

---

# Java Avancé

## Cours 5 : Programmation Parallèle et Asynchrone

---

Arsène Lapostolet & Nada Nahle

# Introduction

---

# Thread

**Thread** : fil d'exécution. Plusieurs en même temps possible.

- Améliorer la vitesse d'un process : attention à l'overhead
- Permettre un cas d'utilisation : ex modèle client-serveur

# Thread Java

- Classe Thread, construit à partir d'un Runnable
- Lancer avec la méthode start()

```

1  var oddThread = new Thread(() -> {
2      IntStream
3          .range(0, 100)
4          .filter(number -> number % 2 != 0)
5          .forEach(System.out::println);
6  });
7
8

```

Java

```
9 var evenThread = new Thread(() -> {
10     IntStream
11         .range(0, 100)
12         .filter(number -> number % 2 == 0)
13         .forEach(System.out::println);
14 });
15
16 evenThread.start();
17 oddThread.start();
18
19 Thread.sleep(5000);
```



Des questions ?

# Synchronisation

---

# Joindre

- Attendre la fin d'un thread

```
1 evenThread.join();  
2 oddThread.join();
```

**Java**

à la place de `Thread.sleep()`



# Ressource partagée et section critique

- Section critique : à exécuter atomiquement sinon problème
- `synchronized` : un seul thread à la fois sur la section
- Collections synchronisées : `Collections.synchronizedList(new ArrayList<>())`
- Attention au coût
- Section critiques synchronisées : **code thread-safe**

# Ressource partagée et section critique

```
1 public class Hotel {
2
3     private final int roomsCount;
4     private int bookedRoomsCount = 0;
5
6     public Hotel(int roomsCount) {
7         this.roomsCount = roomsCount;
8     }
9
10    public int getAvailableRoomCount(){
11        return roomsCount - bookedRoomsCount;
12    }
```

Java

```

13     public void bookRooms(int numberOfBookedRooms){
14         if(getAvailableRoomCount() - numberOfBookedRooms < 0){
15             throw new IllegalArgumentException("Not enough
rooms available");
16         }
17         else {
18             bookedRoomsCount += numberOfBookedRooms;
19         }
20     }
21 }
22 var hotel = new Hotel(20);
23 var reservationThread1 = new Thread(() -> hotel.bookRooms(3));
24 var reservationThread2 = new Thread(() -> hotel.bookRooms(7));
25

```

# Ressource partagée et section critique

```

1 public synchronized void bookRooms(int numberOfBookedRooms) Java
2 {
3     if(getAvailableRoomCount() - numberOfBookedRooms < 0){
4         throw new IllegalArgumentException("Not enough rooms
5         available");
6     }
7     else {
8         bookedRoomsCount += numberOfBookedRooms;
9     }
10 }
```

# Coordonner

- Coordonner des threads entre eux
- `wait` : attendre un signal
- `notify` : envoyer un signal
- Dans un bloc `synchronized`

# Coordonner

```

1  var token = new Object();
2
3  var oddThread = new Thread(() -> {
4      var oddNumbers = ...
5      for (var oddNumber : oddNumbers) {
6          synchronized (token) {
7              token.wait();
8              System.out.println(oddNumber);
9              token.notify();
10         }
11     }
12 });
  
```

Java

```
13 var evenThread = new Thread(() -> {
14     var evenNumbers = ...
15     for (var evenNumber : evenNumbers) {
16         System.out.println(evenNumber);
17         synchronized (token) {
18             token.notify();
19             token.wait();
20         }
21     }
22 });
```



Des questions ?



# Programmation Asynchrone

---

# Définition

- Abstraction au dessus des threads : tâches
- Mutualiser les threads
- Optimiser le temps CPU (I/O non bloquantes)

## Notions :

- Promesse : tâche qui va être réalisée de façon asynchrone, on aura la résolution dans le futur
- Continuation : s'exécute sur le résultat de la promesse

# Notion d'I/O non bloquantes

- CPU-bound : calculs, algos
- I/O-bound : attendre (appel réseau, système de fichiers)

*Attente I/O = Temps CPU gaché*

**Objectif** : le CPU fait autre chose quand il attend l'I/O

# Asynchrone en Java : ThreadPool

Groupe de threads qui vont traiter une série de tâches

```
1 ExecutorService threadPool =  
  Executors.newFixedThreadPool(4);  
2  
3 threadPool.submit(() -> {  
4  
5     ...  
6  
7 } )
```

**Java**

# Promesse en Java : CompletableFuture

```
1 CompletableFuture<String> task = CompletableFuture
2   .supplyAsync(() -> {
3       var result = ... opération longue async...
4       return result;
5   });
6
7 task.thenAccept((String result) -> println(result));
8
9 task.exceptionally(exception -> {
10     exception.printStackTrace();
11     return "";
12 });
```

Java