



Java pour le développement de clients lourds

« Drag and Drop » et transfert de données

Mickaël BARON - 2007 (Rév. Août 2009)
mailto:baron.mickael@gmail.com ou mailto:baron@ensma.fr

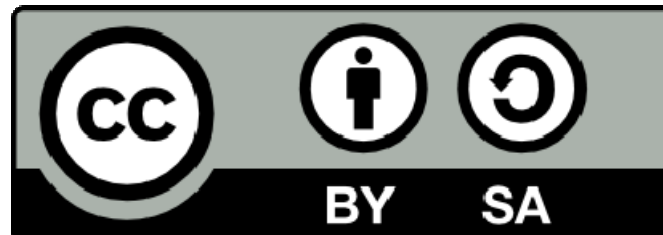
Licence

Creative Commons

Contrat Paternité

Partage des Conditions Initiales à l'Identique

2.0 France



<http://creativecommons.org/licenses/by-sa/2.0/fr>

Mise en œuvre du « Drag and Drop »

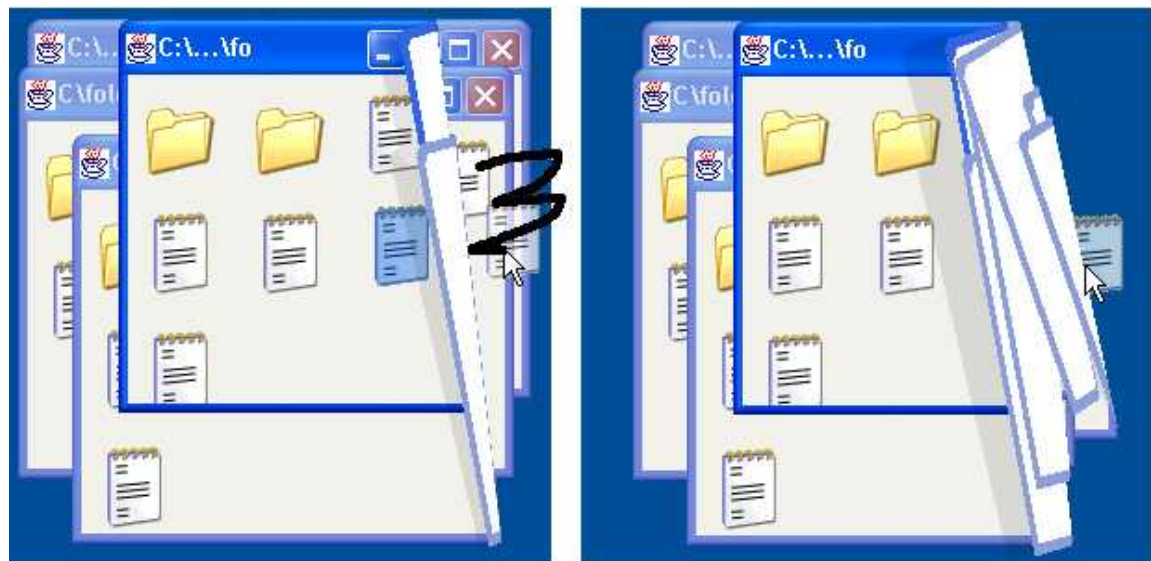
- La technique du « Drag and Drop » ou « Déposé et Collé » permet d'effectuer un transfert de données par manipulation directe
- Possibilité d'effectuer un « Drag and Drop » interne à l'application et externe (entrée ou sortie des données de ou vers l'application)
- Caractéristiques du « Drag and Drop »
 - Un composant source
 - Un composant destination
 - Un objet de transfert du composant source à destination
 - Type de « Drag and Drop » : déplacement ou copie de l'objet
 - Une image qui affiche l'état du « Drag and Drop »
 - Information pour différencier une copie et un déplacement
 - Fantôme pour donner un aperçu

Mise en œuvre du « Drag and Drop » avec Java

- Java a introduit la métaphore du « Drag and Drop » à partir de la version 1.2 : ~ 10 classes du package *java.awt.dnd*. *
 - *DragGestureRecognizer*, *DragGestureListener*, *DragSourceContext*, ...
- Une nouvelle API beaucoup plus légère et plus simple a été proposée à partir de la version 1.4
 - *TransferHandler*
 - *Transferable* (commun avec l'API précédente)
- Le but de cette partie est de présenter l'API pour le « Drag and Drop » de manière à pouvoir
 - Effectuer des « Drag and Drop » sur les composants basiques et complexes
 - Comprendre le mécanisme de transfert de données

Mise en œuvre du « Drag and Drop » avec Java

- Ressources pour construire cette partie
 - SUN : <http://java.sun.com/docs/books/tutorial/uiswing/misc/dnd.html>
- Informations complémentaires concernant des effets graphiques pour le Drag & Drop en Java
 - Effets graphiques ...
 - <http://gfx.developpez.com/tutoriel/java/swing/drag>
 - Fold n'Drop une technique pour faciliter le D&D avec plusieurs fenêtres
 - <http://www.dgp.toronto.edu/~dragice/foldndrop/>

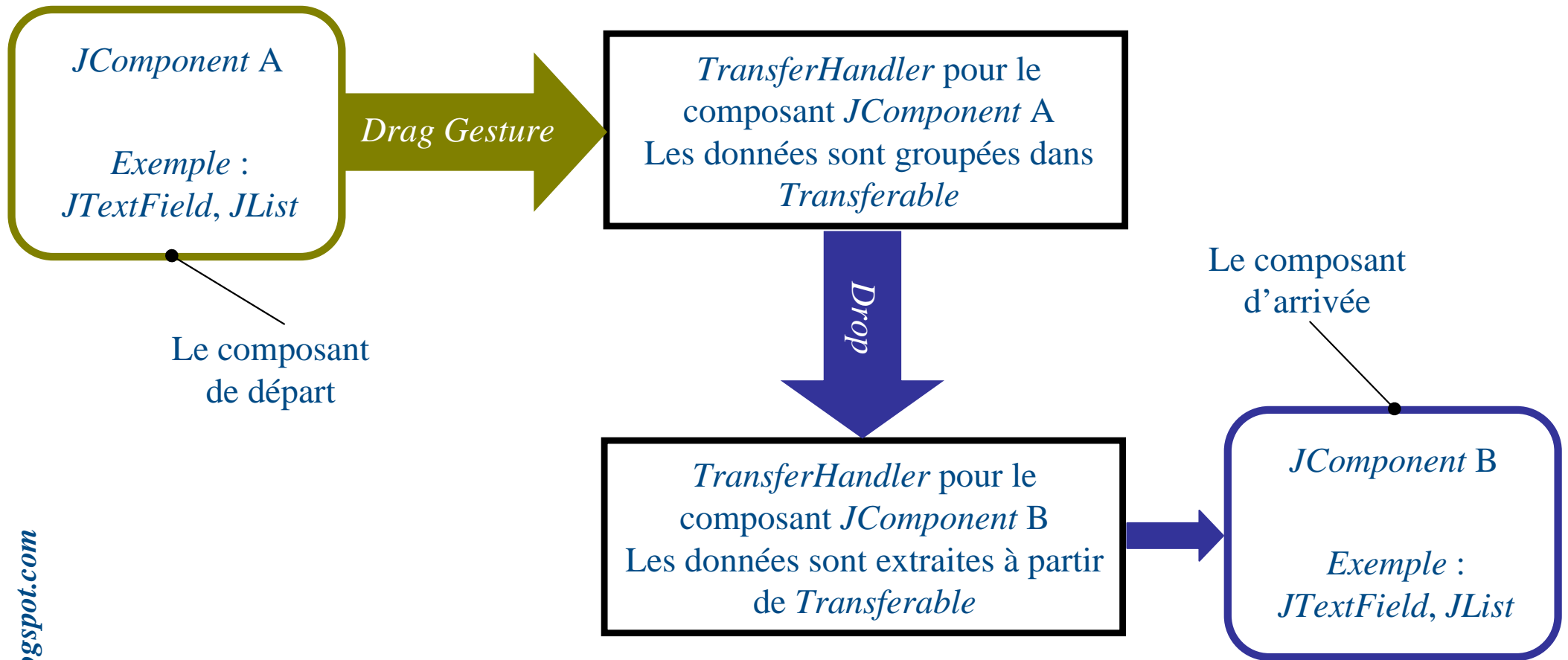


Mise en œuvre du « Drag and Drop » avec Java

- Le technique du « Drag and Drop » repose sur la capacité du transfert de données
- Le transfert des données peut se faire
 - Entre composants
 - Entre applications Java
 - Entre applications natives et applications Java
- La capacité du transfert de données peut prendre deux formes
 - « Drag and Drop » par l'utilisation de la manipulation directe
 - « Presse-papier » via les concepts de copier/couper/coller

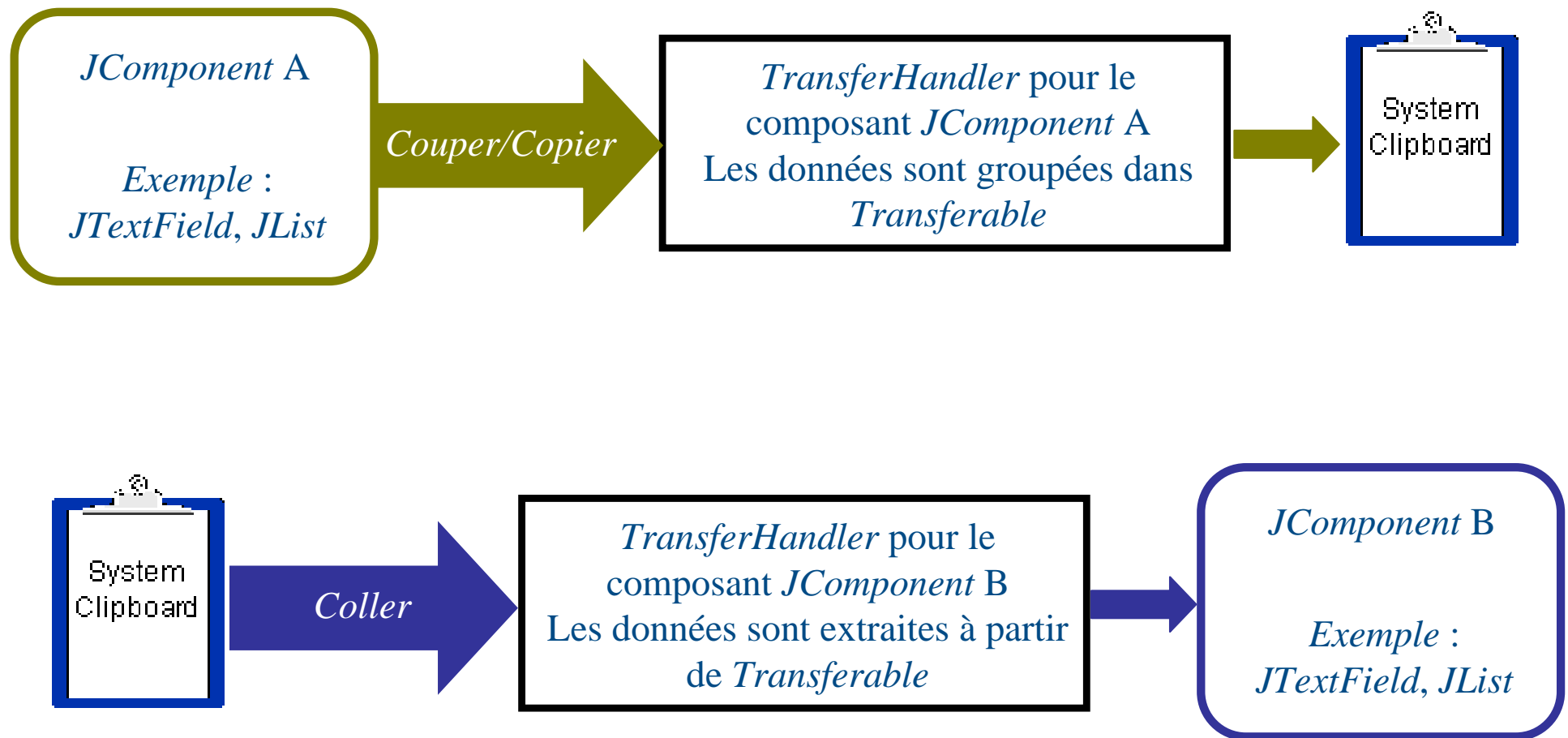
Mise en œuvre du « Drag and Drop » avec Java

► Principe du « Drag and Drop » en Java



Mise en œuvre du « Drag and Drop » avec Java

➤ Principe du « Couper/Copier/Coller » en Java



Mise en œuvre du « Drag and Drop » avec Java

- L'objet *TransferHandler* est le cœur du système de transfert de données
- De nombreux composants Swing fournissent le support du « Drag and Drop » et du transfert de données

Component	Drag* Copy	Drag* Move	Drop	Cut	Copy	Paste
JColorChooser**	✓		✓			
JEditorPane	✓	✓	✓	✓	✓	✓
JFileChooser***	✓				✓	
JFormattedTextField	✓	✓	✓	✓	✓	✓
JList	✓				✓	
JPasswordField	n/a	n/a	✓	n/a	n/a	✓
JTable	✓				✓	
JTextArea	✓	✓	✓	✓	✓	✓
TextField	✓	✓	✓	✓	✓	✓
JTextPane	✓	✓	✓	✓	✓	✓
JTree	✓				✓	

* Activable/Désactivable par le composant

** Importer/Exporter de type Color

*** Exporter une liste de fichier



Dans certains cas le « Drop » n'est pas implémenté car dépendant du modèle (voir dans la suite)

Mise en œuvre du « Drag and Drop » : composant

- Les composants listés dans le tableau précédent permettent d'activer ou pas la capacité de glisser les données
 - *setDragEnabled(boolean p)* : active ou désactive la possibilité de glisser des données
- Au contraire les composants non présents dans le tableau ne permettent pas de gérer directement le glisser
- Pour tous les composants possibilité de modifier l'objet *TransferHandler*
 - *setTransferHandler(TransferHandler p)* : modifie l'objet de transfert de données
- Exemple : activer ou pas la possibilité de glisser un objet

```
JList myList = new JList();  
myList.setDragEnabled(true);  
myList.setTransferHandler(new MyTransferHandler());  
JLabel myLabel = new JLabel();  
myLabel.setDragEnabled(true); // Erreur  
myLabel.setTransferHandler(new MyTransferHandler());
```

Le *JLabel*
n'offre pas la
possibilité de
gérer le
glisser

Mise en œuvre du « Drag and Drop » : *TransferHandler*

- L'objet *TransferHandler* est le cœur du transfert de données
- Cette classe permet de gérer les autorisations de glisser et déposer mais effectue aussi les effets de bords
- Construction d'un objet *TransferHandler*
 - *TransferHandler()* : crée un objet *TransferHandler*
 - *TransferHandler(String)* : crée un objet *TransferHandler* avec le nom de la propriété à transférer. Exemple *new TransferHandler(« String »)*
- On distingue les méthodes liées à l'importation pour le composant cible (reçoit des données) et les méthodes liées à l'exportation pour le composant source (émet des données)
- Pour avoir l'effet désiré vous aurez besoin de redéfinir certaines méthodes de la classe *TransferHandler*

Mise en œuvre du « Drag and Drop » : TransferHandler

- Considérons le cas où l'on souhaite importer des données dans un composant
- Au moment du **déposer** il est intéressant de savoir si le composant cible peut accepter ou pas les données
 - *boolean canImport(JComponent c, DataFlavor[] d)* : retourne *true* si les données *d* sont acceptées par le composant cible *c* sinon retourne *false*
- Le système de transfert de données modifie le curseur pour indiquer que le composant accepte les données
- Dans le cas où l'importation est possible le système de transfert de données appelle
 - *boolean importData(JComponent c, Transferable t)* : importe les données *t* dans le composant cible *c*. Si l'importation est correctement effectuée retourne *true* et effectue l'exportation sinon *false*

Mise en œuvre du « Drag and Drop » : TransferHandler

- Considérons le cas où l'on souhaite exporter des données d'un composant
- Au moment du **glisser** le système de transfert de données prépare les données du composant source
 - *Transferable createTransferable(JComponent c)* : retourne les données à transférer où *c* correspond au composant source
- Possibilité de déterminer le type de « Drag'n Drop »
 - *int getSourceActions(JComponent c)* : retourne le type de « Drag'n Drop » *COPY, MOVE, COPY_OR_MOVE* ou *NONE*
- Une fois l'importation des données sur le composant cible, le composant source peut effectuer un dernier traitement
 - *exportDone(JComponent c, Transferable t, int ty)* : effectue un traitement sur le composant *c* (la source), des données *t* selon le mode *ty* (copie, déplacement, les deux ou aucun)

Mise en œuvre du « Drag and Drop » : DataFlavor

- Les format des objets glissés sont identifiés par des objets de type *DataFlavor*
- Un objet *DataFlavor* est juste un objet java qui représente un type MIME comparable à un identifiant (ne stocke pas de données)
- Les types MIME permettent aux applications de reconnaître les types de données. Exemple « text/html », « image/png »
- La classe *DataFlavor* contient des constantes pour les types les plus courants
 - *stringFlavor* : *DataFlavor* représentant la classe *String*
 - *imageFlavor* : *DataFlavor* représentant la classe *Image*
 - *javaFileListFlavor* : *DataFlavor* représentant la classe *java.util.List* où chaque élément doit être un objet *File*
 - *javaJVMLocalObjectMimeType* : *DataFlavor* représentant une classe de type *Transferable*

Mise en œuvre du « Drag and Drop » : DataFlavor

- Pour créer un objet *DataFlavor* il faut fournir à la construction le type MIME
 - *DataFlavor(String mimeType)* throws *ClassNotFoundException* : construit un objet *DataFlavor* avec le type MIME *mimeType*
- Exemple : construction d'un *DataFlavor* pour la classe *Voiture*

```
String mimeType = DataFlavor.javaJVMLocalObjectMimeType + ";class=" +
    Voiture.class.getName();

try {
    myVoitureDataFlavor = new DataFlavor(mimeType);
} catch (ClassNotFoundException e) {
    return myVoitureDataFlavor;
}
```



Faites attention à la notion de package pour vos classes quand vous construisez un *DataFlavor*



Prenez toujours en compte le *DataFlavor* de type *stringFlavor* qui reste un *DataFlavor* « universel »

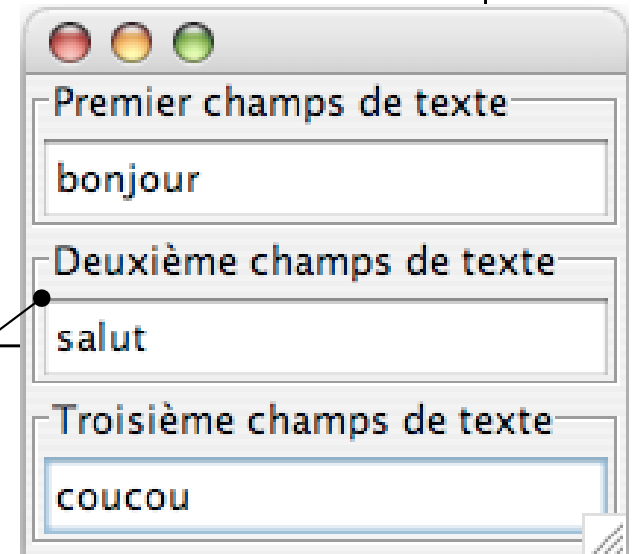
Mise en œuvre du « Drag and Drop » : Transferable

- L'interface *Transferable* s'occupe de stocker les données qui sont manipulées par le système de transfert
 - *DataFlavor[] getTransferDataFlavors()* : la liste des *DataFlavor* supportées pour que les composants cibles potentiels sachent si ils peuvent recevoir la donnée
 - *boolean isDataFlavorSupported(DataFlavor flavor)* : retourne *true* si *flavor* est accepté sinon *false*
 - *Object getTransferData(DataFlavor flavor) throws IOException, UnsupportedFlavorException* : retourne un objet selon la valeur de *flavor*
- Il existe une implémentation de l'interface *Transferable* qui s'occupe de la gestion des objets de type *String*
 - *StringSelection(String pvalue)* : crée un objet *Transferable* spécialisé dans la gestion des objets *String*

Mise en œuvre du « Drag and Drop » : l'exemple ...

➤ Exemple : trois champs de texte

```
public class SimpleExample extends JFrame {  
    ...  
    public SimpleExample() {  
        JTextField myTopTF = new JTextField();  
        myTopTF.setDragEnabled(true);  
        myTopTF.setTransferHandler(new MyTransferHandler());  
  
        JTextField myMiddleTF = new JTextField();  
        myMiddleTF.setTransferHandler(new MyTransferHandler());  
        myMiddleTF.setDragEnabled(true);  
  
        JTextField myBottomTF = new JTextField();  
        myBottomTF.setDragEnabled(true);  
        ...  
    }  
}
```



Seul le *JTextField* du haut et du milieu
a un *TransferHandler* personnalisé.
Le *JTextField* du bas garde son
TransferHandler par défaut

Mise en œuvre du « Drag and Drop » : Objet à transférer

➤ Exemple : objet particulier à transférer entre des composants

```
public class MyObject {
    private String name;
    private int year;

    public MyObject(String value) {
        name = value;
        year = 10;
    }
    public String getName() {
        return name;
    }
    public int getYear() {
        return year;
    }
    public String toString() {
        return this.name + " " + this.year;
    }
}
```

Cette classe décrit
l'objet qui sera
transféré pour notre
exemple

Dans la majorité des cas il s'agit
d'un objet de type *String*

Mise en œuvre du « Drag and Drop » : Transferable

➤ Exemple : objet *Transferable* avec un *DataFlavor* particulier

```

public class TransferableExample implements Transferable {

    public static DataFlavor myData;
    private MyObject myValue;

    public TransferableExample(String value) {
        String mimeType = DataFlavor.javaJVMLocalObjectMimeType +
            ";class=" + MyObject.class.getName();
        try {
            myData = new DataFlavor(mimeType);
            myValue = new MyObject(value);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public DataFlavor[] getTransferDataFlavors() {
        return new DataFlavor[] {myData, DataFlavor.stringFlavor};
    }
    ...
}

```

Création d'un *DataFlavor*
qui correspond à notre
objet *MyObject*

Mise en œuvre du « Drag and Drop » : Transferable

➤ Exemple (suite) : objet *Transferable* avec un *DataFlavor* ...

```
public boolean isDataFlavorSupported(DataFlavor df) {  
    return (df.equals(myData) ||  
           df.equals(DataFlavor.stringFlavor));  
}
```

```
public Object getTransferData(DataFlavor df) throws  
    UnsupportedFlavorException, IOException {
```

```
    if (df == null) {  
        throw new IOException();  
    }
```

```
    if (df.equals(myData)) {  
        return myValue;  
    }
```

```
    if (df.equals(DataFlavor.stringFlavor)) {  
        return myValue.toString();
```

```
    } else {  
        throw new UnsupportedFlavorException(df);  
    }
```

```
    }  
}
```

Cette méthode retourne selon le *DataFlavor* la valeur résultant du « Drag'n Drop »

Toujours prévoir un *DataFlavor* de type *stringFlavor*

Mise en œuvre du « Drag and Drop » : TransferHandler

➤ Exemple : importation de données dans un *JComponent*

```
public class MyTransferHandler extends TransferHandler {  
    public boolean canImport(JComponent cp, DataFlavor[] df) {  
        for (int i = 0; i < df.length; i++) {  
            if (df[i].equals(TransferableExample.myData)) {  
                return true;  
            }  
            if (df[i].equals(DataFlavor.stringFlavor)) {  
                return true;  
            }  
        }  
        return false;  
    }  
    public boolean importData(JComponent cp, Transferable df) {  
        if (df.isDataFlavorSupported(TransferableExample.myData)) {  
            try {  
                Object myObject = df.getTransferData(TransferableExample.myData);  
                MyObject value = (MyObject)myObject;  
                ((JTextField)cp).setText(value.getName());  
                return true;  
            } catch (Exception e) {  
                e.printStackTrace();  
            }  
        }  
        ...  
    }  
}
```

Cette méthode détermine si *cp* peut accepter les *DataFlavor df*

Si importation autorisée alors réalisation de l'importation

Se produit quand un composant source transmet des données via notre *TransferHandler*

Mise en œuvre du « Drag and Drop » : TransferHandler

➤ Exemple (suite) : importation de données dans un *JComponent*

Suite de la méthode
importData

```
} else if (df.isDataFlavorSupported(DataFlavor.stringFlavor)) {  
    try {  
        Object myObject = df.getTransferData(DataFlavor.stringFlavor);  
        if (myObject != null) {  
            ((JTextField)cp).setText((String)myObject);  
            return true;  
        }  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}  
return false;  
}
```

S'il ne s'agit pas d'un *DataFlavor stringFlavor* ni *myData* ou qu'il y a un problème pour extraire les données alors ne rien faire

Se produit quand un composant source transmet des données via un *TransferHandler* différent de *MyTransferHandler*

Mise en œuvre du « Drag and Drop » : TransferHandler

➤ Exemple : exportation de données d'un *JComponent*

Suite de la classe
MyTransferHandler

```
protected Transferable createTransferable(JComponent cp) {  
    return new TransferableExample(((JTextField)cp).getText());  
}  
  
protected void exportDone(JComponent cp, Transferable t, int type) {  
    if (type == TransferHandler.MOVE) {  
        ((JTextField)cp).setText("Vide");  
    } else if (type == TransferHandler.COPY) {  
        ((JTextField)cp).setText("Copy");  
    }  
}  
  
public int getSourceActions(JComponent cp) {  
    return COPY_OR_MOVE;  
}  
}
```

Traitement sur le
composant source

Détermine le type de
« Drag'n Drop »

Mise en œuvre du « Drag and Drop » : TransferHandler

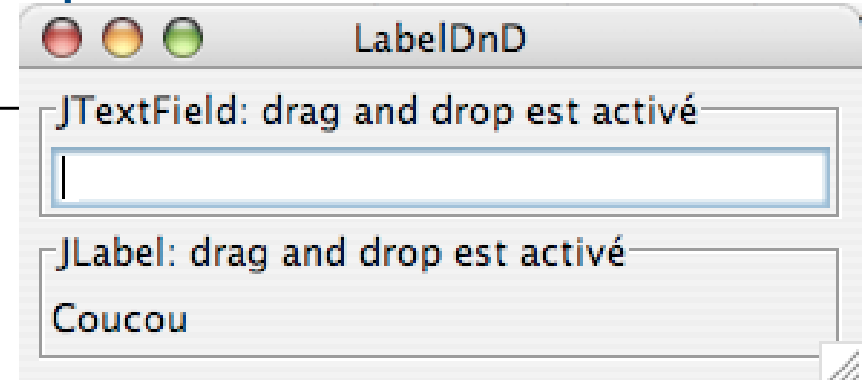
- La classe *TransferHandler* fournit également deux autres méthodes permettant de déclencher explicitement le processus de transfert
 - Pour les composants ne comportant pas de processus de transfert
 - Aiguiller le transfert : « Drag'n Drop » ou « Presse-Papier »
- Le déclenchement explicite doit être réalisé par des événements bas niveaux (*mousePressed* par exemple)
 - *exportAsDrag(JComponent cp, InputEvent ev, int t)* : initialise un processus de transfert « Drag'n Drop » de type *t* de la source *cp*
 - *exportToClipboard(JComponent cp, Clipboard cb, int t)* : initialise un processus de transfert « Presse-Papier »
- Il existe une dernière méthode qui doit permettre de modifier l'icône qui s'affiche pendant le processus de transfert
 - *Icon getVisualRepresentation(Transferable)* : retourne l'icône de transfert

Mise en œuvre du « Drag and Drop » : exemples

➤ Exemple : ajouter le support du DND pour un *JLabel*

```
public class LabelDND extends JFrame {
    public LabelDND() {
        ...
        JTextField myTF = new JTextField();
        myTF.setDragEnabled(true);

        JLabel myLabel = new JLabel("Coucou");
        // myLabel.setDragEnabled(true); Erreur méthode inexistante
        myLabel.setTransferHandler(new TransferHandler("text"));
        myLabel.addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                JComponent c = (JComponent)e.getSource();
                TransferHandler handler = c.getTransferHandler();
                handler.exportAsDrag(c,e,TransferHandler.COPY);
            }
        });
    }
}
```



Déclenche le DND du
composant *JLabel*

Mise en œuvre du « Drag and Drop » : exemples

► Exemple : Importation de fichiers dans une application Java

```
public class ImportFiles extends JFrame {
    public ImportFiles() {
        ...
        myFiles = new JTextArea();
        JScrollPane myScrollPane = new JScrollPane(myFiles);
        myFiles.setDragEnabled(false);
        myFiles.setTransferHandler(new MyTransferHandler());
        getContentPane().add(BorderLayout.CENTER, myScrollPane);
        this.pack();
        this.setVisible(true);
    }
    class MyTransferHandler extends TransferHandler {
        public boolean canImport(JComponent cp, DataFlavor[] df) {
            for (int i = 0; i < df.length; i++) {
                if (df[i].equals(DataFlavor.javaFileListFlavor)) {
                    return true;
                }
                if (df[i].equals(DataFlavor.stringFlavor)) {
                    return true;
                }
            }
            return false;
        }
        ...
    }
}
```

Mise en œuvre du « Drag and Drop » : exemples

➤ Exemple (suite) : Importation de fichiers dans une application

```

public boolean importData(JComponent cp, Transferable tr) {
    if (this.hasFileFlavor(tr.getTransferDataFlavors())) {
        try {
            List files = (List)(tr.getTransferData(DataFlavor.javaFileListFlavor));
            String myList = "";
            for (int i = 0; i < files.size(); i++) {
                File file = (File)files.get(i); myList += file.getName() + "\n";
            }
            myFiles.setText(myList); return true;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

    if (this.hasStringFlavor(tr.getTransferDataFlavors())) {
        try {
            String myData =(String)(tr.getTransferData(DataFlavor.stringFlavor));
            myFiles.setText(myData);
        } catch (Exception e) { e.printStackTrace(); }
    }
    return false;
}

```

Y-a-t-il un *DataFlavor* de type *javaFileListFlavor*

Y-a-t-il un *DataFlavor* de type *StringFlavor*