INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE

Java

benjamin.nguyen@insa-cvl.fr

Création d'Interfaces Graphiques

D'après Tutorial SWING (Java 8)

https://docs.oracle.com/javase/tutorial/uiswing/

Cours commun avec Y. Viémont (UVSQ)

But du cours (compétences)

- Savoir coder une application complexe en Java (avec des tests unitaires)
- Savoir débugger une application complexe en Java
- Savoir gérer les fichiers en Java
- Savoir gérer la persistence en Java avec la sérialisation
- Savoir utiliser les threads en Java
- Savoir créer une interface graphique simple en Java (approche modèle MVC)

Si le temps le permet …

- Connaître les bibliothèques cryptographiques en Java
- Comprendre et avoir utilisé le mécanisme d'introspection en Java
- Savoir gérer la persistence en Java avec une base de données

La conception d’une interface graphique

- On obtient très rapidement un code complexe, voir inextricable
- Surtout si on fait de la génération automatique

- **But de ce cours :**
  - faire comprendre les concepts pour pouvoir modéliser une GUI “légère” en surcouche d’une application Java
  - savoir développer une GUI “légère” en Java

Apport de l’utilisation de bibliothèques standard en Java (SWING)

- Gérer de manière unifiée les interactions avec l’utilisateur
- Contrôler la concurrence sur les dessins à l’écran (affichage)
- Contrôler l’accès aux données partagées
- Maîtriser la complexité

La fenêtre “HelloWorld”

Afficher une fenêtre en Java avec le texte “Hello World”

- Voir plusieurs manières de l’implémenter
  - HelloWorldJustFrame (ChatGPT)
  - HelloWorldSwing (Tutoriel Java Swing officiel Oracle)
  - HelloWorldRunner + HelloWorldWindow (ma proposition)

Exemple d’utilisation de CheckBox (et fichiers) … et tout le reste !

- CheckBoxDemo (Oracle)

- Voir nombreux autres exemples sur la page : https://docs.oracle.com/javase/tutorial/uiswing/examples/components/index.html

Approche générale

- Encapsulation des évènements matériels → évènements sémantiques
- Programmation objet
- Programmation évènementielle
- Architecture MVC (Modèle, Vue, Controleur)
- Indépendance du “look and feel”

Evènements

- Matériel (bas niveau)
  - Clic
  - Mouvement de la souris
  - Appui sur une touche
  - Déclanchement de l’horloge
- Sémantique
  - Appui sur un bouton
  - Sélection d’un choix de menu
  - Ouverture / fermeture de fenêtre
  - Timer
  - Changement de propriété

Traduction matériel → sémantique

- {Evenements matériels} + contexte actif → évènement sémantique
- Contexte actif :
  - Fenêtre active
  - Sélection (simple ou multiple)
  - Position de la souris sur un objet visible
- Regroupement d'évènements matériels
  - Alt+Clic
  - Drag & drop
  - Pincer balayer, etc

Programmation objet et programmation évènementielle

OBJET

- Dispose d'une interface partagée
- Maîtrise son propre comportement
- Encapsule ses données privées

EVENEMENTIEL

- Un gestionnaire central d'évènements (dispatcher)
  - reçoit les évènements
  - les ordonnance dans une file
  - les traite séquentiellement ou en parallèle
  - distribue selon les évènements le travail à d'autres tâches
- Les objets
  - produisent des évènements

L’approche MVC (Modèle, Vue, Controleur)

- Modèle
  - Gestion des données applicatives
  - Réalisation des traitements correspondants
- Vue
  - Gestion de l’affichage et présentation des objets
  - Création des évènements (matériel → sémantiques)
- Contrôleur
  - Gestion / distribution des évènements
  - Synchronisation des actions

Exemple : Le bouton “incrémenter”

Diagramme des appels de méthodes

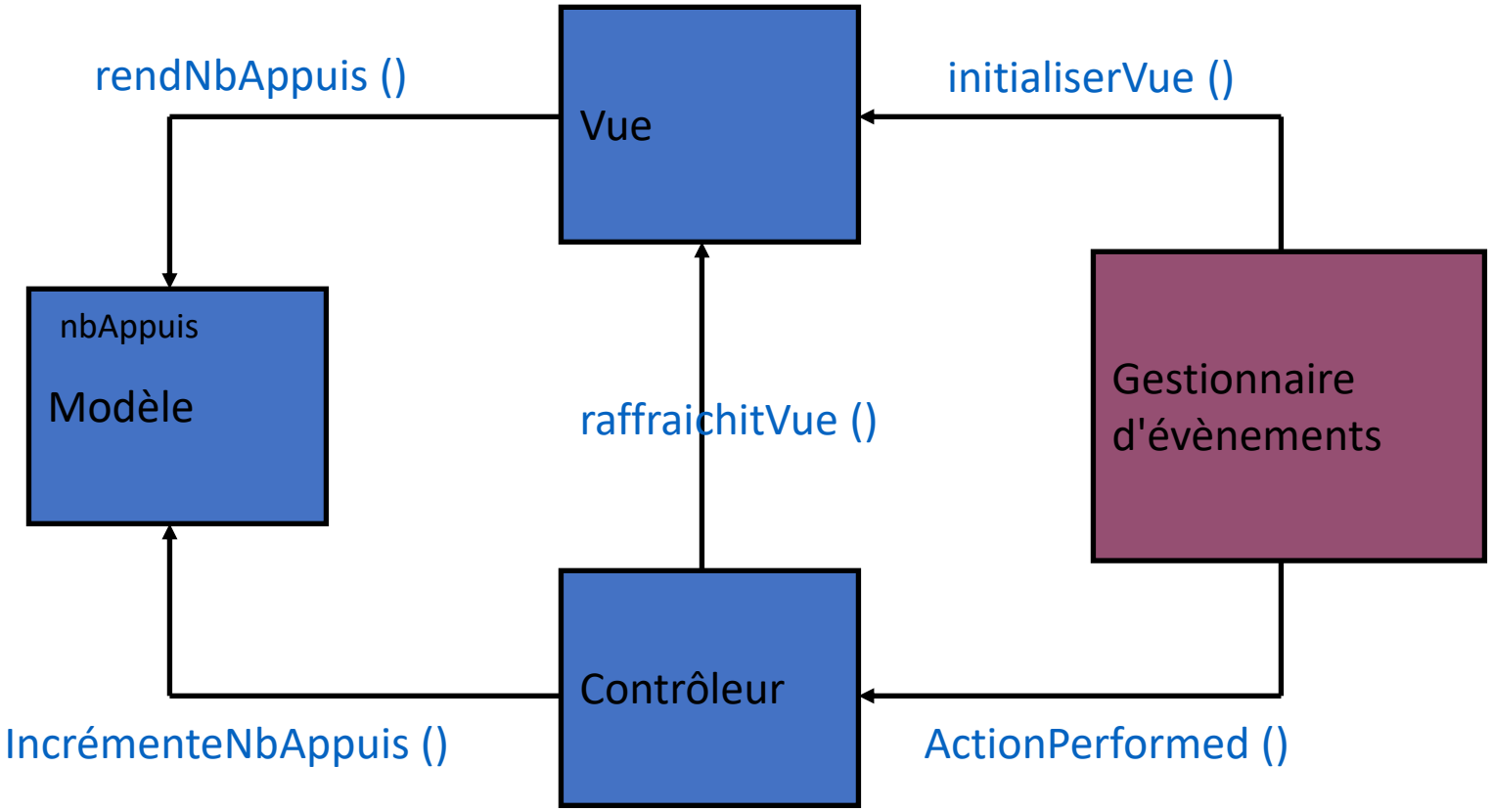
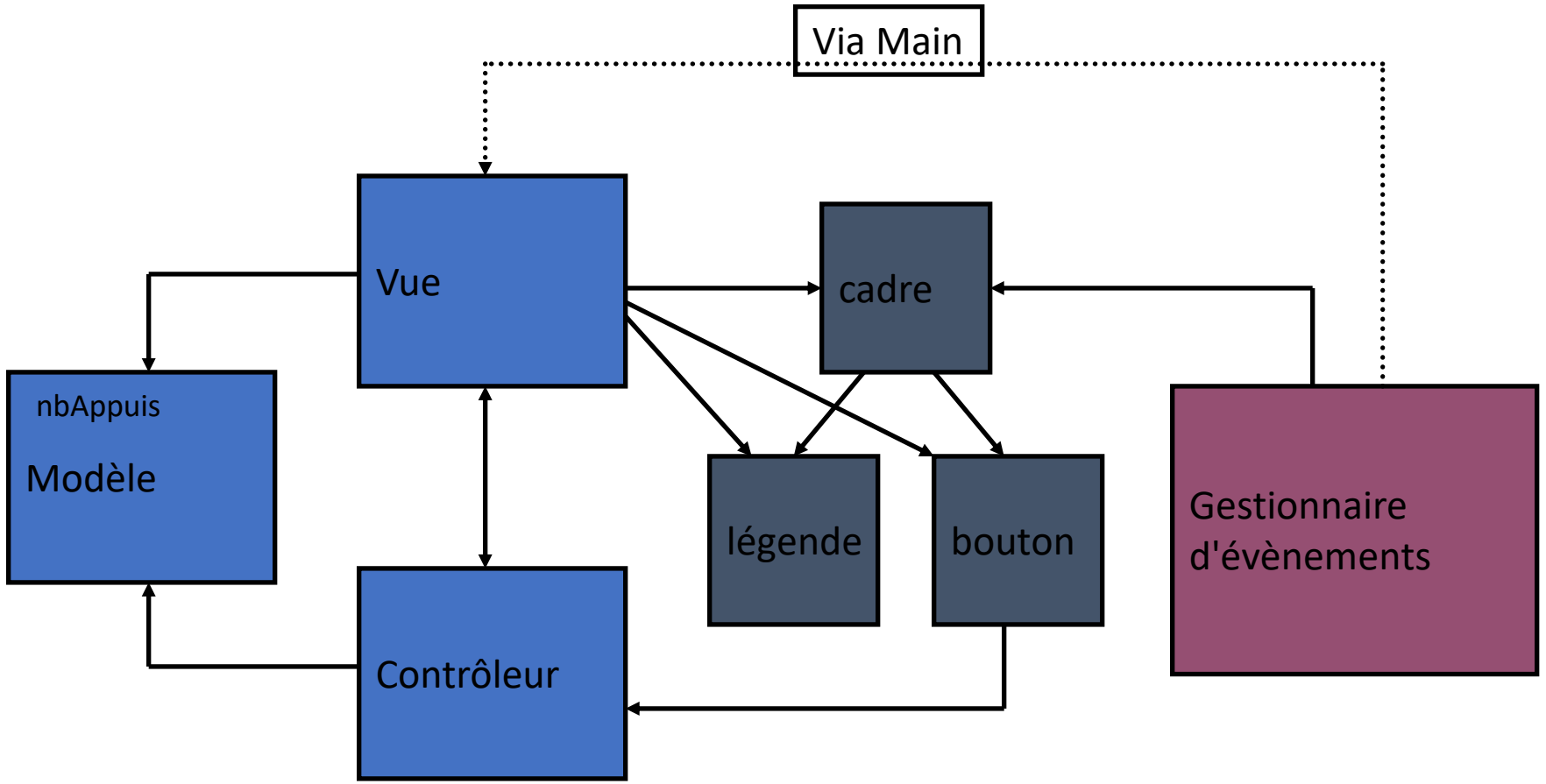


Diagramme des références d’objets



Explication de code

- Modele, Vue, Controleur, et Main
- vs. CodeChatGPT (Controller, CounterModel, CounterView, MVCExample)

Ingédient pour le Controleur : ActionListener

- Un controleur va implémenter l'interface ActionListener
- Nécessite d'implémenter la méthode

public void actionPerformed(ActionEvent e)

- Important : 1 seule méthode actionPerformed par ActionListener !
- Si on veut avoir plusieurs actions différentes :
  - Plusieurs contrôleurs
  - Un seul contrôleur, mais analyse de l'évènement -- par exemple avec la méthode getSource() et/ou getActionCommand()

- Liste de tous les Listeners : (voir listeners / Beeper.java)

https://docs.oracle.com/javase/tutorial/uiswing/examples/events/index.html#Beeper

Exemples de Listeners

- Listener basique : Beeper
- Listener FocusEvent, ComponentEvent, MouseEvent, MouseMoveEvent, …
- Plusieurs Listeners (MultiListener)

Gestion d’évènements multiples

- Voir code ButtonDemo (tuto Oracle)

Les listeners …

Voir : https://docs.oracle.com/javase/tutorial/uiswing/events/api.html

La concurrence en SWING

- Une GUI ne doit JAMAIS “freeze”. Le framework Swing fonctionne de la manière suivante :
  - Threads initiaux qui exécutent le code initial de l’application
  - *Event dispatch* Thread, qui gère tout le code des évènements. La plupart du code du framework Swing s’exécute sur ce thread
  - *Worker* Threads (ou Thread en arrière plan) où les tâche consommatrices de ressources sont utilisées (javax.swing.SwingWorker)
- Important : ces Threads sont créés dans le cadre du framework Swing (d’autres Threads peuvent bien sûr être créés)

- Voir code : Flipper.java (programme qui lance une pièce)

Fonctionnement

- Thread principal : créer un objet Runnable qui initialise la GUI et la plannifie pour l'éxécution sur l'event dispatch Thread.
- **Important :** Pour que les interactions soient détectées, il faut que les composants soient lancés sur l'event dispatch thread c-à-d lancés avec la commande SwingUtilities.invokeLater() ou invokeAndWait()
- Une fois la GUI créée, le programme réagit aux évènements de la GUI, chacun causant l'exécution d'une courte tâche sur l'event dispatch Thread. Les tâches plus lourdes doivent être exécutées sur des Worker Threads.

L’Event Dispatch Thread

- Toute la gestion des évènements en Swing est exécutée sur *un* seul Thread, l’event dispatch Thread, pour éviter les soucis de concurrence (méthodes non Thread-safe). Seules les méthodes spécifiquement indiquées comme Thread Sage peuvent être lancées depuis n’importe quel Thread.
- On peut savoir si on est dans l’event dispatch thread en lançant la méthode SwingUtilities.isEventDispatchThread()

Les Workers

- Si on a une longue tâche à lancer, il faut l'exécuter dans un Worker Thread, et doit être réalisée par une classé héritant de SwingWorker et devant implémenter la méthode doInBackground()

Le SwingWorker utilise des types génériques pour indiquer

Le type de retour

Le type d'entrée

- La méthode doInBackground() est lancée dans un nouveau Thread lorsque le Thread principal appelle execute()
- La méthode done() est lancée dans le Thread principal *une fois la tâche terminée*.
- Voir ExempleSimpleWorker.java

Exemples utiles pour le TD

- ExempleJFileChooser.java
- ExempleJTextArea.java