# Projet Systèmes et Algorithmiques Repartis

Université Paris-Dauphine


Kacem BOUHMADI
Michael GOLETTO
Salim GUENNOUNI
Raphael LUMBROSO


Département MIDO
Master 1 Informatiques des Organisations
Parcours MIAGE 2011/2012
UE : Systèmes et Algorithmiques Repartis
Enseignant : Joyce El Haddad

# Contenu

## Introduction

Nous étudions dans ce projet la réalisation de threads et des mécanismes de synchronisation à travers un modèle Producteur/Consommateur, un exemple classique de synchronisation de 2 threads, l'un qui produit des informations qu'il dépose dans un tampon de taille bornée, l'autre qui les retire une à une pour les consommer. Ce modèle pour être généralisé à plusieurs threads qui produisent et consomment en même temps.

Si l'on considère chaque case du tampon comme une ressource, le but est de synchroniser les différents threads de sorte à ne pas écraser de l'information dans une case, ou ne pas retirer d'une case de l'information inexistante. De plus, on souhaite que les informations produites soient consommées dans le même ordre. Pour ce faire, le tampon est géré de façon circulaire.

Le modèle Producteur/Consommateur est un modèle d'interaction sous-jacent au modèle Client/Serveur.

## Modélisation et diagramme de classes

Dans cette partie, nous traiterons en détail notre modélisation technique du projet en présentant un diagramme de classes.

Le modèle Producteur/Consommateur que nous devions réaliser pour le projet implique forcément une répartition des classes dans deux environnements différents : Serveur et Client. Nous avons donc présenté ces deux environnements comme étant deux packages différents. Par contrainte de clarté nous avons décidé de mettre les classes propres à l'interface graphique des clients dans un 3ème package « Affichage ».

Figure 1Diagramme de classes

## Package « Serveur »



Figure 2 : Package Serveur

Ce package contient les classes et interfaces qui doivent être présentes coté serveur.

L'interface TamponInterface doit être compilée et partagée entre le serveur et les différents producteurs et consommateurs. Elle permet de définir les méthodes dîtes « primitives » que peuvent utiliser les clients via un objet distant, on trouve notamment les définitions suivantes :

- *DemandeInsertion* : permet de faire une demande d'insertion dans le tampon ;
- *Extraire* : permet d'extraire un message du tampon ;
- *Inserer* : permet d'insérer dans le tampon le message passé en paramètre ;
- *RetirerReservation* : permet de retirer une réservation faite par un producteur dans le tampon.

En plus d'implémenter l'interface TamponInterface, la classe TamponInterfaceImpl contient les attributs suivants :

- fileAttente : c'est une liste chainée qui permet de réserver de la place pour les producteurs ayants fait une demande d'insertion. L'algorithme utilisé est expliqué plus en détail dans la deuxième partie de ce rapport ;
- tampon : c'est le tampon qui contient les messages produits ;
- in : indice du message à insérer dans le tampon ;
- out : indice du message à extraire dans le tampon ;
- taille : taille du tampon ;
- nbmess : nombre de messages dans le tampon ;
- nbmessReserves : nombre de cases réservés dans le tampon ;

Les méthodes de cette classe sont juste l'implémentation des méthodes définies dans l'interface.

Pour des raisons de simplicité nous avons décidé de créer une « structure complexe » pour les messages. Il était nécessaire pour le bon déroulement de notre algorithme que chaque message possède des

caractéristiques propres à lui. Nous avons donc décidé de créer une classe Message contenant les attributs suivants :

- debut : c'est la date de création du message ;
- idClient : l'identifiant du producteur du message ;
- message : c'est le message sous forme de chaine de caractères ;
- timeout : durée d'expiration d'un message.

Cette classe possède deux constructeurs, le premier prend en paramètre l'identifiant du producteur et le deuxième prend en paramètre le message en plus de l'id. Nous n'avons pas jugé nécessaire de montrer ici les getters/setters de cette classe.

Enfin, la classe TamponServeur permet juste d'orchestrer le serveur en lançant l'implémentation de l'interface et en publiant le nom de l'objet dans le service de nommage.

## Package « Client »



Figure 3 : Package Client

Ce package contient les classes Producteur et Consommateur, qui se chargent respectivement de produire des messages et de les consommer.

Dans la classe Producteur on trouve les attributs suivants :

- Id : c'est un identifiant unique propre à chaque producteur. Il nous permet dans notre algorithme de réserver une place dans le tampon pour l'insertion d'un nouveau message ;
- Talon : c'est l'objet distant qu'on récupère via RMI. Il nous permet d'utiliser les méthodes de l'interface *TamponInterface* ;
- url : c'est l'url qui nous permet d'accéder à l'objet distant en utilisant RMI.

Les méthodes de cette classe sont :

- *Producteur* : permet d'instancier un nouveau producteur ;
- *demandeInserer* : cette méthode permet de faire une demande d'insertion dans le tampon. Elle renvoie un True si le producteur est autorisé à produire et False dans le cas contraire ;
- *produire* : permet d'insérer dans le tampon le message passé en paramètre ;
- *retirerReservation* : permet retirer la réservation d'un producteur. Elle est utile dans le cas où l'utilisateur clique sur le bouton « annuler » après avoir fait une demande d'insertion.
-

La classe Consommateur contient quant à elle deux attributs : « talon » et « url ». Ils jouent le même rôle que dans la classe Producteur sauf qu'ils sont utilisés ici pour consommer des messages au lieu de les produire. De

même, les méthodes « Consommer » et « Consommateur » permettent, respectivement, de consommer un message du tampon et d'instancier un nouveau consommateur.

## Package « Affichage »

Figure 4 : Package Affichage

Dans ce package nous avons mis les classes permettant la réalisation de l'interface graphique des clients. Nous n'avons pas jugé nécessaire de les présenter ici car elles n'ont aucun intérêt autre celui de réaliser les IHM.

## Algorithme de gestion implémenté

Nous avons cherché un algorithme qui donnait des résultats le plus satisfaisant possible dans les cas extrêmes tel que celui où les producteurs font des demandes d'insertions mais n'envoient jamais leurs messages.

La réservation d'une case définie dans le tampon lors d'une demande d'insertion induit un risque d'incohérence dans la gestion circulaire du tampon. Dans le cas où deux producteurs font des demandes d'insertions, mais uniquement le second envoie son message, il ne pourra être consommé qu'à partir du moment où son prédécesseur aura envoyé son message, et que celui-ci aura été consommé. La consommation est donc bloquée pour un temps indéterminé… Même si l'on décide de retirer la case réservée au bout d'un certain temps, cette case vide devra être « sautée » lors de la consommation, ce qui implique une gestion plus complexe des indices.

Nous avons donc opté pour une solution implémentant deux tampons : le premier contient les messages envoyés par les producteurs (nous l'appellerons *Tmess*), le second (que nous appellerons *Tres*) enregistre les réservations que les producteurs ont effectués avant qu'ils n'envoient leurs messages. Afin de permettra la mise en place de cet algorithme, nous avons utilisé les variables suivantes :

```
Message[] tampon; Message[] tampon;
LinkedList<Message> fileAttente; // FIFO
int taille, in, out, nbmess, nbmessReserves;
```

Chacun de ces tampons contient des objets de type *Message* qui contiennent les informations relatives à un message. Cet objet possède les attributs suivants :

- début et fin (avec fin = début + timeout)
- message de type String qui contient le message
- idClient de type long qui représente l'identifiant unique du client

```
private long idClient; // Identifiant unique du client
private String message;
private Date debut;
private Date fin; // fin = debut + timeout
```

Lorsqu'un producteur envoie son message, on vérifie que celui-ci est bien présent dans la liste des producteurs ayant fait une demande d'insertion.

Nous déterminons la taille théorique du remplissage du tampon grâce à la somme des tailles de *Tmess* et de *Tres*.

Dans le cas, où il y a une demande d'insertion alors que le tampon est théoriquement plein, on procède à un retrait du plus ancien producteur ayant effectué sa demande si un délai (définit dans le programme) est dépassé.

Dans notre code cela se traduit par :

```
if((nbmess + nbmessReserves) == taille && !fileAttente.isEmpty())
```

La première condition permet de vérifier si le *Tmes* est plein, si c'est la cas on vérifie qu'il y a des demandes d'insertions. Si le test est positif, on vérifie si le premier élément a son *timeout* de dépasser :

```
if(it.next().getFin().before(new Date(System.currentTimeMillis()) )){
   fileAttente.removeFirst();
   nbmessReserves--;
```

SI c'est le cas, on le retire sa demande d'insertion et on décrémente le nombre d'emplacement réservé.

Code complet de la demande d'insertion :

```
public boolean demandeInsertion(long c) throws RemoteException{
      synchronized(tampon){
         synchronized (fileAttente){
            if((nbmess + nbmessReserves) == taille &&
!fileAttente.isEmpty()){
               Iterator<Message> it = fileAttente.iterator();
               while(it.hasNext()){
                  if(it.next().getFin().before(new
Date(System.currentTimeMillis()))){
                     fileAttente.removeFirst();
                     nbmessReserves--;
                     System.out.println ("*** Serveur libere une place
pour " + c);
                     break;
                  }
               }
            }

            if((nbmess + nbmessReserves) < taille){
               Message m = new Message(c);
               fileAttente.add(m);
               nbmessReserves++;
               System.out.println ("*** Serveur reserve une place pour "
+ c);
               return true;
            }
            return false;
         }
      }
   }
```

Si jamais le producteur envoie son message alors sa réservation a été supprimée, on lui retourne un message d'erreur qui indique que son message n'a pas pu être inséré.

Dans le cas où la demande d'insertion est toujours active, on recherche dans la file d'attente la réservation qui correspond à ce producteur.

```
public boolean inserer(long idClient, String message) throws
RemoteException{
     synchronized(tampon){
         synchronized (fileAttente){
            if(!fileAttente.isEmpty()){
               Iterator<Message> it = fileAttente.iterator();
               Message m;
               while(it.hasNext()){
                  m = it.next();
                  if(m.getIdClient() == idClient){
                     tampon[in] = m;
                     it.remove();
                     m.setMessage(message);
                     in = (in + 1) % taille;
                     nbmess++;
                     nbmessReserves--;
                     System.out.println ("*** Serveur insere un message
pour " + idClient);
                     return true;
                  }
               }
            }
            return false;
         }
      }
   }
```

L'extraction de messages vérifie uniquement que *Tmes* ne soit pas vide.

```
public String extraire() throws RemoteException{
      synchronized(tampon){
         if(nbmess > 0){
            Message m = tampon[out];
            out = (out + 1) % taille;
            nbmess--;
            System.out.println ("*** Consommateur extrait le message de "
+ m.getIdClient() +  "--> nouvelle taille = " + nbmess);
            return m.getMessage();
         }
         return null;
      }
   }
```

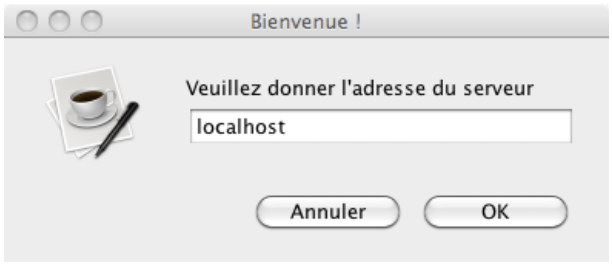
## Interface graphique

Une interface graphique a été réalisée pour faciliter l'utilisation de l'application. Elle sera déployée chez les clients (producteurs et consommateurs) et pourra remplacer l'interface en ligne de commande pour ces derniers. En ce qui concerne le serveur, un simple journal sera affiché en temps réel sur un terminal qui affichera les différentes requêtes des clients.

### Lancement de l'application *(Saisi de l'IP Serveur)*

Au premier lancement de l'application, le programme demande à l'utilisateur Client de saisir l'adresse IP du serveur distant.

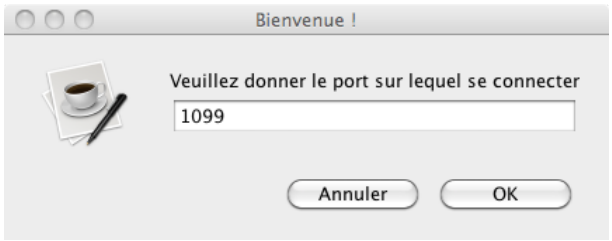
Valeur par défaut : localhost (127.0.0.1).




### Saisi du port Serveur

La seconde IHM demande à l'utilisateur de saisir le numéro de port auquel le serveur est connecté.

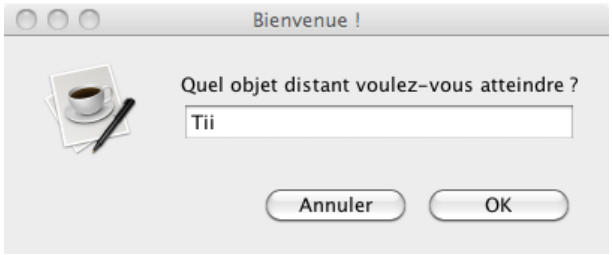
Valeur par défaut : 1099.




### Saisi du nom d'objet distant

Cette fenêtre permet récupérer le nom d'objet distant à joindre. Celui-ci fait référence à l'objet Tampon mis à disposition par le serveur.

Valeur par défaut : Tii.



## Choix Producteur/Consommateur

L'utilisateur choisi son mode de fonctionnement, soit un producteur qui enverra des messages au serveur, soit un consommateur qui pourra lire les messages envoyés par ces derniers.


## Scénario 1 : Producteur

Dans le cas du Producteur, l'utilisateur peut décider de produire à tout moment en appuyer sur le bouton « PRODUIRE ». Celui-ci déclenche une demande d'autorisation d'insertion gérée par la méthode *boolean demandeInsertion(long idClient)*.


## Demande d'insertion accepté

Dans le cas ou le producteur peut réservé une case dans le tampon, une autorisation lui est attribué, il peut ainsi envoyer son message.

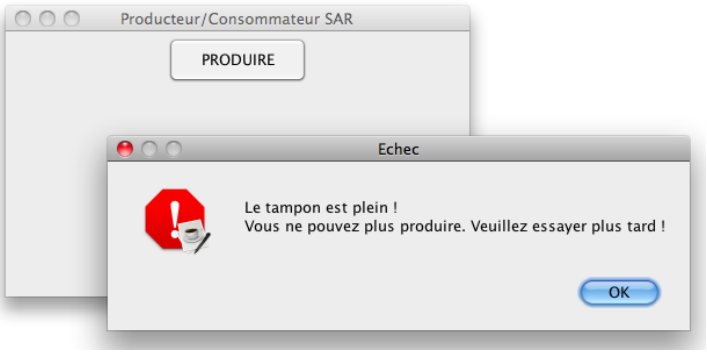
Une fenêtre « pop-up » confirme à l'utilisateur l'autorisation de production et lui demande de saisir son message texte. Il peut à présent envoyer ce dernier, il dispose d'un certain temps limité « timeout » ou la case lui est réservée. Une fois ce « timeout » écoulé, il n'y a aucune garantie que le message puisse être envoyé. Ce cas sera détaillé dans la suite.


Le timeout est défini par défaut à 10s dans la classe Message :

```
private static final int timeout = 10000;
```

## Demande d'insertion refusé

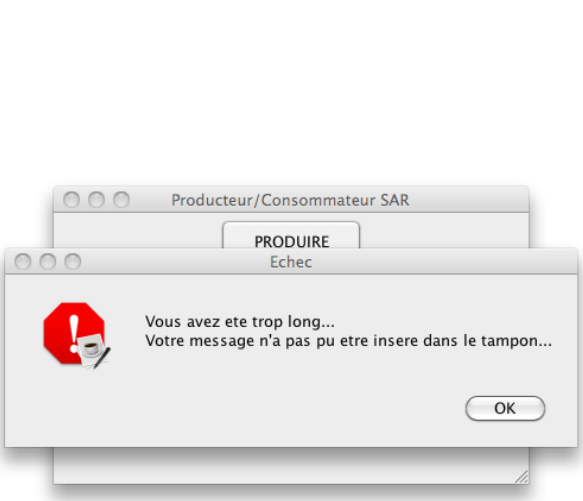
Dans le cas ou le tampon est plein, ou que le nombre de réservations maximum ait été atteint, le programme vérifie en premier si la réservation en tête de la file d'attente n'a pas expiré, dans le cas échéant il renvoi un refus au producteur, sinon il nettoie la tête de liste et confirme une autorisation de production.




## Cas extrême : autorisation accepté, insertion impossible *(scénario vol de place)*

Un des cas le plus important géré est le cas ou deux producteurs essaient de produire en même temps, sauf qu'il ne reste qu'une seule place dans le tampon.

- Producteur 1 effectue une demande d'insertion mais n'envoie pas son message
- Producteur 2 effectue aussi une demande d'insertion, mais le tampon théorique étant à présent plein, le programme va vérifier si la réservation précédente faite par Producteur 1 a expiré. Si ce n'est pas le cas, le Producteur 2 reçoit un refus lui notifiant que le tampon est plein. Si la réservation a expiré, celle-ci est supprimée et une autorisation est octroyée au Producteur 2, qui a la possibilité d'envoyer son message.
- Producteur 1 a toujours la possibilité de saisir son message puisqu'il est resté bloqué sur la fenêtre de saisie, sauf que le tampon est à présent plein, son message sera refusé puisqu'aucune réservation ne lui correspond. Un avertissement est affiché à l'utilisateur « Vous avez été trop long… ».





## Optimisation du cas Producteur

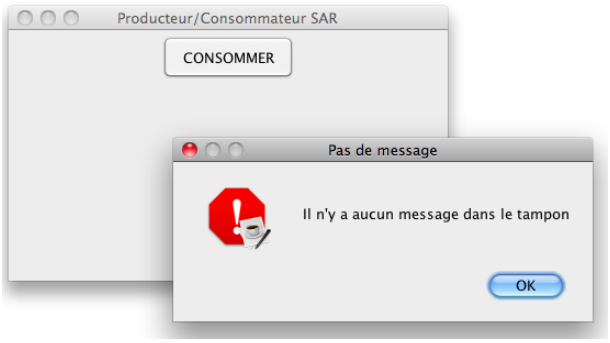
Quelques optimisations ont été aussi réalisées du côté du Producteur afin d'éviter un maximum de cas indésirables, comme le cas ou ce dernier saisit un message vide, un avertissement est levé.


## Scénario 2 : Consommateur

Le comportement de l'IHM Consommateur est différent de celle du producteur, et plus simple. Le l'utilisateur peut choisir de consommer un message à tout moment en appuyant sur le bouton « CONSOMMER ». Aucune demande d'autorisation n'est réellement effectuée puisque dans le cas ou le tampon est vide, un avertissement le signale, sinon le message est retourné.

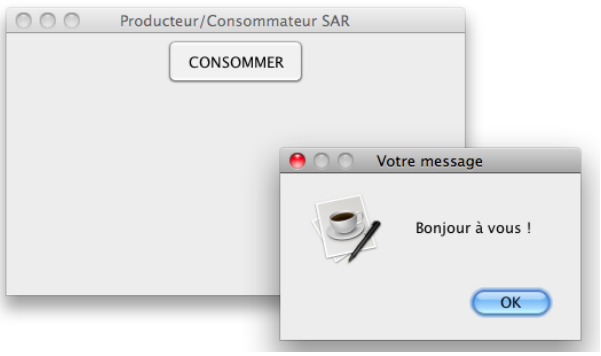

## Aucun message à extraire *(tampon vide)*

Un simple message d'avertissement est signalé si une demande d'extraction a été effectuée sur un tampon vide.



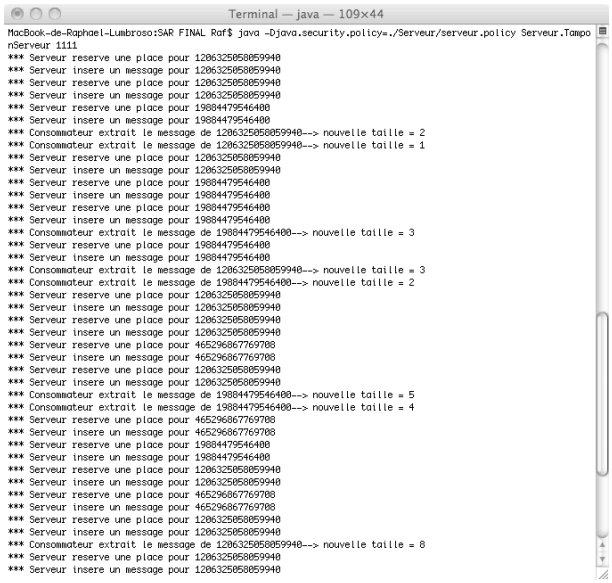

## Extraction d'un message *(tampon non-vide)*

L'extraction d'un message est géré par la méthode *String extraire()*, qui retourne le message consommé si le tampon n'est pas vide. Une simple fenêtre permet d'afficher son contenu.



## Serveur

Du côté du serveur, un simple « log » des requêtes est affiché, il permet de lister les différentes actions faites sur le serveur.




## Documentation utilisateur

L'application est organisée en 3 répertoires (packages) :

- ./Serveur
- ./Client
- ./Affichage

Le premier concerne le serveur, les deux derniers concernent le client.

### Serveur

Afin de faciliter l'exécution du serveur, un script *MultiServeur.sh* a été crée pour automatiser l'exécution. Ce fichier ne peut être utilisé que sur les postes de travail de l'université, en l'occurrence au *Crio Unix*, et n'a été testé que sur des machines Linux (Ubuntu).

Ce script permet de créer un répertoire dans le dossier Public de l'étudiant, et d'y copier le Stub généré. Ainsi il peut être joignable à distance à partir d'un poste différent au CRIO.

```
#!/bin/sh

# Adresse IP de la machine local. Chaque machine du CRIO UNIX dispose
d'une Inet (ifconfig)
$1 = 192.168.0.68

# Numero etudiant, sous forme de 8 chiffres
$2 = 21001232

# Dossier Public de l'etudiant $2 (Doit etre un dossier joignable a
distance : ftp, http, etc)
$3 = /criounix/share.nfs/users/student/$2/Public/

# Clean
rm -f ./Serveur/*.class

# Compilation des differents .java
javac ./Serveur/*.java
```

```
# Generation du Stub & Skel
rmic -vcompat Serveur.TamponInterfaceImpl

# Port. Attention il doit correspondre au port mentionner dans le main de
la classe TamponServeur
rmiregistry 1099 &

# Creation d'un repertoire diedie RMIPgm/ dans le dossier Public de
l'etudiant $2
mkdir $3/RMIPgm/

# Copie du Stub dans le repertoire Public/RMIPgm/ de $2
cp ./Serveur/TamponInterfaceImpl_Stub.class $3/RMIPgm/

# Lancement du Serveur
java -Djava.security.policy=./Serveur/serveur.policy -
Djava.rmi.server.hostname=$1 -Djava.rmi.server.codebase=$3
Serveur.TamponServeur
```

## Client

Cette partie concerne l'exécution d'un client en utilisant un script *MultiClient.sh*. Il permet de lancer l'application graphique.

Ce script est basé sur le même principe que le script du chapitre précédent, le programme se connecte au dossier Public de l'étudiant spécifié pour y récupérer le Stub, ensuite lance son interface graphique *AffichageSAR* qui permet à l'utilisateur de saisir l'adresse IP du serveur distant, le port ainsi que l'objet à joindre (Si aucune valeur saisie, les valeurs par défaut seront utilisées)

```
#!/bin/sh

# Adresse IP de la machine local. Chaque machine du CRIO UNIX dispose
d'une inet (ifconfig)
$1 = 192.168.0.68

# Numero etudiant, sous forme de 8 chiffres
$2 = 21001232

# Dossier Public de l'etudiant $2 (Doit etre un dossier joignable a
distance : ftp, http, etc)
$3 = /criounix/share.nfs/users/student/$2/Public/

# Clean
rm -f ./Client/*.class

# Compilation des differents .java
javac ./Client/*.java

# Lancement du Client
java -Djava.security.policy=./Client/client.policy -
Djava.rmi.server.hostname=$1 -Djava.rmi.server.codebase=$3
Affichage.AffichageSAR
```

Le fonctionnement est très simple et nécessite très peu de manipulation.

TP - *Salim GUENNOUNI*

# Salim GUENNOUNI

TP1 :

- Exercice 1, 2 & 3 (Réalisé pendant la séance de TP en binôme avec Thérèse de La Barthe)
- Exercice 4 réalisé en binôme avec Michael GOLETTO (triFusion)

TP2 : réalisé en binôme avec Michael GOLETTO

TP3 : réalisé en binôme avec Michael GOLETTO

## TP1

DeuxThreads.java :

```
public class DeuxThreads extends Thread {
  public void run () {
        for (int i=0; i<10; i++){
              System.out.println("Nouveau thread " + this.getName());
        }
  }

  public static void main (String [] args){
        DeuxThreads th = new DeuxThreads ();
        th.start();
        //DeuxThreads th2 = new DeuxThreads ();
        //th2.start();
        for (int i=0; i<10; i++){
              System.out.println("Main thread " +
currentThread().getName());
              System.out.flush();
              //yield();
              try {Thread.sleep(1);} catch (InterruptedException e) {
                    e.printStackTrace();}
        }
  }
}

//Q1.3  La commande yield() passe la main a chaque fois. Il ne pourra pas
afficher deux fois
//      de suite
//      La methode sleep endort le thread courant, les autres thread
peuvent s'executer.
```

Compteur.java :

```
public class Compteur extends Thread {

  public Compteur (int n){
        this.setName( "Thread "+n);
  }

  public void run() {
        for (int i=0; i<10; i++){
              System.out.println ( this.getName() + ":" + i);
              try {
                    Thread.sleep((long)(Math.random()*50));
              } catch (InterruptedException e) {
                    e.printStackTrace();
                    }
        }
        System.out.println (this.getName() + " a fini de compter jusqu'à
10.");
  }

  public static void main (String [] args){
        int n = (int) java.lang.Integer.decode(args[0]);
        Compteur[] th = new Compteur[n];
        System.out.println(n);

        for (int i=0; i<n; i++){
              Compteur tmp = new Compteur(i);
              th[i] = tmp;
              tmp.start();
              try {
                    th[i].join();
```

```
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        System.out.println("Le main se termine");
  }
}
Compte.java :
public class Compte {
  private int solde=0;
  public void ajouter (int somme) {
        solde = solde + somme;
        System.out.println(" ajout de "+somme);
  }

  public void retirer (int somme){
        solde = solde - somme;
        System.out.println(" retrait de "+somme);
  }

  public void operationNulle(int somme){
        solde = solde + somme;
        System.out.print(" ajout de " + somme + ",");
        solde = solde - somme;
        System.out.print(" et retrait de "+somme + ".");
        System.out.println();
  }

  public int getSolde(){
        return solde;
  }
}
```

Operation.java :

```
public class Operation extends Thread {
  private Compte compte;
  public Operation(String nom, Compte compte){
        super(nom);
        this.compte=compte;
  }

  public void run(){
        while(true){
                synchronized (this.compte) {
                        int i= (int) (Math.random()*10);
                        String nom=this.getName();
                        System.out.print(nom);
                        this.compte.operationNulle(i);
                        int montant=this.compte.getSolde();
                        if (montant !=0) {
                                System.out.println(nom +" solde =" +montant);
                                System.exit(1);
                        }
                }
        }
  }

  public static void main (String[] args) {
        Compte c = new Compte();
        for (int i=1; i<=2; i++) {
                Operation op= new Operation("op" + i,c);
                op.start();
        }
  }
```

```
}
```

## TP2 (en binôme avec Michael GOLETTO)

Client.java :

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

public class Client {
  private Socket soc;
  private BufferedReader bf;
  private PrintWriter pw;


public Client(String IPAdress, int port) {
        try {
               soc = new Socket(IPAdress, port);
               bf = new BufferedReader(new
InputStreamReader(soc.getInputStream()));
               pw = new PrintWriter(soc.getOutputStream(), true);
        } catch (UnknownHostException e) {
               e.printStackTrace();
        } catch (IOException e) {
               e.printStackTrace();
        }
  }

  public void write(String str) {
        pw.println(str);
        pw.flush();
        System.out.println("Le client a envoyé le message suivant : " +
str);
  }

  public String read() {
        try {
               String str;
               str = bf.readLine();
               System.out.println("Le client a reçu le message suivant :
" + str);
               return str;
        } catch (IOException e) {
               e.printStackTrace();
        }
        return null;
  }

  public void close() {
        try {
               soc.close();
               bf.close();
               bf.close();
        } catch (IOException e) {
               e.printStackTrace();
        }
  }

  /**
   * Exercice 2.1
   * Question 1.1
   */
  /*
  public static void main(String[] args) {
```

```
        Client c = new Client("localhost", 8080);
        c.write("Bonjour");
        c.read();
        c.close();
  }
  */

  /**
   * Exercice 2.1
   * Question 1.2
   */
  public static void main(String[] args) {
        boolean bonjour = true;
        int i = 0;
        Client c = new Client("localhost", 8080);
        while(i++ < 2) {
              if(bonjour) {
                    c.write("Bonjour");
                    bonjour = false;
              } else {
                    c.write("Client dit : " + Math.random());
              }
              c.read();
        }
        c.close();
  }
}
```

Serveur.java :

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

public class Serveur {
  private ServerSocket srv;
  private Socket soc;
  private BufferedReader bf;
  private PrintWriter pw;


public Serveur(int port) {
        try {
              srv = new ServerSocket(port);

        } catch (IOException e) {
              e.printStackTrace();
        }
  }

  public void attendreConnection() {
        try {
              soc = srv.accept();
              bf = new BufferedReader(new
InputStreamReader(soc.getInputStream()));
              pw = new PrintWriter(soc.getOutputStream(), true);

        } catch (IOException e) {
              e.printStackTrace();
        }
  }


  public void write(String str) {
        pw.println(str);
        System.out.println("Le serveur a envoyé le message suivant : " +
str);
```

```
  }

  public String read() {
        try {
              String str;
              str = bf.readLine();
              System.out.println("Le serveur a reçu le message suivant :
" + str);
              return str;
        } catch (IOException e) {
              e.printStackTrace();
        }
        return null;
  }

  public void close() {
        try {
              srv.close();
              soc.close();
              bf.close();
              bf.close();
        } catch (IOException e) {
              e.printStackTrace();
        }
  }


  /**
   * Exercice 2.1
   * Question 1.1
   */
  /*
  public static void main(String[] args) {
        Serveur s = new Serveur(8080);
        s.attendreConnection();
        s.read();
        s.write("Bienvenu");
        s.close();
  }
  */

  /**
   * Exercice 2.1
   * Question 1.2
   * On suppose qu'une lecture à null dans le tampon indique la fin de la
conversation avec le client
   */
  public static void main(String[] args) {
        boolean bienvenu = true;
        Serveur s = new Serveur(8080);
        while(true) {
              s.attendreConnection();
              while(s.read() != null) {
                    if(bienvenu) {
                          s.write("Bienvenu");
                          bienvenu = false;
                    } else {
                          s.write("Serveur dit : " + Math.random());
                    }
              }
        }
  }
}
```

ServeurMultiThread.java :

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

public class ServeurMultiThread implements Runnable {
  private ServerSocket srv;

  public ServeurMultiThread(int port) {
        try {
              srv = new ServerSocket(port);
        } catch (IOException e) {
              e.printStackTrace();
        }
  }

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

  public static void main(String[] args) {
        ServeurMultiThread serveur = new ServeurMultiThread(8080);
        serveur.run();
  }
}
```

ThreadClient.java :

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

public class ThreadClient extends Thread {
  private Socket s;
  private BufferedReader bf;
  private PrintWriter pw;
  private int compteurMessageServeur;

  public ThreadClient(Socket s) {
        super();
        compteurMessageServeur = 0;
        this.s = s;
  }

  public void ouvrirConnexion() {
        try {
              bf = new BufferedReader(new
InputStreamReader(s.getInputStream()));
              pw = new PrintWriter(s.getOutputStream(), true);
        } catch (IOException e) {
              e.printStackTrace();
        }
  }

  public void fermerConnexion() {
        try {
              pw.close();
              bf.close();
              s.close();
        } catch (IOException e) {
              e.printStackTrace();
        }
```

```
  }

  public void write(String s) {
        pw.println(s);
        pw.flush();
        System.out.println("Le serveur " + getName() + " a envoyé le
message suivant : " + s);
  }

  public String read() {
        try {
              String str;
              str = bf.readLine();
              if(!str.isEmpty())
                    System.out.println("Le serveur " + getName() + " a
reçu le message suivant : " + str);
              return str;
        } catch (IOException e) {
              e.printStackTrace();
        }
        return null;
  }

  public void run() {
        ouvrirConnexion();
        write("Bienvenu");
        String reponse = read();
        while(!reponse.isEmpty() && !reponse.equals("Bye")) {
              try {
                    sleep(1000);
              } catch (InterruptedException e) {
                    e.printStackTrace();
              }
              write("Message" + compteurMessageServeur++);
              reponse = read();
        }
        fermerConnexion();
  }
}
```

ClientMultiThread.java :

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

public class ClientMultiThread extends Thread {
  private Socket soc;
  private BufferedReader bf;
  private PrintWriter pw;
  private int compteurMessageClient;

  public ClientMultiThread(String IPAdress, int port) {
        super();
        compteurMessageClient = 0;
        try {
              soc = new Socket(IPAdress, port);
        } catch (UnknownHostException e) {
              e.printStackTrace();
        } catch (IOException e) {
              e.printStackTrace();
        }
  }

  public void open() {
        try {
```

```
                bf = new BufferedReader(new
InputStreamReader(soc.getInputStream()));
                pw = new PrintWriter(soc.getOutputStream(), true);
        } catch (IOException e) {
                e.printStackTrace();
        }
 }

 public void close() {
        try {
                soc.close();
                bf.close();
                bf.close();
        } catch (IOException e) {
                e.printStackTrace();
        }
 }

 public void write(String str) {
        pw.println(str);
        pw.flush();
        System.out.println("Le client " + getName() + " a envoyé le
message suivant : " + str);
 }

 public String read() {
        try {
                String str;
                str = bf.readLine();
                if(!str.isEmpty())
                        System.out.println("Le client " + getName() + " a
reçu le message suivant : " + str);
                return str;
        } catch (IOException e) {
                e.printStackTrace();
        }
        return null;
 }

 public void run() {
        int i = 0;
        String reponse;
        open();
        write("Bonjour");
        while(i++ < 10) {
                reponse = read();
                write("Je suis le client " + soc.getInetAddress() + " et
j'ai fait "
                                + compteurMessageClient++ + "appels.");
                try {
                        sleep(2000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
        write("Bye");
        close();
 }

 public static void main(String[] args) {
        ClientMultiThread c1 = new ClientMultiThread("localhost", 8080);
        ClientMultiThread c2 = new ClientMultiThread("localhost", 8080);
        c1.start(); c2.start();
 }

}
```

ServeurIRC.java :

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

public class ServeurIRC implements Runnable {
  private ServerSocket srv;

  public ServeurIRC(int port) {
        super();
        try {
              srv = new ServerSocket(port);
        } catch (IOException e) {
              e.printStackTrace();
        }
  }

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

  public static void main(String[] args) {
        ServeurIRC serveur = new ServeurIRC(8080);
        serveur.run();
  }

}
```

ThreadClientIRC.java :

```
import java.io.*;
import java.net.*;
import java.util.*;

public class ThreadClientIRC extends Thread {
  private Socket s;
  private BufferedReader bf;
  private PrintWriter pw;
  private static Vector<ThreadClientIRC> clientsConnectes = new
Vector<ThreadClientIRC>();
  private String pseudo;

  public ThreadClientIRC(Socket s) {
        super();
        this.s = s;
  }

  public boolean add(ThreadClientIRC arg0) {
        return clientsConnectes.add(arg0);
  }

  public boolean removeElement(Object arg0) {
        return clientsConnectes.removeElement(arg0);
  }

  public void ouvrirConnexion() {
```

```
        try {
                bf = new BufferedReader(new
InputStreamReader(s.getInputStream()));
                pw = new PrintWriter(s.getOutputStream(), true);
        } catch (IOException e) {
                e.printStackTrace();
        }
 }

 public void fermerConnexion() {
        try {
                pw.close();
                bf.close();
                s.close();
        } catch (IOException e) {
                e.printStackTrace();
        }
 }

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

 public synchronized void writeAll(String s) {
        for(ThreadClientIRC e : clientsConnectes) {
                if (e != this) e.write(s);
        }
 }

 public String read() {
        try {
                String str = "";
                if(bf.ready()) str = bf.readLine();
                if(!str.isEmpty())
                        System.out.println("Le serveur a reçu le message
suivant : " + str);
                return str;
        } catch (IOException e) {
                e.printStackTrace();
        }
        return "";
 }

 private String getClientsConnectes() {
        String res = "";
        if(clientsConnectes.isEmpty()) return "0 client connecte.";
        for(ThreadClientIRC e : clientsConnectes) {
                res += e.getName() + "\n";
        }
        return res;
 }

 public void run() {
        String reponse;
        ouvrirConnexion();
        write("Bienvenu. Veuillez saisir votre pseudo : ");
        reponse = read();
        while(reponse.isEmpty()) {
                reponse = read();
        }
        this.pseudo = reponse;
        setName(pseudo);
        write("Voici la liste des utilisateurs connectes :\n" +
getClientsConnectes());
        synchronized (clientsConnectes) {
```

```
                add(this);
                writeAll(pseudo + " vient de rejoindre la conversation.");
        }
        reponse = read();
        while(!reponse.equals("Quit")) {
                if(!reponse.isEmpty()) {
                        synchronized (clientsConnectes) {
                                writeAll(pseudo + "> " + reponse);
                        }
                }
                reponse = read();
        }
        synchronized (clientsConnectes) {
                removeElement(this);
                writeAll(pseudo + " vient de quitter la conversation.");
        }
        fermerConnexion();
  }
}
```

ClientIRC.java :

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

public class ClientIRC extends Thread {
  private Socket soc;
  private BufferedReader bf;
  private PrintWriter pw;

  public ClientIRC(String IPAdress, int port) {
        super();
        try {
                soc = new Socket(IPAdress, port);
        } catch (UnknownHostException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
  }

  public void open() {
        try {
                bf = new BufferedReader(new
InputStreamReader(soc.getInputStream()));
                pw = new PrintWriter(soc.getOutputStream(), true);
        } catch (IOException e) {
                e.printStackTrace();
        }
  }

  public void close() {
        try {
                soc.close();
                bf.close();
                bf.close();
        } catch (IOException e) {
                e.printStackTrace();
        }
  }

  public void write(String str) {
        pw.println(str);
        pw.flush();
  }
```

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

	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() {
		open();
		String lectureClavier = null;
		String lectureMessages = null;
		while(true) {
			lectureClavier = null;
			lectureMessages = null;
			lectureMessages = read();
			if(lectureMessages != null && !lectureMessages.isEmpty())
{
				System.out.println(lectureMessages);
			}
			lectureClavier = readClavier();
			if(lectureClavier != null && !lectureClavier.isEmpty() &&
!lectureClavier.equals("Quit")) {
				write(lectureClavier);
			}
			if(lectureClavier != null &&
lectureClavier.equals("Quit")) {
				write("Quit");
				close();
				System.exit(0);
			}
		}
	}

	public static void main(String[] args) {
		ClientIRC c = new ClientIRC("localhost", 8080);
		c.start();
	}

}
```

TP - *Michael GOLETTO*

# Michael GOLETTO

TP1 :

- Exercice 1, 2 & 3 réalisé individuellement
- Exercice 4 réalisé en binôme avec Salim GUENNOUNI (triFusion)

TP2 : réalisé en binôme avec Salim GUENNOUNI (voir sa partie)

TP3 : réalisé en binôme avec Salim GUENNOUNI

## TP1

Compte.java :

```
public class Compte {
  private int solde = 0;
  public void ajouter (int somme) {
         solde=solde+somme;
         System.out.println(" ajout de " + somme);
  }
  public void retirer (int somme) {
         solde = solde-somme;
         System.out.println(" retrait de " + somme);
  }
  public void operationNulle(int somme) {
         solde = solde + somme;
         System.out.print(" ajout de " + somme + ",");
         solde = solde - somme;
         System.out.print(" et retrait de " + somme +".");
         System.out.println();
  }
  public int getSolde(){
         return  solde;
  }
}
```

Compteur.java :

```
import java.util.ArrayList;
import java.util.Iterator;

public class Compteur extends Thread {

  public Compteur(int nbr){
         this.setName("Thread " + nbr);
  }

  public void run() {
         for (int i=0; i<10; i++) {
                System.out.println(this.getName() + " : " + (i+1));
                int tps = (int)(Math.random() * 5000);
                try {
                       this.sleep(tps);
                }
                catch (InterruptedException ie) {
                       ie.getMessage();
                }

         }
         System.out.println( this.getName() + " a fini de compter jusqu'à
10");
  }


  public static void main (String[] args) {

         if (args.length > 0){
                int nbr = Integer.parseInt(args[0]);

                ArrayList<Compteur> cmptList = new ArrayList<Compteur>();

                for (int i=0; i < nbr; i++){
                       cmptList.add(new Compteur (i+1));
                       cmptList.get(i).start();
                }
```

```
                Iterator<Compteur> it = cmptList.iterator();
                while(it.hasNext()){
                        try {
                                it.next().join();
                        } catch (InterruptedException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
                System.out.println("Main fini");
        }
  }

}
```

DeuxThread.java

```
public class DeuxThreads extends Thread {

  public void run() {
        for (int i=0; i<10; i++) {
                System.out.println("Nouveau Thread :"+ this.getName());
        }
  }

  public static void main (String[] args) {
        DeuxThreads th = new DeuxThreads ();
        th.setName("Fils");
        th.start();
        for (int i=0; i<10; i++){
                System.out.println("Main Thread : "+
currentThread().getName());
                System.out.flush();

                //currentThread().yield();

                try {
                currentThread().sleep(1);
                }
                catch (InterruptedException ie) {
                        ie.getMessage();
                }

        /*
        Dans le cas du sleep(1), le thread redonnera la main à la JVM
puis attendra 1ms avant de la redemander, tandis que le yield() redonne
la main mais se remet de suite dans la file d'attente de la JVM.
        Plus concrètement, le spleep permettra que plus d'un thread
s'execute s'ils sont plus court que 1ms.
        */
        }
  }
}
```

Operation.java

```
public class Operation extends Thread {
  private Compte compte;
  public Operation(String nom, Compte compte){
        super(nom);
        this.compte = compte;
  }
  public void run(){
```

```
/* QUESTION 2 */
/*
        while(true){
                synchronized(compte){
                        int i = (int) (Math.random() * 10);
                        String nom = this.getName();
                        System.out.print(nom);
                        this.compte.operationNulle(i);
                        int montant = this.compte.getSolde();
                        if (montant != 0){
                                System.out.println(nom + "solde = "
+montant);
                                System.exit(1);
                        }
                }
        }
*/
/* QUESTION 2 ALTERNATIVE */
/* Mettre les méthodes et non l'objet dans des syncronized */

/* QUESTION 3 */
        while(true){
                synchronized(compte){
                        int i = (int) (Math.random() * 10);
                        String nom = this.getName();
                        System.out.print(nom);
                        this.compte.ajouter(i);
                        System.out.print(nom);
                        this.compte.retirer(i);
                        int montant = this.compte.getSolde();
                        if (montant != 0){
                                System.out.println(nom + "solde = "
+montant);
                                System.exit(1);
                        }
                }
        }
 }
 public static void main (String[] args){
        Compte c = new Compte();
        for (int i = 1; i <= 2; i++){
                Operation op = new Operation("op"+i,c);
                op.start();
        }
 }
/*
* Deux thread différent travail sur le même objet.
* Il arrive qu'un ajout ou une soustratcion s'effectue entre un même
Ajout-Retrait.
* Le solde renvoyé ne sera donc pas nul
*/
}
```

TriParallele.java

```
public class TriParallele extends Thread {
 private int[] t;
 private int debut;
 private int fin;

 private TriParallele(int[] t, int debut, int fin){
        this.t = t;
        this.debut = debut;
        this.fin = fin;
 }
```

```
  private void permuter (int a, int b){
        int temp = t[a];
        t[a] = t[b];
        t[b] = temp;
  }

  public void run(){
        if ((fin - debut) < 2){
              if (t[debut] > t[fin]){
                    permuter(debut, fin);
              }
        }
        else {

              int milieu = debut + (fin-debut) / 2;
/* extraire un partie du tableau de debut à milieu */
              TriParallele t1 = new TriParallele(t ,debut, milieu);
              t1.start();
/* extraire un partie du tableau de milieu + 1 à fin */
              TriParallele t2 = new TriParallele(t , milieu+1, fin);
              t2.start();
/* Attendre la fin des autres threads */
              try {
                    t1.join();
                    t2.join();
              } catch (InterruptedException e) {
                    e.printStackTrace();
              }
              trifusion(debut, fin);
        }
  }
  private void trifusion (int debut, int fin){
        int[] tfusion = new int[fin-debut+1];
        int milieu = (debut + fin) / 2;
        int i1 = debut;
        int i2 = milieu + 1;
        int ifusion = 0;
        while(i1 <= milieu && i2 <= fin){
              if(t[i1] < t[i2]){
                    tfusion[ifusion++] = t[i1++];
              }
              else {
                    tfusion[ifusion++] = t[i2++];
              }
        }
        if (i1 > milieu){
              for (int i = i2; i <= fin; i++){
                    tfusion[ifusion++] = t[i];
              }
        }
        else {
              for (int i = i1; i <= milieu; i++){
                    tfusion[ifusion++] = t[i];
              }
        }
        for (int i = 0, j = debut; i <= fin-debut; i++, j++){
              t[j] = tfusion[i];
        }
  }


  public static void main(String[] args){
        int [] tableau = {5, 8, 3, 2, 7, 10, 1, 12, 4};

        TriParallele t = new TriParallele(tableau, 0, tableau.length-1);
        t.start();
```

```
        try {
                t.join();
        } catch (InterruptedException e) {
                e.printStackTrace();
        }

        for (int i = 0; i < tableau.length; i++){
                System.out.println(tableau[i] + ";");
        }
  }
}
```

## TP3 : En binôme avec Salim GENNOUNI

CalculInterface .java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface CalculInterface extends Remote {
  public int somme(int a, int b) throws RemoteException;
}
```

CalculInterfaceImpl .java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class CalculInterfaceImpl extends UnicastRemoteObject implements
CalculInterface {

  protected CalculInterfaceImpl() throws RemoteException {
        super();
  }

  public int somme(int a, int b)  throws RemoteException {
        return a+b;
  }

}
```

CompteurInterface .java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface CompteurInterface extends Remote {
  public int nbOccurrences(String c, String mot) throws RemoteException;
}
```

CompteurInterfaceImpl.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class CompteurInterfaceImpl extends UnicastRemoteObject implements
CompteurInterface {

  protected CompteurInterfaceImpl() throws RemoteException {
        super();
  }

  public int nbOccurrences(String c, String mot) throws RemoteException {
```

```
        int l = mot.length();
        int nb = 0;
        for(int i = 0; i < l; i++) {
               if((mot.substring(i, i+1)).equals(c)) nb++;
        }
        return nb;
  }

}
```

Serveur.java

```
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;

public class Serveur {

  public static void main(String[] args) throws RemoteException {
        CompteurInterfaceImpl co = new CompteurInterfaceImpl();
        CalculInterfaceImpl ca = new CalculInterfaceImpl();
        try {
               Naming.rebind("rmi://localhost:2001/Compteur", co);
               Naming.rebind("rmi://localhost:2001/Calcul", ca);
        } catch (RemoteException e) {
               e.printStackTrace();
        } catch (MalformedURLException e) {
               e.printStackTrace();
        }
  }

}

/* 1.5
javac *.java
rmic -vcompat CompteurInterfaceImpl CalculInterfaceImpl
*/

/* 1.6
rmiregistery &
*/

/* 1.7
javac *.java
rmic -vcompat CompteurInterfaceImpl CalculInterfaceImpl
rmiregistry &
java -Djava.security.policy=serveur.policy Serveur &
java -Djava.security.policy=client.policy Client rmi://localhost/
*/

/* 1.9
javac *.java
rmic -vcompat CompteurInterfaceImpl CalculInterfaceImpl
*/

/* 1.16
Serveur :
javac *.java
rmic -vcompat CompteurInterfaceImpl CalculInterfaceImpl
rmiregistry -J-Djava.security.policy=serveur.policy 2001 &
java -Djava.security.policy=serveur.policy Serveur &
Client :
javac *.java
java -Djava.security.policy=client.policy Client rmi://localhost:2001/
*/
```

Serveur.policy

```
grant{
  permission java.security.AllPermission;
};
```

Client.java

```
import java.net.MalformedURLException;
import java.rmi.*;

public class Client {
  String url;
  CompteurInterface co;
  CalculInterface ca;

  public Client(String url) {
        if(System.getSecurityManager() == null){
              System.setSecurityManager(new RMISecurityManager());
        }
        this.url = url;
        try {
              co = (CompteurInterface) Naming.lookup(url+"Compteur");
              ca = (CalculInterface) Naming.lookup(url+"Calcul");
        } catch (MalformedURLException e) {
              e.printStackTrace();
        } catch (RemoteException e) {
              e.printStackTrace();
        } catch (NotBoundException e) {
              e.printStackTrace();
        }
  }

  public int sommeCL(int a, int b) throws RemoteException{
        return ca.somme(a,b);
  }

  public int nbOccurrencesCL(String c, String mot) throws RemoteException{
        return co.nbOccurrences(c,mot);
  }

  public static void main(String[] args) throws Exception{
        Client cl = new Client(args[0]);
        System.out.println("Nous cherchons les r dans zorro : " +
cl.nbOccurrencesCL("r", "zorro"));
        System.out.println("Nous additionnons 17 et 24 : " +
cl.sommeCL(17, 24));
  }

}
```

# TP – Kacem BOUHMADI & Raphael LUMBROSO

# TP 1

## EXERCICE 1.0

```
public class DeuxThreads extends Thread {
    public void run() {
        for(int i=0; i < 10; i++)
            System.out.println("nouveau thread");
    }

    public static void main(String[] args) {
        DeuxThreads th = new DeuxThreads();
        th.start();
        for(int i=0; i < 10; i++)
        {
            System.out.println("Main thread");
            System.out.flush();
        }
    }
}
```

## EXERCICE 1.1

```
public class DeuxThreads extends Thread {
    public void run() {
        this.setName("thread fils");

        for(int i=0; i < 10; i++)
            System.out.println("Thread fils " + this.getName());
    }

    public static void main(String[] args) {
        DeuxThreads th = new DeuxThreads();
        th.start();

        currentThread().setName("Main Thread");

        for(int i=0; i < 10; i++)
        {
            // Dans le tread principal on peut pas faire this.getNAme()
car c'est un thread spécial ....
            System.out.println("Thread pere :
"+currentThread().getName());
            System.out.flush();
        }
    }
}
```

## EXERCICE 1.2

```
public class DeuxThreads extends Thread {
    public void run() {
        this.setName("thread fils");

        for(int i=0; i < 10; i++)
        {
            System.out.println("Thread fils " + this.getName());
        }
    }

    public static void main(String[] args) {
        DeuxThreads th = new DeuxThreads();
```

```
        th.start();

        currentThread().setName("Main Thread");

        for(int i=0; i < 10; i++)
        {
            System.out.println("Thread pere :
"+currentThread().getName());
            System.out.flush();
            Thread.yield();
        }
    }
}
```

## EXERCICE 1.3

```
public class DeuxThreads extends Thread {
    public void run() {
        this.setName("thread fils");

        for(int i=0; i < 10; i++)
        {
            System.out.println("Thread fils " + this.getName());
        }
    }

    public static void main(String[] args) {
        DeuxThreads th = new DeuxThreads();
        th.start();

        currentThread().setName("Main Thread");

        for(int i=0; i < 10; i++)
        {
            System.out.println("Thread pere :
"+currentThread().getName());
            System.out.flush();
            try {
                Thread.sleep(1);
            } catch(Exception e)
            {
                System.out.println(e);
            }
            // Avec sleep on peut avoir plusieurs appels à la suite de
père si le processeur le décide, avec yield() le processus père est mis
en état pause donc on peut pas avoir deux père à la suite SAUF à la fin
(si le père fils se termine avant)
        }
    }
}
```

## EXERCICE 2.1

```
public class Compteur extends Thread
{
    public int NumThreadCourant;

    public void run()
    {
        this.setName("Thread " + NumThreadCourant);
        System.out.println(this.getName());

    }

    public Compteur(int i)
```

```
    {
        NumThreadCourant = i;
    }

    public static void main(String[] args)
    {
        args = new String[] { "10" };
        if(args.length == 0)
        {
            System.out.print("Erreur");
            return;
        }
        int nbThread = Integer.parseInt(args[0]);


        for(int i=0; i < nbThread; i++)
        {
            Compteur c = new Compteur(i);
            c.start();
        }
    }
}
```

## EXERCICE 3.1

```
public class Operation extends Thread
{
    private Compte compte;

    public Operation(String nom, Compte compte)
    {
        super(nom);
        this.compte = compte;
    }

    public void run()
    {
        while(true)
        {
            int i = (int) (Math.random() * 10);

            String nom = this.getName();
            System.out.println(nom);

                this.compte.operationNulle(i);

                int montant = this.compte.getSolde();
                if(montant != 0)
                {
                    System.out.println("~~~~~~~~~~~~~~~~> "+nom + "
solde= "+montant);
                    System.exit(1);
                }
        }
    }


    public static void main(String[] args)
    {
        Compte c = new Compte();
        for(int i=1; i <=2; i++)
        {
            Operation op = new Operation("op"+i, c);
```

```
            op.start();
        }
    }

}



public class Compte
{
    private int solde=0;

    public void ajouter(int somme)
    {
        solde+=somme;
        System.out.println(" ajout de "+somme);
    }

    public void retirer(int somme)
    {
        solde -= somme;
        System.out.println(" retrait de "+somme);
    }

    public void operationNulle(int somme)
    {
        solde += somme;
        System.out.print(" ajout de "+somme+" , ");

        solde -= somme;
        System.out.println(" retrait de "+somme);
    }

    public int getSolde() { return solde; }

}
```

## EXERCICE 3.2

```
public class Operation extends Thread
{
    private Compte compte;

    public Operation(String nom, Compte compte)
    {
        super(nom);
        this.compte = compte;
    }

    public void run()
    {
        while(true)
        {
            int i = (int) (Math.random() * 10);

            String nom = this.getName();
            System.out.println(nom);

            synchronized(compte)
            {
                this.compte.operationNulle(i);

                int montant = this.compte.getSolde();
```

```
                if(montant != 0)
                {
                    System.out.println("~~~~~~~~~~~~~~~~> "+nom + "
solde= "+montant);
                    System.exit(1);
                }
            }
        }
    }


    public static void main(String[] args)
    {
        Compte c = new Compte();
        for(int i=1; i <=2; i++)
        {
            Operation op = new Operation("op"+i, c);
            op.start();
        }
    }

}
```

## EXERCICE 3.3

```
public class Operation extends Thread
{
    private Compte compte;

    public Operation(String nom, Compte compte)
    {
        super(nom);
        this.compte = compte;
    }

    public void run()
    {
        while(true)
        {
            int i = (int) (Math.random() * 10);

            String nom = this.getName();
            System.out.println(nom);

            //On peut aussi synchronized operationNulle et getSolde mais
c'est moins pratique
            synchronized(compte)
            {
                //this.compte.operationNulle(i);

                this.compte.ajouter(i);
                this.compte.retirer(i);

                int montant = this.compte.getSolde();
                if(montant != 0)
                {
                    System.out.println("~~~~~~~~~~~~~~~~> "+nom + "
solde= "+montant);
                    System.exit(1);
                }
```

```
            }
        }
    }


    public static void main(String[] args)
    {
        Compte c = new Compte();
        for(int i=1; i <=2; i++)
        {
            Operation op = new Operation("op"+i, c);
            op.start();
        }
    }

}
```

## EXERCICE 4.1

```
public class TriParallele extends Thread{

    private int[] t;
    private int debut, fin;

    public int getDebut() {
        return debut;
    }

    public int getFin() {
        return fin;
    }


    public TriParallele(int[] t) {
        this.t = t;
    }

    public TriParallele(int[] t, int debut, int fin) {
        this.t = t;
        this.debut = debut;
        this.fin = fin;
    }

    private void permuter(int a, int b) {
        int temp = t[a];
        t[a] = t[b];
        t[b] = temp;
    }

    private void trier(int debut, int fin) {
        if (fin - debut < 2) {
            if (t[debut] > t[fin]) {
                permuter(debut, fin);
            }
        }
        else {
            int milieu = debut + (fin - debut) / 2;
            trier(debut, milieu);
            trier(milieu + 1, fin);
```

```
            triFusion(debut, fin);
        }
    }

    private void triFusion(int debut, int fin) {
        int[] tFusion = new int[fin - debut + 1];
        int milieu = (debut + fin) / 2;
        int i1 = debut,
        i2 = milieu + 1;
        int iFusion = 0;
        while (i1 <= milieu && i2 <= fin) {
            if (t[i1] < t[i2]) {
                tFusion[iFusion++] = t[i1++];
            }
            else {
                tFusion[iFusion++] = t[i2++];
            }
        }
        if (i1 > milieu) {
            for (int i = i2; i <= fin; i++) {
                tFusion[iFusion++] = t[i];
            }
        }
        else {
            for (int i = i1; i <= milieu; i++) {
                tFusion[iFusion++] = t[i];
            }
        }
        for (int i = 0, j = debut; i <= fin - debut; i++, j++ ) {
            t[j] = tFusion[i];
        }
    }

    public void run(){
        int debut = getDebut();
        int fin = getFin();
        if (fin - debut < 2) {
            if (t[debut] > t[fin]) {
                permuter(debut, fin);
            }
        } else {
            int milieu = (fin + debut) / 2;
            TriParallele t1 = new TriParallele(t,debut, milieu);
            TriParallele t2 = new TriParallele(t,milieu+1,fin);
            t1.start();
            t2.start();
            try{
                t1.join();
                t2.join();
                triFusion(debut, fin);
            }catch (java.lang.InterruptedException ie){
                ie.printStackTrace();
            }
        }
    }


    public static void main(String[] args) {
        int[] tableau1 = {5, 8, 3, 2, 7, 10, 1, 12, 4};
        int[] tableau2 = {5, 8, 3, 2, 7, 10, 1, 12, 4};
        TriParallele t1 = new TriParallele(tableau1);
        t1.trier(0, tableau1.length - 1);
        for (int i = 0; i <tableau1.length; i++) {
            System.out.print(tableau1[i] + " ; ");
        }
        System.out.println();
```

```
        TriParallele t2 = new TriParallele(tableau2, 0, tableau2.length -
1);
        t2.start();
        try{
            t2.join();
        }catch (Exception ie){
            ie.printStackTrace();
        }
        for (int i = 0; i <tableau2.length; i++) {
            System.out.print(tableau2[i] + " ; ");
        }
    }
}
```

## TP 2

### Client.java

```
import java.net.*;
import java.io.*;
import java.util.*;


public class Client extends Thread{
    Socket sock;

    public Client(InetAddress ad, int port){
        try{
            sock = new Socket(ad, port);
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    }

    public Client(){
        try{
            sock = new Socket(InetAddress.getLocalHost(),8080);
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    }
    public Client(int port){
        try{
            sock = new Socket(InetAddress.getLocalHost(),port);
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    }

    public void run(){
        try{
            int numero = (int)(Math.random()*200);
            PrintWriter pw = new PrintWriter(sock.getOutputStream(),
true);
            BufferedReader br = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
            int i=0;
            while(i<10){
                String reply = br.readLine();
                System.out.println("Client"+numero+" : j'ai recu
"+reply);
                i++;
```

```
                pw.println("Je suis le client"
+InetAddress.getLocalHost()+" et j'ai fait "+i+" appels");
                sleep(2000);
            }
            pw.println("Bye");
            sock.close();
            pw.close();
            br.close();
        }catch(IOException ioe){
            ioe.printStackTrace();
        }catch(InterruptedException ie){
            ie.printStackTrace();
        }
    }


}
```

## ClientIRC.java

```
import java.net.*;
import java.io.*;
import java.util.*;


public class ClientIRC extends Thread{
    Socket sock;
    String pseudo;
    PrintWriter pw;
    BufferedReader br;

    public ClientIRC(String ad, int port, String pseudo){
        try{
            sock = new Socket(ad, port);
            pw = new PrintWriter(sock.getOutputStream(), true);
            br = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
            this.pseudo = pseudo;

        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    }

    public ClientIRC(int port, String pseudo){
        try{
            sock = new Socket(InetAddress.getLocalHost(),port);
            pw = new PrintWriter(sock.getOutputStream(), true);
            br = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
            this.pseudo = pseudo;
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    }


    public void run(){
        try{
            pw.println(pseudo);
            Saisie s = new Saisie(pw);
```

```
            s.start();
            String reply = "";
            while("exit".compareTo(reply) != 0){
                reply = br.readLine();
                System.out.println(reply);
            }
            pw.close();
            br.close();
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    }

    public static void main(String[] args) throws InterruptedException{
        ClientIRC c = new ClientIRC(args[0], 2445, args[1]);
        c.start();
        c.join();
        System.out.println("FIN");
    }


}
```

## ClientMultiThread.java

```
import java.net.*;
import java.io.*;
import java.util.*;


public class ClientMultiThread extends Thread{

    public ClientMultiThread(){
    }


    public void run(){
        for(int i=0;i<20;i++){
            Client c = new Client(2345);
            c.start();
        }
    }

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

}
```

## ClientThread.java

```
import java.net.*;
import java.io.*;
import java.util.*;
```

```
public class ClientThread extends Thread{
    Socket sock;

    public ClientThread(Socket sock){
        this.sock = sock;
    }



    public void run(){
        try{
            PrintWriter pw = new PrintWriter(sock.getOutputStream(),
true);
            BufferedReader br = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
            int i=0;
            pw.println("Bonjour du serveur");
            String reply;
            while(true){
                reply = br.readLine();
                System.out.println("Serveur : j'ai recu "+reply);
                if("".compareTo(reply) == 0 || "Bye".compareTo(reply) ==
0){
                    break;
                }else{
                    sleep(1000);
                    i++;
                    pw.println("Phrase"+i+" du serveur");
                }
            }
            sock.close();
            pw.close();
            br.close();
        }catch(IOException ioe){
            ioe.printStackTrace();
        }catch(InterruptedException ie){
            ie.printStackTrace();
        }
    }


}
```

## Saisie.java

```
import java.net.*;
import java.io.*;
import java.util.*;


public class Saisie extends Thread{
    PrintWriter pw;
    BufferedReader br;

    public Saisie(PrintWriter pw){
        this.pw = pw;
    }


    public void lire(InputStreamReader in) {
        try {
            br = new BufferedReader(in);
            String message = "";
```

```
            while ("exit".compareTo(message) != 0) {
                message = br.readLine();
                pw.println(message);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    private void saisieClavier() {
        lire(new InputStreamReader(System.in));
    }

    public void run(){
        while(true){
            saisieClavier();
        }
    }


}
```

## Serveur.java

```
import java.net.*;
import java.io.*;
import java.util.*;

public class Serveur extends Thread{

    ServerSocket servSock;

    public Serveur(int port){
        try{
            servSock = new ServerSocket(port);
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    }

    public void run(){
    try{
        Socket clientServiceSocket = servSock.accept();

        PrintWriter pw = new
PrintWriter(clientServiceSocket.getOutputStream(), true);
        BufferedReader br = new BufferedReader(new
InputStreamReader(clientServiceSocket.getInputStream()));

        String request = "";
        String reply = "Bienvenue";

        int i = 0;
        while(i<10){
            request = br.readLine();
            System.out.println("Serveur : jai recu "+request);
            pw.println(reply+i);
            i++;
        }
            pw.println("fin");
            pw.close();
            br.close();
```

```
            clientServiceSocket.close();
            servSock.close();

    }catch(IOException ioe){
        ioe.printStackTrace();
    }

    }

    public static void main(String[]args){
        Serveur s = new Serveur(2345);
        s.start();
    }
}
```

## ServeurIRC.java

```
import java.net.*;
import java.io.*;
import java.util.*;

public class ServeurIRC extends Thread{

    ServerSocket servSock;
    Vector<ThreadClientIRC> v;
    int nbClients = 0;

    public ServeurIRC(int port){
        try{
            servSock = new ServerSocket(port);
            v = new Vector<ThreadClientIRC>();
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    }

    synchronized public void envoyerATous(String message){
        PrintWriter pw;
        ThreadClientIRC tci;
        for(int i=0; i<v.size();i++){
            tci = (ThreadClientIRC) v.elementAt(i);
            pw = tci.getPrintWriter();
            pw.println(message);
        }
    }

    synchronized public void enleverClient(int i){
        nbClients--;
        if(v.elementAt(i) != null){
            v.removeElement(v.elementAt(i));
        }
    }

    synchronized public int ajouterClient(ThreadClientIRC tci){
        nbClients++;
        v.addElement(tci);
        return v.size()-1;
    }

    synchronized public int getNbClients(){
        return nbClients;
    }
```

```
    synchronized public String getListeClients(){
        String s="";
        ThreadClientIRC tci;
        for(int i=0;i<v.size();i++){
            tci = (ThreadClientIRC) v.elementAt(i);
            s+= tci.getPseudo();
            s+= "; ";
        }
        return s;
    }

    public void run(){
        while(true){
            try{
                ThreadClientIRC tci = new
ThreadClientIRC(servSock.accept(), this);
                tci.start();
            }catch(IOException ioe){
                ioe.printStackTrace();
            }
        }
    }

    public static void main(String[] args){
        ServeurIRC si = new ServeurIRC(2445);
        si.start();
    }

}
```

## ServeurMultiThread.java

```
import java.net.*;
import java.io.*;
import java.util.*;

public class ServeurMultiThread extends Thread{

    ServerSocket servSock;

    public ServeurMultiThread(int port){
        try{
            servSock = new ServerSocket(port);
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    }

    public void run(){
        while(true){
            try{
                ClientThread ct = new ClientThread(servSock.accept());
                ct.start();
            }catch(IOException ioe){
                ioe.printStackTrace();
            }
        }
    }

    public static void main(String[] args){
        ServeurMultiThread smt = new ServeurMultiThread(2345);
        smt.start();
    }
```

```
}
```

## ThreadClientIRC.java

```
import java.net.*;
import java.io.*;
import java.util.*;


public class ThreadClientIRC extends Thread{
    Socket sock;
    String pseudo;
    PrintWriter pw;
    BufferedReader br;
    ServeurIRC si;
    int numeroClient = 0;

    public ThreadClientIRC(Socket sock, ServeurIRC si){
        try{
            this.sock = sock;
            this.si = si;
            pw = new PrintWriter(sock.getOutputStream(), true);
            br = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
            numeroClient = si.ajouterClient(this);
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    }

    public String getPseudo() {
        return pseudo;
    }

    public PrintWriter getPrintWriter(){
        return pw;
    }



    public void run(){
        try{
            pseudo = br.readLine();
            pw.println("Bienvenue sur le serveur IRC "+pseudo);
            pw.println("Les personnes présentes sur le chat sont : "+
si.getListeClients());
            si.envoyerATous(pseudo+" vient de se connecter");
            String reply = "";
            while("exit".compareTo(reply) != 0){
                reply = br.readLine();
                if("exit".compareTo(reply) != 0){
                    si.envoyerATous(pseudo+" > "+reply);
                }
            }
            pw.println("exit");
            si.envoyerATous(pseudo+" a quitté le chat");
            si.enleverClient(numeroClient);
            si.envoyerATous("Les personnes présentes sur le chat sont :
"+ si.getListeClients());
            sock.close();
            pw.close();
            br.close();
```

```
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    }


}
```

# TP 3

## Package Client

### Client.java

```
import java.rmi.*;

public class Client{

    String url;
    CompteurInterface co;
    CalculInterface ca;

    public Client(String url) throws Exception{
        if(System.getSecurityManager() == null){
            System.setSecurityManager(new RMISecurityManager());
        }
        this.url = url;
        co = (CompteurInterface) Naming.lookup(url+"Compteur");
        ca = (CalculInterface) Naming.lookup(url+"Calcul");
    }

    public int ajouter(int a, int b) throws RemoteException{
        return ca.Somme(a,b);
    }

    public int combien(String c, String mot) throws RemoteException{
        return co.NbOccurences(c,mot);
    }

    public static void main(String[]args) throws Exception{
        Client t = new Client(args[0]);
        System.out.println("Le resultat de 3 + 4 est : "+t.ajouter(3,4));
        System.out.println("Dans jaaoeiproeniehaubqpihaomb il y a
"+t.combien("a","jaaoeiproeniehaubqpihaomb")+ "a");

    }

}
```

### Client.policy

```
grant{
  permission java.security.AllPermission;
};
```

## Package Serveur

### CalculInterface.java

```
import java.rmi.*;
```

```
public interface CalculInterface extends Remote {
    public int Somme (int a, int b) throws RemoteException;
}
```

### CalculInterfaceImpl.java

```
import java.rmi.*;

public interface CalculInterface extends Remote {
    public int Somme (int a, int b) throws RemoteException;
}
```

### CompteurInterface.java

```
import java.rmi.*;

public interface CompteurInterface extends Remote {
    public int NbOccurences (String c, String mot) throws
RemoteException;
}
```

### CompteurInterfaceImpl.java

```
import java.rmi.server.*;
import java.rmi.*;

public class CompteurInterfaceImpl extends UnicastRemoteObject implements
CompteurInterface {

    public CompteurInterfaceImpl() throws RemoteException{
    }

    public int NbOccurences (String c, String mot){
        int l = mot.length();
        int nb = 0;
        for(int i=0; i<l; i++){
            if((mot.substring(i, i+1)).equals(c)) nb++;
        }
        return nb;
    }
}
```

### Serveur.java

```
import java.rmi.*;

public class Serveur{
    public static void main(String[] args) throws Exception{
        CompteurInterfaceImpl co = new CompteurInterfaceImpl();
        CalculInterfaceImpl ca = new CalculInterfaceImpl();
        Naming.rebind("rmi://localhost:2001/Compteur", co);
        Naming.rebind("rmi://localhost:2001/Calcul", ca);
    }
}
```

Serveur.policy

```
grant{
  permission java.security.AllPermission;
};
```