Chapter 1

Matériel

1.1 Machine virtuelle

1.1.1 Environnement de travail

Cette partie explique les intstallations effectuées sur la machine virtuelle, nous détaillerons plus tard le choix des programmes installés. Pour travailler sur la machine virtuelle, nous nous sommes connectés en *ssh*.

Système d'exploitation

Le système d'exploitation de la machine virtuelle est *CentOS*, pour le savoir il suffit de lire le contenu du fichier */proc/version*. Dans notre cas, il contient :

```
Linux version 3.10.0-1062.4.3.el7.x86_64 (mockbuildkbuilder.bsys.centos.org) (gcc version 4.8.5
20150623 (Red Hat 4.8.5-39) (GCC) ) #1 SMP Wed Nov 13 23:58:53 UTC 2019
```

Cette information nous permettra de savoir comment installer les programmes dont nous auront besoin.

1.1.2 Installation et paramétrage des logiciels

Installation des logiciels

Pour installer un programme, on peut utiliser le gestionnaire de paquets de CentOS : *yum*. Il faudra ensuite lancer les commandes suivantes, qui installent respectivement Java, le *Système de Gestion de Base de Données*, screen, netcat, nmap

```
1                    sudo yum install java-1.8.0-openjdk
2                    sudo yum install mysql-server
3                    sudo yum install screen
4                    sudo yum install nc
5                    sudo yum install nmap
```

Paramétrage du SGBD

Il a tout d'abord fallu mettre le mot de passe administrateur pour la connexion au SGBD grâce à la commande suivante.

```
1                    sudo mysqladmin variables -u root -p
```

On peut ensuite modifier le contenu du SGBD en utilisant l'interpréteur de commandes mysql :

```
mysql -u root -p
```

Il faut enfin créer puis sélectionné la base de données que l'on veut utiliser :

```
CREATE DATABASE <nom de la base de données>
USE DATABASE <nom de la base de données>
```

Exécution du serveur java

Screen est un logiciel qui permet de créer des terminaux (appelés *sessions*), de s'y déconnecter puis reconnecté autant que désiré. Ces terminaux continueront leur exécution en arrière plan. On peut donc utiliser l'entrée et la sortie standard pour communiquer avec le programme. Pour se connecter à une session ou la créer si elle n'existe pas, on utilise l'option -R.

```
screen -R <nom_du_terminal>
```

1.2 Ordiphone

Chapter 2

Choix BDD

2.1 Base de données

2.1.1 Choix de la base de donnée

Nous avons tout d'abord dû choisir quelle technologie nous allions utilisé pour notre base de données. Nous connaissions en connaissions un nombre suffisant pour ne pas chercher autre chose et juste faire un choix.

Technologies que nous connaissions

Nom	Niveau de connaissance	Facilité d'intégration	License	
PostGreSQL	Inconnu	Oui	Oui	
MySQL	Bon	Oui	Oui	GPLv2
Oracle Database	Bon	Oui	Oui	
sqlite3	Bon	Non	Oui	

2.2 Communication avec l'application

2.2.1 Serveur

2.2.2 Protocole

Chapter 3

Serveur

3.1 Architecture

3.1.1 Processus

Le serveur utilise une architecture avec plusieurs processus. Le processus principal attend une connection provenant d'un client (application). A chaque connection un nouveau processus est créé pour intéragir avec le client. Ce processus attend donc un message du client, exécute la commande SQL nécessaire pour obtenir une réponse et enfin, il répond au client en formattant les données. Un processus supplémentaire existe pour pouvoir travailler directement avec le serveur, sans passer par un client. Pour cela, l'entrée et la sortie standard sont utilisées et il faut donc un accès direct à la machine.

3.1.2 Classes

Le processus principal n'utilise qu'une seule classe *Serveur*, tout comme le processus de communication direct *LocalCommand*. Cependant le processus de communication avec le client est séparé en plusieurs objets : *Client*, *SQLHandler* et *CommunicationHandler*. Le premier est le processus en lui même et utilise les deux autres pour effectuer les tâches qui lui sont assignées. L'objet *SQLHandler* accède à la base de données et formatte les réponses. Enfin le *CommunicationHandler* gère la communication réseau avec le client, il permet la réception et l'envoie de chaîne de caractères. Enfin deux classes sont utilisées pour facilité le travail de conversion des formats pour les points : *Point3D* et *Point4D* qui correspondent respectivement à une coordonnée dans l'espace : (x, y, z) et à une coordonnée dans l'espace et le temps : (x, y, z, t).

3.1.3 Echange type

TODO : Diagramme de séquence

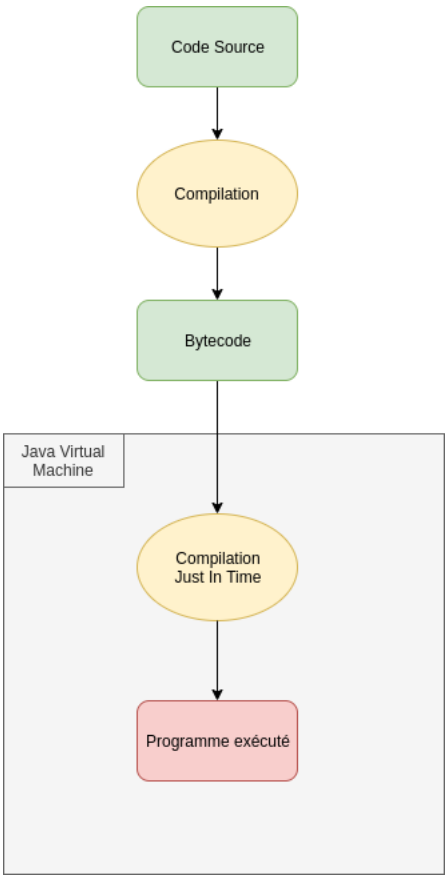
3.2 Implémentation

3.2.1 Langage

La création du serveur devait à l'origine être rapide car on le considérait comme étant annexe. C'est pourquoi on a choisi l'utilisation d'une technologie que nous connaissions déjà : *Java*. Ce langage contient dans sa librairie standard tout ce qu'il faut pour développer un serveur. De plus il existe une librairie java développée par Oracle pour la communication avec les bases de données MySQL.

Machine Virtuelle

Le *Java* est un langage compilé particulier. Le code est tout d'abord compilé dans un langage intermédiaire appelé *Bytecode*. Puis, il est exécuté dans une machine virtuelle appelée la *Java Virtual Machine* en utilisant une nouvelle étape de compilation *Just In Time*.



Les programmes développés et compilés en Java peuvent être exécuté sur toutes les machines possédant la JVM installée sans avoir aucun changement de code source ou bien de paramêtre de compulation.

3.2.2 Fonctionnement des processus

L'organisation des processus en java est particulière. En effet, il n'y a qu'un seul processus lourd (à la différence des forks du langage C). Ce processus lourd unique est la JVM. Tous les autres processus, y compris le processus principal de notre programme ne sont que des processus légers. Il y a donc un partage des resources, et l'échange de données est plus simple. Il faut attention faire cependant aux accès simultanés aux ressources.

Création de processus

Le processus principal de notre programme est créé automatiquement par la JVM et il exécute le code de la fonction *public static void main(String[] args)* qui est le point d'entrée de notre programme. Pour construire d'autres processus il y a plusieurs manières. On va ici se concentrer sur la classe *Thread* et l'interface *Runnable*.

Utilisation de *Thread*

Un *Thread* est un objet qui exécute du code dans un autre processus. Pour cela, il suffit de créer un Thread et de le lancer en utilisant la méthode *start()*.

```
1 Thread monThread = new Thread();
2 monThread.start();
```

Pour changer le code exécuté, il suffit de redéfinir la méthode *void run()* de Thread.

```
1 Thread monThread = new Thread() {
2     @Override
3     public void run() {
4         System.out.println("Nouveau processus");
5     }
6 }
7                     monThread.start();
```

3.2.3 Patron de conception

TODO : Fabric/Builder sur le serveur pour les clients
TODO : Singleton Serveur + (LocalCommand ???)

3.2.4 Communication réseau

Connection avec le client

En Java, on utilise les objets de type *Socket* pour faire de la communication en réseau. C'est objets utilisent le protocole TCP pour communiquer et permettent donc de s'assurer de l'état de la connection. Un premier socket (*ServerSocket*) permet d'attendre qu'un client se connecte et de créer un *Socket* pour communiquer avec lui.

```
1 ServerSocket socket = new ServerSocket(PORT);
2 while (true) {
3     Socket clientSocket = socket.accept();
4 }
```

Ensuite pour échanger avec le client, on utilise les flux d'entrée et de sortie fournis par le Socket. Les *BufferedReader* et *PrintWriter* sont des objets qui permettent de traiter les flux plus simplement, grâce à des chaînes de caractères.

```
1 InputStream input = socket.getInputStream();
2 OutputStream output = socket.getOutputSteam();
3
4 BufferedReader reader = new BufferedReader(new InputStreamReader(input));
5 PrintWriter writer = new PrintWriter(output);
```

Protocole de communication

Liste des messages possibles dans le sens clients → serveur

Commande	Utilisation
Subscribe:$< id >:< mdp >$	Permet de s'inscrire
Connect:$< id >:< mdp >$	Permet de se connecter à son compte
History:$< debut >:< fin >$	Permet de récupérer x trajets entre début et fin (en id)
Projects:$< debut >:< fin >$	Permet de récupérer les x projets entre début et fin
NewP:$< nom >:< x + y + z; ... >$	Ajoute un nouveau projet
NewJ:$< nom >:< x + y + z + t; ... >$	Ajoute un nouveau trajet
EditP:$< id >:< x + y + z; ... >$	Modifie un projet

Liste des messages possibles dans le sens serveur → client

Commande	Utilisation
Subscribed:$< id >$	Confirme l'inscription
Unsubscribed	Erreur lors de l'inscripton
Connected:$< id >$	Valide la connection à son compte
Unconnected	Erreur lors de la connection
Project:$< id >:< nom >:< x + y + z; ...; x + y + z >$	Envoie des informations sur un projet
Journey:$< id >:< nom >:< d >:< x + y + z + t... >$	Envoie des informations sur un projet

Chapter 4

Problèmes rencontrés

4.1 VPN

TODO: Problème de connexion au Téléphone

4.2 VM