1 AWT - The Abstract Window Toolkit

At this point, we will deviate from the "most elegant" path of teaching Java. We will be following the "most practical" path of learning for a while, before we come back to the convetional order of things.

Rather than getting bored with object-oriented mumbo-jumbo at this point, we will learn how to create simple graphical user interfaces (GUI) using the abstract window toolkit (AWT) in Java. It should also provide a concrete example for class inheritance which we will return to later.

The AWT is the class library which Java provides for the purpose of creating (almost) platform-independent user interfaces. It contains most of the "conventional" GUI components, such as buttons, selection boxes, text input boxes and so on. We will first see how to create a stand-alone Java *application*. Later, we will see how to embed applications in a browser – in other words, how to create *applet*s.

2 The `Frame`

The Java documentation defines a `Frame` as "a top level window with a title and window". `Frame` resides in the package `java.awt`, like many of the other components we will examine. It has four different constructors, only two of which we shall examine now:

```
Frame

public Frame()

    Constructs a new instance of Frame that is initially
    invisible. The title of the Frame is empty.
    See Also:
        Component.setSize(int, int), Component.setVisible(boolean)

Frame

public Frame(String title)

    Constructs a new, initially invisible Frame object
    with the specified title.

    Parameters:
        title - the title to be displayed in the frame's border.
        A null value is treated as an empty string, "".

    See Also:
        Component.setSize(int, int), Component.setVisible(boolean)
```

These definitions are directly from Java documentation. The mentioned "title" will appear in the title bar of the newly created window. Note that, whether or not a title is provided, the newly created `Frame` will be invisible.

In order to test our first `Frame`, we need to know a minimum of two more methods of the `Frame` class, but we shall learn three just to add some fun. Here they are:

```
setVisible

public void setVisible(boolean b)

    Shows or hides this component depending on
    the value of parameter b.
    Parameters:
        b - If true, shows this component; otherwise,
        hides this component.
    Since:
        JDK1.1
    See Also:
        isVisible()

setSize

public void setSize(int width,
                    int height)

    Resizes this component so that it has
    width width and height.
    Parameters:
        width - The new width of this component in pixels.
        height - The new height of this component in pixels.
    Since:
        JDK1.1
    See Also:
        getSize(), setBounds(int, int, int, int)

setLocation

public void setLocation(int x, int y)

    Moves this component to a new location. The
    top-left corner of the new location
    is specified by the x and y parameters in the coordinate
    space of this component's parent.
    Parameters:
        x - The x-coordinate of the new location's
        top-left corner in the parent's coordinate space.
        y - The y-coordinate of the new location's
        top-left corner in the parent's coordinate space.
    Since:
        JDK1.1
    See Also:
        getLocation(), setBounds(int, int, int, int)
```

Note that some of the words used in the documentation may not be familiar to you. This is just fine, at least for now. Now we have enough information to display a window of any size on our screen at any point. Here is the example code to do that:

```
import java.awt.*;

public class FrameTest {

    public static void main (String[] args) {
        Frame theFrame = new Frame("This is our first frame!");
        theFrame.setSize(300, 200);
theFrame.setLocation(200, 200);
        theFrame.setVisible(true);

    }
}
```

Type in this program, and execute it. If you did everything right, you should see a small window somewhere around the top-left of your screen. Note that you can not close the window. This is because we are not handling any events yet. So, you must stop this program by using the IDE.

3 The `Label`

A `Label` is a rather dull user interface component. Its only use is to display a text message, which can *not* be edited by the user. It may, however, be modified by the program itself.

`Label` has exactly three constructors, as below:

```
Label

public Label()

    Constructs an empty label. The text of the label is the
    empty string "".

Label

public Label(String text)

    Constructs a new label with the specified string of text, left
    justified.
    Parameters:
        text - the string that the label presents. A null value
        will be accepted without causing a NullPointerException
        to be thrown.

Label

public Label(String text, int alignment)
```

```
    Constructs a new label that presents the specified string of
    text with the specified alignment. Possible values for
    alignment are Label.LEFT, Label.RIGHT, and Label.CENTER.
    Parameters:
        text - the string that the label presents. A null value
        will be accepted without causing a
        NullPointerException to be thrown.
        alignment - the alignment value.
```

So once we have created a `Label`, how to we tell Java to place that label within something else (our `Frame`, for instance)? The answer to that is the following method:

```
add

public Component add(Component comp)

    Adds the specified component to the end of this
    container.
    Parameters:
        comp - the component to be added.
    Returns:
        the component argument.
```

Now, edit the code you already entered to look like the following:

```
import java.awt.*;

public class FrameTest {


    public static void main (String[] args) {
        Frame theFrame = new Frame("This is our first frame!");
        theFrame.setSize(300, 200);
        theFrame.setLocation(200, 200);

        Label theLabel = new Label("Hello World!");
        theFrame.add(theLabel);
        theFrame.setVisible(true);
    }
}
```

Note that we moved the `setVisible()` call down to the end. This is in order to make sure Java knows what to display before the window is in fact displayed. Otherwise, we would have to tell Java to update the view, but we don't want to complicate things right now.

Run the code. If everything went fine, you should see "Hello, World!" displayed in the window. However, it might not be exactly where you expected it to be. We will come back to that issue soon enough.

4 The `Button`

A `Button` is exactly what it sounds like, a button. It has two very simple constructors:

```
Button

public Button()

    Constructs a Button with no label.

Button

public Button(String label)

    Constructs a Button with the specified label.
    Parameters:
        label - A string label for the button.
```

Just to give it a test drive, edit the code again, to look like this:

```
import java.awt.*;

public class FrameTest {

    public static void main (String[] args) {
        Frame theFrame = new Frame("This is our first frame!");
        theFrame.setSize(300, 200);
        theFrame.setLocation(200, 200);
        Button theButton = new Button("Press me!");
        theFrame.add(theButton);
        theFrame.setVisible(true);
    }
}
```

When you run this, you will see that the whole `Frame` has become one huge button. Do as the button says and push it a few times to see how it works. Then, stop the program as before.

5 Layout Managers

When we add more than one component to a `Frame` (or any other `Container`, for that matter) we need to tell Java how we want those components laid out. To do that, we need to define the layout manager for the `Frame`. There are a few layout managers, and we will examine one right now.

6 `BorderLayout`

`BorderLayout` is the default layout for a `Frame`. The Java documentation states the following:

```
A border layout lays out a container, arranging and resizing
its components to fit in five regions: north, south, east, west,
and center. Each region is identified by a corresponding
constant: NORTH, SOUTH, EAST, WEST, and CENTER. When adding
a component to a container with a border layout, use one of
these five constants, for example:

    Panel p = new Panel();
    p.setLayout(new BorderLayout());
    p.add(new Button("Okay"), BorderLayout.SOUTH);


As a convenience, BorderLayout interprets the absence of a
string specification the same as the constant CENTER:

    Panel p2 = new Panel();
    p2.setLayout(new BorderLayout());
    p2.add(new TextArea());
```

Now, modify the example program to test out `BorderLayout`.

7 The **`TextField`**

A TextField object is a text component that allows for the editing of a single line of text.

Constructor	Explanation
`TextField()`	Constructs a new text field.
`TextField(int columns)`	Constructs a new empty text field with the specified number of columns.
`TextField(String text)`	Constructs a new text field initialized with the specified text.
`TextField(String text, int columns)`	Constructs a new text field initialized with the specified text to be displayed, and wide enough to hold the specified number of columns.

Method	Explanation
`int getColumns()`	Gets the number of columns in this text field.
`void setColumns(int column)`	Sets the number of columns in this text field.
`String getText()`	Gets the text that is presented by this text component. (Inherited from `java.awt.TextComponent`)
`void setText(String t)`	Sets the text that is presented by this text component to be the specified text.
`char getEchoChar()`	Gets the character that is to be used for echoing.
`void setEchoChar(char c)`	Sets the echo character for this text field.
`boolean echoCharIsSet()`	Indicates whether or not this text field has a character set for echoing.

8 The `TextArea`

A `TextArea` object is a multi-line region that displays text. It can be set to allow editing or to be read-only.

Constructor	Explanation
`TextArea()`	Constructs a new text area.
`TextArea(int rows, int columns)`	Constructs a new empty text area with the specified number of rows and columns.
`TextArea(String text)`	Constructs a new text area with the specified text.
`TextArea(String text, int rows, int columns)`	Constructs a new text area with the specified text, and with the specified number of rows and columns.
`TextArea(String text, int rows, int columns, int scrollbars)`	Constructs a new text area with the specified text, and with the rows, columns, and scroll bar visibility as specified.

The `scrollbars` argument can be one of `TextArea.SCROLLBARS_BOTH`,
`TextArea.SCROLLBARS_HORIZONTAL_ONLY`,
`TextArea.SCROLLBARS_NONE`,
`TextArea.SCROLLBARS_VERTICAL_ONLY`.

Method	Explanation
`void append(String str)`	Appends the given text to the text area's current text.
`int getColumns()`	Gets the number of columns.
`void setColumns(int columns)`	Sets the number of columns.
`int getRows()`	Gets the number of rows.
`void setRows(int rows)`	Sets the number of rows.
`int getScrollbarVisibility()`	Gets an enumerated value that indicates which scroll bars the text area uses.
`void insert(String str, int pos)`	Inserts the specified text at the specified position in this text area.
`public String getText()`	Gets the text that is presented by this text component. (Inherited from `java.awt.TextComponent`)
`public void setText(String t)`	Sets the text that is presented by this text component to be the specified text. (Inherited from `java.awt.TextComponent`)

9 The `Checkbox`

A check box is a graphical component that can be in either an "on" (`true`) or "off" (`false`) state. Clicking on a check box changes its state from "on" to "off," or from "off" to "on."

Constructor	Explanation
`Checkbox()`	Creates a check box with no label.
`Checkbox(String label)`	Creates a check box with the specified label.
`Checkbox(String label, boolean state)`	Creates a check box with the specified label and sets the specified state.
`Checkbox(String label, CheckboxGroup group, boolean state)`	Constructs a check box with the specified label, set to the specified state, and in the specified check box group.

Several check boxes can be grouped together under the control of a single object, using the `CheckboxGroup` class. In a check box group, at most one button can be in the "on" state at any given time. Clicking on a check box to turn it on forces any other check box in the same group that is on into the "off" state.

The `CheckboxGroup` class is a simple class:

Method	Explanation
`String getLabel()`	Gets the label of this check box.
`void setLabel(String label)`	
`boolean getState()`	Determines whether this check box is in the "on" or "off" state.
`void setState(boolean state)`	Sets the state of this check box to the specified state.
`CheckboxGroup getCheckboxGroup()`	Determines this check box's group.
`void setCheckboxGroup(CheckboxGroup g)`	Sets this check box's group to be the specified check box group.

Constructor	Explanation
`CheckboxGroup()`	Creates a new instance of `CheckboxGroup`.
Method	**Explanation**
`Checkbox getSelectedCheckbox()`	Gets the current choice from this check box group.
`void setSelectedCheckbox(Checkbox box)`	Sets the currently selected check box in this group to be the specified check box.

10 The `Choice`

The `Choice` class presents a pop-up menu of choices. The current choice is displayed as the title of the menu.

Constructor	Explanation
`Choice()`	Creates a new choice menu.
Method	**Explanation**
`void add(String item)`	Adds an item to this Choice menu.
`void addItem(String item)`	Adds an item to this Choice.
`String getItem(int index)`	Gets the string at the specified index in this Choice menu.
`int getItemCount()`	Returns the number of items in this Choice menu.
`int getSelectedIndex()`	Returns the index of the currently selected item.
`String getSelectedItem()`	Gets a representation of the current choice as a string.
`void insert(String item, int index)`	Inserts the item into this choice at the specified position.
`void remove(int position)`	Removes an item from the choice menu at the specified position.
`void remove(String item)`	Remove the first occurrence of item from the Choice menu.
`void removeAll()`	Removes all items from the choice menu.
`void select(int pos)`	Sets the selected item in this Choice menu to be the item at the specified position.
`void select(String str)`	Sets the selected item in this Choice menu to be the item whose name is equal to the specified string.

11 The `List`

The List component presents the user with a scrolling list of text items. The list can be set up so that the user can choose either one item or multiple items.

Constructor	Explanation
`List()`	Creates a new scrolling list.
`List(int rows)`	Creates a new scrolling list initialized with the specified number of visible lines.
`List(int rows, boolean multipleMode)`	Creates a new scrolling list initialized to display the specified number of rows.

Method	Explanation
`void add(String item)`	Adds the specified item to the end of scrolling list.
`void add(String item, int index)`	Adds the specified item to the the scrolling list at the position indicated by the index.
`void deselect(int index)`	Deselects the item at the specified index.
`String getItem(int index)`	Gets the item associated with the specified index.
`int getItemCount()`	Gets the number of items in the list.
`String[] getItems()`	Gets the items in the list.
`int getRows()`	Get the number of visible lines in this list.
`int getSelectedIndex()`	Gets the index of the selected item on the list.
`int[] getSelectedIndexes()`	Gets the selected indexes on the list.
`String getSelectedItem()`	Get the selected item on this scrolling list.
`String[] getSelectedItems()`	Get the selected items on this scrolling list.
`boolean isIndexSelected(int index)`	Determines if the specified item in this scrolling list is selected.
`boolean isMultipleMode()`	Determines whether this list allows multiple selections.
`void remove(int position)`	Remove the item at the specified position from this scrolling list.
`void remove(String item)`	Removes the first occurrence of an item from the list.
`void removeAll()`	Removes all items from this list.
`void replaceItem(String newValue, int index)`	Replaces the item at the specified index in the scrolling list with the new string.
`void select(int index)`	Selects the item at the specified index in the scrolling list.
`void setMultipleMode(boolean b)`	Sets the flag that determines whether this list allows multiple selections.