# Java Avancé

## Modèle de mémoire, publication et Opérations atomiques

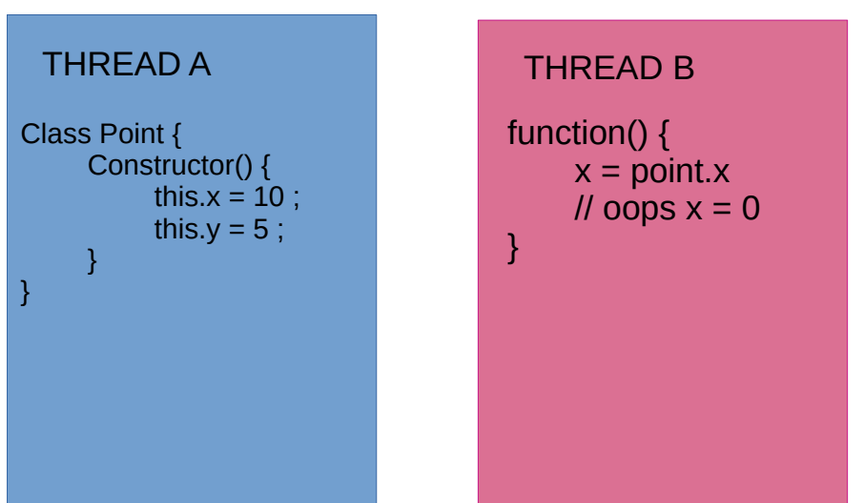
**Guillaume Cau**

# Récapitulatif

- Un champ "volatile" indique à la machine de récupérer la valeur en RAM.
- Les variables "final" n’ont pas de problèmes de publication.
- Il faut faire attention à certaines instructions qui ont  l’air atomique mais qui ne le sont pas.
- Le package java.util.concurrent.atomic contient différents objets permettant d’effectuer des opérations atomiques.

# Les problèmes de publications

- Un problème de publication survient lorsque un thread accède aux champs d'un objet non-initialisé.



# Les variables final

- Pour se protéger des problèmes de publication, nous pouvons utiliser les variables final :

  ```
  private final x = 3
  ```

- Une variable final garantie que le champ est initialisée avant d'y accéder.

- Cependant, une variable final ne peux plus être modifiée. Ce n'est donc pas forcément le comportement souhaité.

# Les VarHandles

- Un VarHandle représente le champ d’une classe, tout comme un MethodHandle représente une méthode d’une classe.
- Les VarHandles peuvent être récupérées depuis un objet lookup.
- Il existe différents types de VarHandles

# Le VarHandle de variable

- Un VarHandle peut représenter une variable de la classe
- Récupération du VarHandle :

  lookup().findVarHandle(<class>, <var_name>, <var_type>)
- Récupération **"threadsafe"** :

  vh.getVolatile(<current_object>)
- Modification **"threadsafe"** :

  vh.compareAndSet(<current_object>, <old_vale>, <new_value>)

# Le VarHandle de tableau

- Un VarHandle peut également représenter les éléments  d'un tableau (Utile pour remplacer volatile)

- Récupération du VarHandle  :

  lookup().arrayElementVarHandle(<array_type>)

- Récupération **"threadsafe"**  :

  (CAST)vh.getVolatile(<array>, <index>)

- Modification **"threadsafe"**  :

  vh.compareAndSet(<array>, <index>, <old_value>, <new_value>)