

## Práctica 4. Patrones Estructurales

### Objetivos y Método de trabajo

Con esta práctica se pretende que el alumno, matriculado en la asignatura de Diseño y Programación Orientada a Objetos, ponga en práctica los conocimientos adquiridos en las clases teóricas correspondientes a la parte dedicada a patrones estructurales. Con este fin, al alumno se le proporciona un supuesto práctico que deberá resolver utilizando los patrones presentado en clase.

- A desarrollar en 2 turnos

### 5.1 La interfaz 2D de JAVA

#### *Introducción al API 2D de Java*

El API 2D de Java introducido en el JDK 1.2 proporciona gráficos avanzados en dos dimensiones, texto, y capacidades de manejo de imágenes para los programas Java a través de la extensión del AWT. Este paquete de rendering soporta líneas artísticas, texto e imágenes en un marco de trabajo flexible y lleno de potencia para desarrollar interfaces de usuario, programas de dibujo sofisticados y editores de imágenes.

El API 2D de Java proporciona:

- Un modelo de rendering uniforme para pantallas e impresoras.
- Un amplio conjunto de primitivos geométricas, como curvas, rectángulos, y elipses y un mecanismo para renderizar virtualmente cualquier forma geométrica.
- Mecanismos para detectar esquinas de formas, texto e imágenes.
- Un modelo de composición que proporciona control sobre cómo se renderizan los objetos solapados.
- Soporte de color mejorado que facilita su manejo.
- Soporte para imprimir documentos complejos.

#### *Rendering en Java 2D*

El mecanismo de rendering básico es el mismo que en las versiones anteriores del JDK -- el sistema de dibujo controla cuándo y cómo dibuja un programa. Cuando un componente necesita ser mostrado, se llama automáticamente a su método **paint** o **update** dentro del contexto **Graphics** apropiado.

El API 2D de Java presenta [java.awt.Graphics2D](#), un nuevo tipo de objeto **Graphics**. **Graphics2D** desciende de la clase [Graphics](#) para proporcionar acceso a las características avanzadas de rendering del API 2D de Java.

Para usar las características del API 2D de Java, tenemos que forzar el objeto **Graphics** pasado al método de dibujo de un componente a un objeto **Graphics2D**.

```
public void paint (Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    ...
}
```

Nota: Cuando necesitemos modificar el aspecto que ofrece un componente ya sea AWT o Swing tan sólo deberemos implementar el método **Paint**.

### **Punteado de Gráficos Primitivos**

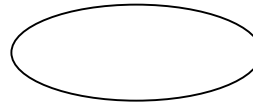
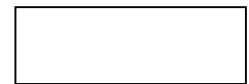
Cada una de las formas dibujadas por el JApplet está construidas de geometrías y está dibujada a través de **Graphics2D**. Las variables **rectHeight** y **rectWidth** de este ejemplo definen las dimensiones del espacio en que se dibuja cada forma, en pixels. La variables **x** e **y** cambian para cada forma para que sean dibujadas en formación de parrilla.

```
// draw Line2D.Double
g2.draw(new Line2D.Double(x, y+rectHeight-1,
                          x + rectWidth, y));

// draw Rectangle2D.Double
g2.draw(new Rectangle2D.Double(x, y, rectWidth, rectHeight));

// draw Ellipse2D.Double
g2.draw(new Ellipse2D.Double(x, y,
                              rectWidth,
                              rectHeight));

// draw String
g2.drawString("DPOO", 100, 100);
```

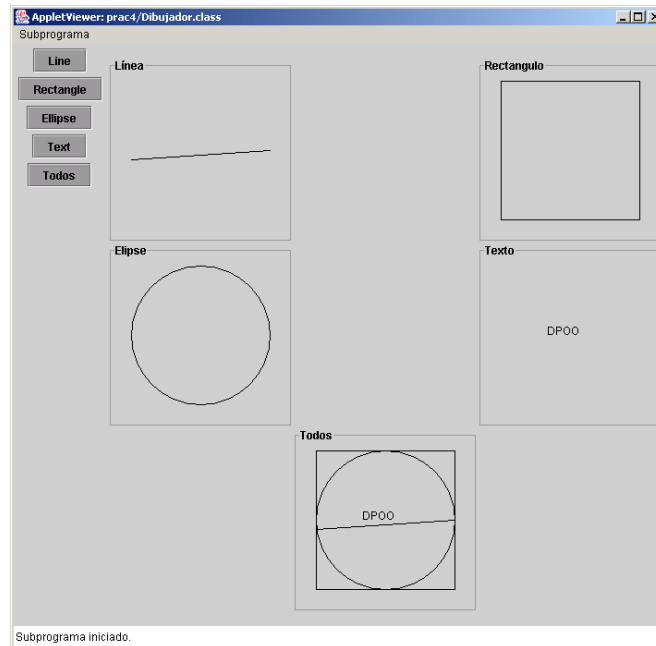


DPOO

### **Ejemplo**

Se pretende realizar una aplicación con un JApplet o una JFrame cuya función sea detectar el pulsado del ratón sobre unos determinados botones de selección (JToggleButton), los cuales permitirán dibujar formas simples (líneas, rectángulos, círculos, texto, etc). Cada botón mostrará cada uno de los gráficos que se pueden crear (se podrán tener pulsados más de un botón al mismo tiempo). Tendremos un último panel donde aparecerán todas las figuras que se muestran en los otros paneles.

La apariencia del cliente sería la siguiente:



**Solución:**

Se debe sobrescribir/implementar el método paint de la ventana principal y sustituirlo por este código:

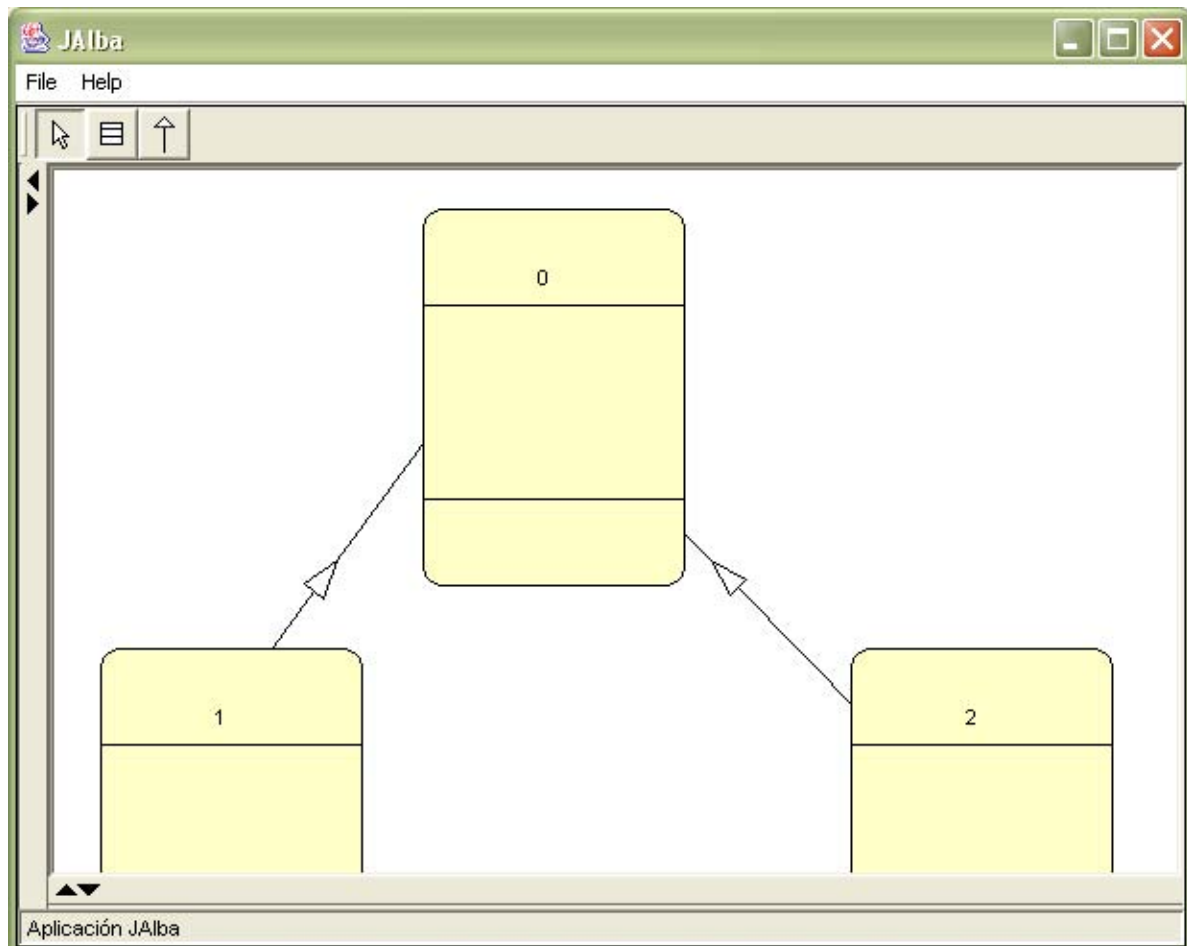
```
public void paint (Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    if(this.jToggleButton1.isSelected() || todos){
        // draw Line2D.Double
        g2.draw(new Line2D.Double(x, y, x+lineHeight, y+lineWidth));
    }
    if(this.jToggleButton2.isSelected() || todos){
        // draw Rectangle2D.Double
        g2.draw(new Rectangle2D.Double(x, y, rectWidth, rectHeight));
    }
    if(this.jToggleButton3.isSelected() || todos){
        // draw Ellipse2D.Double
        g2.draw(new Ellipse2D.Double(x, y, ellipseWidth, ellipseHeight));
    }
    if(this.jToggleButton4.isSelected() || todos){
        // draw String
        g2.drawString("DPOO", x, y);
    }
    if(this.jToggleButton5.isSelected())
        todos = true;
    else todos = false;
}
```

Nota: Tanto los valores x e y como los width y Height se deben sustituir por los lugares donde se deben pintar. Para su realización se debe tomar como referencia los paneles usados para mostrarlos de tal forma que aunque aumente o disminuya la ventana principal esto no suponga que su aspecto varíe.

## Desarrollo

Desarrollar un entorno gráfico para la construcción de diagramas de clases. En este entorno se deben proporcionar facilidades para la representación de clases y de relaciones entre ellas. Las relaciones representadas estarán restringidas a relaciones de generalización o herencia únicamente. La aplicación debe gestionar los diagramas introducidos permitiendo la definición de clases y de relaciones entre ellas.

La restricción mas importante para la realización es el uso apropiado de alguno de los patrones estructurales vistos en teoría y la justificación de su uso. Se aconseja el uso de Poseidón para generar el diagrama de clases, así como para generar el esqueleto java con el que comenzar a implementar la práctica.



## Anexo

En los anexos podréis encontrar alguna de las clases y métodos que podéis utilizar para el desarrollo de la práctica.

### **JComponent**

La clase **JComponent** es la clase base o SuperClase de prácticamente todos los componentes Swing excepto de alguno de los contenedores más básicos. Para utilizar esta clase debemos crear una clase que herede de JComponent y una vez instanciada debemos añadirla a un contenedor que esté asociado a un componente del mas alto nivel como puede ser un JFrame, JDialog o JApplet. Ya que estos contenedores poseen mecanismos que les permiten dibujarse a si mismos y a todos los componentes que contienen.

Así por ejemplo si deseamos añadir un componente (componente1) a un panel llamado jPanel1 tan solo deberemos llamar a su método add de la siguiente forma: jPanel1.add(componente1); (si deseamos quitarlo para no volverlo a visualizar llamaremos al método jPanel1.remove(componente1)).

Una vez añadido al panel si queremos acceder a él desde el componente podremos hacerlo a través del método getParent() el cual nos devolverá un objeto del tipo Container que referenciará al panel.

Como ya hemos comentado en el caso de que queramos que nuestro componente personalizado se dibuje de forma especial deberá ser sobrescrito el método paint. Aunque deberemos tener en cuenta que los límites del componente tienen que estar fijados. Estos límites nos van a permitir delimitar el área de pintado del componente para ello debemos hacer uso del método setBounds(int x, int y, int w, int h). Donde x e y nos indicarán la posición x e y respecto del contenedor padre y, w y h nos indicarán la anchura y altura respectivamente.

Para terminar la explicación de la clase JComponent comentaremos como hacer nuestro componente susceptible a los eventos del ratón. Para ello lo primero que debemos hacer es que nuestro componente implemente las interfaces MouseListener y MouseMotionListener. Una vez hecho esto tan solo es necesario en el constructor añadir this.addMouseListener(this); this.addMouseMotionListener(this); lo que nos permitirá dotar a nuestro componente de sensibilidad al ratón. También deberemos sobrescribir o implementar los métodos de las interfaces. Los métodos necesarios serían los siguientes:

1. public void mouseClicked(MouseEvent e); \*Al hacer clic sobre el componente
2. public void mousePressed(MouseEvent e); \*Al pulsarlo sobre el componente
3. public void mouseReleased(MouseEvent e); \*Al soltarlo sobre el componente
4. public void mouseEntered(MouseEvent e); \*Al entrar en el componente
5. public void mouseExited(MouseEvent e); \*Al salir del componente
6. public void mouseDragged(MouseEvent e); \*Al arrastrar sobre el componente
7. public void mouseMoved(MouseEvent e); \*Al moverlo sobre el componente

## ***Pintar la herencia***

Con el siguiente método obtendremos un polígono en forma de herencia, el resultado se puede visualizar directamente en el método paint si lo dibujamos a través del método draw de un objeto de tipo Graphics o Graphics2D.

```
public GeneralPath arrows(int x1, int y1, int x2, int y2, double size) {
    GeneralPath polygon;
    int xf = x2;
    int yf = y2;
    x2 +=(x1-x2)/2;
    y2 +=(y1-y2)/2;
    Point2D p1 = new Point2D.Double(x1,y1);
    Point2D p2 = new Point2D.Double(x2,y2);
    double dx = x2-x1;
    double dy = y2-y1;
    double ra = java.lang.Math.sqrt(dx*dx + dy*dy);
    dx /= ra;
    dy /= ra;
    int x3 = (int)Math.round(x2-dx*size);
    int y3 = (int)Math.round(y2-dy*size);
    double r = 0.3*size;
    int x4 = (int)Math.round(x3+dy*r);
    int y4 = (int)Math.round(y3-dx*r);
    int x5 = (int)Math.round(x3-dy*r);
    int y5 = (int)Math.round(y3+dx*r);
    polygon = new GeneralPath(GeneralPath.WIND_EVEN_ODD,3);
    polygon.moveTo(x1, y1); polygon.lineTo(x3,y3);
    polygon.moveTo(x4,y4); polygon.lineTo(x2,y2);
    polygon.lineTo(x5,y5);polygon.lineTo(x4,y4);
    polygon.moveTo(x2,y2); polygon.lineTo(xf,yf);
    return polygon;
}
```