

Universität Osnabrück  
Fachbereich Mathematik / Informatik  
Informatik  
WS 1999 / 2000  
6.754 Java APIs und Tools  
Leitung: B. Kühl

## Drag'n'Drop mit Java 2

Britta Koch  
Am Salzmarkt 3, WG 11  
49074 Osnabrück  
M.A. Computerlinguistik und Künstliche Intelligenz  
CL/KI(5) / Philosophie(5) / Informatik(5)

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>2</b>
<b>2</b>	<b>Transferables</b>	<b>2</b>
2.1	Codebeispiel: . . . . .	3
<b>3</b>	<b>Drag'n'Drop API</b>	<b>3</b>
3.1	Draggable Component . . . . .	3
3.2	Codebeispiel: . . . . .	4
3.3	Droppable Component . . . . .	5
3.4	Codebeispiel . . . . .	5
3.5	Grafische Darstellung des Event-Flusses . . . . .	6
<b>4</b>	<b>Applet-Besonderheiten</b>	<b>6</b>
<b>5</b>	<b>Besonderheiten, Probleme und Bugs</b>	<b>7</b>
<b>6</b>	<b>Links</b>	<b>8</b>

## 1 Einführung

Fast jeder von uns hat einmal Drag'n'Drop benutzt: eine Datei im Explorer kopiert oder in einen Editor gezogen...jetzt gibt es auch in Java plattformunabhängig die Möglichkeit, dies zu implementieren. Ab Java 2 (JDK 1.2) ist dieses durch die `Datatransfer-API` möglich, sowohl in Swing als auch im AWT. Man kann Drag'n'Drop-Vorgänge sogar programmatisch auslösen! Einen groben Überblick über die nötigen Elemente bietet die Abbildung:



Abbildung 1: Grober Überblick über das Konzept

Im Folgenden werde ich zunächst auf `Transferables` eingehen, danach die Drag'n'Drop-API behandeln und schließlich noch etwas zu Besonderheiten, Problemen und bekannten Bugs berichten.

## 2 Transferables

Das Interface `java.awt.datatransfer.Transferable` verpackt Daten. Es ist serializable. Wenn über ein `Transferable` innerhalb der gleichen JVM Daten geschickt werden, kann man dies mit einer Objektreferenz machen, ansonsten über einen `InputStream` mit anderen JVMs oder sogar mit nativen Anwendungen kommunizieren. Wichtig sind in diesem Zusammenhang die `DataFlavors`, die ähnlich MIME-Typen den Inhalt beschreiben, z. B. `plainTextFlavor`, `stringFlavor`, `javaFileListFlavor`.

Es ist ratsam, seine eigenen `Transferable` zu schreiben, da die einzige `Transferable` von Sun, `StringSelection`, von `StringReader` und nicht von `InputStream` abstammt und daher `ClassCastExceptions` verursacht. Wenn einem die `DataFlavors` nicht reichen, kann man auch da neue implementieren - eine genauere Beschreibung würde hier aber zu weit führen. Mit Hilfe von `Transferables` kann man über ein Clipboard auch Cut'n'Paste implementieren.

Falls man Java-Objekte in eine `Transferable` stecken will, sollte man bei `java.awt.Image` beachten, dass es nicht serializable ist, der Abkömmling `javax.swing.ImageIcon` allerdings schon! Ausserdem muss man beim Drag'n'Drop von Bildern unter Win32 aufpassen: das Clipboard unterstützt keine Formate wie JPEG oder PNG, sondern BMP u. ä.

Als Codebeispiel folgt eine selbst implementierte rudimentäre `StringTransferable`.

## 2.1 Codebeispiel:

```
class StringTransferable implements Transferable{ //weil StringSelection irgendwo buggy is

    String data;
    DataFlavor remoteFlavor = DataFlavor.plainTextFlavor;
    DataFlavor localFlavor = DataFlavor.stringFlavor;
    DataFlavor [] flavors = {remoteFlavor, localFlavor};

    public StringTransferable(String data){
        this.data = data;
    }

    public DataFlavor[] getTransferDataFlavors(){
        return flavors;
    }

    public boolean isDataFlavorSupported(DataFlavor flavor){
        return (flavor == remoteFlavor) || (flavor == localFlavor);
    }

    public Object getTransferData(DataFlavor flavor) throws UnsupportedFlavorException,
        if(!isDataFlavorSupported(flavor)){
            throw new UnsupportedFlavorException(flavor);
        }
        if(flavor.equals(localFlavor)){ //Daten zurueckgeben, lokal
            return data;
        }
        return new ByteArrayInputStream(data.getBytes()); //oder an andere JVM
    }

}
```

## 3 Drag'n'Drop API

### 3.1 Draggable Component

Im Package `java.awt.dnd` gibt es verschiedene Klassen. Die `DnDConstants` definieren Integer-Konstanten für die verschiedenen Drag'n'Drop-Aktionen wie Copy, Move oder Link. Cursor dafür werden von der `DragSource` als statische Variablen bereitgestellt.

Die Quelle einer Dragaktion sollte mit `java.awt.dnd.DragSource`, `java.awt.dnd.DragGestureRecognizer`, `java.awt.dnd.DragGestureListener`, `java.awt.datatransfer.Transferable` und `java.awt.dnd.DragSourceListener` assoziiert sein. Eine `DragSource` bekommt man mit `public static DragSource.getDefaultDragSource`, diese hält für die Lebensdauer der JVM. Ein `DragGestureRecognizer` kümmert sich um verschiedene Drags (Copy, Link, Move) und wie sei auf verschiedenen Plattformen begonnen werden (Mac, Motif, Win32). Mit `dragSource.createDefaultDragGestureRecognizer(Component, Aktion,`

Listener) wird ein neuer DragGestureRecognizer zur Verfügung gestellt. Mit Hilfe der `dragGestureRecognized`-Methode gibt dieser an den DragGestureListener weiter, wann und was für ein Drag gestartet wurde.

Die beiden Interfaces `DragGestureListener` und `DragSourceListener` muss man selbst implementieren. Wenn der `DragGestureListener` benachrichtigt wurde, überprüft er das Event, ob auch ein Drag gestartet wurde, der erlaubt ist, holt sich die zugehörigen Daten und verpackt sie in eine `Transferable`, die dann in `startDrag` gesteckt wird.

Ein `DragSourceListener` hat die Methoden `dragEnter`, `dragExit` und `dragOver`, wo man z. B. das Aussehen des Cursors verändern kann. Unter nicht-Win32-Systemen kann man auch eigene Bilder für den Cursor verwenden. Die Methode `dropActionChanged` wird aufgerufen, wenn sich die Drop-Aktion verändert hat - der User hat z. B. mit Control und Maus angefangen zu draggen und dann Control losgelassen. So kann man überprüfen, ob das Dragen dann noch erlaubt ist.

Am Ende des Drags wird `dragDropEnd` aufgerufen - das dazugehörige Event kann man dann mit `getDropSuccess()` nach dem Erfolg des Drags fragen und z. B. bei Move das gedragte Objekt löschen.

### 3.2 Codebeispiel:

```
class DGListener implements DragGestureListener{

    public void dragGestureRecognized(DragGestureEvent dge){ // los, drag!
        try{
            Transferable t = getTransferableAt(DnDList.this.getSelectedIndex());
            dragSource.startDrag(dge, okCursor, t, sourceListener);
        }
        catch(InvalidDnDOperationException e){
            e.printStackTrace();
        }
    }
}

class DSListener implements DragSourceListener{

    public void dragEnter(DragSourceDragEvent dse){ //darf das hierlang?
        DragSourceContext context = dse.getDragSourceContext();
        int action = dse.getDropAction();
        if((action & accept) == 0){ //falsche Aktion
            context.setCursor(noCursor);
        }
        else{
            context.setCursor(okCursor);
        }
    }

    public void dragOver(DragSourceDragEvent dse){ // oder hierdrueber
        dragEnter(dse);
    }
}
```

```

public void dragExit(DragSourceEvent dse){ // raus gehts
}

public void dragDropEnd(DragSourceDropEvent dse){ // drag ist mit drop zuende
    if(dse.getDropSuccess() == false){//drop tat nicht
        return;
    }
    int action = dse.getDropAction();
    if((action & accept) == 0){ // falsche Aktion
        return;
    }
    try{
        Object data = dse.getDragSourceContext().getTransferable().getTransferData(loc
        if(data == null ||!(data instanceof String)){
            return;
        }
        removeItem(data); // das gedropte hier loeschen
    }
    catch(Throwable t){
        t.printStackTrace();
    }
}

public void dropActionChanged(DragSourceDragEvent dse){ //z.B. ctrl loslassen
    dragEnter(dse); //ueberpruefen
}
}

```

### 3.3 Droppable Component

Eine Droppable Component sollte mit einem `java.awt.dnd.DropTarget` und einem `java.awt.dnd.DropTargetListener` assoziiert sein. Das `DropTarget` wird einfach mit der Component, den Aktionen und dem Listener erzeugt.

Der Listener braucht eine Assoziation mit einer Component für `dragUnder`-Effekte bei `dragEnter`, `dragOver` und `dragExit`. In `drop` geschieht dann der Drop: es wird abgecheckt, ob der Drop auch erlaubt ist, dann wird die `Transferable` geholt, evtl. anhand des `DataFlavors` nochmal auf Verwendbarkeit überprüft und dann werden die Daten benutzt (eingefügt / gelinkt etc.).

### 3.4 Codebeispiel

```

class DTListener implements DropTargetListener{

    public void dragEnter(DropTargetDragEvent e){ //darf man hierher draggen?
        if(isDragOK(e) == false){
            e.rejectDrag(); //elegant
            return;
        }
        e.acceptDrag(accept); //das auch
    }
}

```

```

    }
    public void dragExit(DropTargetEvent e){ //drag vorbei
    }

    public void dragOver(DropTargetDragEvent e){//drueber wird gedraggt
        if(isDragOK(e) == false){
            e.rejectDrag();
            return;
        }
        e.acceptDrag(accept);
    }

    public void drop(DropTargetDropEvent e){ //drop
        Object data;
        if((data = validateDrop(e)) == null){ //nix ok
            e.rejectDrop(); //ich habe fertig
            return;
        }
        e.acceptDrop(accept); //doch ok
        if(!doTheDrop(e, data)){ //aber hat trotzdem nicht geklappt
            e.dropComplete(false); //also nicht vorbei
            return;
        }
        e.dropComplete(true); //hier doch
    }

    public void dropActionChanged(DropTargetDragEvent e){ //wieder ctrl losgelassen
        if(isDragOK(e) == false){
            e.rejectDrag();
            return;
        }
        e.acceptDrag(accept);
    }
}

```

### 3.5 Grafische Darstellung des Event-Flusses

Siehe Abbildung 2, Seite 7.

## 4 Applet-Besonderheiten

Für Applets mit Drag und Drop braucht man einen Browser, der Java 2 oder entsprechende Plugins unterstützt. Gewisse Properties müssen in der Datei policy gesetzt sein:

```

grant {
    permission java.awt.AWTPermission "accessEventQueue";
    permission java.awt.AWTPermission "setDropTarget";
}

```

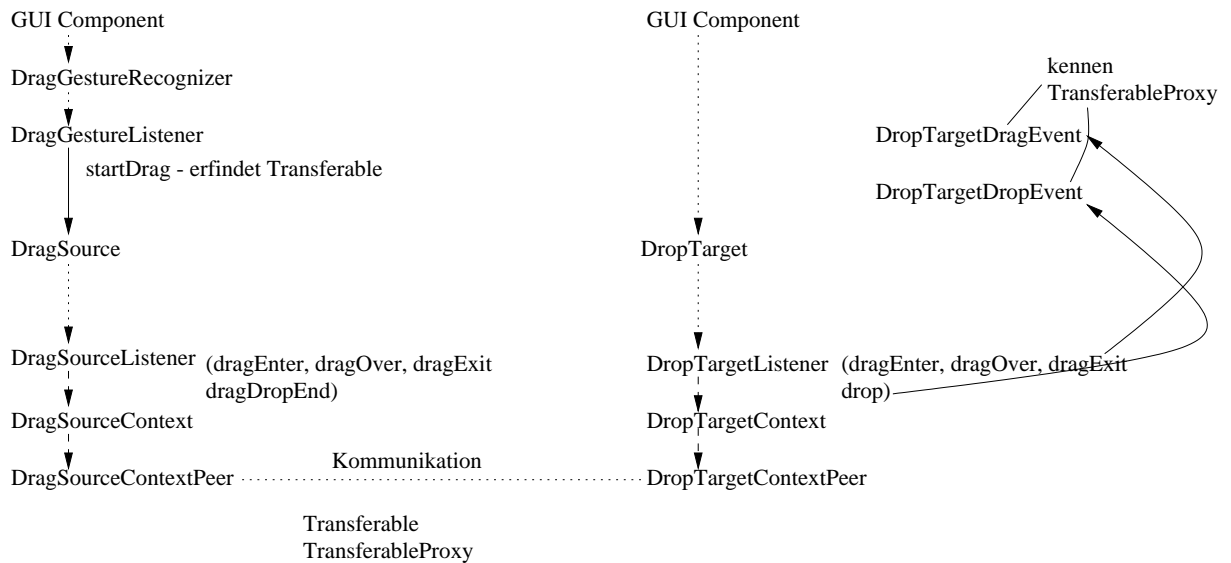


Abbildung 2: Der Event-Fluss beim Drag'n'Drop

```

permission java.awt.AWTPermission "accessClipboard";
permission java.awt.AWTPermission "acceptDropBetweenAccessControllerContexts";
permission java.awt.AWTPermission "listenToAllAWTEvents";
};

```

Mit `appletviewer -J-Djava.security.policy=policy index.html` kann man dann sein DnD-Applet aufrufen. Allerdings sollte man die DropTargets nicht in der `init()` oder der `start()`-Methode erzeugen, sondern in einem Extra-Thread. Ansonsten kann man nur draggen, aber nicht droppen.

## 5 Besonderheiten, Probleme und Bugs

- Wenn man `JFileChooser` oder `JColorChooser` Drag'n'Drop enabled machen möchte, sollte man sie ableiten und eine über den ganzen Dialog gelegte `Glasspane` als `DropTarget` benutzen.
- Möchte man bei einer Drag'n'Drop-Aktion beim Benutzer nachfragen, ist dies etwas komplizierter: da Swing nicht Thread-safe ist, müsste man die Nachfrage an ein `invokeLater()` weitergeben, aber selbst da passieren anscheinend deadlocks: Abhilfe ist es, das Paket `SwingWorkers` zu benutzen (muss man sich extra von der `SwingConnection` herunterladen).
- Auf ein Textfeld kann man droppen, aber nicht draggen, da dies Probleme mit dem Cursor gibt.
- Wenn ein selbst gesetzter Cursor beim Draggen flackert, so muss man erst den Default-Cursor auf `null` setzen und dann den eigenen verwenden.



- Bei einer `JTabbedPane` kann man Drag'n'Drop nur mit dem ersten Panel verwenden: soll dies auch auf anderen Panels geschehen, so muss man `JTabbedPane` ableiten und `findComponentAt` überschreiben.
- Mit `CardLayout` kann man *kein* Drag'n'Drop verwenden.

## 6 Links

Der gesamte Code : <http://www-lehre.uni-osnabrueck.de/~bkoch/japi/DnDList.java>

Tolle Tutorials von Gene DeLisa in der JavaWorld:

Teil 1 : <http://www.javaworld.com/javaworld/jw-03-1999/jw-03-dragndrop.html>

Teil 2 : <http://www.javaworld.com/javaworld/jw-08-1999/jw-08-draganddrop.html>