

Thread

- ♦ Un thread est un fil d'activité, un traitement exécutable en parallèle d'autres.
- ♦ Un thread possède une priorité. Les threads de forte priorité sont exécutés préférentiellement par rapport aux threads de faible priorité.
- ♦ La méthode **main** d'un programme est exécutée par un thread de la JVM. La JVM continue d'exécuter des threads jusqu'à :
 - un appel à la méthode **exit** de la classe **System**
 - tous les threads ont terminé leur méthode **run** ou ont lancé une exception au cours de l'exécution de leur méthode **run**

classe Thread

- **Implantation de l'interface Runnable**
la classe Thread définit des méthodes telles que :
 - `public Thread()`
 - `public Thread(Runnable target)`
 - `public long getId()`
 - `public int getPriority()`
 - `public void run()` //à redéfinir
 - `public void setPriority(int newPriority)`
 - `public static void sleep(long millis)`
 - `public void start()`
 - `public static void yield()`

Interface Runnable

- ♦ L'interface **Runnable** devrait être implantée par toute classe dont les instances doivent être exécutées par un thread. La classe doit définir une méthode sans paramètre appelée **run**.
- ♦ Un objet d'une classe qui implante **Runnable** peut s'exécuter sans dériver de la classe **Thread** en instanciant un **Thread** en se passant lui-même comme argument du constructeur.

Exemples de tâche

```
class CompteurV1 extends Thread {
    public void run() {
        for (int i=1; i<=100; i++) {
            System.out.println(i);
        }
    }
}

class CompteurV2 implements Runnable {
    public void run() {
        for (int i=1; i<=100; i++) {
            System.out.println(i);
        }
    }
}
```

Création de tâches

```
public class essaiTaches {
    public static void main(String arg[]) {
        Thread tache1 = new CompteurV1();
        Thread tache2 = new Thread(new CompteurV2());
        tache1.start();
        tache2.start();
    }
}
```

Suspension, reprise, arrêt

```
class Compteur extends Thread {
    private boolean arret = false;
    private boolean pause = false;
    public void run() {
        int i=1;
        while (!arret) {
            if (!pause) {
                System.out.println(getId() + " : " + i++);
                if (i>100) return;
            }
        }
    }
    public void suspend() { pause = true; }
    public void reprend() { pause = false; }
    public void termine() { arret = true; }
}
```

Répartition du temps entre tâches

- ♦ Trois priorités sont prédéfinies comme variables de la classe **Thread**
 - **MIN_PRIORITY**
 - **NORM_PRIORITY**
 - **MAX_PRIORITY**
- ♦ Pour intervenir sur la répartition du temps entre tâches, on intervient sur la priorité du thread en appelant **setPriority** avec une de ces valeurs comme argument.

Exemple de répartition du temps

```
class Compteur extends Thread {
    public Compteur(int priorite) {
        super();
        setPriority(priorite);
    }
    public void run() {
        for (int i=1; i<=100; i++) {
            System.out.println(getId() + " : " + i);
        }
    }
}
```

Exemple de répartition du temps...

```
public class Priorites {
    public static void main(String arg[]) {
        Thread t1 = new Compteur( MIN_PRIORITY),
            t2 = new Compteur(NORM_PRIORITY),
            t3 = new Compteur( MAX_PRIORITY);
        t1.start();
        t2.start();
        t3.start();
    }
}
```

Synchronisation de tâches

- ♦ La mise en attente de tâches qui veulent utiliser une ressource déjà utilisée passe par la synchronisation.
- ♦ On peut synchroniser des tâches :
 - en verrouillant la ressource utilisée
 - en verrouillant la méthode qui utilise la ressource

Exemple1 de synchronisation

```
public class synchronisation {
    public static void main(String arg[]) {
        new Impression("doc1").start();
        new Impression("doc2").start();
    }
}

class Impression extends Thread {
    private String nomDocument;
    public Impression(String nom) { nomDocument = nom; }
    public void run() {
        synchronized(System.out) {
            for(int i=1; i<=6; i++) {
                System.out.println("page "+i+" de "+nomDocument);
            }
        }
    }
}
```

Exemple2 de synchronisation

```
class synchronisation {
    public static void main(String arg[]) {
        Imprimante i = new Imprimante();
        new Impression("doc1", i).start();
        new Impression("doc2", i).start();
    }
}

class Imprimante {
    public synchronized void imprime(String[] texte) {
        for (int i=0; i<texte.length; i++)
            System.out.println(s[i]);
    }
}
```

Exemple2 de synchronisation...

```
class Impression extends Thread {
    private String nomDoc;
    private Imprimante impr;
    private String texte[];
    Impression(String nomDoc, Imprimante impr) {
        this.nomDoc = nomDoc;
        this.impr = impr;
    }
    public void run() {
        texte = new String[6];
        for(int i=0; i<6; i++) {
            texte[i] = "page "+i+" de "+nomDoc;
        }
        impr.imprime(texte);
    }
}
```

Threads dans une application Java

- ♦ Un programme en mode *console* exécute au moins 2 threads :
 - Celui qui exécute la méthode **main**
 - Le Garbage Collector
- ♦ Une application graphique y ajoute :
 - Le thread de gestion des événements graphiques (EDT : Event Dispatching Thread)