Java Swing Components: An Overview

Contents of lecture 5:

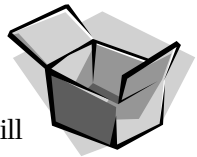
1. **Swing: Overview + Components**
   - More Swing details
   - Swing Components

Important! This lecture, more so than any other, will only present an overview of the topic. This is because the various components within Java are too numerous to consider in depth (given the available time). Ideally, you should experiment with the various components (both those covered within this lecture, and others) in order to gain familiarity and ease of use with the components.

Overview of how to write a GUI program

The three steps towards creating a GUI based Java program are as follows:

1. Declare controls (i.e. components). If necessary, the controls can be subclassed if specialised behavior is needed.
2. Implement and register event-handling interfaces, so that actions occurring to the controls perform the desired behavior.
3. Add the controls to a container. Again the containers can be subclassed if necessary.



In the last lecture we looked at event handling within Java. In this lecture we will explore the different components Java makes available to the programmer. The next lecture will explore containers.

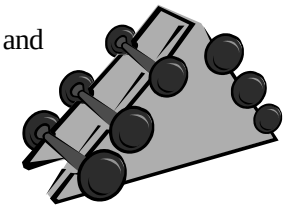
Some terminology: AWT, JFC, Swing

- AWT stands for *Abstract Window Toolkit*. It refers to a collection of basic GUI components, and other features, e.g. layout managers, component printing, etc.
- JFC stands for *Java Foundation Classes*. The JFC contains a much larger set of classes than the AWT set. Mostly it adds to the functionality of the AWT, however, in some cases it replaces the functionality of the AWT. The JFC contains accessibility functions, 2D drawing libraries, pluggable look and feel, etc.
- The *Swing* component set is part of the JFC. It contains a number of GUI components (buttons, textfields, scrollpanes, etc.) that are intended to be direct replacements of the corresponding AWT GUI components. Note, that Swing only replaces a subsection of the AWT (the GUI components), other aspects of the AWT (e.g. layout managers, etc.) remain unchanged.

More terminology: Light/heavyweight Components

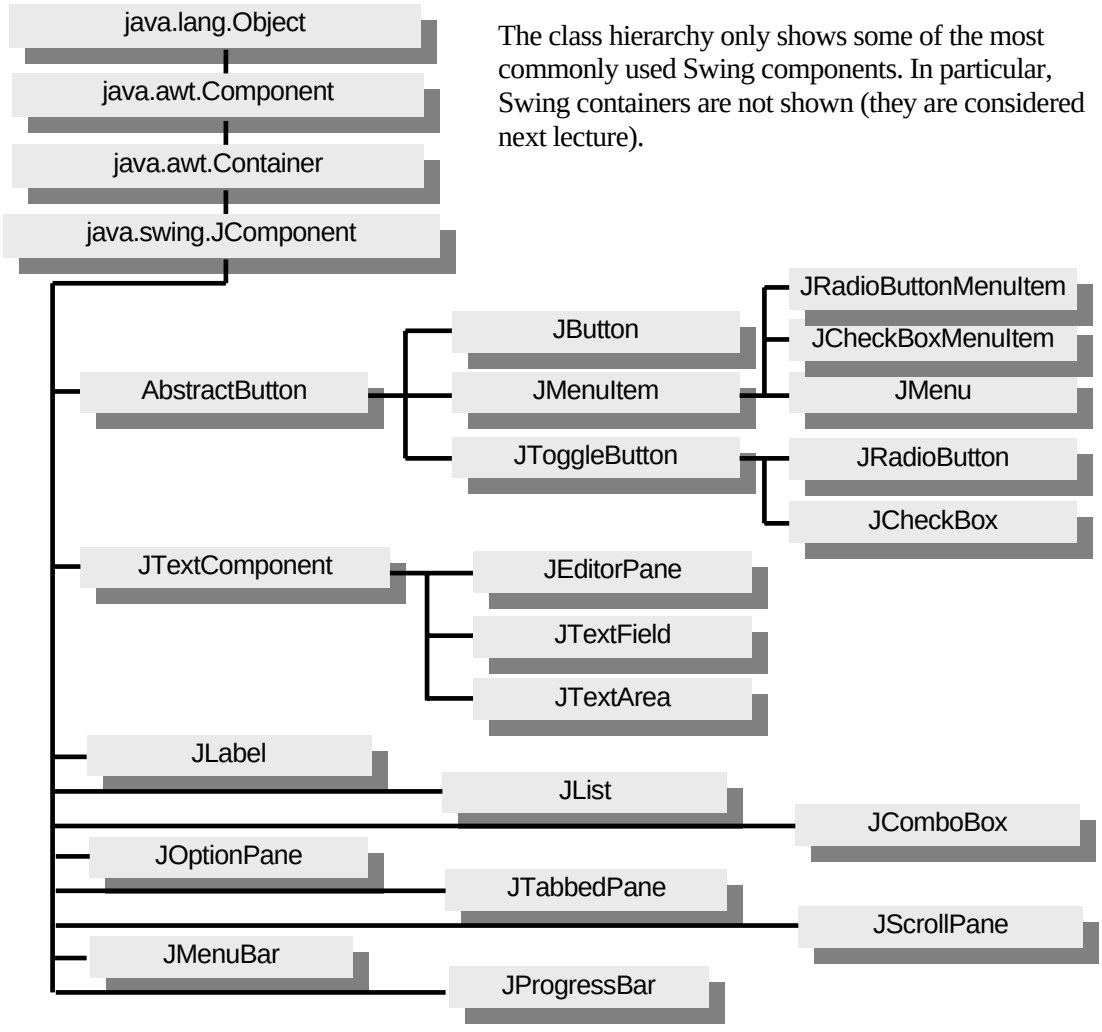
In AWT all components are termed *heavyweight* components. The reason for this is that an AWT component relies on the underlying operating system (e.g. Windows, Solaris, etc.) to provide the component (i.e. to paint it, redraw it, etc.). For example, an AWT button, when running on MacOS is actually a MacOS button. The term *heavyweight* does not really provide a clear indication of this dependency.

All Swing components are *lightweight* components. A lightweight component does not use a corresponding peer or native component. Instead, it is drawn by the Java VM (and not the underlying OS).



Heavyweight and *lightweight* components have a number of relative advantages and disadvantages. For example, lightweight components are platform independent, and easily support transparencies, etc., whereas heavyweight components are generally drawn considerably faster than lightweight components (due to hardware acceleration). Where possible, AWT and Swing GUI components should not be mixed with one another.

Overview of Swing Components



The class hierarchy only shows some of the most commonly used Swing components. In particular, Swing containers are not shown (they are considered next lecture).

public abstract class Component

Within Java a Component is defined as an object having a graphical representation that can be displayed on the screen and interact with the user. The class contains a large number of methods that are common to all components. An overview of some of the most useful methods contained within this class follows (in no particular order):

AddSomethingListener methods

The Component class supports the addComponentListener, addFocusListener, addKeyListener, addMouseListener and addMouseMotionListener methods.

There are also corresponding removeSomethingListener methods.

Foreground and background methods

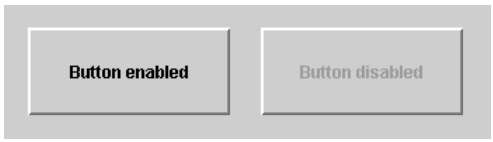
Every component can query its foreground and background colours through the use of the public Color getForeground() and public Color getBackground() methods. Likewise, the colours can be specified through the public void setBackground(Color c) and public void setForeground(Color c) methods.

Settings the size/position

The size and position of a component can be varied using a number of different methods. In particular, the public Dimension getSize() and pubic void setSize(int width, int height) methods permit the component to be resized. The public void setLocation(Point p) and public void setLocation(int x, int y) methods can be used to reposition the component.

Enabling, making visible, or the opposite

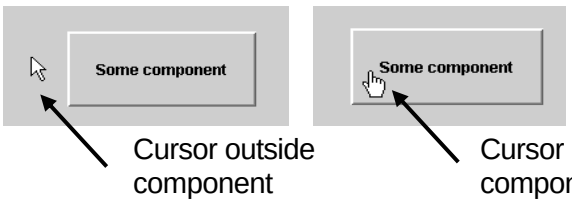
Components can be enabled or disabled (if disabled the user cannot interact with the component, i.e. use it) through the use of the public void setEnabled(boolean b) method. The public boolean isEnabled() method returns the state of the component. Disabled is also termed 'grayed out'



Likewise, a component can be made visible (or invisible) through the use of the public void setVisible(boolean b) method, and checked through the use of the public boolean isVisible() method. Components (e.g. buttons) can be disabled when it is inappropriate for the user to select them.

Setting the Cursor

The mouse cursor to be used within the component can be set using the public void setCursor(Cursor cursor) method. Likewise, the current cursor can be obtained using the public Cursor getCursor(), e.g.



The built in cursors include the following:

Cursor Name	Description
CROSSHAIR_CURSOR	Crosshair cursor
DEFAULT_CURSOR	Default arrow cursor
HAND_CURSOR	Hand cursor
MOVE_CURSOR	Move cursor
TEXT_CURSOR	Text cursor
WAIT_CURSOR	Wait cursor

For a given component, the cursor shape can be set to one of the built-in cursors as follows:

```
someComponent.setCursor(
        Cursor.getPredefinedCursor(
                Cursor.HAND_CURSOR ) );
```

The programmer can also define new cursor shapes (check the Sun site for further information).

Changing fonts within a component

The font that is used within a Component can be changed using the public void setFont(Font f) methods, and queried using the public Font getFont() method, e.g.

```
JButton1.setFont(new Font("Serif", Font.BOLD, 16));
JButton2.setFont(new Font("Dialog", Font.BOLD, 16));
```

Determining the parent of a component

Components are generally associated with a Container (to which they are added). The public Container getParent() method permits the container object, to which the component belongs, to be retrieved.

Painting the component

A number of different methods are contained within the Component class that permit the programmer to control when (and how) the component is drawn/redrawn on the screen.

repaint(): A number of repaint methods are offered. Repaint should be called whenever some change occurs to a graphical object. When repaint is called it simply results in a queued called to the update method being generated (i.e. it is the responsibility of the update method to actually redrawn the component, repaint simply signifies that the update method should be called). The following repaint methods are available:

Method	Purpose
public void repaint()	Repaints the component as soon as possible.
public void repaint(int x, int y, int width, int height)	Repaints the specified rectangle of the component ASAP.

update(): The AWT calls the public void update(Graphics g) method in response to a call from the repaint method. The update method firstly clears the component by filling it with the background colour. Next, the method calls the paint method (it is the responsibility of the paint method to actually draw the component on the screen).

paint(): The public void paint(Graphics g) method actually paints the component onto the specified graphics context.

The update and paint methods accept a Graphics object as a parameter. A Graphics object permits an application to draw to a component (offering methods such as drawRect, drawString, fillPolygon, etc.). Given a particular component, a Graphics context can be obtained through the public Graphics getGraphics() method.

Full details on the Component, Graphics, JComponent, JButton, etc. classes can be found at: http://java.sun.com/products/jdk/1.2/docs/api/index.html

Summary of Component class

A summary of methods within the Component class can be found below (note, not all methods are shown):

public abstract class Component extends java.lang.Object
implements ImageObserver, MenuContainer, Serializable

Method Name	Description
add(PopupMenu popup)	Adds the specified popup menu to the component.
addComponentListener(ComponentListener l)	Adds the specified component listener to receive component events from this component.
addFocusListener(FocusListener l)	Adds the specified focus listener to receive focus events from this component.
addKeyListener(KeyListener l)	Adds the specified key listener to receive key events from this component.
addMouseListener(MouseListener l)	Adds the specified mouse listener to receive mouse events from this component.
addMouseMotionListener(MouseMotionListener l)	Adds the specified mouse motion listener to receive mouse motion events from this component.
contains(Point p), contains(int x, int y)	Checks whether this component "contains" the specified point, where the point's x and y coordinates are defined to be relative to the coordinate system of this component.
createImage(int width, int height)	Creates an off-screen drawable image to be used for double buffering.
createImage(ImageProducer producer)	Creates an image from the specified image producer.
doLayout()	Prompts the layout manager to lay out this component.
getAlignmentX()	Returns the alignment along the x axis.
getAlignmentY()	Returns the alignment along the y axis.
getBackground()	Gets the background color of this component.
getBounds()	Gets the bounds of this component in the form of a Rectangle object.
getComponentAt(Point p), getComponentAt(int x, int y)	Returns the component or subcomponent that contains the specified point.
getCursor()	Gets the cursor set on this component.
getFont()	Gets the font of this component.
getFontMetrics(Font font)	Gets the font metrics for the specified font.
getForeground()	Gets the foreground color of this component.
getGraphics()	Creates a graphics context for this component.
getLocation()	Gets the location of this component in the form of a point specifying the component's top-left corner.
getLocationOnScreen()	Gets the location of this component in the form of a point specifying the component's top-left corner in the screen's coordinate space.

getMaximumSize()	Gets the maximum size of this component.
getMinimumSize()	Gets the mininimum size of this component.
getName()	Gets the name of the component.
getParent()	Gets the parent of this component.
getPreferredSize()	Gets the preferred size of this component.
getSize()	Returns the size of this component in the form of a Dimension object.
getToolkit()	Gets the toolkit of this component.
invalidate()	Invalidates this component.
isEnabled()	Determines whether this component is enabled.
isShowing()	Determines whether this component is showing on screen.
isValid()	Determines whether this component is valid.
isVisible()	Determines whether this component is visible.
paint(Graphics g)	Paints this component.
paintAll(Graphics g)	Paints this component and all of its subcomponents.
print(Graphics g)	Prints this component.
printAll(Graphics g)	Prints this component and all of its subcomponents.
remove(MenuComponent popup)	Removes the specified popup menu from the component.
removeComponentListener(ComponentListener l)	Removes the specified component listener so that it no longer receives component events from this component.
removeFocusListener(FocusListener l)	Removes the specified focus listener so that it no longer receives focus events from this component.
removeKeyListener(KeyListener l)	Removes the specified key listener so that it no longer receives key events from this component.
removeMouseListener(MouseListener l)	Removes the specified mouse listener so that it no longer receives mouse events from this component.
removeMouseMotionListener(MouseMotionListener l)	Removes the specified mouse motion listener so that it no longer receives mouse motion events from this component.
repaint(long tm, int x, int y, int width, int height)	Repaints the specified rectangle of this component within tm milliseconds.
repaint(long tm)	Repaints the component.
repaint(int x, int y, int width, int height)	Repaints the specified rectangle of this component.
repaint()	Repaints this component.
requestFocus()	Requests that this component get the input focus.
setBackground(Color c)	Sets the background color of this component.
setBounds(Rectangle r)	Moves and resizes this component to conform to the new bounding rectangle r.
setBounds(int x, int y, int width, int height)	Moves and resizes this component.
setCursor(Cursor cursor)	Set the cursor image to a predefined cursor.
setEnabled(boolean b)	Enables or disables this component, depending on the value of the parameter b.
setFont(Font f)	Sets the font of this component.
setForeground(Color c)	Sets the foreground color of this component.
setLocation(Point p), setLocation(int x, int y)	Moves this component to a new location.
setName(String name)	Sets the name of the component to the specified string.
setSize(int width, int height), setSize(Dimension d)	Resizes this component so that it has specified width and height.
setVisible(boolean b)	Shows or hides this component depending on the value of parameter b.
toString()	Returns a string representation of this component and its values.
transferFocus()	Transfers the focus to the next component.
update(Graphics g)	Updates this component.
validate()	Ensures that this component has a valid layout.

Summary of Graphics class

A summary of methods within the Graphics class can be found below (note, the list is not exhaustive)

public abstract class Graphics extends java.lang.Object

Method Name	Description
clearRect(int x, int y, int width, int height)	Clears the specified rectangle by filling it with the background color of the current drawing surface.
clipRect(int x, int y, int width, int height)	Intersects the current clip with the specified rectangle.
copyArea(int x, int y, int width, int height, int dx, int dy)	Copies an area of the component by a distance specified by dx and dy.
create()	Creates a new Graphics object that is a copy of this Graphics object.
create(int x, int y, int width, int height)	Creates a new Graphics object based on this Graphics object, but with a new translation and clip area.
dispose()	Disposes of this graphics context and releases any system resources that it is using.
draw3DRect(int x, int y, int width, int height, boolean raised)	Draws a 3-D highlighted outline of the specified rectangle.
drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)	Draws the outline of a circular or elliptical arc covering the specified rectangle.
drawBytes(byte[] data, int offset, int length, int x, int y)	Draws the text given by the specified byte array, using this graphics context's current font and color.
drawChars(char[] data, int offset, int length, int x, int y)	Draws the text given by the specified character array, using this graphics context's current font and color.
drawImage(Image img, int x, int y, ImageObserver observer)	Draws as much of the specified image as is currently available.
drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)	Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.
drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver observer)	Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.
drawLine(int x1, int y1, int x2, int y2)	Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.
drawOval(int x, int y, int width, int height)	Draws the outline of an oval.
drawPolygon(int[] xPoints, int[] yPoints, int nPoints), drawPolygon(Polygon p)	Draws a closed polygon defined by arrays of x and y coordinates.
drawRect(int x, int y, int width, int height)	Draws the outline of the specified rectangle.
drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)	Draws an outlined round-cornered rectangle using this graphics context's current color.
drawString(String str, int x, int y)	Draws the text given by the specified string, using this graphics context's current font and color.
fill3DRect(int x, int y, int width, int height, boolean raised)	Paints a 3-D highlighted rectangle filled with the current color.
fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)	Fills a circular or elliptical arc covering the specified rectangle.
fillOval(int x, int y, int width, int height)	Fills an oval bounded by the specified rectangle with the current color.
fillPolygon(Polygon p), fillPolygon(int[] xPoints, int[] yPoints, int nPoints)	Fills the polygon defined by the specified Polygon object with the graphics context's current color.
fillRect(int x, int y, int width, int height)	Fills the specified rectangle.
fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)	Fills the specified rounded corner rectangle with the current color.

finalize()	Disposes of this graphics context once it is no longer referenced.
getClip()	Gets the current clipping area.
getColor()	Gets this graphics context's current color.
getFont()	Gets the current font.
getFontMetrics(Font f), getFontMetrics()	Gets the font metrics for the specified font.
setClip(Shape clip)	Sets the current clipping area to an arbitrary clip shape.
setClip(int x, int y, int width, int height)	Sets the current clip to the rectangle specified by the given coordinates.
setColor(Color c)	Sets this graphics context's current color to the specified color.
setFont(Font font)	Sets this graphics context's font to the specified font.
setXORMode(Color c1)	Sets the paint mode of this graphics context to alternate between this graphics context's current color and the new specified color.
toString()	Returns a String object representing this Graphics object's value.
translate(int x, int y)	Translates the origin of the graphics context to the point (x, y) in the current coordinate system.

public abstract class Container extends Component

As noted, JComponent extends Container, which in turn extends Component. This is not an ideal relationship (JComponent should ideally directly extend Component, however, to increase the functionality of the Swing set it was necessary to extend the Container class).

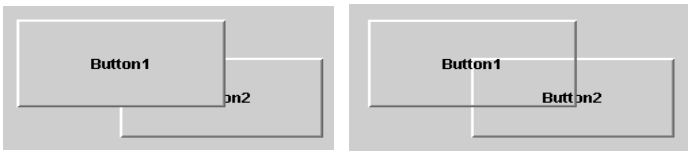
The Container class will be explored in more detail next lecture (it is not considered further within this lecture).

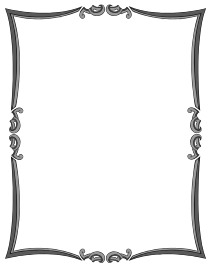
public abstract class JComponent extends Container

JComponent is the base class for all Swing components. An overview of the more commonly used methods within this class follows:

Opaque JComponents

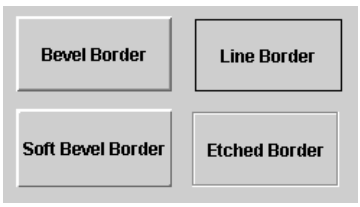
The public void setOpaque(boolean b) method can be used to determine if the component should be opaque, or not. The public boolean isOpaque() can be used to query if the component is opaque.





Setting the borders of Swing components

Swing components can have an associated border. Examples include those shown below:



Borders can be specified through the public void setBorder(Border b) method, and likewise retrieved via the public Border getBorder() method. Border is actually an interface that contains a number of methods, namely: getBorderInserts, isBorderOpaque, paintBorder, etc. Java provides a number of abstract border classes which the programmer can extend and implement, namely: BevelBorder, SoftBevelBorder, EtchedBorder, LineBorder, TitledBorder, and MatteBorder.

Borders can be set as follows:

```
import javax.swing.border.*;

JButton aButton = new JButton( "Go" );
aButton.setBorder( new BevelBorder( BevelBorder.RAISED ) );
```

Painting a JComponent

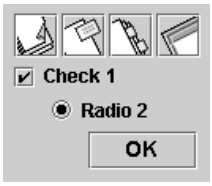
Apart from the normal painting methods offered by the Component class, JComponent offers a number of other useful methods, e.g. the public void setDoubleBuffered(boolean b) method determines if double buffering should be used when painting the component (useful for GUIs containing moving elements).

Additionally, the public void paintImmediately(int x, int y, int w, int h) and public void paintImmediately(Rectangle r) methods can be used to force the JComponent to be updated immediately. This method will eventually migrate to the Component class.

Overview of common Swing controls

An overview of the most common Swing controls now follows (containers will be explored next lecture):

Check boxes, radio buttons and buttons



Java provides a number of different types of button, all of which expand the AbstractButton class (menu components are also derived from this class and are considered later).

Aside: The AbstractButton class provides the following useful methods: addActionListener, addChangeListener, addItemListener, setIcon, setText, etc, (with corresponding remove/get methods)

An image can be added to any type of Swing based button as follows:

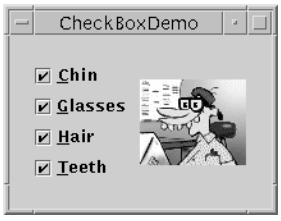
```
ImageIcon someIcon = new ImageIcon("images/icon.gif");
JButton button = new JButton("Name", someIcon );
```

Brief details of the different types of button now follows:

public class JButton extends AbstractButton

A JButton provides basic button functionality (and needs no further exploration).

public class JCheckBox extends JToggleButton



The JCheckBox class, as well as the JRadioButton class extends the JToggleButton class (which in turn extends the AbstractButton class). Check boxes and radio buttons are similar to one another; however, they differ in terms of how they can be selected. For example, given a group of check boxes, any number (including none) can be selected at any one time, whereas, a group of radio buttons can only have one button selected at a time.

The following code snippet shows how two JCheckBox objects can be defined and linked together through the use of an item listener.

```
// Define the two check boxes
first = new JCheckBox( "First" );
second = new JCheckBox( "Second" );

// Initially select both boxes
first.setSelected( true );
second.setSelected( true );
```

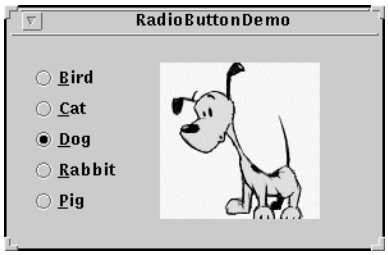
```
// Register a listener with the check boxes
CheckBoxListener myListener = new CheckBoxListener();
first.addItemListener( myListener );
second.addItemListener( myListener );

// Definition of the check box listener
class CheckBoxListener implements ItemListener
{
        public void itemStateChanged( ItemEvent e )
        {
                Object source = e.getItemSelectable();
                if( source == first )
                { // Whatever is needed here. }
                if( source == second )
                { // Whatever is needed here }
        }
}
```

Typically, it is not necessary to write an ItemListener (it is only needed if you want to perform some action as soon as the buttons are clicked). Simply use the someCheckBox.isSelected() method to determine if the box has been selected.

public class JRadioButton extends JToggleButton

Radio buttons are a group of buttons in which only one button can be selected at a time (the term comes from old car radio buttons, for which you could only push down one button at a time).

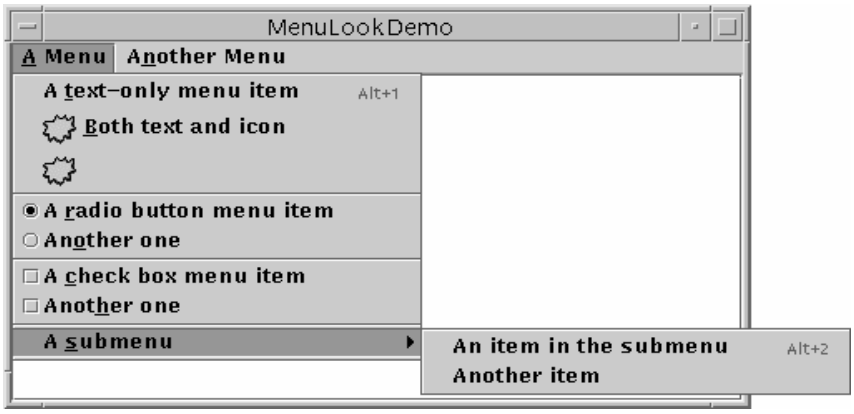


Radio buttons can be used in much the same way that check boxes can be used.

Buttons, check boxes, and radio buttons are primarily intended to extract information from the user.

Menus within Java

Menus are also derived from the AbstractButton class (mostly as they are something selectable by the user). Menus are unlike other components in that they only appear either in a *menu bar* or alternatively as a *popup menu*. An overview of the different JMenuItem classes (and the JMenuBar class) follows:



public class JMenuItem extends AbstractButton

This is the superclass of all types of menu item. Any object that extends this class can potentially be added to a menu bar object (JMenuBar). By default, a JMenuItem contains a string of text, and potentially an icon, indicating to the user the function of this menu item.

public class JCheckBoxMenuItem extends JMenuItem

This is a menu item that can be selected or deselected (if selected it will appear with a checkmark next to it). The isSelected and setSelected methods can be used to determine or set the state of the menu item.

public class JRadioButtonMenuItem extends JMenuItem

This type of menu item is similar to JCheckBoxMenuItem except that only one radio button within a group can be selected at a time.

public class JMenu extends JMenuItem

A JMenu object contains a number of menu items (i.e. forming a menu). It extends the JMenuItem class as a menu can itself be added to a menu (i.e. becoming a sub-menu).

public class JMenuBar extends JComponent

Unlike menu items, the JMenuBar component does not extend the JMenuItem class. Instead, a JMenuBar object is a container class for JMenu objects (i.e. holding and displaying any menus constructed by the programmer). The sample code shown opposite illustrates how a menu can be created:

```
// Create a menu, and add some items to it
JMenu firstMenu = new JMenu( “File” );
JMenuItem open = new JMenuItem( “Open” );
JMenuItem close = new JMenuItem( “Close” );
firstMenu.add( open );
firstMenu.add( second );

// Create another menu, with a submenu+checkbox
JMenu secondMenu = new JMenu( “Misc” );
JMenu subMenu = new JMenu( “Sub-menu” );
JCheckBoxMenuItem check = new JCheckBoxMenuItem( “Action?” );
check.setSelected( true );
subMenu.add( check );
secondMenu.add( subMenu );

// Define a menu bar, add the menus to it
JMenuBar menuBar = new JMenuBar();
menuBar.add( firstMenu ); menuBar.add( secondMenu );

// Add menu bar to JFrame/JApplet/JDialog
addJMenuBar( menuBar );
```

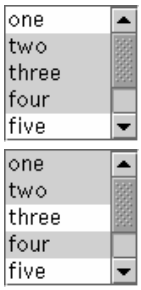
Useful Swing Components

In what follows a brief overview of several useful Swing components is presented. However, only an overview is given as to their use. You should consult Sun’s Java Swing Tutorial for additional information.

public class JList extends JComponent

A JList presents the user with a group of items, displayed in a column and selectable by the user. The options selected by the user can be determined using the getSelectedIndex() and getSelectedIndices() methods.

JLists can be created using a number of different selection criteria, namely:



- SINGLE_SELECTION entails that only one item can be selected at a time.
- SINGLE_INTERVAL_SELECTION entails that multiple, contiguous items can be selected.
- MULTIPLE_INTERVAL_SELECTION entails that any selection of items can be made.

public class JSlider extends JComponent

A JSlider is ideally used in those situations where the user can specify a numeric value, bounded by maximum and minimum values. The use of a slider, as opposed to a text field ensures that input errors are eliminated.

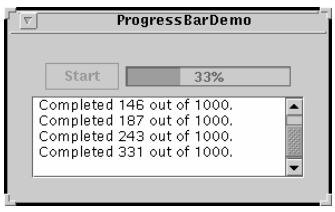

```
// Create a new JSlider object
JSlider slider = new JSlider( JSlider.HORIZONTAL,
                        iMinValues, iMaxValue, iStartingValue );

// Add a change listener
slider.addChangeListener( new mySlideListener() );

// Define the look of the slider
slider.setMajorTickSpacing( 10 );
slider.setMinorTickSpacing(1);
slider.setPaintTicks(true);
slider.setPaintLabels(true);
```

public class JProgressBar extends JComponent

Often in complex programs a certain task may take a longish time to complete. In such cases it is advisable that the program indicates to the user how long the task will take to complete. The JProgressBar class provides an easy means of accomplishing this.



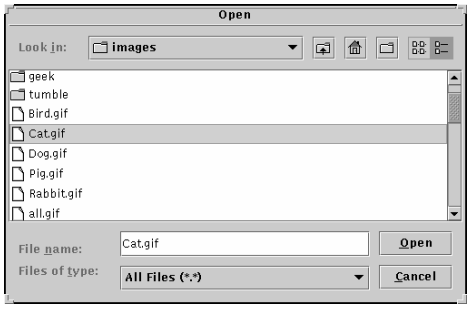

```
// Create a new JProgressBar object
JProgressBar progressBar =
          new JProgressBar( iStartValue, iEndValue );

// Define initial value, and give the slider a border
progressBar.setValue(0); progressBar.setBorderPainted(true);


// Update the progress bar at some point
progressBar.setValue( progressBar.getValue() + 1 );
```

public class JFileChooser extends JComponent

File choosers are provided for easy GUI navigation, permitting the user to select a file or directory. The JFileChooser object returns the user's selection; it is the responsibility of the program to determine what happens to the selected file.



The file selected by the user can be retrieved using the public File getFile() method.

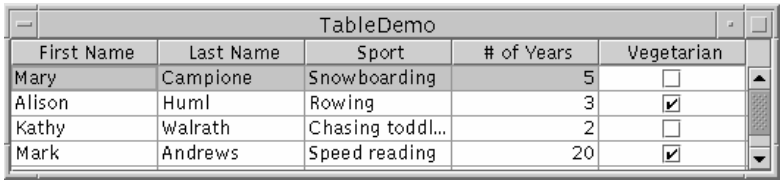
```
// Create a JFileChooser object
JFileChooser chooser = new JFileChooser();

// Create and define an ExtensionFileFIlter
ExtensionFileFilter filter = new ExtensionFileFilter();
filter.addExtension( "jpg" );
filter.addExtension( "gif" );
filter.setDescription( "JPG & GIF Images" );
chooser.setFileFilter( filter );

// Display the file chooser and print the file selection
int returnVal = chooser.showOpenDialog( parent );
if( returnVal = JFileChooser.APPROVE_OPTION )
        System.out.println( "Open file to be = "
                + chooser.getSelectedFile().getName() );
```

public class JTable extends JComponent

Swing also supports table objects, providing an easy way in which tabular data can be presented to the user.



The code needed to construct the opposite table follows:

```
Object[][] data = { {"Mary", "Campione",  "Snowboarding",
                        new Integer(5), new Boolean(false)},
                    {"Alison", "Huml", "Rowing", new Integer(3),
                        new Boolean(true)},
                    {"Kathy", "Walrath", "Chasing toddlers",
                        new Integer(2), new Boolean(false)},
                    {"Mark", "Andrews", "Speed reading",
                        new Integer(20), new Boolean(true)},
                    {"Angela", "Lih", "Teaching high school",
                        new Integer(4), new Boolean(false)}
                };
String[] columnNames = {"First Name", "Last Name",
                "Sport", "# of Years", "Vegetarian"};
final JTable table = new JTable(data, columnNames);
```

A JTable object can easily be placed within a scroll pane, as follows:

```
JScrollPane scrollPane = new JScrollPane(table);
table.setPreferredScrollableViewportSize( new Dimension(500, 70));
```

Reminder! Reminder! Reminder!

This lecture has simply introduced some Swing components. You should extend the introduction by further exploring the various components within your own programs.

Summary of JComponent class

A summary of methods within the JComponent class can be found below (Note the list is not exhaustive)

public abstract class JComponent extends javax.swing.Container implements Serializable

Method Name	Description
addPropertyChangeListener(String propertyName, PropertyChangeListener listener)	Add a PropertyChangeListener for a specific property.
contains(int x, int y)	Give the UI delegate an opportunity to define the precise shape of this component for the sake of mouse processing.
createToolTip()	Returns the instance of JToolTip that should be used to display the tooltip.
getAccessibleContext()	Get the AccessibleContext associated with this Jcomponent
getActionForKeyStroke(KeyStroke aKeyStroke)	Return the object that will perform the action registered for a given keystroke.
getAlignmentX()	Overrides Container.getAlignmentX to return the vertical alignment.
getAlignmentY()	Overrides Container.getAlignmentY to return the horizontal alignment.
getAutoscrolls()	Returns true if this component automatically scrolls its contents when dragged, (when contained in a component that supports scrolling, like Jviewport
getBorder()	Returns the border of this component or null if no border is currently set.
getInsets()	If a border has been set on this component, returns the border's insets, else calls super.getInsets.
getRootPane()	Returns the JRootPane ancestor for a component
getToolTipLocation(MouseEvent event)	Return the tooltip location in the receiving component coordinate system If null is returned, Swing will choose a location.
getToolTipText()	Return the tooltip string that has been set with setToolTipText()
getTopLevelAncestor()	Returns the top-level ancestor of this component (either the containing Window or Applet), or null if this component has not been added to any container.
isDoubleBuffered()	Return whether the receiving component should use a buffer to paint.
isLightweightComponent(Component c)	Returns true if this component is a lightweight.
isOpaque()	Returns true if this component is completely opaque.
paintImmediately(Rectangle r), paintImmediately(int x, int y, int w, int h)	Paint the specified region now.
requestDefaultFocus()	Request the focus for the component that should have the focus by default.
reshape(int x, int y, int w, int h)	Moves and resizes this component.
setAlignmentX(float alignmentX)	Set the the vertical alignment.
setAlignmentY(float alignmentY)	Set the the horizontal alignment.
setBorder(Border border)	Sets the border of this component.
setDoubleBuffered(boolean aFlag)	Set whether the receiving component should use a buffer to paint.
setEnabled(boolean enabled)	Sets whether or not this component is enabled.
setOpaque(boolean isOpaque)	If true the components background will be filled with the background color.
setVisible(boolean aFlag)	Makes the component visible or invisible.

Common Swing GUI Components

A graphical overview of common Swing GUI components now follows (Note this material has been directly taken from Sun's Java site – more complete details of components can be found there).

Top-level Containers

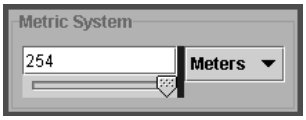
Applet

Dialog

Frame

General-purpose Containers

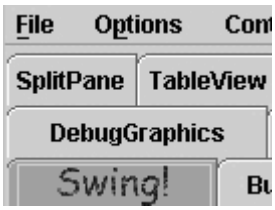
Panel



Scroll Pane

Split Pane

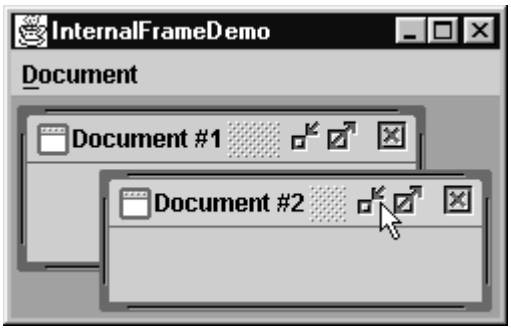
Tabbed Pane



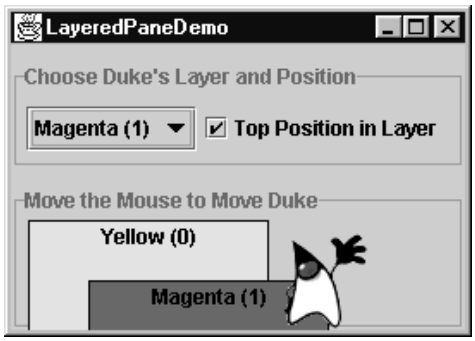
Toolbar

Special -purpose containers

Internal Frame

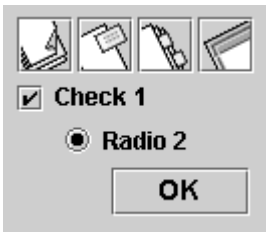


Layered Pane

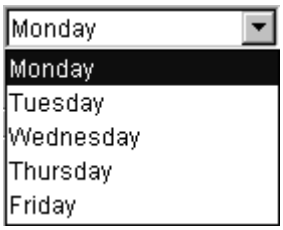


Basic Controls

Buttons



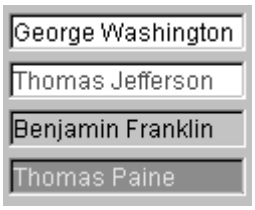
Combo box



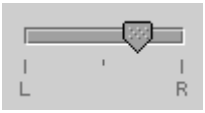
List

Menu

Text fields

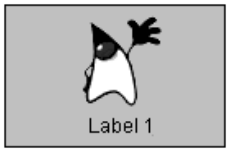


Slider



Uneditable Information Displays

Label

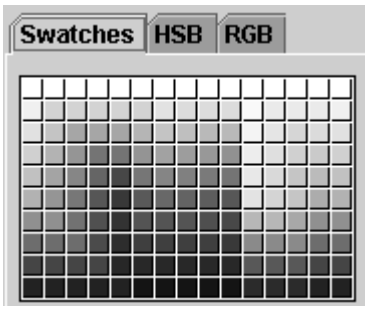


Progress bar

Tool tip

Editable Displays of Formatted Information

Colour Chooser

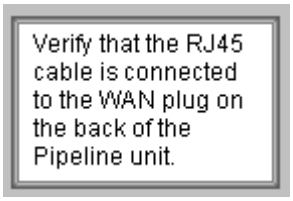


File Chooser

Table

First Na...	Last Name
Mark	Andrews
Tom	Ball
Alan	Chung
Jeff	Dinkins

Text



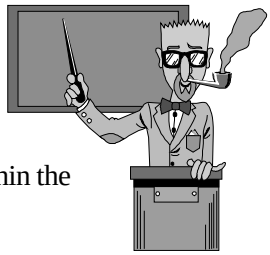
Tree

Practical 5

After this lecture you should explore the fifth practical pack which should enable you to investigate the material in this lecture.

Learning Outcomes

Once you have explored and reflected upon the material presented within this lecture and the practical pack, you should:



- Appreciate the differences between AWT/heavyweight and Swing/lightweight components.
- Have knowledge of the Component and JComponent classes, including commonly employed functionality within both classes and their role within the Swing component hierarchy.
- Have knowledge of common Swing components and be able to employ these components within Java programs.

More comprehensive details can be found in the CSC735 Learning Outcomes document.