Fabien Coelho
École des mines de Paris
fabien@coelho.net

Composé avec LaTeX

Utilisation des *threads* de JAVA

- exécution en *parallèle* d'un même programme
  plusieurs *fils* d'exécution
- données et code *partagé* : économie de mémoire
- moins lourd qu'un *processus* système
- implémenté par le système, ou bien émulé par `java`
- support en JAVA (langage et librairie) pour :
  - création d'une *thread*
  - démarrage et arrêt
  - synchronisation (conflits d'accès sur des données)
  - gestion d'un groupe de *threads*

Interface `Runnable`

- contient la méthode `void run();`
- méthode exécutée par une *thread*
- sorte de fonction `main()` pour une *thread* !

Classe `Thread` implémente `Runnable`

- construction d'une *thread* prête à démarrer
- avec un objet implémentant `Runnable`
- démarrage avec `void start();`
- exécute alors le `run()` de l'objet en **parallèle**


```
class FaitQuelqueChose implements Runnable
{
  // attribut propre à la thread
  protected String msg;

  // constructeur
  public FaitQuelqueChose(String msg) { this.msg = msg; }

  // méthode d'exécution de la thread
  public void run()
  {
    System.out.println("run: (" + msg + ") début !");
    try { Thread.sleep(5000); } catch (Exception e) {}
    System.out.println("run: (" + msg + ") milieu...");
    try { Thread.sleep(5000); } catch (Exception e) {}
    System.out.println("run: (" + msg + ") fin !");
} }
```



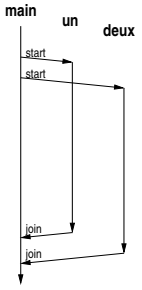
```
public class ThreadEx {
  static public void main(String[] args) throws Exception {
      System.out.println("main: démarre un");
      Thread t1 = new Thread(new FaitQuelqueChose("un"));
      t1.start();

      try { Thread.sleep(2500); } catch (Exception e) {}

      System.out.println("main: démarre deux");
      Thread t2 = new Thread(new FaitQuelqueChose("deux"));
      t2.start();

      System.out.println("main: attente fin d'exécution...");
      t1.join(); System.out.println("main: t1 est fini");
      t2.join(); System.out.println("main: t2 est fini");
} }
```


Commentaires sur `ThreadEx`

- classe `FaitQuelqueChose` implémente `Runnable`
  méthode `run()` à fournir
  affiche quelques messages avec son `msg`
- attributs pour le paramétrage de l'exécution
- exécution du `main()`
  - création et démarrage de 2 *threads* supplémentaires
  - passage d'un objet `Runnable` qui sera *exécuté*
  - attente de leur terminaison

Exécution de `ThreadEx`

```
shell> java ThreadEx
main: démarre un
run: (un) début !
main: démarre deux
main: attente fin d'exécution...
run: (deux) début !
run: (un) milieu...
run: (deux) milieu...
run: (un) fin !
main: t1 est fini
run: (deux) fin !
main: t2 est fini
```




```
public class ThreadEg extends Thread
{
  protected String msg;           // attributs
  protected int compte;

  public ThreadEg(String m, int c) // constructeur
  {
    msg = m;
    compte = c;
  }

  public void run()          // méthode d'exécution
  {
    for (int i=0; i<compte; i++) {
      System.out.println(msg + " : " + i);
      try { Thread.sleep(4000); } catch (Exception e) { }
    }
```

```
  }

  // fonction de test
  static public void main(String[] args)
  {
    Thread t1 = new ThreadEg("casimir", 5);
    Thread t2 = new ThreadEg("leonard", 3);
    t1.start();
    t2.start();
    System.out.println("ok");
  }
}
```


– extension directe de `Thread`
  qui implémente `Runnable`

```
shell> java ThreadEg
casimir : 0
ok
leonard : 0
casimir : 1
leonard : 1
casimir : 2
leonard : 2
casimir : 3
casimir : 4
```


Classe `Thread`

– nombreux constructeurs avec `ThreadGroup Runnable String`
  `Thread t = new Thread(group, runnable, "nom du thread");`

  ThreadGroup groupe de threads

  Runnable object à exécuter

  String nom de la thread

– caractéristiques : nom, priorité, groupe...

– principales méthodes et fonctions disponibles
  démarrage initial, rendez-vous final, attente...

```
public class Thread implements Runnable
{
  // constructeurs...
  // méthodes principales
  public void start();
  public void run();
  public void join();
  public void interrupt();
  public boolean isInterrupted();
  // fonctions...
  static public void sleep(long ms);
  static public boolean interrupted();
```


– implémente `Runnable`

```
class Fait extends Thread {
  public void run() { ... }
}
  ...
  // utilisation ailleurs !
  Fait f = new Fait(); f.start();
```

– *thread* courante de l'exécution
  `Thread moi = Thread.currentThread();`


```
public class ThreadInter extends Thread {
  public void run() {
    try {
      System.out.println("running...");
      for (long l = 0; l<30000000; l++);
      Thread.sleep(1000L);
      System.out.println("done...");
    }
    catch (InterruptedException e) {
      System.err.println(e);
  }  }
  static public void main(String[] args) throws Exception {
    Thread t = new ThreadInter();
    t.start();
    t.interrupt();
    System.err.println("interrupted!");
} }
```


Exécution de `ThreadInter`

– interruption par exception `InterruptedException`
– dans les méthodes `wait join sleep`
– éventuellement interruption des I/O...

```
shell> java ThreadInter
interrupted!
running...
java.lang.InterruptedException: sleep interrupted
```


Besoin de synchronisation

– données **partagées** et threads :
  incohérences en cas de modifications concurrentes
  bogues non déterministes
– exemple : attribut modifié simultanément par deux threads

`o.x = o.x + 1;`

thread 1 lit x, vaut 3
thread 1 calcule, résultat 4
thread 1 stocke 4 dans x

thread 2 lit x, vaut 3
thread 2 calcule, résultat 4
thread 2 stocke 4 dans x

Synchronisation en java

- verrouillage d'une *instance* par une *thread*
- autres *threads* en attentes

  si synchronisent la **même** instance !
- syntaxe du `synchronized`

```
synchronized (x) {   // x est une instance...
  x.cpt = x.cpt + 1; // accès à un attribut
  x.inc();           // modification par une méthode
}
```

- méthode `synchronized` : bloque l'instance `this`

```
synchronized void addElement(...) { ...
```

- fonction `synchronized` : accès unique à la classe


```
{
  protected int total; // attribut

  public Compteur()  // constructeur
  { total=0; }

  public int getTotal()
  { return total; }
  public void incremente()
  {
    int tmp = total;
    // simule un calcule un peu lent...
    try { Thread.sleep(1); } catch (Exception e) {}
    total = tmp + 1;
  }
}
```



```
class Comptage extends Thread
{
  protected Compteur compte;
  public Comptage(Compteur c) { compte = c; }
  public void run()
  {
    for (int i=0; i<10; i++)
    {
      // accès directe au compte, non exclusif
      compte.incremente();
    }
  }
}
```



```
class ComptageMieux extends Thread
{
  protected Compteur compte;
  public ComptageMieux(Compteur c) { compte = c; }
  public void run()
  {
    for (int i=0; i<10; i++)
    {
      synchronized (compte) // vérouillage du compteur
      {
        // accès exclusif ici
        compte.incremente();
      }
    }
  }
}
```



```
public class Synchro
{
  static public void main(String[] args) throws Exception
  {
    Compteur total1 = new Compteur(); // compteur partagé
    Thread t1 = new Comptage(total1);
    Thread t2 = new Comptage(total1);
    t1.start(); t2.start(); t1.join(); t2.join();
    System.out.println("Comptage : " + total1.getTotal());

    Compteur total2 = new Compteur(); // compteur partagé
    Thread t3 = new ComptageMieux(total2);
    Thread t4 = new ComptageMieux(total2);
    t3.start(); t4.start(); t3.join(); t4.join();
    System.out.println("ComptageMieux : " + total2.getTotal());
} }
```


Exécution du programme `Synchro`

```
shell> java Synchro
Comptage : 11
ComptageMieux : 20
```

Remarques

- on aurait pu synchroniser la méthode `incremente`
- bien mieux, car évite l'erreur dans `Comptage`

Risque d'étreinte fatale *(dead lock)*

- vérouillage de plusieurs instances par des *threads*
- les threads 1 et 2 ont besoin de `o1` et `o2`
- vérouillage en un ordre inverse

  thread 1 vérouille `o1` — *thread 2* vérouille `o2`

  thread 1 attend *thread 2* pour vérouiller `o2`

  thread 2 attend *thread 1* pour vérouiller `o1`

  blocage indéfini des deux threads !
- solution : faire très attention !

  vérouillage dans le même sens des instances ?

  ne vérouiller qu'une instance à la fois ?

  détection à l'exécution ? (base de données. . .)

Blocage et déblocage sur une instance

```
public class Object {
  public void wait();      // blocage...
  public void notify();    // déblocage !
  public void notifyAll(); // déblocages !
```

Exemple `Course`

```
shell> java Course
ready?
steady.
go!
zou!
zou!
zou!
```

```
{
  protected Object bang;
  public Course(Object b) { bang = b; }
  public void run()
  {
    try
    {
      synchronized (bang)
      {
        bang.wait();
      }
      System.out.println("zou!");
    }
    catch (InterruptedException e) {
      System.err.println(e);
  } }
```



```
  {
    Object pistolet = new Object();
    new Course(pistolet).start();
    new Course(pistolet).start();
    new Course(pistolet).start();
    System.out.println("ready?");
    Thread.sleep(1000);
    System.out.println("steady.");
    Thread.sleep(1000);
    System.out.println("go!");
    synchronized (pistolet) {
      pistolet.notifyAll();
    }
  }
}
```


Classe `ThreadGroup`

- groupe de *threads* (surprise !) hiérarchisées
- notamment utilisée par le système
- éventuellement utile pour certaine applications
- manipulation direct d'un groupe : `interrupt destroy`...

```
public class ThreadGroup {
  // constructeurs...
  public ThreadGroup(ThreadGroup parent, String name);
  // méthodes...
  public ThreadGroup getParent();
  public void enumerate(ThreadGroup[]);
  public void interrupt();
```


Synchronisation des collections

- par défaut **non synchronisées**
  optimisation car petit surcoût à l'exécution
- encapsulation de la synchronisation au cas par cas
  fonctions `synchronized...` de `Collections`

```
List potage = Collections.synchronizedList(new ArrayList());
Set salade = Collections.synchronizedSet(new HashSet());
Map fruits = Collections.synchronizedMap(new TreeMap());
```


Conseils pour la constitution de thread

- méthode `run()` pour une instance
  équivalent fonction `main(...)` pour une classe
- communication après lancement difficile...
  exécution indépendante, *vie sa vie*
- systématiquement :
  1. données nécessaires en **attributs** !
  2. constructeur initialise les attributs
  3. méthode `run()` pour exécuter la tâche

Package `java.net`

- manipulation réseau au niveau TCP/IP ou UDP/IP
- adresse machine : `InetAddress`
- UDP : envoi de messages bruts (*unreliable*)
  `DatagramPacket DatagramSocket MulticastSocket`
- TCP : gestions de connexions (suite cohérente de paquets)
  classes client et serveur : `Socket ServerSocket`
  communications : flux d'octets (`InputStream OutputStream`)
- applications : `URL URLConnection HttpURLConnection`...

Classe `InetAddress`

- représente l'adresse internet d'une machine
- obtention par des **fonctions** (pas constructeur), `UnknownHostException`
- sous classes `Inet4Address` et `Inet6Address`

```
public class InetAddress
{
  // fonctions...
  static public InetAddress getLocalHost() throws ...
  static public InetAddress getByName(String name) throws ...
  static public InetAddress[] getAllByName(String name) throws ...
  // méthodes...
  public byte[] getAddress();
  public String getHostAddress();
  public String getHostName();
```



```
import java.net.*;
public class Internet
{
  static public void main(String[] args) throws Exception
  {
    InetAddress a = InetAddress.getLocalHost();
    System.out.println("cet hôte : " + a);

    InetAddress[] ta = InetAddress.getAllByName(args[0]);

    System.out.println("machine : " + args[0]);
    for (int i=0; i<ta.length; i++)
      System.out.println(i + " : " + ta[i]);
  }
}
```

Fonctionnement de `InetAddress`

- interroge le service de nommage
- fichier `/etc/hosts`, NIS, DNS

```
shell> java Internet blonville
cet hôte : leuven.ensmp.fr/10.2.14.128
machine : blonville
0 : blonville/193.48.171.236
```


UDP : User Datagram Protocol

- protocole de bas niveau sur IP
- envoi de données entre deux couples numéro IP/port
  `192.54.172.245 :5342` → `192.54.172.242 :53`
- exemples de protocoles utilisant UDP
  DNS (Domain Name Service, port 53)
  TFTP (Trivial File Transfert Protocol, port 69)
  NTP (Network Time Protocol, port 123)
  NFS (Network File System, port 2049)
- *unreliable* : comme le courrier de la poste !
  pas de garantie d'arrivée, pas d'ordre d'arrivée,
  pas de vérification, pas d'accusé de réception, pas de connexion, etc.

Classe `DatagramPacket`

- paquet de données brutes
- destiné/provenant d'une machine (`InetAddress`)/port (`int`)
- informations *mises à jour* lors des réceptions

```
public class DatagramPacket {
  public void setAddress(InetAddress i);
  public InetAddress getAddress();
  public void setPort(int port);
  public int getPort();
  public void setData(byte[] buffer);
  public byte[] getData();
  public void setLength(int len);
  public int getLength();
```


Classe `DatagramSocket`

- envoi et réception d'un `DatagramPacket`
- écoute/émet sur un port et une adresse (passé au constructeur)
- options : délais maximum d'attente...

```
public class DatagramSocket {
  public void receive(DatagramPacket p);
  public void send(DatagramPacket p);
  public void setSoTimeout(int d);
  public int getSoTimeout();
```



```
import java.net.*;

public class ReceveurUDP
{
  // USAGE: java ReceveurUDP ip port
  static public void main(String[] args) throws Exception
  {
    // création du serveur UDP
    InetAddress addr = InetAddress.getByName(args[0]);
    int port = Integer.parseInt(args[1]);
    DatagramSocket sock = new DatagramSocket(port, addr);

    // paquet pour recevoir les données
    byte[] data = new byte[1000];
    DatagramPacket paq = new DatagramPacket(data, data.length);
```



```
    // boucle de réception
    while (true)
    {
      // réception
      sock.receive(paq);

      // affiche le contenu du paquet sur sortie standard
      System.out.println(paq.getAddress() + ":" + paq.getPort());
      for (int i=0, len=paq.getLength(); i<len; i++)
        System.out.write(data[i]);
      System.out.println();

      // remise à zéro
      paq.setLength(data.length);
    }
  }
}
```



```
import java.net.*;
import fr.ensmp.util.EzInput;

public class EnvoyeurUDP
{
  // USAGE: java EnvoyeurUDP src-host src-port dst-host dst-port
  static public void main(String[] args) throws Exception
  {
    // création du client UDP
    InetAddress s_addr = InetAddress.getByName(args[0]);
    int s_port = Integer.parseInt(args[1]);
    DatagramSocket sock = new DatagramSocket(s_port, s_addr);

    // destination
    InetAddress d_addr = InetAddress.getByName(args[2]);
    int d_port = Integer.parseInt(args[3]);
```



```
    // boucle d'envoi à partir de l'entrée standard
    String line;
    while ((line=EzInput.readLine()) != null)
    {
      byte[] data = line.getBytes();
      DatagramPacket p =
        new DatagramPacket(data, data.length, d_addr, d_port);

      // envoie !
      sock.send(p);
    }
  }
}
```

```
shell> java ReceveurUDP localhost 2000
localhost/127.0.0.1:1500
bonjour serveur
localhost/127.0.0.1:1500
au revoir
localhost/127.0.0.1:2222
hello
```

Côté envoyeur (voir aussi `nc -u`)

```
shell> java EnvoyeurUDP localhost 1500 localhost 2000
bonjour serveur
au revoir

shell> java EnvoyeurUDP localhost 2222 localhost 2000
hello
```


Classe `MulticastSocket`

- extension de `DatagramSocket`
- données destinées à plusieurs machines (*broadcast*)
- groupes de machines définies par un *masque*
- spécification du *time to live* (TTL) : nombre de *hops* (sauts)

TCP : *Transmission Control Protocol*

- assure une liaison, une connexion *persistante*
- échange de données bidirectionnel
  similaire à une communication téléphonique
  appel et conversation
- assure l'acheminement, la vérification...
  renvoie les paquets si nécessaire

Classe `Socket`

- connexion entre deux machines (IA/Port ⟶ IA/Port)
- établissement de la connexion à la création
- deux flux d'octets, quelques options paramétrables

```
public class Socket {
  public OutputStream getOutputStream(); // vers destinataire
  public InputStream getInputStream(); // du destinataire
  public void close(); // fini !
```

```
Socket so = new Socket("deauville", 80);
PrintWriter pw = new PrintWriter(so.getOutputStream());
pw.println("GET / HTTP/1.0");
pw.println();
pw.flush();
```



```
import java.net.*; import java.io.*;
public class ClientTCP {
  // USAGE: java ClientTCP hôte-dest
  static public void main(String[] args) throws Exception
  {
    InetAddress addr = InetAddress.getByName(args[0]);
    Socket so = new Socket(addr, 80);

    PrintWriter pw = new PrintWriter(so.getOutputStream());
    pw.print("GET / HTTP/1.1\r\nHost: " + args[0] + "\r\n\r\n");
    pw.flush(); // envoie le paquet

    BufferedReader br = new BufferedReader
      (new InputStreamReader(so.getInputStream()));
    System.out.println(br.readLine()); // première ligne
    so.close();
} }
```


Exécution de `ClientTCP`

```
shell> java ClientTCP www.ensmp.fr
HTTP/1.1 200 OK
```


Classe `ServerSocket`

- création avec le numéro de port à écouter
- attend une ouverture de connexion sur un IA/Port
- appel bloquant de `accept()`, retourne une `Socket` ouverte
- utilisation typique avec des *threads*
  création d'une thread pour gérer la connexion

```
ServerSocket sso = new ServerSocket(1045);
while (true) {
  Socket so = sso.accept(); // attend...
  // traitement
}
```


Protocole pour `ServerTCP`

- *ouverture de la connexion*
- ⟵ *salutation du serveur*
- ⟶ *message du client*
- ⟵ *remerciement du serveur*
- *fermeture de la connexion*

```
import java.io.*;

public class ServerTCP
{
  // USAGE: java ServerTCP port
  static public void main(String[] args) throws Exception
  {

    // création du serveur
    int port = Integer.parseInt(args[0]);
    ServerSocket serv = new ServerSocket(port);

    System.out.println("serveur : " + serv);
```



```
    Socket client = serv.accept();
    System.out.println(client);

    PrintWriter pw = new PrintWriter(client.getOutputStream());
    BufferedReader br = new BufferedReader
      (new InputStreamReader(client.getInputStream()));

    pw.println("Bonjour, donne moi un message d'une ligne !");
    pw.flush();

    String msg = br.readLine();
    System.out.println("message : " + msg);

    pw.println("Merci pour ce message."); pw.flush();
    client.close();
} } }
```


Côté ServerTCP

```
shell-leuven> java ServerTCP 7685
serveur : ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=7685]
Socket[addr=blonville/193.48.171.236,port=1110,localport=7685]
message : A Fontainebleau, Devant l'hotel de l'aigle noir
Socket[addr=chailly99/193.48.171.215,port=2387,localport=7685]
message : Il y a un taureau sculpte par Rosa Bonheur
```

Côté client

```
shell-blonville> telnet leuven 7685
Bonjour, donne moi un message d'une ligne !
A Fontainebleau, Devant l'hotel de l'aigle noir
Merci pour ce message.
shell-chailly99> telnet leuven 7685
Bonjour, donne moi un message d'une ligne !
Il y a un taureau sculpte par Rosa Bonheur
Merci pour ce message.
```


URL : *Uniform Resource Locator*

- localisation d'une resource (contenu) sur l'internet
- protocole, machine, port, répertoires, fichier...

  `ftp ://ftp.irisa.fr/pub/gnu`

  `http ://www.cri.ensmp.fr/mastere/index.html`
- classe URL : localisation, obtention d'une ressource

  doit connaître le protocole !
- classes *abstraites* `URLConnection HttpURLConnection`


```
import java.net.*;
public class Get
{
  // usage : java Get url-http...
  static public void main(String[] args) throws Exception
  {
    URL url = new URL(args[0]);
    HttpURLConnection uc =
      (HttpURLConnection) url.openConnection();
    System.out.println("status " + uc.getResponseMessage());
    System.out.println(uc.getDate() + "ms depuis 1970");
    System.out.println(uc.getContentLength() + " octets");
    System.out.println(uc.getContentType());
    // System.out.println("document : " + uc.getContent());
  }
}
```


Récupération d'une URL avec Get

- document : sous forme de flux ?

```
shell> java Get http://palo-alto2.ensmp.fr/
status OK
1007149097000ms depuis 1970
1358 octets
text/html
```


Serveur réseau multi-thread

- gestion en parallèle des clients
- création d'une *thread* pour chacun

  une instance doit implémenter le protocole

  attributs nécessaires ? gestion des erreurs ?
- évite blocage des autres clients

```
ServerSocket sso = new ServerSocket(...);
while (true) {
  Socket client = sso.accept();
  Gestion g = new Gestion(client);
  Thread t = new Thread(g);
  t.start();
}
```



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

public class BonjourMerci extends Thread
{
  // les attributs
  Socket so;             // connexion au client
  String bonjour, merci; // paramétrage du protocole

  // un constructeur
  public BonjourMerci(Socket s, String b, String m)
  {
    so = s;
    bonjour = b;
    merci = m;
  }
```

```
    try {
      PrintWriter pw = new PrintWriter(so.getOutputStream());
      BufferedReader br = new BufferedReader
        (new InputStreamReader(so.getInputStream()));

      pw.println(bonjour); // message d'accueil
      pw.flush();          // envoie le paquet

      System.out.println(so + " " + br.readLine());

      pw.println(merci); pw.flush(); // thanks + envoie le paquet

      so.close();
    }
    catch (Exception e) { System.err.println(so + " " + e); }
} }
```



```
public class SertBjMc
{
  // usage : java SertBjMc port
  static public void main(String[] args) throws Exception
  {
    int port = Integer.parseInt(args[0]);
    ServerSocket serv = new ServerSocket(port);
    System.out.println(serv);
    while (true)
    {
      Socket client = serv.accept();
      Thread t = new BonjourMerci(client, "Hello", "Thanks");
      t.start(); // démarre la thread
    }
  }
}
```


Côté serveur `SertBjMc`

```
shell> java SertBjMc 5345
ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=5345]
Socket[...] Un peu plus loin tout autour, Il y a la foret
Socket[...] Un peu plus loin encore, Joli corps
...
```

Côté clients, éventuellement simultanés

```
shell> telnet leuven 5345
Hello
Un peu plus loin tout autour, Il y a la foret
Thanks
...
```


Serveur TCP/IP générique ?

- classe `Serveur` utilisé pour plusieurs protocoles
  cœur du serveur (boucle accept) dans une thread
- les spécificité ne doivent pas apparaître !
  classe de gestion du service générique ?
  construction de la gestion générique ?
  configuration du protocole générique ?
- solution générique basée sur quatre classes :

  ConfService contient la configuration du service

  Usine produit des gestionnaires du service configurés

  Service le service rendu en *thread*

  Serveur le serveur générique en *thread*


```
import java.net.*;

class ConfService {
  // attributs éventuels
  // constructeurs...
}

interface Usine {
  public Runnable gereService(Socket so, ConfService c);
}

public class Serveur extends Thread
{
  // attributs d'un serveur
  protected int port;          // port TCP à écouter
  protected Usine atelier;     // production pour gerer le service
  protected ConfService conf;  // contient la configuration
```



```
  // constructeur d'un serveur
  public Serveur(int p, Usine a, ConfService c)
  { port = p; atelier = a; conf = c; }

  // code très minimal du serveur...
  public void run() {
    try {
      ServerSocket serveur = new ServerSocket(port);
      while (true) {
        Socket client = serveur.accept();
        Runnable service = atelier.gereService(client, conf);
        Thread t = new Thread(service);
        t.start();
      }
    }
    catch (Exception e) { System.err.println(e); }
} }
```



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

class ConfBonjour extends ConfService
{
  protected String bonjour, merci;
  public ConfBonjour(String b, String m)
  { bonjour = b; merci = m; }
}

class UsineBonjour implements Usine
{
  public Runnable gereService(Socket s, ConfService c)
  {
    return new Bonjour(s, (ConfBonjour) c);
  }
}
```



```
class Bonjour implements Runnable
{
  // attributs
  protected Socket so;
  protected ConfBonjour conf;

  // constructeur
  public Bonjour(Socket s, ConfBonjour c) { so = s; conf = c; }

  // rend le service
  public void run()
  {
    try {
      // ... conf.bonjour conf.merci so.getInputStream() ....
    }
    catch(Exception e) { System.err.println(e); }
} }
```

```java
// exemple de programme principal
public class ServeurBonjour
{
  static public void main(String[] args) throws Exception
  {
    ConfBonjour conf = new ConfBonjour("NiHao", "TsieTsie");
    UsineBonjour usine = new UsineBonjour();
    Serveur serv1 = new Serveur(1234, usine, conf);
    Serveur serv2 = new Serveur(4321, usine, conf);
    serv1.start();
    serv2.run();
  }
}
```


Titre