

# Programmation concurrente et Java

---

## Les Threads en Java : une première approche

jean-michel Douin, [douin@cnam.fr](mailto:douin@cnam.fr)

Version du 16 Octobre 2003

[http://jfod.cnam.fr/tp\\_cdi/{douin/}](http://jfod.cnam.fr/tp_cdi/{douin/})

# Bibliographie utilisée

---

## Java

- **Java Threads. S.Oaks & H.Wong. O'Reilly 1997**
- **Concurrent Programming in Java.D. Lea.. Java Series. Addison Wesley.1997**
- The Java Tutorial. M.Campione, K. Walrath. Java Series. Addison Wesley.1996.
- The Java Programming language. K. Arnold, J.Gosling. Addison Wesley.1996.
- Java As a Concurrent Language. A.Skowronski.  
<http://www.openface.ca/~skow/623/623.html>
- Java & Trade: concurrency, synchronisation and Inheritance, david Holmes. Microsoft Research Institute. Macquarie University, Sydney.Voir sur internet
- <http://www.EckelObjects.com> chapitre 14: multiple threads

## Programmation concurrente

- Monitors, an operating system structuring concept. C.A.R. Hoare. CACM, vol 17, n°10, Oct.1974. pages 549-557
- Concurrent Programming: Principles and Practice.G.Andrews, Benjamin Cummings-91
- Techniques de synchronisation pour les applications parallèles.G Padiou, A.Sayah, Cepadues editions 1990

# Sommaire

---

- Approche "bas-niveau" : L'API Thread
  - les méthodes *init*, *start*, *stop*, *sleep* et *yield*
  - Moniteur de Hoare
  - wait, notify et variable de condition
  - Schéma producteur/consommateur
  - Passage de message et Rendez-Vous
  - Ordonnancement
  
- Programmation concurrente en Java par Doug Lea
  - Approche méthodologique
  - conception utilisant les Patterns

# Introduction

---

- *Les Threads Pourquoi ?*
- Entrées sorties non bloquantes
- Alarmes, Réveil, Déclenchement périodique
- Tâches indépendantes
- Algorithmes parallèles
- Modélisation d 'activités parallèles
- Méthodologies
- ...
  
- **note** :Un *Thread* est associé à la méthode *main* pour une application Java en autonome. Ce *Thread* peut en engendrer d 'autres...

# L' API Thread Introduction

---

- La classe Thread est prédéfinie (package java.lang)
- Création d'une nouvelle instance
  - Thread unThread = **new** Thread()
    - (*un Thread pour processus allégé...*)
- Exécution du processus
  - unThread.start();
    - Cet appel engendre l'exécution de la méthode unThread.run(), (rend eligible *unThread* dont le corps est défini par la méthode *run()*)
    - new Thread().start()

# L' API Thread Introduction, exemple

---

- class ThreadExtends extends Thread {
- **public void run(){**
- while(true){
- System.out.println("dans ThreadExtends.run");
- }
- }
- }
- }
  
- public class Thread1 {
  
- public static void main(String args[]) {
- ThreadExtends t0 = new ThreadExtends();
- **t0.start();**
- while(true){
- System.out.println("dans Thread1.main");
- }
- }
- }
- }

# exemple, trace d 'exécution

---

- *dans Thread1.main*
- *dans Thread1.main*
- *dans Thread1.main*
- *dans ThreadExtends.run*
- *dans ThreadExtends.run*
- *dans Thread1.main*
- *dans Thread1.main*
- *dans ThreadExtends.run*
- *dans ThreadExtends.run*
- *dans ThreadExtends.run*
- *dans Thread1.main*
- *dans Thread1.main*
- *dans Thread1.main*
- *dans Thread1.main*
- *dans ThreadExtends.run*
- ==> Ordonnanceur de type tourniquet sous Windows 95

# L' API Thread

---

- *import java.lang.Thread* // par défaut pour *java.lang*

## Les constructeurs publics

- *Thread();*
- *Thread(Runnable target);*
- ...

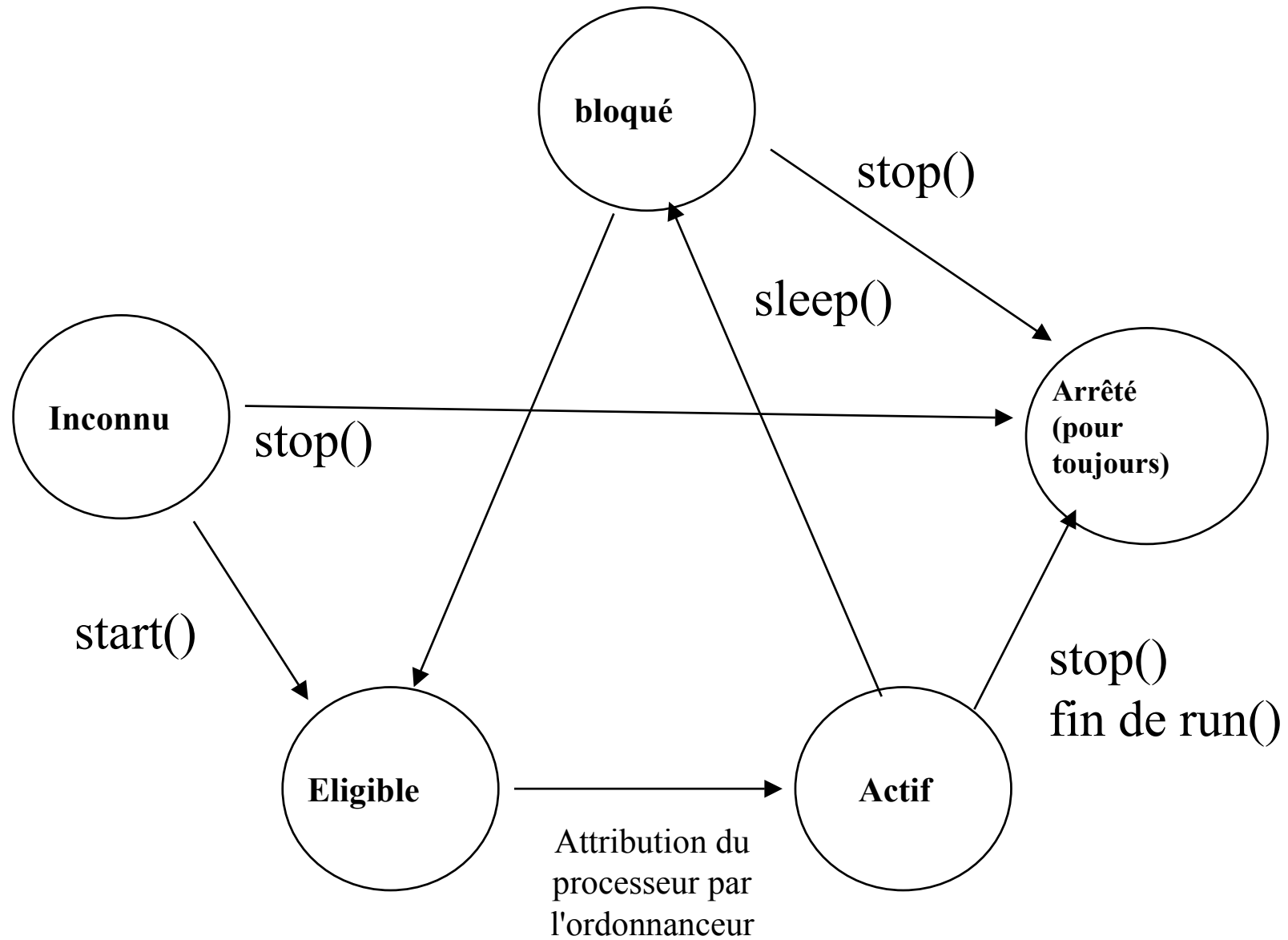
## Les méthodes publiques

- *void start();*
- *void run();*
- *void stop();*
- *static void sleep(long ms);*
- *static native Thread currentThread();*
- ...



# Etats d'un "Thread"

---



# Un exemple simple par *extends*

---

- class Ping extends Thread{
- **public void run(){**
- System.out.println("Ping");
- }
- }
- class Pong extends Thread{
- **public void run(){**
- System.out.println("Pong");
- }
- }
- public class ThreadPingPong{
- public static void main(String args[]){
- Ping ping = new Ping();
- Pong pong = new Pong();
- ping.start();
- pong.start();
- }}

Trace d 'exécution <i>Ping</i> <i>Pong</i>
--

# Un autre exemple [extrait de Java Threads p17]

---

- `import java.awt.*;`
- `class TimerThread extends Thread{`
- `private Component comp;`
- `private int timeDiff;`
- `public TimerThread(Component comp, int timeDiff){`
- `this.comp = comp;`
- `this.timeDiff = timeDiff;`
- `}`
- `public void run(){`
- `while( true){`
- `try{`
- `comp.repaint(); // déclenchement cyclique`
- `sleep(timeDiff);`
- `} catch(Exception e) {}`
- `}`
- `}`
- `}`

# Un autre exemple suite

---

- `public class Animation extends java.applet.Applet {`
- `private Image Diapositives[];`
- **`private TimerThread timer;`**
  
- `public void init() {`
- `Diapositives = new Image[100];`
- `// int i = 0;String Nom; // chargement des Diapositives`
- `Diapositives[i] = getImage(getCodeBase(), Nom + ".jpg");`
- `}`
  
- `public void start() {`
- `timer = new TimerThread(this, 1000); //this est un Component`
- **`timer.start();`**
- `}`
  
- `public void stop() {`
- **`timer.stop(); timer = null;`**
- `}`
- `public void paint( Graphics g) {`
- `g.drawImage(Diapositives[i], 0, 0, null);`
- `}}`

# Un autre exemple par *implements*

---

- class Thread2 **implements Runnable**{
- public void run(){
- while(true){
- System.out.println("ping");
- }
- }
- }
  
- public class UneApplette extends java.applet.Applet{
- public void init(){
- **Runnable** r = new Thread2();
- Thread t = new Thread( r); // ou t = new Thread( new Test());
- t.start();
- }
- }

```
public interface Runnable{  
    public abstract void run();  
}
```

# Un « bon » style par *implements*

---

```
public class UneApplet2 extends java.applet.Applet implements Runnable{

    private Thread t;

    public void init(){
        Thread t = new Thread( this);
        t.start();
    }

    public void stop(){ t.interrupt(); }

    public void run(){
        while(!t.interrupted()){
            ....
        }
    }
}
```

# Création de Thread en autonome

---

```
class AutoThread implements Runnable{
    private Thread local;
    public AutoThread(){
        local = new Thread( this);
        local.start();
    }
    public void run(){
        if( local ==Thread.currentThread()){
            while(true){
                System.out.println("dans AutoThread.run");
            }
        }
    }
}
```

```
public class Thread3 {
    public static void main(String args[]) {
        AutoThread auto1 = new AutoThread();
        AutoThread auto2 = new AutoThread();
        while(true){
            System.out.println("dans Thread3.main");
        }
    }
}
```

Trace d'exécution  
*dans AutoThread.run*  
*dans AutoThread.run*  
*dans AutoThread.run*  
*dans AutoThread.run*  
*dans AutoThread.run*  
*dans Thread3.main*  
*dans Thread3.main*  
*dans AutoThread.run*

# Passage de paramètres

---

```
class AutoThreadParam implements Runnable{
    private Thread local;
    private String param;
    public AutoThreadParam(String param) {
        this.param = param;
        local = new Thread( this);
        local.start();
    }
    public void run(){
        if( local == Thread.currentThread()){
            while(true){
                System.out.println("dans AutoThreadParam.run"+ param);
            } } } }
```

```
public class Thread4 {
    public static void main(String args[]) {
        AutoThreadParam auto1 = new AutoThreadParam("auto 1");
        AutoThreadParam auto2 = new AutoThreadParam ("auto 2");
        while(true){
            System.out.println("dans Thread4.main");
        } } }
```



# Arrêts d 'un Thread

---

- Sur le retour de la méthode *run()* le Thread s 'arrête
- Si un autre Thread invoque la méthode *interrupt()* celui-ci s 'arrête en levant une exception
- Si n'importe quel Thread invoque *System.exit()* ou *Runtime.exit()*, tous les Threads s 'arrêtent
- Si la méthode *run()* lève une exception le Thread se termine ( avec libération des ressources)
- *destroy()* et *stop()* ne sont plus utilisés, non sûr

# Quelques méthodes publiques de "Thread"

---

- *// page 476, chapter 25, Java in a Nutshell, 2nd edition*
- *final void suspend();*
- *final void resume();*
- *static native void yield();*
- *final native boolean isAlive();*
- *final void setName(String Nom);*
- *final String getName();*
- *static int enumerate(Thread threadArray[]);*
- *static int activeCount();*
- ...
- *public void run(); // class Thread ... implements Runnable*
- ...

# Priorités et Ordonnancement

---

- Pré-emptif, le processus de plus forte priorité devrait avoir le processeur
- Arnold et Gosling96 : *When there is competition for processing resources, threads with higher priority are generally executed in preference to threads with lower priority. Such preference is not, however, a guarantee that the highest priority thread will always be running, and thread priorities cannot be used to implement mutual exclusion.*
- Priorité de 1 à 10 (par défaut 5). Un thread adopte la priorité de son processus créateur (*setPriority(int p)* permet de changer celle-ci)
- Ordonnancement dépendant des plate-formes (.....)
  - Tourniquet facultatif pour les processus de même priorité,
  - Le choix du processus actif parmi les éligibles de même priorité est arbitraire,
  - La sémantique de la méthode *yield()* n'est pas définie, certaines plate-formes peuvent l'ignorer ( en général les plate-formes implantant un tourniquet)

Et le ramasse-miettes ?

# Concurrence, synchronisation en Java

---

Moniteur de Hoare 1974

Moniteur en Java : usage du mot-clé *synchronized*

*// extrait de java.lang.Object;*

*Attentes*

*final void wait() throws InterruptedException*

*final native void wait(long timeout) throws InterruptedException*

*...*

*// Notifications*

*final native void notify()*

*final native void notifyAll()*

# Moniteur de Hoare [PS90]

---

- Un moniteur définit une forme de module partagé dans un environnement parallèle
- Il encapsule des données et définit les procédures d'accès à ces données
- Le moniteur assure un accès en exclusion mutuelle aux données qu'il encapsule
  
- Synchronisation par les variables conditions :
  - Abstraction évitant la gestion explicite de files d'attente de processus bloqués*
- L'opération *wait* bloque l'appelant
- L'opération *notify* débloque éventuellement un processus bloqué à la suite d'une opération *wait* sur le même moniteur
  - Variables conditions de Hoare (le processus signaleur est suspendu au profit du processus signalé); Variables conditions de Mesa(ordonnancement moins stricte)

# Moniteur en Java, *usage de synchronized*

---

- Construction *synchronized*
- *synchronized(obj){*
- *// ici le code atomique sur l 'objet obj*
- *}*
  
- *class C {*
- *synchronized void p(){ .....}*
- *}*
- *///// ou /////*
- *class C {*
- *void p(){*
- *synchronized (this){ .....}*
- *}}*

# Une ressource en exclusion mutuelle

---

```
public class Ressource {  
    private double valeur;  
  
    synchronized double lire(){  
        return valeur;  
    }  
  
    synchronized void ecrire(double v){  
        valeur = v;  
    }  
}
```

Le langage garantit l'atomicité en lecture et écriture des variables des types primitifs comme *byte, char, short, int, float, reference (Object)* mais ce n'est pas le cas pour *long* et *double*

# Une file de messages (héritage et moniteur)

---

```
import java.util.Vector;
public class FileDeMessages {
    private Vector laFile = new Vector();

    public synchronized void envoyer(Object obj) {
        laFile.addElement(obj);
    }

    public synchronized Object recevoir () {
        if (laFile.size() == 0){
            return null;
        }else{
            Object obj = laFile.firstElement();
            laFile.removeElementAt(0);
            return obj;
        }
    }
}
```



# Mécanisme de synchronisation[PS90]

---

- L'exclusion mutuelle assurée sur les procédures d'accès n'est pas une règle d'ordonnancement universelle
- L'évolution d'un processus peut-être momentanément bloqué tant qu'une condition n'est pas vérifiée
- Lorsque la condition attendue devient vraie, un ou plusieurs processus peuvent continuer
- Mécanisme de synchronisation en Java lié au moniteur
  - la classe `java.lang.Object`
  - `wait()` et `notify()`

# Une ressource synchronisée

---

```
public class Ressource {
    private double valeur;
    private boolean valeurEcrit=false;

    synchronized double lire(){
        while(!valeurEcrit)
            try { wait();
                }catch(InterruptedException ie){}
        valeurEcrit = false;
        return valeur;
    }

    synchronized void ecrire(double v){
        valeur = v;
        valeurEcrit = true;
        notify();
    }
}
```

---

# Discussion

# Le joueur de ping-pong revisité

---

```
// Entrelacement de ping et de pong
public class TableEnSimple implements Table{
    private int coup = PING;
    private long nombreDeCoups = 0;

    public synchronized void jouerPing(){
        if (coup == PING) try{
            wait();
        }catch(InterruptedException e){};
        coup = PING;
        notify();
    }

    public synchronized void jouerPong(){
        if (coup == PONG) try{
            wait();
        }catch(InterruptedException e){};
        coup = PONG;
        notify();
    }
}
```

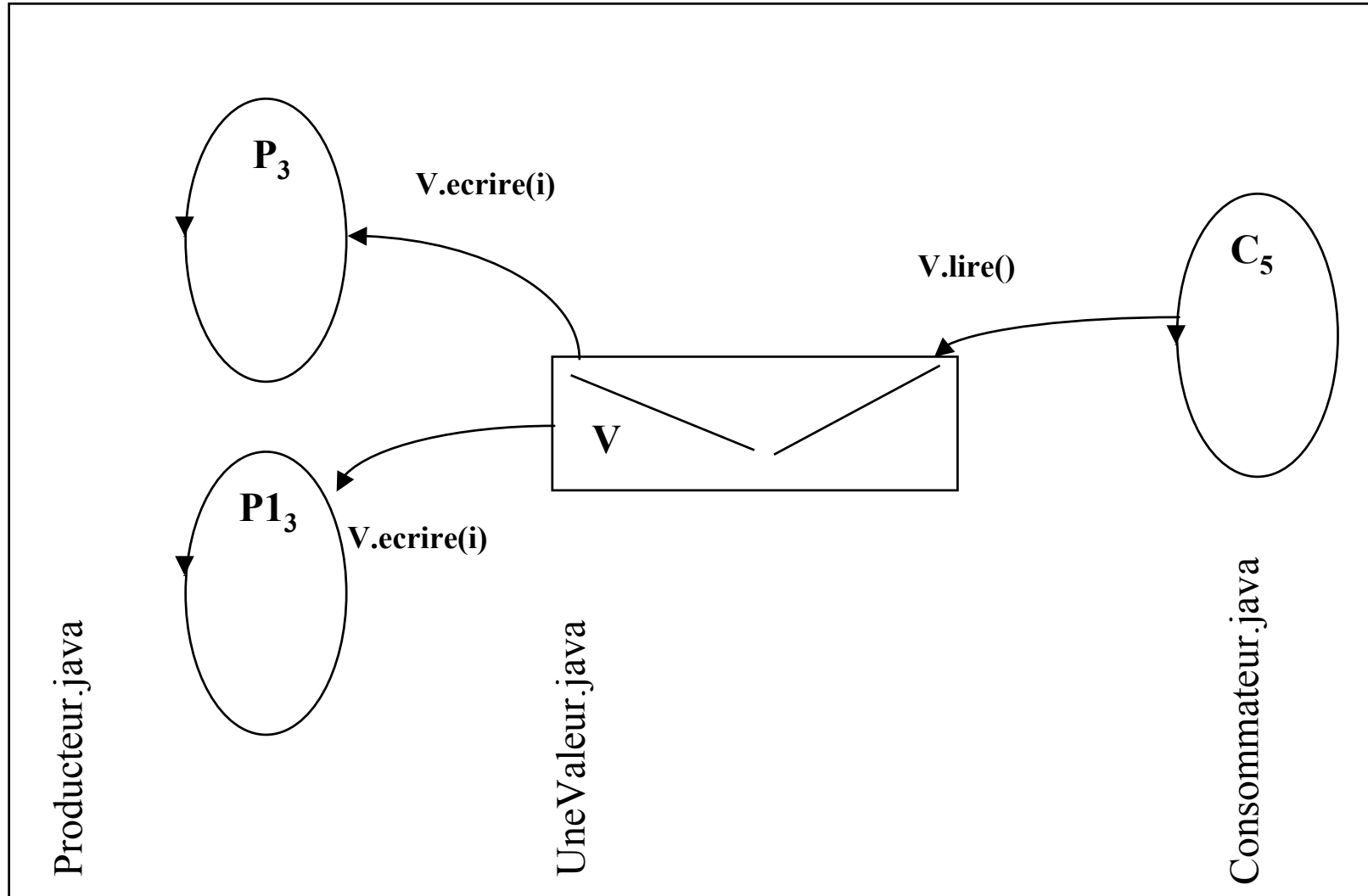
```
interface Table {
    final static int PING=0;
    final static int PONG=1;
    void jouerPing();
    void jouerPong();
}
```

# Le joueur de ping-pong revisité (2)

---

- `public class Joueur extends Thread{`
- `private static TableEnSimple table;`
- `private static Joueur jojo;`
  
- `public static void main(String args[]){`
- `while (true){`
- `table.jouerPing();` *// le main est un joueur ping*
- `}}`
  
- `public void run(){`
- `while (true){`
- `table.jouerPong();`
- `}}}`
  
- `static {`
- `table = new TableEnSimple();`
- `jojo = new Joueur();`
- `jojo.start(); }}`

# Moniteur et variable condition



Communication en synchronisation faible

# Utilisation : ProdCons.java

---

```
public class ProdCons {

    public static void main(String Args[]) {
        Thread Main = Thread.currentThread();

        UneValeur V = new UneValeur();
        Producteur P = new Producteur(V);
        Producteur P1 = new Producteur(V);
        Consommateur C = new Consommateur(V);

        P.start("Producteur P",3);
        P1.start("Producteur P1",3);
        C.start("Consommateur_C",5);
        try {
            Main.sleep(2000); // ou Thread.sleep
            P1.stop();
            Main.sleep(2000);
            P.stop();
            C.stop();
        } catch (Exception e) {}
    }
}
```

# Moniteur et variable condition: UneValeur.java

---

```
public class UneValeur {  
    private int val = 0;  
    private boolean val_Presente = false;  
  
    public synchronized int lire(){  
        if (!val_Presente){  
            try{ wait(); }catch(Exception e){}  
        }  
        val_Presente = false;  
        notify();  
        return val;  
    }  
    public synchronized void ecrire(int x){  
        if (val_Presente){  
            try{  
                wait();  
            }catch(Exception e){}  
        }  
        val = x; val_Presente = true;  
        notify();  
    }  
}
```



# Exemple d'utilisation : Producteur.java

---

```
public class Producteur extends NewThread{
    private UneValeur V;
    Producteur(UneValeur V){
        this.V = V;
    }

    public void run(){
        int i = 0;
        System.out.println("+" + this.toString());
        while(true){
            V.ecrire(i);
            System.out.println("V.ecrire(" + i +)");
            i++;
        }
    }
}

public class NewThread extends Thread{
    public synchronized void start(String Nom, int Priorite){
        setName(Nom);
        setPriority(Priorite);
        super.start();
    }
}
```

# Exemple d'utilisation : Consommateur.java

---

```
public class Consommateur extends NewThread{
    private UneValeur V;

    Consommateur(UneValeur V){
        this.V = V;
    }

    public void run(){
        int i;
        System.out.println("+ " + this.toString());
        while(true){
            i = V.lire();
            System.out.println("V.lire() == " + i );
        }
    }
}
```

# Moniteur et variable condition

---

```
public class UneValeur {  
    private int val = 0;  
    private boolean val_Presente = false;  
  
    public synchronized int lire(){  
        while (!val_Presente){  
            try{ wait();    }catch(Exception e){}  
        }  
        val_Presente = false;  
        notifyAll();  
        return val;  
    }  
    public synchronized void ecrire(int x){  
        while (val_Presente){  
            try{ wait();    }catch(Exception e){}  
        }  
        val = x;  
        val_Presente = true;  
        notifyAll();  
    }  
}
```

# notify et notifyAll

---

- Commentaires ...

# Schéma d'une condition gardée

---

- **synchronized** (this){
- while (!condition){
- try{
- **wait()**;
- } catch (Exception e) {}
- }
- }
- **synchronized** (this){
- condition = true;
- **notify()**; // ou **notifyAll()**
- }

notes : **synchronized(this)** engendre la création d'un moniteur associé à **this**  
**wait()**, ou **notify()** (pour **this.wait()** ou **this.notify()**)

**this.wait()** : mise en attente sur le moniteur associé à **this**, **this.notify()** réveille l'un des processus en attente sur ce moniteur (lequel ?)

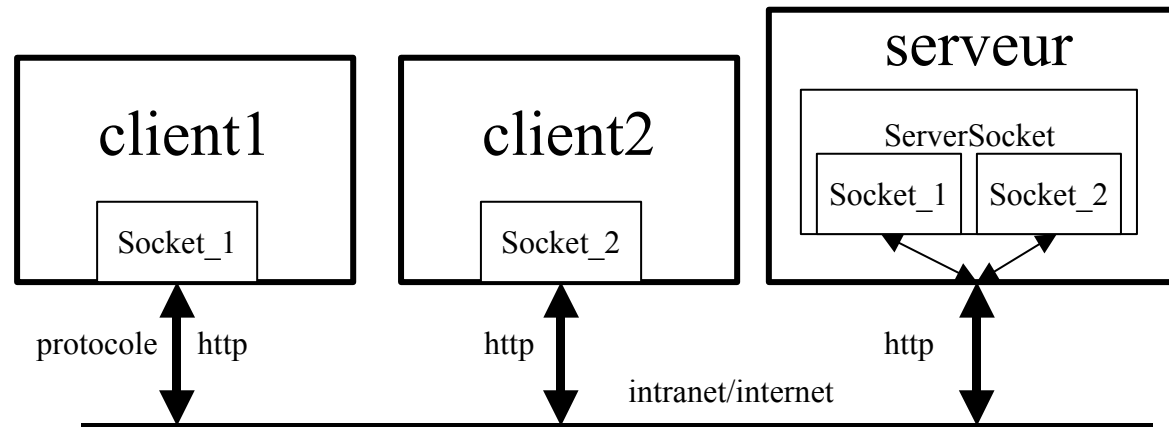
---

## Exemple

- **Un serveur Web**

# Serveur Web en Java

---



```
import java.net.Socket;  
import java.net.ServerSocket;
```

# Schéma d'un serveur en Java(1)

---

```
import java.net.*;
import java.io.*;
public class ServeurWeb_UneSeuleRequete{

    public static void main(String [] args) throws IOException{

        ServerSocket serveur = new ServerSocket(8080);
        Socket s = serveur.accept();

        try{
            PrintStream out;
            out = new PrintStream(s.getOutputStream());
            BufferedReader in;
            in = new BufferedReader(new InputStreamReader(s.getInputStream()));
            String req = in.readLine();

            // traitement de la requête selon le protocole HTTP
            ...
        }
    }
}
```



# Schéma d'un serveur en Java(2)

---

```
public class ServeurWeb{ // un Thread à chaque requête

    public static void main(String [] args) throws IOException{
        ServerSocket serveur = new ServerSocket(8080);
        while(true){
            Thread t = new Connexion(serveur.accept());
            t.start();
        }
    }

    private class Connexion implements Runnable{
        private Socket s;
        public Connexion(Socket s){this.s = s;}
        public void run(){
            PrintStream out = new PrintStream(s.getOutputStream());
            BufferedReader in;
            in = new BufferedReader(new InputStreamReader(s.getInputStream()));
            String req = in.readLine();

            // traitement de la requête selon le protocole HTTP
        }
    }
}
```

# Traitement d'une requête

---

```
StringTokenizer st = new StringTokenizer(req);
if((st.countTokens()>=2) && st.nextToken().equals("GET")){
try{String name = st.nextToken();
    if (name.startsWith("/")) name = name.substring(1);
    File f = new File(name);
    DataInputStream fis = new DataInputStream(new FileInputStream(f));
    byte[] data = new byte[(int)f.length()]; fis.read(data);

    out.print("HTTP/1.0 200 OK\r\n");
    out.print("Content-Type: text/html \r\n ");
    out.print("Content-Length: " + f.length() + "\r\n\r\n");
    out.write(data);
}catch(FileNotFoundException e){
    out.print("HTTP/1.0 404 Not Found\r\n");
    out.print("Content-Type: text/html\r\n");
    String str = "<HTML><BODY><H1>" + " Ce fichier n'existe pas !" +
        "</H1></BODY>\r\n\r\n";
    out.print("Content-Length: " + str.length() + "\r\n\r\n");
    out.print(str);
    ...
}
```

# Schéma d'un serveur en Java(3)

---

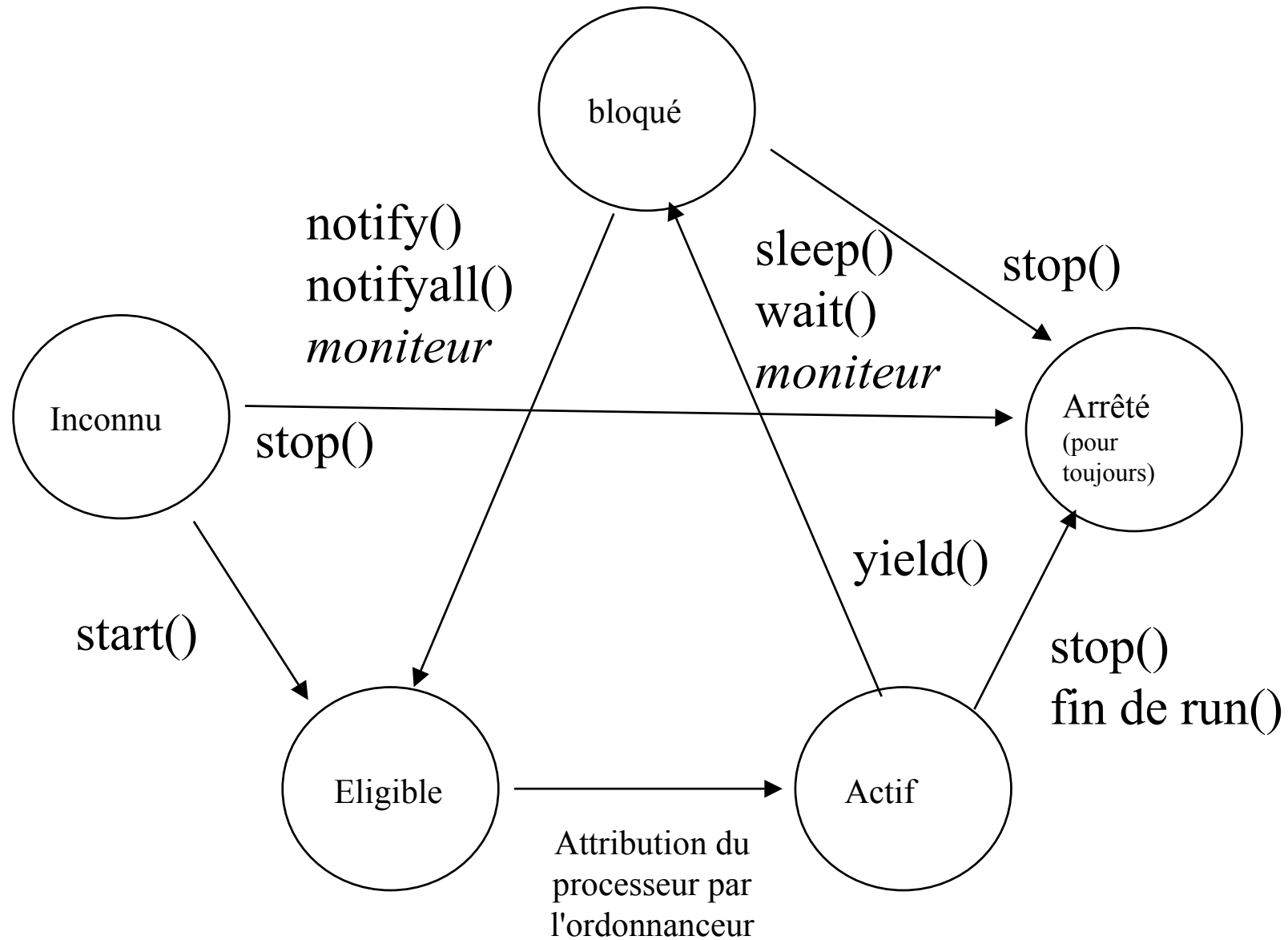
```
public class ServeurWeb{ // avec un Pool de Thread

    public static void main(String [] args) throws IOException{
        ServerSocket serveur = new ServerSocket(8080);
        while(true){
            pool.execute(new Connexion(serveur.accept()));
        }
    }

    private class Connexion implements Runnable{
        private Socket s;
        public Connexion(Socket s){this.s = s;}
        public void run(){
            PrintStream out = new PrintStream(s.getOutputStream());
            BufferedReader in;
            in = new BufferedReader(new InputStreamReader(s.getInputStream()));
            String req = in.readLine();

            // traitement de la requête selon le protocole HTTP
        }
    }
}
```

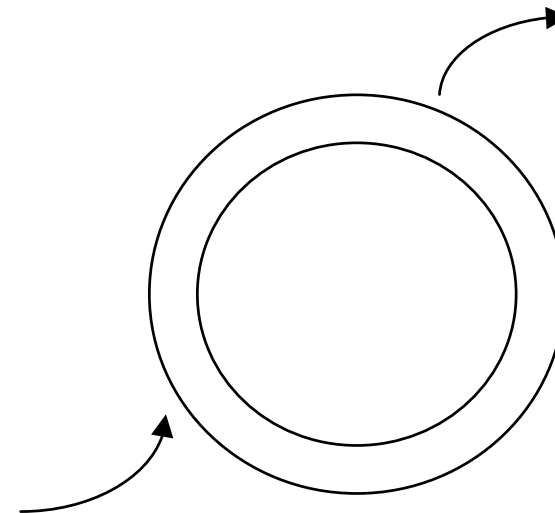
# Etats d'un "Thread"



# Moniteur et variable condition: UnBuffer.java

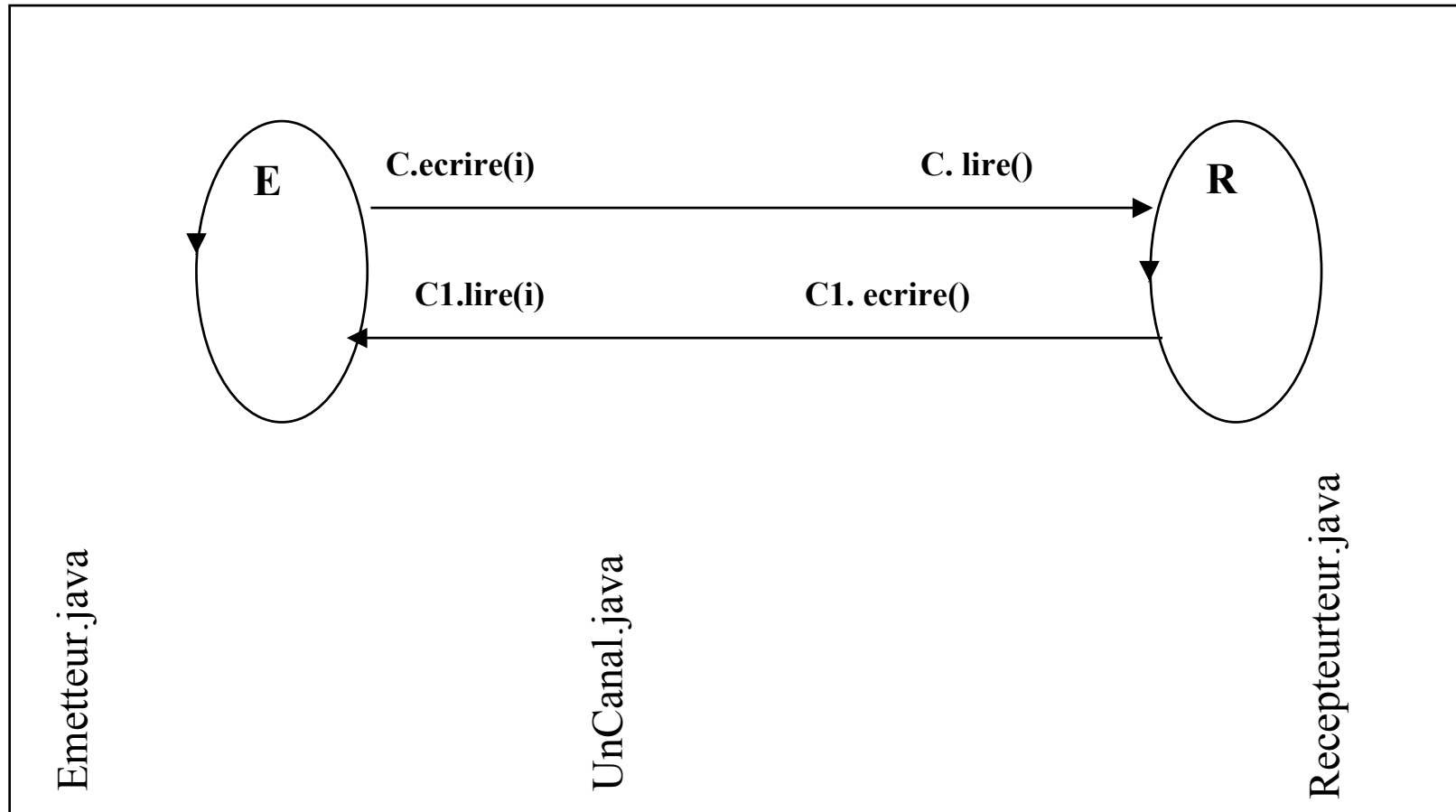
---

- class UnBuffer **extends** java.util.Vector {
- public UnBuffer ( int taille ) {super(taille); }
  
- public synchronized void deposer (Object item) {
- while (elementCount==capacity()) {
- try {
- wait();
- } catch (Exception e) {}
- }
- **notifyAll();**
- addElement(item);
- }
  
- public synchronized Object retirer () {
- while (isEmpty()) {
- try {
- wait();
- } catch (Exception e) {}
- }
- **notifyAll();**
- Object firstElement = firstElement();
- removeElementAt(0);
- return firstElement;
- }}



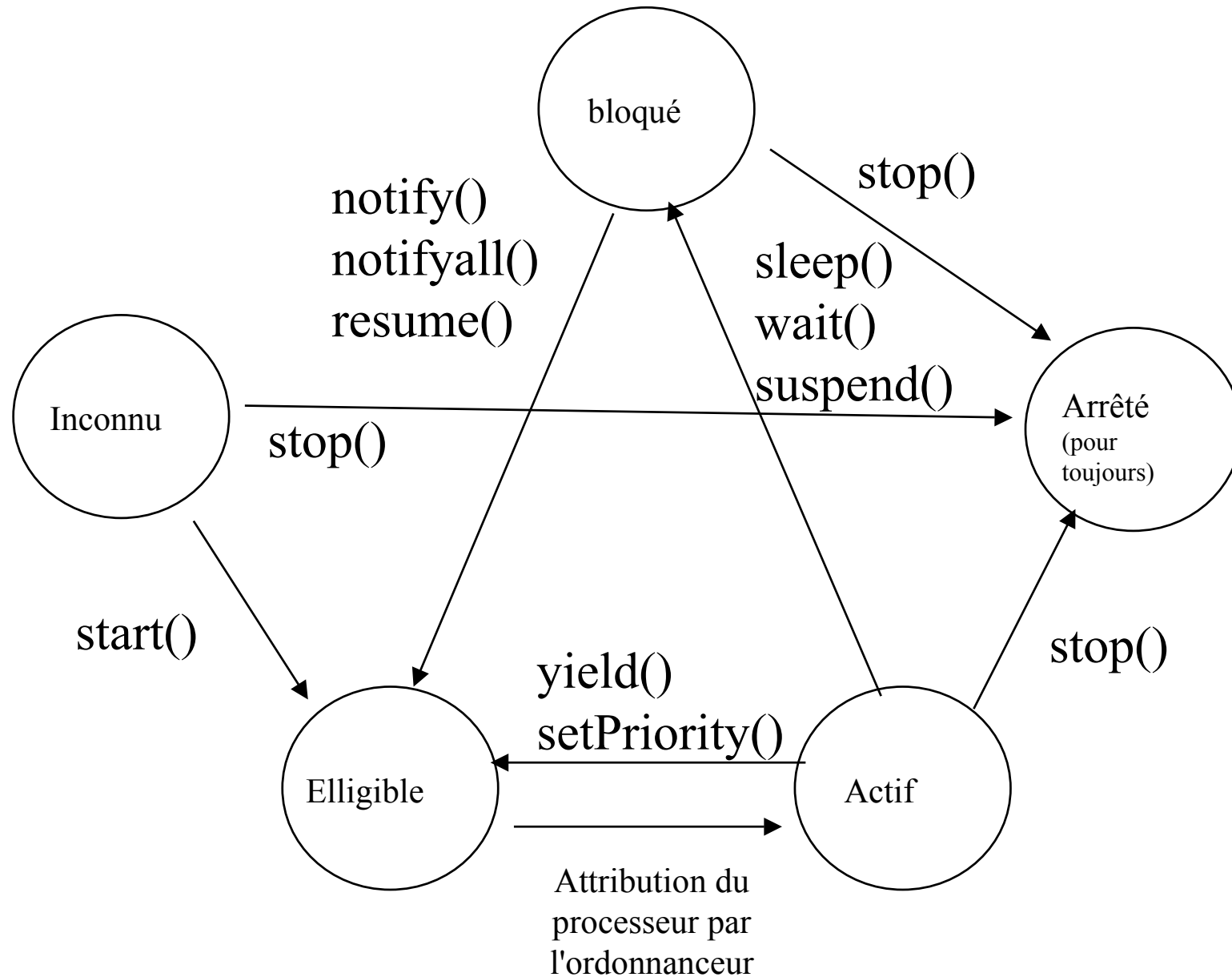
# Passage de message en Rendez-vous

Communication en synchronisation forte, uni-directionnelle, point à point, sans file d'attente( modèle CSP)



TestRendezVous.java

# Etats d'un "Thread"



# Ordonnancement

---

- A priorité égale : la spécification n'impose rien
  - Pas de soucis d'équité
  - A chaque ré-ordonnancement (si il existe) le processeur est attribué à un autre processus de priorité égale
  - Après un wait() c'est l'un des processus qui est choisi
- Sous Windows 95(JDK1.1.4), et MacOS (JDK 1.0.2)
  - Un tourniquet est déjà installé
- Sous Solaris 2.x (JDK 1.0.2 et 1.1)
  - Pas de tourniquet



# Ordonnanceur de type tourniquet

---

- `class SimpleScheduler extends Thread { // Java Threads page 139`
- `private int timeslice;`
- `SimpleScheduler(int t){`
- `timeslice = t;`
- `setPriority( Thread.MAX_PRIORITY);`
- `setDaemon(true);`
- `}`
- `public void run (){`
- `while(true){`
- `try{`
- `sleep(timeslice);`
- `}catch(Exception e) {}`
- `}`
- `}`
- `}`

# Exemple d'utilisation

---

- class Balle extends Thread{
- private String Id;
- Balle( String Id){
- this.Id = Id;
- }
- public void run(){
- while(true){
- System.out.println(Id);
- }
- }
- }

```
public class TestSimpleScheduler{  
  
    public static void main(String args[]){  
        new SimpleScheduler(100).start();  
        Balle ping = new Balle("ping");  
        Balle pong = new Balle("pong");  
  
        ping.start();  
        pong.start();  
    }  
}
```

# Ordonnanceurs

---

- Autres stratégies d'ordonnement
- Ordonneurs à échéance
  - <http://www-cad.eecs.berkeley.edu/~jimmy/java/>
  - <http://gee.cs.oswego.edu/dl/cpj/>

# interrupt() : levée de *InterruptedException*

- class Cyclic extends Thread{
- private Thread t;
- Cyclic(Thread t){ // interruption de t chaque seconde
- this.t = t;
- setPriority(Thread.MAX\_PRIORITY);
- }
- public void run (){
- while(true){
- try{
- sleep(1000);
- **t.interrupt();**
- }catch(Exception e){}
- }
- }
- }

# TestInterrupt()

---

- `public class TestInterrupt{`
- `public static void main(String args[]){`
- `Thread Main = Thread.currentThread();`
- `new Cyclic(Main).start();`
- `Object obj = new Object();`
- `while(true){`
- `try{`
- `synchronized(obj){`
- `obj.wait();`
- `}`
- `}catch(InterruptedException e){`
- `System.out.println("InterruptedException");`
- `// chaque seconde`
- `}`
- `}}}`

# java.lang.ThreadGroup

---

- Java.lang.ThreadGroup (p.477, Java in a Nutshell,2nd)
- Rassembler plusieurs Thread
  - Changer l'état de tous les Threads d'un groupe
    - suspend, resume, stop, setMaxPriority
  - Interroger l'état des Threads...
    - isDaemon, isDestroyed, parentOf...
- Hiérarchiser les groupes de Threads
  - ThreadGroup(ThreadGroup parent,...), getParent

# ThreadGroup, un exemple revisité : le Pipeline

---

- class Element implements Runnable{
- ....
- Element(**ThreadGroup group**,
- UnCanal entree, UnCanal sortie,Operation opr){
- ...
- **local = new Thread(group,this);**
- ... }}

- class Pipeline{
- ...
- **private ThreadGroup group;**
- public Pipeline(int size, Operation opr){
- **group = new ThreadGroup(this.toString());**
- for(int i=0;i<size-1;i++){
- new Element(**group**,pipe[i],pipe[i+1],opr);
- }
- ....
- public void detruire(){
- **group.stop();**
- }}

# Premières conclusions

---

- Mécanismes de bas-niveau
- Nécessité d'une méthodologie voir doug Lea
- Commentaires ...