IPOO

- Docentes
  - Profesores Adjuntos:
    - Carlos Di Cicco.
    - Diego de la Riva.
  - JTPs:
    - Federico Naso (Junín) .
      Nelson Di Grazia (Pergamino) .
  - Ayudante:
    - Sebastián Sottile (Junín).

Aplicaciones GUI en Java

Tecnologías GUI de Java

- AWT (Abstract Window Toolkit)
- JFC (Java Foundation Classes). Mejor conocida como Swing
- SWT (Standard Widget Toolkit)

Programación JFC (Swing)

- Frames
  - JFrame
  - JDialog
- Panels
  - Layouts
  - Boxes
- Componentes (widgets)
  - JComponent
  - JLabel
  - JTextField
  - JButton
  - JList

Aplicaciones GUI en Java

- Para crear una aplicación con Swing se debe:
  - Crear un JFrame
  - Llenarlo de components según los requerimientos de la aplicación
  - Mostrar el JFrame en pantalla invocando el método `setVisible(true)`
- Ejemplo
  - `public static void main(String[] args)`
  - `{`
  - `    JFrame frame = new JFrame();`
  - `    frame.setVisible(true);`
  - `}`

Aplicaciones GUI en Java

- Se acostumbra (pero no es obligatorio) declarar una subclase de JFrame y en el constructor llenar el Frame de componentes

```
public class FrameAlumnos extends JFrame {

    JTextField nombre;

    JTextField fechaNac;

    FrameAlumnos() {

        JPanel contentPane = (JPanel) getContentPane();

        nombre = new JTextField();

        contentPane.add(nombre);

        FechaNac = new JtextField();

        contentPane.add(fechaNac);

    }
```

Ubicación de componentes en un Frame.
Layout

- La clase JPanel es un contenedor de objetos que pueden ser desplegados
- Un JFrame tiene un panel principal que se obtiene invocando `getContentPane()`
  - `JFrame frame = new JFrame();`
  - `JPanel contentPane = (JPanel) frame.getContentPane();`
  - `. . .`
- Un panel puede contener componentes finales (JLabel, JTextField, etc.) u otros paneles (Jpanel)
- Esto permite acomodar las cosas en el Frame
- Se puede utilizar posicionamiento absoluto (x, y) pero esto no es recomendable

Administrador de disposición

¿Debo darle un tamaño a cada componente y un lugar dentro de la pantalla?

No necesariamente!!!

La ubicación y tamaño de un componente en un contenedor están determinados por el administrador de disposición o Layout . Cada contenedor mantiene una referencia a una instancia de un Layout .

El Administrador de Disposición toma el control de todos las componentes en el contenedor. El Layout es responsable de calcular el tamaño preferido del objeto en el contexto del tamaño de la pantalla actual. El programador no necesita definir el tamaño y posición de cada uno de los componentes de la ventanas, esa lógica la tiene incorporada el Administrador de Disposición.

Sin embargo, es posible controlar el tamaño de las componentes y su ubicación manualmente. Para hacerlo se debe deshabilitar el administrador para ese contenedor así: `contenedor.setLayout(null);`

Layouts

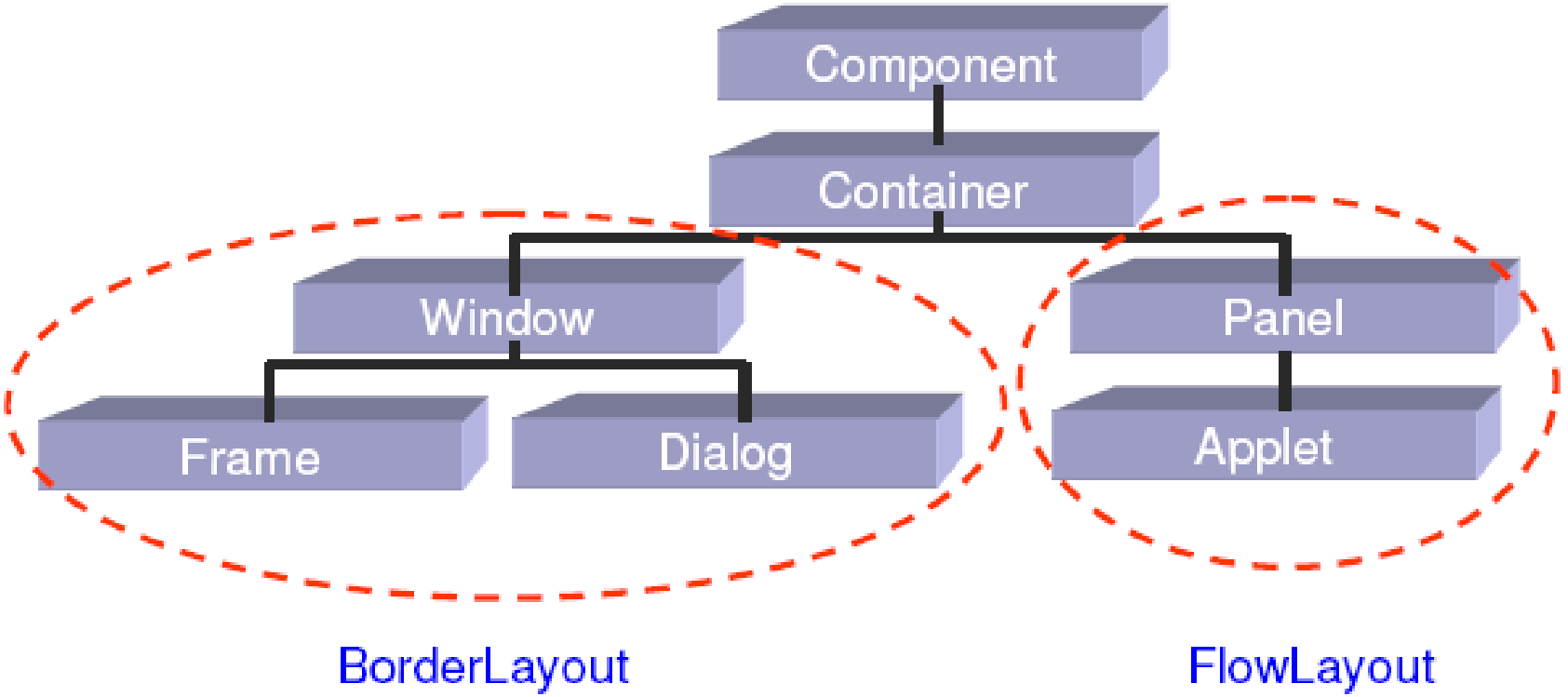
Cada contenedor tienen asociado un **administrador** por defecto, el cual puede cambiarse invocando el método del contenedor **setLayout()**. Entonces, ¿hay diferentes tipos de Layouts?

Existen 5 tipos de administradores de disposición:

- BorderLayout
- CardLayout
- FlowLayout
- GridBagLayout
- GridLayout

Layouts

Cada tipo de **contenedor** tiene asociado un objeto Layout por defecto, que puede cambiarlo utilizando el método **setLayout(nuevoContenedor)**.



Layouts

```
package clase16.ejemplo1;

import java.awt.*;

public class LayoutTest {

    private Frame frame;
    private Button b1;
    private Button b2;

    public LayoutTest() {
        frame = new Frame("GUI");
        b1 = new Button("Presionar");
        b2 = new Button("No Presionar");
    }

    public void LaunchFrame() {
        frame.setLayout(new FlowLayout());
        frame.add(b1);
        frame.add(b2);
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        LayoutTest test = new LayoutTest();
        test.LaunchFrame();
```

Crea tres Objetos: un Frame y dos botones

Se le cambia el Layout por defecto

se agregan dos botones al frame

Este método le indica al frame que defina un tamaño para incluir apropiadamente los componentes que contiene.

Creación de un Frame con dos botones

Layouts: FlowLayout

Este layout ubica a las componentes línea por línea. Cada vez que se completa una línea se comienza una nueva. No restringe el tamaño de las componentes que gestiona y admite que tengan el tamaño preferido.

El alineamiento por defecto que el **administrador** usa para sus componentes es centrado, aunque se puede cambiar para que use alineación a izquierda o a derecha.

Los constructores de FlowLayout son:

```
FlowLayout();
FlowLayout(int align);
FlowLayout(int align, int hgap, int vgap);
```

Los valores para align pueden ser:
FlowLayout.LEFT
FlowLayout.CENTER
FlowLayout.RIGHT

Los parámetros hgap y vgap se utilizan para asignar un espaciado horizontal y vertical entre las componentes

Ejemplo:

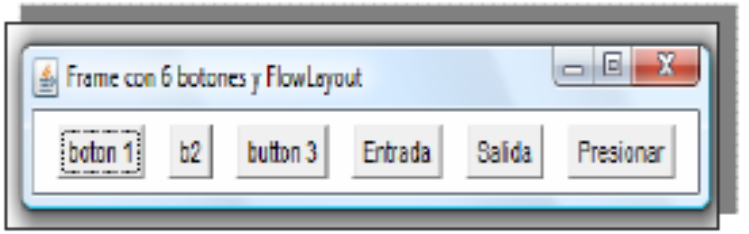
```
contendor.setLayout(new FlowLayout(FlowLayout.CENTER,5,10);
```

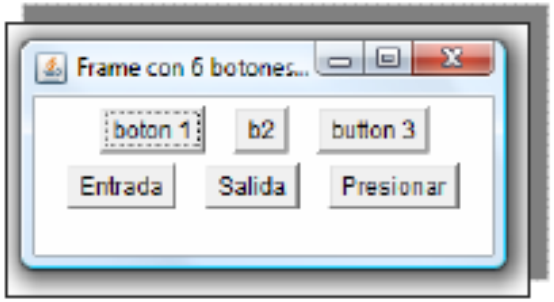
Layouts: FlowLayout. Ejemplo

```
package clase16.ejemplo1;
import java.awt.*;
public class FlowLayoutTest {
    private Frame frame;
    private Button b1,b2,b3,b4,b5,b6;

    public FlowLayoutTest() {
        frame = new Frame("Frame con 6 botones y FlowLayout");
        b1 = new Button("boton 1");
        b2 = new Button ("b2");
        b3 = new Button("button 3");
        b4 = new Button ("Entrada");
        b5 = new Button ("Salida");
        b6 = new Button("Presionar");
    }

    public void launchFrame() {
        frame.setLayout(new FlowLayout(FlowLayout.CENTER,15,5));
        frame.add(b1);
        frame.add(b2);
        frame.add(b3);
        frame.add(b4);
        frame.add(b5);
        frame.add(b6);
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        FlowLayoutTest test = new FlowLayoutTest();
```

Espaciado entre los componentes

Creación de un Frame con FlowLayout





Así queda el frame cuando se le cambia el tamaño durante la ejecución. Siempre se ve bien y mantiene espacio entre componentes.

Layouts: BorderLayout

Este Layout ofrece un esquema complejo para la ubicación de componentes en uncontenedor. Su estructura básica está compuesta por cinco regiones:

BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.EAST,

BorderLayout.WEST, BorderLayout.CENTER.

Layouts: BorderLayout

- Cuando se crea un **BorderLayout,** con el constructor por defecto, no existe espacio entre las regiones que administra.

  New BorderLayout();

- Si queremos que exista un espacio entre los componentes, se debe usar el constructor con 3 argumentos:

  new BorderLayout(hgap,vgap) => new BorderLayot(3,3);

- Cuando queremos agregar una componente en una región determinada, se debe usar el constructor con 2 argumentos:

  new BorderLayot(button,BorderLayout.RIGTH);

- Se puede agregar solo una componente por región. Si se agregan más sólo una quedará visible.

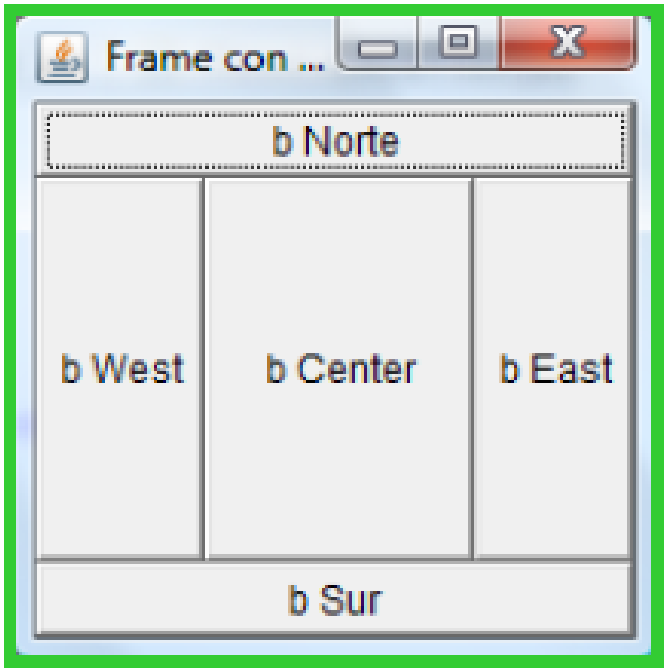
Layouts: BorderLayout. Ejemplo

```
package clase16.ejemplo1;
import java.awt.*;
public class BorderLayoutTest {
    public Frame frame;
    public Button bN,bS,bC,bE,bW;

    public BorderLayoutTest() {
        frame = new Frame("Frame con BorderLayout");
        bN= new Button("b Norte");
        bS = new Button ("b Sur");
        bW = new Button ("b West");
        bE = new Button ("b East");
        bC = new Button ("b Center");
    }
    public void launchFrame() {
        frame.add(bN,BorderLayout.NORTH);
        frame.add(bS,BorderLayout.SOUTH);
        frame.add(bW,BorderLayout.WEST);
        frame.add(bE,BorderLayout.EAST);
        frame.add(bC,BorderLayout.CENTER);
        frame.setSize(200,200);
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        BorderLayoutTest test = new BorderLayoutTest( );
        test.launchFrame();
    }
```

Crea botones con etiquetas

Agrega botones en las distintas regiones

Creación de un Frame con su BorderLayout



No se especificó el administrador de disposición a utilizar ya que **Frame** tiene por **defecto** al **BorderLayout**

Layouts: GridLayout

Este **Layout** ofrece flexibilidad para ubicar componentes en una grilla. En el momento de su creación hay que indicarle la cantidad de filas y columnas que tendrá.

Por ejemplo si queremos que tenga 2 filas y 3 columnas hacemos así:

new GridLayout(2,3);

El ancho de todas las celdas es idéntico y está determinado por el resultado de la división del ancho disponible por la cantidad de columnas. De la misma manera se obtiene el alto de todas las celdas.

El orden en que los componentes se agregan a la grilla determina la celda que se ocupa. Las celdas se ocupan de izquierda a derecha y de arriba hacia abajo.

La siguiente instrucción crea una grilla, la cantidad de celdas está dada por los argumentos rows y cols y el espaciado entre las componentes por hgap y vgap:

new GridLayout(int rows, int cols, int hgap, int vgap);

Layouts: GridLayout

Qué pasa si no se a priori cuántas filas o columnas tendré?

Se puede usar el 0 (cero). La cantidad de columnas o la cantidad de filas puede cero e indica un valor flexible.

Ejemplos:

```
new GridLayout(0,3);
new GridLayout(0,0);
new GridLayout(3,0);
```

→ Indica 3 columnas y filas la cantidad necesaria

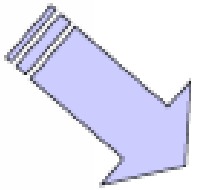
→ Cantidad flexible de filas y columnas

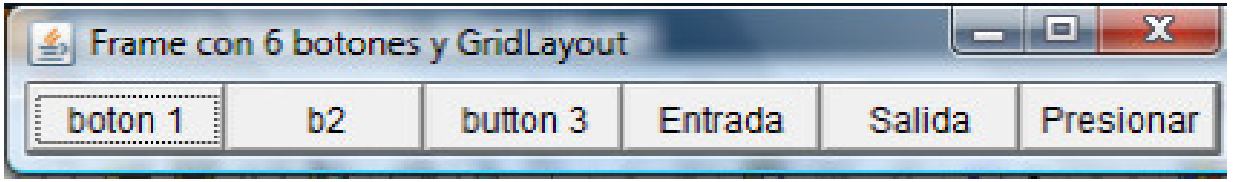
→ Indica 3 filas y cantidad flexible de columnas

Layouts: GridLayout. Ejemplo

```
package clase16.ejemplo1;
import java.awt.*;
public class GridLayoutTest {
    private Frame frame;
    private Button b1,b2,b3,b4,b5,b6;

    public GridLayoutTest() {
        frame = new Frame("Frame con 6 botones y GridLayout");
        b1 = new Button("boton 1");
        b2 = new Button ("b2");
        b3 = new Button("button 3");
        b4 = new Button ("Entrada");
        b5 = new Button ("Salida");
        b6 = new Button("Presionar");
    }

    public void launchFrame() {
        frame.setLayout(new GridLayout());
        frame.add(b1);
        frame.add(b2);
        frame.add(b3);
        frame.add(b4);
        frame.add(b5);
        frame.add(b6);
        frame.pack();
        frame.setVisible(true);
    }
```

Creación de un Frame con un GridLayout



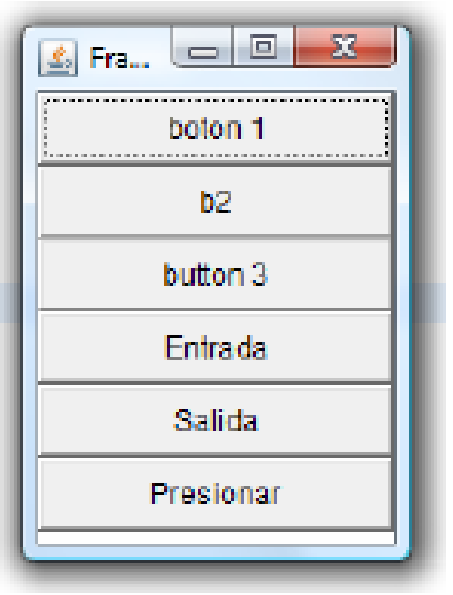


Al no indicarle filas y columnas crea una columna por componente en una sola fila

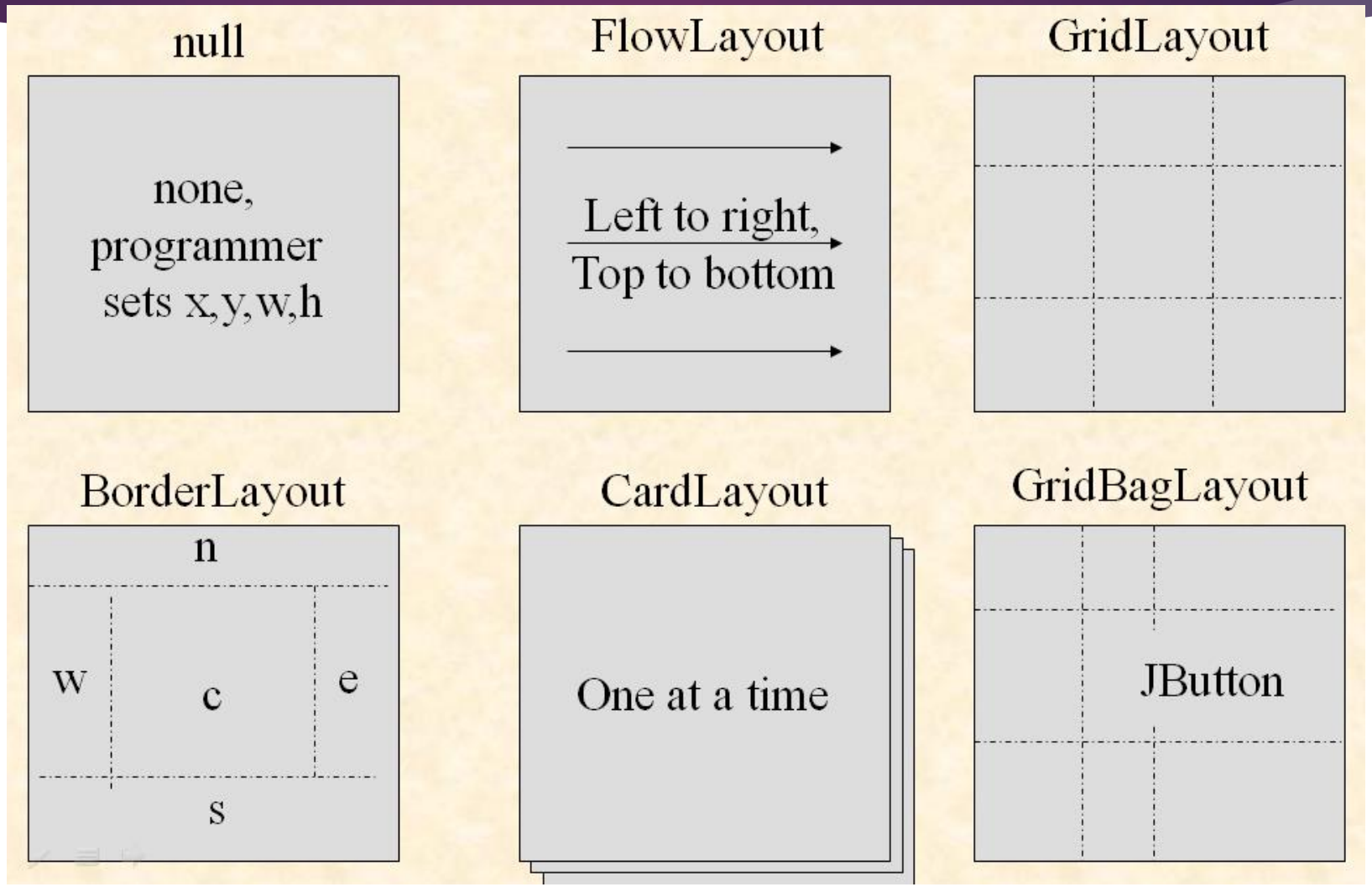
Layouts: GridLayout. Ejemplo

```
package clase16.ejemplo1;
import java.awt.*;
public class GridLayoutTest {
    private Frame frame;
    private Button b1,b2,b3,b4,b5,b6;

    public GridLayoutTest() {
        frame = new Frame("Frame con 6 botones y GridLayout");
        b1 = new Button("boton 1");
        b2 = new Button ("b2");
        b3 = new Button("button 3");
        b4 = new Button ("Entrada");
        b5 = new Button ("Salida");
        b6 = new Button("Presionar");
    }

    public void launchFrame() {
        frame.setLayout(new GridLayout(0,1));
        frame.add(b1);
        frame.add(b2);
        frame.add(b3);
        frame.add(b4);
        frame.add(b5);
        frame.add(b6);
        frame.pack();
        frame.setVisible(true);
```

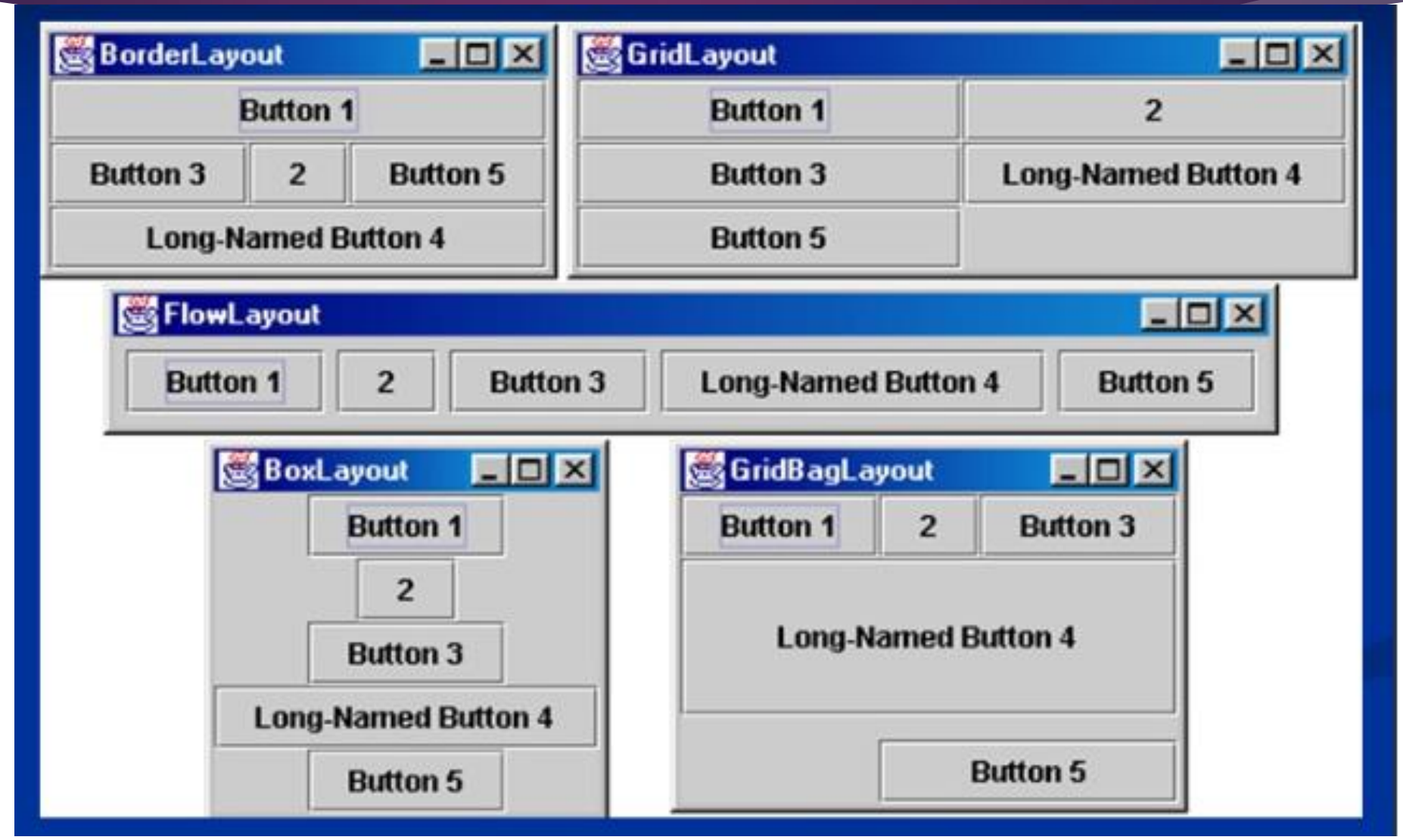
Creación de un Frame con un GridLayout



Layouts


null
none,
programmer
sets x,y,w,h
FlowLayout
Left to right,
Top to bottom
GridLayout
BorderLayout
n
w
c
e
s
CardLayout
One at a time
GridBagLayout
JButton

Layouts


BorderLayout
Button 1
Button 3
2
Button 5
Long-Named Button 4
GridLayout
Button 1
2
Button 3
Long-Named Button 4
Button 5
FlowLayout
Button 1
2
Button 3
Long-Named Button 4
Button 5
BoxLayout
Button 1
2
Button 3
Long-Named Button 4
Button 5
GridBagLayout
Button 1
2
Button 3
Long-Named Button 4
Button 5

Aplicaciones GUI en Java

Eventos

- Swing permite el desarrollo de aplicaciones manejadas por eventos (event driven)
- Un evento es un click del ratón, una tecla oprimida, seleccionar una opción en un menú, etc.
- Para procesar estos eventos una aplicación debe definir unos event listener.
- Un event listener es un objeto que se registra con un componente para un evento en particular.
- Cuando ocurre el evento el listener es notificado mediante la invocación de un método

Aplicaciones GUI en Java

- Para crear un listener la aplicación debe declarar una clase que implementa alguna interfaz listener (listener interface)
- Las interfaces listener más usadas son:
  - ActionListener
  - MouseListener
  - KeyListener
  - WindowListener

Aplicaciones GUI en Java

Ejemplo

```
class EdadKeyListener implements KeyListener {
    void keyTyped(KeyEvent e) {
        char c = e.getKeyChar();
        if (!Character.isDigit(c) {
            // error, solo se permiten números
        }
    }
    void keyPressed(KeyEvent e) {
    }
    void keyReleased(KeyEvent e) {
    }
}
```

Aplicaciones GUI en Java

- Existen unas clases adapter que implementan la interfaz correspondiente y definen métodos vacíos

- Esto da la facilidad de que no hay que implementar los métodos que no interesan

```
class EdadKeyListener extends KeyAdapter {
    void keyTyped(KeyEvent e) {
        char c = e.getKeyChar();
        if (!Character.isDigit(c) {
            // error, solo se permiten números
        }
    }
}
```

Aplicaciones GUI en Java

Ejemplo de uso

- `TextField edad = new TextField(3);`
- `edad.addKeyListener(new EdadKeyListener());`

Aplicaciones GUI en Java

- Cada componente permite agregar un ActionListener que está relacionado con el tipo del componente
- Por ejemplo si a un botón se le agrega un ActionListener el mismo es invocado cuando se oprime el botón ya sea con el mouse o con el teclado.
- No es necesario definir KeyEvents o MouseEvents para esto
- Los ActionListener se usan principalmente con botones y menús
- Sin embargo también pueden ser usados con otros tipos de componente.
- Por ejemplo, para un TextField el ActionListener es invocado cuando el cursor sale del TextField
- Las listas y tablas utilizan unos listeners especiales denominados SelectionListener

Aplicaciones GUI en Java

Ejemplo de un ActionListener

```
JButton btnCancelar = new
JButton("Cancelar");
btnCancelar.addActionListner(new
CancelarListener());


class CancelarListener implements
ActionListener {
    void actionPerformed(ActionEvent e) {
        dialogo.setVisible(false);
    }
}
```

Aplicaciones GUI en Java

Diálogos

- Un diálogo es un frame que permite recolectar datos para realizar algún procesamiento
- En Java existe una clase JDialog para este fin.
- JDialog es subclase de JFrame y permite definir diálogos modales y no modales
- Si un diálogo es modal cuando se activa no se puede acceder a ningún otro elemento del programa
- Si el diálog es modal se abre la ventana (window) del diálogo pero el usuario puede seleccionar y trabajar con otras ventanas de la aplicación
- En Swing si el diálogo es modal el hilo que abre el diálogo se bloquea hasta que el diálogo sea cerrado.

Aplicaciones GUI en Java

- Para crear un diálogo modal se debe especificar en el constructor
  - `JDialog dlg = new Jdialog(frame, "Titulo", true);`
- El tercer parámetro es booleano e indica si el diálogo es modal
- Generalmente se crea una subclase de JDialog:
  - `public class DialogoDatos extends Jdialog {`
  - `    JTextField nombre;`
  - `    ...`
  - `    public DialogoDatos(JFrame frame) {`
  - `        super(frame, "Título", true);`
  - `        ContentPane cp = (ContentPane) getContentPane();`
  - `        ...`
  - `    } }`

Aplicaciones GUI en Java

- Ejemplo de uso
  - `DialogoDatos dlg = new DialogoDatos(this);`
  - `dlg.setVisible(true);`
  - `String nombre = dlg.getNombre();`
  - `...`
- Si el diálogo no es modal el código que sigue a la instrucción continúa ejecutándose en un hilo paralelo.
-
- Por lo tanto la lógica a ejecutar debe colocarse en los métodos de los ActionListener definidos en el diálogo

Aplicaciones GUI en Java

Como dibujar cosas en una ventana

- La clase JComponent es la superclase de la cual se derivan todos los componentes en Swing
- Se pueden crear subclases de JComponent para definir nuevos componentes
- El método `paintComponent(Graphics g)` definido en Jcomponent se usa para dibujar el componente
- Este método es invocado por Swing cada vez que se requiere repintar el componente (por ejemplo, si la ventana estaba debajo de otra y va a pasar a primer plano, o si estaba minimizada y se va restaurar)
- A continuación se presenta la forma de hacer un componente para dibujar figuras geométricas en la pantalla

Aplicaciones GUI en Java

```
public class SuperficieDibujo extends JComponent {
      private int width;
      private int height;
      private Image image;
      private Graphics graphics;
      public SuperficieDibujo(int w, int h) {
            this.width = w;
            this.height = h;
            image = new BufferedImage(w, h,
               BufferedImage.TYPE_INT_RGB);
            graphics = image.getGraphics();
            this.borrar();
      }
```

Aplicaciones GUI en Java

```
	public void dibujarRectangulo(Rectangle rect)
	{
		graphics.setColor(Color.BLACK);
		graphics.drawRect(rect.x, rect.y,
rect.width,
				rect.height);
		this.repaint();
	}

	public void paintComponent(Graphics g)
	{
		g.drawImage(image, 0, 0, null);
	}
```

Aplicaciones GUI en Java

Listas

- La clase JList representa una lista de valores
  - `JList lista = new Jlist();`
- Para que la lista tenga scrollbars se debe colocar dentro de un ScrollPane:
  - `JScrollPane sp = new JScrollPane(lista);`
  - `sp.setPreferredSize(new Dimension(80, 260));`
- Los elementos de una lista se guardan en un objeto de una clase que implementa la interfaz ListModel
- La interfaz ListModel define métodos para obtener los elementos de la lista mediante un índice y para obtener el tamaño de la lista

Aplicaciones GUI en Java

- La clase DefaultListModel implementa ListModel
  - `lista = new JList();`
  - `ListModel listModel = new DefaultListModel();`
  - `lista.setModel(listModel);`
- Para agregar o borrar elementos a la lista se usan métodos definidos en DefaultListModel
  - `String s = "Hola";`
  - `listModel.addElement(s);`
  - `listModel.remove(2);`
- Se se requiere funcionalidad adicional se puede definir una subclase de AbstractListModel

Aplicaciones GUI en Java

Tablas

- La clase JTable representa una tabla de valores
  - `JTable table = new JTable()`
- Para que la tabla tenga scrollbars se debe colocar dentro de un ScrollPane:
  - `JScrollPane sp = new JScrollPane(table);`
  - `sp.setPreferredSize(new Dimension(80, 260));`

- Los elementos de una tabla se guardan en un objeto de una clase que implementa la interfaz TableModel
- La interfaz TableModel define, entre otros, métodos para obtener los elementos de la tabla mediante dos índices (fila y columna) y para obtener el tamaño de la tabla

Aplicaciones GUI en Java

- La clase DefaultTableModel implementa TableModel
  - `Jtable table = new JTable();`
  - `TableModel tblModel = new DefaultTableModel();`
  - `table.setModel(tblModel);`
- Para agregar o borrar elementos a la tabla se usan métodos definidos en DefaultTableModel
  - `Vector fila = new Vector();`
  - `tblModel.addRow(fila);`
  - `tblModel.removeRow(1);`
- Se se requiere funcionalidad adicional se puede definir una subclase de AbstractTableModel