Other Components

JEditorPane

- Used to display text in a graphical component
- Suited for displaying text read from files
- `JEditorPane` supplies the basic functions of a text editor automatically.
  - You can place the cursor, insert or delete text and also copy (using CNTR-C and CNTRV).

- `JEditorPane()`
- `void read(Reader myReader, Object description)`
  - reads the text supplied by myReader and displays it in the editor area.
  - In the second argument description further information on the type of text can be given; we do not use this here, i.e. we set description = null.
- `getText()`
  - returns the text currently displayed in the editor area as one string, including the line-end characters.

JEditorPane

- `setText(String text)`
  - Sets the text content of the editor pane.
- `getText(String text)`
  - Gets the text content of the editor pane.
- `setContentType(String type)`
  - Sets the content type of the editor pane (e.g., "text/plain," "text/html," "text/rtf").
- `getContentType()`
  - Returns the content type of the editor pane.
- `setEditable(boolean editable)`
  - Sets whether the editor pane is editable or not.
- `getSelectedText()`
  - Returns the currently selected text.

JEditorPane example

```
import java.awt.*;
import javax.swing.*;

public class JEditorPaneSetFontExample {
    public static void main(String[] args)
    {
           // Create a JFrame
            JFrame frame = new JFrame("JEditorPane setFont() Example");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(400, 300);

            // Create a JEditorPane
            JEditorPane editorPane = new JEditorPane();
            // Set content type to plain text
            editorPane.setContentType("text/plain");

            // Set text content
            editorPane.setText("Test For EditorPane");

            // Create a custom font
            Font customFont  = new Font("Serif", Font.BOLD, 16);

            // Set the font for the editor pane
            editorPane.setFont(customFont);

            // Add the JEditorPane to the frame
            frame.add(editorPane);

            // Make the frame visible
            frame.setVisible(true);
      }
}
```

JEditorPane example 2

```
import SimpleFrame.SimpleFrame;

import java.io.*;
import javax.swing.*;
import java.awt.*;

public class TextDisplayFrame extends SimpleFrame
{
  private JEditorPane textDisplayPane;

  public TextDisplayFrame(String filename)
  {
   textDisplayPane = new JEditorPane();
   this.getContentPane().add(textDisplayPane,BorderLayout.CENTER);
   this.setSize(200,160);

   File readfile  = new File(filename);

   try{
    FileReader fr = new FileReader(readfile);
    textDisplayPane.read(fr,null);
   }catch(IOException e){
    System.out.println("Problems opening or reading "+readfile.getPath());
   }

  }
}
```

```
public class TextDisplayDriver
{
  // Adujust paths if necessary!!
  private static String path = "./its/TestData/";
  private static String fileName = "testtext1.txt";

  public static void main(String[] args)
  {
    TextDisplayFrame TAF = new TextDisplayFrame(path+fileName);
   TAF.showIt("Text Display");
  }
}
```

JEditorPane example 3

```
import java.io.*;
import java.awt.event.*;
import javax.swing.*;
public class JEditorPaneSave implements ActionListener {
   JFrame myFrame = null;
   JEditorPane myPane = null;
   public static void main(String[] a) {
      (new JEditorPaneSave()).test();
   }
   private void test() {
      myFrame = new JFrame("JEditorPane Save Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setSize(300,200);

      myPane = new JEditorPane();
      myPane.setContentType("text/plain");
      myPane.setText(
         "JEditorPane is a text component to edit various kinds of"
         +" content.\n\nThis component uses implementations of the"
         +" EditorKit to accomplish its behavior.");
      myFrame.setContentPane(myPane);

      JMenuBar myBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myBar.add(myMenu);
      myFrame.setJMenuBar(myBar);
      myFrame.setVisible(true);
   }
   private JMenu getFileMenu() {
      JMenu myMenu = new JMenu("File");
      JMenuItem myItem = new JMenuItem("Open");
      myItem.addActionListener(this);
      myMenu.add(myItem);

      myItem = new JMenuItem("Save");
      myItem.addActionListener(this);
      myMenu.add(myItem);
      return myMenu;
   }
```

```
   public void actionPerformed(ActionEvent e) {
        String cmd = e.getActionCommand();
        try {
           if (cmd.equals("Open")) {
              FileReader in = new FileReader("JEditorPane.txt");
              char[] buffer = new char[1024];
              int n = in.read(buffer);
              String text = new String(buffer, 0, n);
              myPane.setText(text);
              in.close();
           } else if (cmd.equals("Save")) {
              FileWriter out = new FileWriter("JEditorPane.txt");
              out.write(myPane.getText());
              out.close();
           }
        } catch (Exception f) {
             f.printStackTrace();
        }
    }
```

Scrolling

- To display long texts or large drawings that do not fit into the window.
- Scrolling is implemented by class `JScrollPane`
- The scroll bars are on the right and at the bottom by default
- The part where the component is displayed is called *`viewport`*.
  - If the whole component fits into the viewport the scroll bars disappear.
- `setHorizontalScrollBarPolicy(int policy)` determines when the scroll bars are visible.
  - Class `JScrollPane` defines constant values for policy
    - `HORIZONTAL_SCROLLBAR_ALWAYS, HORIZONTAL_SCROLLBAR_AS_NEEDED and HORIZONTAL_SCROLLBAR_NEVER`
- `setVerticalScrollBarPolicy(int policy)` determines when the scroll bars are visible.
  - Policy has similar values to the ones for the horizontal scrollbar.

JScrollPane with text example

```
import SimpleFrame.SimpleFrame;
import java.io.*;
import java.awt.*;
import javax.swing.*;
public class TextDisplayScrollFrame extends SimpleFrame
{
  private JEditorPane TextDisplayPanel;

  public TextDisplayScrollFrame(String filename)
  {

   TextDisplayPanel = new JEditorPane();

   JScrollPane scrollPane  = new JScrollPane(TextDisplayPanel);
   this.getContentPane().add(scrollPane,BorderLayout.CENTER);

   File readfile  = new File(filename);

   try{
    FileReader fr = new FileReader(readfile);
    TextDisplayPanel.read(fr,null);
   }catch(IOException e){
    System.out.println("Problems openeing or reading "+readfile.getName());
   }
 }
}
```

```
public class TextDisplayScrollDriver
{

 // This variable has to be set according to your system
 private static String fileName = "testtext3.txt";

  public static void main(String[] args)
  {
    TextDisplayScrollFrame TAF = new TextDisplayScrollFrame(fileName);
    TAF.showIt("Text with scrolling");
  }
}
```

JScrollPane with drawing example

```
import SimpleFrame.SimpleFrame;
import javax.swing.*;
public class DrawingDisplayScrollFrame extends SimpleFrame {

  public DrawingDisplayScrollFrame() {
    DrawingDisplayScrollPanel drawPane = new DrawingDisplayScrollPanel();
    JScrollPane scrollPane = new JScrollPane(drawPane);
    this.getContentPane().add(scrollPane);
    this.setSize(340,250);
   }
}
```

```
import java.awt.*;
import javax.swing.*;

public class DrawingDisplayScrollPanel extends JPanel {

  public DrawingDisplayScrollPanel() {
    this.setBackground(Color.white);
    this.setPreferredSize(new Dimension(250,250));
   }

  public void paintComponent(Graphics g){
    super.paintComponent(g);
    Color oldColor = g.getColor();
    g.setColor(Color.red);
    g.drawRect(0,0,249,249);
    g.drawString("Border of preferred size.",10,240);
    g.setColor(Color.blue);
    g.fillOval(300,150,20,20);
    g.drawString("This is outside the preffered size",260,180);
    g.setColor(oldColor);
  }
}
```

```
public class DrawingDisplayScrollDriver {
  public static void main(String[] args) {
    DrawingDisplayScrollFrame ddsf =
            new DrawingDisplayScrollFrame();
    ddsf.showIt("Drawing with Scrolling",200,200);
  }
}
```

File selection dialogues

- File selection dialogues allow the user to select a file from the computer's mass storage (hard disk).
  - The file selected by the user is returned to the class that started the dialogue.
- `JFileChooser`: The navigation in `JFileChooser` is (mostly) done as in the file selection dialogues of the operating system.
  - One can change directories, to choose a file from a list, or by typing its name into a text field.
  - There are buttons labelled 'Open' and 'Cancel'.
    - We do not have to implement any listener for these buttons. All the functions one expects from a file selection dialogue are supplied by `JFileChooser`

JFileChooser

- `JFileChooser(String startDirectory);`
  - generates a file selection dialogue which is not yet visible. When it becomes visible the files in startDirectory are listed, provided that this string specifies an existing directory.
- `int showOpenDialog(Component parent);`
  - displays the file selection dialogue for opening files.
  - component parent is usually the component in which the showOpenDialog was called.
  - On closing, a file selection dialogue returns an integer value which is one of the following integer constants:
  - APPROVE_OPTION indicates that the ‘Open’ (or ‘Save’, in case of a save dialogue) button of the dialogue has been clicked.
  - CANCEL_OPTION indicates that the ‘Cancel’ button of the dialogue has been clicked.
- `int showSaveDialog(Component parent);`
  - displays the file selection dialogue for saving files; the rest is as for showOpenDialog
- `File getSelectedFile();`
  - returns the selected file in a variable of type File if APPROVE_OPTION has been selected.

- Only after the dialogue is closed can the parent component be used again.
  - This is to avoid unwanted interactions such as modifying a file in the parent frame while it is being saved.

User-defined dialogues

- Create your own dialogue class by extending the Swing class `JDialog`.
- A `JDialog` is much like a `Jframe`
- A `JDialog` has a content pane into which further components are embedded.

- `JDialog(Frame parent, String title, boolean modal)`
- `getContentPane.add(Component comp)`
- `setVisible(boolean b)`
- `pack()`