**INSTITUTO POLITECNICO NACIONAL**

**Escuela Superior de Cómputo**

**Aplicaciones para Comunicaciones de Red**

**Moreno Cervantes Axel Ernesto**

**Tarea: Implementación de Drag and Drop en Swing**

**Miguel Ángel Morales García**

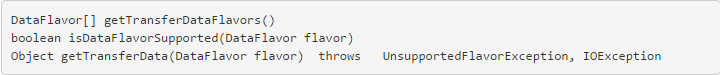
**Grupo: 3CM6**

**Drag and Drop**

DnD es una forma de transferencia de archivos en una aplicación. Nos permite transferir datos arrastrándolos desde un componente y soltándolos en otro. El componente de donde se arrastra se llama fuente arrastrada (drag source) y el componente en donde se suelta se llama objetivo soltado (drop target).

La implementación de DnD se realiza mediante el uso de un Transferable Object y la interfaz Transferable Interface en la librería java.awt.datatransfer.

Transferable Interface contiene los siguientes métodos:



Para habilitar el componente a arrastrar:



Todos los componentes de texto tienen soporte para DnD (JFileChooser, JColorChooser, JList, JTree, and JTable).

Por ejemplo, para arrastrar desde un JTextField llamado nameFld a un JTextArea llamado descTxtArea debemos escribir las siguientes líneas:



Las acciones de transferencia son representadas por las constantes declaradas en la clase TransferHandler:

* TranferHandler.COPY
* TranferHandler.MOVE
* TranferHandler.COPY\_OR\_MOVE
* TranferHandler.LINK
* TranferHandler.NONE

**Métodos de la clase TransferHandler**

En el corazón del mecanismo de transferencia de datos está la clase TransferHandler. Esta clase provee de un mecanismo sencillo de un JComponent a otro. Los siguientes métodos se usan para acoplar un TransferHandler a un componente:

* setDragEnabled(boolean) – Enciende el soporte de arrastrado. Por default es falso. Es definido en cada componente que soporta el arrastrado.
* setDropMode(DropMode) – Configura cómo son determinadas las zonas de arrastrado.
* setTransferHandler(TransferHandler) – Para conectar importación o exportación de datos.
* getSourceAction(JComponent) – Es usado para consultar qué acciones son soportadas por el componente fuente como COPY, MOVE o LINK.
* createTransferable(JComponent) – Envuelve los datos que serán exportados dentro de un objeto Transferable.
* exportDone(JComponent, Transferable, int) – Es invocado cuando la exportación es completada. Cuando la acción es MOVE, la información necesita ser removida de la fuente después de la transferencia.
* canImport(TransferHandler.TransferSupport) – Este método es llamado repetidamente durante el acto de arrastrar y regresa True si el área debajo del cursor puede aceptar la transferencia o falso si la transferencia es rechazada.
* importData(TransferHandler.TransferSupport) – Este método es llamado en un arrastrado exitosa (o pegar) e inicializa la transferencia al componente destino.

La clase TransferSupport respalda el proceso de transferencia ofreciendo diferentes métodos usados para acceder a los detalles de la transferencia. Es decir para obtener información de la clase TransferHandler.

* Component getComponent() – Regresa el componente destino.
* Int getDropAction() – regresa la acción elegida cuando la transferencia es de arrastrado.
* Int getUserDropAction – regresa la acción de arrastrado del usuario.
* Int getSourceDropActions() – Regresa las acciones soportadas por un componente.
* Transferable getTransferable() – Regresa la información Transferable de esta transferencia.
* DropLocation getDropLocation() – Regresa la ubicación del componente de arrastrado.

**Ejemplo**

import java.util.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.text.\*;

import java.awt.datatransfer.\*;

import javax.swing.\*;

import javax.swing.text.\*;

import javax.swing.tree.\*;

import javax.swing.table.\*;

public class BasicDnD extends JPanel implements ActionListener {

private static JFrame frame;

private JTextArea textArea;

private JTextField textField;

private JList list;

private JTable table;

private JTree tree;

private JColorChooser colorChooser;

private JCheckBox toggleDnD;

public BasicDnD() {

super(new BorderLayout());

JPanel leftPanel = createVerticalBoxPanel();

JPanel rightPanel = createVerticalBoxPanel();

//Create a table model.

DefaultTableModel tm = new DefaultTableModel();

tm.addColumn("Column 0");

tm.addColumn("Column 1");

tm.addColumn("Column 2");

tm.addColumn("Column 3");

tm.addRow(new String[]{"Table 00", "Table 01", "Table 02", "Table 03"});

tm.addRow(new String[]{"Table 10", "Table 11", "Table 12", "Table 13"});

tm.addRow(new String[]{"Table 20", "Table 21", "Table 22", "Table 23"});

tm.addRow(new String[]{"Table 30", "Table 31", "Table 32", "Table 33"});

//LEFT COLUMN

//Use the table model to create a table.

table = new JTable(tm);

leftPanel.add(createPanelForComponent(table, "JTable"));

//Create a color chooser.

colorChooser = new JColorChooser();

leftPanel.add(createPanelForComponent(colorChooser, "JColorChooser"));

//RIGHT COLUMN

//Create a textfield.

textField = new JTextField(30);

textField.setText("Favorite foods:\nPizza, Moussaka, Pot roast");

rightPanel.add(createPanelForComponent(textField, "JTextField"));

//Create a scrolled text area.

textArea = new JTextArea(5, 30);

textArea.setText("Favorite shows:\nBuffy, Alias, Angel");

JScrollPane scrollPane = new JScrollPane(textArea);

rightPanel.add(createPanelForComponent(scrollPane, "JTextArea"));

//Create a list model and a list.

DefaultListModel listModel = new DefaultListModel();

listModel.addElement("Martha Washington");

listModel.addElement("Abigail Adams");

listModel.addElement("Martha Randolph");

listModel.addElement("Dolley Madison");

listModel.addElement("Elizabeth Monroe");

listModel.addElement("Louisa Adams");

listModel.addElement("Emily Donelson");

list = new JList(listModel);

list.setVisibleRowCount(-1);

list.getSelectionModel().setSelectionMode(ListSelectionModel.MULTIPLE\_INTERVAL\_SELECTION);

list.setTransferHandler(new TransferHandler() {

public boolean canImport(TransferHandler.TransferSupport info) {

// we only import Strings

if (!info.isDataFlavorSupported(DataFlavor.stringFlavor)) {

return false;

}

JList.DropLocation dl = (JList.DropLocation)info.getDropLocation();

if (dl.getIndex() == -1) {

return false;

}

return true;

}

public boolean importData(TransferHandler.TransferSupport info) {

if (!info.isDrop()) {

return false;

}

// Check for String flavor

if (!info.isDataFlavorSupported(DataFlavor.stringFlavor)) {

displayDropLocation("List doesn't accept a drop of this type.");

return false;

}

JList.DropLocation dl = (JList.DropLocation)info.getDropLocation();

DefaultListModel listModel = (DefaultListModel)list.getModel();

int index = dl.getIndex();

boolean insert = dl.isInsert();

// Get the current string under the drop.

String value = (String)listModel.getElementAt(index);

// Get the string that is being dropped.

Transferable t = info.getTransferable();

String data;

try {

data = (String)t.getTransferData(DataFlavor.stringFlavor);

}

catch (Exception e) { return false; }

// Display a dialog with the drop information.

String dropValue = "\"" + data + "\" dropped ";

if (dl.isInsert()) {

if (dl.getIndex() == 0) {

displayDropLocation(dropValue + "at beginning of list");

} else if (dl.getIndex() >= list.getModel().getSize()) {

displayDropLocation(dropValue + "at end of list");

} else {

String value1 = (String)list.getModel().getElementAt(dl.getIndex() - 1);

String value2 = (String)list.getModel().getElementAt(dl.getIndex());

displayDropLocation(dropValue + "between \"" + value1 + "\" and \"" + value2 + "\"");

}

} else {

displayDropLocation(dropValue + "on top of " + "\"" + value + "\"");

}

/\*\* This is commented out for the basicdemo.html tutorial page.

\*\* If you add this code snippet back and delete the

\*\* "return false;" line, the list will accept drops

\*\* of type string.

// Perform the actual import.

if (insert) {

listModel.add(index, data);

} else {

listModel.set(index, data);

}

return true;

\*/

return false;

}

public int getSourceActions(JComponent c) {

return COPY;

}

protected Transferable createTransferable(JComponent c) {

JList list = (JList)c;

Object[] values = list.getSelectedValues();

StringBuffer buff = new StringBuffer();

for (int i = 0; i < values.length; i++) {

Object val = values[i];

buff.append(val == null ? "" : val.toString());

if (i != values.length - 1) {

buff.append("\n");

}

}

return new StringSelection(buff.toString());

}

});

list.setDropMode(DropMode.ON\_OR\_INSERT);

JScrollPane listView = new JScrollPane(list);

listView.setPreferredSize(new Dimension(300, 100));

rightPanel.add(createPanelForComponent(listView, "JList"));

//Create a tree.

DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode("Mia Familia");

DefaultMutableTreeNode sharon = new DefaultMutableTreeNode("Sharon");

rootNode.add(sharon);

DefaultMutableTreeNode maya = new DefaultMutableTreeNode("Maya");

sharon.add(maya);

DefaultMutableTreeNode anya = new DefaultMutableTreeNode("Anya");

sharon.add(anya);

sharon.add(new DefaultMutableTreeNode("Bongo"));

maya.add(new DefaultMutableTreeNode("Muffin"));

anya.add(new DefaultMutableTreeNode("Winky"));

DefaultTreeModel model = new DefaultTreeModel(rootNode);

tree = new JTree(model);

tree.getSelectionModel().setSelectionMode

(TreeSelectionModel.DISCONTIGUOUS\_TREE\_SELECTION);

JScrollPane treeView = new JScrollPane(tree);

treeView.setPreferredSize(new Dimension(300, 100));

rightPanel.add(createPanelForComponent(treeView, "JTree"));

//Create the toggle button.

toggleDnD = new JCheckBox("Turn on Drag and Drop");

toggleDnD.setActionCommand("toggleDnD");

toggleDnD.addActionListener(this);

JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL\_SPLIT,

leftPanel, rightPanel);

splitPane.setOneTouchExpandable(true);

add(splitPane, BorderLayout.CENTER);

add(toggleDnD, BorderLayout.PAGE\_END);

setBorder(BorderFactory.createEmptyBorder(5,5,5,5));

}

protected JPanel createVerticalBoxPanel() {

JPanel p = new JPanel();

p.setLayout(new BoxLayout(p, BoxLayout.PAGE\_AXIS));

p.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));

return p;

}

public JPanel createPanelForComponent(JComponent comp,

String title) {

JPanel panel = new JPanel(new BorderLayout());

panel.add(comp, BorderLayout.CENTER);

if (title != null) {

panel.setBorder(BorderFactory.createTitledBorder(title));

}

return panel;

}

private void displayDropLocation(final String string) {

SwingUtilities.invokeLater(new Runnable() {

public void run() {

JOptionPane.showMessageDialog(null, string);

}

});

}

public void actionPerformed(ActionEvent e) {

if ("toggleDnD".equals(e.getActionCommand())) {

boolean toggle = toggleDnD.isSelected();

textArea.setDragEnabled(toggle);

textField.setDragEnabled(toggle);

list.setDragEnabled(toggle);

table.setDragEnabled(toggle);

tree.setDragEnabled(toggle);

colorChooser.setDragEnabled(toggle);

}

}

/\*\*

\* Create the GUI and show it. For thread safety,

\* this method should be invoked from the

\* event-dispatching thread.

\*/

private static void createAndShowGUI() {

//Create and set up the window.

frame = new JFrame("BasicDnD");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

//Create and set up the content pane.

JComponent newContentPane = new BasicDnD();

newContentPane.setOpaque(true); //content panes must be opaque

frame.setContentPane(newContentPane);

//Display the window.

frame.pack();

frame.setVisible(true);

}

public static void main(String[] args) {

//Schedule a job for the event-dispatching thread:

//creating and showing this application's GUI.

javax.swing.SwingUtilities.invokeLater(new Runnable() {

public void run() {

//Turn off metal's use of bold fonts

UIManager.put("swing.boldMetal", Boolean.FALSE);

createAndShowGUI();

}

});

}

}