Base de programmation Objet en JAVA. 5ème partie.

Frédéric Vernier
(Université Paris-Sud / LRI / LIMSI-CNRS)
Frederic.Vernier,@limsi.fr

Ce cours reprend en grande partie le matériel pédagogique mis au point par Jérôme Nobécourt, Christian Jacquemin et Claude Barras pour l'enseignement de Java en 2001 et 2002 en FIIFO et celui trouvé sur Internet
(http://www.laltruiste.com/, etc.)

Plan 9: Thread = fil d'exécution

1. What's that
2. Ordi. fou ou rebelle
3. Processus / Thread
4. Le Thread Java
5. Exemple
6. Timer:Simple et Utile
7. Exemple 1/2
8. Exemple 2/2
9. Résultat
10. Gène et collaboration entre Thread
14. Exemple
15. Lancement
16. Résultat
17. Synchronisation
18. Exemple
19. Synchronisation solidaire
20. Un Thread dans tous ses états
21. Libération du verrou
22. Producteur-Consomateur

What's that ?

- Threads = processus léger
- Un ordinateur est une machine à exécuter pas à pas un programme.
- Un programme est une suite d'instructions.
- A un moment donné l'ordinateur exécute une et une seule instruction à un endroit précis du programme
- Le seul destin de l'ordinateur après cet instant c'est de passer à l'instruction suivante ou de sauter ailleurs dans le programme si l'instruction du programme dit « saute là-bas»
- Si ça plante c'est vraiment la fin !

Ordinateur rebelle ou fou

- Ce qui serait bien
  - L'exécution de plusieurs programmes en même temps
  - Le plantage d'un programme n'entraîne pas celui des autres
  - L'inactivité d'un programme n'entraîne pas celle de l'ordi qui passe son temps à faire tourner les programmes qui ont du boulot sur la planche !
- Ce qui existe
  - Les processus système. Un processus = Un exécutable (ou presque). Très indépendants
- C'est comme si un ordinateur avait un dédoublement de personnalité.
  - En fonction du besoin des programmes l'ordinateur se met dans la peau des programmes un à un.

Processus / Thread

- Pour simplifier
  - 2 processus = 2 programmes. Pas facile pour le programmeur de partager des variables (mémoire) d'appeler des fonctions de l'autre prg, etc…
  - 2 Threads = 1 programme
  - 1 Thread = 1 position dans le programme
  - 1 Thread = 1 position de départ
- En Java
  - 2 Threads = 1 machine virtuelle Java
  - 1 Thread = 1 objet

Le Thread Java

- Il existe toujours un premier Thread qui commence à exécuter la fonction main()
  - (Alors que dans la première classe il n'y a pas forcement d'instance)
- Dans les interfaces graphiques
  - Il existe un autre Thread qui attend les actions de l'utilisateur et déclenche les écouteurs
  - Il existe un autre Thread qui redessine les composants graphiques qui ont besoin de l'être
- On peut créer ses propres Threads

Exemple

- Tutorial Java ->Essential ->Threads
- http://java.sun.com/docs/books/tutorial/essential/threads/index.html
  - 3 applets de tris en parallèle
  - Comparaison des vitesses
  - Garantie de l'équité du CPU par rapport aux algos

UNIVERSITÉ PARIS-SUD 11

Timer : Simple et Utile

- Pragmatiquement
  - Un Thread pour repousser une action dans le temps
    - Dans 3 secondes, si l'utilisateur n'a pas bougé sa souris, afficher un popup disans « Ce bouton sert à quitter le document »
  - Un Thread pour répéter une action régulièrement
    - Tous les 0.5 secondes, redessine la barre de progression.
- Pour ces cas simple, pas besoin de faire des Threads compliqués : Utiliser un Timer !

Exemple

- Tutorial Java ->Essential ->Threads ->Timer
- http://java.sun.com/docs/books/tutorial/essential/threads/timer.html
  - Déclencher une action tout en continuant
  - Déclencher une action après n millisecondes
  - Déclencher des actions toutes les p millisecondes

Thread

- Plus général
  - Un Thread peut attendre : Thread.sleep()
  - Un Thread peut boucler : while
  - Mais il peut aussi
    - Attendre en fonction du temps écoulé
    - Attendre en fonction de l'utilisateur
    - Répéter de plus en plus lentement (connexion réseau)
  - Décider des priorités entre Threads
    - Affichage multimédia = plus prioritaire (temps réel)
    - Connexion réseau = moins prioritaire mail que http

Exemple

```
class PrimeThread extends Thread {
  PrimeThread() {
  }

  public void run() {
    while (null==null)
      System.out.println("Moi je suis dans run(),
                          je suis un autre thread !!!");
  }
};
//-------------------------------------------------------------
public class TestThread {
  public static void main(String[] ags) {

    Thread Thread1 = new PrimeThread ();
    Thread1.start();

    while (null==null)
      System.out.println("Moi je suis dans main()");
  }
}
```

Attention aux deux noms de fonction

- run() = description interne de la vie du Thread
- start() = lancement externe du thread !

Résultat

Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!

Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans run(), je suis un autre thread !!!
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
Moi je suis dans main()
…

Gène et collaboration entre Thread

- 1 Thread peut passer la main avec yield()
  - On aurra des lignes biens alternées
- 2 Threads peuvent ne jamais parcourir les mêmes ligne du code Java (chacun ses fonctions) comme on vient de le voir
- Parcourir la même fonction de la même instance (ou une autre fct. ou fct. statique) et donc partager les mêmes champs.
  - Collaboration (multi-proc)
  - Risque de gêne

Exemple

```
Public Class PrimeThread extends Thread
  int Valeur = 0; // Valeur est une variable d'instance

  public void run() {
    while (null==null) {
      System.out.println("thread 18 fait son boulot
      doJob(18);
    }
  }
  public void doJob(int newValeur_Arg) {
    int OldValeur = Valeur;
    Valeur        = newValeur_Arg;
    System.out.println("  Valeur = "+Valeur);
    if (Valeur != newValeur_Arg) {
      System.out.println(" ZUT Val="+Valeur+" et Arg="+newValeur_Arg);
      System.exit(0);
    }
    Valeur = OldValeur;
  }
}
```

Attention aux deux appels de fonctions

- doJob() est appelé par run() et…

Lancement

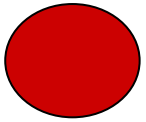
```
public class TestThread2 {

public static void main(String[] ags) {

    PrimeThread Thread1 = new PrimeThread ();
    Thread1.start();

    while (null==null) {
      System.out.println("L'annee utilisee est : ");
      Thread1.doJob(1995);
}
```

Attention aux deux appels de fonctions

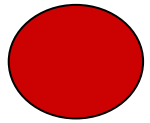
- doJob() est aussi appelé par main()

Résultat

thread 18 fait son boulot :
  Valeur = 18
thread 18 fait son boulot :
  Valeur = 18
thread 18 fait son boulot :
  Valeur = 18
  Valeur = 1995
L'annee utilisee est :
  Valeur = 1995
L'annee utilisee est :
  Valeur = 1995
L'annee utilisee est :
  Valeur = 1995
L'annee utilisee est :
  Valeur = 1995
L'annee utilisee est :
  Valeur = 1995



L'annee utilisee est :
  Valeur = 1995
L'annee utilisee est :
  Valeur = 1995
L'annee utilisee est :
  Valeur = 1995
L'annee utilisee est :
  Valeur = 1995
L'annee utilisee est :
  Valeur = 1995
 ZUT Val=1995 et Arg=18
L'annee utilisee est :
  Valeur = 1995
L'annee utilisee est :
  Valeur = 1995
L'annee utilisee est :
  Valeur = 1995



Synchronisation

- Si on met yield() ça alterne mieux (plus de gros blocs) mais ça plante quand même (ça met 3ou 4 secondes au lieu de 1)
- Le yield() interrompt mais rien ne garantit qu'il n'y aura pas d'autres interruptions
- Ici l'erreur arrive vite car il y a peu de lignes de code, les chances d'une interruption au mauvais endroit sont élevées, sur un code de 5000 lignes avec juste 2ou 3 lignes à ne pas interrompre …
  - P.S. : 1ligne != un cycle
- On rajoute **synchronized** à doJob()

Exemple de synchronisation

```
public synchronized  void doJob(int newValeur_Arg) {
    int OldValeur = Valeur;
    Valeur        = newValeur_Arg;
    System.out.println("  Valeur = "+Valeur);
    if (Valeur != newValeur_Arg) {
      System.out.println("  ZUT ALORS Valeur = "+Valeur);
      System.exit(0);
    }
    Valeur = OldValeur;
  }
};
```

- Exclusion :
  - Les fonctions synchronized ne laissent pas entrer un Thread tant qu'un autre Thread est déjà dedans (i.e en train d'exécuter cette fct.)
  - L'autre Thread dort à l'entrée

Synchronisation solidaire

- Solidarité entre fonctions synchronisées de la même instance :
  - 2 fonctions différentes (disons qu'on duplique doJob() en doJob1() et doJob2()) Si un Thread est déjà à l'intérieur de n'importe quelle fonction synchronized de l'objet ça bloque aussi l'entrée des autres !
  - 2 fonction s de 2 objets (instances) différente ne sont pas solidaire
  - On peut créer une instance « bidon » juste pour synchroniser ce que l'on veut (appeée verrou)

Un Thread dans tous ses états

- isAlive() signifie qu'un Thead a été démarré et n'a pas encore finit
  - Impossible de distingué les 2 autres cas ! GLURP
- Actif ou pas
  - Signifie que si le CPU est libre je vais travailler.
  - Vrai sauf sleep(1000) en ms ou wait()
  - IMPOSSIBLE DE SAVOIR (RE-GLURP)
    - Intérieur = normal car si un Thread n'est pas actif il ne peut appeler aucune fonction
    - Extérieur = Compréhensible car le temps dire qu'un Thread dort il pourrai déjà être réveillé ()

Libération du verrou = wait()

- Dans une fonction synchronized
  - Verrou bloqué à l'entrée
  - Verrou débloqué à la sortie
  - Si au milieu un Thread se rend compte qu'il ne peux pas faire son boulot car un autre Thread bloqué doit faire son boulot avant ?
    INTERBLOQUAGE = cauchemar de codeur
- Exemple du producteur-consommateur
  - Consommateur ne consomme que le produit
  - Producteur ne stocke pas (attend consommation)
  - Tout doit être consommé

Producteur - Consommateur

- http://java.sun.com/docs/books/tutorial/essential/threads/synchronization.html
- Déverrouille = wait();
- Notifier = notifyAll()

UNIVERSITÉ PARIS-SUD 11

Priorité

- Un Thread hérite de la priorité de son créateur
- Fonction de priorité
  - Thread1.setPriority() = modification de la priorité
  - Constante Thread.MIN_PRIORITY (resp. MAX)
- Politique
  - Si 2 Threads se battent pour obtenir la main …
  - Si un Thread de plus haute priorité veut la main il l'obtient
  - Si 2 Threads de même priorité se battent
    - Time slicing

Groupe de Thread

- Les groupes servent à lancer des Threads ensemble, les figer ensembles, etc.

```
public Thread(ThreadGroup grp, Runnable rnbl, String nam)
```

- Sous groupes et hiérarchie
- Suspension et Arrêt en commun

Conseils

- Faire des traces compréhensibles (Nom ou No de Thread)
- Encapsuler + synchroniser set() et get()
- yield() dans les grosses boucles
- Mesurer les performances du système (Hz)
- Coupler un Thread avec un état
  - Une variable int + constantes de valeurs possibles
  - NOT_STARTED, STARTING, STARTED, FINISHED, etc…

Plan 10: Réseau, RMI, Applet, …

1. Réseau
2. Désignation
3. URLConnection
4. Socket
5. ServerSocket
6. Server : Initialisation
7. Server : déroulement
8. Raffinements
9. Server2 : Initialisation
10. Server2 : déroulement
11. RMI
12. JDBC
13. Applet
14. Servlet
15. Potion Magique
16. Conclusion
17. Perspectives
18. Fin

Réseau

- Faire le point sur ce dont on a besoin
  - Faire communiquer 2 applications
  - Transférer des ressources à travers le réseau
  - Centraliser les infos sur un serveur
- Réseau = package compliqué
  - Beaucoup de notions
  - URL, InetAddress, Socket, client-serveur (vues en cours)
  - Spécifique Java
    - Encapsulation d'objets pour le réseau
    - Applet / Servlet / RMI

UNIVERSITÉ PARIS-SUD 11

Désignation

- Objet d'adresse d'ordinateur sur le réseau
  - inetAddress : adresse réseau (www.merl.com, 129.88.32.24) : gestion de la résolution de nom et de l'espace multicast (le cas échéant)
- Objets d'adresse de ressources sur le réseau
  - URL : http://java.sun.com:80/api/index.html#chapter1
    - Gestion du protocole et du port
    - Gestion des ressources (chemin et paramètres)
  - URI : comme URL mais non restreint au Web
- Gestion de IP v6 (nom entre crochets)
  - Voir documentation dans le futur

URLConnection

- Cas simple
  - InputStream IS1= URL1.openStream();
- Cas plus compliqué
  - URL1.openConnection()
  - Gestion du cache
  - Gestion de l'interaction (authentification)
  - Gestion des en-têtes (dates, taille, type, etc.)
  - Rapatriement de la ressource

UNIVERSITÉ PARIS-SUD 11

Socket

- Objet représentant un lien vers un ordinateur
  - Client
- Concept réseau par excellence
  - Sur un ordinateur
  - Pointe vers un autre ordinateur du réseau
  - Sur un port (nombre entre 1 et …beaucoup)
  - Communication en entrée (réception d'infos)
  - Communication en sortie (envoi d'infos)
  - new Socket("www.lemonde.fr", 80);
  - getInputStream() et getOutputStream() comme point de départ de la communication

ServerSocket

- Socket en attente de connexion
  - Constructeur (initialisation)
  - accept()
    - mise en attente (bloquant)
    - retour de la fonction : une socket normale !
  - close() : arrêt propre
- Exemple du client-serveur
  - Ultra important !
  - Utilise des Thread(s)

UNIVERSITÉ PARIS-SUD 11

Server : Initialisation

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

public class EchoServer {
  public static void main(String[] args) throws
  IOException {
    ServerSocket s = new ServerSocket(8008);
    try {
      Socket socket      = s.accept();
      BufferedReader in = new BufferedReader(
        new InputStreamReader(
          socket.getInputStream()));
      PrintWriter out =new PrintWriter(
        new BufferedWriter(
          new OutputStreamWriter(
            socket.getOutputStream())),true);
```

UNIVERSITÉ PARIS-SUD 11

Server : Déroulement

```
        while (true) {
            String str = in.readLine();
            if (str.equals("END"))
              break;
            System.out.println("Echoing: " + str);
            out.println(str);
        }
    } finally {
      System.out.println("closing...");
      socket.close();
      s.close();
    }
}
```

Client : Initialisation

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

public class EchoClient {
  public static void main(String[] args) {
    InetAddress addr = InetAddress.getByName(null);
    Socket socket = new Socket(addr, 8008);
    try {
      BufferedReader in = new BufferedReader(
        new
  InputStreamReader(socket.getInputStream()));
      PrintWriter out = new PrintWriter(
        new BufferedWriter(new OutputStreamWriter(
          socket.getOutputStream())),true);
```

UNIVERSITÉ PARIS-SUD 11

Client : Déroulement

```
for(int i = 0; i < 10; i ++) {
        out.println(" ligne " + i);
        String str = in.readLine();
        System.out.println(" recu : "+str);
      }
      out.println("END");
    } finally {
    System.out.println("closing...");
    socket.close();
  }
}
```

Raffinements

- Pour se parler à soi-même

```
InetAddress.getByName(null);
InetAddress addr = InetAddress.getByName("127.0.0.1");
InetAddress addr = InetAddress.getByName("localhost");
```

- Pour parler aux autres
  - liste de contacts dans un fichier
  - Adresses de broadcast
  - Tolérance aux pannes de réseau
- Serveur multi-clients
  - Pour l'instant quand le serveur envoie ou reçoit il n'écoute pas !
  - Thread pour écouter + Thread pour envoyer/recevoir

UNIVERSITÉ PARIS-SUD 11

Server2 : Initialisation

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

public class EchoServer extends Thread{
  private Socket socket = null;
  public static void main(String[] args) throws IOException {
    ServerSocket s = new ServerSocket(8008);
    while (true)
        try {
        Socket socket     = s.accept();
        new EchoServer(socket);
      }
    }
  }
  public EchoServer (Socket socket) {
    this.socket = socket;
    start();
  }
```

Server2 : Déroulement

```
Public void run() {
 BufferedReader in = new BufferedReader(
        new InputStreamReader(
          socket.getInputStream()));
      PrintWriter out = new PrintWriter(
        new BufferedWriter(
          new OutputStreamWriter(
            socket.getOutputStream())),true);

   while (true) {
        String str = in.readLine();
        if (str.equals("END"))
          break;
        System.out.println("Echoing: " + str);
        out.println(str);
   }
  } finally {...
```

UNIVERSITÉ PARIS-SUD 11

RMI

- Remote Method Invocation
  - Appel d'une méthode à distance
    - Identification de la classe, de l'instance et de la méthode
  - Client-Serveur
    - Appelant = client, Appelé = serveur
  - Passage d'objets
    - Paramètres, objet de retour
    - Sérialisation
  - Appels synchrones
- Exemple : HelloWorld

JDBC

- Accès aux BD depuis votre code JAVA
  - Chargement du pilote
  - Connexion de type « réseau » avec la BD
  - Création dynamique de la requête (manip. String)
  - Récupération des lignes de tables
- Exemple : PDH

Applet

- Code java dans une page Web (!=javascript)
- Taille et position de l'applet comme pour une image dans HTML
- Initialisation init() au lieu de main(), 1ère instance par défaut
- Sécurité : l'applet a peu de droit (HD, réseau)
  - SecurityManager du browser
- Exemple : Vitesse (après)

UNIVERSITÉ PARIS-SUD 11

Servlet

- Autre dev. kit : J2EE (Entreprise Edition)
- Création de page web dynamique (cgi, scripts, etc.) en java
- Serveur de page + VM Java (Evite le chargement de la VM à chaque requête)
- Manipulation du protocole HTTP
- Manipulation du format de document HTML

Potion magique

- Serveur
  - Servlet Java (Création de la Page web)
  - JDBC
  - Threads (1 par client)
- Client
  - Applet JAVA (décrite sur la page web)
  - SWING (Interface)
- Communication (http) entre client et serveur
  - Echange d'information formatée (XML) et d'objets

Conclusion

- Cours de programmation objet
  - Répétition pour certains
  - Jamais les notions assez claires
- Cours de Java
  - Pourquoi pas C++ ?
  - Manque de pratique
- Cours des technologies associées à Java
  - Echantillon, goût, saveur mais peu de concret
  - Non exhaustif (J2EE, JavaMail, JAI, JMF, etc.)
  - Awareness !

Perspectives

- Court terme
  - Pratique : projet de prog + TER 2nd semestre
  - Autres packages selon votre curiosité
- Moyen Terme
  - Génie logiciel (Debuggage, conception UML,)
  - Design Pattern (Patron de conception)
  - Liens entre matières informatiques (BD, réseau…)
- Long Terme
  - Vie professionnelle liée à l'informatique
  - Vie personnelle de technophile

Fin

- Evaluation du cours
- Examen à préparer
- Poly de cours à récupérer
- Curiosité et Imagination