IJA – TD 8

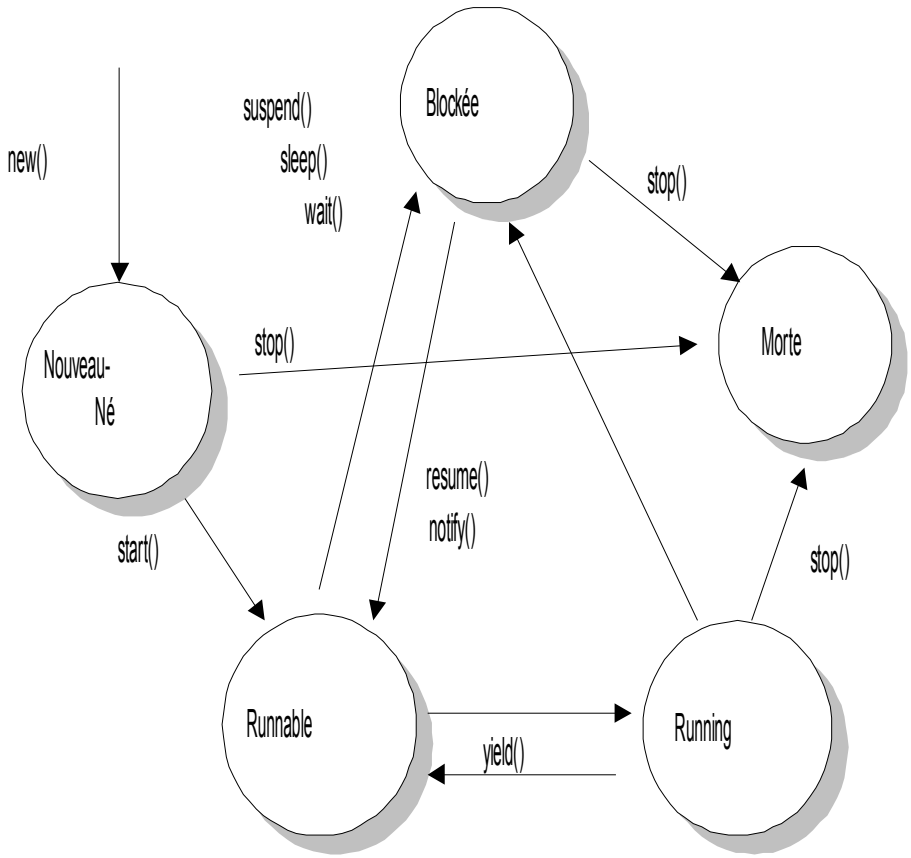
Les threads en Java

Les threads – Objectifs

- Thread = fil d'exécution dans un processus
- Permet au sein d'un même programme (processus) d'exécuter parallèlement plusieurs tâches, de manière indépendante ou de manière synchronisée
- Tout objet Java s'exécute dans au moins un thread

Les threads – Vie et mort

- Exécution de tâches en //
- Mémoire, Code et Ressources partagés
- Économie de ressources
- Un thread ~= méthode qui rend immédiatement la main
- Exemple événements (IHM, GC)
- + priorités
- + synchronisation
  - (moniteur, synchronized)
- Implantation dépendante du SE



Les threads – Syntaxe 1

```
public class Compteur extends Thread {
   public void run() {...}
}

Compteur c = new Compteur();
c.start();
```

- L'héritage à partir de Thread est contraignant car il empêche tout autre héritage

Les threads – Syntaxe 2

```
public class Compteur implements Runnable {
   public void run() {...}
}

Compteur c = new Compteur();
new Thread(c).start();
```

- L'implémentation de l'interface Runnable permet une grande flexibilité : possibilité d'héritage et d'implémentation d'autres interfaces

Exo

- Écrire le thread Compteur des deux manières possibles
- Tester l'exécution du thread à partir d'un client et à partir de l'objet lui-même

Les threads – Démarrage, arrêt

- `run, start` : démarrage du thread
- `sleep` : pause de x ms du thread
- `yield` : un autre thread peut prendre la main
- `interrup` : interruption du thread

- `isAlive` : le thread est vivant ?
- `isInterrupted` : le thread est interrompu ?

Les threads – Ordonnancement et priorités

- L'ordonnancement est en partie dépendant des implémentations
- Pour 2 threads de même priorité, par défaut : *round robin*
- T1 cède la place a T2 quand `sleep, wait,` bloque sur un `yield, synchronized`

- `getPriority, setPriority` : récupération ou changement de la priorité d'exécution d'un thread par rapport aux autres

Les threads – Synchronisation

- `synchronized` sur un objet : accès critique par un seul thread à la fois à cet objet

```
public class Compteur implements Runnable {
   private int c;
   public void run() {...}
   public void increment() {
    synchronized(this) { c++; }
   }
}
```

Les threads – Synchronisation

- Synchronisation temporelle :
  - `join` : attend la fin d'un thread
  - `wait` : thread bloqué jusqu'à ce qu'un autre thread appelle `notify` ou `notifyAll` (NB: `wait` libère le verrou)
  - `notify` : débloque un thread bloqué par `wait` (le premier en queue : FIFO)
  - `notifyAll` : débloque tous les threads bloqués par `wait`

Les threads – Exemple

```
public class Compteur implements Runnable {
   int c = 0;
   boolean continuer=true; // Pattern de l'inversion de contrôle
   public synchronized void stop() { this.continuer = false; }
   public void run() {
    while (continuer)  {
        this.c++;
        System.out.println ("Compteur = "+this.c);
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Exo

- Écrire un compteur dont le délai entre chaque incrémentation est paramétrable
- Écrire un client qui lance une ‘course de compteur’ et vérifier par l’affichage du compteur que les threads s’exécutent bien en parallèle
- *Modifier le compteur pour que le thread de tête attende son poursuivant lorsqu’il arrive sur un nombre divisible par 10 (à la maison)*

Exo (à travailler)

```
trier(debut, fin) {
    si (fin - debut < 2) { // Fin
            si (t[debut] > t[fin])
            echanger(t[debut], t[fin])
 }
 sinon {
            milieu = (debut + fin) / 2
            trier(debut, milieu)
            trier(milieu + 1, fin)
            triFusion(debut, fin) // Tri fusion des moitiés
    }
}
```

Exo (à travailler)

- Les deux tris qui sont effectués avant la fusion sont indépendants l'un de l'autre et il est donc facile de les faire exécuter en parallèle par deux threads
- Implanter une version « threadée » de cet algorithme de tri en les synchronisant avec la méthode `join`
- Implanter une autre version « threadée » de cet algorithme de tri en les synchronisant avec les méthodes `wait` et `notify`

IJA – TD 8

Les applets en Java

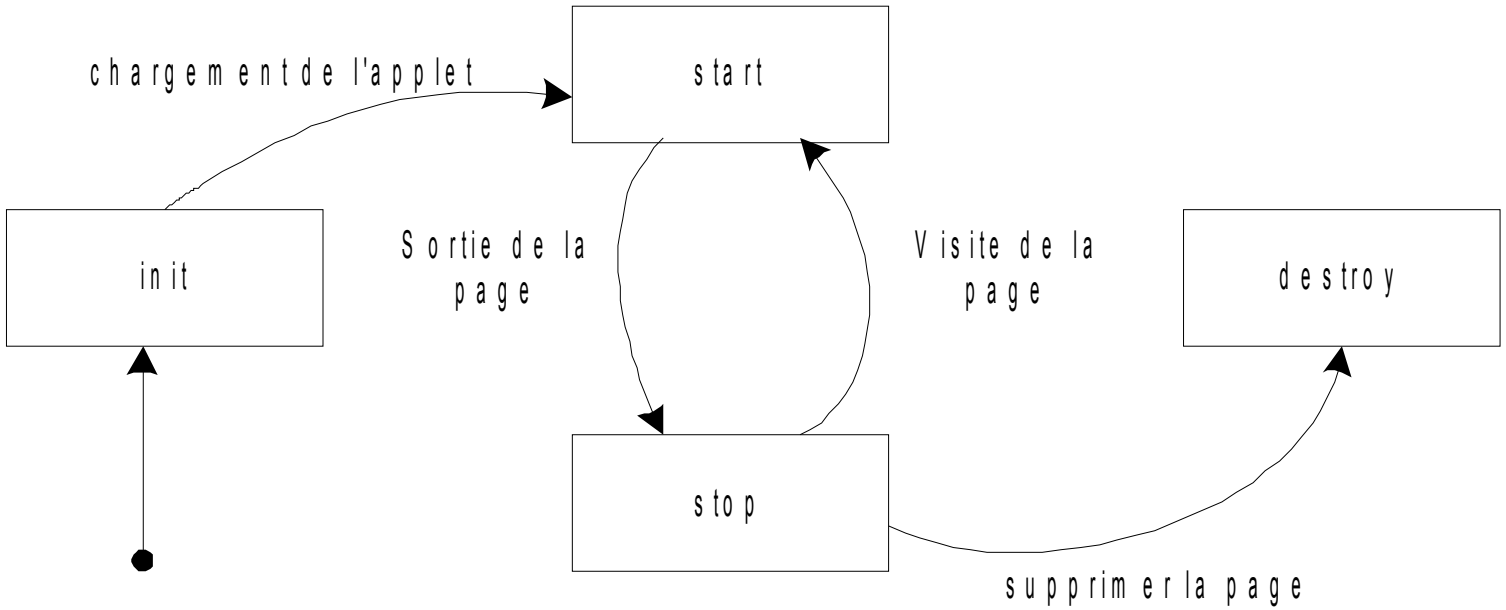
Les applets – Objectifs

- Une applet est une classe compilée héritant de `java.applet.Applet`.
- Une applet est à la fois un composant graphique (hérite de `Panel`) et un thread d'exécution

Les applets – Objectifs

- Elle est diffusée par un serveur web dans une page HTML
- Elle est téléchargée puis exécutée par le browser.
- Elle est soumise au *Security Manager* du browser
  - pas d'accès en lecture ni en écriture sur le disque du browser
  - connexion réseau uniquement sur le serveur d'origine
  - pas de chargement de librairie native
  - pas de lancement de processus, ...

Les applets – Vie et mort



Les applets – Syntaxe

```
public class BonjourApplet extends Applet {
   String msg = "bonjour";
   public void init() {...}
   public void start() {...}
   public void paint(java.awt.Graphics g)
   {g.drawString(msg, 20, 20);}
   public void stop() {...}
   public void destroy() {...}
}
```

```
<HTML>
  <BODY>
    <APPLET code="BonjourApplet.class"
   codebase="http://citi.insa-lyon.fr/~flemouel/applets/"
   width=300 height=200>
    <PARAM name="message" value="Bonjour">
   </applet>
  </BODY>
</HTML>
```

Les applets – Démarrage, arrêt

- `init` : méthode appelée au chargement de l'applet (avant le `start`)
- `start` : méthode appelée pour démarrer l'applet
- `isActive` : l'applet est vivante ?
- `stop` : méthode appelée pour arrêter l'applet
- `destroy` : méthode appelée pour détruire l'applet (libération des ressources, etc.) (`stop` doit toujours être appelée avant)

Les applets – Exemple

```
public class BonjourApplet extends Applet {
   StringBuffer buffer;
   public void init() {
    buffer = new StringBuffer();
    this.addItem("initialisation..."); }
   public void start() { this.addItem("démarrage..."); }
   public void stop() { this.addItem("arrêt..."); }
   public void destroy() { this.addItem("déchargement..."); }
   private void addItem(String newWord) {
    buffer.append(newWord); repaint(); }
   public void paint(Graphics g) {
    g.drawRect(0, 0, size().width - 1, size().height - 1);
    g.drawString(buffer.toString(), 5, 15); } // Message
}
```

Exo

- Écrire l'applet précédente affichant « Bonjour ! » à l'aide de la fonction `drawString` de la classe `Graphics`. Visualisez-là dans une page Web à l'aide d'un navigateur
- Modifier l'applet pour expérimenter d'autres méthodes de la classe `Graphics`, comme `drawLine`, `drawRect`, `drawImage`, `fillRect`, etc.

Exo (à travailler)

- A l'aide des interfaces `MouseListener` et `MouseMotionListener`, écrivez une applet `Tableau` qui permet de tracer une ligne sur simple pression du bouton de la souris
- A l'aide de la classe `Button` et de l'interface `ActionListener`, modifiez le tableau précédent pour y inclure trois fonctionnalités permettant de :
  - Passer d'un mode de tracé de ligne à tracé de points
  - Changer de couleur
  - Effacer le tableau