

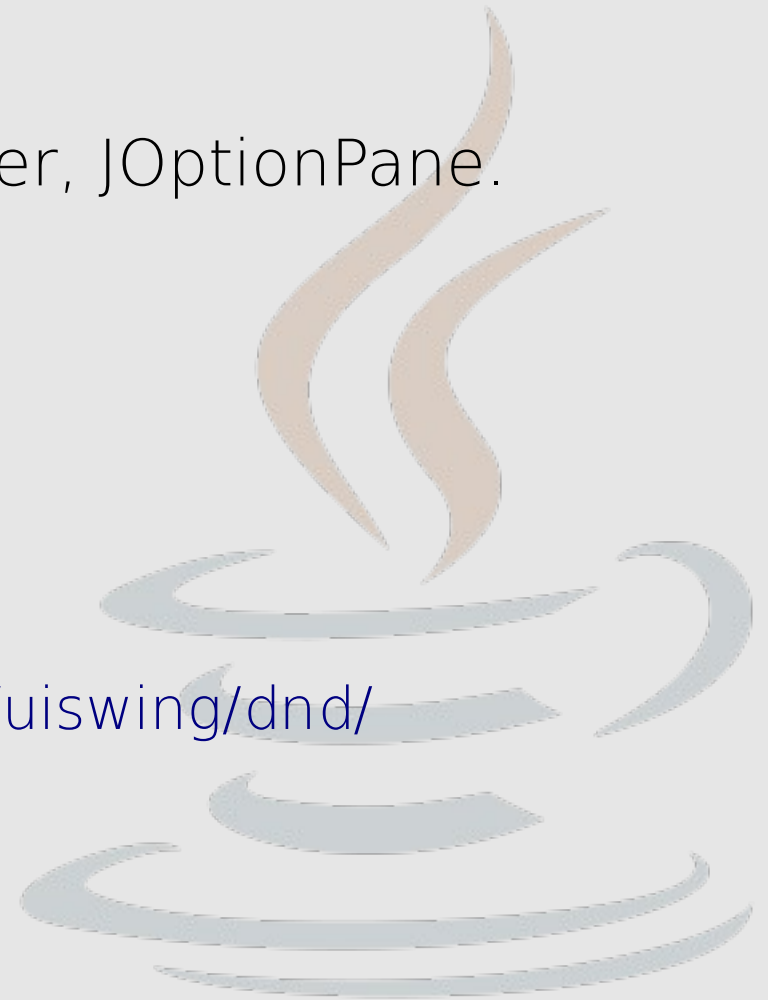
SWING c.d.

ZAGADNIENIA:

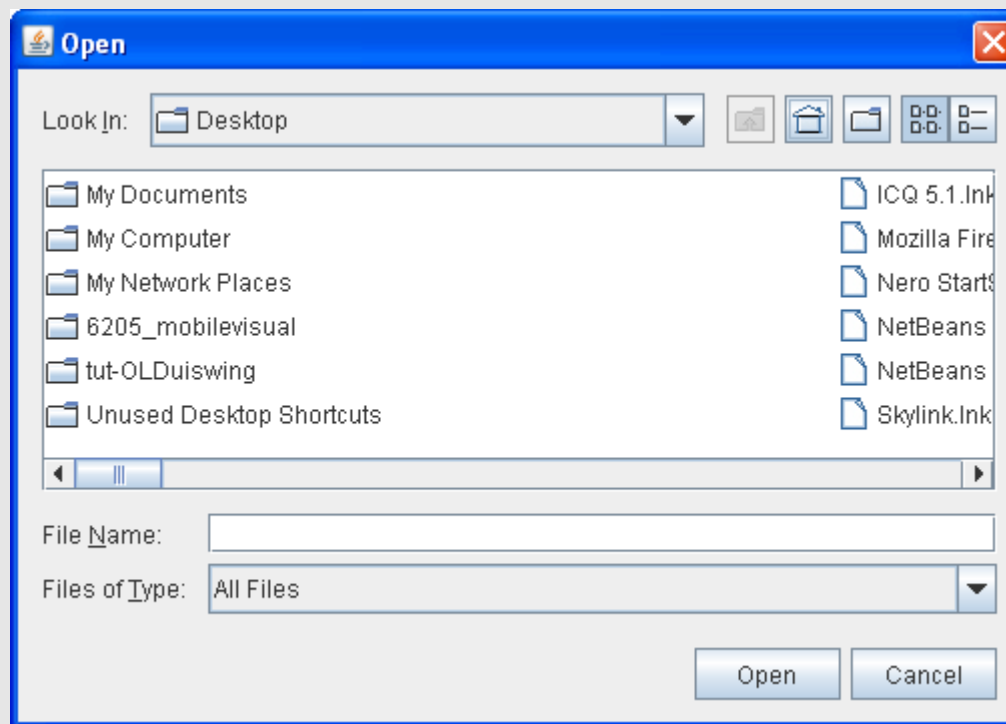
- przydatne narzędzia: JFileChooser, JOptionPane.
- drag'n drop,
- menu kontekstowe.

MATERIAŁY:

<http://docs.oracle.com/javase/tutorial/uiswing/dnd/>



JFileChooser



<http://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html>

JFileChooser

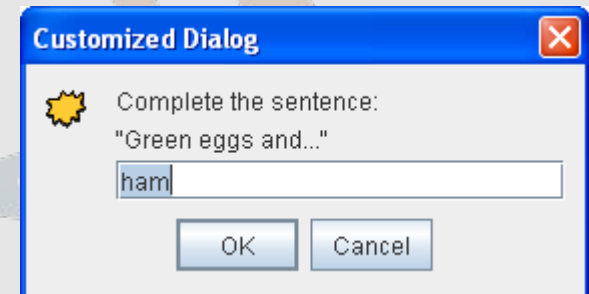
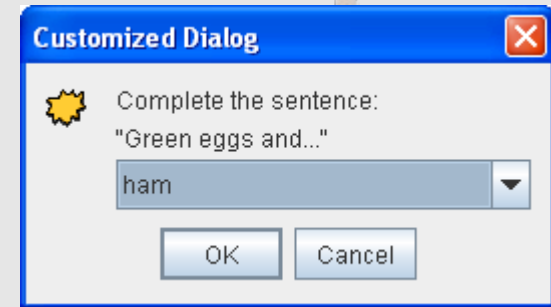
```
JFileChooser chooser = new JFileChooser();
// filtr plików
FileNameExtensionFilter filter = new FileNameExtensionFilter(
    "Obrazy JPG i GIF", "jpg", "gif");
chooser.setFileFilter(filter);
// wyświetlenie okienka
int ret = chooser.showOpenDialog(parent);
// Jesli wcisnieto ok lub open
if(ret == JFileChooser.APPROVE_OPTION) {
    System.out.println("Wybrales plik: " +
        chooser.getSelectedFile().getName());
}
```

JOptionPane

 <p>Message Eggs aren't supposed to be green. OK</p>	<pre>//default title and icon JOptionPane.showMessageDialog(frame, "Eggs are not supposed to be green.");</pre>
 <p>Inane warning Eggs aren't supposed to be green. OK</p>	<pre>//custom title, warning icon JOptionPane.showMessageDialog(frame, "Eggs are not supposed to be green.", "Inane warning", JOptionPane.WARNING_MESSAGE);</pre>
 <p>Inane error Eggs aren't supposed to be green. OK</p>	<pre>//custom title, error icon JOptionPane.showMessageDialog(frame, "Eggs are not supposed to be green.", "Inane error", JOptionPane.ERROR_MESSAGE);</pre>
 <p>A plain message Eggs aren't supposed to be green. OK</p>	<pre>//custom title, no icon JOptionPane.showMessageDialog(frame, "Eggs are not supposed to be green.", "A plain message", JOptionPane.PLAIN_MESSAGE);</pre>
 <p>Inane custom dialog Eggs aren't supposed to be green. OK</p>	<pre>//custom title, custom icon JOptionPane.showMessageDialog(frame, "Eggs are not supposed to be green.", "Inane custom dialog", JOptionPane.INFORMATION_MESSAGE, icon);</pre>

JOptionPane

```
Object[] possibilities = {"ham", "spam", "yam"};  
String s = (String)JOptionPane.showInputDialog(  
    frame,  
    "Complete the sentence:\n"  
    + "\"Green eggs and...\"",  
    "Customized Dialog",  
    JOptionPane.PLAIN_MESSAGE,  
    icon,  
    possibilities, // null  
    "ham");
```



<http://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html#features>

Drag and Drop

Wiele komponentów standardowo obsługuje drag'n drop (a dokładniej drop). Jeśli źródłem danych ma być komponent Swing, to należy na nim wykonać metodę: **setDragEnabled(true)**.

Drag'n Drop funkcjonalnie jest równoważne kopiowaniu przez schowek (**ctrl-c**, **ctrl-v**). Aby całkowicie kontrolować ten proces należy poznać kilka klas odpowiedzialnych za jego realizację.

<http://docs.oracle.com/javase/tutorial/uiswing/dnd/index.html>

Transferable

Przenoszone dane są reprezentowane przez interfejs **Transferable**.

- **Object** `getTransferData(DataFlavor flavor)` - zwraca transferowany obiekt
- **DataFlavor[]** `getTransferDataFlavors()` - zwraca wszystkie dostępne postacie przenoszonego obiektu,
- **boolean** `isDataFlavorSupported(DataFlavor flavor)` - zwraca informację, czy obiekt jest dostępny w odpowiedniej postaci

DataFlavor

Przenoszony obiekt jest zwykle dostępny w wielu postaciach. Przykładowo, zdjęcie może być reprezentowane przez zbiór pixeli (grafika), nazwę pliku, w którym jest zapisane, zbiór bajtów, czy też zakodowaną w formacie **Base64** zawartość.

W związku z tym, w zależności od tego, gdzie takie zdjęcie przenosimy (wklejamy) możemy zobaczyć je pod inną postacią. Te postacie są reprezentowane przez klasę **DataFlavor**.

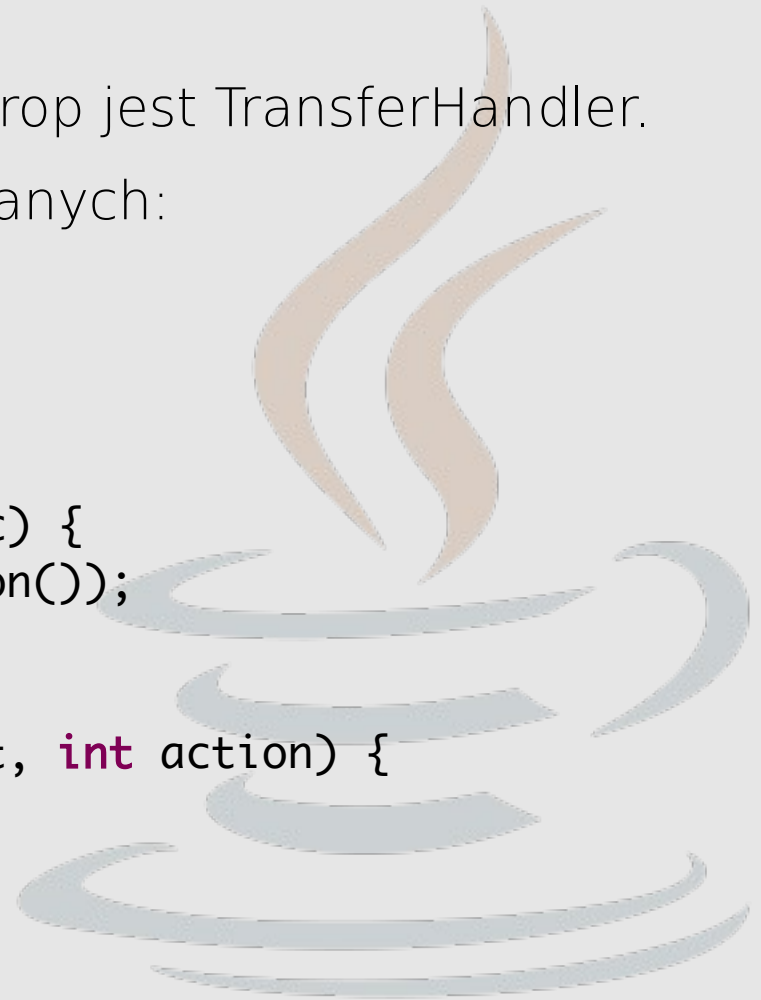
TransferHandler

Obiekt, który zarządza procesem drag'n drop jest TransferHandler. Posiada on metody służące do eksportu danych:

```
int getSourceActions(JComponent c) {  
    return COPY_OR_MOVE;  
}
```

```
Transferable createTransferable(JComponent c) {  
    return new StringSelection(c.getSelection());  
}
```

```
void exportDone(JComponent c, Transferable t, int action) {  
    if (action == MOVE) {  
        c.removeSelection();  
    }  
}
```



TransferHandler

oraz importu:

`canImport(TransferHandler.TransferSupport)` — zwraca true jeśli komponent znajdujący się pod kursorem myszki akceptuje przenoszony obiekt

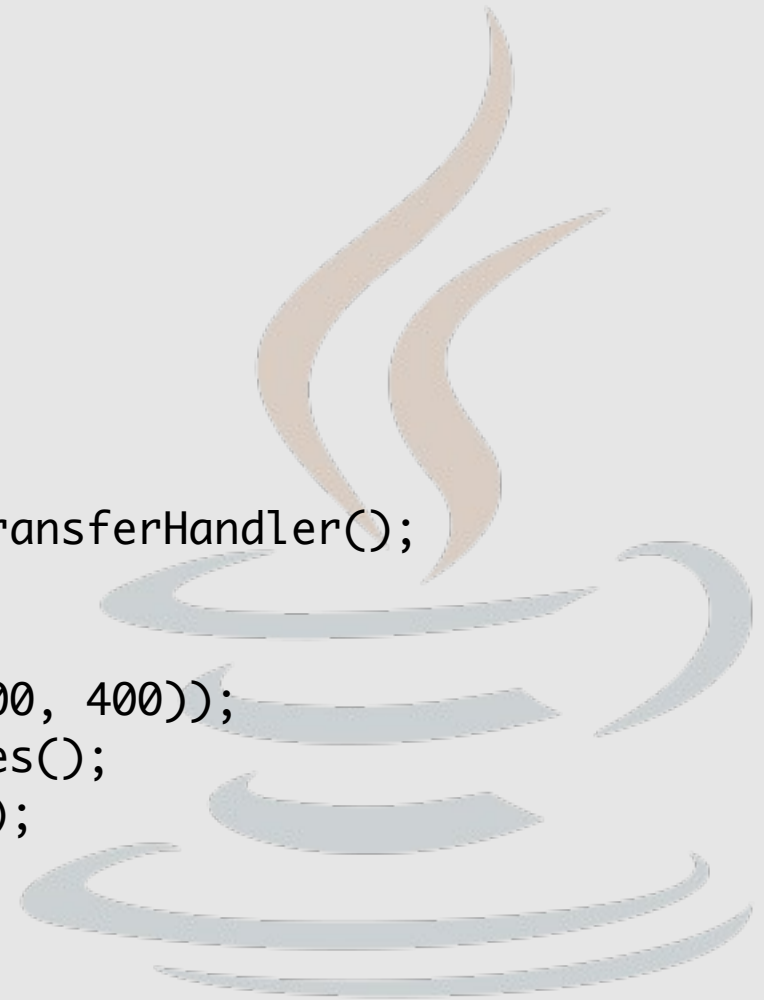
`importData(TransferHandler.TransferSupport)` — metoda jest wywoływana po upuszczeniu (drop) obiektu. zwraca true jeśli import obiektu zakończył się powodzeniem.

PRZYKŁAD

```
import java.awt.*;
import java.awt.datatransfer.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

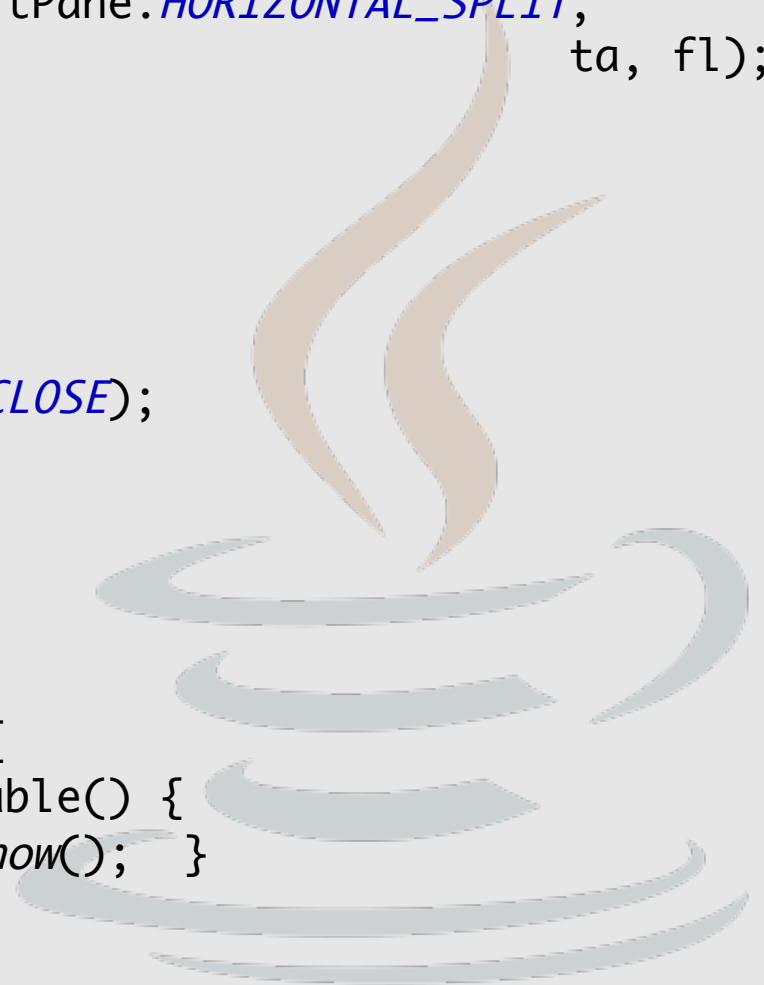
public class JDnDFrame extends JFrame {

    public JDnDFrame() {
        FileTransferHandler fth = new FileTransferHandler();
        JTextArea ta = new JTextArea();
        ta.setTransferHandler(fth);
        ta.setPreferredSize(new Dimension(600, 400));
        File[] fa = (new File(".")).listFiles();
        JList<File> fl = new JList<File>(fa);
        fl.setTransferHandler(fth);
        fl.setDragEnabled(true);
    }
}
```



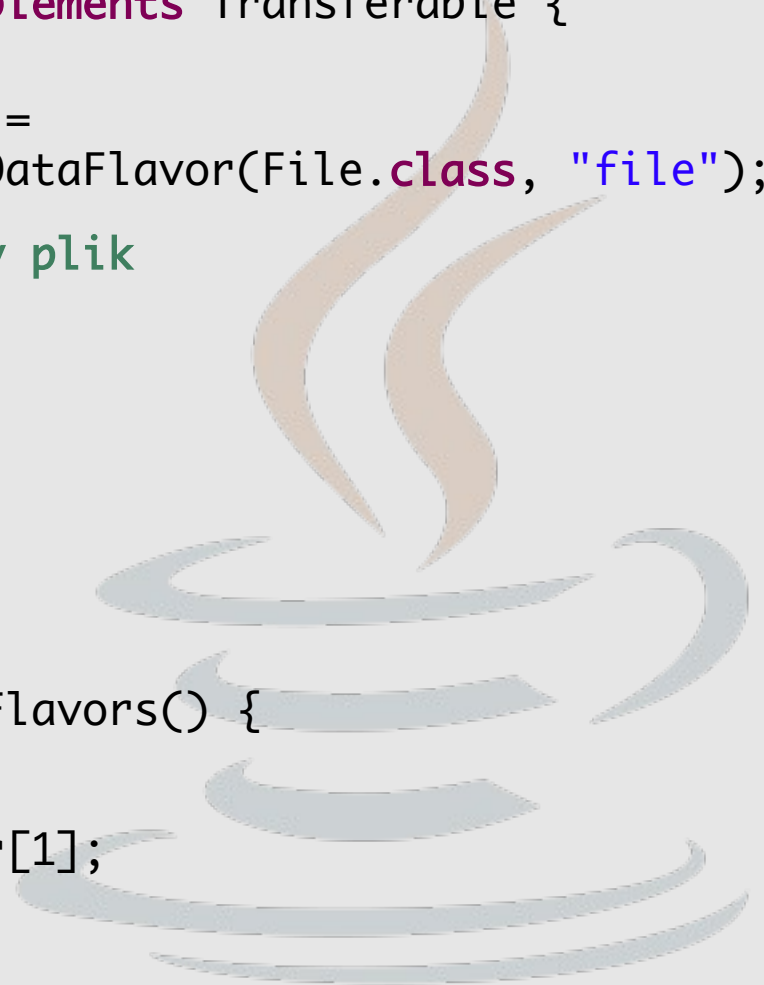
PRZYKŁAD

```
JSplitPane sp = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,  
                                ta, fl);  
this.getContentPane().add(sp);  
}  
  
public static void createAndShow() {  
    JDnDFrame f = new JDnDFrame();  
    f.setDefaultCloseOperation(EXIT_ON_CLOSE);  
    f.pack();  
    f.setLocationRelativeTo(null);  
    f.setVisible(true);  
}  
  
public static void main(String[] args) {  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() { createAndShow(); }  
    });  
}
```

The Java logo is positioned on the right side of the slide, partially overlapping the code. It consists of a stylized coffee cup with steam rising from it, rendered in a light blue and grey color scheme.

PRZYKŁAD

```
private static class FileTransferable implements Transferable {  
  
    public static DataFlavor fileFlavor =  
        new DataFlavor(File.class, "file");  
  
    // tutaj przechowujemy transferowany plik  
    private File file;  
  
    public FileTransferable(File f) {  
        this.file = f;  
    }  
  
    @Override  
    public DataFlavor[] getTransferDataFlavors() {  
        // tylko jedna postać  
        DataFlavor[] df = new DataFlavor[1];  
        df[0] = fileFlavor;  
        return df;  
    }  
}
```



PRZYKŁAD

```
@Override
public boolean isDataFlavorSupported(DataFlavor flavor) {
    return (flavor == fileFlavor);
}
```

```
@Override
public Object getTransferData(DataFlavor flavor)
    throws UnsupportedOperationException, IOException {
    if (isDataFlavorSupported(flavor))
        return this.file;

    // nie obsługujemy innych postaci obiektu
    throw new UnsupportedOperationException(flavor);
}
}
```

PRZYKŁAD

```
private static class FileTransferHandler extends TransferHandler
                                         implements ActionListener{

    private JPopupMenu popup;

    // bez tej metody nie rozpocznie się eksport obiektu
    public int getSourceActions(JComponent c) {
        return COPY_OR_MOVE;
    }

    // tworzy obiekt Transferable zawierający przenoszony element
    protected Transferable createTransferable(JComponent c) {
        Object obj = ((JList<File>) c).getSelectedValue();
        return new FileTransferable((File) obj);
    }

    // handler akceptuje wszystko. W praktyce to powinno być
    // zaimplementowane porządnie, ale tutaj takie zachowanie nie
    // będzie przeszkadzać
    public boolean canImport(TransferSupport ts) {
        return true;
    }
}
```

PRZYKŁAD

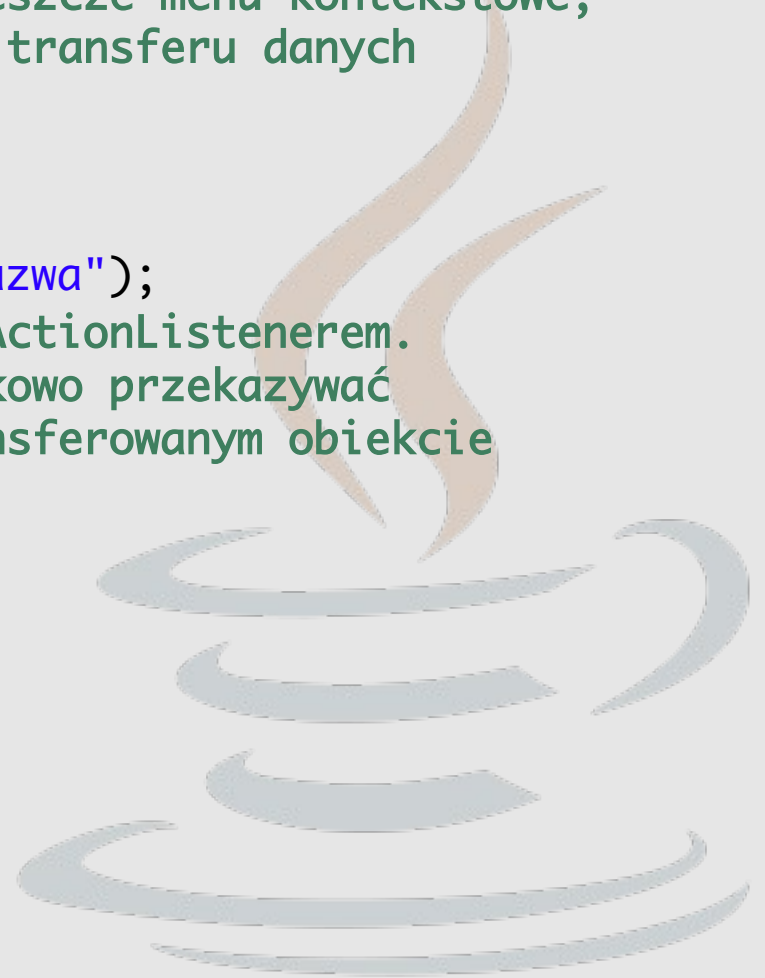
```
private JTextArea destination;
private File file;

public boolean importData(TransferSupport ts) {
    // tutaj niepotrzebne, ale ogólnie powinno być – żeby nie
    // importować nieobsługiwanych obiektów
    if (!canImport(ts))
        return false;
    try {
        this.file = (File)
            ts.getTransferable().getTransferData(
                FileTransferable.fileFlavor);
        this.destination = (JTextArea) ts.getComponent();
        Point p = ts.getDropLocation().getDropPoint();
        this.popup.show(ts.getComponent(), p.x, p.y);
        return true;
    } catch (UnsupportedFlavorException | IOException e) { }
    return false;
}
```


PRZYKŁAD

// Konstruktor, w którym tworzymy jeszcze menu kontekstowe,
// które będzie nam określać sposób transferu danych

```
public FileTransferHandler(){
    super();
    popup = new JPopupMenu();
    JMenuItem mi = new JMenuItem("nazwa");
    // TransferHandler jest również ActionListenerem.
    // Dzięki temu nie trzeba dodatkowo przekazywać
    // listenerowi informacji o transferowanym obiekcie
    mi.addActionListener(this);
    mi.setActionCommand("name");
    popup.add(mi);
    mi = new JMenuItem("zawartosc");
    mi.addActionListener(this);
    mi.setActionCommand("content");
    popup.add(mi);
}
```



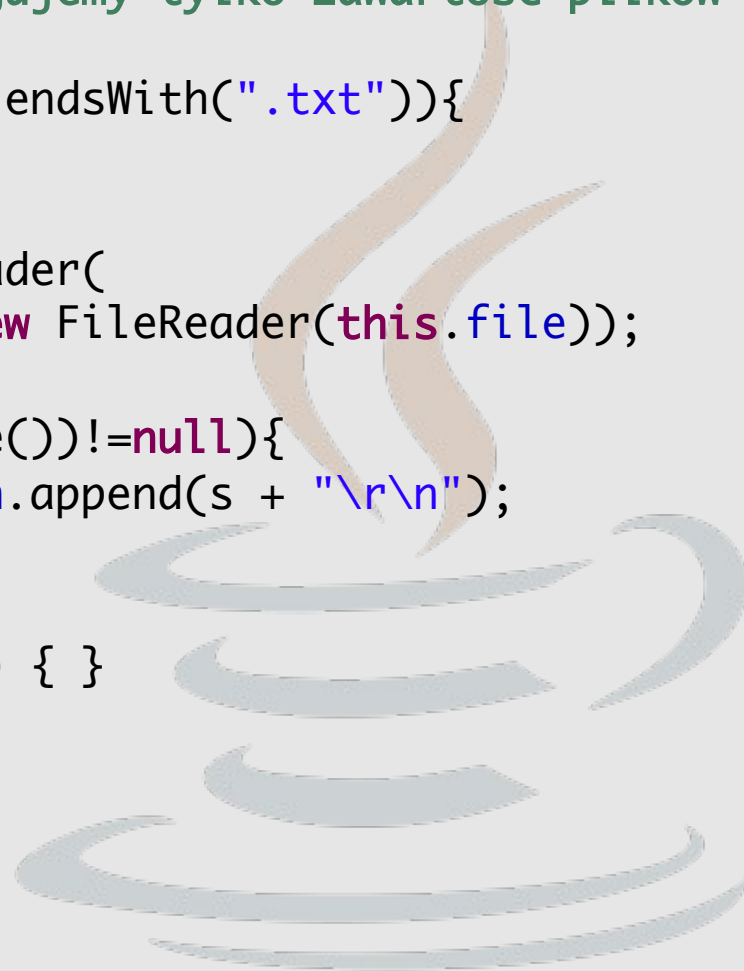
PRZYKŁAD

```
@Override
public void actionPerformed(ActionEvent e) {
    // przenosimy plik pod jako nazwe
    if (e.getActionCommand().equals("name")){
        // dodajemy ją na koncu tekstu, ale możnaby też sprawdzić
        // pozycje kursora i umieścić ten tekst w określonej
        // pozycji
        this.destination.append(this.file.getName() + "\r\n");
    }else if (e.getActionCommand().equals("content")){
        // zawartosc katalogu to nazwy plików, które zawiera
        if (this.file.isDirectory()) {
            for(String s : this.file.list())
                this.destination.append(s + "\r\n");
        }
    }
}
```

PRZYKŁAD

```
// w przypadku plików obsługujemy tylko zawartosc plikow
// tekstowych
else if(this.file.getName().endsWith(".txt")){
    BufferedReader br;
    try {
        br = new BufferedReader(
                new FileReader(this.file));

        String s;
        while((s=br.readLine())!=null){
            this.destination.append(s + "\r\n");
        }
        br.close();
    } catch (IOException e1) { }
}
}
}
}
```



DZIĘKUJĘ ZA UWAGĘ