante.equals(activite)){
				if (activiteCourante.notes.containsKey(nomClient)){
					return activiteCourante.notes.get(nomClient);
				}
				else {
					return null;
				}
			}
		}
		return null;
	}

	public boolean verifierActivite (String nomActivite, float noteMin,
float noteMax){

		Iterator<Activite> it = activites.iterator();
		while(it.hasNext()){
			Activite activiteCourante = it.next();
			if (activiteCourante.nom == nomActivite &&
activiteCourante.getMoyenne() > noteMin && activiteCourante.getMoyenne()
< noteMax){
				return true;
			}
		}
		return false;

	}

	/*
	 * Classe Interne Activié : Celle-ci présente les avantages de ...
	 * Une Activité est forcement lié a une ville
	 * Bémol : On ne peut faire une recherche des villes qui contiennent une
activité
	 *		sans perdre enormément de temps
	 */
	public class Activite {

		private String nom;
		private Calendar debut;
		private Calendar fin;
		private String type;
```

```
	private String informations;
	private HashMap <String, String> commentaires;
	private HashMap <String, Float> notes;

	/*
	 * On stockera <Nom_CLient, Commentaires> ou <Nom_Client, Notes>
	 * Afin de permettre la modification
	 * Logiquement, un même utilisateur ne peut mettre deux
commentaires ou notes
	 */


	public Activite(String nom, int yearDebut, int monthDebut, int
dateDebut, int hourOfDayDebut,
			int minuteDebut, int secondDebut, int yearFin, int
monthFin, int dateFin, int hourOfDayFin,
			int minuteFin, int secondFin, String type, String
informations) {
		this.setNom(nom);
		debut = Calendar.getInstance(new Locale("French"));
		fin = Calendar.getInstance(new Locale("French"));
		setDebut(yearDebut, monthDebut, dateDebut, hourOfDayDebut,
minuteDebut, secondDebut);
		setFin(yearFin, monthFin, dateFin, hourOfDayFin,
minuteFin, secondFin);
		this.setType(type);
		this.setInformations(informations);

		commentaires = new HashMap<String, String>();
		notes = new HashMap<String, Float>();
	}

	public String getNom() {
		return nom;
	}

	public void setNom(String nom) {
		this.nom = nom;
	}

	public final void setDebut(int year, int month, int date, int
hourOfDay,
			int minute, int second) {
		debut.set(year, month, date, hourOfDay, minute, second);
	}

	public Calendar getDebut(){
		return debut;
	}

	public final void setFin(int year, int month, int date, int
hourOfDay,
			int minute, int second) {
		fin.set(year, month, date, hourOfDay, minute, second);
	}

	public Calendar getFin(){
		return fin;
	}

	public String getType() {
		return type;
	}

	public void setType(String type) {
		this.type = type;
	}
```

```
	public String getInformations() {
		return informations;
	}

	public void setInformations(String informations) {
		this.informations = informations;
	}

	public float getMoyenne(){
		float moyenne = 0;
		int i = 0;
		if(!notes.isEmpty()) {
			for(Float noteCourante : notes.values()){
				i++;
				moyenne += noteCourante;
			}
			return moyenne/i;
		}
		return -1;
	}

	public String toString() {
		return "Nom : " + nom + "    Debut  : " +
debut.getTime().toString()
				+ "    Fin : " + fin.getTime().toString()
				+ "    Type : " + type + "    Note : " +
getMoyenne() + "\n";
	}

	public synchronized Float ajouterNote(String nomClient, Float
value) {
		return notes.put(nomClient, value);
	}

	public synchronized String ajouterCommentaire(String nomClient,
String value) {
		return commentaires.put(nomClient, value);
	}

	public String getAppreciation(){
		String res = toString() + "\n";
		res += "Note\t\t\tCommentaire";
		Set<String> noteCle = notes.keySet();
		Set<String> commentairesCle = commentaires.keySet();
		String cle;
		Iterator<String> it = noteCle.iterator();
		while(it.hasNext()){
			cle = it.next();
			if(notes.containsKey(cle)){
				res += notes.get(cle) + "\t\t\t";
			} else {
				res += "\t\t\t\t";
			}
			if(commentaires.containsKey(cle)){
				res += commentaires.get(cle) + "\n";
				commentairesCle.remove(cle);
			} else {
				res += "\n";
			}
		}
		it = commentairesCle.iterator();
		while(it.hasNext()){
			cle = (String) it.next();
			res += "\t\t\t\t";
			res += commentaires.get(cle) + "\n";
		}
```

```
                return res;
        }


  }


}
```

Package Client

Client.java

```
package Client;

import java.io.*;
import java.net.*;

import Client.Connexion.UDPReceive;

public class Client extends Thread {

  Connexion co;

  public Client (){
        try {
                this.co = new Connexion(new Socket("localhost", 8080));
        } catch (UnknownHostException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
  }

  public String readClavier() {
        String ligne = null;
        InputStreamReader buffer = new InputStreamReader(System.in);
        BufferedReader clavier = new BufferedReader(buffer);
        try {
                if(clavier.ready()) ligne = clavier.readLine();
        } catch (IOException e) {
                e.printStackTrace();
        }
        return ligne;
  }

  public void run() {
        int etat = 0;
        boolean exit = false;
        Integer aleatoire;
        String login, aleaStr;

        boolean scenario = true;

        co.openConnection();
        //String lectureClavier = null;
        String lectureMessages = null;
        String lectureClavier = null;
        UDPReceive udp = null;
        while(!exit) {
                lectureClavier = null;
                lectureMessages = null;
                lectureMessages = co.read();
                if(lectureMessages != null && !lectureMessages.isEmpty())
{
                        if(lectureMessages.startsWith("<PORT>")) {
```

```
                                String[] tabSplit =
lectureMessages.split(">");
                                int portSplit =
Integer.parseInt(tabSplit[1]);
                                System.out.println("Le port est : " +
portSplit);
                                udp = co.new UDPReceive(portSplit);
                                udp.start();
                        } else if
(lectureMessages.equalsIgnoreCase("Exit")){
                                co.write("Exit");
                                //udp.close();
                                co.closeConnection();
                                udp.interrupt();
                                //System.exit(0);
                                exit = true;
                        } else {
                                if
(!lectureMessages.equalsIgnoreCase("<STOP>")){
                                        System.out.println(lectureMessages);

                                        continue;
                                }

                                if (scenario){

                                        /**** SCENARIO ****/
                                        /**
                                Liste des états :
                                0 : Login
                                1 : Menu Principal [1-4]
                                2 : Saisir Ville [1-2]
                                3 : Saisir Activité [1-4]
                                4 : Menu Action [1-5]
                                5 : Saisir Commentaire
                                6 : Saisir Note
                                7 : Saisir Note pour Note & Commentaire
                                8 : Saisir Commentaire pour Note &
Commentaire
                                9 : Menu Alerte [1-2]
                                10 : Saisir Note Minimum
                                11 : Saisir Note Minimum pour Note Mininim et
Note Maximum
                                12 : Saisir Note Maximum pour Note Mininim et
Note Maximum
                                13 : Affichage des Appréciations
                                14 : Affichage des villes
                                15 : Saisir Ville [1-2] pour Affichage des
activités
                                16 : Quitter


                                         **/

                                        switch(etat){
                                        case 0 :
                                                // Login
                                                login = "Login : " +
System.currentTimeMillis();
                                                System.out.println(login);
                                                co.write(login);
                                                etat = 1;
                                                break;

                                        case 1 :
                                                // Menu Principal
                                                // Compris entre 1 et 4
```

```
                                        aleatoire = (int) ((int) 1 +
Math.random() * 4);
                                        System.out.println(aleatoire);
                                        co.write(aleatoire.toString());
                                        if(aleatoire == 1){
                                               etat = 2;
                                        } else if (aleatoire == 2){
                                               // Retour sur le menu
initiale
                                        } else if (aleatoire == 3){
                                               etat = 15;
                                        } else if (aleatoire == 4){
                                               // Quitter
                                               etat = 16;
                                        }
                                        break;

                                 case 2 :
                                        // Menu Choix Ville
                                        // Compris entre 1 et 2
                                        aleatoire = (int) (1 +
Math.random() * 2);
                                        System.out.println(aleatoire);
                                        co.write(aleatoire.toString());
                                        etat = 3;
                                        break;

                                 case 3 :
                                        // Menu Choix Activité
                                        // Compris entre 1 et 4
                                        aleatoire = (int) (1 +
Math.random() * 4);
                                        System.out.println(aleatoire);
                                        co.write(aleatoire.toString());
                                        etat = 4;
                                        break;

                                 case 4 :
                                        // Menu Choix
Noter/Commenter/NoterEtCommenter/Alerte/AfficherAppreciation
                                        // Compris entre 1 et 5
                                        aleatoire = (int) (1 +
Math.random() * 5);
                                        System.out.println(aleatoire);
                                        co.write(aleatoire.toString());
                                        if(aleatoire == 1){
                                               etat = 6;
                                        } else if (aleatoire == 2){
                                               etat = 5;
                                        } else if (aleatoire == 3){
                                               etat = 7;
                                        } else if (aleatoire == 4){
                                               etat = 9;
                                        } else if (aleatoire == 5){
                                               etat = 13;
                                        }
                                        break;

                                 case 5 :
                                        aleaStr = "Commentaire : " +
System.currentTimeMillis();
                                        System.out.println(aleaStr);
                                        co.write(aleaStr);
                                        etat = 1;
                                        break;

                                 case 6 :
```

```
                                        aleatoire = (int) (Math.random()
* 20);
                                        System.out.println(aleatoire);
                                        co.write(aleatoire.toString());
                                        etat = 1;
                                        break;

                                case 7 :
                                        aleatoire = (int) (Math.random()
* 20);
                                        System.out.println(aleatoire);
                                        co.write(aleatoire.toString());
                                        etat = 8;
                                        break;

                                case 8 :
                                        aleaStr = "Commentaire : " +
System.currentTimeMillis();
                                        System.out.println(aleaStr);
                                        co.write(aleaStr);
                                        etat = 1;
                                        break;

                                case 9 :
                                        aleatoire = (int) ( 1 +
Math.random() * 2);
                                        System.out.println(aleatoire);
                                        co.write(aleatoire.toString());
                                        if (aleatoire == 1){
                                                etat = 10;
                                        } else if (aleatoire == 2){
                                                etat = 11;
                                        }
                                        break;

                                case 10 :
                                        // Insérer note min
                                        // Compris entre 0 et 20
                                        aleatoire = (int) (Math.random()
* 20);
                                        System.out.println(aleatoire);
                                        co.write(aleatoire.toString());
                                        etat = 1;
                                        break;

                                case 11 :
                                        // Insérer note min
                                        // Compris entre 0 et 20
                                        aleatoire = (int) (Math.random()
* 20);
                                        System.out.println(aleatoire);
                                        co.write(aleatoire.toString());
                                        etat = 12;
                                        break;

                                case 12 :
                                        // Insérer note max
                                        // Compris entre 0 et 20
                                        aleatoire = (int) (Math.random()
* 20);
                                        System.out.println(aleatoire);
                                        co.write(aleatoire.toString());
                                        etat = 1;
                                        break;

                                case 13 :
                                        // Affichage uniquement
```

```
                                        etat = 1;
                                        break;

                                case 14 :
                                        // Affichage uniquement
                                        etat = 1;
                                        break;

                                case 15 :
                                        // Menu Choix Ville
                                        // Compris entre 1 et 2
                                        aleatoire = (int) (1 +
Math.random() * 2);

                                        System.out.println(aleatoire);
                                        co.write(aleatoire.toString());
                                        etat = 14;
                                        break;

                                case 16 :
                                        // Affichage uniquement
                                        co.write("Exit");
                                        //udp.close();
                                        //co.closeConnection();
                                        exit = true;
                                        break;
                                }

                                /**** SCENARIO ****/

                        }
                }
        }

        if (!scenario){


                lectureClavier = readClavier();


                if(lectureClavier != null &&
!lectureClavier.isEmpty() && !lectureClavier.equals("Quit")) {
                        co.write(lectureClavier);
                }
                /*
                if(lectureClavier != null &&
lectureClavier.equals("Exit")) {
                        co.write("Exit");
                        //udp.close();
                        co.closeConnection();
                        udp.interrupt();
                        //System.exit(0);
                        exit = true;
                }
                 */
        }
    }
    co.closeConnection();
    udp.interrupt();
    System.exit(0);
 }

 public static void main (String args[]){
        Client c = new Client();
        c.start();
 }

}
```

Connexion.java

```
package Client;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.Socket;
import java.net.SocketException;
import java.util.Iterator;

import Serveur.Ville.Activite;

public class Connexion {
  private Socket s;
  private PrintWriter out;
  private BufferedReader in;

  public Connexion(Socket s) {
        this.s = s;
  }

  public void openConnection() {
        try {
              in = new BufferedReader(new
InputStreamReader(s.getInputStream()));
              out = new PrintWriter(s.getOutputStream(), true);
        } catch (IOException e) {
              e.printStackTrace();
        }
  }

  public void closeConnection() {
        try {
              in.close();
              out.close();
              s.close();
        } catch (IOException e) {
              e.printStackTrace();
        }
  }

  public void write(String s) {
        out.println(s);
        out.flush();
  }

  public String read() {
        try {
              if(in.ready()) return in.readLine();
        } catch (IOException e) {
              e.printStackTrace();
        }
        return null;
  }

  protected class UDPReceive extends Thread {
        private DatagramSocket socketUDP;
        private DatagramPacket bufferUDP;

        public UDPReceive(int port) {
              try {
```

```
                        socketUDP = new DatagramSocket(port);
                        bufferUDP = new DatagramPacket(new byte[1024],
1024);
                } catch (SocketException e) {
                        e.printStackTrace();
                }
        }

        private String receive() {
                try {
                        socketUDP.receive(bufferUDP);
                } catch (IOException e) {
                        e.printStackTrace();
                }
                String str = new String(bufferUDP.getData(),
bufferUDP.getOffset(), bufferUDP.getLength());
                return str;

        }

        public void close() {
                socketUDP.disconnect();
                socketUDP.close();
        }

        public void run() {
                while(true) {
                        // Intruction bloquante
                        System.out.println(receive());
                }
        }
  }
}
```