Projet INFO4B

Exploitation des données de LiChess

Clothilde STRAINCHAMPS

Meryem BELASSEL

MI4-01

Clothilde STRAINCHAMPS
Meryem BELASSEL
MI4-01

Plan :

Clothilde STRAINCHAMPS
Meryem BELASSEL
MI4-01

-----INTRODUCTION-----

Parmi les trois projets proposés, on a choisi le premier sujet qui est « L'exploitation des données de LiChess », parce qu'il nous paraissait le plus abordable et c'était aussi un projet qui est assez proche de ce qu'on faisait lors des Tps.

Tout d'abord, le projet consiste à télécharger un fichier du site web : www.lichess.org open database qui va nous donner un fichier texte avec des données dans un certain ordre précis. Le but de notre projet est d'ouvrir une connexion entre des clients et un serveur, où un client introduit des commandes auxquelles le serveur doit répondre. On va pouvoir manipuler les données de ce fichier grâce aux différentes classes, accompagnées de certaines méthodes, des constructeurs et des attributs.

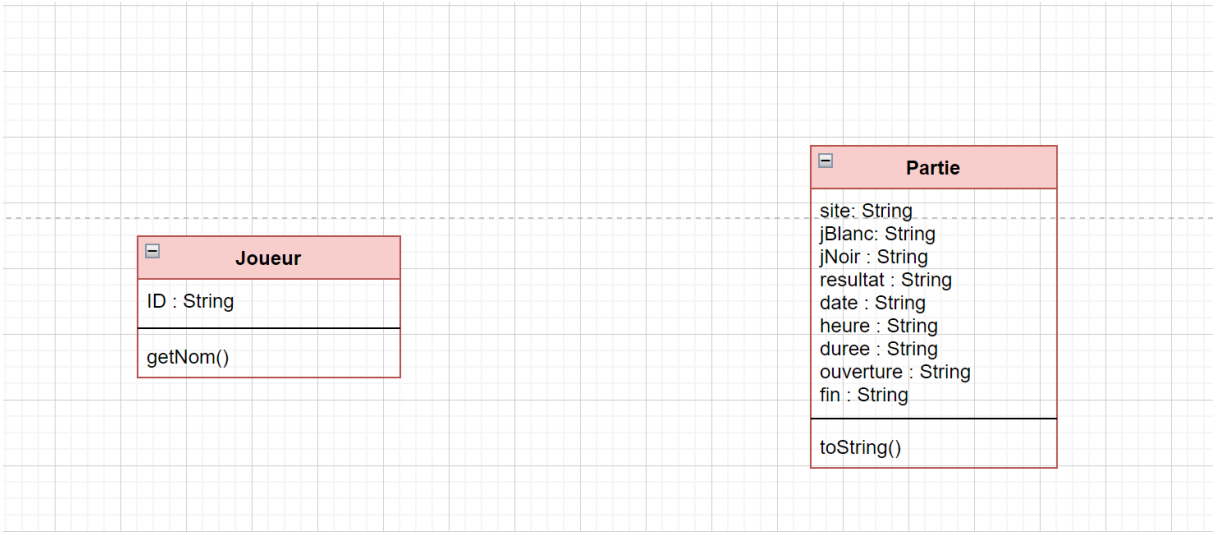
Avant tout, commençons par présenter c'est quoi « LiChess ». Lichess s'agit d'un site WEB conçu pour de jeu d'échecs créé en 2010 par un développeur français Thibault Duplessis. C'est le deuxième site d'échecs le plus fréquenté au monde, avec plus de 3 millions de parties jouées par jour. Le site permet de jouer des parties en direct contre des adversaires humains et programmes d'échecs à des intervalles de temps contrôlés. Il y aussi un système de classement proche du classement Elo : le classement **Glicko**(ça représente des méthodes de mesure de la force de joueurs dans les échecs).

Par la suite, on va présenter les différentes classes qu'on a créé et implémenté, commenter le code et essayer d'expliquer les méthodes distinctes, aussi bien qu'interpréter ces derniers à l'aide du cours.

Clothilde STRAINCHAMPS
Meryem BELASSEL
MI4-01

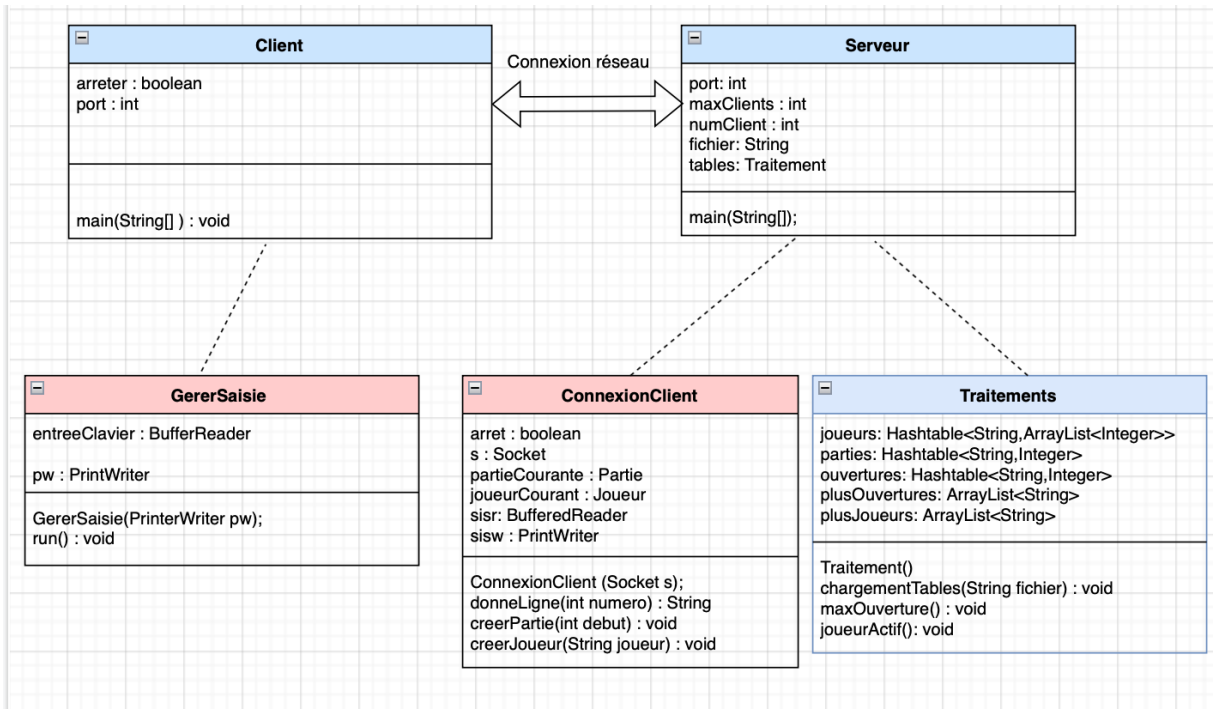
REPRÉSENTATION DES CLASSES IMPLÉMENTEES :

CLASSES UTILES :



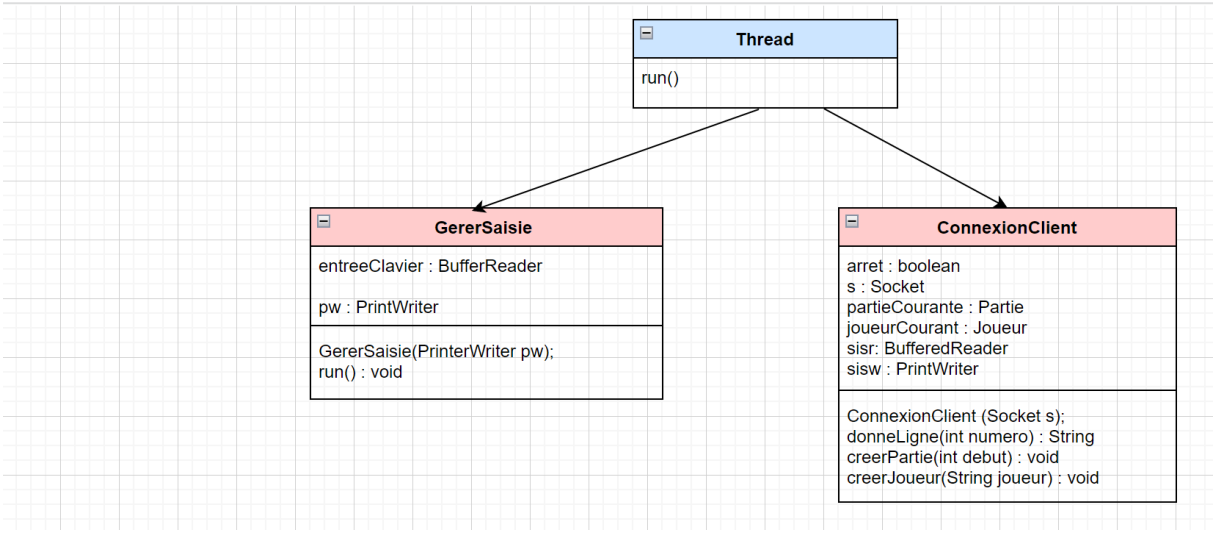
Les classes utiles sont les deux classes Joueur et Partie. Ces derniers vont être utilisés par la classe Serveur.

CLASSES SERVEUR ET CLIENT :

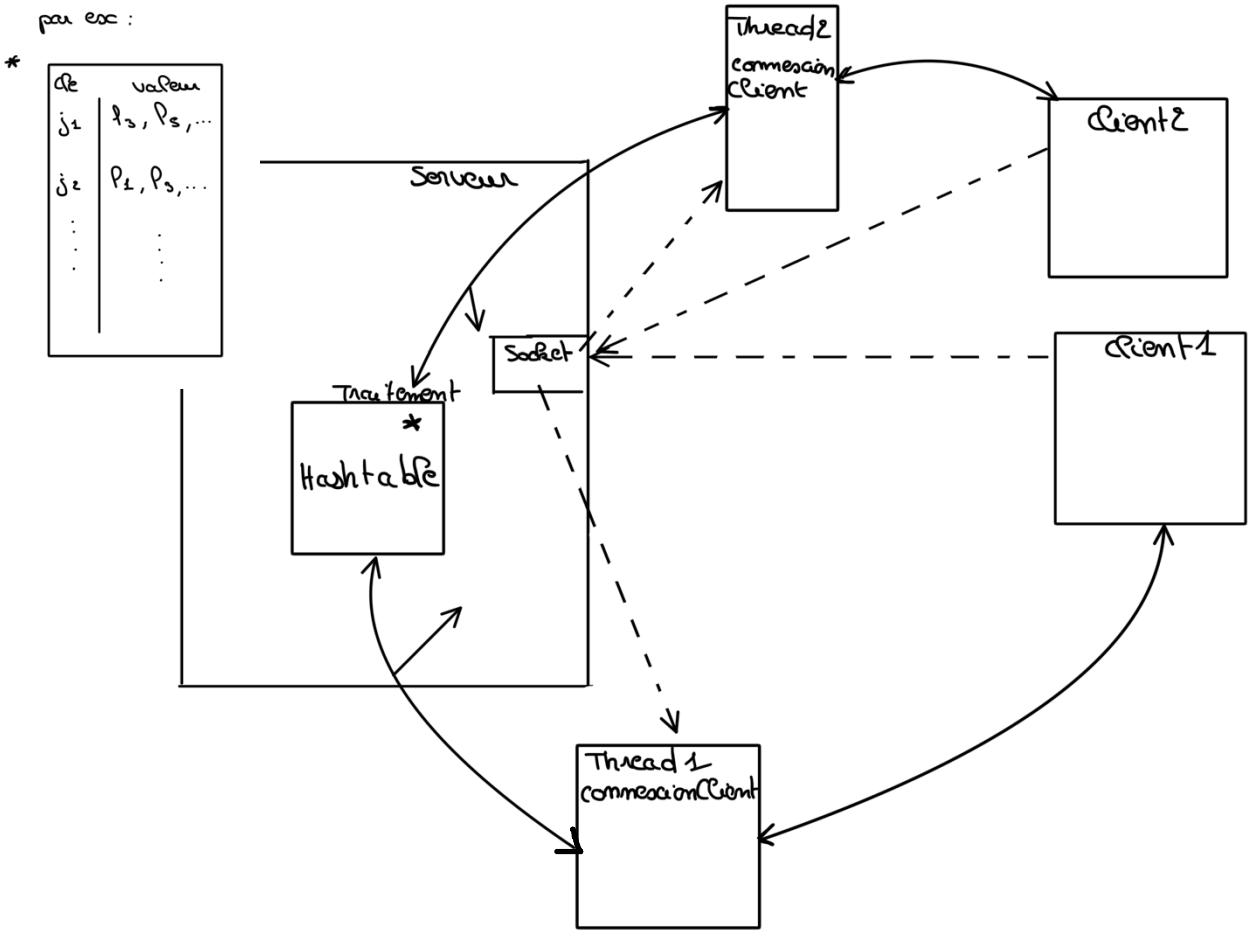


Clothilde STRAINCHAMPS
Meryem BELASSEL
MI4-01

LES THREADS :



DESCRIPTION DES STRUCTURES DE DONNEES :



Clothilde STRAINCHAMPS
Meryem BELASSEL
MI4-01

EXPLICATION DE NOS CHOIX :

Tout d'abord, nous avons commencé à implémenter la classe Traitement. Cette classe consiste à traiter une première fois le fichier, dans cette première lecture, nous remplissons des Hashtables, qui plus tard, nous servira à traiter une partie spécifique du fichier.

Dans ces Hashtables, nous avons choisi de stocker uniquement des lignes du fichier, pour ne pas saturer la RAM de la machine virtuelle JAVA avec de nombreux objets de type joueur ou partie. Cela optimise également le chargement du serveur.

Ensuite, le serveur est donc composé de l'objet Traitement. Ce qui veut dire que les Hashtables ne sont pas à sérialiser, car elles ne passent pas dans aucun flux.

Traitement est également composé d'attributs statiques comme l'ouverture la plus jouée ou le joueur le plus actif, parce qu'ils restent les mêmes pour le fichier et donc au moment de demande d'un client, aucune recherche dans les Hashtables n'est à faire, ce qui apporte une réponse instantanée au client, ce qui est plutôt avantageux pour le client.

Pour le serveur, nous avons choisi les Socket de type TCP/IP, car malgré le peu des données envoyées, on a préféré ce mode-ci par confort.

Notre Serveur est un serveur multi-clients, il dédit donc un thread client à chaque nouveau client se connectant. C'est donc lui qui reçoit les commandes envoyées par le client, traite sa demande puis renvoie une réponse, et en fonction de la demande, cela peut prendre plus ou moins du temps, car le client dédié au client va se servir des données des Hashtables pour savoir précisément quelle(s) ligne(s) doit il relire.

Concernant le client, nous avons fait quelque chose de basique. Le client peut entrer et recevoir des chaînes de caractères.

De plus, nous ne rencontrons pas de problème de concurrence, car aucun thread ne modifie une ressource.

Nous n'avons pas eu besoin de sérialiser ce que le serveur envoie au Client, car le serveur n'envoie que des chaînes de caractères.

Clothilde STRAINCHAMPS
Meryem BELASSEL
MI4-01

LA SPÉCIFICATION DE CLASSES PRINCIPALES ET DES MÉTHODES ESSENTIELLES :

CLASSES UTILES :

Ces classes nous servent à mettre en forme les données récupérées lors d'une lecture d'une partie du fichier.

Nous avons choisi de déclarer les attributs en final, parce qu'ils ne sont pas modifiables au cours du temps. Il n'est donc pas nécessaire de créer des setters. L'implémentation des getters peut être également négligée, car pour nous, ils n'ont pour but que l'affichage.

CLASSE CLIENT :

```
package Client;
import java.io.*;
import java.net.*;

import ClassesUtiles.Partie;
/**
 *
 * @author strainchampsclothilde
 */
class Client {
   static boolean arreter=false;//boolean servant à la déconnexion du client
   static int port = 8080;

   // Le client attend comme argument l'adresse du serveur (ex. : java Client 127.0.0.1 ou localhost pour l'exécuter)
   public static void main(String[] args) throws Exception {
      Socket socket = new Socket(args[0], port);
      System.out.println("SOCKET = " + socket);
      // illustration des capacites bidirectionnelles du flux:
      //BufferReader: object servant à lire les lignes du terminal côté client
      BufferedReader sisr = new BufferedReader(
                                 new InputStreamReader(socket.getInputStream()));
      //PrintWriter: object servant à envoyer une ligne de caractères
      PrintWriter sisw = new PrintWriter(new BufferedWriter(
                                 new OutputStreamWriter(socket.getOutputStream())),true);

      GererSaisie saisie=new GererSaisie(sisw);  // création d'un thread GererSaisie, avec pour paramètre notre PrintWriter
                                                 // déclaré au dessus
      saisie.start();// départ du thread
      String str;

      while(arreter!=true){ // tant que le client ne veut pas ce déconnecter en tapant "END"
         str = sisr.readLine(); // notre BufferReader lit ce qui nous est envoyé de la part du serveur
         System.out.println(str); // et nous l'affiche dans le terminal côté client
      }

      System.out.println("Déconnexion réussite!");
      //sisw.println("END") ;
```

Clothilde STRAINCHAMPS
Meryem BELASSEL
MI4-01

```
        // le client ferme les flux
        sisr.close();
        sisw.close();
        socket.close();
    }
}
// classe gérant la saisie dans le terminal côté client
class GererSaisie extends Thread{
    // attributs
    private BufferedReader entreeClavier;
    private PrintWriter pw;

    //constructeur standard
    public GererSaisie(PrintWriter pw){
        entreeClavier = new BufferedReader(new InputStreamReader(System.in));
        this.pw=pw;
    }

    //méthode run() définissant les "actions" du thread
    public void run(){
        String str;
        try{
            do {
                str = entreeClavier.readLine();
                pw.println(str);
            } while (!str.equals("sortie"));
        }catch(IOException e){e.printStackTrace();}
        Client.arreter=true;
    }
}
```

CLASSE SERVEUR :

ALGORITHME DE CONNEXION DU SERVEUR :

```
// Pour utiliser un autre port pour le serveur, l'exécuter avec la commande :
// java Serveur 8081
public static void main(String[] args) throws Exception {
    if (args.length != 0) {
        port = Integer.parseInt(args[0]);
    }
    try (// 1 - Ouverture du ServerSocket par le serveur
    ServerSocket s = new ServerSocket(port)) {
        System.out.println("SOCKET ECOUTE CREE => " + s);
        System.out.println("Tables chargées");// signal côté serveur pour nous indiqué que le serveur est prêt à être
                                              // utilisé et peut acceuillir un client
        while (numClient < maxClients) {//tant que le serveur n'est pas "plein"n 50 connexion
            /*
            * 2 - Attente d'une connexion client (la méthode s.accept() est bloquante
            * tant qu'un client ne se connecte pas)
            */
            Socket soc = s.accept();
            /*
            * 3 - Pour gérer plusieurs clients simultanément, le serveur attend que les
            * clients se connectent,
            * et dédie un thread à chacun d'entre eux afin de le gérer indépendamment des
            * autres clients
            */
            ConnexionClient cc = new ConnexionClient(soc);
            System.out.println("NOUVELLE CONNEXION - SOCKET => " + soc);
            numClient++;
            cc.start();
        }
    }
}
```

Clothilde STRAINCHAMPS
Meryem BELASSEL
MI4-01

CLASSE CONNEXIONCLIENT :

Cette classe est une sous-classe de thread, qui nous sert à gérer les commandes envoyées par le client et par la suite lui renvoyer les informations demandées. Le thread ConnexionClient est dédié par le serveur, à un client unique, ce qui veut dire que chaque client à son thread.

☆ Algorithme pour renvoyer une ligne précise :

```
public String donneLigne(int numero) throws IOException{//méthode qui retourne la ligne à analyser
    BufferedReader reader = new BufferedReader(new FileReader(Serveur.fichier));
    for(int i = 0; i < numero; i++){
        reader.readLine();
    }

    String ligneVoulue = reader.readLine();
    return ligneVoulue;
}
```

Son fonctionnement est qu'il va lire rapidement les lignes du fichier, et se positionner sur la ligne que l'on souhaite.

☆ Algorithme pour initialiser une partie courante :

```
public void creerPartie(int debut) throws IOException {// méthode qui initialise l'attribut partieCourante
    String ligne;
    String site = "";
    String jBlanc = "";
    String jNoir = "";
    String resultat = "";
    String date = "";
    String heure = "";
    String ouverture = "";
    String duree = "";
    String fin = "";

    for (int i = debut-1; i <= debut + 16; ++i) {
        ligne = donneLigne(i);

        // pour récuperer le numéro de la partie
        Pattern pSite = Pattern.compile("Site " + "\\\"" + "https://lichess.org/" + "[a-zA-Z_0-9]*" + "\\\"");
        Matcher mSite = pSite.matcher(ligne);

        // pour prendre les joueurs
        Pattern pJoueurBlanc = Pattern.compile("White " + "\\\"" + ".*" + "\\\"");
        Matcher mJoueurBlanc = pJoueurBlanc.matcher(ligne);

        Pattern pJoueurNoir = Pattern.compile("Black " + "\\\"" + ".*" + "\\\"");
        Matcher mJoueurNoir = pJoueurNoir.matcher(ligne);

        // pour le résultat
        Pattern pResultat = Pattern.compile("Result " + "\\\"" + "[_0-9]*" + "-" + "[_0-9]*" + "\\\"");
        Matcher mResultat = pResultat.matcher(ligne);

        // Pour le jour de la partie
        Pattern pDate = Pattern.compile("UTCDate " + "\\\"" + "[_0-9]*" + "." + "[_0-9]*" + "." + "[_0-9]*" + "\\\"");
        Matcher mDate = pDate.matcher(ligne);

        // pour l'heure de la parite
        Pattern pHeure = Pattern.compile("UTCTime " + "\\\"" + "[_0-9]*" + ":" + "[_0-9]*" + ":" + "[_0-9]*" + "\\\"");
        Matcher mHeure = pHeure.matcher(ligne);

        // pour l'ouverture
        Pattern pOuverture = Pattern.compile("Opening " + "\\\"" + ".*" + "\\\"");
        Matcher mOuverture = pOuverture.matcher(ligne);
```

Clothilde STRAINCHAMPS
Meryem BELASSEL
MI4-01

```
    // pour la durée de la partie
    Pattern pDuree = Pattern.compile("TimeControl " + "\\\"" + "[_0-9]*" + "[^\s]" + "[_0-9]*" + "\\\"");
    Matcher mDuree = pDuree.matcher(ligne);

    // pour la fin
    Pattern pFin = Pattern.compile("Termination " + "\\\"" + ".*" + "\\\"");
    Matcher mFin = pFin.matcher(ligne);

    if (mSite.find()) {
      site = mSite.group().substring(26, mSite.group().length() - 1);
    }
    if (mJoueurBlanc.find()) {
      jBlanc = mJoueurBlanc.group().substring(7, mJoueurBlanc.group().length() - 1);
    }
    if (mJoueurNoir.find()) {
      jNoir = mJoueurNoir.group().substring(7, mJoueurNoir.group().length() - 1);
    }
    if (mResultat.find()) {
      resultat = mResultat.group().substring(8, mResultat.group().length() - 1);
    }
    if (mDate.find()) {
      date = mDate.group().substring(9, mDate.group().length() - 1);
    }
    if (mHeure.find()) {
      heure = mHeure.group().substring(9, mHeure.group().length() - 1);
    }

    if (mOuverture.find()) {
      ouverture = mOuverture.group().substring(9, mOuverture.group().length() - 1);
    }
    if (mDuree.find()) {

      duree = mDuree.group().substring(13, mDuree.group().length() - 1);
    }
    if (mFin.find()) {
      fin = mFin.group().substring(13, mFin.group().length() - 1);
    }
  }
  partieCourante = new Partie(site, jBlanc, jNoir, resultat, date, heure, duree, ouverture, fin);
}
```

Cette méthode permet d'initialiser l'attribut de la la partie courante. Son fonctionnement est plutôt simple : On récupère la ligne à analyser, grâce à l'algorithme précédent, puis regarder si cette ligne est compatible avec l'un des patterns qu'on définit, qui eux sont spécifiques à un certain type de données à récupérer. Une fois toutes les lignes voulues sont lues, on initialise notre attribut avec les éléments récupérés.

Clothilde STRAINCHAMPS
Meryem BELASSEL
MI4-01

☆ Algoritime pour initialiser le joueur :

```
public void creerJoueur(String joueur) throws IOException{//méthode qui permet d'initialiser l'attribut joueurCourant
  String ligne;
  String site = "";
  ArrayList<String> listeParties = new ArrayList<String>();

  for (int i = 0; i < Serveur.tables.joueurs.get(joueur).size() ; ++i) {
    ligne = donneLigne(Serveur.tables.joueurs.get(joueur).get(i));

    // pour récuperer le numéro de la partie
    Pattern pSite = Pattern.compile("Site " + "\\\"" + "https://lichess.org/" + "[a-zA-Z_0-9]*" + "\\\"");
    Matcher mSite = pSite.matcher(ligne);

    if (mSite.find()) {
      site = mSite.group().substring(26, mSite.group().length() - 1);
      listeParties.add(site);
    }
  }
  System.out.println("fin");
  joueurCourant = new Joueur(joueur,listeParties);
}
```

Cette méthode suit le même principe que la méthode ci-dessus, on récupère juste l'information associée au joueur et ion initialise son attribut avec les éléments qu'on récupère.

→La réception des commandes de la part du client est gérée par la méthode run() du thread grâce à un switch.

✂ CLASSE TRAITEMENT :

Dans ce projet les donées recueuillit sont stockées dans des hashtables. La classe Traitement lit le fichier une première fois en entier avant que le thread dédié au client le relise une deuxième foissur une partie bien précise.

joueurs est une hashtable ayant pour clé un nom de joueur et pour valeur les lignes des parties dans lesquelles ils ont joué.

parties est une hashtable ayant pour clé un nom de partie et pour valeur la ligne où elle commence.

ouvertures est une hashtable ayant pour clé un nom d'ouverture et pour valeur le nombre de fois qu'elle à été jouée.

joueurActif est une hashtable ayant pour clé un nom de joueur et pour valeur le nombre de partie qu'il à joué.

plusOuverture et plusJoueur sont des arrayList contenant l'ouverture la plus jouée et le joueur ayant joué le plus de parties.

Clothilde STRAINCHAMPS
Meryem BELASSEL
MI4-01

☆ Algorithme pour remplir les hashtables :

```
// méthode remplissant nos hashtables
public void chargementTables(String fichier)throws IOException{
    //initialisation des hashtables
    String ligne, joueur1, joueur2, partie, ouverture;
    ArrayList<Integer> ligneCite = new ArrayList<Integer>();

    //lecture du fichier et remplissage des tables
    BufferedReader f=new BufferedReader(new FileReader(fichier));
    int comptLigne = 0;
    while ((ligne=f.readLine())!=null)
    {
        comptLigne =comptLigne + 1;
        //pour prendre les lignes des joueurs
        Pattern pJoueurBlanc = Pattern.compile("White "+"\\\""+".*"+"\\\"");
        Matcher mJoueurBlanc = pJoueurBlanc.matcher(ligne);
        Pattern pJoueurNoir = Pattern.compile("Black "+"\\\""+".*"+"\\\"");
        Matcher mJoueurNoir = pJoueurNoir.matcher(ligne);

        //pour prendre les lignes des débuts de partie
        Pattern pSite = Pattern.compile("Site "+"\\\""+"https://lichess.org/"+"[a-zA-Z_0-9]*"+"\\\"");
        Matcher mSite = pSite.matcher(ligne);

        // pour l'ouverture
        Pattern pOuverture = Pattern.compile("Opening " + "\\\"" + ".*" + "\\\"");
        Matcher mOuverture = pOuverture.matcher(ligne);

        while(mJoueurBlanc.find())
        {
            joueur1 = mJoueurBlanc.group().substring(7, mJoueurBlanc.group().length()-1);
            if (joueurs.containsKey(joueur1))
            {
                joueurs.get(joueur1).add(comptLigne-2);
            } else
            {
                ArrayList<Integer> lignesJoueurs = new ArrayList<Integer>(Arrays.asList(comptLigne-2));
                joueurs.put(joueur1, lignesJoueurs);
            }
            if (joueursActif.containsKey(joueur1))
            {
                int rep = joueursActif.get(joueur1);
                rep = rep +1;
                joueursActif.remove(joueur1);
                joueursActif.put(joueur1, rep);
```

Pour prendre les informations précise que nous voulions dans le fichier, nous avons opté pour une approche syntaxique pércise avec des expressions régulières, car nous avons remarqué que les données du fichier sont invariablement organisées d'une certaine façon dans tout le fichier.

La lecture du fichier se fait avec un BufferReader.

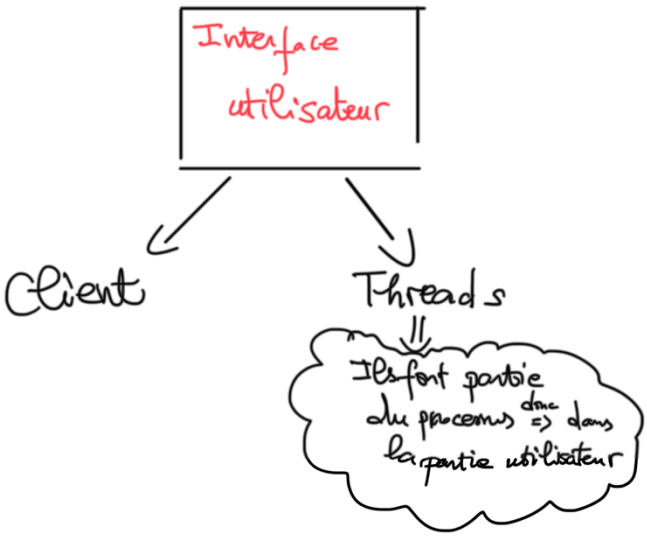
```
            } else{
                joueursActif.put(joueur1, 1);
            }
        }

        while(mJoueurNoir.find())
        {
            joueur2 = mJoueurNoir.group().substring(7, mJoueurNoir.group().length()-1);
            if (joueurs.containsKey(joueur2))
            {
                joueurs.get(joueur2).add(comptLigne-3);
            } else
            {
                ArrayList<Integer> lignesJoueurs = new ArrayList<Integer>(Arrays.asList(comptLigne-3));
                joueurs.put(joueur2, lignesJoueurs);
            }
            if (joueursActif.containsKey(joueur2))
            {
                int rep = joueursActif.get(joueur2);
                rep = rep +1;
                joueursActif.remove(joueur2);
                joueursActif.put(joueur2, rep);
            } else{
                joueursActif.put(joueur2, 1);
            }
        }


        while(mSite.find())
        {
            partie = mSite.group().substring(26, mSite.group().length()-1);
            if (parties.containsKey(partie))
            {
                parties.get(partie);
            } else{
                parties.put(partie, comptLigne);
            }
        }
        while(mOuverture.find())
        {
            ouverture = mOuverture.group().substring(9, mOuverture.group().length() - 1);
            if (ouvertures.containsKey(ouverture))
            {
                int rep = ouvertures.get(ouverture);
                rep = rep +1;
                ouvertures.remove(ouverture);
                ouvertures.put(ouverture, rep);
            } else{
                ouvertures.put(ouverture, 1);
            }
        }
    }
    f.close();
}
```

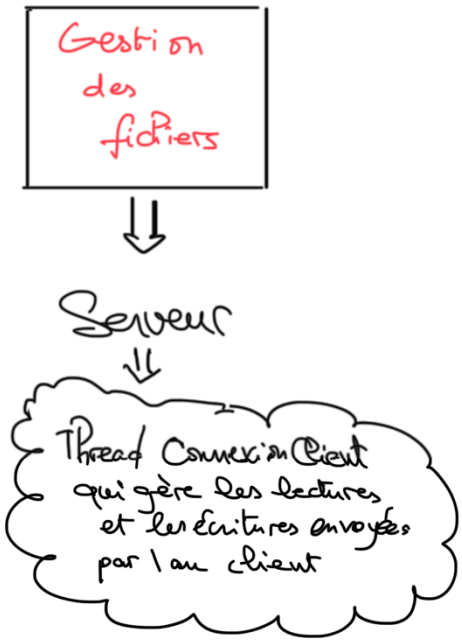
Les hashtables sont ensuite remplient en fonction de quel pattern est détecté.

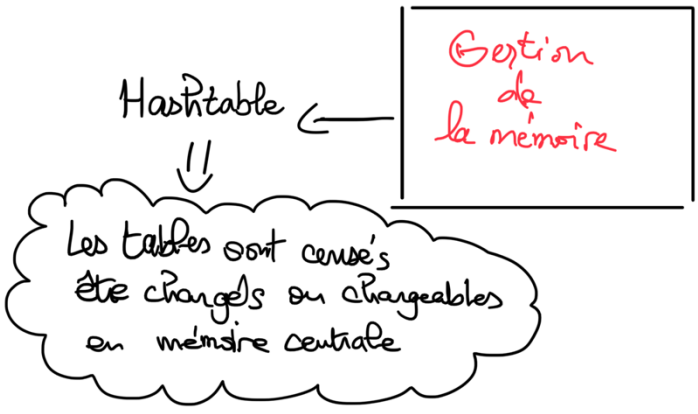
Des méthodes maxOuvertures et joueurActif sont également implémentés pour trouver dans une hashtable la clé ayant la plus grande valeur.

Clothilde STRAINCHAMPS
Meryem BELASSEL
MI4-01

ARCHITECTURE LOGICIEL DÉTAILLÉ







Clothilde STRAINCHAMPS
Meryem BELASSEL
MI4-01

TEST

Au cours de ce projet de nombreux tests ont été réalisés pour tester indépendamment les algorithmes présenter ci-dessus, avant qu'ils ne soient ajoutés au projet final.

Pour nos tests nous avons pris une petite partie du fichier à analiser (environ 3 parties) pour que le temps de traitement (lecture) à chaque fois ne soit pas trop long.

Le premier test était pour lire une première fois le fichier et remplir une hashtable. La difficulté à été de choisir comment récupérer précisément les informations que nous voulions. Nous avons testé deux manières de faire : une avec un objet de type StringTokenizer et ses méthodes avec des délimiteurs pour les chaînes de caractères, mais cela ne marchait pas, les délimiteurs étaient mal réglés et nous ne trouvions pas la bonne méthode pour fixer ce problème, d'où par la ensuite l'utilisation d'expression régulière sur chaque ligne.

Le deuxième de test était pour créer un objet Partie, donc récupérer la ligne où commence une partie précisement (récupération dans la hashtable de cette information) et à partir de là remplir notre objet Partie correctement, pour la partie demandée par l'utilisateur. C'est alors là que nous avons eu un biais de confirmation. Pour cause un fichier de données réduit a trop peu de parties et le fait de demander souvant la même partie. Nous nous sommes rendu compte de ce problème lors des tests finaux, (au moment de tester notre serveur en général).

Puis pour finir quelques tests réseaux entre le serveur et ses clients, en faisant imprimer toutes les informations que l'un envoie à l'autre et vis et versa...

Clothilde STRAINCHAMPS
Meryem BELASSEL
MI4-01

PROBLÈMES RENCONTRÉES :

1. Quelles informations stocker dans les Hashtables de la classe traitement ?
   L'une des premières questions que nous nous sommes posés en commençant le projet était : comment organiser nos données (joueurs et parties) se trouvant dans le fichier ? Nous avons tout d'abord pensé, que stocker des objets complets dans les Hashtables pour une clé quelconque donnée « était » une bonne idée. Cependant, ayant implémenté cette idée avant nous, ont saturé la RAM de la machine virtuelle. Résultat : point de départ. Mais ensuite en en discutant lors du Tp, on a pu résoudre le problème.

2. Construction des objets traités par le thread Client du serveur :
   À cause des tests faussés, nous avons mis beaucoup de temps à résoudre ce problème, qui était de bien isoler une ligne précise à partir des millions de lignes de ce fichier.

3. Établir la connexion entre le serveur et les clients :
   Ce problème est lié au problème n°2, car du coup, cela nous renvoyait des choses fausses ou rien du tout, et bloquait le serveur dans certains cas.

LES AMÉLIORATIONS :

I. Dédier plusieurs threads pour remplir les Hashtables → gain de temps.
II. Créer une base de données à la première lecture → gain de temps au chargement du serveur (plus de relecture aux prochaines mises en route) + libération de la RAM

-----CONCLUSION-----

Dans sa globalité, le programme développé fonctionne. Bien que certaines fonctionnalités soient à ameliorer.