Drag and Drop in Swing

In computer graphical user interfaces, drag-and-drop is the action of (or support for the action of) clicking on a virtual object and dragging it to a different location or onto another virtual object. In general, it can be used to invoke many kinds of actions, or create various types of associations between two abstract objects.

Drag and Drop

Drag and drop functionality is one of the most visible aspects of the graphical user interface. Drag and drop operation enables users to do complex things intuitively.

Usually, we can drag and drop two things: data or some graphical objects. If we drag an image from one application to another, we drag and drop binary data. If we drag a tab in Firefox and move it to another place, we drag and drop a graphical component.

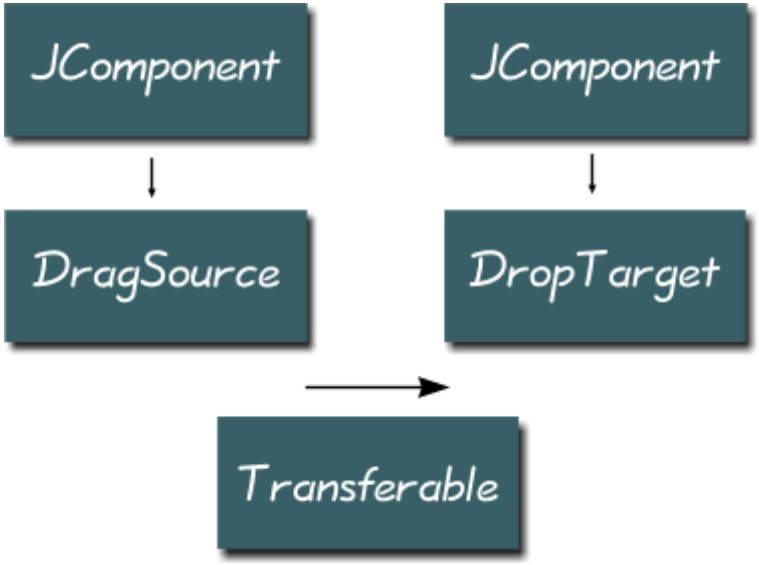

Figure: Drag and drop in Swing

The component, where the drag operation begins must have a DragSource object registered. A DropTarget is an object responsible for accepting drops in an drag and drop operation. A Transferable encapsulates data being transferred. The transferred data can be of various type. A DataFlavor object provides information about the data being transferred.

Several Swing components have already a built-in support for drag and drop operations. In such cases, a Swing programmer uses a TransferHandler to manage the drag and drop functionality. In situations, where there is no built-in support, the programmer has to create everything from scratch.

Swing text drag and drop example

We will demonstrate a simple drag and drop example. We will work with built-in drag and drop support. We utilise a TransferHandler class.

SimpeDnD.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.TransferHandler;
import java.awt.Container;
import java.awt.EventQueue;

public class SimpleDnD extends JFrame {

    private JTextField field;
    private JButton button;
```

```
    public SimpleDnD() {

        initUI();
    }

    private void initUI() {

        setTitle("Simple Drag & Drop");

        button = new JButton("Button");
        field = new JTextField(15);

        field.setDragEnabled(true);
        button.setTransferHandler(new TransferHandler("text"));

        createLayout(field, button);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }

    private void createLayout(JComponent... arg) {

        Container pane = getContentPane();
        GroupLayout gl = new GroupLayout(pane);
        pane.setLayout(gl);

        gl.setAutoCreateContainerGaps(true);
        gl.setAutoCreateGaps(true);

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addComponent(arg[0])
                .addComponent(arg[1])
        );

        gl.setVerticalGroup(gl.createParallelGroup(GroupLayout.Alignment.BASELINE)
```

```
                .addComponent(arg[0])
                .addComponent(arg[1])
        );

        pack();
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

            SimpleDnD ex = new SimpleDnD();
            ex.setVisible(true);
        });
    }
}
```

In our example we have a text field and a button. We can drag a text from the field and drop it onto the button.

```
field.setDragEnabled(true);
```

The text field has a built in support for dragging. We must enable it.

```
button.setTransferHandler(new TransferHandler("text"));
```

The TransferHandler is a class responsible for transferring data between components. The constructor takes a property name as a parameter.

Swing icon drag & drop

Some of the Java Swing components do not have built in drag support. JLabel component is such a component. We have to code the drag functionality ourselves.

The following example shows how to drag and drop icons. In the previous example, we used a text property. This time we use an icon property.

IconDnd.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.TransferHandler;
import java.awt.Container;
import java.awt.EventQueue;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

/*
Java Swing tutorial
Drag and drop icons

Author: Jan Bodnar
Website: http://zetcode.com
 */

public class IconDnD extends JFrame {


    public IconDnD() {
```

```
        initUI();
    }

    private void initUI() {

        ImageIcon icon1 = new ImageIcon("src/main/resources/sad.png");
        ImageIcon icon2 = new ImageIcon("src/main/resources/plain.png");
        ImageIcon icon3 = new ImageIcon("src/main/resources/smile.png");

        JLabel label1 = new JLabel(icon1, JLabel.CENTER);
        JLabel label2 = new JLabel(icon2, JLabel.CENTER);
        JLabel label3 = new JLabel(icon3, JLabel.CENTER);

        MouseListener listener = new DragMouseAdapter();
        label1.addMouseListener(listener);
        label2.addMouseListener(listener);
        label3.addMouseListener(listener);

        JButton button = new JButton(icon2);
        button.setFocusable(false);

        label1.setTransferHandler(new TransferHandler("icon"));
        label2.setTransferHandler(new TransferHandler("icon"));
        label3.setTransferHandler(new TransferHandler("icon"));
        button.setTransferHandler(new TransferHandler("icon"));

        createLayout(label1, label2, label3, button);

        setTitle("Icon Drag & Drop");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }

    private class DragMouseAdapter extends MouseAdapter {

        public void mousePressed(MouseEvent e) {
```

```
            JComponent c = (JComponent) e.getSource();
            TransferHandler handler = c.getTransferHandler();
            handler.exportAsDrag(c, e, TransferHandler.COPY);
        }
    }

    private void createLayout(JComponent... arg) {

        Container pane = getContentPane();
        GroupLayout gl = new GroupLayout(pane);
        pane.setLayout(gl);

        gl.setAutoCreateContainerGaps(true);
        gl.setAutoCreateGaps(true);

        gl.setHorizontalGroup(gl.createParallelGroup(GroupLayout.Alignment.CENTER)
                .addGroup(gl.createSequentialGroup()
                        .addComponent(arg[0])
                        .addGap(30)
                        .addComponent(arg[1])
                        .addGap(30)
                        .addComponent(arg[2])
                )
                .addComponent(arg[3], GroupLayout.DEFAULT_SIZE,
                        GroupLayout.DEFAULT_SIZE, Integer.MAX_VALUE)
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addGroup(gl.createParallelGroup()
                        .addComponent(arg[0])
                        .addComponent(arg[1])
                        .addComponent(arg[2]))
                .addGap(30)
                .addComponent(arg[3])
        );

        pack();
```

```
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

            IconDnD ex = new IconDnD();
            ex.setVisible(true);
        });
    }
}
```

In the code example, we have two labels and a button. Each component displays an icon. The two labels enable drag gestures, the button accepts a drop gesture.

```
label1.setTransferHandler(new TransferHandler("icon"));
label2.setTransferHandler(new TransferHandler("icon"));
label3.setTransferHandler(new TransferHandler("icon"));
```

The drag support is not enabled by default for the label. We register a custom mouse adapter for both labels.

```
label1.setTransferHandler(new TransferHandler("icon"));
label2.setTransferHandler(new TransferHandler("icon"));
label3.setTransferHandler(new TransferHandler("icon"));
button.setTransferHandler(new TransferHandler("icon"));
```

Each of the components has a TransferHandler class for an icon property. The TransferHandler is needed for both drag sources and drag targets as well.

```
JComponent c = (JComponent) e.getSource();
TransferHandler handler = c.getTransferHandler();
handler.exportAsDrag(c, e, TransferHandler.COPY);
```

These code lines initiate the drag support. We get the drag source. In our case it is a label instance. We get its transfer handler object and finally initiate the drag support with the exportAsDrag() method call.

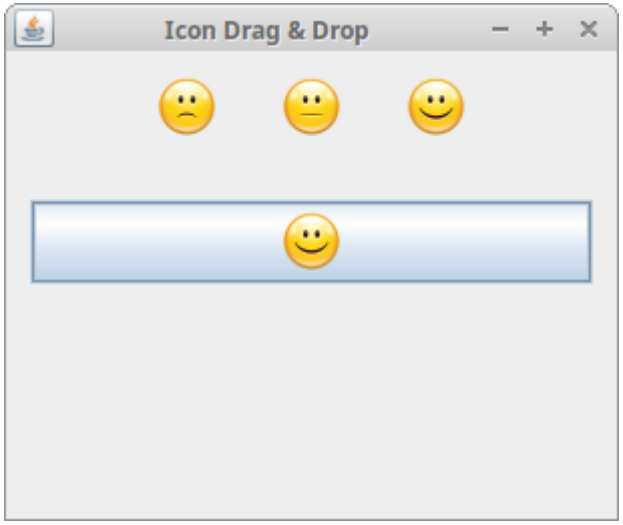

Figure: Icon drag & drop example

Swing JList drop example

Some components do not have a default drop support. One of them is JList. There is a good reason for this. We do not know if the data will be inserted into one row, or two or more rows. So we must implement manually the drop support for the list component.

The following example inserts comma or space separated text into the rows of a JList component. Otherwise, the text goes into one row.

ListDnD.java

```
package com.zetcode;

import javax.swing.DefaultListModel;
import javax.swing.DropMode;
import javax.swing.GroupLayout;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.ListSelectionModel;
import javax.swing.TransferHandler;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.datatransfer.DataFlavor;
import java.awt.datatransfer.Transferable;

/*
Java Swing tutorial
JList drag and drop example

Author: Jan Bodnar
Website: http://zetcode.com
 */

public class ListDnD extends JFrame {

    private JTextField field;
    private DefaultListModel model;

    public ListDnD() {

        initUI();
    }

    private void initUI() {
```

```
        JScrollPane scrollPane = new JScrollPane();
        scrollPane.setPreferredSize(new Dimension(180, 150));

        model = new DefaultListModel();
        JList list = new JList(model);

        list.setDropMode(DropMode.INSERT);
        list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        list.setTransferHandler(new ListHandler());

        field = new JTextField(15);
        field.setDragEnabled(true);

        scrollPane.getViewport().add(list);

        createLayout(field, scrollPane);

        setTitle("ListDrop");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }

    private class ListHandler extends TransferHandler {

        public boolean canImport(TransferSupport support) {

            if (!support.isDrop()) {
                return false;
            }

            return support.isDataFlavorSupported(DataFlavor.stringFlavor);
        }

        public boolean importData(TransferSupport support) {

            if (!canImport(support)) {
```

```
                return false;
            }

            Transferable transferable = support.getTransferable();
            String line;

            try {
                line = (String) transferable.getTransferData(DataFlavor.stringFlavor);
            } catch (Exception e) {
                return false;
            }

            JList.DropLocation dl = (JList.DropLocation) support.getDropLocation();
            int index = dl.getIndex();

            String[] data = line.split("[,\\s]");

            for (String item : data) {

                if (!item.isEmpty())
                    model.add(index++, item.trim());
            }

            return true;
        }
    }

    private void createLayout(JComponent... arg) {

        Container pane = getContentPane();
        GroupLayout gl = new GroupLayout(pane);
        pane.setLayout(gl);

        gl.setAutoCreateContainerGaps(true);
        gl.setAutoCreateGaps(true);

        gl.setHorizontalGroup(gl.createSequentialGroup()
```

```
                .addComponent(arg[0])
                .addComponent(arg[1])
        );

        gl.setVerticalGroup(gl.createParallelGroup(GroupLayout.Alignment.BASELINE)
                .addComponent(arg[0])
                .addComponent(arg[1])
        );

        pack();
    }

    public static void main(String[] args) {
        EventQueue.invokeLater(() -> {

            ListDnD ex = new ListDnD();
            ex.setVisible(true);
        });
    }
}
```

In the example, we have a text field and a list component. The text in the text field can be dragged and dropped into the list. If the text is comma separated with a comma or a space character, the words will be split into rows. If not, the text is inserted into one row.

```
list.setDropMode(DropMode.INSERT);
```

Here we specify a drop mode. The DropMode.INSERT specifies, that we are going to insert new items into the list component. If we chose DropMode.INSERT, we would drop new items onto the existing ones.

```
list.setTransferHandler(new ListHandler());
```

We set a custom transfer handler class.

```
field.setDragEnabled(true);
```

We enable the drag support for the text field component.

```
public boolean canImport(TransferSupport support) {

    if (!support.isDrop()) {
        return false;
    }

    return support.isDataFlavorSupported(DataFlavor.stringFlavor);
}
```

This method tests the suitability of a drop operation. We filter out the clipboard paste operations and allow only String drop operations. If the method returns false, the drop operation is cancelled.

```
public boolean importData(TransferSupport support) {
...
}
```

The importData() method transfers the data from the clipboard or from the drag and drop operation to the drop location.

```
Transferable transferable = support.getTransferable();
```

The Transferable is the class, where the data is bundled.

```
line = (String) transferable.getTransferData(DataFlavor.stringFlavor);
```

We retrieve our data.

```
JList.DropLocation dl = (JList.DropLocation) support.getDropLocation();
int index = dl.getIndex();
```

We get a drop location for the list. We retrieve the index, where the data will be inserted.

```
String[] data = line.split("[,\\s]");

for (String item : data) {

    if (!item.isEmpty())
        model.add(index++, item.trim());
}
```

We split the text into parts and insert it into one or more rows.

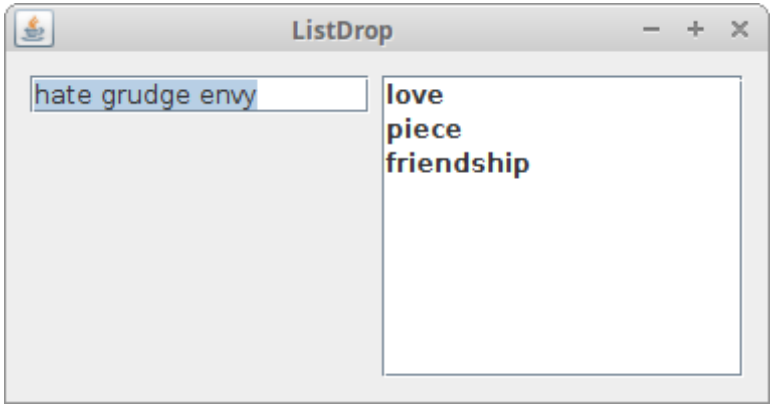

Figure: JList drop example

The previous examples used components with built-in drag and drop support. Next we are going to create a drag and drop functionality from scratch.

Swing drag Gesture

In the following example we will inspect a simple drag gesture. We will work with several classes needed to create a drag gesture. A DragSource, DragGestureEvent, DragGestureListener, Transferable.

DragGesture.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Container;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.datatransfer.DataFlavor;
import java.awt.datatransfer.Transferable;
import java.awt.dnd.DnDConstants;
import java.awt.dnd.DragGestureEvent;
import java.awt.dnd.DragGestureListener;
import java.awt.dnd.DragSource;

/*
Java Swing tutorial
Drag gesture example

Author: Jan Bodnar
Website: http://zetcode.com
 */


public class DragGesture extends JFrame implements
        DragGestureListener, Transferable {

    public DragGesture() {
```

```
        initUI();
    }

    private void initUI() {

        JPanel redPanel = new JPanel();
        redPanel.setBackground(Color.red);
        redPanel.setPreferredSize(new Dimension(120, 120));

        DragSource ds = new DragSource();

        ds.createDefaultDragGestureRecognizer(redPanel,
                DnDConstants.ACTION_COPY, this);

        createLayout(redPanel);

        setTitle("Drag Gesture");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }

    public void dragGestureRecognized(DragGestureEvent event) {

        System.out.println("grag gesture");
        Cursor cursor = null;

        if (event.getDragAction() == DnDConstants.ACTION_COPY) {

            cursor = DragSource.DefaultCopyDrop;
        }

        event.startDrag(cursor, this);
    }

    public Object getTransferData(DataFlavor flavor) {

        return null;
```

```
    }

    public DataFlavor[] getTransferDataFlavors() {

        return new DataFlavor[0];
    }

    public boolean isDataFlavorSupported(DataFlavor flavor) {

        return false;
    }

    private void createLayout(JComponent... arg) {

        Container pane = getContentPane();
        GroupLayout gl = new GroupLayout(pane);
        pane.setLayout(gl);

        gl.setAutoCreateContainerGaps(true);
        gl.setAutoCreateGaps(true);

        gl.setHorizontalGroup(gl.createSequentialGroup()
                .addGap(50)
                .addComponent(arg[0])
                .addGap(50)
        );

        gl.setVerticalGroup(gl.createSequentialGroup()
                .addGap(50)
                .addComponent(arg[0])
                .addGap(50)
        );

        pack();
    }

    public static void main(String[] args) {
```

```
        EventQueue.invokeLater(() -> {

            DragGesture ex = new DragGesture();
            ex.setVisible(true);
        });
    }
}
```

This simple example demonstrates a drag gesture. The drag gesture is created when we click on a component and move a mouse pointer, while the button is pressed. The example shows how we can create a DragSource for a component.

```
public class DragGesture extends JFrame implements
   DragGestureListener, Transferable {
```

DragGesture implements two interfaces. DragGestureListener listens for drag gestures. Transferable handles data for a transfer operation. In the example, we will not transfer any data. We will only demonstrate a drag gesture. So the three necessary methods of the Transferable interface are left unimplemented.

```
DragSource ds = new DragSource();

ds.createDefaultDragGestureRecognizer(redPanel,
        DnDConstants.ACTION_COPY, this);
```

Here we create a DragSource object and register it for the panel. The DragSource is the entity responsible for the initiation of the drag and drop operation. The createDefaultDragGestureRecognizer() associates a drag source and DragGestureListener with a particular component.

```
public void dragGestureRecognized(DragGestureEvent event) {

}
```

The dragGestureRecognized() method responds to a drag gesture.

```
Cursor cursor = null;

if (event.getDragAction() == DnDConstants.ACTION_COPY) {
    cursor = DragSource.DefaultCopyDrop;
}

event.startDrag(cursor, this);
```

The startDrag() method of the DragGestureEvent finally starts the drag operation. We specify two parameters: the cursor type and the Transferable object.

```
public Object getTransferData(DataFlavor flavor) {

    return null;
}

public DataFlavor[] getTransferDataFlavors() {

    return new DataFlavor[0];
}

public boolean isDataFlavorSupported(DataFlavor flavor) {

    return false;
}
```

The object that implements the Transferable interface must implement these three methods. There is not functionality yet.

A complex drag and drop example

In the following example, we create a complex drag and drop example. We create a drag source a drop target and a transferable object.

ComplexDnD.java

```
package com.zetcode;

import javax.swing.GroupLayout;
import javax.swing.JButton;
import javax.swing.JColorChooser;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Container;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.datatransfer.DataFlavor;
import java.awt.datatransfer.Transferable;
import java.awt.datatransfer.UnsupportedFlavorException;
import java.awt.dnd.DnDConstants;
import java.awt.dnd.DragGestureEvent;
import java.awt.dnd.DragGestureListener;
import java.awt.dnd.DragSource;
import java.awt.dnd.DropTarget;
import java.awt.dnd.DropTargetAdapter;
import java.awt.dnd.DropTargetDropEvent;

/*
```

```
Java Swing tutorial
Complex drag and drop example

Author: Jan Bodnar
Website: http://zetcode.com
 */

public class ComplexDnD extends JFrame
        implements DragGestureListener {

    private JPanel leftPanel;

    public ComplexDnD() {

        initUI();
    }

    private void initUI() {

        JButton colourBtn = new JButton("Choose Color");
        colourBtn.setFocusable(false);

        leftPanel = new JPanel();
        leftPanel.setBackground(Color.red);
        leftPanel.setPreferredSize(new Dimension(100, 100));

        colourBtn.addActionListener(event -> {

            Color color = JColorChooser.showDialog(this, "Choose Color", Color.white);
            leftPanel.setBackground(color);
        });

        JPanel rightPanel = new JPanel();
        rightPanel.setBackground(Color.white);
        rightPanel.setPreferredSize(new Dimension(100, 100));

        MyDropTargetListener mtl = new MyDropTargetListener(rightPanel);
```

```
        DragSource ds = new DragSource();
        ds.createDefaultDragGestureRecognizer(leftPanel,
                DnDConstants.ACTION_COPY, this);

        createLayout(colourBtn, leftPanel, rightPanel);

        setTitle("Complex drag and drop example");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }

    public void dragGestureRecognized(DragGestureEvent event) {

        Cursor cursor = null;
        JPanel panel = (JPanel) event.getComponent();

        Color color = panel.getBackground();

        if (event.getDragAction() == DnDConstants.ACTION_COPY) {
            cursor = DragSource.DefaultCopyDrop;
        }

        event.startDrag(cursor, new TransferableColor(color));
    }

    private class MyDropTargetListener extends DropTargetAdapter {

        private final DropTarget dropTarget;
        private final JPanel panel;

        public MyDropTargetListener(JPanel panel) {
            this.panel = panel;

            dropTarget = new DropTarget(panel, DnDConstants.ACTION_COPY,
                    this, true, null);
        }
```

```
    public void drop(DropTargetDropEvent event) {

        try {

            Transferable tr = event.getTransferable();
            Color col = (Color) tr.getTransferData(TransferableColor.colorFlavor);

            if (event.isDataFlavorSupported(TransferableColor.colorFlavor)) {

                event.acceptDrop(DnDConstants.ACTION_COPY);
                this.panel.setBackground(col);
                event.dropComplete(true);
                return;
            }

            event.rejectDrop();
        } catch (Exception e) {

            e.printStackTrace();
            event.rejectDrop();
        }
    }
}

private void createLayout(JComponent... arg) {

    Container pane = getContentPane();
    GroupLayout gl = new GroupLayout(pane);
    pane.setLayout(gl);

    gl.setAutoCreateContainerGaps(true);
    gl.setAutoCreateGaps(true);

    gl.setHorizontalGroup(gl.createSequentialGroup()
            .addComponent(arg[0])
```

```
                .addGap(30)
                .addComponent(arg[1])
                .addGap(30)
                .addComponent(arg[2])
        );

        gl.setVerticalGroup(gl.createParallelGroup()
                .addComponent(arg[0])
                .addComponent(arg[1])
                .addComponent(arg[2])
        );

        pack();
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

            ComplexDnD ex = new ComplexDnD();
            ex.setVisible(true);
        });
    }
}


class TransferableColor implements Transferable {

    protected static final DataFlavor colorFlavor =
            new DataFlavor(Color.class, "A Color Object");

    protected static final DataFlavor[] supportedFlavors = {
            colorFlavor,
            DataFlavor.stringFlavor,
    };

    private final Color color;
```

```
    public TransferableColor(Color color) {

        this.color = color;
    }

    public DataFlavor[] getTransferDataFlavors() {

        return supportedFlavors;
    }

    public boolean isDataFlavorSupported(DataFlavor flavor) {

        return flavor.equals(colorFlavor) ||
                flavor.equals(DataFlavor.stringFlavor);
    }


    public Object getTransferData(DataFlavor flavor)
            throws UnsupportedFlavorException {

        if (flavor.equals(colorFlavor)) {
            return color;
        } else if (flavor.equals(DataFlavor.stringFlavor)) {
            return color.toString();
        } else {
            throw new UnsupportedFlavorException(flavor);
        }
    }
}
```

The code example shows a button and two panels. The button displays a colour chooser dialog and sets a colour for the first panel. The colour can be dragged into the second panel.

This example enhances the previous one. We will add a drop target and a custom transferable object.

```
MyDropTargetListener mtl = new MyDropTargetListener(rightPanel);
```

We register a drop target listener with the right panel.

```
event.startDrag(cursor, new TransferableColor(color));
```

The startDrag() method has two parameters. A cursor and a Transferable object.

```
public MyDropTargetListener(JPanel panel) {
    this.panel = panel;

    dropTarget = new DropTarget(panel, DnDConstants.ACTION_COPY,
        this, true, null);
}
```

In the MyDropTargetListener we create a drop target object.

```
Transferable tr = event.getTransferable();
Color color = (Color) tr.getTransferData(TransferableColor.colorFlavor);

if (event.isDataFlavorSupported(TransferableColor.colorFlavor)) {

    event.acceptDrop(DnDConstants.ACTION_COPY);
    this.panel.setBackground(color);
    event.dropComplete(true);
    return;
}
```

We get the data being transferred. In our case it is a colour object. Here we set the colour of the right panel.

```
event.rejectDrop();
```

If the conditions for a drag and drop operation are not fulfilled, we reject it.

```
protected static DataFlavor colorFlavor =
    new DataFlavor(Color.class, "A Color Object");
```

In the TransferableColor, we create a new DataFlavor object.

```
protected static DataFlavor[] supportedFlavors = {
    colorFlavor,
    DataFlavor.stringFlavor,
};
```

Here we specify, what data flavors we support. In our case it is a custom defined colour flavor and a predefined DataFlavor.stringFlavor.

```
public Object getTransferData(DataFlavor flavor)
        throws UnsupportedFlavorException {

    if (flavor.equals(colorFlavor)) {
        return color;
    } else if (flavor.equals(DataFlavor.stringFlavor)) {
        return color.toString();
    } else {
        throw new UnsupportedFlavorException(flavor);
    }
}
```

The getTransferData() return an object for a specific data flavour.

This part of the Java Swing tutorial was dedicated to Swing drap and drop operations.