CardLayout

- The class **CardLayout** arranges each component in the container as a card.

- *Only one card is visible at a time*, and the container acts as a stack of cards.

- **Constructors:**
  - **CardLayout()**

    -Creates a new card layout with gaps of size zero.
  - **CardLayout(int hgap, int vgap)**

    -Creates a new card layout with the specified horizontal and vertical gaps.

CardLayout

- Cards are typically held in an object of type **Panel.**

- Panel must have **CardLayout** selected as its layout manager.

- **For Add component:**
  - void add(Component *panelObj, Object name);*

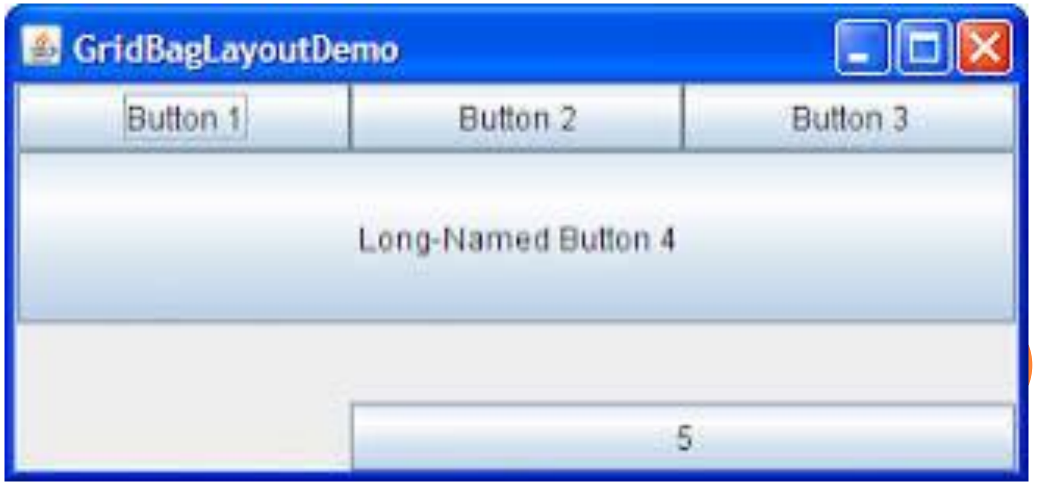
- **Methods:**
  - void first(**Container *deck***)
  - void last(**Container *deck***)
  - void next(**Container *deck***)
  - void previous(**Container *deck***)
  - void show(**Container *deck***, ***String cardName***)

GridBagLayout

- **GridBagLayout** is one of the *most flexible — and complex —* layout managers the Java platform provides.

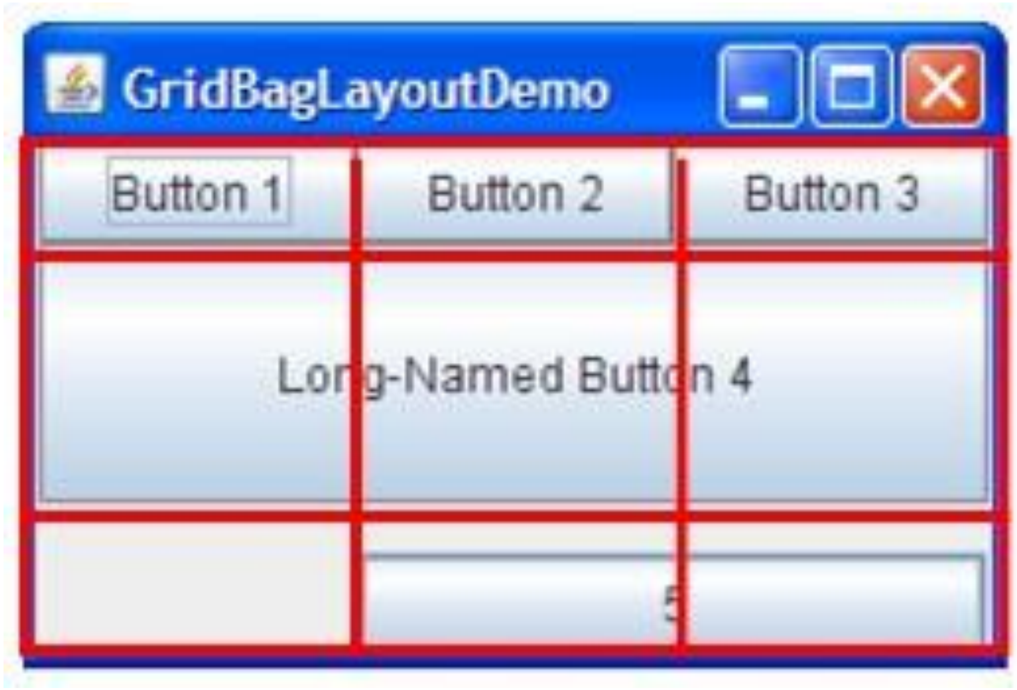
- A **GridBagLayout** places components in a grid of rows and columns, allowing specified components to span multiple rows or columns.

- Essentially, **GridBagLayout** places components in rectangles (cells) in a grid, and then uses the components' preferred sizes to determine how big the cells should be.



GridBagLayout

- The following figure shows the grid for the preceding applet.
- As you can see, the grid has three rows and three columns.
- The button in the second row spans all the columns; the button in the third row spans the two right columns.



GRIDBAGLAYOUT

- Each **GridBagLayout** object maintains a dynamic rectangular grid of cells, with each component occupying one or more cells, called its *display area*.

- Each component managed by a grid bag layout is associated with an instance of **GridBagConstraints** that specifies how the component is laid out within its display area.

- Maximum capacity of a screen using **GridBagLayout** in
  - Java 1.0 is **128 128 cells.**
  - Java 1.1 is **512 512 cells.**

- ***Constructor:***
  - **GridBagLayout()**

GridBagLayout

Field	Purpose
int anchor	Specified the location of a component within a cell. The default is **GridBagConstraints.CENTER**
int fill	Specifies how a component is resized if the component is smaller than its cell. Valid values are **GridBagConstraints.NONE(the default)** **GridBagConstraints.HORIZONTAL** **GridBagConstraints.VERTICAL** **GridBagConstraints.BOTH**
int gridheight	Specified the height of the component in terms of cells. **The default is 1.**
int gridwidth	Specified the width of the component in terms of cells. **The default is 1.**
int gridx	Specifies the X coordinate of the cell to which the component will be added. **GridBagConstraints.RELATIVE**
int gridy	Specifies the Y coordinate of the cell to which the component will be added.

GridBagConstraints

- When a component is smaller than its cell, you can use the anchor field to specify where within the cell the components top-left corner will be located.

- There are *three types* values that you can give to a anchor.

GridBagConstraints

- **absolute** values are:
  - **GridBagConstraints.CENTER**
  - **GridBagConstraints.SOUTH**
  - **GridBagConstraints.SOUTHEAST**
  - **GridBagConstraints.SOUTHWEST**
  - **GridBagConstraints.EAST**
  - **GridBagConstraints.NORTH**
  - **GridBagConstraints.NORTHEAST**
  - **GridBagConstraints.NORTHWEST**
  - **GridBagConstraints.WEST**

GridBagConstraints

- The second type of values that can be given to anchor field is **relative**

- **GridBagConstraints.FIRST_LINE_END**
- **GridBagConstraints.LINE_END**
- **GridBagConstraints.FIRST_LINE_START**
- **GridBagConstraints.LINE_START**
- **GridBagConstraints.LAST_LINE_END**
- **GridBagConstraints.PAGE_END**
- **GridBagConstraints.LAST_LINE_START**
- **GridBagConstraints.PAGE_START**

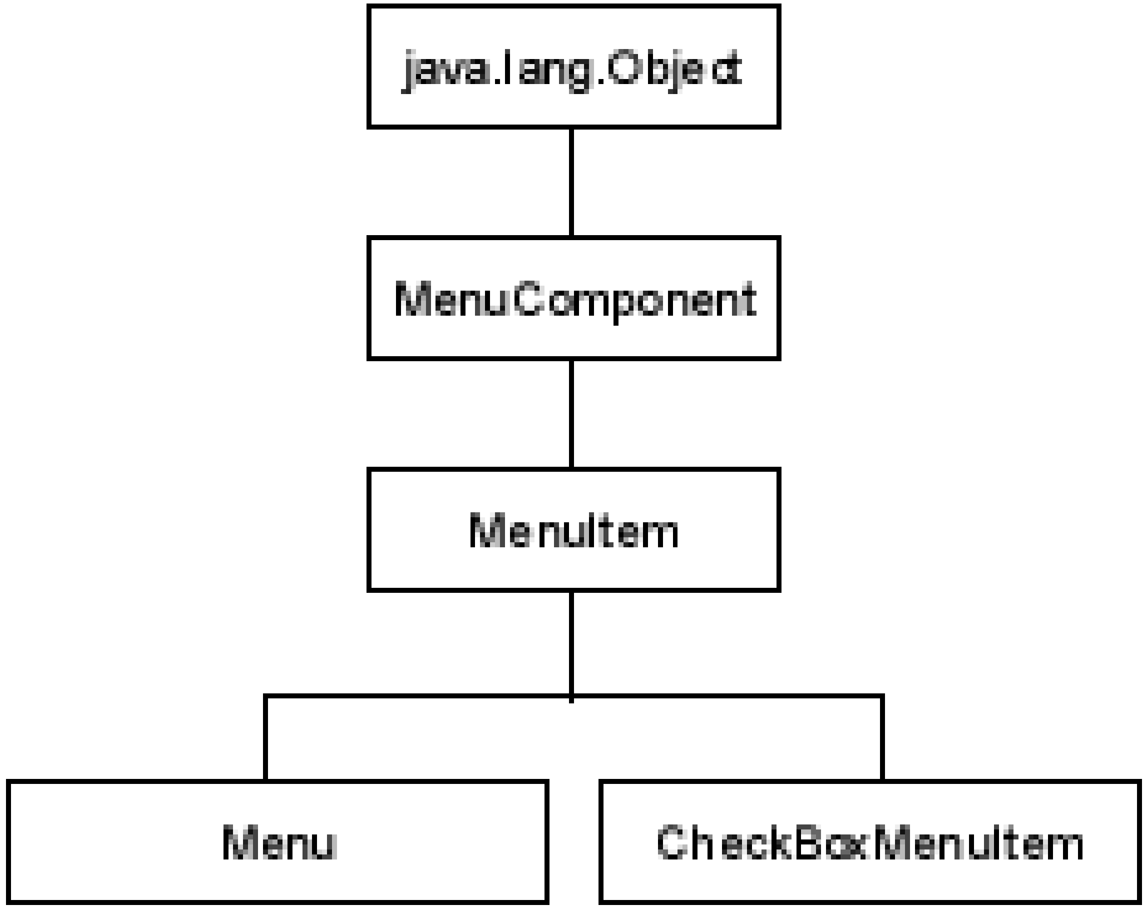
GridBagConstraints

- The third type of values**(describes placement)** that can be given to anchor allows you to position the components relative to the **baseline** of the row
  - **GridBagConstraints.BASELINE**
  - **GridBagConstraints.BASELINE_LEADING**
  - **GridBagConstraints.BASELINE_TRAILING**
  - **GridBagConstraints.ABOVE_BASELINE**
  - **GridBagConstraints.BELOW_BASELINE**

GridBagConstraints

- For customize a **GridBagConstraints** object by setting one or more of its instance variables:
  - **gridx, gridy:**
    - Specifies the cell at the upper left of the component's display area, where the upper-left-most cell has address gridx = 0, gridy = 0.
  - **gridwidth, gridheight:**
    - Specifies the number of cells in a row (for gridwidth) or column (for gridheight) in the component's display area. The default value is 1.
  - **fill:**
    - Used when the component's display area is larger than the component's requested size to determine whether (and how) to resize the component.

MenuBar *AND* Menu



MenuBar *AND* Menu

- Top-level window can have a menu bar associated with it.
- A menu bar displays a list of top-level menu choices.
- Each choice is associated with a drop-down menu.
- **Classes:**
  - **MenuBar** : Contains one or more Menu objects
  - **Menu** : Contains one or more MenuItem objects
  - **MenuItem** : Object something selected by user.
- Since ***Menu is a subclass of MenuItem***, a hierarchy of nested submenus can be created.

MenuBar ***AND*** Menu

- It is also possible to include checkable menu items

- These are menu options of type **CheckboxMenuItem** and will have a check mark next to them when they are selected.

MenuBar *AND* Menu

- To create a menu bar, first create an instance of **MenuBar** **(Only default Constructor is available).**
- Set **MenuBar** using **setMenuBar**(**MenuBarObject**)
- Next, create instances of **Menu that will define the selections displayed on the bar.**
- **Constructors:**
  - **Menu( )**
  - **Menu(String *optionName*)**
  - **Menu(String *optionName, boolean removable*)**
- **Individual menu items constructors:**
  - **MenuItem( )**
  - **MenuItem(String *itemName*)**
  - **MenuItem(String *itemName, MenuShortcut keyAccel*)**

MenuBar ***AND*** Menu

- **Disable or enable a menu item by using:**
  - void setEnabled(boolean *enabledFlag)*
  - boolean isEnabled( )

- **Label set and get using:**
  - void setLabel(String *newName)*
  - String getLabel( )

- Checkable menu item by using a subclass of **MenuItem called CheckboxMenuItem:**
  - CheckboxMenuItem( )
  - CheckboxMenuItem(String *itemName)*
  - CheckboxMenuItem(String *itemName, boolean on)*

MenuBar *AND* Menu

- **Status about checkable MenuItem:**
  - boolean getState( )
  - void setState(boolean *checked)*
- **For add MenuItem:**
  - MenuItem add(MenuItem *item)*
- **For add Menu**
  - Menu add(Menu *menu)*
- **To get Item from Menu:**
  - Object getItem( )

MenuBar ***AND*** Menu (Event handling)

- Menus generate events only when an item of type **MenuItem** or **CheckboxMenuItem** is selected.
- They do not generate events when a menu bar is accessed to display a drop-down menu, for example.
- Each time a menu item is selected, an **ActionEvent** object is generated.
- However, you can specify a different action command string by calling **setActionCommand( )** on the menu item.

MenuBar ***AND*** Menu (Event handling)

- Each time a check box menu item is checked or unchecked, an **ItemEvent** object is generated.
- Thus, you must implement the **ActionListener** and/or **ItemListener** interfaces in order to handle these menu events.
- The **getItem( )** method of **ItemEvent** returns a reference to the item that generated this event.
- The general form of this method is shown here:

  Object getItem( )

Dialog Box

- **Dialog boxes** are primarily used to *obtain user input*.
- They are similar to frame windows, except that dialog boxes are always child windows of a top-level window.
- *Dialog boxes don't have menu bars.*
- In other respects, dialog boxes function like frame windows.
- Dialog boxes may be ***modal or modeless.***
  - **When a *modal dialog box is active***, *all input is directed to it until it is closed.*
  - **When a *modeless dialog box is active***, *input focus can be directed to another window in your program.*

DIALOG BOX

- ***Constructors:.***
  - Dialog(Frame *parentWindow, boolean mode)*
  - Dialog(Frame *parentWindow, String title, boolean mode)*

- **To create Dialog Box:**
  - Create Frame or Applet
  - Create another class which extends Dialog class.
  - Call this new class from Frame/Applet class.
  - In constructor of Extended Dialog class, use super method and pass values to constructor of Dialog.

File Dialog

- Java provides a built-in dialog box that lets the user specify a file.

- To create a file dialog box, instantiate an object of type **FileDialog.**

- **Constructor:**
  - FileDialog(Frame *parent, String boxName)*
  - FileDialog(Frame *parent, String boxName, int how)*
  - FileDialog(Frame *parent)*

- If *how* is **FileDialog.LOAD**, then the box is selecting a file for reading.
- If *how* is **FileDialog.SAVE**, the box is selecting a file for writing.
- If *how* is omitted, the box is selecting a file for reading.

- ***Methods:***
  - ***String getDirectory( )***
  - ***String getFile( )***