JAVA

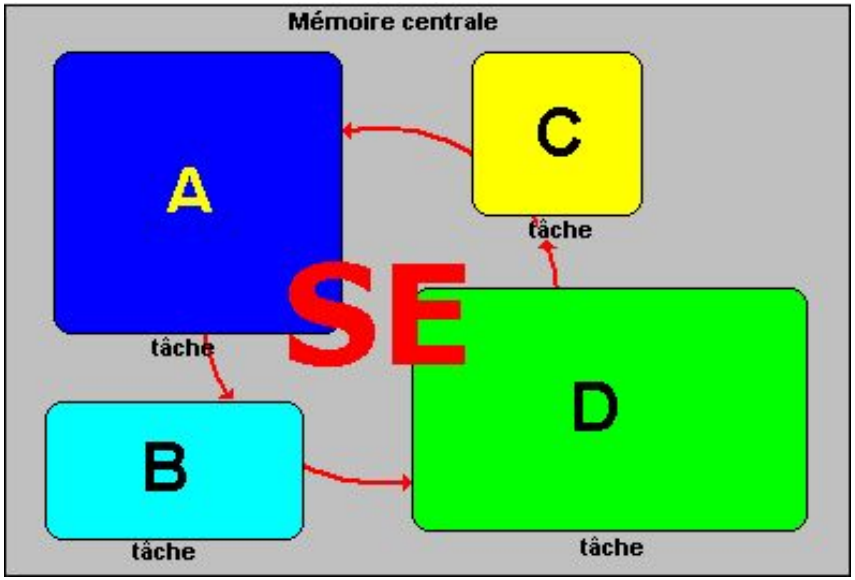
Les Thread

Introduction

- Les threads ou “processus légers” sont des unités d'exécution autonomes qui peuvent effectuer des tâches, en parallèle avec d'autres thread.
- Le flot de contrôle d'un thread est séquentiel. Plusieurs threads peuvent être associés à un “processus lourd”.
- En Java, le processus lourd est la JVM.
- Les threads coopèrent entre eux en échangeant des valeurs par la mémoire commune (du processus lourd)
- Grâce au “multi-thread”, même sur une machine monoprocesseur, l'ordinateur donne l'impression d'effectuer plusieurs tâches en parallèle.

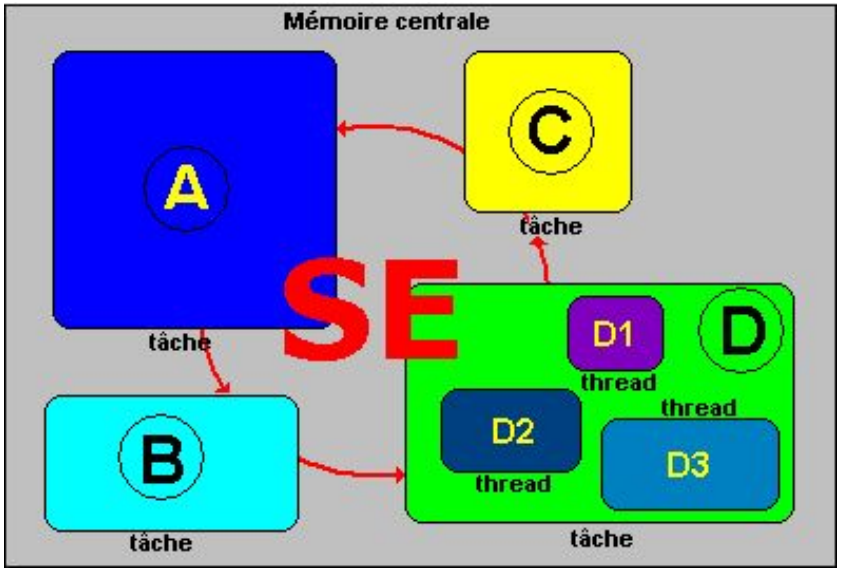
Introduction

- Le noyau du système d'exploitation SE, conserve en permanence le contrôle du temps d'exécution en distribuant cycliquement des tranches de temps à chacune des applications A, B, C et D.
- Une application représente dans le système un processus



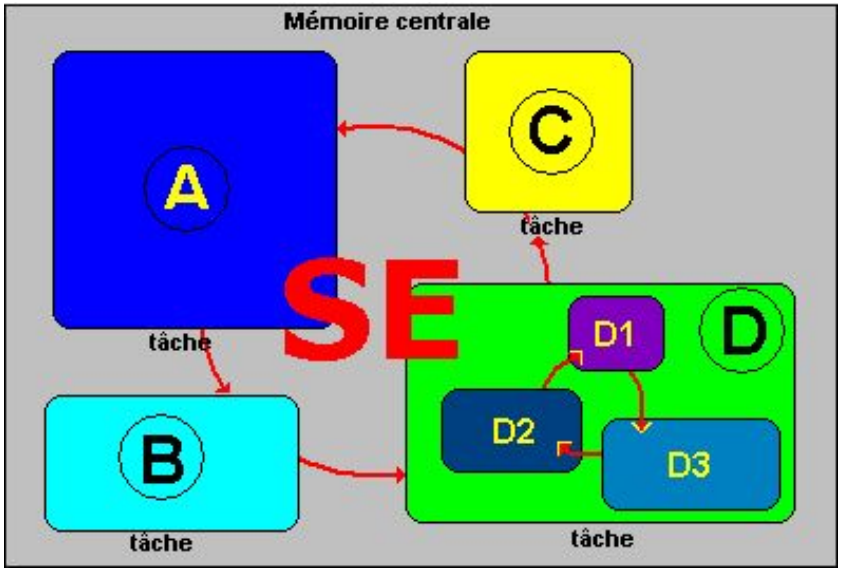
Introduction

- chaque processus peut lui-même fonctionner comme le système d'exploitation en lançant des sous-tâches internes au.
- Ces sous-tâches sont nommées Threads. Ci-dessous nous supposons que l'application D exécute en même temps les 3 Threads D1, D2 et D3



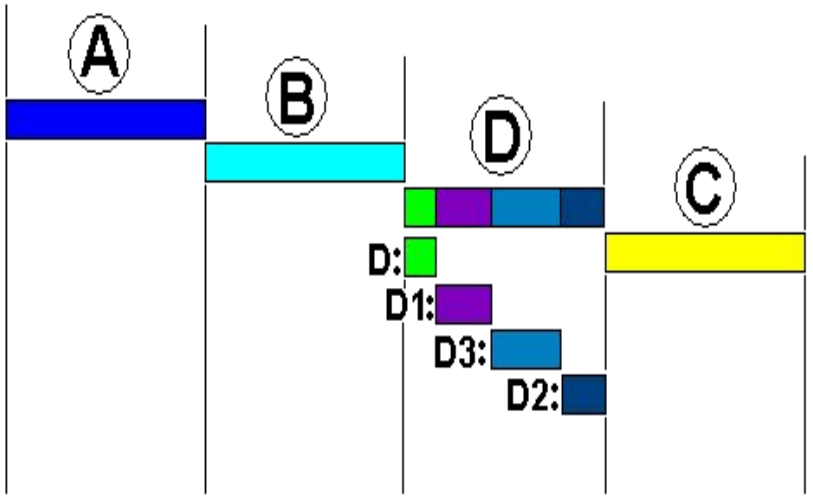
Introduction

- La commutation entre les threads d'un processus fonctionne de le même façon que la commutation entre les processus, chaque thread se voit alloué cycliquement, lorsque le processus D est exécuté une tranche de temps.
- Le partage et la répartition du temps sont effectués uniquement par le système d'exploitation



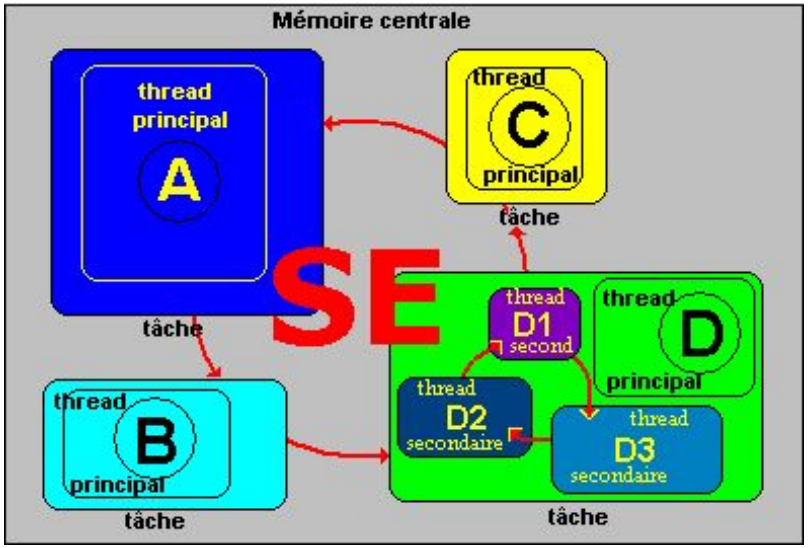
Introduction

- Le système alloue le même temps d'exécution à chaque processus
- Lorsque le tour vient au processus D de s'exécuter dans sa tranche de temps, il exécutera une sous-tranche pour D1, pour D2, pour D3 et attendra le prochain cycle.



Introduction

- Lorsqu'un programme Java s'exécute le processus associé comporte automatiquement un thread appelé thread principal.
- les autre thread s'appellent des threads secondaires.



Les Threads en Java

- En Java c'est l'interface **Runnable** qui permet l'utilisation des threads
- Cette interface est implantée par la classe **Thread** du package **java.lang**
- Java offre deux façons différentes pour définir un threads :
    - Soit par **implémentation** de l'interface **Runnable**.
    - Soit par **héritage** de la classe j**ava.lang.Thread**

La class Thread

- La classe Thread est définie dans le package java.lang. Elle implémente l'interface Runnable .
- Elle possède plusieurs constructeurs :
  - Thread()
  - Thread(String name)
  - Thread(Runnable target)
  - Thread(ThreadGroup group, String name)
  - ...
- Une fois créé, on peut configurer cet objet pour:
  - Lui associer une priorité
  - l'exécuter en invoquant sa méthode **start()** qui invoquer la méthode **run()**
  - **...La** methode run() dans la class Thread est vide !!!

La class Thread

- Créer une instance de type Thread dont l'implémentation de la méthode run() va contenir les traitements à exécuter.

```
public class Compte {
    public static void main(String[] args) {
        Thread t = new Thread() {
            public void run() {
                System.out.println("Un traitement");
            }
        };
        t.start();
    }
}
```

La class Thread

- Et Si la classe **Compte** veut étendre une **autre** classe et en faire une sous-classe de **Thread** également?

```
class Compte extends Thread {
   int valeur;

   Compte(int val) {
      valeur = val;
   }

   public void run() {
      try {
         for (;;) {
            System.out.print(valeur + " ");
            sleep(100);
         }
      } catch (InterruptedException e) {
            return;
        }
   }

   public static void main(String[] args) {
      new Compte(1).start();
      new Compte(2000).start();
   }
}
```

L’interface Runnable

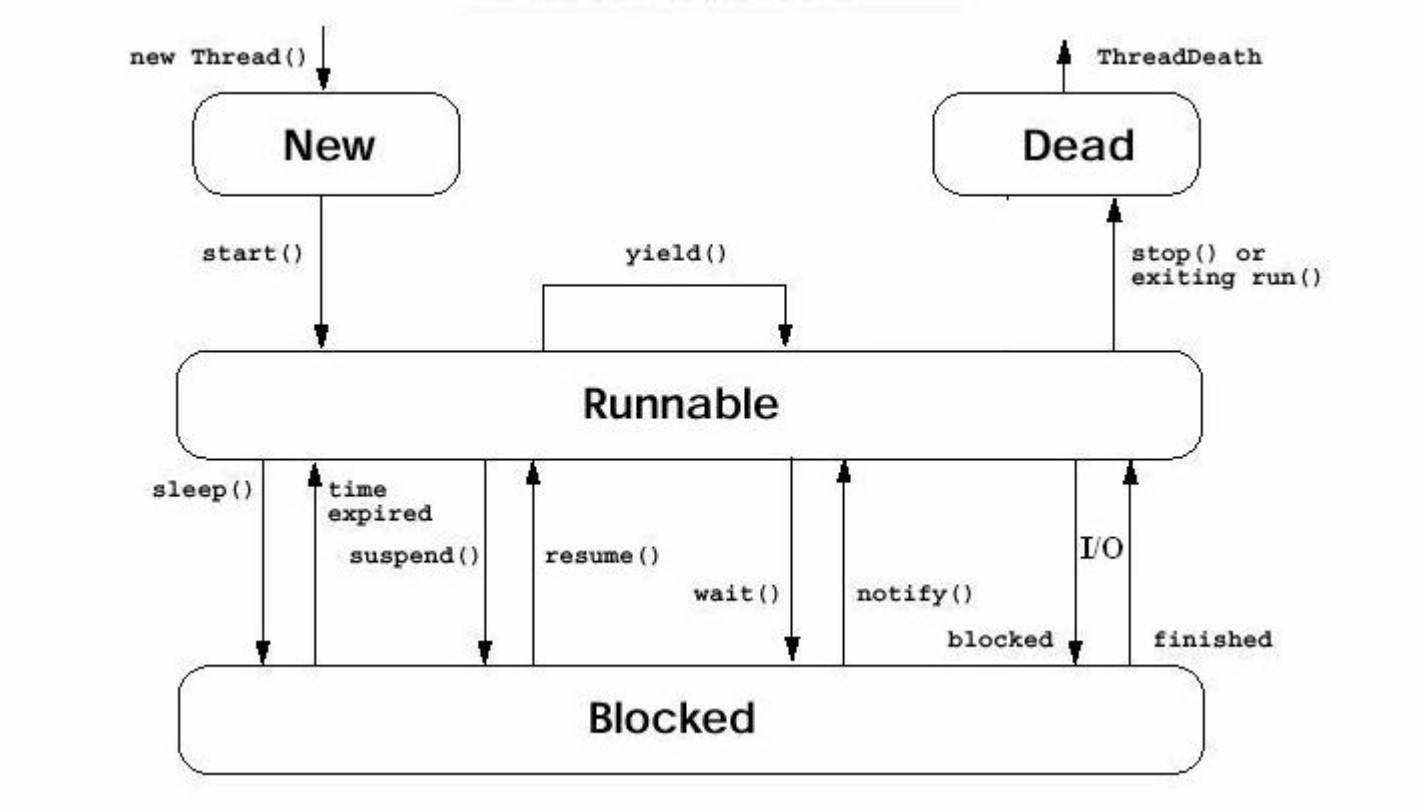
- Cette interface ne définit qu'une seule méthode : void run().
- Dans les classes qui implémentent cette interface, la méthode run() doit être redéfinie pour contenir le code des traitements qui seront exécutés dans le thread.

```
class Compte implements Runnable {
   int valeur;

   Compte(int val) {
      valeur = val;
   }

   public void run() {
      try {
         for (;;) {
            System.out.println(valeur + " ");
            Thread.sleep(100);
         }
      } catch (InterruptedException e) {
            return;
      }
   }

   public static void main(String[] args) {
      Runnable compte1 = new Compte(1);
      Runnable compte2 = new Compte(2000);
      new Thread(compte1).start();
      new Thread(compte2).start();
   }
}
```

L’interface Runnable vs la class Thread

- Il est préférable d'utiliser l'implémentation de Runnable car :
  - elle permet à la classe d'hériter au besoin d'une classe mère
  - elle permet une meilleure séparation des rôles
  - elle évite des erreurs car il suffit simplement d'implémenter la méthode run()
- Il est possible d'utiliser une instance de type Runnable pour plusieurs threads si l'implémentation est thread-safe.
- Il ne faut pas invoquer la méthode run() d'un thread car les traitements seront exécutés dans le thread courant mais ne seront pas exécutés dans un thread dédié.

Cycle de vie d'un Thread



Exercice

Nous vous proposons de programmer une simulation du problème *du robinet qui remplit d'eau une baignoire qui fuit*.

- débit du robinet est connu et paramétrable.
- La baignoire a une fuite dont le débit est connu et paramétrable.
- Dès que la baignoire est entièrement vide on colmate la fuite.
- On arrête le programme dès que la baignoire est pleine que la fuite soit colmatée ou non.