Interfaces Graphiques

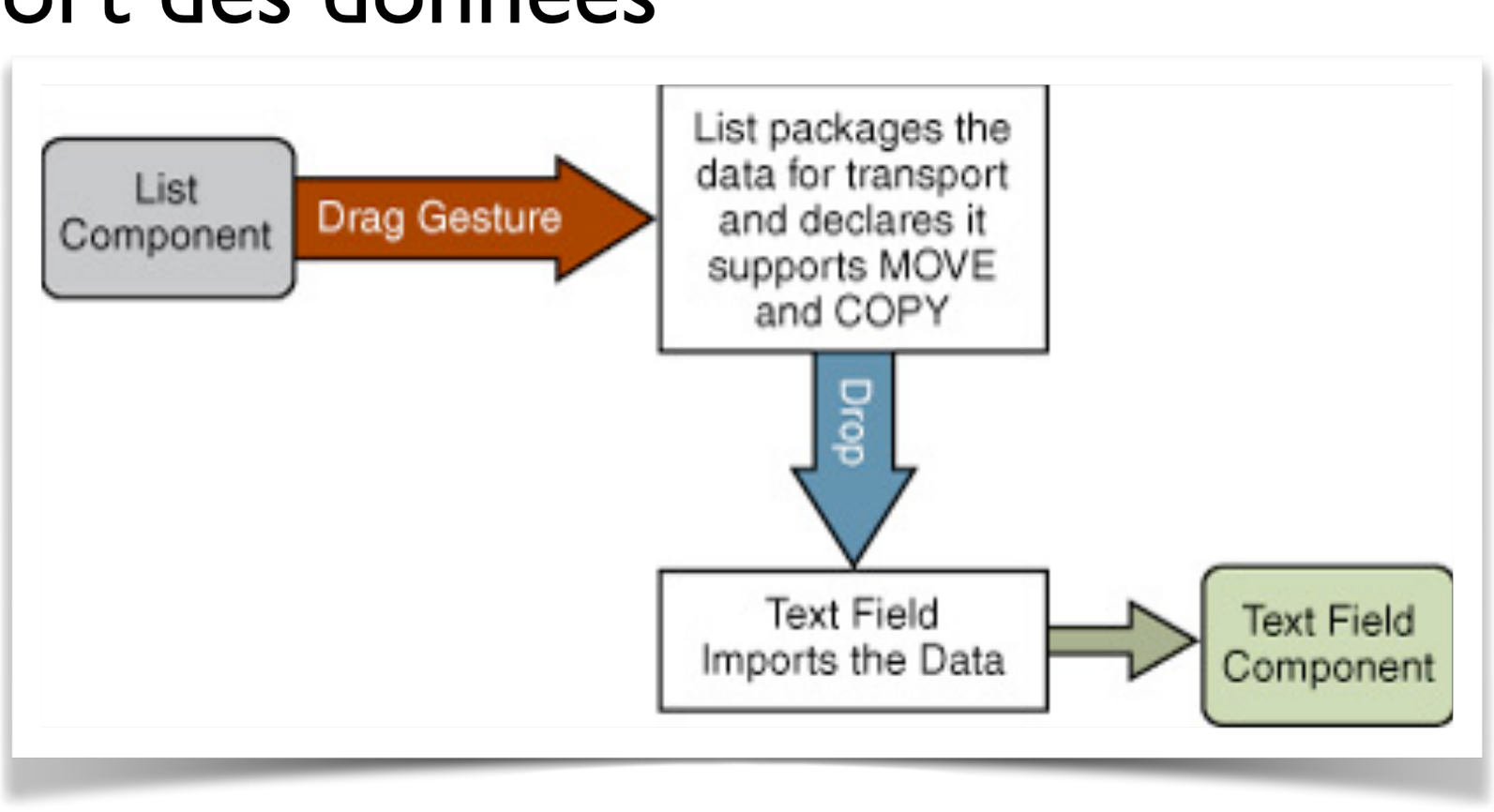
drag'n'drop

Jean-Baptiste.Yunes@univ-paris-diderot.fr
Université Paris Cité

Transfert de données

- deux formes
  - le Glisser-Déposer
    - transfert direct par action à la souris
  - le Presse-Papier
    - transfert indirect par couper/copier - coller

- source
  - ButtonPress + Move (Drag Gesture)
  - préparation de l'export des données
- destination
  - *polling* (de la part du mécanisme de DnD)
  - import des données



repris du Tutoriel Java

- `TransferHandler`
  - une classe encapsulant les primitives de gestion de transfert de données entre une source et une destination dans une opération de Glisser-Déposer
- `JComponent`
  - `setTransferHandler(TransferHandler)`
    - pour associer à **ce** composant un gestionnaire de transfert qui sera utilisé lors d'un DND

- Intéressons-nous au côté **dépôt**, deux choses importantes :
  - recevoir des événements indiquant qu'une opération de transfert est en cours, en mode *polling*, et décider si l'on est prêt à recevoir les données
  - recevoir les données si l'opération est réellement effectuée

- pour recevoir des données il suffit de
  - `boolean canImport(TransferHandler.TransferSupport)`
    - méthode régulièrement appelée pendant un Glisser-Déposer et qui doit renvoyer vrai si à cet instant l'opération Déposer peut être acceptée
  - `boolean importData(TransferHandler.TransferSupport)`
    - méthode appelée lorsque l'opération Déposer est tenté

- la classe `TransferHandler.TransferSupport`
  - encapsule les informations utiles au transfert lui-même, on y trouve:
    - `boolean isDrop()`
      - est-ce une opération de Déposer ?
    - `boolean isDataFlavorSupported(DataFlavor)`
      - est-ce que ces données peuvent-être récupérées selon le format ?

- `Transferable getTransferable()`
  - permet de récupérer l'instance d'un `Transferable` (interface) permettant de récupérer les données selon différents formats
- `Transferable` (voir le détail plus loin, représente les données à échanger)
  - `Object getTransferData(DataFlavor)`

- la class `DataFlavor`
  - décrit une préférence de représentation pour des données `Transferables` durant un Glisser-Déposer
    - essentiellement un type mime associé à une classe Java
- ex. : un fichier est glissé, lors du dépôt on peut vouloir : son contenu, son nom, etc.

- L'exemple `DNDExample.java`
  - permet de Glisser-Déposer la couleur d'un `JColorChooser` sur un `JPanel` et donc d'en changer immédiatement la couleur de fond
  - le `DataFlavor` de la couleur d'un `JColorChooser` est :
    - `DataFlavor.javaJVMLocalObjectMimeType + ";class="+Color.class.getName()`

- Attention le `JColorChooser` comme les `JEditorPane`, `JFileChooser`, `JFormattedTextField`, `JList`, `JTable`, `JTextArea`, `JTextField`, `JTextPane`, `JTree`:
- doit déclarer accepter d'être l'origine d'un DnD : `setDragEnabled(true)`

- Intéressons-nous au **début** d'une telle opération, deux choses importantes:
  - détecter l'action utilisateur démarrant l'opération de Glisser
  - déclarer les données

- pour initier un Glisser-Déposer il faut :
  - détecter un geste utilisateur
  - exporter un Glisser
    - `exportAsDrag(JComponent,InputEvent,int)` pour déclarer le démarrage d'un DND d'un certain type depuis le composant et grâce à l'événement
- ou exporter un copier dans le presse-papier
  - `exportToClipboard(JComponent,ClipBoard,int)`

- pour émettre des données il faut depuis le `TransferHandler`
  - `int getSourceActions(JComponent)`
    - pour déclarer les actions de transfert supportées (combinaison de `COPY`, `MOVE`, `LINK`) depuis le composant donné
  - `Transferable createTransferable(JComponent)`
    - pour déclarer créer l'objet de transfert
  - `void exportDone(JComponent,Transferable,int)`
    - pour libérer les données lorsque l'opération est terminée ou annulée

- `Transferable` une interface avec trois méthodes
  - `Object getTransferData(DataFlavor)`
    - pour obtenir les données selon le format
  - `DataFlavor [] getDataFlavors()`
    - pour obtenir l'ensemble des formats supportés
  - `boolean isDataFlavorSupported(DataFlavor)`
    - pour déterminer si un format particulier est supporté

Un exemple simple

- L'exemple `DND2Example.java`
  - permet de faire Glisser la couleur de fond d'un composant vers un autre capable de l'accepter
  - on exportera une couleur au même format que ceux des `JColorChooser`

- le Glisser-Déposer à grain fin permet de contrôler plus finement les événements qui concernent la destination
  - `void` `java.awt.Component.`**`setDropTarget(DropTarget)`**
- la classe `DropTarget` permet d'établir dynamiquement une relation entre
  - le composant sur lequel le glissé s'effectue
  - les événements du Glisser-Déposer
    - `DropTarget(Component,DropTargetListener)`

- interface `DropTargetListener`
  - `void dropEnter(DropTargetDragEvent)`
  - `void dropExit(DropTargetEvent)`
  - `void dropOver(DropTargetDragEvent)`
  - `void drop(DropTargetDropEvent)`
    - note : `dropComplete(boolean)` pour spécifier que le dépôt s'est réalisé avec succès ou non
  - `void dropActionChanged(DropTargetDragEvent)`

Un exemple à grain fin

- L'exemple `DNDFinExample.java`
- permet de décorer le composant destinataire lorsqu'un Glisser opère par-dessus

Glisser-Déposer à grain fin - DragSource

- le Glisser-Déposer à grain fin permet de contrôler finement les événements qui concernent la source

- la détection d'un mouvement de Glisser-Déposer n'est pas évidente
  - la classe `DragGestureRecognizer`
    - permet d'obtenir la détection d'un mouvement initial de Glisser-Déposer
      - cette détection appelle la méthode `dragGestureRecognized(…)` d'un `DragGestureListener`

- un `DragGestureRecognizer` ne peut-être obtenu qu'à partir d'un `DragSource`
  - la création d'un `DragSource` peut-être obtenue de plusieurs manières
    - `DragSource();`
    - `DragSource().getDefaultDragSource();`
- `createDefaultDragGestureRecognizer( Component, int, DragGestureListener)`

- `void dragGestureRecognized(DragGestureEvent)`
  - appelée lorsqu'un geste de début de Glisser est détecté
  - le `DragGestureEvent` peut alors décider d'amorcer le Glisser-Déposer
- `startDrag(…)`
  - qui permet de paramétrer l'opération
    - `Cursor`, `Image`, `Point`, `Transferable`, `DragSourceListener`

- On peut vouloir obtenir un *scrolling* automatisé durant une opération de dépôt
  - il suffit que le composant implémente l'interface `Autoscroll`
    - `void autoscroll(Point);`
      - appelé durant l'*autoscrolling*
    - `Insets getAutoscrollInsets();`
      - appelée pour déterminer les zones d'*autoscrolling*

Un exemple complet à grain fin

- L'exemple `DNDASExample.java`
- illustre l'ensemble des mécanismes de Glisser-Déposer à grain fin