320341 Programming in Java

Fall Semester 2014

Lecture 15: More GUI Components

Instructor: Jürgen Schönwälder

Slides: Bendick Mahleko

Objectives

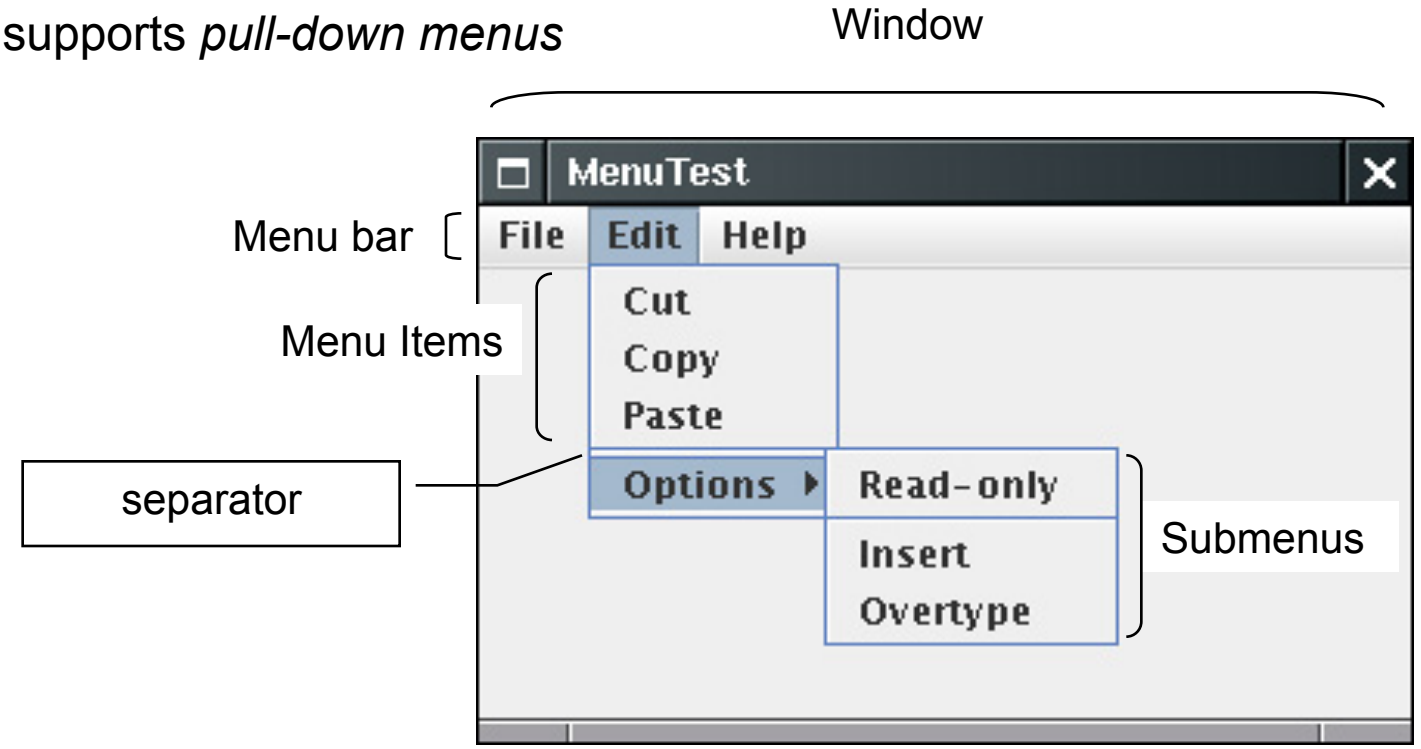
This lecture presents the following

- Menus
- Advanced Layout Managers
- Dialogs

Menus

Pull Down Menus

- Swings supports *pull-down menus*



- The **Menu bar** – contains names of *pull-down menus*
- Clicking on **menu item** sends a message to the handler for that menu item

Menus

Menus can be attached only to objects of the classes that provide the method ***setJMenuBar***

- Two such classes are **JFrame** and **JApplet**

- Classes used to declare menus are:
  - **JMenuBar**
  - **JMenu**
  - **JMenuItem**
  - **JCheckBoxMenuItem &**
  - **JRadioButtonMenuItem**

Menus

Building Pull-down Menus

- Create a menu bar

```
JMenuBar menuBar = new JMenuBar();
```

- A menu bar can appear anywhere, but normally you add it at the top of a frame

```
frame.setMenuBar(menuBar);
```

- For each menu, create a menu object

```
JMenu editMenu = new JMenu("Edit");
```

- Add top level menus to the menu bar

```
menuBar.add(editMenu);
```

Menus

Building Pull-down Menus

- Add *menu items*, *separators* and *submenus* to the menu object

```
JMenuItem pasteItem = new JMenuItem("Paste");
editMenu.add(pasteItem);
editMenu.addSeparator();

JMenu optionsMenu = . . .;
// a submenu
editMenu.add(optionsMenu);
```

Menus

Install an *action listener* for each menu item

```
ActionListener listener = . . .;
pasteItem.addActionListener(listener);
```

- The method **JMenu**.*add*(**String** s) adds a menu item to the end of a menu
- The method returns the created menu item, thus add listener as follows

```
JMenuItem pasteItem = editMenu.add("Paste");
pasteItem.addActionListener(listener);
```

Menus

Installing a listener using an **Action** name

- Convenient for activating menu items using other UI elements

```
Action exitAction = new
  AbstractAction("Exit") // menu item text goes here
  {
    public void actionPerformed(ActionEvent event) {
      // action code goes here
      System.exit(0);
    }
  };
```

- Add action to the menu

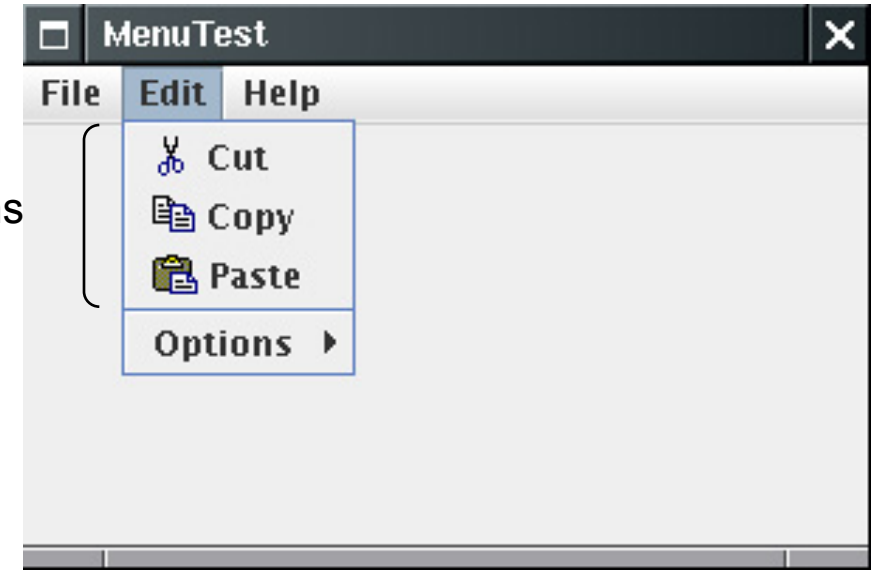
```
JMenuItem exitItem = fileMenu.add(exitAction);
```

Icons in Menu Items

- Menu items can have a *text label*, just an *icon* or *both*

```
JMenuItem cutItem = new JMenuItem("Cut", new ImageIcon("cut.gif"));
```

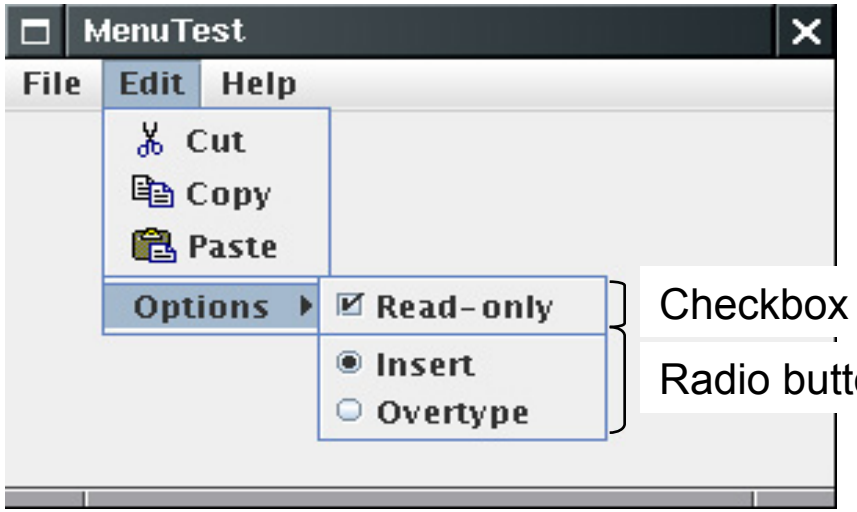
Menu items with icons



Menus

Checkbox and **Radio Button** Menu Items

- Display a checkbox or radio button next to the name



Treat these menu items just as you would any other

- Ex. Create a Checkbox menu item as follows:

```
JCheckBoxMenuItem readonlyItem = new JCheckBoxMenuItem("Read-only");
optionsMenu.add(readonlyItem);
```

Menus

JACOBS UNIVERSITY

Checkbox and Radio Button Menu Items

- The radio box menu items are handled similarly

```
ButtonGroup group = new ButtonGroup();
JRadioButtonMenuItem insertItem = new JRadioButtonMenuItem("Insert");
insertItem.setSelected(true);
JRadioButtonMenuItem overtypeItem = new JRadioButtonMenuItem("Overtype");
group.add(insertItem);
group.add(overtypeItem);
optionsMenu.add(insertItem);
optionsMenu.add(overtypeItem);
```

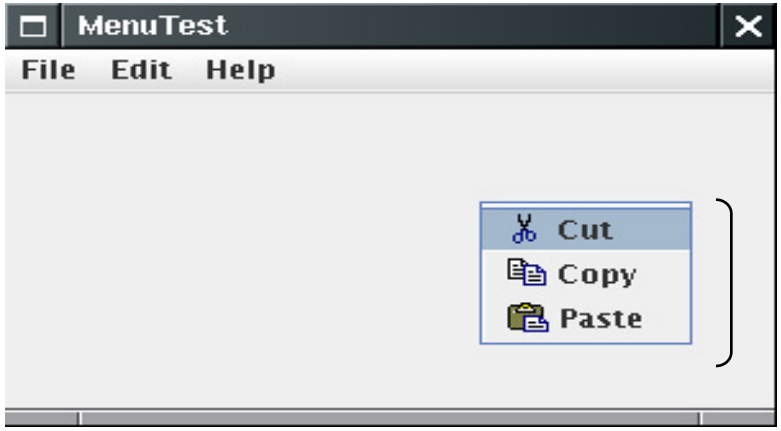
It is not necessary to be notified at exact moment an item is selected

- Use *isSelected* method to test the current state of the menu item

Menus

Pop-up Menus

- A menu that *is not attached to a menu bar*, *but floats around the surface*
- A pop-up menu has no title



Pop-up menu

Menus

Creating a pop-up menu

1. create pop-up menu object

```
JPopupMenu popup = new JPopupMenu();
```

2. create menu items, install listeners & add menu items to the pop-up menu

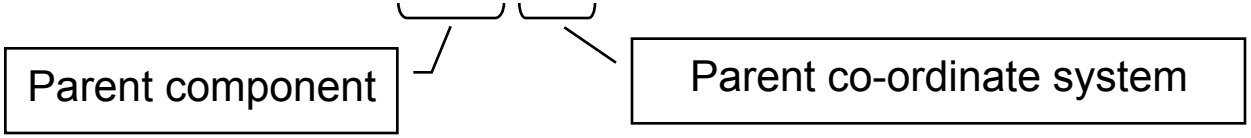
```
JMenuItem item = new JMenuItem("Cut");
item.addActionListener(listener);
popup.add(item);
```

Menus

Displaying Pop-up Menus

- Explicitly display pop-up menu using *show* method

```
popup.show(panel, x, y);
```



Using pop-up trigger (usually right mouse button on Windows & Linux)

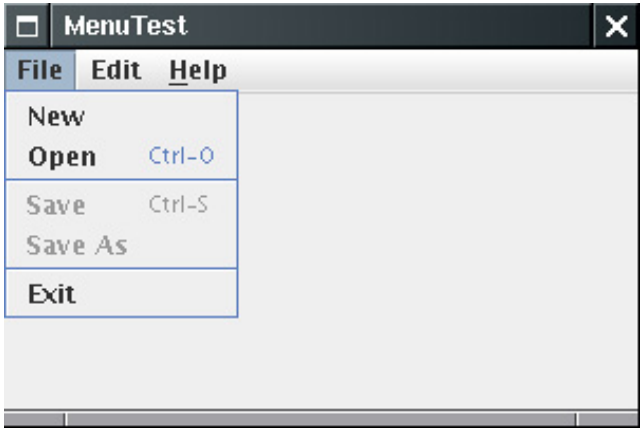
- Pops-up a menu when user clicks on a component using pop-up trigger

```
component.setComponentPopupMenu(popup);
```

Menus

Enable/ Disable Menus Items

- Use *setEnabled* method to enable/ disable a menu item



Strategy

- Register listener for “menu selected” (**javax.swing.event.MenuListener** interface) event

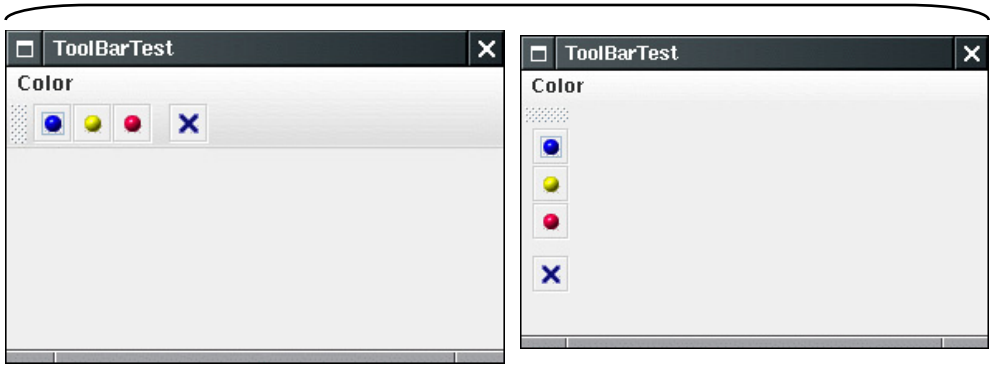
```
public void menuSelected(MenuEvent event) {
        saveAction.setEnabled(!readonlyItem.isSelected());
        saveAsAction.setEnabled(!readonlyItem.isSelected());
}
```

Menus

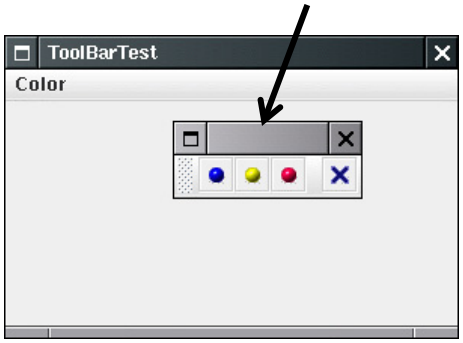
Toolbars

- A button bar giving quick access to commonly used commands in a program

Can drag buttons to four borders of the frame



Toolbar can be detached from frame



Menus

JACOBS UNIVERSITY

Programming toolbars

- Create tools bar object and add components into the toolbar

Create toolbar object

Add component/ Action objects to the toolbar

Add toolbar to frame

```
JToolBar bar = new JToolBar();

bar.add(blueButton);

…

add(bar, BorderLayout.NORTH);
```

Dialog Boxes

JACOBS UNIVERSITY

AWT distinguishes between dialog boxes

- Modal dialog boxes
- Modeless dialog boxes

Modal Dialog Boxes

- Won't allow users to interact with remaining windows of the application until he/ she deals with current dialog
- Used when information is needed from user before execution can proceed e.g., supplying a file name for a file to be read
- Application proceeds when the user closes the (modal) dialog box

Modeless Dialog Boxes

- Lets the user enter information in both dialog box and remainder of the application

Dialog Boxes

JACOBS UNIVERSITY

Option Dialogs

- Simple dialogs for asking a user single piece of information
- The **JOptionPane** has four static methods to show these simple dialogs

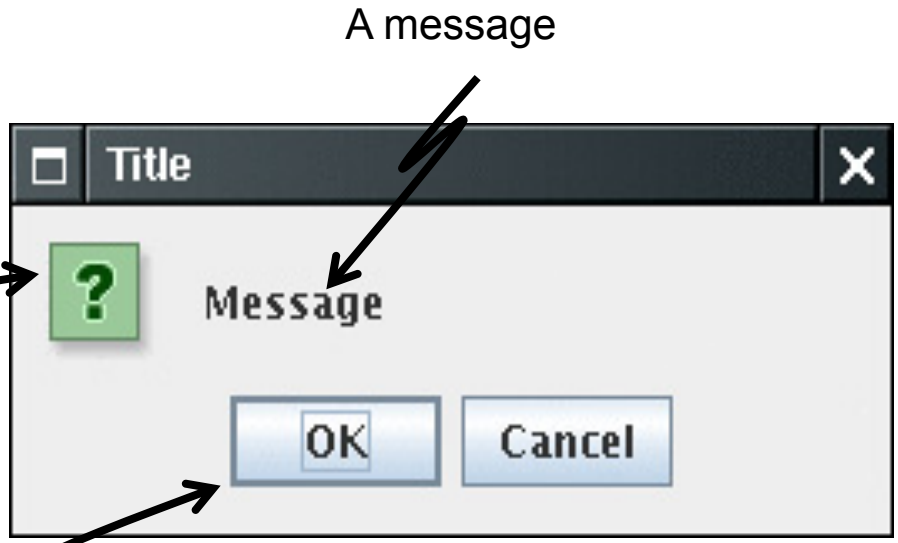
showMessageDialog	Show a message and wait for the user to click OK
showConfirmDialog	Show a message and get a confirmation (like OK/ Cancel)
showOptionDialog	Show a message and get a user option from a set of options
showInputDialog	Show a message and get one line of user input

Dialog Boxes

An Option Dialog

Icon depends on message types

- **ERROR_MESSAGE**
- **INFORMATION_MESSAGE**
- **WARNING_MESSAGE**
- **QUESTION_MESSAGE**
- **PLAIN_MESSAGE**



Option Buttons depend on
- Dialog type
- Option type

Dialog Boxes

JACOBS UNIVERSITY

Return Values

showMessageDialog	None
showConfirmDialog	An integer representing the chosen option
showOptionDialog	An integer representing the chosen option
showInputDialog	The string that the user supplied or selected

Dialog Boxes

JACOBS UNIVERSITY

Option Dialogs Summary

1. Choose the dialog type (message, confirmation, option, or input)

2. Choose the icon (error, information, warning, question, none, or custom)

3. Choose the message (string, icon, custom component, or a stack of them)

4. For a confirmation dialog, choose the option type (default, Yes/No, Yes/No/Cancel, or OK/Cancel)

Dialog Boxes

JACOBS UNIVERSITY

Option Dialogs Summary cont …

5. For an option dialog, choose the options (strings, icons, or custom components) and the default option

6. For an input dialog, choose between a text field and a combo box.

7. Locate the appropriate method to call in the **JOptionPane** API

Example

```
int selection = JOptionPane.showConfirmDialog(parent,   "Message", "Title",
        JOptionPane.OK_CANCEL_OPTION,
        JOptionPane.WARNING_MESSAGE);
if (selection == JOptionPane.OK_OPTION) . . .
```

Dialog Boxes

JACOBS UNIVERSITY

Creating Dialogs

1. Derive a class from **JDialog**
2. Call constructor of superclass **JDialog**, specify owner frame
3. Add UI components of dialog box
4. Add event handlers
5. Set the size of dialog box

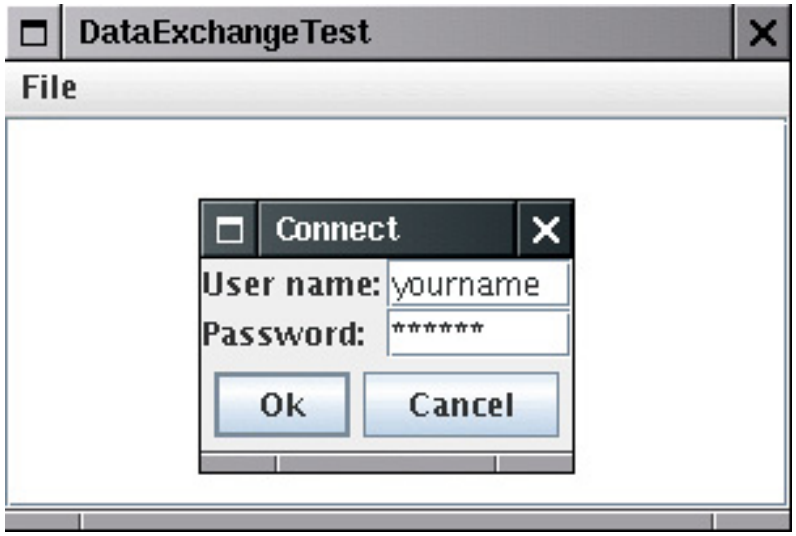
Dialog Boxes

```
public AboutDialog extends JDialog {
   public AboutDialog(JFrame owner) {
      super(owner, "About DialogTest", true);
      add(new JLabel(
         "<html><h1><i>Core Java</i></h1><hr>By Cay Horstmann and Gary
        Cornell</html>"),
         BorderLayout.CENTER);
      JPanel panel = new JPanel();
      JButton ok = new JButton("Ok");
      ok.addActionListener(new
         ActionListener()  {
            public void actionPerformed(ActionEvent event) {
               setVisible(false);
            }
         });
      panel.add(ok);
      add(panel, BorderLayout.SOUTH);
      setSize(250, 150);
   }
}
```

Dialog Boxes

Data Exchange

- Most common reason to use dialog is to get information from the user
- Example: A dialog to obtain a user name and a password



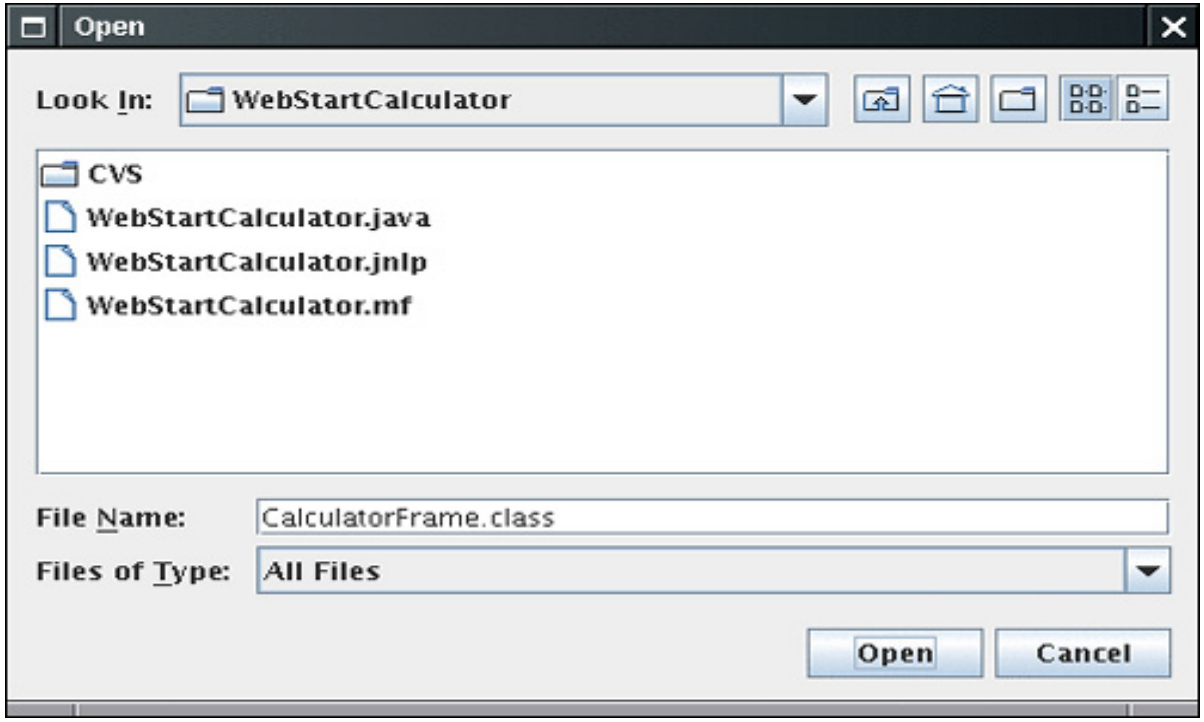
Dialog Boxes

JACOBS UNIVERSITY

File Dialogs

- Swing provides a **JFileChooser** class that allows to display a file dialog box
- **JFileChooser** dialogs are always modal
- **JFileChooser** is not a subclass of **JDialog**
- Call the method *showOpenDialog* to display a dialog for opening a file
- Call the method *showSaveDialog* to display a dialog for saving a file

Example



File Dialogs – Setup Steps

1. Make a **JFileChooser** object

2. Set the directory by calling the *setCurrentDirectory* method

3. Supply a default file name using the *setSelectedFile* method

4. To enable the user to select multiple files in the dialog, call the *setMultiSelectionEnabled* method

Dialog Boxes

JACOBS UNIVERSITY

File Dialogs – Setup Steps cont …

5. To restrict the display of files in the dialog to a particular type use a file filter

6. Use the *setFileSelectionMode* method to set files, directories or both for selection

7. Show the dialog box by calling the *showOpenDialog* or *showSaveDialog* method

8. Get the selected file or files with the *getSelectedFile*() or *getSelectedFiles*() method

Reading Assignment

Core Java 2 Volume I, Chapter 9 and 10. User Interface Components with Swing, Deploying Applets and Applications by Horstmann and Cornell