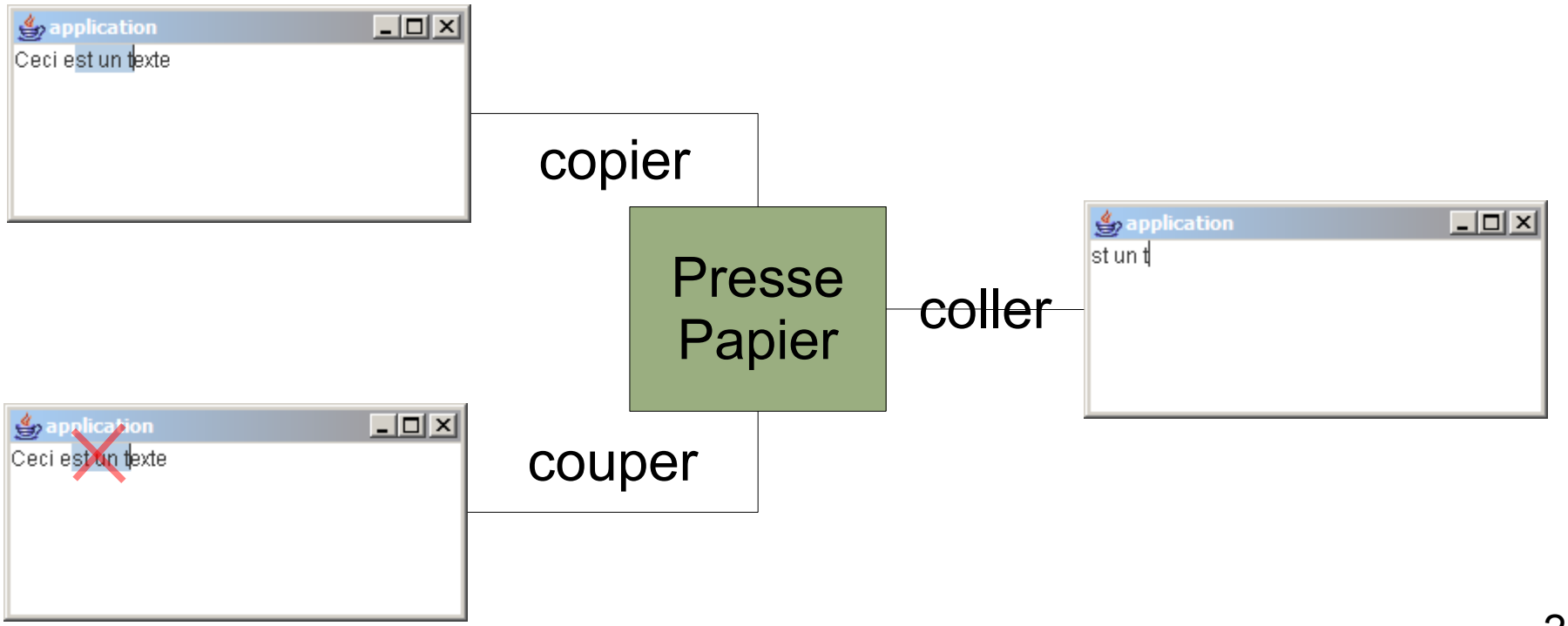
Copier/Coller & Drag'n Drop

Rémi Forax

Le principe du copier/coller

- Le copier/coller permet de transférer du texte (ou n'importe quoi d'autre) d'une application vers une autre en transitant par le presse papier



Le presse papier

- Il existe plusieurs presse-papiers :
  - Le presse papier système

```
Toolkit toolkit=Toolkit.getDefaultToolkit();
toolkit.getSystemClipboard()
```

  - Le presse papier de sélection

```
Toolkit toolkit=Toolkit.getDefaultToolkit();
toolkit.getSelectionClipboard()
```

  - Des presses papiers locaux

```
new Clipboard(String name)
```

- Le package correspondant est **java.awt.datatransfert**

Le presse papier (2)

- Le presse papier contient un même objet mais sous plusieurs représentations possibles (**DataFlavor**).

Texte →

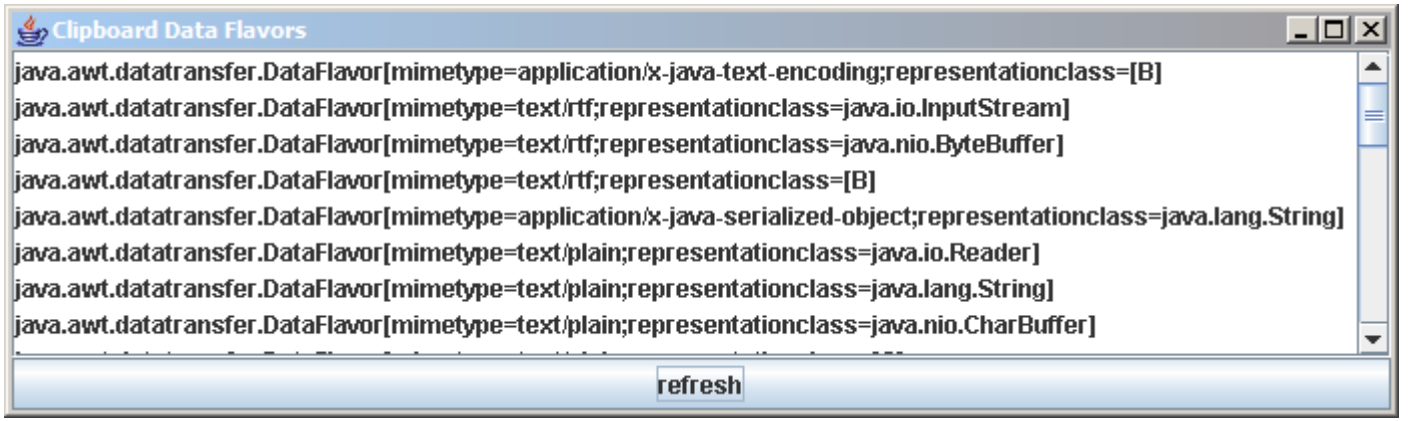
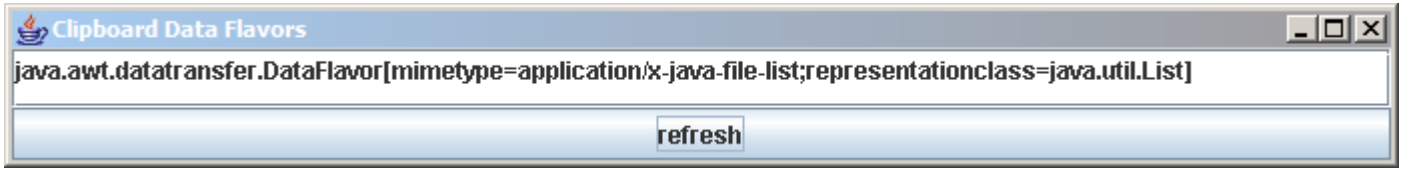


Image →

Fichiers →



Le presse-papier (3)

- L'interface du presse papier
  - Les types de données
    - boolean isDataFlavorAvailable(DataFlavor flavor)
    - DataFlavor[] getAvailableDataFlavors()
  - La gestion du contenu
    - Object getData(DataFlavor flavor)
    - Transferable getContents(Object requestor)
    - setContents(Transferable contents, ClipboardOwner owner)
- Attention la méthode getContent() peut consommer beaucoup de mémoire !

DataFlavor

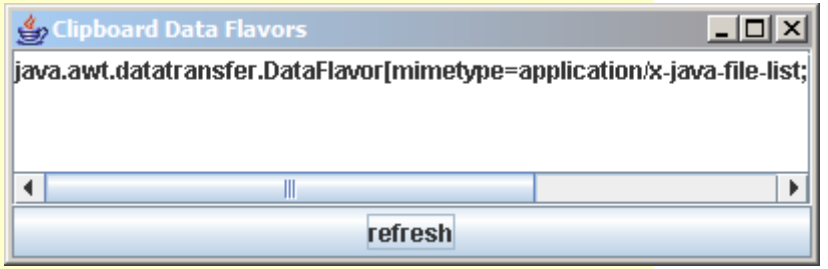
- Chaine de caractère representant un type de donnée dans le presse-papier
- Contient :
  - Le type MIME (text/plain, image/gif etc.)
  - La classe Java correspondante
  - L'encoding pour les caractères
- Forme : mimeType=...;representationClass=...;charset=...;

DataFlavor (2)

- Constantes :
  - **Texte** : DataFlavor.stringFlavor
  - **Image** : DataFlavor.imageFlavor
  - **Fichiers** : DataFlavor.javaFileListFlavor
- Constructeurs
  - **DataFlavor**(Class<?> clazz, String text)
    passe les objet sous forme sérializé
  - **DataFlavor**(String mime,String text)
    La chaine mime doit être encodé en utilisant :
    - Un type MIME classique (text/plain, image/jpg, etc.)
    - DataFlavor.javaJVMLocalObjectMimeType
    - DataFlavor.javaSerializedObjectMimeType

DataFlavor et Presse-papier

```
public static void setListModel(JList list,Clipboard clipboard) {
  final DataFlavor[] flavors;
  try {
    flavors=clipboard.getAvailableDataFlavors();
  } catch (IllegalStateException e) {
    return;
  }
  list.setModel(new AbstractListModel() {
    public int getSize() {
      return flavors.length;
    }
    public DataFlavor getElementAt(int index) {
      return flavors[index];
    }
  });
}
public static void main(String[] args) {
  Toolkit toolkit=Toolkit.getDefaultToolkit();
  final Clipboard clipboard=toolkit.getSystemClipboard();
  final JList list=new JList();
  JButton button=new JButton("refresh");
  button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      setListModel(list,clipboard);
    }
  });
```



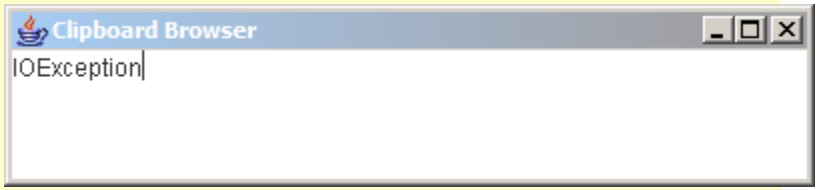
Verrou et évènement

- Sous Windows, lorsqu'une application accède au presse papier, aucune autre ne peut y accéder
- Les méthodes d'accès en Java renvoie alors l'exception **IllegalStateException**
- Il est possible d'être averti des actions effectués sur le presse papier
  - addFlavorListener(FlavorListener listener)
  - removeFlavorListener(FlavorListener listener)
  - FlavorListener[] getFlavorListeners()

Exemple

- Obtenir le contenu sous forme de chaîne de caractère

```
private static String getStringFromClipboard(Clipboard clipboard) {
  boolean available;
  try {
    available=clipboard.isDataFlavorAvailable(DataFlavor.stringFlavor);
  } catch (IllegalStateException e) {
    return null;
  }
  if (!available)
    return null;
  try {
    return (String)clipboard.getData(DataFlavor.stringFlavor);
  } catch (UnsupportedFlavorException e) {
    throw new AssertionError(e); // pas censé arriver ici
  } catch (IOException e) {
    throw new AssertionError(e); // pareil
  }
}
```



Exemple (suite)

- On s'enregistre en tant que listener

```
static void refresh(Clipboard clipboard, JTextArea area) {
  String text=getStringFromClipboard(clipboard);
  if (text!=null)
    area.setText(text);
}
public static void main(String[] args) {
  final JTextArea area=new JTextArea();
  Toolkit toolkit=Toolkit.getDefaultToolkit();
  final Clipboard clipboard=toolkit.getSystemClipboard();
  clipboard.addFlavorListener(new FlavorListener() {
    public void flavorsChanged(FlavorEvent event) {
      refresh(clipboard,area);
    }
  });
  refresh(clipboard,area);

  JFrame frame=new JFrame("Clipboard Browser");
  frame.setContentPane(area);
  frame.setSize(400,300);
  frame.setVisible(true);
}
```

Ecrire du texte dans le presse-papier

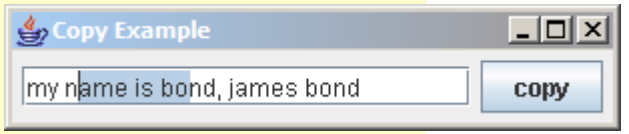
- Doit utiliser la méthode :
  - setContents(Transferable contents, ClipboardOwner owner)
- L'interface Transferable décrit l'objet qui sera inséré dans le presse-papier :
  - boolean isDataFlavorSupported(DataFlavor flavor)
  - DataFlavor[] getTransferDataFlavors()
  - Object getTransferData(DataFlavor flavor)
- StringSelection implante Transferable pour les String

Exemple d'action

- Implanter l'action « copy »

```
Toolkit toolkit=Toolkit.getDefaultToolkit();
final Clipboard clipboard=toolkit.getSystemClipboard();

final JTextField field=new JTextField(20);
class CopyAction extends AbstractAction implements CaretListener {
  public CopyAction() {
    super("copy");
  }
  public void caretUpdate(CaretEvent e) {
    setEnabled(field.getSelectedText()!=null);
  }
  public void actionPerformed(ActionEvent e) {
    String text=field.getSelectedText();
    clipboard.setContents(new StringSelection(text),null);
  }
}
CopyAction copy=new CopyAction();
field.addCaretListener(copy);
JButton button=new JButton(copy);
JPanel panel=new JPanel();
panel.add(field);
panel.add(button);
```



Ecrire un Transferable d'image

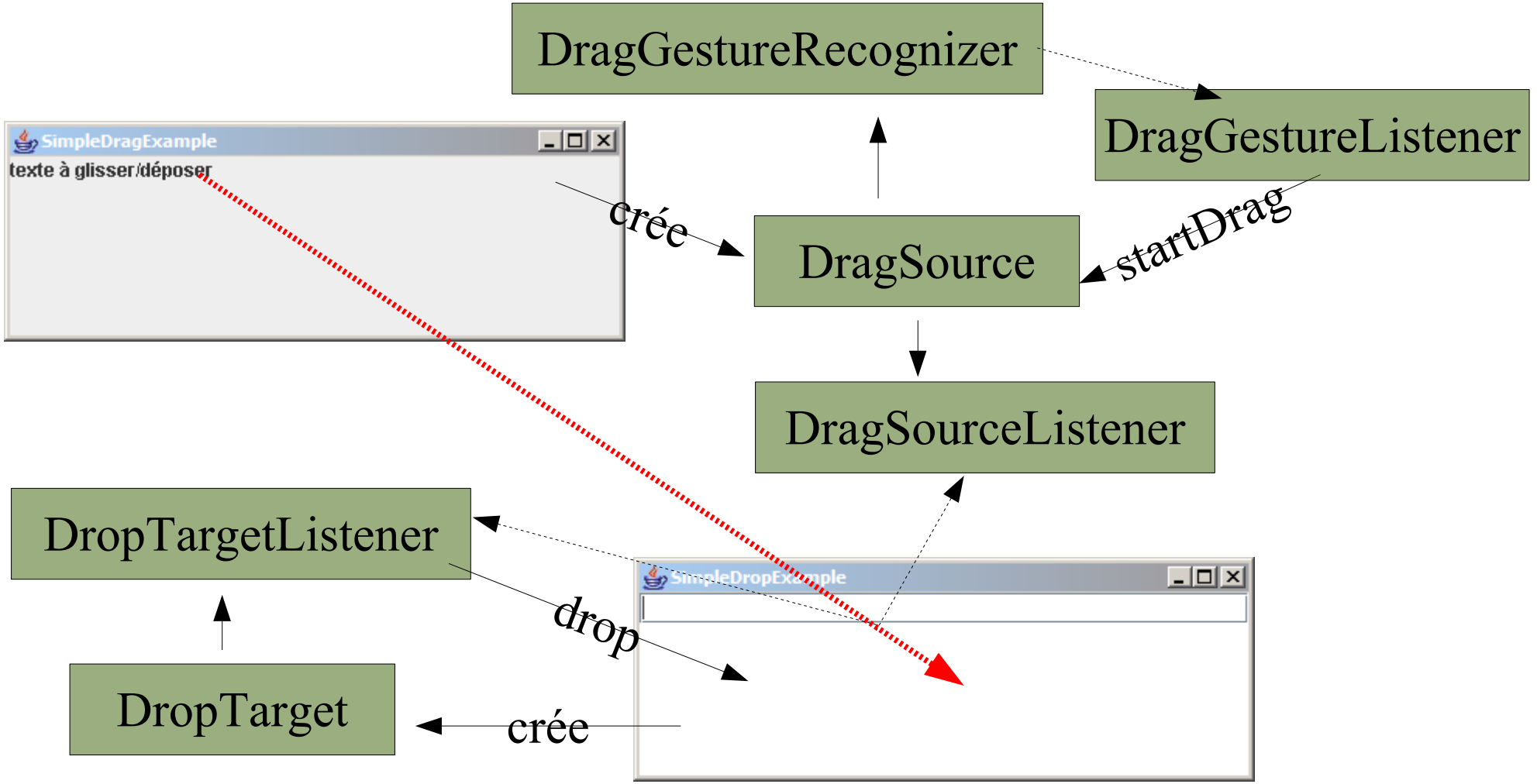
- Ecrire un Transferable pour les images

```
public class ImageSelection implements Transferable  {
  public ImageSelection(Image image) {
    this.image = image;
  }
  public DataFlavor[] getTransferDataFlavors() {
    return SUPPORTED_FLAVORS;
  }
  public boolean isDataFlavorSupported(DataFlavor flavor) {
    return DataFlavor.imageFlavor.equals(flavor);
  }
  public Object getTransferData(DataFlavor flavor)
    throws UnsupportedFlavorException,IOException {

    if (!isDataFlavorSupported(flavor)) {
      throw new UnsupportedFlavorException(flavor);
    }
    return image;
  }
  private final Image image;
  private static final DataFlavor[] SUPPORTED_FLAVORS=
    new DataFlavor[] {DataFlavor.imageFlavor};
}
```

Le Drag & Drop

- Schéma générale du Dnd avec swing



DragSource

- DragSource est l'objet permettant de démarrer le DnD, il existe deux façons d'obtenir cet objet
- En demandant le DragSource de la plateforme

```
DragSource.getDefaultDargSource()
```

- En créant son DragSource locale

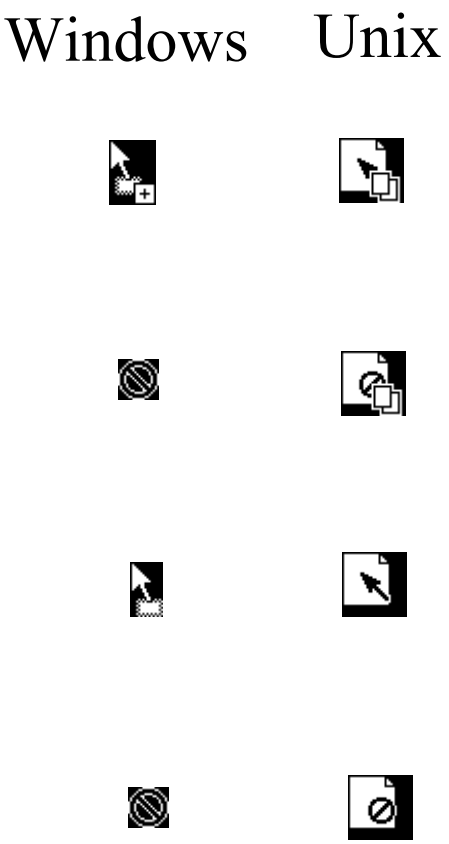
```
new DragSource()
```

DragSource (2)

- Initier le Dnd se fait par l'intermédiaire de la méthode startDrag()
  - startDrag(DragGestureEvent trigger, Cursor dragCursor, Transferable transferable, DragSourceListener dsl)
    - **DragGestureEvent** correspond à l'évènement utilisateur (à la souris) ayant déclenché le Dnd
    - **Cursor** est le curseur indiquant le type d'action possible pour le Dnd
    - **Transferable** est l'objet contenant les informations à passer lors du Dnd
    - **DragSourceListener** permet d'être averti des actions de l'utilisateur sur le zone de drop

Les curseurs du Dnd

- DragSource définie des constantes pour les curseurs du Dnd :
  - DefaultCopyDrop
    - Action copy, drop accepté
  - DefaultCopyNoDrop
    - Action copy, drop pas accepté
  - DefaultModeDrop
    - Action move, drop accepté
  - DefaultMoveNoDrop
    - Action move, drop pas accepté



DragGestureRecognizer

- La classe DragGestureRecognizer permet d'indiquer quels sont les actions a effectuer pour démarer un Dnd
  - dragSource.createDefaultDragGestureRecogniz er(Component component, int actions, DragGestureListener listener)
    - Component le composant source du Dnd
    - Actions parmis : **DndConstants.ACTION_COPY, DndConstants.ACTION_COPY_OR_MOVE, DndConstants.ACTION_MOVE**
    - DragGestureListener : permet d'être averti que l'utilisateur demande un Dnd
  - Il est possible de créer son propre recognizer

DragGestureListener

- Obtient un événement indiquant une demande par l'utilisateur d'un Dnd
  - dragGestureRecognized(DragGestureEvent dge)
- DragGestureEvent
  - int getDragAction(), action demandé par l'utilisateur
  - startDrag(Cursor dragCursor, Transferable transferable, DragSourceListener dsl) équivalent au startDrag() sur le DragSource

DragSourceListener

- Appelé lors du passage sur une zone de drop
  - dragEnter(DragSourceDragEvent dsde)
    - Appelé lors de l'entrée dans la zone de drop
  - dragOver(DragSourceDragEvent dsde)
    - Appelé sur la zone de drop
  - dropActionChanged(DragSourceDragEvent dsde)
    - Appelé lors d'un changement d'action
  - dragExit(DragSourceEvent des)
    - Appelé lors d'une sortie de zone
  - dragDropEnd(DragSourceDropEvent dsde)
    - Appelé à la fin d'un Dnd

Exemple utilisant DragSource

- Exemple avec un label

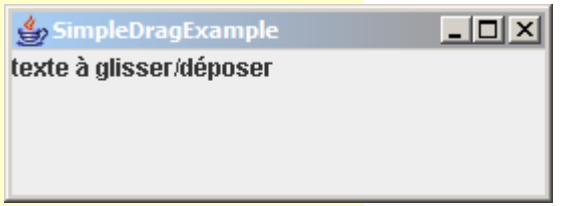
```
final JLabel label=new JLabel("texte à glisser/déposer");

DragSource dragSource=DragSource.getDefaultDragSource();

final DragSourceListener dragListener=new DragSourceListener() {
  ... // le code du dragListener
};

DragGestureListener gestureListener=new DragGestureListener() {
  public void dragGestureRecognized(DragGestureEvent dge) {
    Transferable transferable = new StringSelection(
      label.getText());
    dge.startDrag(DragSource.DefaultCopyNoDrop,
                  transferable, dragListener);
  }
};

dragSource.createDefaultDragGestureRecognizer(
  label,DnDConstants.ACTION_COPY,gestureListener);
```



Exemple suite (DragSourceListener)

```
public void dragEnter(DragSourceDragEvent dsde) { doDrag(dsde); }
public void dragOver(DragSourceDragEvent dsde) { doDrag(dsde); }
public void dropActionChanged(DragSourceDragEvent dsde) {
  doDrag(dsde);
}
public void dragExit(DragSourceEvent dse) {
  dse.getDragSourceContext().setCursor(DragSource.DefaultCopyNoDrop);
}
private void doSourceDrag(DragSourceDragEvent dsde) {
  DragSourceContext context = dsde.getDragSourceContext();
  //System.out.println(dsde.getUserAction());
  //System.out.println(dsde.getTargetActions());
  //System.out.println(dsde.getDropAction());
  if ((dsde.getDropAction() & DnDConstants.ACTION_COPY)!=0) {
    context.setCursor(DragSource.DefaultCopyDrop);
  } else {
    context.setCursor(DragSource.DefaultCopyNoDrop);
  }
}
public void dragDropEnd(DragSourceDropEvent dsde) {
  if (!dsde.getDropSuccess())
    return;
  if (dsde.getDropAction()==DnDConstants.ACTION_MOVE) {
    // faire qqchose dans le cas d'un move
  }
}
```

DropTarget

- DropTarget est un objet associé à un composant indiquant que celui-ci peut recevoir un DnD
  - DropTarget(Component component, int actions, DropTargetListener listener)
    - Component, composant recevant le Dnd
    - Actions traités par le composant
    - DropTargetListener indique comment réagir selon les événements du Dnd
  - Opérations :
    - is/setActive() indique que l'on peut recevoir un Dnd

DropTargetListener

- Appelé lors su passage sur la zone de drop
  - dragEnter(DropTargetDragEvent dtde)
    - Appelé lors de l'entrée dans la zone de drop
  - dragOver(DropTargetDragEvent dtde)
    - Appelé sur la zone de drop
  - dropActionChanged(DropTargetDragEvent dtde)
    - Appelé lors d'un changement d'action
  - dragExit(DropTargetEvent dte)
    - Appelé lors d'une sortie de zone
  - drop(DropTargetDropEvent dtde)
    - Appelé lors d'un drop

Exemple de DropTarget

```
final JTextArea area=new JTextArea();

DropTargetListener dropListener=new DropTargetListener() {
  public void dragEnter(DropTargetDragEvent dtde) { doDrag(dtde); }
  public void dragOver(DropTargetDragEvent dtde) { doDrag(dtde); }
  public void dropActionChanged(DropTargetDragEvent dtde) {
    doDrag(dtde);
  }
  public void dragExit(DropTargetEvent dte) {
  }
  private void doTargetDrag(DropTargetDragEvent dtde) {
    if (dtde.isDataFlavorSupported(DataFlavor.stringFlavor)) {
      int position=area.viewToModel(dtde.getLocation());
      area.setCaretPosition(position);
      area.getCaret().setVisible(true);
      dtde.acceptDrag(dtde.getDropAction());
    }
    else
      dtde.rejectDrag();
  }
  // manque le drop
};

area.setDropTarget(
  new DropTarget(area,DConstants.ACTION_COPY_OR_MOVE, dropListener));
```

Exemple (suite)

- La méthode drop

```
public void drop(DropTargetDropEvent dtde) {
  dtde.acceptDrop(dtde.getDropAction());
  String text;
  try {
    text=(String)dtde.getTransferable().
      getTransferData(DataFlavor.stringFlavor);
  } catch (UnsupportedFlavorException e) {
    dtde.dropComplete(false);
    return;
  } catch (IOException e) {
    dtde.dropComplete(false);
    return;
  }
  area.insert(text,area.getCaretPosition());
  area.getCaret().setVisible(false);
  dtde.dropComplete(true);
}
```

Dnd et Java 1.4

- La version 1.4 de Java possède des méthodes permettant de simplifier le Copier/Coller et le Dnd
- **Component.setDragEnabled**()
  le composant permet alors le Drag
- **Component.setTransfertHandler**(handler)
  le composant permet le Drop et peut implanter cut/copy/paste

Dnd et Composants de Swing

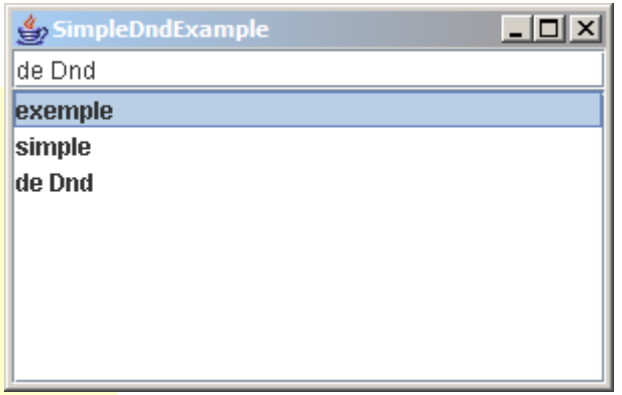
- Composants de Swing implante par défaut :
  - JList, JTree, JTable :
    - Drag Copy
    - Copy (vers presse papier)
  - JTextField, JTextArea, JTextPane, JEditorPane :
    - Drag Copy/Move
    - Drop
    - Cut/Copy/Paste
- Pour que le Drag marche il faut : setDragEnabled(true)

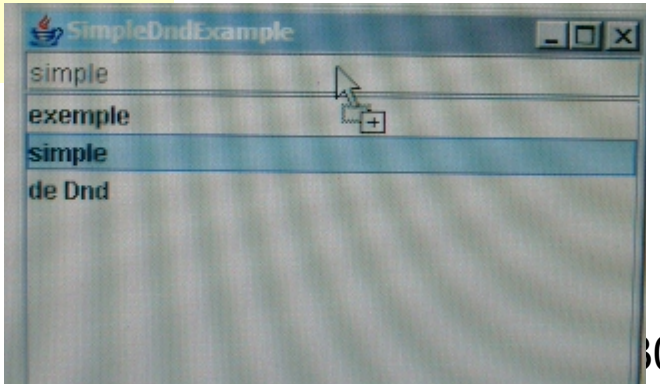
Exemple

- En utilisant setDragEnabled(true)

```
public static JList createList(String... args) {
  JList list=new JList(args);
  list.setDragEnabled(true);
  return list;
}
public static void main(String[] args) {
  JTextField field=new JTextField();
  JList list=createList("exemple","simple","de Dnd");

  JFrame frame=new JFrame("SimpleDndExample");
  frame.getContentPane().add(field,BorderLayout.NORTH);
  frame.getContentPane().add(new JScrollPane(list));
  frame.pack();
  frame.setVisible(true);
}
```





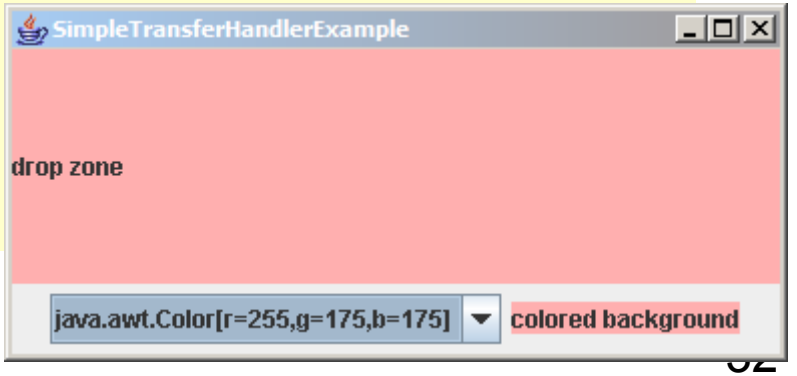
TransferHandler

- Classe qui permet a des composants swing d'échanger des transferables soit par copier/coller soit par Dnd
- Implantation par défaut de TransfertHandler permet de transférer la valeur d'une propriété (bean) d'un composant à un autre
- Un Transferable possède des méthodes :
  - D'export d'objet
  - D'import d'objet

Exemple de TransfertHandler

- En utilisant le transfertHandler par défaut :

```
final JLabel back=new JLabel("colored background");
back.setOpaque(true);
final TransferHandler handler=new TransferHandler("background");
back.setTransferHandler(handler);
back.getDropTarget().setActive(false);
back.addMouseListener(new MouseAdapter() {
  public void mousePressed(MouseEvent event) {
    handler.exportAsDrag(back,event,TransferHandler.COPY);
  }
});
final JComboBox colorBox=new JComboBox(colors);
colorBox.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    Color color=(Color)colorBox.getSelectedItem();
    back.setBackground(color);
  }
});
JLabel label=new JLabel("drop zone");
label.setOpaque(true);
label.setTransferHandler(handler);
```



Export/TransferHandler

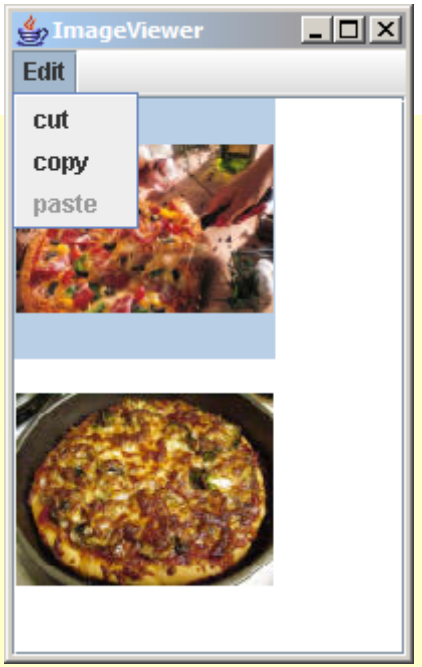
- Méthode d'export (Drag ou cut/copy)
  - protected Transferable createTransferable(JComponent c)
    - Créé un transferable représentant la donnée à transférer
  - int getSourceActions(JComponent c)
    - Indique quelles sont les actions possible (Copy/Move, les 2)
  - void exportAsDrag(JComponent c, InputEvent e, int action)
    - Appelé lors de l'initiation d'un Drap sur le composant
  - void exportToClipboard(JComponent c, Clipboard clip, int action)
    - Appelé lors d'un cut/copy sur un composant
  - protected void exportDone(JComponent source, Transferable data, int action)
    - Appelé une fois l'export effectué (drop ou paste)

Import/TransferHandler

- Méthode d'import (Drop ou paste)
  - boolean canImport(JComponent c, DataFlavor[] transferFlavors)
    - Un drop ou un paste est-il possible ?
  - boolean importData(JComponent comp, Transferable data)
    - Drop ou paste

Un TransferHandler d'image

- Ecrire un TransferHandler pour une liste d'images




```
class ImageTransferHandler extends TransferHandler {
  public ImageTransferHandler(ImageListModel model) {
    this.model=model;
  }
  public int getSourceActions(JComponent c) {
    return COPY_OR_MOVE;
  }
  protected Transferable createTransferable(JComponent c) {
    JList list=(JList)c;
    int index=list.getSelectedIndex();
    if (index==-1)
      return null;
    return new ImageSelection(model.getElementAt(index));
  }
  protected void exportDone(JComponent source, Transferable data, int action) {
    if (action==MOVE)
      model.remove(((ImageSelection)data).getImage());
  }
  ...
```

Un TransferHandler d'image (2)

- Import d'une image

```
  public boolean canImport(JComponent comp, DataFlavor[] flavors) {
    return Arrays.asList(flavors).contains(DataFlavor.imageFlavor);
  }
  public boolean importData(JComponent c, Transferable t) {
    if (t.isDataFlavorSupported(DataFlavor.imageFlavor)) {
      Image image;
      try {
        image = (Image)t.getTransferData(DataFlavor.imageFlavor);
      } catch (UnsupportedFlavorException e) {
        throw new AssertionError(e);
      } catch (IOException e) {
        return false;
      }
      JList list=(JList)c;
      int index=list.getSelectedIndex();
      model.add(index+1,image);
      return true;
    }
    return false;
  }
  private final ImageListModel model;
}
```

Les actions Cut & Copy

```
static void exportToClipboard(TransferHandler handler,JList list,
  Clipboard clip, int action) {
  try {
    handler.exportToClipboard(list,clip,action);
  } catch(IllegalStateException e) { // do nothing }
}
public static void main(String[] args) {
  ...
  final Clipboard clipboard=Toolkit.getDefaultToolkit().getSystemClipboard();
  final Action cut=new AbstractAction("cut") {
    public void actionPerformed(ActionEvent e) {
      exportToClipboard(transferHandler,list,clipboard,TransferHandler.MOVE);
    } };
  cut.setEnabled(false);
  final Action copy=new AbstractAction("copy") {
    public void actionPerformed(ActionEvent event) {
      exportToClipboard(transferHandler,list,clipboard,TransferHandler.COPY);
    } };
  copy.setEnabled(false);
  list.getSelectionModel().addListSelectionListener(
    new ListSelectionListener() {
    public void valueChanged(ListSelectionEvent e) {
      boolean enabled=list.getSelectedIndex()!=-1;
      cut.setEnabled(enabled);
      copy.setEnabled(enabled);
    } });
```

L'action Paste

```
final Action paste=new AbstractAction("paste") {
  public void actionPerformed(ActionEvent e) {
    transferHandler.importData(list,clipboard.getContents(this));
  }
};
paste.setEnabled(false);
clipboard.addFlavorListener(new FlavorListener() {
  public void flavorsChanged(FlavorEvent event) {
    try {
      paste.setEnabled(transferHandler.
        canImport(list,clipboard.getAvailableDataFlavors()));
    } catch(IllegalStateException e) {
      paste.setEnabled(false);
    }
  }
});
```