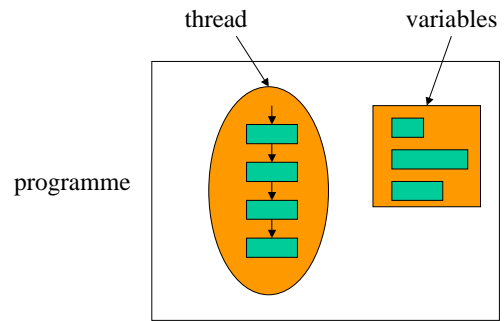


# Chapitre 7

## Les Threads

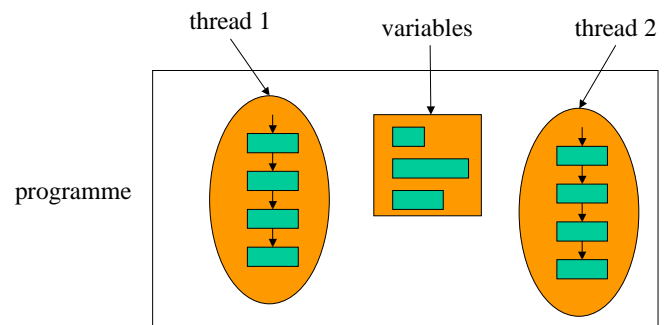
## Un programme avec 1 thread



## Plan

- Introduction
- Synchronisation

## Un programme avec 2 threads



## Introduction

## Langage multithread

- Le langage Java incorpore des primitives de multithread au sein même du langage grâce aux classes :  
Thread                      ThreadGroup,  
ThreadLocal                ThreadDeath
- Les langages C et C++ doivent effectuer des appels aux primitives de multithread du système d'exploitation.

## Thread

- Classiquement, un programme est composé d'un simple flux séquentiel d'exécution que l'on appelle thread (« fil »).
- Toutefois, il est possible qu'un programme soit composé de plusieurs threads qui s'exécutent de façon simultanée.
- Il est à noter que les données du programme sont accessibles par les différents threads.

## Interface « non réactive »

- Imaginons une interface graphique avec 2 boutons et 2 champs de saisie.
- Lorsqu'on clique sur l'un des boutons, un texte est placé dans un champ de saisie après une temporisation.
- L'interface graphique n'est plus réactive tant que la réponse à l'événement n'est pas finie d'être apportée.
- Exemple : [InterfaceNonReactive.java](#)

## Interface « réactive »

- Il est nécessaire de lancer un nouveau thread lorsque le temps d'exécution de la réponse à apporter à un événement peut s'avérer long.
- Exemple : [InterfaceReactive.java](#)

05/12/03

Les threads

page 9

## Partage de la mémoire

- Des données peuvent être partagées par plusieurs threads.
- Le plus souvent, il est nécessaire d'établir une politique de synchronisation.
- Sans cela, il est possible par exemple qu'un thread modifie une donnée pendant qu'un autre thread la consulte.
- Il peut en résulter des états incohérents

05/12/03

Les threads

page 13

## Créer un thread

- Il est nécessaire de procéder en trois étapes :
  - créer une sous-classe de Thread et redéfinir la méthode run()
  - créer un objet de cette sous-classe
  - exécuter la méthode start() sur cet objet
- Exemple : [PlusFort.java](#)

05/12/03

Les threads

page 10

## Objets « moniteurs »

- Une synchronisation entre threads est nécessaire (la plupart du temps) lors d'un accès à un objet.
- Pour mettre en place cette synchronisation, il suffit d'utiliser le mot-clef synchronized dans le contexte de l'objet.
- Un tel objet s'appelle alors un moniteur.

05/12/03

Les threads

page 14

## Une alternative

- Il est possible d'implémenter l'interface Runnable qui comporte la méthode :

```
public void run()
```
- Il est néanmoins plus judicieux d'étendre la classe Thread (qui par ailleurs implémente cette interface).

05/12/03

Les threads

page 11

## Verrou

- Un moniteur dispose d'un verrou.
- Ce verrou est activé :
  - lors de l'appel à une méthode synchronized de la classe du moniteur
  - lors de l'accès à un bloc synchronized portant sur le moniteur.
- Tout thread tentant alors d'exécuter une méthode ou d'accéder à un bloc synchronized de ce moniteur doit patienter.
- Le verrou est désactivé à la fin de la méthode synchronized.

05/12/03

Les threads

page 15

## Synchronisation

## Techniques de synchronisation

- Il est possible de gérer l'accès à des données partagées de 3 manières :
  - en contrôlant la synchronisation au niveau d'une méthode
  - en contrôlant la synchronisation au niveau d'un bloc
  - en contrôlant la synchronisation avec les méthodes wait et notify (notifyAll)

05/12/03

Les threads

page 12

05/12/03

Les threads

page 16

## Synchronisation « méthode »

- Dans la classe de l'objet dont on souhaite contrôler l'accès aux données, il est possible de définir certaines méthodes sensibles comme étant synchronized.
- Il s'agit d'un simple mot-clef qui se place devant l'en-tête de la méthode.
- Il n'est pas possible que 2 threads exécutent simultanément 2 méthodes synchronized.
- Une file d'attente est alors gérée.

05/12/03

Les threads

page 17

## Exemple 1/2

- Considérons 2 cambrioleurs qui « vident » en parallèle le coffre d'une banque.
- Les 2 cambrioleurs effectuent plusieurs passes (5 millions de francs à chaque passe).
- En ne tenant pas compte de la synchronisation, on obtient un état incohérent :
- Exemple : [Cambrioleurs1.java](#)

05/12/03

Les threads

page 21

## Exemple 1/2

- Considérons une banque et plusieurs banquiers qui établissent des virements entre différents comptes.
- En ne tenant pas compte de la synchronisation, on peut obtenir un état incohérent.
- Exemple : [TransfertBank1.java](#)

05/12/03

Les threads

page 18

## Exemple 2/2

- En plaçant les opérations sensibles de la méthode run() de Cambrioleur dans un bloc synchronized (pour l'objet coffre), le problème disparaît.
- Exemple : [Cambrioleurs2.java](#)

05/12/03

Les threads

page 22

## Exemple 2/2

- Dans ce cas précis, il suffit simplement de définir comme étant synchronized la méthode virement de la casse Banque.
- Exemple : [TransfertBank2.java](#)

05/12/03

Les threads

page 19

## Synchronisation wait/notify

- Il est parfois nécessaire d'accéder aux données d'un objet avant de s'apercevoir que l'état de celui-ci ne permet pas de continuer plus avant.
- Il est alors possible d'utiliser la méthode wait afin de :
  - désactiver le verrou
  - attendre la notification d'un autre thread (indiquant que l'état de l'objet a changé).

05/12/03

Les threads

page 23

## Synchronisation « bloc »

- Il est possible de contrôler l'accès aux données d'un objet au niveau d'un simple bloc (y compris à l'extérieur de la classe de l'objet).
- La syntaxe est la suivante :

```
synchronized(objet)
{
    ...
}
```
- L'objet est alors verrouillé jusqu'à la fin du bloc.

05/12/03

Les threads

page 20

## Exemple 1/2

- Considérons un producteur qui produit des articles et un consommateur qui les consomme.
- Chaque nouveau produit doit être consommé (une seule fois).
- En utilisant 2 threads et en ne tenant pas compte de la synchronisation, on obtient (la majorité des cas) une incohérence.
- Exemple : [ProducteurConsommateur1.java](#)

05/12/03

Les threads

page 24

## Exemple 2/2

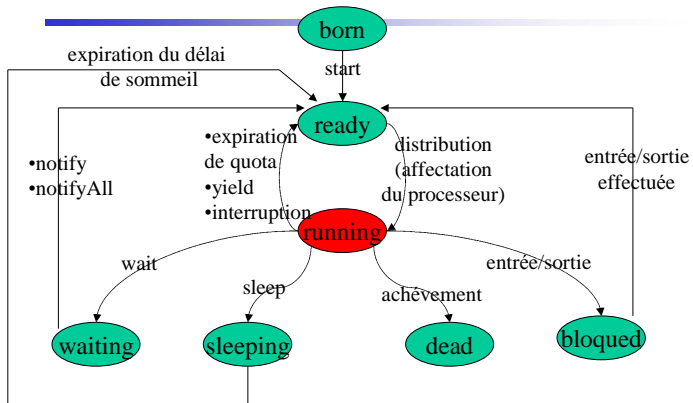
- Il est possible de régler le problème
  - en utilisant un booléen pour savoir qui doit agir (le producteur ou le consommateur)
  - en utilisant la méthode wait quand le personnage (thread) doit attendre son tour
  - en utilisant la méthode notify quand le personnage a fini son tour.
- Exemple : [ProducteurConsommateur2.java](#)

05/12/03

Les threads

page 25

## Cycle de vie d'un thread



05/12/03

Les threads

page 26