GUI Programming with Java

More components and layout managers

- We will look at…

- Password fields
- Combo Boxes
- Lists
- Radio Buttons
- Check boxes
- Layout managers

More SWING Components

SWING so far….

- To date we have examined the history and development motivations for the SWING classes. We have compared SWING to AWT and looked at the advantages SWING offers software developers.
- We have examined processes for creating windows (called frames in JAVA) and accessing the content pane for windows.
- We have looked briefly at panels and motivations for grouping and laying out components using panel sets.
- Finally we have examined a number of basic SWING GUI components including
  - JFrame, JPanel, JLabel, JButton and JTextField
- In this session we will also look at layout managers which allow us to effectively layout GUI components on screen.

What's next in SWING

- We will now examine a number of SWING components which we will encounter as developers.
- We will not be examining all components available to us in SWING. In fact we could spend several weeks studying them and still not have them 100% covered. It is intended that these introduction sessions to SWING will give you a grounding in the libraries use and available classes.
- Components that we will examine in this session include

Password Fields	A field that can be used to enter hidden passwords.
Combo Boxes	A simple list of items from which the user can choose.
List's	Basically performs the same function as a combo box but enables the user to choose a single value or multiple values.
Check Boxes	A GUI component that is either selected or not selected (true / false).
Radio Buttons	A radio button (also know as option buttons) enable you to choose a single item from a list of choices.

Password Fields

- The JPasswordField is a JTextField that refuses to display its contents openly.
- By default, the mask character is the asterisk ('*'). However, you can change this with the setEchoChar() method.
- sample 1 (Password.JAVA) in the samples folder is an example of using password fields please refer to lecture 3 sample programs folder.
- To create a new password field we use the code:
  JPasswordField pass1 = new JPasswordField(20);
- To set the character that is displayed when we press a key we can use the code pass1.setEchoChar ('?');
- **Question:** How can we access the text that the user enters in the password field?

Combo Box

- A combo box, also known as a choice, it's a simple list of items from which the user can choose. It is useful in limiting a users range of choices and avoids the cumbersome validation of input data.
- sample 2 (Combo.JAVA) in the samples folder for this lecture, is an example of using a combo box (there is also a Java 1.7+ version).
- *A new combo box can be created using the standard new operator. e.g.* *JComboBox combo1 = new JComboBox(); //Pre Java 1.7 syntax!*
- An item can be added to a Combo Box by using the addItem method. e.g. combo1.addItem (choices[i]);
- We can allow users to type suggested entries by using the setEditable method. e.g. combo2.setEditable(true);
- We can control the number of rows that are displayed by using the setMaximumRowCount method. e.g. combo2.setMaximumRowCount(4);

List

- A list is a component that basically performs the same function as a combo box but enables the user to choose a single value or multiple values. It is a very powerful component and its full description is beyond the range of this course. However I do encourage students to refer to the excellent API docs and experiment with this component

- The easiest way to create a list (pre Java 1.7) is to create an identifier and using the new operator and pass it an array of list choices.

```
JList list = new JList(choices); //choices is an array of items
```

- If we want our list to scroll we must add it to a Scroll Pane (JScrollPane). The Scroll Pane containing the list should then be added to the container object (content pane or panel etc).

```
JScrollPane pane = new JScrollPane(list);
c.add(pane);
```

- Please take a look at sample 3 (SimpleList.java) for an example of creating a simple list from an array.

List (2)

- The following properties of JList may be very useful.

selectedIndex	An int value indicating the index of the selected item in the list.
selectedIndices	An array of int values representing the indices of the selected items in the list.
selectedValue	The first selected value in the list
selectedValues	An array of objects representing selected values in the list.
visibleRowCount	The preferred number of rows in the list that can be displayed without a scrollbar. The default value is 8.

Note for changes since Java 1.7

- For some components, e.g., the JList and the JComboBox since Java 1.7 the way you declare these components has changed from older versions
- If you have an older version of Java installed you should upgrade to Java 1.7 at least (later releases are already in mainstream use)
- Examples Combo_Java_1_7.java and SimpleList_1_7.java show examples of the changes as of Java 1.7 for some components
- Essentially the difference is in the instantiation of the objects, you must include the type within the angle brackets <>, as follows:

```
JComboBox<String> myCombo = new JComboBox<String>();


JList<String> myList = new JList<String>();
```

Check Box

- A check box is a component that enables the user to toggle a choice on or off (true or false), like a light switch.
- Although Swing provides a default graphic to signify JCheckBox selection, you also can specify your own Icon objects for both the checked and unchecked state.
- sample 4 (Check.java) in the samples folder for this lecture, is an example of using check boxes.
- Sample 5 (CheckBoxTest.java) is an advanced sample demonstrating the use of checkboxes. Some features of this program (event handling) have not been covered yet. Can you figure out what the code does at this stage? (don't worry we'll cover this in later lectures – Event Handling).

Check Box (2)

- The easiest way to set up a check box is via its constructor. To set up a check box which is initially in an unchecked state, we use the following code `JCheckBox bold = new JCheckBox("Bold");`
- The above line of code created an object reference called bold which is references an instance of JCheckbox. The title of the checkbox is passed to the constructor (“Bold”).
- To set up a check box which has an initial state (unchecked or checked state), we use the following code

  `JCheckBox plain = new JCheckBox(“Plain”,true);`
- The above line of code created an object reference called plain which references an instance of JCheckBox. The title of the checkbox is passed to the constructor. As well as the title we pass a true or false (Boolean) value to the constructor to indicate the initial state of the checkbox (true = checked / false = unchecked).

Radio Button

- Radio buttons also known as option buttons, enable you to choose a single item from a list of choices.
- In appearance radio buttons resemble check boxes with the exception that instead of being square and ticked/empty radio buttons are round and are filled or empty.
- The methods for creating a radio button are very similar to check boxes. Please see sample 6(basic) and sample 7(demonstrates grouping) and sample 8(advanced) for examples of applications using radio buttons.
- To set up a radio button which has an initial state (unchecked or checked state), we use the following code JRadioButton plain = new JCheckBox("Plain",true);
- The above line of code created an object called plain which is an instance of JRadioButton. The title of the radio button is passed to the constructor. As well as the title we pass a true or false (Boolean) value to the constructor to indicate the initial state of the radio button (true = checked / false = unchecked).

Radio Button (2)

- Radio buttons are similar to check boxes in that they have two states.
- However radio buttons normally appear as a group in which only one radio button can be selected at a time.
- Selecting other radio buttons in the group automatically forces other radio buttons in the group to be deselected.
- Radio buttons are used to represent a set of mutually exclusive options.
- The logical relationship between radio buttons is maintained by a ButtonGroup object (package javax.swing). It is not a GUI component, rather an object used to logically group them.
- For an example of ButtonGroup usage lets take a look at sample 7 - RadioGroup.java.
- A new Button group can be created using the new operator and buttons added to the group using the add() method.

Layout Managers

Layout Managers

- Java applications and applets use layout managers to arrange display components, e.g. buttons, text fields, lists, etc.
- Every container, by default, has a layout manager .
- If a container's default layout manager doesn't suit your needs, you can easily replace it with another one.
- The Java platform supplies layout managers that range from the very simple (FlowLayout and GridLayout) to the special purpose (BorderLayout and CardLayout) to the very flexible (GridBagLayout and BoxLayout).
- This part of the lecture gives you an overview of some layout managers that the Java platform provides, gives you some general rules for using layout managers, and then tells you how to use each of the provided layout managers. It also points to examples of using each layout manager.

General Rules for Using Layout Managers

- As a rule, the only time you have to think about layout managers is when you create a JPanel or add components to a content pane.
- If you don't like the default layout manager that a panel or content pane uses, then you can change it to a different one.
- When you add components to a panel or content pane, the arguments you specify to the add method depend on the layout manager that the panel or content pane is using.
- The other Swing containers are more specialized and tend to hide the details of which layout manager (if any) they use. For example, a scroll pane relies on a layout manager named ScrollPaneLayout, but you don't need to know that to use a scroll pane.

How to Choose a Layout Manager

- Layout managers have different strengths and weaknesses.
- **Scenario:** You need to display a component in as much space as it can get.
  - Consider using BorderLayout or GridBagLayout.
- **Scenario:** You need to display a few components in a compact row at their natural size.
  - Consider using a JPanel to group the components and using either the JPanel's default FlowLayout manager or the BoxLayout manager.
- **Scenario:** You need to display a few components of the same size in rows and columns.
  - GridLayout is perfect for this.
- **Scenario:** You need to display a few components in a row or column, possibly with varying amounts of space between them, custom alignment, or custom component sizes.
  - BoxLayout is perfect for this.
- **Scenario:** You have a complex layout with many components.
  - Consider either using GridBagLayout or grouping the components into one or more JPanels to simplify layout. Each JPanel might use a different layout manager.

How to Create a Layout Manager and associate it with a Container

- Each container either has a layout manager or uses absolute positioning. All JPanel objects are initialized to use a FlowLayout. Content panes (the main containers in JApplet, JDialog, and JFrame objects) use BorderLayout, by default.

- If you want to use a container's default layout manager, you don't have to do a thing. The constructor for the container creates a layout manager instance and initializes the container to use it.

```
myPanel.setLayout(new BorderLayout());
```

- Here is an example of making a FlowLayout object the layout manager for the content pane:

```
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
```

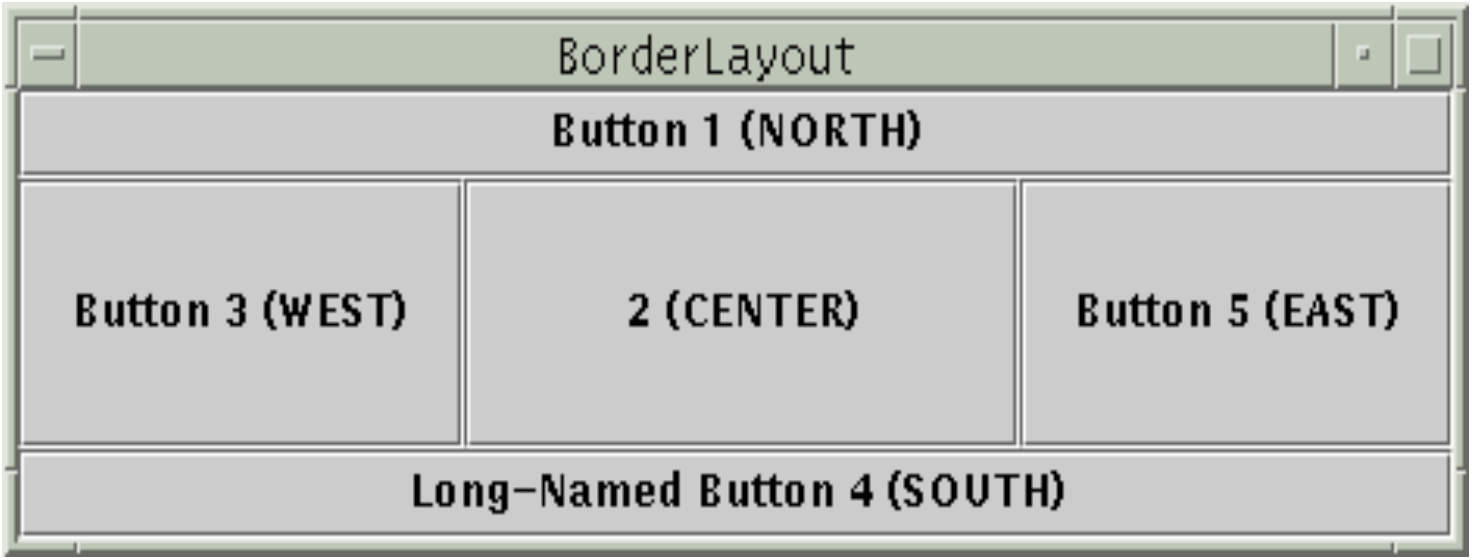
When Is a Layout Manager Consulted?

- A container's layout manager is automatically consulted each time the container might need to change its appearance.
- If you change the size of a component, even indirectly by changing its font, for example, the component should automatically resize and repaint itself.
- In this lecture we will start our look at layout managers with a look at three basic layout managers

1. How to Use BorderLayout
2. How to Use FlowLayout
3. How to Use GridLayout

How to Use BorderLayout

- BorderLayout is the default layout manager for every content pane.
- A BorderLayout has five areas available to hold components: north, south, east, west, and center. All extra space is placed in the center area.
- A sample application with five buttons is shown below. The buttons have been laid out using the Borderlayout manager.



How to Use BorderLayout(2)

- BorderLayout has five areas: north, south, east, west, and center.
- If you enlarge the window, the center area gets as much of the available space as possible.
- The other areas expand only as much as necessary to fill all available space.
- Often, a container uses only one or two of the areas of the BorderLayout -- just the center, or center and south, for example.
- For an example of an application that uses border layout please refer to sample 9 – borderlayout.java.

How to Use BorderLayout(3)

- All our examples that use BorderLayout specify the component as the first argument to the add method. For example:

  add(component, BorderLayout.CENTER) //preferred

- However, you might see code in other programs that specifies the component second. For example, the following are alternate ways of writing the preceding code:

  add(BorderLayout.CENTER, component) //valid but old-fashioned

  or

  add("Center", component) //valid but error prone

The BorderLayout API

- By default, a BorderLayout puts no gap between the components it manages.
- You can specify gaps (in pixels) using the following constructor:

  BorderLayout(int *horizontalGap*, int *verticalGap*)

- You can also use the following methods to set the horizontal and vertical gaps, respectively:

  void setHgap(int) void setVgap(int)

How to Use FlowLayout

- FlowLayout is the default layout manager for every JPanel.
- It simply lays out components from left to right, starting new rows if necessary.

- FlowLayout puts components in a row, sized at their preferred size.
- If the horizontal space in the container is too small to put all the components in one row, FlowLayout uses multiple rows.
- Within each row, components are centered (the default), left-aligned, or right-aligned as specified when the FlowLayout is created.

How to Use FlowLayout(2)

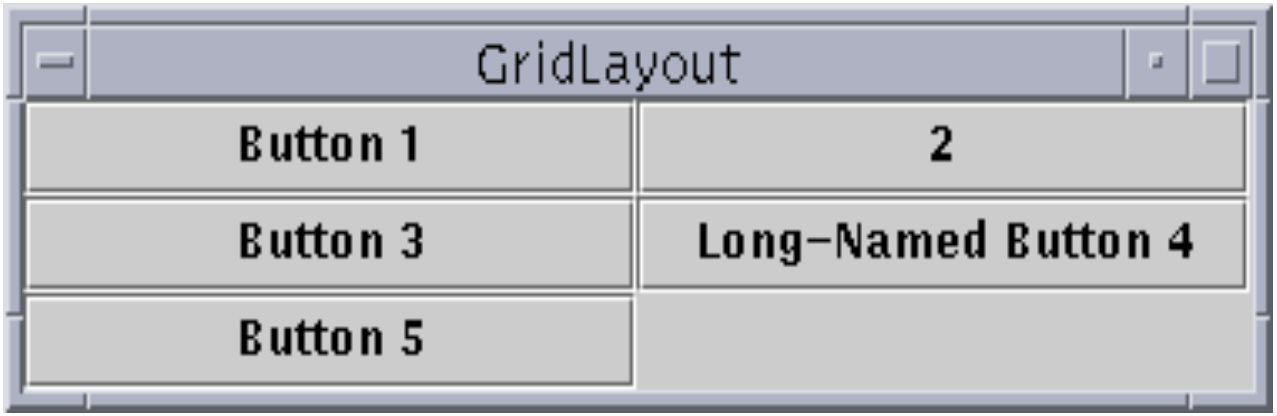
- For an example of an application that uses border layout please refer to sample 10 – FlowWindow.java.
- The FlowLayout class has three constructors:

  public FlowLayout()
  public FlowLayout(int *alignment*)
  public FlowLayout(int *alignment*, int *horizontalGap*, int *verticalGap*)

- The *alignment* argument must have the value FlowLayout.LEFT, FlowLayout.CENTER, or FlowLayout.RIGHT.
- The *horizontalGap* and *verticalGap* arguments specify the number of pixels to put between components.
- If you don't specify a gap value, FlowLayout uses 5 for the default gap value.

How to Use GridLayout

- GridLayout simply makes a bunch of components equal in size and displays them in the requested number of rows and columns.



- A GridLayout places components in a grid of cells.
- Each component takes all the available space within its cell, and each cell is exactly the same size.
- If you resize the GridLayout window, you'll see that the GridLayout changes the cell size so that the cells are as large as possible, given the space available to the container.

How to Use GridLayout(2)

- For an example of an application that uses grid layout please refer to sample 11 – GridWindow.java.
- The GridLayout class has two constructors:
- public GridLayout(int *rows*, int *columns*) public GridLayout(int *rows*, int *columns*, int *horizontalGap*, int *verticalGap*)
- At least one of the *rows* and *columns* arguments must be nonzero.
- The *horizontalGap* and *verticalGap* arguments to the second constructor allow you to specify the number of pixels between cells.
- If you don't specify gaps, their values default to zero.

Summary

- The JPasswordField is a JTextField that refuses to display its contents openly.
- A combo box, also known as a choice, is a simple list of items from which the user can choose. It is useful in limiting a users range of choices and avoids the cumbersome validation of input data.
- A list is a component that basically performs the same function as a combo box but enables the user to choose a single value or multiple values.
- A check box is a component that enables the user to toggle a choice on or off (true or false), like a light switch.
- Radio buttons also known as option buttons, enable you to choose a single item from a list of choices.

Summary (2)

- Java applications and applets use layout managers to arrange display components, e.g. buttons, text fields, lists, etc.
- As a rule, the only time you have to think about layout managers is when you create a JPanel or add components to a content pane .
- If you don't like the default layout manager that a panel or content pane uses, then you can change it to a different one.
- A container's layout manager is automatically consulted each time the container might need to change its appearance.
- In this session we examined three basic layout managers

  BorderLayout, FlowLayout, GridLayout

END OF SESSION