

# Interfaces Graphiques

« La concurrence est rude mais nécessaire,  
nul ne doit y échapper... »

Proverbe ultra-libéral

Jean-Baptiste.Yunes@u-paris.fr

Université Paris Cité

©2025



- Swing n'est pas thread-safe!!!
- pour en savoir plus, lisez « Swing's Threading Policy » <https://docs.oracle.com/en/java/javase/21/docs/api/java.desktop/javax/swing/package-summary.html>
- In general **Swing is not thread safe.**

### Swing's Threading Policy

In general Swing is not thread safe. All Swing components and related classes, unless otherwise documented, must be accessed on the event dispatching thread.

Typical Swing applications do processing in response to an event generated from a user gesture. For example, clicking on a JButton notifies all ActionListeners added to the JButton. As all events generated from a user gesture are dispatched on the event dispatching thread, most developers are not impacted by the restriction.

Where the impact lies, however, is in constructing and showing a Swing application. Calls to an application's main method, or methods in Applet, are not invoked on the event dispatching thread. As such, care must be taken to transfer control to the event dispatching thread when constructing and showing an application or applet. The preferred way to transfer control and begin working with Swing is to use `invokeLater`. The `invokeLater` method schedules a Runnable to be processed on the event dispatching thread. The following two examples work equally well for transferring control and starting up a Swing application:

- Réactivité
- Concurrency
  - threads *initiaux*
  - thread principal de l'interface (distribution)
  - threads annexes de travail
- Ces threads sont automatiquement créés...

- thread principal (event dispatching thread / UI Thread)
- contrôle la distribution des événements
- **exécute sous son contrôle** le code de traitement associé aux événements (réaction)
- **exécute sous son contrôle** de nombreuses sollicitations externes (commandes)

- interface réactive
  - à réception d'événements le thread principal ne doit pas exécuter de code trop long (en temps)
  - sinon, il faut utiliser des threads de travail

- Threads initiaux
  - Application
    - un seul thread appelant `main ( )`

- dans ces threads initiaux
  - on initialise l'application
- l'initialisation de l'interface doit être réalisée sous le contrôle du thread principal (UI Thread)
  - créer un Runnable
  - demander à l'exécuter par le thread principal
    - `SwingUtilities.invokeLater(Runnable)`
    - `SwingUtilities.invokeAndWait(Runnable)`
      - une simple encapsulation des méthodes correspondantes de `java.awt.EventQueue`

- Threads de travail
  - s'occupent de réaliser des tâches longues
    - sont en arrière-plan
      - une telle tâche peut être représentée par une instance de `SwingWorker`



- mécanismes
  - exécution d'une méthode à la terminaison
  - fournit un résultat asynchrone (un futur)
  - peut exécuter du code sous contrôle du thread principal

- Comment intégrer correctement une longue tâche à une interface graphique ?
- Rappel : les `Listeners` doivent répondre rapidement
  - nécessité de passer par des threads annexes
    - la classe `SwingWorker` fournit tous les services utiles et la bonne abstraction

- `SwingWorker<T, V>`
  - `T` type de la valeur finale produite par le calcul
  - `V` type des valeurs intermédiaires

- Encapsule :
  - un thread annexe de travail
  - des mécanismes de synchronisation vis-à-vis de l'exécution de la tâche
  - des mécanismes de synchronisation avec le thread d'interface

- la création du thread de travail et son exécution est obtenue par
  - `execute()`
    - attention cette méthode ne peut être employée deux fois!
      - un **thread ne s'exécute qu'une seule fois**
- on peut obtenir le résultat du calcul par appel à
  - `T get()`
    - `T` est le type générique de la valeur de retour du `SwingWorker<T, V>`

- le calcul du thread annexe est implanté dans la méthode
  - `T doInBackground()`
    - `T` est le type générique de la valeur de retour du calcul (`SwingWorker<T, V>`)
- lorsque `doInBackground()` termine, la méthode
  - `void done()` est appelée **dans le contexte du thread de l'interface**
    - on y place habituellement de quoi *nettoyer* l'interface

- `doInBackground()` peut avoir besoin d'effectuer mises-à-jour de l'interface, donc sous le contrôle du thread d'interface
- `publish(V... valeurs)` exécutée habituellement dans le thread de calcul
- permet d'envoyer des valeurs de type générique `V` (`SwingWorker<T, V>`) à la méthode
- `process(V... valeurs)` est exécutée normalement dans le thread de l'interface
- permet de récupérer les valeurs envoyées par le thread de calcul





- `ProgressBarExample.java`
- un exemple illustrant l'usage d'un `SwingWorker` pour
  - obtenir la réalisation d'une tâche
  - l'affichage de sa progression dans l'interface