Interfaces Gráficas de Usuario

La programación orientada a objetos favorece la construcción y mantenimiento de sistemas de software complejos desarrollados para ambientes dinámicos. El encapsulamiento, la herencia y el polimorfismo permiten producir componentes reusables y fáciles de extender, como las que se requieren cuando se desarrolla una **interfaz gráfica de usuario**.

Una **interfaz** es un medio que permite que dos entidades se comuniquen. En una **interfaz de usuario** una de las entidades es una **persona** y la otra es un **sistema** que el usuario intenta controlar. El sistema puede ser una herramienta, un automóvil o cualquier dispositivo electrónico, en particular una computadora.

En un automóvil el conductor visualiza el estado a través del velocímetro o indicadores de combustible y controla el vehículo a través del embriage, el acelerador, la palanca de cambios, el volante, etc. En una computadora el usuario ejecuta acciones como oprimir teclas o el botón del ratón y estas acciones son percibidas por la interfaz del sistema.

Las **interfaces gráficas de usuario (GUI)** explotan la capacidad de las computadoras para reproducir imágenes y gráficos en pantalla y brindan un ambiente amigable, simple e intuitivo. Como su nombre lo indica, los tres elementos que definen a un GUI son:

- **Interfaz**: medio de comunicación entre entidades
- **Gráfica**: incluye ventanas, menúes, botones, texto, imágenes, etc.
- **Usuario**: persona que usa la interfaz para controlar un sistema

Las GUI reemplazaron a las interfaces de **líneas de comandos** en la cuales el usuario interactuaba con el sistema a través de una consola. En una GUI la interacción se realiza a través de:

- Iconos
- Ventanas
- Menúes desplegables
- Hipertexto
- Manipulación basada en *arrastrar y soltar*

Un **ícono** es un pictograma que representa a una entidad como un archivo, una carpeta, una acción o una aplicación.

Una **ventana** es una porción de la pantalla que sirve como una pantalla más pequeña. Un **menú** es una lista de opciones alternativas ofrecidas al usuario. Un menú desplegable es aquel que “cuelga” de una opción de otro menú.

Un **hipertexto** es un texto organizado mediante una red. Si el texto se organiza en red y además incluye imágenes, sonido, animación, se llama **hipermedio**.

La **manipulación basada en arrastrar y soltar** presupone el uso del ratón como dispositivo de entrada asociado a un cursor.

Todos estos elementos fueron determinantes para el desarrollo de GUI, aunque en el momento que fueron concebidos tuvieron poco éxito comercial porque la tecnología resultaba insuficiente para lograr implementaciones eficientes.

Una GUI se construye a partir de una **colección de componentes** con una **representación gráfica**. Por ejemplo, el botón para cerrar una ventana, la barra de desplazamiento de una ventana y la ventana misma.

Algunas componentes tienen la capacidad para **percibir eventos** generados por las acciones del usuario. Un usuario realiza una **acción** que genera un **evento** que es detectado por una **componente** lo cual provoca una **reacción**. La creación de componentes reactivas requiere que el lenguaje brinde algún mecanismo para el **manejo de eventos**. El **flujo de ejecución** va a quedar determinado por los eventos que generan los usuarios sobre las componentes de la interfaz.

La construcción de GUI en Java requiere integrar los conceptos centrales de la POO: **herencia, polimorfismo, ligadura dinámica y encapsulamiento**.

Objetos gráficos y Clases gráficas

Las componentes son las partes individuales a partir de las cuales se conforma una interfaz gráfica. Cada componente de una GUI está asociada a un **objeto gráfico**. Un objeto gráfico es una instancia de una **clase gráfica**. Algunos de los atributos de un objeto gráfico son **atributos gráficos** y parte del comportamiento ofrece **servicios gráficos**.

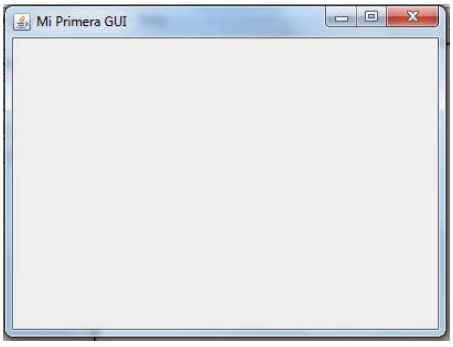
Algunos componentes son **contenedores** de otros componentes. Un contenedor tiene atributos especiales como por ejemplo el **diagramado** de acuerdo al cual se organizan las componentes contenidas.

Una **ventana** es un contenedor de alto nivel. Un **frame** es un tipo especial de ventana sobre el que puede ejecutarse una aplicación. Un frame tiene atributos gráficos como tamaño, barra de título, marco, panel de contenido y tres botones. Cada una de estas componentes tiene también sus atributos gráficos.

La ejecución del siguiente segmento de código:

```
import javax.swing.*;
class PrimerEjemplo {
 public static void main(String args[ ]) {
   JFrame f = new JFrame("Mi Primera GUI");
   f.setSize(400, 300) ;
   f.setVisible(true) ;    }}
```

Crea un frame como el que sigue:



La creación de un objeto de la clase `JFrame` requiere importar el paquete Swing. La instrucción:

```
   JFrame f = new JFrame("Mi Primera GUI");
```

Crea un objeto de la clase `JFrame` usando el constructor que recibe una cadena de caracteres con la que se inicializa el título de la barra de título. La clase `JFrame` brinda servicios para modificar los atributos gráficos. La variable `f` mantiene una referencia a un objeto de clase `JFrame` y puede recibir cualquiera de los mensajes provistos en esa clase o en sus clases ancestro. Las instrucciones:

```
   f.setSize(400, 300) ;
   f.setVisible(true) ;
```

Envían mensajes al frame para establecer su tamaño y hacerlo visible. Un efecto equivalente para el usuario, se consigue definiendo una clase que extiende a `JFrame` y creando un objeto de esta clase derivada.

```
import java.awt.*;
import javax.swing.*;
class MiVentana extends JFrame{
    public MiVentana() {
      super("Mi ventana");
      setSize(400, 300);
      getContentPane().setBackground(Color.BLUE);
      setDefaultCloseOperation(EXIT_ON_CLOSE);}}
```

La instrucción `super("Mi ventana")` invoca al constructor provisto por `JFrame` con la cadena que aparecerá en la barra de título como parámetro.

La instrucción `setSize(400, 300)` establece el ancho y la altura del frame en pixels.

El mensaje `getContentPane()` obtiene el panel de contenido del frame, el objeto que retorna es de clase `Container`. Este objeto recibe a continuación el mensaje `setBackground(Color.BLUE)` cuyo efecto es establecer el color del fondo.

El mensaje `setDefaultCloseOperation` se va a ligar a un método provisto por `JFrame`. El parámetro indica qué hacer cuando la ventana se cierra, en este caso terminar la aplicación.

El constructor envía tres mensajes al mismo objeto que se está creando, la instancia de `MiVentana`, que es también instancia de `JFrame`.

El método `main` crea ahora una instancia de `MiVentana`:

```
import javax.swing.*;
class SegundoEjemplo {
  public static void main(String args[ ])    {
    MiVentana f= new MiVentana();
    f.setVisible(true);    }
}
```

Como antes, la creación del objeto ejecuta el constructor y activa el frame. El objeto ligado a `f` puede recibir cualquiera de los mensajes provistos por la clase `JFrame`. La barra de título y los botones son componentes **reactivas**, reaccionan con un click del mouse.

Construcción de una GUI

La construcción de una GUI requiere:

- **Diseñar** la interfaz de acuerdo a las especificaciones
- **Implementar** la interfaz usando las facilidades provistas por el lenguaje

El **diseño** de una interfaz gráfica abarca tres aspectos:

- Definir las componentes
- Organizar las componentes estableciendo el diagramado de las componentes contenedoras
- Decidir cómo reacciona cada componente ante las acciones realizadas por el usuario

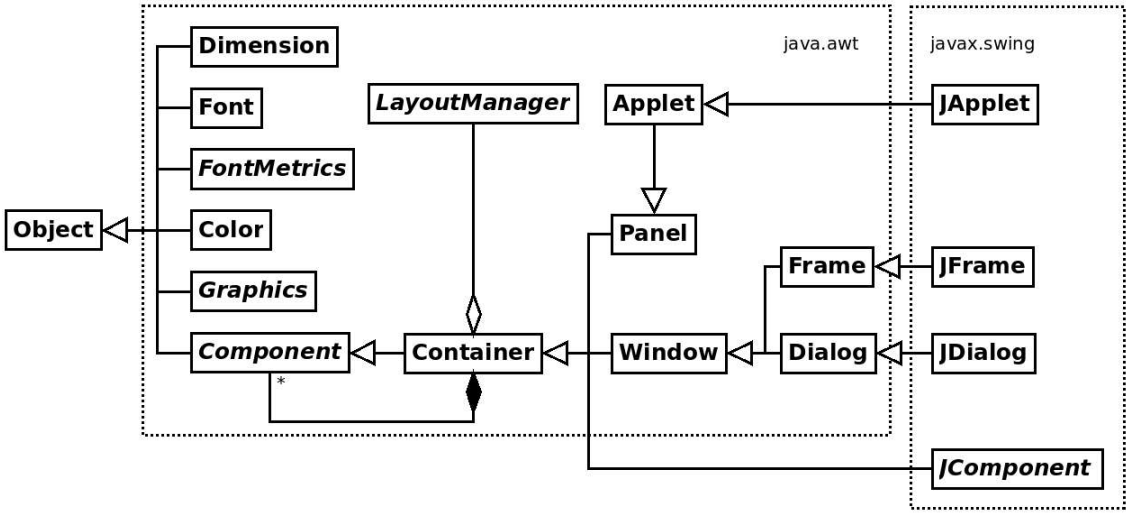
La **implementación** de una interfaz gráfica consiste en:

- **Crear** un objeto gráfico para cada componente de la GUI e **insertarlo** en otras componentes contenedoras
- **Definir el comportamiento** de las componentes reactivas en respuesta a las acciones del usuario

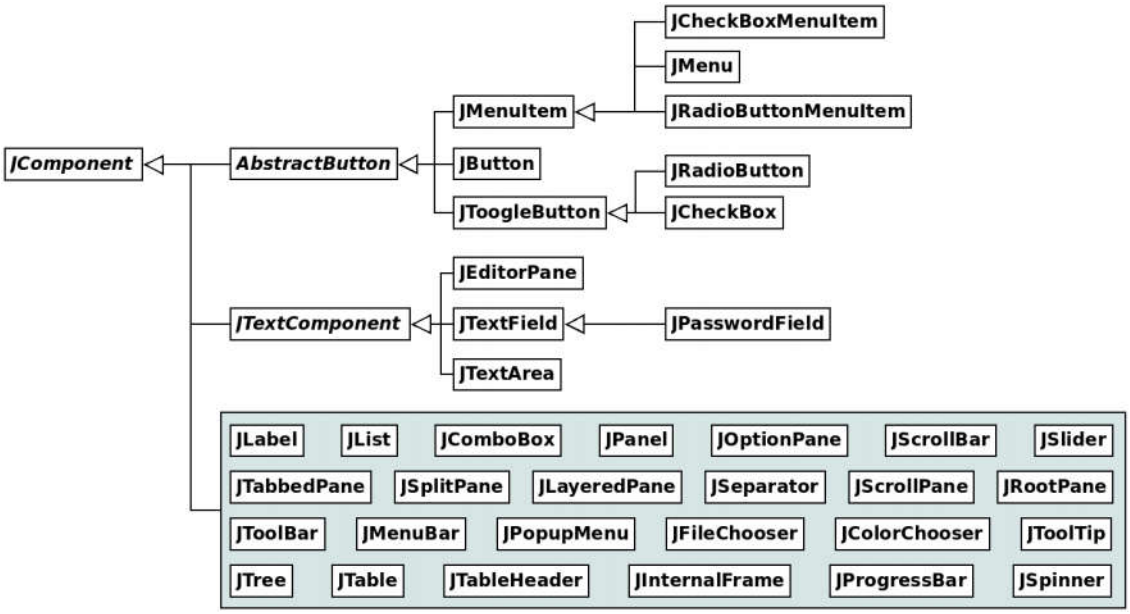
El diseño es una etapa fundamental, pero el objetivo en este libro no es diseñar interfaces gráficas, sino implementar el diseño dado para una GUI.

Tipos de Componentes

Las **componentes** de las GUI que se desarrollan en los ejemplos que siguen, son instancias de alguna de las clases provistas por los paquetes gráficos AWT (Abstract Window Toolkit) o Swing o de una clase derivada de ellas. AWT y Swing son paquetes gráficos independientes de la plataforma, que permiten desarrollar interfaces gráficas. Swing no reemplaza a AWT sino que lo usa y agrega nuevas clases, como muestra la figura.



Ambos paquetes brindan una colección de clases que pueden usarse para crear distintos **tipos de componentes**, como por ejemplo, botones, cajas de texto, menúes, etc. La jerarquía de clases que modela los diferentes tipos de componentes es:



Cada clase provista por AWT o Swing puede ser usada para:

- Crear objetos gráficos asociados a las componentes de la interfaz
- Definir clases más específicas a partir de las cuales se crearán componentes

Cada componente tiene una apariencia estándar pero también puede configurarse de acuerdo a las pautas de diseño que se adopten. El tamaño de cada componente se mide en pixels. Un **pixel** es la unidad de espacio más pequeña que puede mostrarse en pantalla. La **resolución** de una pantalla se mide de acuerdo a la cantidad de pixels que puede mostrar. Cuanto mayor sea la cantidad de pixels en la ventana, mayor será la resolución.

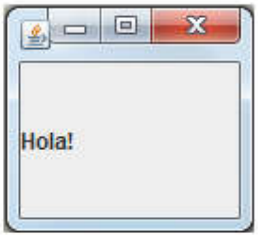
En los ejemplos que siguen se construyen GUI definiendo clases que extienden a `JFrame`, las cuales incluyen como atributos de instancia a objetos gráficos de alguna clase derivada de la clase `JComponent`.

Etiquetas

Una **etiqueta** es un objeto gráfico **pasivo** que permite mostrar un texto o una imagen. En Java se puede crear una etiqueta definiendo un objeto de clase `JLabel`. Los constructores provistos por la clase permiten establecer texto, imagen y/o alineación horizontal de la imagen:

```
JLabel()
JLabel (Icon image)
JLabel (Icon image, int horAlig)
JLabel (String text)
JLabel (String text, Icon icon, int horAlig)
```

Por ejemplo, la siguiente GUI:



Se activa al crear un objeto de clase `FrameConEtiqueta` definida a continuación:

```
import javax.swing.*;
class FrameConEtiqueta extends JFrame{
//Objeto gráfico
    JLabel etiqueta;
//Constructor
    public FrameConEtiqueta (String tit) {
      super(tit);
      setSize(100, 120);
      etiqueta= new JLabel("Hola!");
      getContentPane().add(etiqueta);
      setDefaultCloseOperation(EXIT_ON_CLOSE);}
}
```

La clase define un único atributo de instancia que es un objeto de la clase `JLabel`. Cada una de las instrucciones del constructor tiene el siguiente efecto:

- Invoca al constructor de `JFrame` con el texto de la barra de título.
- Establece el tamaño del frame.
- Crea una etiqueta estableciendo su texto.
- Recupera el panel del contenido del frame e inserta en su interior la etiqueta.
- Establece terminar la aplicación cuando el usuario cierre la ventana.

El método `main` en la clase `VentanaEtiqueta` crea un objeto de clase `FrameConEtiqueta`:

```
import javax.swing.*;
class VentanaEtiqueta {
 public static void main(String args[ ])    {
  FrameConEtiqueta f= new FrameConEtiqueta("Ventana con Etiqueta");
  f.setVisible(true);    }
}
```

El objeto ligado a la variable `f` tiene todos los atributos y servicios de `FrameConEtiqueta`, `JFrame` y de sus ancestros en la jerarquía de herencia. La segunda instrucción envía el mensaje `setVisible` al frame con el valor `true` como parámetro, su efecto es obviamente hacer visible el frame. Muchos otros objetos gráficos pueden recibir este mensaje, de modo que el parámetro permite ocultarlos o hacerlos visibles.

La etiqueta tiene asociado un texto y también es posible establecer un ícono:

```
import javax.swing.*;
class UnaImagen extends JFrame{
//Objeto gráfico
    JLabel etiquetaRobot;
public UnaImagen (String tit) {
   super(tit);
   setSize(400, 300);
   etiquetaRobot = new JLabel("Junior");
```

```
   etiquetaRobot.setIcon(new ImageIcon("junior.gif"));
   getContentPane().add(etiquetaRobot);
   setDefaultCloseOperation(EXIT_ON_CLOSE); }
}
```

En este caso se establece un ícono para la etiqueta usando la imagen almacenada en el archivo junior.gif. Como antes, para que la etiqueta sea visible es necesario insertarla en el panel de contenido del frame. El panel de contenido es un objeto contenedor, en el que se insertan componentes gráficas usando el mensaje `add`.

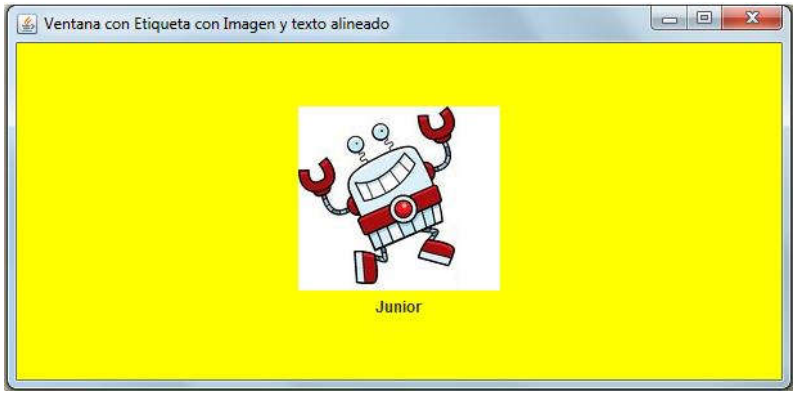
En el siguiente ejemplo se establece la alineación del texto y de la imagen:

```
import java.awt.*;
import javax.swing.*;
class EtiquetaCentro extends JFrame{
JLabel etiquetaRobot;
//Constructor
    public EtiquetaCentro (String tit) {
      super(tit);
      setSize(600, 300);
/*Crea la etiqueta y establece la imagen y la alineación del texto y
de la imagen*/
      etiquetaRobot = new JLabel("Junior");
      etiquetaRobot.setIcon(new ImageIcon("junior.gif"));
      etiquetaRobot.setHorizontalAlignment(JLabel.CENTER);
      etiquetaRobot.setHorizontalTextPosition(JLabel.CENTER);
      etiquetaRobot.setVerticalTextPosition(JLabel.BOTTOM);
/*Establece el color e inserta la etiqueta en el panel de contenido
*/
      getContentPane().setBackground(Color.YELLOW);
      getContentPane().add(etiquetaRobot);
      setDefaultCloseOperation(EXIT_ON_CLOSE);}
}
```

El mensaje:

```
getContentPane()
```

Obtiene el panel de contenido del frame. El panel de contenido es un atributo de instancia del frame y tiene a su vez atributos que pueden modificarse, como por ejemplo el color.



Cuando varias componentes se insertan en el panel de contenido, una queda superpuesta sobre la otra. Para distribuirlas de modo que todas queden visibles es necesario establecer el **diagramado** y también definir un tamaño adecuado para el frame.

El **organizador de diagramado** es un atributo de todos los objetos gráficos contenedores que determina como se distribuyen las componentes contenidas. Algunas de las clases provistas para crear organizadores son `BorderLayout`, `FlowLayout`, `GridLayout`.

`FlowLayout`: Distribuye los componentes uno al lado del otro comenzando en la parte superior. Por omisión provee una alineación centrada, pero también puede alinear a la izquierda o derecha.

`BorderLayout`: Divide el contenedor en cinco regiones: NORTH, SOUTH, EAST, WEST y CENTER, admite un único componente por región.

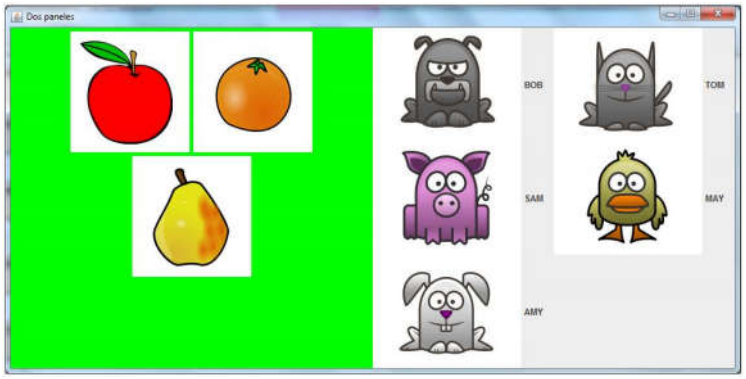
`GridLayout`: Divide el contenedor en una grilla de n filas por m columnas, con todas las celdas de igual tamaño.

Paneles

Aunque es posible insertar las componentes gráficas directamente sobre el panel de contenido del frame, resulta más sencillo organizarlas en paneles, esto es, instancias de la clase `JPanel()`.

Un **panel** es un contenedor de otras componentes gráficas. Los paneles se organizan en forma jerárquica. El principal atributo de un panel es el organizador de diagramado que establece cómo se distribuyen las componentes en su interior, tal como sucede con el panel de contenido.

Así, es posible establecer un organizador de diagramado para el panel de contenido y luego insertar en él dos o más paneles. Cada uno de estos paneles puede tener un diagramado diferente. En cada uno de ellos se insertan componentes que se distribuyen de acuerdo al diagramado establecido. El siguiente ejemplo inserta dos paneles sobre el panel de contenido, cada uno con un diagramado diferente y conteniendo un número distinto de etiquetas.



La GUI incluye dos arreglos de etiquetas. Los elementos de uno de los arreglos se insertan en uno de los paneles y quedan distribuidos de acuerdo a su organizador de diagramado. Los elementos del segundo arreglo se insertan en el otro panel, distribuidos de acuerdo a otro organizador.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class DosPaneles extends JFrame {
//Objetos gráficos
    JLabel [] mascota;
    JLabel [] fruta;
    JPanel panelIzquierdo,panelDerecho;
    Container contenedor;
    public DosPaneles (String tit){
      //Establece los atributos del frame
      super(tit);
      setSize(1000,500);
       setDefaultCloseOperation(EXIT_ON_CLOSE);
      //Establece los atributos del panel de contenido
      contenedor = getContentPane();
      contenedor.setLayout(new GridLayout(1,2));
      contenedor.setBackground(Color.GREEN);
      //Crea los objetos gráficos
      panelIzquierdo = new JPanel();
      panelDerecho = new JPanel();
      mascota = new JLabel[5];
      fruta = new JLabel[3];
//Establece los atributos de los paneles
      panelIzquierdo.setLayout(new FlowLayout());
      panelIzquierdo.setBackground(Color.GREEN);
      panelDerecho.setLayout(new GridLayout(3,2));
//Crea las etiquetas
      mascota[0] = new JLabel("BOB");
      mascota[0].setIcon(new ImageIcon("Perro.gif"));
      mascota[1] = new JLabel("TOM");
      mascota[1].setIcon(new ImageIcon("Gato.gif"));
      mascota[2] = new JLabel("SAM");
      mascota[2].setIcon(new ImageIcon("Cerdo.gif"));
      mascota[3] = new JLabel("MAY");
      mascota[3].setIcon(new ImageIcon("Pato.gif"));
      mascota[4] = new JLabel("AMY");
      mascota[4].setIcon(new ImageIcon("Conejo.gif"));
```

```
      fruta[0] = new JLabel("");
      fruta[0].setIcon(new ImageIcon("manzana.gif"));
      fruta[1] = new JLabel("");
      fruta[1].setIcon(new ImageIcon("naranja.gif"));
      fruta[2] = new JLabel("");
      fruta[2].setIcon(new ImageIcon("pera.gif"));
      /*Inserta las componentes en los paneles
      y los paneles en el panel de contenido*/
      for (int i=0;i<5;i++)
        panelDerecho.add(mascota[i]);
      for (int i=0;i<3;i++)
        panelIzquierdo.add(fruta[i]);
      contenedor.add(panelIzquierdo);
      contenedor.add(panelDerecho); }
}
```

Cada componente de la GUI, en este caso las etiquetas y los paneles, está asociada a un objeto gráfico. Los atributos de la clase DosPaneles mantienen justamente referencias a cada uno de estos objetos.

El comportamiento de estas componentes está establecido en las clases provistas por Swing y AWT. La ventaja de trabajar con distintos paneles va a ser más evidente cuanto mayor sea el número de componentes y cuanto más compleja sea su organización.

Botones

Un **botón** es una componente **reactiva**, esto es, puede percibir la acción del usuario y reaccionar de acuerdo al comportamiento establecido por el programador. Un botón se crea como una instancia de la clase `JButton`.

Como las etiquetas, sus atributos principales son texto e imagen. El texto y la imagen tienen atributos alineación vertical y horizontal. Cada botón tiene también un **color**, un **borde**, una **letra mnemónica** y una **forma**, y además puede estar habilitado o no.

Algunos de los constructores provistos para `JButton` son:

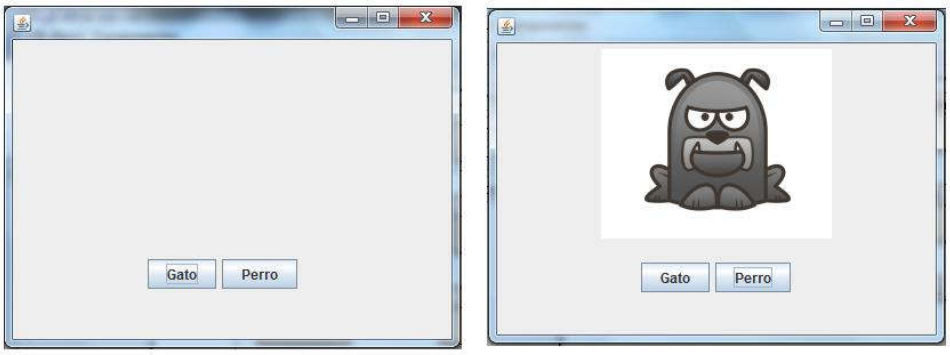
```
JButton ()
JButton (Icon image)
JButton (String text)
JButton (String text, Icon icon)
```

La clase `Dos_Botones` que se muestra a continuación permite crear un frame con dos paneles. En el primer panel se inserta una etiqueta y en el segundo se colocan dos botones. Cuando el usuario oprime un botón cambia la imagen de la etiqueta según la opción elegida.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class DosBotones extends JFrame {
//Componentes graficas
   private Container contenedor;
   private JButton botonGato, botonPerro;
   private JLabel imagen;
   private JPanel panelImagen,panelBotones;
   public DosBotones() {
//Obtiene el panel de contenido y crea los objetos gráficos
    contenedor = getContentPane();
    botonGato = new JButton("Gato");
```

```
    botonPerro = new JButton("Perro");
    imagen = new JLabel();
    panelImagen = new JPanel();
    panelBotones = new JPanel();
//Establece los atributos del frame
    setSize(400, 300);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    armaGUI();}
private void armaGUI(){
//Establece los atributos de los paneles
     contenedor.setLayout (new GridLayout(2,1));
     panelImagen.setLayout (new FlowLayout());
     panelImagen.setPreferredSize(new Dimension(380,200));
     panelBotones.setLayout (new FlowLayout());
     panelBotones.setPreferredSize(new Dimension(380,80));
//Crea un oyente para cada boton y lo registra
     OyenteP oyenteP = new OyenteP();
     OyenteG oyenteG = new OyenteG();
     botonPerro.addActionListener(oyenteP);
     botonGato.addActionListener(oyenteG);
//Inserta la etiqueta y los botones en los paneles
     panelImagen.add(imagen);
     panelBotones.add(botonGato);
     panelBotones.add(botonPerro);
//Inserta los paneles en el panel de contenido
     contenedor.add(panelImagen);
     contenedor.add(panelBotones);
}
private class OyenteP implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        imagen.setIcon(new ImageIcon("Perro.gif"));      }}
private class OyenteG implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        imagen.setIcon(new ImageIcon("Gato.gif"));      }}
}
```

La siguiente figura muestra el frame antes y después de que el usuario oprima el botón con rótulo Perro:



La definición de métodos internos y los comentarios muestran la estructura del código y favorecen la legibilidad.

```
//Obtiene el panel de contenido y crea los objetos gráficos
//Establece los atributos del frame
//Establece los atributos de los paneles
```

```
//Crea un oyente para cada boton y lo registra
//Inserta la etiqueta y los botones en los paneles
//Inserta los paneles en el panel de contenido
```

El panel de contenido del frame se organiza como una grilla de dos filas y una única columna. Los objetos ligados a `panelImagen` y `panelBotones` son instancias de la clase `JPanel` y en ambos el diagramado es un objeto de clase `FlowLayout`. Las componentes se disponen en una fila de un tamaño preestablecido. Si el tamaño es mayor que el requerido, por omisión las componentes quedan centradas.

Una **fuente de evento** es un objeto que está asociado a una componente gráfica y puede percibir una acción del usuario.

En el ejemplo se definen dos objetos fuentes de evento específicos de la aplicación y se ligan a las variables `botonGato` y `botonPerro`.

Un **oyente** es un objeto que espera que ocurra un evento y **reacciona** cuando esto sucede.

Cada botón definido en el ejemplo tiene que reaccionar ante la acción del usuario sobre él; se crean dos **objetos oyente** y se registran a **objetos fuente de evento**. El oyente de `botonPerro` se llama `oyenteP` y es de clase `OyenteP`. En forma análoga, el objeto gráfico `botonGato` se asocia a un objeto oyente `oyenteG` que va detectar las acciones del usuario sobre el botón.

Las clases `OyenteP` y `OyenteG` se definen como internas a la clase `DosPaneles`, de modo que tienen acceso a todos sus atributos de instancia. Cada una implementa a la interface `ActionListerner` provista por Java. La implementación consiste en redefinir el método `actionPerformed` de acuerdo al comportamiento que se requiera en cada botón. En este caso, la imagen de la etiqueta se establece usando la figura almacenada en un archivo.

Java brinda otras clases para crear botones, como por ejemplo `JRadioButton` que permite agrupar botones de modo que solo uno puede estar seleccionado. Los atributos de instancia de la clase son: la cadena, el ícono y el estado.

Los constructores de un objeto de clase `JRadioButton` son:

```
JRadioButton(String s)
JRadioButton(String s, boolean b)
JRadioButton(Icon i)
JRadioButton(Icon i, boolean b)
JRadioButton(String s, Icon i)
JRadioButton(String s, Icon i, boolean b)
JRadioButton()
```

El parámetro de tipo `boolean` establece si el botón está seleccionado.

La clase `RadioButtonDemo` permite crear una GUI como la que muestra la siguiente figura:

La implementación es ahora:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class RadioButtonDemo extends JFrame {
    //Componentes gráficas
    private Container contenedor;
    private JPanel radioPanel,panelImagen ;
    private JLabel  imagen;
    private ButtonGroup group;
    private JRadioButton pato;
    private JRadioButton gato ;
    private JRadioButton perro ;
    private JRadioButton conejo ;
    private JRadioButton cerdo ;
//Constructor
    public RadioButtonDemo() {
        contenedor = getContentPane();
        radioPanel = new JPanel();
        panelImagen = new JPanel();
//Crea el grupo y cada botón
        group = new ButtonGroup();
        pato = new JRadioButton("pato");
        gato = new JRadioButton("gato");
        perro = new JRadioButton("perro");
        conejo = new JRadioButton("conejo");
        cerdo = new JRadioButton("cerdo");
        setSize(400, 320);
       setDefaultCloseOperation(EXIT_ON_CLOSE);
        armaRadioButton();
}
private void armaRadioButton(){
//Establece la imagen inicial para la etiqueta y el tamaño
        imagen = new JLabel(new ImageIcon("pato.gif"));
        imagen.setPreferredSize(new Dimension(180, 180));
//Establece el diagramado de los paneles
        contenedor.setLayout (new BorderLayout());
        radioPanel.setLayout(new GridLayout(0, 1));
//Crea y registra un mismo oyente a todos los botones
        Oyente oyente = new Oyente();
        pato.addActionListener(oyente);
        gato.addActionListener(oyente);
```

```
        perro.addActionListener(oyente);
        conejo.addActionListener(oyente);
        cerdo.addActionListener(oyente);
// Agrupa los botones y activa el boton del pato
        pato.setSelected(true);
        group.add(pato);
        group.add(gato);
        group.add(perro);
        group.add(conejo);
        group.add(cerdo);
//Inserta los botones y la imagen en el panel
        radioPanel.add(pato);
        radioPanel.add(gato);
        radioPanel.add(perro);
        radioPanel.add(conejo);
        radioPanel.add(cerdo);
        panelImagen.add( imagen);
//Inserta los paneles en el contenedor
        contenedor.add(radioPanel, BorderLayout.WEST);
        contenedor.add(panelImagen, BorderLayout.CENTER);    }
private class Oyente implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        String  m = (String)e.getActionCommand();
         imagen.setIcon(new ImageIcon(m + ".gif"));
    }
}
}
```

Luego de crear los objetos de clase `JRadioButton` es necesario insertarlos en un objeto de clase `ButtonGroup`. Solo uno de los objetos del grupo puede estar seleccionado.

En este caso se define una sola clase para implementar la interface `ActionListener`. El método `actionPerformed` recibe como parámetro un objeto de clase `ActionEvent`. El objeto mantiene información referida a la acción del usuario sobre la GUI. Cuando se envía el mensaje `getActionCommand` al objeto `e`, el resultado es el comando que corresponde a la acción del usuario. El resultado es en este caso una cadena de caracteres, que se usa para establecer la imagen de la etiqueta.

Cajas de opciones

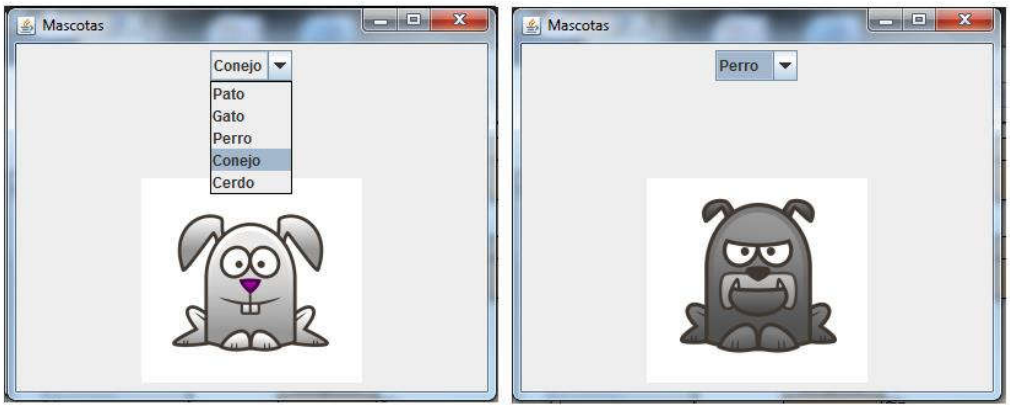
Una **caja de opciones** puede crearse como una instancia de la clase `JComboBox`, editable o no editable. Una caja de opciones no editable consta de una lista de valores drop-down y un botón. Una caja de opciones editable tiene además un campo de texto con un pequeño botón. El usuario puede elegir una opción de la lista o tipear un valor en el campo de texto. El constructor recibe como parámetro un arreglo de objetos de clase `String`.

En el siguiente ejemplo el usuario elige la mascota dentro de la una lista de opciones:

```
import java.awt.*;
import java.awt.Container;
import java.awt.event.*;
import javax.swing.*;
class ComboBoxDemo extends JFrame {
//Opciones de la caja
private String[] mascotas = { "Pato", "Gato", "Perro",   "Conejo",
"Cerdo" };
```

```
//Objetos gráficos
    private Container contenedor;
    private JPanel panelComboBox,panelImagen ;
    private JLabel imagen;
    private JComboBox listaMascotas;
public ComboBoxDemo (String tit) {
//Establece los valores de los atributos del frame
    super(tit);
    setSize(400, 320);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    armaComboBox();}
private void armaComboBox(){
//Obtiene panel de contenido y crea los paneles y la etiqueta
    contenedor = getContentPane();
    panelComboBox = new JPanel();
    panelImagen = new JPanel();
    imagen = new JLabel(new ImageIcon( "conejo.gif"));
/*Crea la caja de opciones y selecciona la opción que corresponde al
ícono establecido en la etiqueta*/
    listaMascotas = new JComboBox(mascotas);
    listaMascotas.setSelectedIndex(3);
//Establece el diagramado y la apariencia de la etiqueta
   contenedor.setLayout (new BorderLayout());
//Crea y registra el oyente
   Oyente oyente = new Oyente ();
   listaMascotas.addActionListener(oyente);
// Inserta los paneles en el panel de contenido
   panelComboBox.add (listaMascotas);
   panelImagen.add(imagen);
   contenedor.add(panelComboBox, BorderLayout.NORTH);
   contenedor.add(panelImagen, BorderLayout.SOUTH);
}
private class Oyente implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        String m = (String)listaMascotas.getSelectedItem();
        imagen.setIcon(new ImageIcon(m + ".gif"));}}
}
```

La siguiente figura muestra el frame en el momento en el que el usuario oprime la flecha de la caja, inmediatamente después de crearse el frame y luego de que elige la opción perro:



El objeto `listaMascotas` se registra a un objeto oyente cuya clase implementa el método `ActionPerformed` de la interface `ActionListener`. En este caso, el mensaje `getSelectedItem` se envía al objeto asociado a `listaMascotas` y el valor que retorna se convierte en un objeto de clase `String`. Esta cadena se utiliza para crear una imagen y establecerla en la etiqueta.

Campos de Texto

Un **campo de texto** es una caja que permite ingresar una línea de texto por teclado. Un objeto de la clase `JTextField` permite mantener un campo de texto. Los atributos son una cadena de caracteres, la cantidad de caracteres que se visualizan en la caja y el modelo a utilizar. Cada vez que el usuario tipea una tecla se crean objetos de clase `KeyEvent` o de la clase `ActionEvent`.

Los constructores provistos por la clase son:

```
JTextField()
JTextField(Document doc, String text, int col)
JTextField(int col)
JTextField(String text)
JTextField(String text, int col)
```

En el siguiente ejemplo el oyente detecta la acción del usuario cuando oprime la tecla Intro después de tipear una cadena.

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
class CajaTexto  extends JFrame {
//Componentes Gráficas
      private Container contenedor;
      private JPanel panelImagen,panelCaja;
      private JLabel imagen;
      private JTextField cTexto;
      public CajaTexto () {
 //Establece los atributos del frame
       super("Mi Mascota");
       setSize(400, 320);
       setDefaultCloseOperation(EXIT_ON_CLOSE);
       setLocationRelativeTo(null);
       armaCaja();}
private void armaCaja(){
//Obtiene el panel de contenido
       contenedor =  getContentPane();
//Crea las componentes gráficas
       panelImagen = new JPanel();
       panelCaja = new JPanel();
       imagen = new JLabel();
       cTexto = new JTextField (8);
//Establece los atributos de los objetos gráficos
       contenedor.setLayout(new BorderLayout());
       TitledBorder borde = new TitledBorder (new   LineBorder
(Color.black,5,true),"");
       imagen.setHorizontalAlignment(JLabel.CENTER);
```

```
        imagen.setPreferredSize(new Dimension(180, 180));
        imagen.setBorder(borde);
//Crea y registra el oyente de la caja
        OyenteCaja oyente = new OyenteCaja();
        cTexto.addActionListener(oyente);
//Inserta las componentes en los paneles
        panelImagen.add(imagen);
        panelCaja.add(cTexto);
        contenedor.add(panelImagen,BorderLayout.NORTH);
        contenedor.add(panelCaja,BorderLayout.SOUTH);
}
private class OyenteCaja implements ActionListener {
     public void  actionPerformed(ActionEvent event){
              String m = cTexto.getText();
              imagen.setIcon(new ImageIcon(m + ".gif")); }   }
}
```

En el momento que se crea la caja de texto se establece el tamaño de la componente, sin embargo, el usuario puede tipear más caracteres que los que se visualizan en la caja. Si la clase del oyente implementa `ActionListener` el método `actionPerformed` recibe como siempre un objeto de clase `ActionEvent`. En este caso, se obtiene la cadena de caracteres que corresponde a parte del nombre del archivo, directamente de la caja de texto.

En este ejemplo se ha establecido la preferencia de tamaño y un borde para la etiqueta.

Paneles de diálogo

Un **panel de diálogo** se usa para leer un valor simple y/o mostrar un mensaje. Los atributos incluyen uno o más botones. El mensaje puede ser un error o una advertencia y puede estar acompañado de una imagen o algún otro elemento. Para definir un diálogo estandar se usa la clase `JOptionPane` que brinda los siguientes servicios:

showMessageDialog: Muestra un diálogo modal con un botón, etiquetado "OK". Se puede especificar el icono y el texto del mensaje y del título. Por omisión el ícono es de información

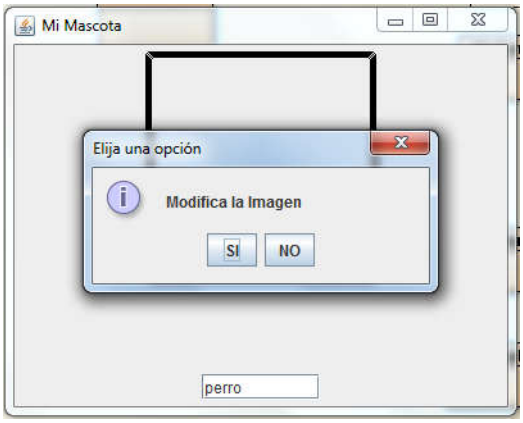
showConfirmDialog: Muestra un diálogo modal con dos botones, etiquetados "Yes" y "No". Por omisión aparece el ícono question.

showOptionDialog: Requiere especificar el texto de los botones.

showInputDialog: Muestra un diálogo modal que obtiene una cadena del usuario. La cadena puede ser ingresada por el usuario en un campo de texto o elegida de un ComboBox no editable.

En cada caso se puede utilizar un icono personalizado, no utilizar ninguno, o utilizar uno de los cuatro iconos estandar de `JOptionPane` (question, information, warning, y error). Por omisión aparece el ícono *question*.

En la clase `CajaTexto` propuesta antes se puede agregar un panel de diálogo con opciones "si" y "no" para consultar al usuario si confirma la modificación de la imagen de la etiqueta.



El código de `actionPerformed` se modifica como sigue:

```
public void  actionPerformed(ActionEvent event){
  String m = cTexto.getText();
  String [] op = {"SI","NO"};
  JOptionPane pd = new JOptionPane();
  int opSel = pd.showOptionDialog(null,
              "Modifica la Imagen",
              "Elija una opción",
               JOptionPane.YES_NO_OPTION,
               JOptionPane.INFORMATION_MESSAGE,
               null, //Icon icon,
               op,
               "Si");//Valor inicial
  if(opSel == 0 )
    imagen.setIcon(new ImageIcon(m + ".gif"));
  cTexto.setText("");
}
```

Las clases `Printing`, `JColorChooser` y `JFileChooser` permiten ofrecer diálogos más específicos.

Todo diálogo es dependiente de un frame. Cuando un frame se destruye, también se destruyen los diálogos que dependen de él. Un diálogo es modal cuando bloquea la entrada de datos del usuario a través de todas las demás ventanas.

Los cuadros de diálogo creados con `JOptionPane` son modales. Para crear un diálogo no modal es posible usar la clase `JDialog`.

Investigue de manera autónoma cuál es la utilidad y cómo se usan las clases `JCheckBox` y `JTextArea`.

Lectura y escritura

Toda la lectura y escritura realizada por componentes de la clase Swing se realiza usando objetos de clase `String`. La entrada y salida de números requiere entonces convertir valores.

La conversión de un valor de tipo `int` a `String` puede implementarse concatenando dos cadenas:

```
String s = ""+42;
```

La conversión de una cadena a un entero requiere usar el método `parseInt` de la clase estática `Integer`:

```
String s = "42";
int num = Integer.parseInt(s);
```

En forma análoga es posible convertir una cadena de caracteres al tipo double:

```
double val = Double.parseDouble("42.5");
```

El siguiente ejemplo propone una GUI para convertir un valor representando una velocidad expresada en millas por horas a kilómetros por horas.

La velocidad en millas se lee en una caja de texto y la velocidad en kilómetros se establece en una etiqueta.

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class CajaTextoMllKm extends JFrame {
//Atributos de instancia
  private JPanel panel;
  private JLabel cartelEntrada,cartelSalida,solucion;
  private JTextField caja;
public CajaTextoMllKm (String tit) {
//Establece los atributos del frame
  super(tit);
  setSize(400, 120);
  setDefaultCloseOperation(EXIT_ON_CLOSE);
  crearComponentes();
  establecerAtributos();
//Crea y registra el oyente de la caja
  OyenteCaja oyente = new OyenteCaja();
  caja.addActionListener(oyente);
  armarPaneles();}
private void crearComponentes(){
//Crea las components gráficas
  panel = new JPanel();
  cartelEntrada = new JLabel();
  cartelSalida = new JLabel();
  velKm = new JLabel("0");
  caja = new JTextField ("",5);}
private void establecerAtributos(){
//Establece los atributos de las componentes gráficas
  panel.setLayout(new GridLayout(4, 1));
  getContentPane().setLayout(new BorderLayout());
  cartelEntrada.setText("Ingrese la velocidad en millas");
  cartelSalida.setText("La velocidad en kilometros es");}
```

```
private void armarPaneles(){
//Inserta las componentes en los paneles
  panel.add(cartelEntrada);
  panel.add(caja);
  panel.add(cartelSalida);
  panel.add(velKm);
  getContentPane().add(panel,BorderLayout.CENTER);}
private class OyenteCaja implements ActionListener {
  public void  actionPerformed(ActionEvent event){
    String v = caja.getText();
    if (!v.equals("")) {
      double mph = Double.parseDouble (v);
      velKm.setText(""+mph*1.621);}}  }
}
```

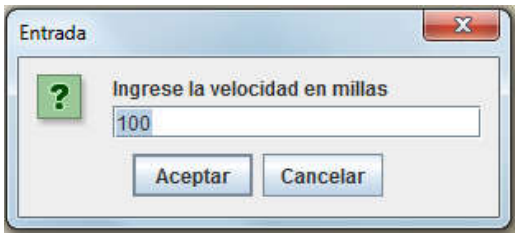
El condicional en el método `actionPerformed` permite evitar una terminación anormal si el usuario no ingresa un valor en la caja de texto.

La clase `DialogoMllKm` propuesta a continuación implementa una GUI similar a la anterior, usando un panel de diálogo:

```
import javax.swing.border.*;
import javax.swing.*;
class DialogoMllKm extends JFrame {
  private JOptionPane dialogo;
  public DialogoMllKm() {
        double mph,kph;
        String cad ;
        dialogo = new JOptionPane();
        cad = dialogo.showInputDialog(null,
              "Ingrese la velocidad en millas",100 );
        if (!cad.equals("")){
          mph = Double.parseDouble(cad);
          kph = 1.621 * mph;
         dialogo.showMessageDialog (null,
         "La velocidad en kilometros es "+kph);}
    System.exit(0);}
}
```

En este caso el constructor crea el panel de diálogo y le envía el mensaje `dialogo.showInputDialog` que permite ingresar un valor, inicializado en 100. El valor ingresado es una cadena de caracteres, que puede estar vacía. Si no está vacía, se convierte en un valor de tipo `double` y luego se computa el cambio de unidad.

La creación de un objeto de clase `DialogoMllKm` genera el siguiente panel:



Si el usuario oprime Aceptar aparece:

La solución propuesta asume que cuando el usuario oprime el botón Aceptar la caja de texto del cuadro de diálogo no está vacía.

La estructura de una GUI

La estructura de una GUI debería favorecer la legibilidad de modo que el código sea fácil de verificar y mantener. El código de cada una de las clases que implementan una GUI en este libro incluye:

- Instrucciones para importar paquetes gráficos
- Declaraciones de atributos de instancia
- Definición de un constructor y algunos métodos internos invocados por el constructor
- Definición de clases internas que implementan interfaces y permiten crear oyentes

Java permite definir clases oyentes anónimas e implementar el método `actionPerformed` directamente en el parámetro real del mensaje `addActionListener`. Esta alternativa compromete la legibilidad del código.

También es posible implementar las clases oyentes a través de clases externas. Esta es una alternativa adecuada para definir interfaces gráficas complejas.

En este libro se adopta la convención de modular la solución definiendo clases oyentes que modelan el comportamiento reactivo de una o más componentes gráficas. Para los casos de estudio planteados, las clases internas son un recurso adecuado.

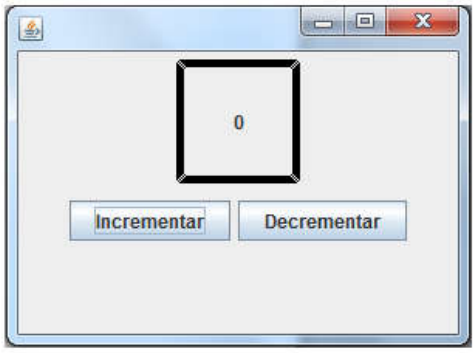
El constructor o los métodos internos invocados por el constructor incluyen instrucciones para:

- Establecer los valores de los atributos gráficos del frame
- Crear objetos gráficos y establecer los valores de sus atributos, algunos de estos objetos serán paneles contenedores
- Crear objetos oyentes y registrarlos a los objetos asociados a componentes reactivas
- Insertar los objetos gráficos en los paneles y los paneles en el panel de contenido

Como ilustra el ejemplo propuesto en esta sección, la interfaz puede incluir también atributos propios de la aplicación, que no corresponden a compontentes gráficas. Como el resto de los atributos de la clase, se declaran privados y solo pueden ser accedidos por los servicios propios y las clases internas.

Es muy importante estructurar el código de la GUI para favorecer la legibilidad. En una interfaz muy simple los comentarios pueden ser suficientes. Cuando el constructor requiere varias líneas de código es aconsejable utilizar métodos auxiliares como se propuso en los ejemplos anteriores. En cualquier caso existen varias maneras de modular.

Dada una GUI como la que muestra la siguiente figura, con dos botones que permiten incrementar y decrementar el número de una etiqueta:



La interfaz requiere crear:

- Una clase que extiende a `JFrame`, llamada `GUI_Contador`.
- Una clase llamada `Contador` con un método `main` que inicia la ejecución al crear una instancia de `GUI_Contador`

La implementación de `Contador` puede ser:

```
class Contador{
public static void main(String[] args) {
  GUI_Contador cuadro = new GUI_Contador();
  cuadro.setVisible(true);}
}
```

La clase `GUI_Contador` crea cinco componentes gráficas

- Dos botones
- Dos paneles
- Una etiqueta

Los atributos de instancia de la clase mantienen justamente referencias a estos objetos. Los dos botones son components reactivas, cada uno queda asociado a un objeto **oyente** que establece su comportamiento.

La estructura del constructor de la clase `GUI_Contador` puede ser:

- Inicializar el contador
- Establecer los valores de los atributos gráficos heredados de la clase `JFrame`.
- Crear los paneles
- Crear la etiqueta para el número, establecer sus atributos e insertar en el panel del contador.
- Crear el botón incrementar, su oyente, registrarlo e insertar en el panel de control.
- Crear el botón decrementar, su oyente, registrarlo e insertar en el panel de control.
- Insertar los paneles en el panel de contenido.

Una implementación de `GUI_Contador` es:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import javax.swing.*;
class GUI_Contador extends JFrame {
//Atributos de instancia incluyendo objetos gráficos
  private int numero;
```

```
  private JPanel pContador,pControl;
  private JLabel numeroEtiqueta;
  private JButton botonIncrementar,botonDecrementar;
  public GUI_Contador() {
//Inicializa el contador
  numero = 0;
//Establece los valores de los atributos del frame
  setLayout(new FlowLayout());
  setSize(300, 320);
  setDefaultCloseOperation(EXIT_ON_CLOSE);
  getContentPane().setLayout(new GridLayout(2,1));
//Crea los paneles
  pContador = new JPanel();
  pControl = new JPanel();
/*Crea la etiqueta,  establece los valores de los atributos y la
inserta en el panel del contador*/
  numeroEtiqueta = new JLabel("" + numero);
  TitledBorder borde = new TitledBorder (new   LineBorder
(Color.black,5,true),"");
  numeroEtiqueta.setHorizontalAlignment(JLabel.CENTER);
  numeroEtiqueta.setPreferredSize(new Dimension(80, 80));
  numeroEtiqueta.setBorder(borde);
  pContador.add(numeroEtiqueta);
/*Crea el boton incrementar, el oyente, lo registra e inserta el
botón en el panel de control*/
  botonIncrementar = new JButton("Incrementar");
  OyenteBotonI incrementar = new OyenteBotonI();
  botonIncrementar.addActionListener(incrementar);
  pControl.add(botonIncrementar);
/*Crea el boton decrementar, el oyente, lo registra e inserta el
botón en el panel de control*/
  botonDecrementar = new JButton("Decrementar");
  OyenteBotonD decrementar = new OyenteBotonD();
  botonDecrementar.addActionListener(decrementar);
  pControl.add(botonDecrementar);
/*Inserta el panel del contador y el panel de control en el panel de
contenido*/
  getContentPane().add(pContador);
  getContentPane().add(pControl);}
 private class OyenteBotonI implements ActionListener {
     public void actionPerformed(ActionEvent event) {
      numero++;
      numeroEtiqueta.setText("" + numero);}
    }
 private class OyenteBotonD implements ActionListener {
     public void actionPerformed(ActionEvent event) {
      numero--;
      numeroEtiqueta.setText("" + numero);}
    }
}
```

Por supuesto el código puede estructurarse siguiendo un criterio diferente.

- Establecer los valores de los atributos gráficos heredados de la clase `JFrame`.
- Crear los paneles.
- Crear la etiqueta.

- Crear el botón incrementar y el botón decrementar.
- Crear el oyente del botón incrementar y el oyente del botón decrementar.
- Registrar cada uno de los dos oyentes al botón.
- Insertar la etiqueta y los botones en el panel que corresponda.
- Insertar los paneles en el panel de contenido.

En la siguiente implementación se definen métodos auxiliares, internos a la clase `GUI_Contador`, para modular la solución.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import javax.swing.*;
class GUI_Contador extends JFrame {
//Atributos de instancia incluyendo objetos gráficos
  private int numero=0;
  private JPanel pContador,pControl;
  private JLabel numeroEtiqueta;
  private JButton botonIncrementar,botonDecrementar;
  public GUI_Contador() {
//Establece los valores de los atributos del frame
  setLayout(new FlowLayout());
  setSize(300, 220);
  setDefaultCloseOperation(EXIT_ON_CLOSE);
  getContentPane().setLayout(new GridLayout(2,1));
//Crea los paneles
  pContador = new JPanel();
  pControl = new JPanel();

  armarEtiqueta();
  armarBotonesyOyentes();
  armarPaneles();}
private void armarEtiqueta(){
/*Crea la etiqueta y establece los valores de los atributos*/
  numeroEtiqueta = new JLabel("" + numero);
  TitledBorder borde = new TitledBorder (new   LineBorder
(Color.black,5,true),"");
  numeroEtiqueta.setHorizontalAlignment(JLabel.CENTER);
  numeroEtiqueta.setPreferredSize(new Dimension(80, 80));
  numeroEtiqueta.setBorder(borde);}
private void armarBotonesyOyentes(){
/*Crea los dos botones, los dos oyentes y los registra*/
  botonIncrementar = new JButton("Incrementar");
  botonDecrementar = new JButton("Decrementar");
  OyenteBotonI incrementar = new OyenteBotonI();
  OyenteBotonD decrementar = new OyenteBotonD();
  botonIncrementar.addActionListener(incrementar);
  botonDecrementar.addActionListener(decrementar);}
private void armarPaneles(){
/*Inserta la etiqueta y los botones en los paneles correspondientes
y los dos paneles en el panel de contenido*/
  pContador.add(numeroEtiqueta);
  pControl.add(botonIncrementar);
  pControl.add(botonDecrementar);
  getContentPane().add(pContador);
  getContentPane().add(pControl); }
```

```
private class OyenteBotonI implements ActionListener {
     public void actionPerformed(ActionEvent event) {
      numero++;
      numeroEtiqueta.setText("" + numero);}
    }
 private class OyenteBotonD implements ActionListener {
     public void actionPerformed(ActionEvent event) {
      numero--;
      numeroEtiqueta.setText("" + numero);}
    }
}
```

En este caso el atributo de instancia `numero` se inicializa en la declaración. Los comentarios no describen la semántica de cada instrucción, sino que resaltan la estructura elegida para organizar el código.

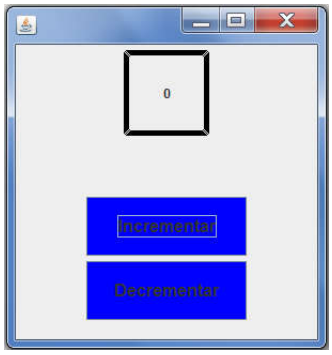
La herencia también permite modular la solución favoreciendo la legibilidad. Los botones se pueden crear con las instrucciones:

```
  botonIncrementar = new BotonContador("Incrementar");
  botonDecrementar = new BotonContador("Decrementar");
```

La clase `BotonContador` extiende a `JButton` estableciendo la misma apariencia para los dos botones de la GUI, excepto el rótulo del botón que varía en cada uno, según se especifica a través de un parámetro:

```
import java.awt.Dimension;
import java.awt.Font;
import javax.swing.*;
class BotonContador extends JButton {
    public BotonContador(String rotulo){
        setPreferredSize(new Dimension(306, 55));
        setFont(new Font("Arial",1,22));
        setText(rotulo);}
}
```

La interfaz es ahora:



Así, el programador puede especializar cualquier clase provista por `JComponent` para obtener soluciones moduladas, estableciendo una única vez los valores de los atributos de todas las componentes gráficas que tengan una misma apariencia.

Programación basada en Eventos

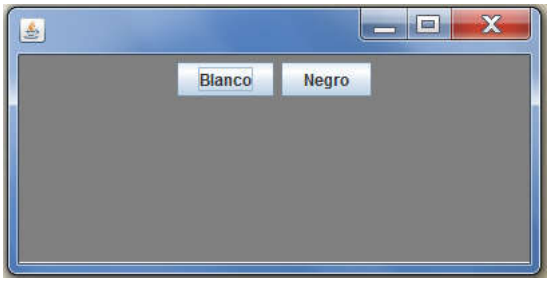
La construcción de una GUI utiliza un modelo de **programación basado en eventos**. En este modelo el orden en el cual se ejecutan las instrucciones de un programa queda determinado por **eventos.** Un evento es una **señal** de que algo ha ocurrido. Así, el flujo de ejecución no lo establece el programador, sino las acciones del usario.

Existen diferentes tipos de eventos, este libro se concentra exclusicamente en aquellos que son generados por acciones del usuario al interactuar con la GUI. Algunas componentes de una GUI son **reactivas**, pueden **percibir** las acciones del usuario y **reaccionar** en respuesta a ellas. Una componente reactiva están asociada a un **objeto fuente del evento** creado por el programador.

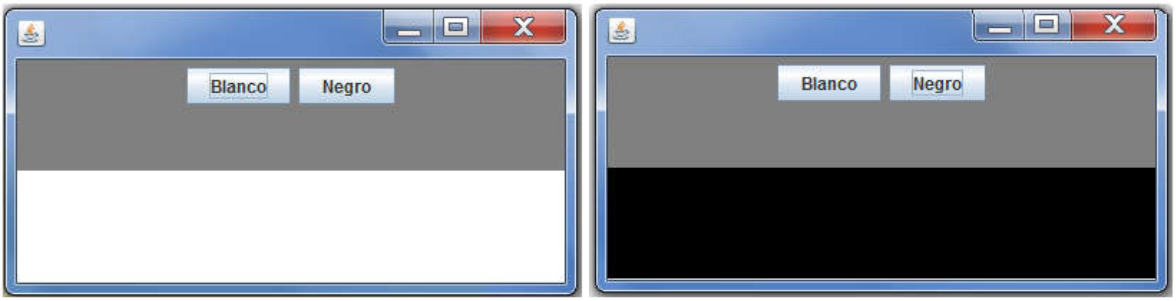
La reacción del sistema en respuesta a la acción del usuario va a quedar determinada por la clase a la que pertenece un **objeto oyente.** El objeto oyente está ligado al objeto fuente de evento a través de una instrucción de **registración**.

Caso de Estudio: Color panel

Implementar una GUI que permita seleccionar el color de un panel. La ventana inicialmente debe aparecer así:



Una vez que el usuario aprieta un botón, el panel se pinta de blanco o de negro como muestra la figura:



El código de la GUI puede ser:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class  GUIFondoWB extends JFrame{
//Objetos gráficos
   private JPanel panelColor, panelBotones;
   private JButton botonBlanco, botonNegro;
public GUIFondoWB() {
   setSize(400, 200);
   setDefaultCloseOperation(EXIT_ON_CLOSE);
   armaBotones();
   armaPaneles(); }
```

```
private void armaBotones(){
//Crea un botón Blanco, su oyente y lo registra
    botonBlanco = new JButton("Blanco");
    OyenteBotonB ponerBlanco= new OyenteBotonB();
    botonBlanco.addActionListener(ponerBlanco);
//Crea el botón Negro, su oyente  y lo registra
    botonNegro = new JButton("Negro");
    OyenteBotonN ponerNegro= new OyenteBotonN();
    botonNegro.addActionListener(ponerNegro);}
private void armaPaneles(){
/*Se establece el diagramado de los Paneles*/
    Container contenedor = getContentPane();
    contenedor.setLayout(new GridLayout(2,1));
    panelBotones = new JPanel();
    panelColor = new JPanel();
    panelColor.setBackground(Color.GRAY);
    panelBotones.setBackground(Color.GRAY);
//Se agregan los botones al panel de botones
    panelBotones.add(botonBlanco);
    panelBotones.add(botonNegro);
/*Se agregan los paneles al panel de contenido*/
    contenedor.add(panelBotones);
    contenedor.add(panelColor);}
private class OyenteBotonB implements ActionListener {
  public void actionPerformed( ActionEvent event) {
     panelColor.setBackground(Color.white); }
}
private class OyenteBotonN implements ActionListener {
  public void actionPerformed( ActionEvent event) {
      panelColor.setBackground(Color.black); }
}
}
```

Para que se produzca una reacción el programador *registra* **objeto oyente** `ponerBlanco` al **objeto fuente de evento** `botonBlanco`. La clase `OyenteBotonB` *implementa* a la interface `ActionListener` provista por Java. La implementación *redefine* el método `actionPerformed`.

Cuando el **objeto fuente de evento** `botonBlanco` de clase `JButton`, detecta la **acción** del usuario sobre el botón, crea implícitamente un **objeto evento** `e` de clase `ActionEvent`. El método `actionPerformed` redefinido en la implementación de la clase `OyenteBotonB`, recibe como parámetro el **objeto evento** creado implícitamente.

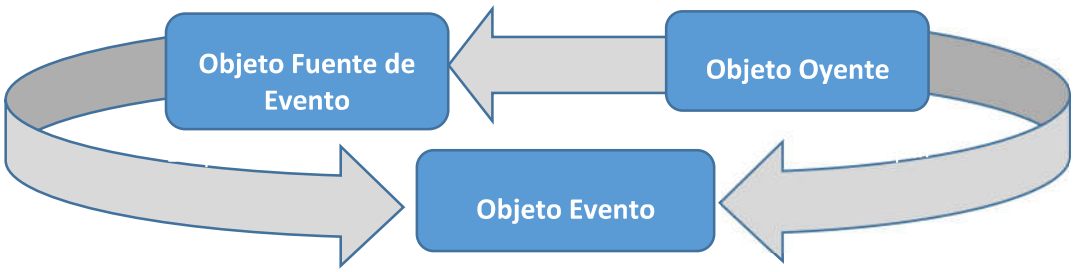
¡Importante!

El programador redefine el método `actionPerformed` que no va a ser invocado explícitamente por él mismo, sino que el mensaje que provoca su ejecución está incluido en algún método definido en una clase provista por Java. Esto es, el programador redefine un método que él mismo NUNCA VA A INVOCAR al menos explícitamente. El método recibe como argumento un **objeto evento** que tampoco ha sido creado explícitamente por el programador, sino que fue generado implícitamente en el momento que el objeto fuente de evento detectó la acción del usuario.

Objetos y Eventos

Algunos de las componentes de una GUI son **reactivas**, están asociadas a **objetos fuente de evento** que ***perciben*** **eventos externos** y ***disparan*** **eventos internos**. Cuando se dispara un evento interno se *crea implícitamente* un **objeto evento de software** que *pasa como parámetro* en un mensaje enviado a un **objeto oyente**.

El objeto oyente *se registra* al objeto fuente de evento asociado a la componente reactiva. En respuesta al mensaje el oyente ejecuta un método que corresponde a la acción del usuario.



Diferentes tipos de componentes requieren implementar diferentes interfaces para manejar los eventos internos que disparan los objetos ligados a las componentes.

Un botón dispara eventos llamados **eventos de acción** que son manejados por **oyentes de acción**. Es decir, un objeto fuente de evento de clase `JButton` detecta un **evento externo** provocado por la **acción** del usuario sobre el botón, *dispara* un **evento interno** que *crea* un objeto evento `e` de clase `ActionEvent`. Los objetos de la clase `ActionEvent` requieren implementar la interface `ActionListener`, escribiendo el código del método `actionPerformed` que recibe al objeto evento como parámetro.

La clase de un **objeto fuente de evento** determina así las clases de los **objetos evento** que se crearán implícitamente. La clase del **objeto evento** determina las interfaces de los **objetos oyente** que se deben implementar para establecer el comportamiento asociado a la componente gráfica sobre la que el usuario realizó una acción.

El **objeto oyente** es instancia de una clase que implementa una interface y redefine el método responsable de reaccionar ante la acción del usuario. El **objeto evento** es un parámetro para el método manejador del evento.

La acción del usuario

A partir del objeto evento generado implícitamente, puede obtenerse la acción del usuario asociada al objeto fuente de evento. Así el mensaje `getActionCommand()` permite definir un único oyente para dos o más componentes.

```
//Crea un oyente, los botones y registra el mismo oyente a ambos
  OyenteBoton o = new OyenteBoton();
  botonBlanco= new JButton("Blanco");
  botonBlanco.addActionListener(o);
  botonNegro = new JButton("Negro");
  botonNegro.addActionListener(o);
```

El **objeto evento** `e` mantiene la información del evento externo producido por la acción del usuario, que permite decidir cómo reaccionar:

```
private class OyenteBoton implements ActionListener {
     public void actionPerformed(ActionEvent e) {
      String op = (String) e.getActionCommand();
      if (op.equals("Blanco"))
         panelColor.setBackGround(Color.RED);
      else
         panelColor.setBackGround(Color.GREEN); }
  }
```

La instrucción

```
        String op = (String)e.getActionCommand();
```

Envía el mensaje `getActionCommand` al parámetro `e` para obtener información referida a la componente de la GUI que detecto la acción del usuario. En el caso de un objeto de clase `JButton`, por omisión la acción es el texto establecido en el botón. En un objeto de clase `JTextField` el mensaje `getActionCommand()` retorna el texto tipeado en la caja.

La fuente del evento

A partir del objeto evento generado implícitamente puede obtenerse el objeto fuente de evento enviando el mensaje `getSource()`.

Caso de Estudio: Botonera

Implementar una GUI que muestre cinco botones, inicialmente verdes. Cada vez que se oprime un botón, el color de ese botón pasa a ser rojo. Si un botón se oprime dos veces no se provoca ningún cambio, esto es, permanece rojo.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class Botonera extends JFrame {
//Declara un arreglo de botones
   private JButton []botones;
   private JPanel panelBotones;
   public Botonera() {
      setSize(400, 200);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      panelBotones = new JPanel();
     armarBotonera();
     getContentPane().add(panelBotones);}
private void armarBotonera (){
/*Crea cada botón, registra su oyente y lo inserta en el panel*/
  botones = new JButton  [5];
  Oyente oyente = new Oyente();
  for (int i=0;i<5;i++){
    botones[i] = new JButton();
    botones [i].setText(i+"");
    botones [i].addActionListener(oyente);
    panelBotones.add(botones [i]); }
}
private class Oyente implements ActionListener {
  public void actionPerformed(ActionEvent e) {
     JButton b = (JButton) e.getSource();
     b.setBackground(Color.RED);
     b.setText(" ");}
  }
```

```
}
```

En este caso se envía el mensaje `getSource()` al objeto evento `e` para obtener la referencia al botón que recibió el mensaje. Luego, esta referencia se usa para enviarle el mensaje `setBackground` al botón que recibió la acción del usuario de modo que se pinte de rojo.

Los botones están agrupados en un arreglo cuyas componentes son de clase `JButton`. Cada botón está rotulado con un número entre 0 y 4. Cada componente del arreglo se registra a un mismo oyente y se inserta en el panel.

Eventos del Mouse

Cuando un usuario interactúa con una interface gráfica realiza acciones que generan eventos de **alto nivel** y de **bajo nivel**. Estos eventos pueden ser percibidos por la aplicación o no. En los ejemplos anteriores se han percibido y manejado eventos de alto nivel.

Si el usuario oprime el mouse sobre un botón y lo suelta sobre el mismo botón, aunque antes de soltarlo lo arrastre fuera de esta componente gráfica, el objeto fuente de evento captura el evento externo y dispara un **evento interno de alto nivel**, esto es, crea un **objeto evento** de clase `ActionEvent`.

Si en cambio el usuario oprime el mouse sobre un botón, lo arrastra y suelta fuera del botón, no se dispara un evento interno de alto nivel y por lo tanto no se crea un objeto evento.

Sin embargo, en ambos casos, se disparan **eventos internos de bajo nivel**, relacionados con el mouse e independientes de la componente gráfica. Estos eventos están asociados a objetos de clase `MouseEvent`, uno creado como consecuencia del click sobre una componente, otro al arrastrar y otro al soltar el botón. Otros eventos de bajo nivel se producen cuando se oprimen teclas o se manipulan las ventanas.

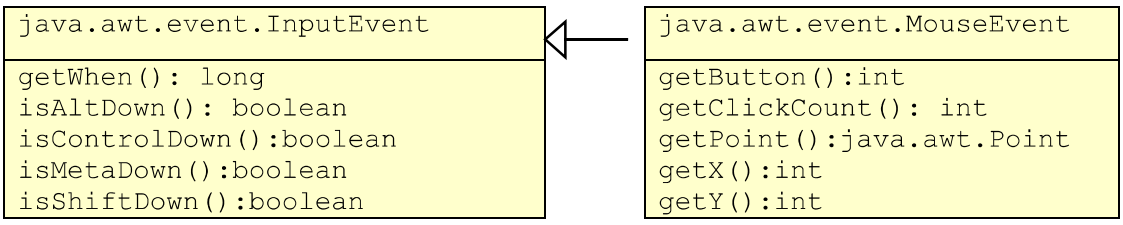
La clase de un **objeto evento** determina las interfaces que se deben implementar para establecer el comportamiento de los **objetos oyentes** asociados a la componente gráfica sobre la que el usuario realizó una acción:

Objeto Evento	Interface de oyente	Manejador
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
MouseEvent	MouseListener MouseMotionListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mouseClicked(MouseEvent)
KeyEvent	KeyListener	keyPressed(KeyEvent) keyRealesed(KeyEvent) keyTyped(KeyEvent)

Los atributos del objeto evento brindan información referida a la ubicación del cursor del mouse, cuántos clicks se hicieron, qué botón que se apretó, etc.

La clase del objeto oyente asociada a una componente que detecta acciones del mouse, tiene que implementar los métodos provistos por la interface `MouseListener` o `MouseMotionListener`. El método `addMouseListener` permite registrar el objeto oyente a la componente que va a percibir los eventos del mouse.

Dado que la clase `MouseEvent` hereda de `InputEvent`, sobre un objeto de la clase `MouseEvent` también pueden usarse los métodos definidos en la clase `InputEvent`.



En el siguiente ejemplo el color de un panel cambiará según los diferentes eventos del mouse que se produzcan como consecuencia de las acciones del usuario. En otro panel se muestra la acción del usuario y la posición del indicador del mouse sobre la componente.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class ControlMouse extends JFrame{
//Atributos de instancia
   private JPanel panelMouse, panelEtiqueta;
   private JLabel eventoMouse;
   private Container contenedor;
//Constructor
public ControlMouse() {
  setSize(700,700);
//Establece los atributos del panel de contenido
  contenedor = getContentPane();
  contenedor.setLayout(new GridLayout(2,0));
//Crea y establece los valores de los atributos de los paneles
  panelMouse = new JPanel();
  panelMouse.setBackground(Color.WHITE);
  panelMouse.setPreferredSize
    (new java.awt.Dimension(300, 150));
  panelEtiqueta = new JPanel();
/*Crea una etiqueta, establece los valores de sus atributos y la
inserta en el panel*/
  eventoMouse = new JLabel();
  eventoMouse.setPreferredSize
   (new    java.awt.Dimension(300,70));
  panelEtiqueta.add(eventoMouse);
//Crea el oyente del mouse y lo registra al panel
  OyenteMouse escuchaMouse = new OyenteMouse();
  panelMouse.addMouseListener(escuchaMouse);
```

```
//Inserta los paneles en el panel de contenido
  contenedor.add(panelMouse);
  contenedor.add(panelEtiqueta);}
private class OyenteMouse implements MouseListener{
public void mouseClicked (MouseEvent e)      {
    String s = new String();
    eventoMouse.setText(s.format("Cliqueó en [%d,%d]",
    e.getX(), e.getY()));
    panelMouse.setBackground(Color.BLUE);}
public void mouseEntered (MouseEvent e)      {
    String s = new String();
    eventoMouse.setText(s.format("Entró en[%d,%d]",
    e.getX(), e.getY()));
    panelMouse.setBackground(Color.RED);}
public void mouseExited (MouseEvent e) {
     String s = new String();
      eventoMouse.setText(s.format(" Salió en [%d,%d]",
     e.getX(), e.getY()));
      panelMouse.setBackground(Color.GREEN);}
public void mouseReleased (MouseEvent e)     {
    String s = new String();
      eventoMouse.setText(s.format(" Soltó en [%d,%d]",
     e.getX(), e.getY()));
      panelMouse.setBackground(Color.MAGENTA);}
public void mousePressed (MouseEvent e)      {
    String s = new String();
      eventoMouse.setText(s.format(" Presionó en [%d,%d]",
     e.getX(), e.getY()));
      panelMouse.setBackground(Color.YELLOW);}}
}
```

La implementación de la interface `MouseListener` exige redefinir cinco métodos para controlar cinco eventos sobre el ratón:

`getButton():int` `getClickCount(): int`

`java.awt.event.MouseListener`
`mouseClicked(e:MouseEvent): void`
`mousePressed(e:MouseEvent): void`
`mouseReleased(e:MouseEvent): void`
`mouseEntered(e:MouseEvent): void`
`mouseExited(e:MouseEvent): void`

La clase oyente puede implementar las dos interfaces provistas para detectar acciones sobre el ratón:

```
private class OyenteMouse implements MouseListener ,
                      MouseMotionListener {
…
public void mouseDragged (MouseEvent e) {
     String s = new String();
     eventoMouse.setText(s.format(" Arrastró a [%d,%d]",
      e.getX(),e.getY()));}
public void mouseMoved(MouseEvent e)  {
    String s = new String();
    eventoMouse.setText(s.format(" Se movió a [%d,%d]",
     e.getX(),e.getY()));}
}
```

En este caso la clase `OyenteMouse` redefine los cinco métodos anteriores y además los dos que corresponden al ratón en movimiento. La estructura de la clase es entonces:

```
private class OyenteMouse implements MouseListener ,
                      MouseMotionListener {
 public void mouseClicked (MouseEvent e)     { … }
 public void mousePressed(MouseEvent e)      { … }
 public void mouseReleased (MouseEvent e)    { … }
 public void mouseEntered (MouseEvent e)     { … }
 public void mouseExited (MouseEvent e)     { … }
 public void mouseDragged (MouseEvent e)     { … }
 public void mouseMoved(MouseEvent e)       { … }
}
```

Con frecuencia una interface de eventos brinda más servicios que los que necesitamos. Por ejemplo, la interface `MouseListener` ofrece cinco servicios, si la GUI solo va a reaccionar ante un click del mouse, se define el código de `mouseClicked`, los otros cuatro quedan sin especificar. Sin embargo, si se define una clase que implementa a esta interface, es necesario implementar todos los servicios provistos.

Encapsulamiento y GUI

En el diseño de una aplicación, la solución se modula de modo tal que cada clase pueda implementarse sin depender de las demás. En el desarrollo de una aplicación en la cual la interacción con el usuario se realiza a través de una GUI, la clase que implementa la interface gráfica de usuario usa a las clases que modelan el problema, sin conocer detalles de la representación. Análogamente, las clases que modelan el problema se diseñan e implementan si saber si la entrada y salida va a hacerse por consola o mediante una GUI.

El siguiente caso de estudio ilustra cómo definir una GUI que permita traducir palabras de inglés a español y de español a inglés, desconociendo cómo se representa internamente el diccionario. La clase `Traductor` es cliente de los servicios provistos por la clase `Diccionario`. La implementación de estos servicios queda encapsulada en la clase proveedora y es transparente para la clase cliente.

Caso de Estudio: Traductor

Implemente una clase GUI_Traductor que permita mostrar la traducción de una palabra ingresada en inglés a español o una palabra ingresada en español a inglés.

La clase GUI_Traductor está asociada a Diccionario. La clase Diccionario encapsula a una colección de pares de palabras, la primera palabra del par está en inglés y la segunda es su traducción al español. Claramente un objeto de clase Diccionario puede utilizarse para traducir de inglés a español o de español a inglés. El diagrama de clases para este problema es:

Par
pal1,pal2:String
<<Constructor>> Par(p1,p2:String) <<Consultas>> obtenerPal1() :String

Diccionario
T [] Par
<<Constructor>> Diccionario() <<Comandos>> insertar(p:Par) <<Consultas>> traducirEngSpa

GUI_Traductor
panelPalabras, panelControl:JPanel botonES,botonEI:JButton cartelEntrada :JLabel cajaPalabra: JTextField diccionario : Diccionario palabra:String <<Constructor>>

obtenerPal2() :String	(pan:String):String traducirEspIng (pal:String):String	GUI_Traductor(tit:String, dic:Diccionario)

Inicialmente la interfaz corresponde a la siguiente figura:

Cuando el usuario oprime intro, luego de completar la cada de texto, se habilitan los dos botones. Cada botón está asociado a un oyente que envía un mensaje al diccionario para buscar la palabra. Si existe la traducción se muestra en un panel de diálogo. En caso contrario se muestra un mensaje, también en un panel de diálogo.

La solución puede implementarse como sigue:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
class GUITraductor extends JFrame {
//Atributos gráficos
        private JPanel panelPalabras, panelControl;
        private JButton botonES,botonEI;
        private JLabel cartelEntrada;
        private JTextField cajaPalabra;
//Atributos de la aplicación
        private Diccionario diccionario;
        private String palabra;

        public GUITraductor (String tit,Diccionario dicc) {
//Establece la apariencia del frame
         super(tit);
         setSize(400, 200);
         setDefaultCloseOperation(EXIT_ON_CLOSE);
         diccionario = dicc;
//Crea los objetos gráficos
        panelPalabras = new JPanel();
       panelControl = new JPanel();
       botonES = new BotonTraductor();
       botonEI = new BotonTraductor();
        cartelEntrada = new JLabel();
        cajaPalabra = new JTextField ("",15);
        armaGUI();}
private void armaGUI(){
        palabra = " ";
        armarPaneles();
       armarBotones();
//Establece los atributos de las etiquetas
```

```
        cartelEntrada.setText("Ingrese una palabra");
        cartelEntrada.setSize(120,15);
        cartelEntrada.setPreferredSize(new
                        java.awt.Dimension(120,15));
       cartelEntrada.setBorder (new LineBorder(
                              new Color(0,0,0), 1, false));
//Establece los atributos de la caja y la registra al oyente
        cajaPalabra.setSize(120,15);
        cajaPalabra.setPreferredSize(new
java.awt.Dimension(120,15));
       cajaPalabra.setBorder(new LineBorder(new
                               Color(100,100,100), 1, false));
       OyentePalabra oyente = new OyentePalabra();
        cajaPalabra.addActionListener(oyente);
//Inserta los objetos gráficos en los paneles
        panelPalabras.add(cartelEntrada,BorderLayout.NORTH);
        panelPalabras.add(cajaPalabra,BorderLayout.CENTER);
        panelControl.add(botonES);
        panelControl.add(botonEI);
//Establece el organizador del panel de contenido
        getContentPane().setLayout(new BorderLayout());
//Inserta los paneles en el panel de contenido
        getContentPane().add(panelPalabras,BorderLayout.NORTH);
        getContentPane().add(panelControl,BorderLayout.SOUTH); }
private void armarPaneles(){
//Establece los atributos de los paneles
        panelPalabras.setLayout(new BorderLayout());
        panelPalabras.setSize(400,90);
        panelPalabras.setPreferredSize (new
                      java.awt.Dimension(400,90));
        panelControl.setLayout(new FlowLayout());
        panelControl.setSize(400,30);
        panelControl.setPreferredSize (new
                           java.awt.Dimension(400,30));
        panelControl.setBackground(Color.GREEN);}
private void armarBotones(){
//Establece los atributos de los botones y registra sus oyentes
      estadoBotones(false);
       botonES.setText ("English to Spanish");
       OyenteBotonES oyenteES = new OyenteBotonES();
        botonES.addActionListener(oyenteES);
       botonEI.setText ("Español a Inglés");
       OyenteBotonEI oyenteEI = new OyenteBotonEI();
        botonEI.addActionListener(oyenteEI);}
private void estadoBotones(boolean estado){
//Habilita o deshabiliata botones
         botonEI.setEnabled(estado);
         botonES.setEnabled(estado);}
private class OyentePalabra implements ActionListener {
       public void  actionPerformed(ActionEvent event){
            palabra =  (cajaPalabra.getText());
            estadoBotones(true);}
}
private class OyenteBotonES implements ActionListener {
   public void  actionPerformed(ActionEvent event){
```

```
      JOptionPane dialogo = new JOptionPane();
      String traducida = diccionario.traducirEngSpa(palabra);
      if (traducida == null)
         dialogo.showMessageDialog(null,
        "La palabra no figura en el diccionario", "",
         dialogo.INFORMATION_MESSAGE );
      else{
         cajaPalabra.setText("");
         estadoBotones(false);
         dialogo.showMessageDialog(null,
         "La traduccion de "+palabra+ " es " +traducida, "",
         dialogo.INFORMATION_MESSAGE );}}
}
private class OyenteBotonEI implements ActionListener {
    public void  actionPerformed(ActionEvent event){
      JOptionPane dialogo = new JOptionPane();
      String traducida = diccionario.traducirEspIng(palabra);
      if (traducida == null)
         dialogo.showMessageDialog(null,
         "La palabra no figura en el diccionario","",
         dialogo.INFORMATION_MESSAGE );
       else{
         cajaPalabra.setText("");
         estadoBotones(false);
         dialogo.showMessageDialog(null,
         "La traduccion de "+palabra+ " es "+traducida,"",
         dialogo.INFORMATION_MESSAGE );}}
   }
}
class BotonTraductor extends JButton{
public BotonTraductor(){
     setSize(20,5);
     setBackground(Color.YELLOW);
     setBorder(new LineBorder(new Color(0,0,0), 1, false));}
}
```

El objeto ligado a la variable `palabra` se inicializa en el oyente de la caja y se usa para buscar la traducción en las clases de los oyentes.

El problema planteado se modela a partir de los tipos de componentes descriptos antes. La principal diferencia con los ejemplos propuestos es que la implementación de la clase oyente requiere acceder a un diccionario. El diccionario es una estructura de datos que mantiene pares de palabras, la primera palabra del par está en inglés y la segunda en español.

Cuando el usuario realiza una acción sobre `botonES` se envía el mensaje `traducirEngSpa(palabra)` al diccionario. Análogamente, cuando el usuario realiza una acción en `botonEI` se envía el mensaje `traducirEngSpa(palabra)` al diccionario.

La clase `GUITraductor` **usa** a la clase `Diccionario` como una caja negra, conoce únicamente la interface y el contrato. Los datos pueden estar modelados por un arreglo o un objeto de clase `Vector`, las búsquedas pueden implicar un recorrido secuencial o la estructura puede estar ordenada y se aplica entonces búsqueda binaria. Todas esas cuestiones quedan escondidas para la interfaz. De manera análoga, es posible cambiar el diseño y la implementación de la GUI, sin afectar a la clase `Diccionario`.

Otra característica que aparece en esta GUI es que los botones se habilitan y deshabilitan.

Herencia, Polimorfismo y GUI

Una GUI en Java se construye a partir de objetos gráficos de algunas de las clases gráficas provistas por los paquetes AWT y Swing o de clases derivadas. Conceptualmente la implementación de GUI está fuertemente ligada a los conceptos de herencia, polimorfismo y vinculación dinámica.

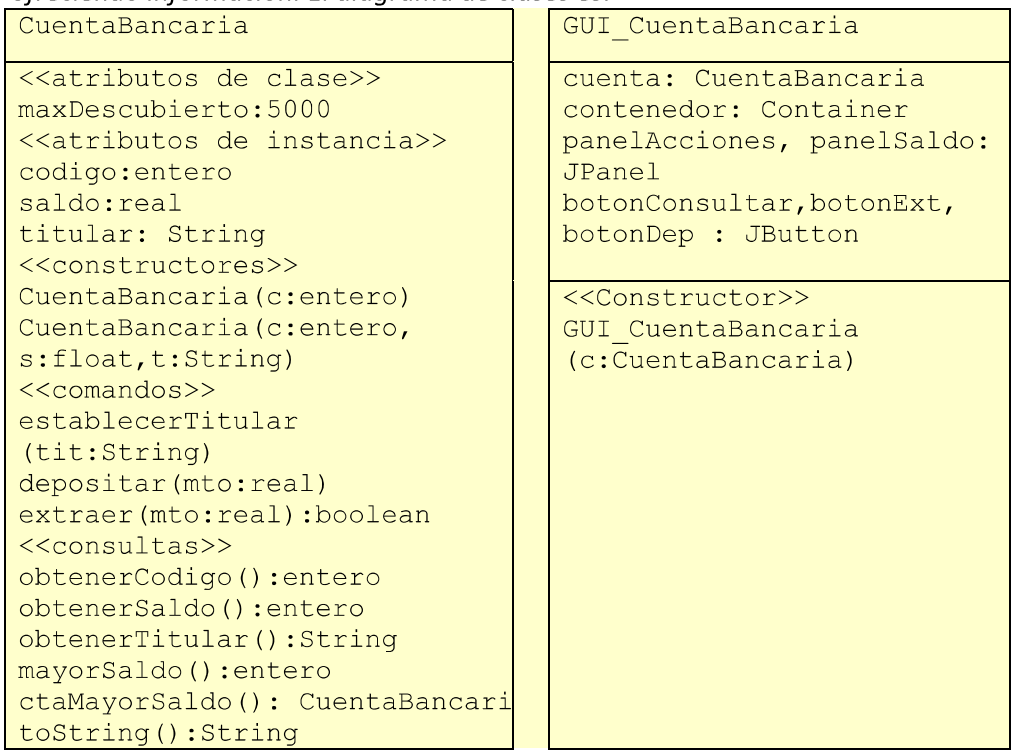
En los ejemplos anteriores se han construido interfaces gráficas creando componentes de alguna clase derivada de la clase `JComponent` provista por el paquete Swing. Esta clase deriva a su vez de `Component` provista por AWT y deriva directamente de `Object`. Cuando una clase extiende a otra **hereda** todos sus métodos de modo que es posible reusar los servicios provistos por las clases más generales.

La clase `Container` de AWT brinda el método `add` heredado por todos sus descendientes. El parámetro formal del método `add` es de clase `Component`.

El siguiente caso de estudio nos permite ilustrar la vinculación entre los conceptos de herencia y polimorfismo con la construcción de interfaces gráficas de usuarios.

Caso de Estudio: Cuenta Bancaria

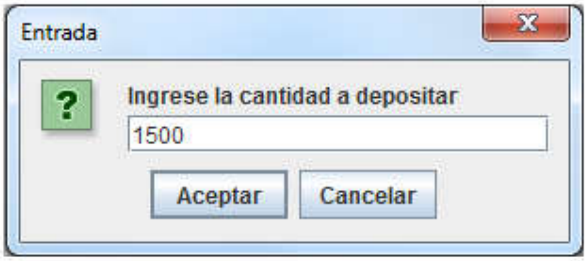
Implemente una GUI que brinde botones para efectuar un depósito o extracción en una cuenta bancaria o para consultar el saldo. Los dos primeros botones se insertan en un panel y el tercero en otro. Si el usuario elige depositar o extraer la interfaz muestra un campo de texto para tipear el monto. La acción de extraer requiere verificar que el monto sea menor al saldo de la cuenta. Si no es posible concretar la extracción, aparece un mensaje indicándolo. Tanto el depósito, como la extracción realizada y la consulta de saldo, concluyen con un panel de diálogo ofreciendo información. El diagrama de clases es:



Requiere código > 0 y saldo>=0	

Inicialmente la interfaz corresponde a la siguiente figura:

Si el usuario oprime Depositar aparece el campo de texto:



Y una vez que el usuario oprime Intro,

El código que modela la solución es:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import javax.swing.*;
class GUI_CuentaBancaria extends JFrame {
//Objetos del problema y objetos gráficos
    private CuentaBancaria cuenta;
    private Container contenedor;
    private JPanel panelAcciones, panelSaldo;
    private JButton botonConsultar,botonExt, botonDep;
    public GUI_CuentaBancaria(CuentaBancaria cta) {
//Establece los atributos del frame
     super("Cuenta Bancaria "+cta.obtenerCodigo());
```

```
     setSize(210, 210);
     setDefaultCloseOperation(EXIT_ON_CLOSE);
// Inicializa la cuenta bancaria
     cuenta =cta;
//Obtiene el panel de contenido y crea los paneles
     contenedor = getContentPane();
     panelAcciones = new JPanel();
     panelSaldo = new JPanel();
//Crear los botones
      botonDep = new JButton();
     botonExt = new JButton();
     botonConsultar = new JButton();
     armarBotones();
     armarPaneles();
}
private void armarBotones(){
//Establece atributos de los botones y los vincula a oyentes
     botonDep.setText("Depositar");
     botonDep.setPreferredSize(new Dimension(124, 50));
     botonDep.setSize(150, 50);
      botonDep.setBorder
        (BorderFactory.createCompoundBorder(
         new LineBorder(new java.awt.Color(0, 0, 0), 1, false),
                        null));
     OyenteDepositar oDepositar = new OyenteDepositar();
     botonDep.addActionListener(oDepositar);

     botonExt.setText("Extraer");
     botonExt.setPreferredSize(new Dimension(124, 50));
     botonExt.setSize(150, 50);
      botonExt.setBorder
         (BorderFactory.createCompoundBorder(
          new LineBorder(new java.awt.Color(0, 0, 0), 1, false),
                        null));
     OyenteExtraer oExtraer = new OyenteExtraer();
     botonExt.addActionListener(oExtraer);
     botonConsultar.setText("Consultar Saldo");
     botonConsultar.setPreferredSize(new Dimension(136, 30));
     botonConsultar.setSize(150, 30);
     botonConsultar.setBorder  (BorderFactory.createBevelBorder
        (BevelBorder.LOWERED));
     OyenteConsultar oConsultar = new OyenteConsultar();
     botonConsultar.addActionListener(oConsultar);}
private void armarPaneles() {
// Diagramado de los paneles
      contenedor.setLayout(new BorderLayout());
      panelAcciones.setBorder
        (BorderFactory.createEtchedBorder(BevelBorder.LOWERED));
      panelAcciones.setPreferredSize(new Dimension(160, 130));
      panelAcciones.setSize(160, 125);
// Agrega botones a los paneles
     panelAcciones.add(botonDep);
      panelAcciones.add(botonExt);
     panelSaldo.add(botonConsultar);
// Agrega los paneles al contenedor
```

```
      contenedor.add(panelAcciones, BorderLayout.NORTH);
      contenedor.add(panelSaldo, BorderLayout.SOUTH);     }
private class OyenteDepositar implements ActionListener {
 public void actionPerformed(ActionEvent event){
  float dep;
  String deposito;
  JOptionPane dialogo = new JOptionPane();
  deposito=dialogo.showInputDialog("Ingrese la cantidad a depositar"
);
  if ((deposito != null) && (deposito.length() > 0)){
    dep = Float.parseFloat(deposito);
    dialogo.showMessageDialog(null,"Usted depositó " + dep+ "
          pesos","Depósito", JOptionPane.PLAIN_MESSAGE );
     cuenta.depositar(dep);}
  }
}
private class OyenteExtraer implements ActionListener {
      public void actionPerformed(ActionEvent event){
  float ext;
  String extraccion;
  JOptionPane dialogo = new JOptionPane();
  extraccion=dialogo.showInputDialog("Ingrese la cantidad a extraer"
);
  if ((extraccion != null) && (extraccion.length() > 0)){
   ext = Float.parseFloat(extraccion);
   if (cuenta.extraer(ext)){
       dialogo.showMessageDialog( null,
                 "Usted extrajo " + ext+ " pesos"," Extracción",
     JOptionPane.PLAIN_MESSAGE );
      cuenta.extraer(ext);}
    else
      dialogo.showMessageDialog( null,
          "Usted NO puede extraer esa cantidad",  "Advertencia",
           JOptionPane.WARNING_MESSAGE ); }
  }
}
private class OyenteConsultar implements ActionListener {
  public void actionPerformed(ActionEvent event){
     JOptionPane dialogo = new JOptionPane();
     if (cuenta.obtenerSaldo()>=0)
            dialogo.showMessageDialog(null,
      "Usted tiene un saldo de " + cuenta.obtenerSaldo()+
      " pesos","SALDO",dialogo.INFORMATION_MESSAGE );
     else
       dialogo.showMessageDialog(null,
          "Usted está en descubierto en " +
          (-1)*cuenta.obtenerSaldo() + " pesos",
           "SALDO", JOptionPane.ERROR_MESSAGE );}
   }
}
```

En las instrucciones:

```
     panelAcciones.add(botonDep);
     contenedor.add(panelAcciones);
```

El método `add` es **polimórfico**, el parámetro real puede ser cualquier clase derivada de `Component`. En la primera instrucción un objeto de clase `JPanel` recibe el mensaje `add` con un parámetro de clase `JButton`. En la segunda un objeto de clase `Container` recibe el mensaje `add` con un parámetro de clase `JPanel`.

En la instrucción:

```
botonDep.addActionListener(oDepositar);
```

El parámetro real `oDepositar` es de clase `OyenteDepositar` que deriva directamente de `Object` e implementa a la interface `ActionListener` provista por Java. En la implementación de `addActionListener` claramente el parámetro formal no puede ser de la misma clase que el parámetro real, que se define en cada aplicación. El método `addActionListener` es **polimórfico**, recibe parámetros de distintas clases, todas derivadas de una misma clase base.

```
botonDep.addActionListener(oDepositar);
botonExt.addActionListener(oExtraer);
botonConsultar.addActionListener(oConsultar);
```

La implementación de oyentes como clases internas solo resulta adecuada para definir GUI simples como las de los ejemplos y casos de estudio propuestos. Aun en estos casos, es conveniente estructurar adecuadamente las soluciones dividiendo el problema en subproblemas. En este ejemplo se modula la solución definiendo métodos internos `armarBotones()` y `armarPaneles()`.

La clase `GUI_CuentaBancaria` usa los servicios provistos por `CuentaBancaria` sin conocer la representación de los datos ni la implementación de las operaciones. En tanto se mantenga el contrato, cualquiera de las dos clases puede modificarse sin afectar a la otra.

Correctitud y GUI

La verificación de una aplicación no garantiza que es correcta pero permite asegurar que funciona de acuerdo a los requerimientos para un conjunto de casos de prueba. Cuando la interacción con el usuario se realiza a través de una GUI el flujo de la ejecución queda determinado por eventos que se generan en respuesta a las acciones del usuario. En este contexto es muy difícil o incluso imposible, anticipar todos los flujos de ejecución posibles. En la verificación se analiza el comportamiento de cada componente considerando diferentes flujos de ejecución.

Caso de Estudio: Extracciones en un cajero

Implemente una GUI que permita realizar extracciones de sumas fijas en un cajero. La GUI incluye tres paneles. En el primero aparece un combo box y un botón, en el segundo una etiqueta y en el tercero un arreglo de 5 botones rotulados con números 100, 200, 500, 750 y 1000. El tercer panel inicialmente no está visible.

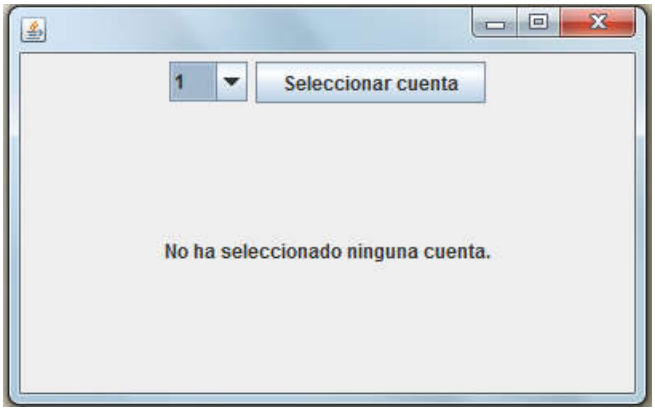
El combo box se inicializa con los códigos de las cuentas bancarias almacenados en una estructura de datos asociada. Cuando el usuario selecciona el código de una cuenta bancaria y oprime el botón seleccionar, se recupera la cuenta bancaria de la cartera y se establece el saldo y el titular como rótulo en la etiqueta. Se pone visible el panel con el arreglo con 5 botones. El usuario selecciona el botón que indica el monto de la extracción. Si el monto es menor que el saldo aparece un panel de opciones mostrando el saldo de la cuenta y se solicita que se confirme o cancele la operación. Si el usuario confirma se modifica el saldo de la cuenta

y se vuelve al estado inicial. Si no es posible realizar la extracción aparece un diálogo adecuado. En este caso la clase GUI_Cajero está asociada a la clase CarteraClientes que a su vez usa los servicios de CuentaBancaria.

Para verificar la aplicación se propone la siguiente una clase tester:

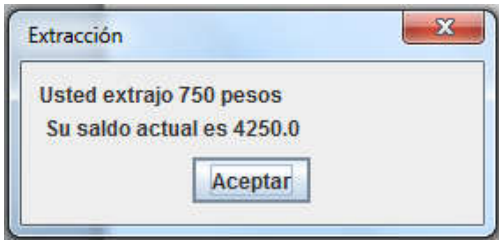
```
class testCajero{
 public static void main(String[] args) {
   CarteraClientes cc= new CarteraClientes(3);
   CuentaBancaria c1=new CuentaBancaria(1, "Juan Paredes");
   c1.establecerSaldo(10000);
   CuentaBancaria c2=new CuentaBancaria(2, "Juan Carlos Petruza");
   c2.establecerSaldo(8000);
   CuentaBancaria c3=new CuentaBancaria(151, "Mariana Marini");
   cc.insertar(c1);
   cc.insertar(c2);
   cc.insertar(c3);
   c3.establecerSaldo(5000);
   GUI_Cajero cajero = new GUI_Cajero(cc);
   cajero.setVisible(true);}
}
```

La clase tester establece las cuentas bancarias de la cartera, sin embargo la verficación depende de los eventos generados por el usuario sobre la GUI, que inicialmente aparecerá como muestra la figura:



Si el usuario selecciona la cuenta 151 de la caja de opciones y oprime el botón:

Si el usuario oprime el botón rotulado con 750, aparece un panel de diálogo como el que sigue:



Y la GUI vuelve a solicitar que se seleccione una cuenta. El código de `GUI_Cajero` es:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
class GUI_Cajero extends JFrame {
//Objetos del problema y objetos gráficos
    private CarteraClientes cuentas;
    private CuentaBancaria cuentaActual;
    private Container contenedor;
    private JPanel panelSeleccion, panelInfo, panelCantidad;
    private JButton botonSeleccionar;
    private JButton[] botonera;
    private JLabel labelInfo;
    private JComboBox comboCuentas;
public GUI_Cajero(CarteraClientes ctas) {
// Inicializa la colección de cuentas
        cuentas =ctas;
//Obtiene el panel de contenido y crea los paneles
        contenedor = getContentPane();
        panelSeleccion = new JPanel();
        panelInfo = new JPanel();
        panelCantidad = new JPanel();
//Crea la etiqueta
        labelInfo = new JLabel("No ha seleccionado ninguna
cuenta.");
        armarComboCuentas();
        armaBotones();
//Establece los atributos del frame
        setSize(400, 250);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        armarGUI();
}
private void armarGUI() {
// Establece el diagramado del panel de contenido
        contenedor.setLayout(new BoxLayout(contenedor,
                             BoxLayout.Y_AXIS));
//Inserta los objetos gráficos a los paneles
        panelSeleccion.add(comboCuentas);
        panelSeleccion.add(botonSeleccionar);
        panelInfo.add(labelInfo);
        panelCantidad.setVisible(false);
//Agrega los paneles al contenedor
        contenedor.add(panelSeleccion);
        contenedor.add(panelInfo);
        contenedor.add(panelCantidad);}
private void armarComboCuentas(){
```

```
        String[] lista= new String[cuentas.cantElementos()];
        int n; CuentaBancaria m;
        for (int i=0; i< cuentas.cantElementos(); i++){
            m = cuentas.obtenerCuenta(i);
            n = m.obtenerCodigo();
            lista[i] = ""+n; }
        comboCuentas = new JComboBox(lista);}
private void armaBotones(){
//Establece atributos del boton y lo vincula a su oyente
        botonSeleccionar = new JButton();
        botonSeleccionar.setText("Seleccionar cuenta");
        botonSeleccionar.addActionListener(new OyenteSeleccionar());
//Crea la botonera y vincula todos los botones al oyente
        int cantBotones=5;
        botonera= new JButton[cantBotones];
        for (int i=0; i<botonera.length;i++){
            botonera[i]= new JButton();
            botonera[i].addActionListener(new OyenteExtraccion());}
        botonera[0].setText("100");
        botonera[1].setText("200");
        botonera[2].setText("500");
        botonera[3].setText("750");
        botonera[4].setText("1000");
        for (int i=0; i<5;i++){
            panelCantidad.add(botonera[i]);}}
    private class OyenteSeleccionar implements ActionListener {
        public void actionPerformed(ActionEvent event){
            int nroCuenta= Integer.parseInt
                (comboCuentas.getSelectedItem().toString());
            cuentaActual=cuentas.recuperarCuenta(nroCuenta);
            if (cuentaActual!=null){
                String mensaje= "Titular: " +
                cuentaActual.obtenerTitular() + " , Saldo="+
                cuentaActual.obtenerSaldo();
                labelInfo.setText(mensaje);
                panelCantidad.setVisible(true);}
        }
    }
    private class OyenteExtraccion implements ActionListener {
        public void actionPerformed(ActionEvent event){
            JButton boton= (JButton) event.getSource();
            int extraccion= Integer.parseInt(boton.getText());
            JOptionPane dialogo = new JOptionPane();
            if (cuentaActual.obtenerSaldo()>=extraccion){
                cuentaActual.extraer(extraccion);
                dialogo.showMessageDialog( null,
                    "Usted extrajo " + extraccion+
                " pesos \n Su saldo actual es "
             + cuentaActual.obtenerSaldo()," Extracción",
               JOptionPane.PLAIN_MESSAGE );
                labelInfo.setText("Seleccione una cuenta ");
                cuentaActual=null;
                panelCantidad.setVisible(false);    }
            else
                dialogo.showMessageDialog( null,
```

```
                    "Usted NO puede extraer esa cantidad",
                "Advertencia", JOptionPane.WARNING_MESSAGE ); }
    }
}
```

El oyente de `botonSeleccionar` recupera la cuenta bancaria con código `nroCuenta` de la cartera de cuentas bancarias y la asigna a `cuentaActual`.

```
cuentaActual=cuentas.recuperarCuenta(nroCuenta);
```

Un mismo objeto va a ser referenciado desde la colección y desde el atributo de instancia `cuentaActual` de `GUI_Cajero`. De modo que cuando el oyente de uno de los botones de la botonera envía el mensaje:

```
cuentaActual.extraer(extraccion);
```

Se modifica el saldo de la cuenta en la cartera de cuentas bancarias.

La clase `GUI_Cajero` usa los servicios de `CarteraClientes` y `CuentaBancaria` sin conocer la representación de los datos ni la implementación de las operaciones.

Productividad y GUI

Una de las razones que hicieron de Java un lenguaje muy utilizado en la industria de software es porque favorece la **reusabilidad** a través del uso de **paquetes** y la **extensibilidad** a través de **herencia**. Cada paquete que se importa se usa como una caja negra, conocimiento únicamente la interface de las clases provistas. Cada clase provista por el lenguaje puede ser utilizada para crear objetos o para definir clases más especializadas, con atributos y servicios específicos de la aplicación

El lenguaje brinda muchos recursos prefabricados que contribuyen a mejorar la productividad, entre ellos una colección de clases gráficas organizadas en forma jerárquica y distribuidas en paquetes. El programador importa solo los paquetes que su aplicación requiere. Cada paquete ofrece una colección de clases que pueden ser usadas para:

- Crear objetos gráficos asociados a las componentes de la interfaz
- Definir clases más específicas a partir de las cuales se crearán componentes.

AWT y **Swing** brindan clases que pueden especializarse para crear botones, cajas de texto, menúes, etc. Una de las ventajas de Swing sobre AWT es que permite desarrollar aplicaciones con una apariencia similar a la de la plataforma subyacente con muy poco esfuerzo. Swing no reemplaza a AWT sino que la usa y agrega nuevas clases.

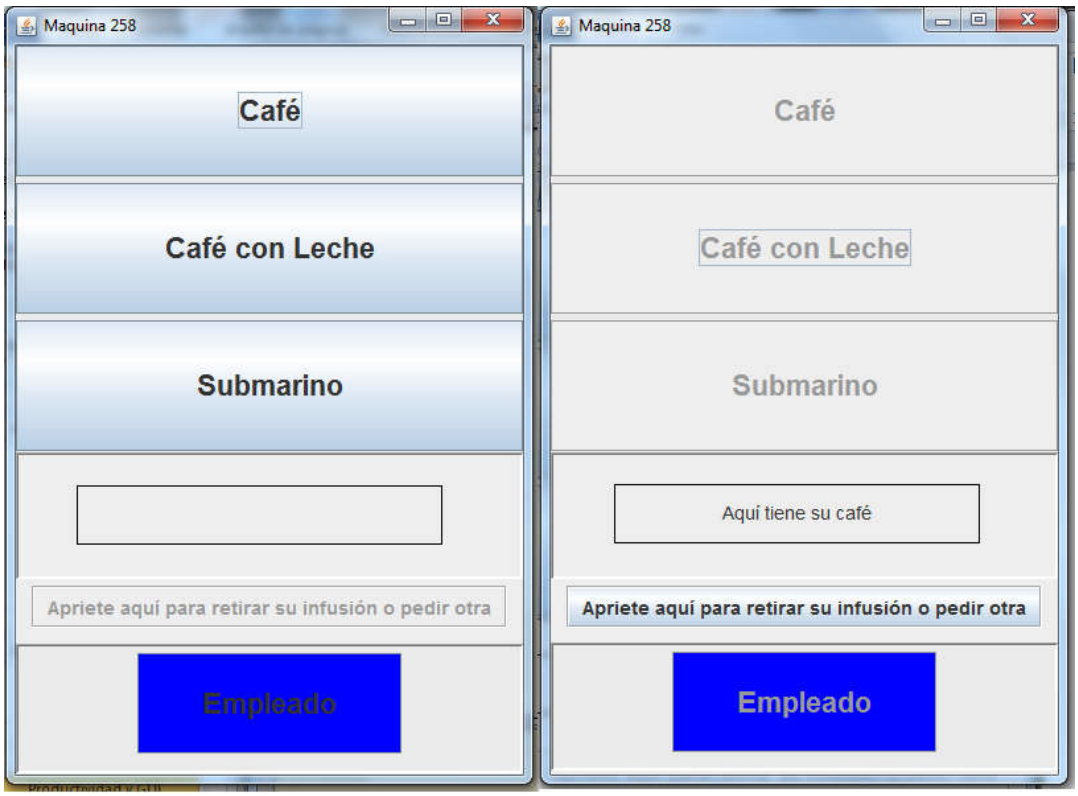
Las clases de los diferentes paquetes están vinculadas a través de relaciones de **asociación** y **herencia**. Un frame por ejemplo tiene una dimensión y tres botones, entre otras componentes. Las clases `Dimension` y `JButton` están asociadas a `JFrame`. La clase `JButton` a su vez hereda de `JComponent`.

En los ejemplos y casos de estudio propuestos se definen clases que extienden a `JFrame` para implementar GUI y se usadan algunas de las clases derivadas de `JComponent` para crear los objetos gráficos asociados a las componentes de estas GUI. En el siguiente caso de estudio se crean botones de una clase definida específicamente para la aplicación. La clase `BotonME` hereda de `JButton`.

Una instancia de `BotonME` puede recibir cualquiera de los mensajes provistos por `JButton`. La definición de esta clase permite crear botones con un diseño específico para esta aplicación.

Caso de Estudio Máquina Expendedora

Implemente una GUI para una máquina expendedora modelo M111 que brinde botones para preparar Café, Café con Leche y Submarino. Los tres botones se incluyen en un panel. Inmediatamente después de que se preparó una infusión, se habilita un botón para retirarla, los demás botones quedan deshabilitados. La información se muestra a través de una etiqueta en un panel. En el último panel el botón Empleado permite recargar la máquina y mostrar un panel de diálogo indicando las cantidades cargadas. La siguiente figura muestra la GUI inicial y luego de oprimir el botón Café:



Los paquetes requeridos para la implementación son:

```
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import javax.swing.BoxLayout;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Color;
import javax.swing.BorderFactory;
import javax.swing.border.BevelBorder;
import javax.swing.border.LineBorder;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.event.*;
import javax.swing.SwingConstants;
```

```
import javax.swing.WindowConstants;
```

Los atributos de instancia de la clase GUI_M111 son:

```
//Atributo de la Aplicacion
    private M111  me;
//Atributos Gráficos
    private Container pc;
    private JPanel panelBotones, panelES, panelRetirar,
panelEmpleado;
    private JLabel cartelBebida;
    private BotonME botonCafe, botonCafeConLeche,botonSubmarino;
    private JButton botonRetirarInfusion;
    private JButton botonEmpleado;
```

La clase `BotonMe` extiende a `JButton` estableciendo la misma apariencia para todos los botones de la GUI, excepto el rótulo del botón que varía en cada uno se especifica a través de un parámetro:

```
import java.awt.Dimension;
import java.awt.Font;
import javax.swing.*;
class BotonME extends JButton {
    public BotonME(String rotulo){
        setPreferredSize(new Dimension(306, 55));
        setFont(new Font("Arial",1,22));
        setText(rotulo);}
}
```

El constructor de la clase `GUI_M111` puede invocar a métodos internos que resuelvan cada uno de los subproblemas que plantea la implementación:

```
public GUI_M111(M111 m)  {
      super("Maquina "+m.obtenerNroSerie());
      pc = getContentPane();

      me = m;
      armarPaneles();
       armarBotones();
       armarEtiquetas();
       insertarComponentes();}
```

El código de cada método interno es:

```
private void armarPaneles() {
//Captura el panel de contenido y establece su layout
  BoxLayout layoutPc =new BoxLayout(pc,BoxLayout.Y_AXIS);
  pc.setLayout(layoutPc);
//Crea los paneles y establece su layout y apariencia
  panelBotones = new JPanel();
  GridLayout panelBotonesLayout = new GridLayout(3, 1);
  panelBotonesLayout.setHgap(5);
  panelBotonesLayout.setVgap(5);
  panelBotones.setLayout(panelBotonesLayout);
  panelBotones.setPreferredSize(new Dimension(392, 320));
  panelBotones.setSize(369, 250);
  panelBotones.setBackground(new Color(235,235,235));
  panelES = new JPanel();
```

```
  panelES.setPreferredSize(new Dimension(392, 101));
  panelES.setSize(369, 51);
  panelES.setBorder(BorderFactory.createBevelBorder
                    (BevelBorder.LOWERED));
  panelRetirar = new JPanel();
  panelRetirar.setPreferredSize(new Dimension(392, 50));
  panelEmpleado = new JPanel();
  panelEmpleado.setPreferredSize(new Dimension(392, 101));
  panelEmpleado.setSize(369, 80);
  panelEmpleado.setBackground(new Color(235,235,235));
  panelEmpleado.setBorder(BorderFactory.createBevelBorder
                          (BevelBorder.LOWERED));
}
```

Cada objeto de clase `BotonMe` es instancia de la clase `JButton` y por lo tanto puede recibir cualquier mensaje que corresponda a un método provisto por esta clase y sus clases ancestro. En el método interno armarBotones el tamaño de `botonEmpleado` de clase `BotonMe` se establece usando los métodos provistos por las clases ancestro.

```
private void armarBotones() {
//Crea el boton de cafe, crea el oyente y lo registra
      botonCafe = new BotonME("Café");
      OyenteCafe oCafe = new OyenteCafe();
      botonCafe.addActionListener(oCafe);
//Crea el boton de cafe con Leche, crea el oyente y lo registra
      botonCafeConLeche = new BotonME("Café con Leche");
      OyenteCafeConLeche oCafeConLeche = new OyenteCafeConLeche();
      botonCafeConLeche.addActionListener(oCafeConLeche);
//Crea el boton de submarino,  crea el oyente y lo registra
      botonSubmarino = new BotonME("Submarino");
      OyenteSubmarino oSubmarino = new OyenteSubmarino();
      botonSubmarino.addActionListener(oSubmarino);
//Crea el boton del Empleado,  crea el oyente y lo registra
      botonEmpleado = new BotonME("Empleado");
      botonEmpleado.setBackground(Color.BLUE);
     botonEmpleado.setPreferredSize(new Dimension(200, 76));
      botonEmpleado.setSize(180, 36);
      OyenteEmpleado oEmpleado = new OyenteEmpleado();
      botonEmpleado.addActionListener(oEmpleado);
//Crea el boton del retirar,  crea el oyente y lo registra
      botonRetirarInfusion = new JButton();
      botonRetirarInfusion.setText("Retire su infusión");
      botonRetirarInfusion.setEnabled(false);
      botonRetirarInfusion.setFont(new Font("SansSerif",1,14));
      botonRetirarInfusion.setBorder(BorderFactory.createEtchedBorde
r
                           (BevelBorder.LOWERED));
      botonRetirarInfusion.setPreferredSize(new Dimension(360, 32));
      OyenteRetirar oRetirar = new OyenteRetirar();
      botonRetirarInfusion.addActionListener(oRetirar);
}
```

Cada etiqueta tiene una apariencia estándar pero también puede configurarse de acuerdo a las pautas de diseño que se adopten.

```
private void armarEtiquetas() {
//Crea las etiquetas y establece sus atributos
```

```
      cartelBebida = new JLabel();
      cartelBebida.setLayout(new FlowLayout());
      cartelBebida.setBorder(new LineBorder
                             (new Color(0,0,0), 1, false));
      cartelBebida.setPreferredSize(new Dimension(277, 45));
      cartelBebida.setHorizontalAlignment(SwingConstants.CENTER);
      cartelBebida.setHorizontalTextPosition(SwingConstants.LEFT);
      cartelBebida.setFont(new Font("Arial",0,14));
}
private void insertarComponentes() {
//Inserta las componentes en los paneles
    panelBotones.add(botonCafe);
    panelBotones.add(botonCafeConLeche);
    panelBotones.add(botonSubmarino);
      pc.add(panelBotones);
      panelES.add(cartelBebida);
      pc.add(panelES);
      panelRetirar.add(botonRetirarInfusion);
      pc.add(panelRetirar);
      panelEmpleado.add(botonEmpleado);
      pc.add(panelEmpleado);
}
```

Los botones se habilitan y deshabilitan según se invoquen los siguientes métodos internos:

```
private void deshabilitarBotones(){
    botonCafe.setEnabled(false);
    botonCafeConLeche.setEnabled(false);
    botonSubmarino.setEnabled(false);
    botonRetirarInfusion.setEnabled(true);
    botonEmpleado.setEnabled(false);
}
private void habilitarBotones(){
    botonCafe.setEnabled(true);
    botonCafeConLeche.setEnabled(true);
    botonSubmarino.setEnabled(true);
    botonRetirarInfusion.setEnabled(false);
    botonEmpleado.setEnabled(true);
}
```

El código de las clases de los oyentes es:

```
class OyenteCafe implements ActionListener{
   public void actionPerformed(ActionEvent evt) {
//prepara el cafe si la cantidad de ingredientes lo permite
         int cantVasos = me.vasosCafe();
         if (cantVasos>=1){
           me.cafe();
            cartelBebida.setText("Aquí tiene su café");
            }
         else
           cartelBebida.setText("No puede preparar café");
        deshabilitarBotones();
        }
}
class OyenteCafeConLeche implements ActionListener {
      public void actionPerformed(ActionEvent evt) {
```

```
//prepara el cafe con leche si la cantidad de ingredientes lo
permite
          int cantVasos = me.vasosCafeConLeche();
          if (cantVasos>=1){
            me.cafeConLeche();
             cartelBebida.setText("Aquí tiene su café con leche");
            }
          else
            cartelBebida.setText("No puede preparar café con leche");
        deshabilitarBotones();
      }
}
class OyenteSubmarino implements ActionListener {
      public void actionPerformed(ActionEvent evt) {
          //Solo prepara el submarino si la cantidad de ingredientes
lo permite
          int cantVasos = me.vasosSubmarino();
          if (cantVasos>=1){
            me.submarino();
             cartelBebida.setText("Aquí tiene su submarino");
            }
          else
            cartelBebida.setText("No puede preparar submarino");
        deshabilitarBotones();}
    }
    class OyenteRetirar implements ActionListener {
       public void actionPerformed(ActionEvent evt) {
          cartelBebida.setText("");;
          habilitarBotones();}
}
class OyenteEmpleado implements ActionListener{
   public void actionPerformed(ActionEvent evt) {

      JOptionPane dialogo = new JOptionPane();
      int c = me.obtenerMaxCafe()-
me.cargarCafe(me.obtenerMaxCafe());
      int a = me.obtenerMaxCacao()-
              me.cargarCacao(me.obtenerMaxCacao());
      int l = me.obtenerMaxLeche()-
              me.cargarLeche(me.obtenerMaxLeche());
      dialogo.showMessageDialog(null,"Se cargo "+c+" grs cafe "
                  +a+" grs cacao "+l+" grs leche ",
                  "Maquina
cargada",JOptionPane.INFORMATION_MESSAGE);
          }
      }
}
```

La clase `GUI_M111` está asociada a la clase `M111` que a su vez hereda de `MaquinaExpendedora`. Las clases oyente usan los servicios provistos para los objetos de clase `M111` sin saber si estos servicios están provistos por esta clase o sus ancestros. La implementación de la GUI puede modificarse sin afectar a sus clases proveedoras y viceversa, en tanto no cambie el contrato establecido.

Problemas Propuestos

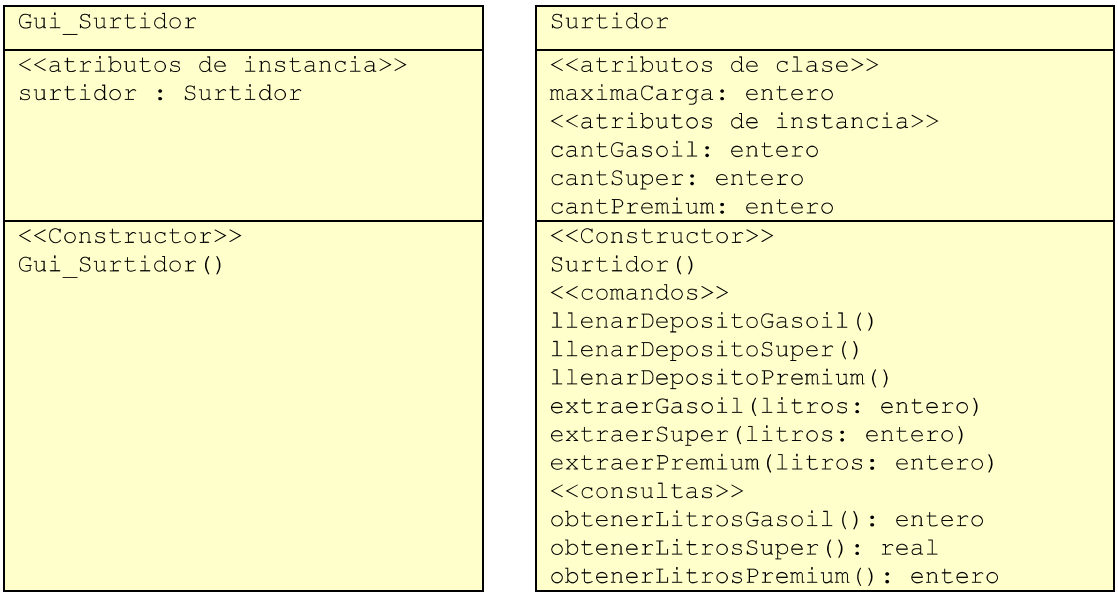
Caso de Estudio: Surtidor Combustible

Implemente la interfaz gráfica de un surtidor de combustible como muestra la figura:

Inicialmente solo están habilitados los botones Comprar y Recargar.

- El panel derecho muestra siempre el estado actual del surtidor. Los tipos de combustible se muestran a través de etiquetas y los litros se muestran usando cuadros de texto no editables.
- Si el usuario elige Comprar se activa la selección de tipo de combustible y se desactivan Comprar y Recargar.
- Al presionar el botón Recargar se recargan todos los depósitos de combustible al máximo posible.
- Con los `JRadioButton` se elige con qué tipo de combustible se desea cargar.
- En el cuadro de texto <valor> se ingresa cuántos litros de desea comprar. Luego de ingresarse el valor se modifica el estado interno del surtidor, se habilitan los botones Comprar y Recargar y se deshabilita el resto.

El siguiente diagrama modela las clases cliente y proveedor.



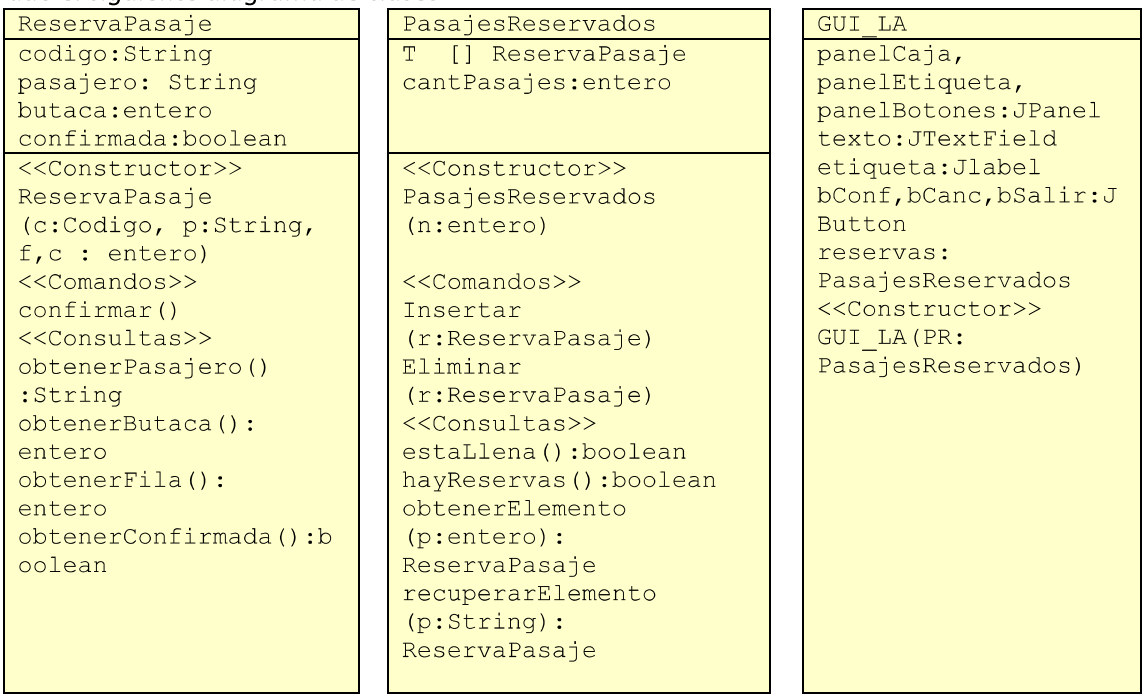
Caso de Estudio: Estación de Servicio

Una estación de servicio dispone de n surtidores representados en una clase Playa que encapsula un arreglo con n elementos de clase Surtidor.

Implemente una interface GUI_Playa con dos paneles, el panel superior muestra un objeto de clase JComboBox con los números de surtidor (1 a 6). El segundo panel tiene el mismo diagramado que en el ejercicio anterior. Inicialmente las componentes del segundo panel no están visibles, recién cuando el usuario selecciona el surtidor, se recupera el surtidor de la playa, se hace visible el segundo panel con los botones Comprar y Recargar activos. Cuando se completa la operación, se vuelve a habilitar la selección de surtidor y los componentes gráficos del segundo panel dejan de estar visibles.

Caso de Estudio: Confirmación Reserva

Dado el siguiente diagrama de clases



Implemente las dos clases y una GUI que incluya tres paneles. En el primero aparece una caja de texto, en el segundo una etiqueta y en el tercero tres botones (Confirmar, Anular y Salir) inicialmente deshabilitados.

Cuando el usuario completa el cuadro de texto con el código de una reserva y oprime enter, el oyente de la caja busca una reserva con ese código y muestra el nombre del pasajero, la fila y la butaca en la etiqueta del segundo panel. Si la reserva no está confirmada habilita los botones del tercer panel. Si la reserva está confirmada o no existe se muestran mensajes adecuados.

Si el usuario oprime el botón Confirmar, se envía el mensaje `confirmar()` *a la reserva. Si oprime Anular, se elimina la reserva. Si oprime Salir, se vuelve al estado inicial.*