PRACTICA CLASE GRAPHICS

Duracion estimada: 200 minutos

Objetivos:

- Habituar al alumno a manejar la clase graphics

Recursos necesarios:

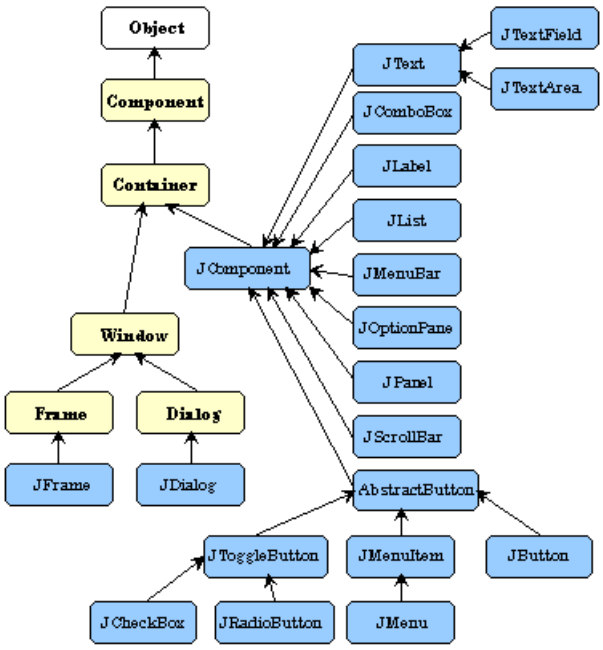
- Guión de la práctica
- Internet

Cada componente tiene un método llamado **paintComponent**, que se encarga de pintarlo en pantalla.

Para realizar un dibujo definido por el programador, basta con heredar de un componente (normalmente un JPanel), y sobreescribir su método paintComponent.

void paintComponent(Graphics g)

LA CLASE GRAPHICS



La clase Graphics es la clase base abstracta para todos los contextos gráficos que permiten que una aplicación se dibuje en componentes que se realizan en varios dispositivos, así como en imágenes fuera de pantalla.

Casi todas las componentes y contenedores de Swing tienen un método **paint(g)** asociado que sirve para dibujarlos en pantalla.

Cada componente tiene un método llamado **paintComponent**, que se encarga de pintarlo en pantalla.

Java invoca este método automáticamente cuando tiene que mostrar, de forma estándar, el componente o contenedor en cuestión (esto es, sus bordes, su título, si lo tiene, etc.)

El método **paint(g)** se redefine cuando se quiere que estos elementos tengan un aspecto particular, por ejemplo, cuando se quiere dibujar algo específico sobre ellos.

El método paint(g) es de la forma

```
public void paint(Graphics g) {
...
}
```

Donde g es un objeto de la clase abstracta Graphics. Todo contenedor o componente que se pueda dibujar en pantalla tiene asociado un objeto g de esta clase, con la información sobre el área de la pantalla que el contenedor o la componente cubre. Además, el objeto g dispone de métodos para hacer gráficos (dibujar círculos, rectángulos, líneas, etc.)

La clase Graphics es una clase abstracta para todos los contextos gráficos. Una vez obtenido el contexto gráfico podemos llamar desde este objeto a las funciones gráficas definidas en la clase Graphics.

Graphics contiene información acerca de la zona que necesita ser redibujada: el objeto donde dibujar, un origen de traslación, el color actual, la fuente actual, etcétera.

La clase Graphics se importa de awt:

import java.awt.*;

Cuando el método paint(g) se ejecuta es porque ha sido invocado por otros métodos, nunca invocado por nosotros, y el parámetro que usa corresponde a un objeto de la clase Graphics asociado al contenedor o componente que estemos manejando.

Se puede invocar por medio del método **repaint**

Cuando se redefine el método paint(g), siempre se comienza con una invocación super.paint(g) al método de la superclase, asegurando así que se dibuja la parte estándar del contenedor o componente que estemos manejando.

Proporciona métodos para dibujar, rellenar, pintar imágenes, copiar áreas y pegar gráficos en pantalla:

Métodos de la clase `Graphics` para pintar líneas, rectángulos y óvalos

Method	Description
`public void drawLine( int x1, int y1, int x2, int y2 )`	
	Draws a line between the point (x1, y1) and the point (x2, y2).
`public void drawRect( int x, int y, int width, int height )`	
	Draws a rectangle of the specified `width` and `height`. The top-left corner of the rectangle has the coordinates (x, y).
`public void fillRect( int x, int y, int width, int height )`	
	Draws a solid rectangle with the specified `width` and `height`. The top-left corner of the rectangle has the coordinate (x, y).
`public void clearRect( int x, int y, int width, int height )`	
	Draws a solid rectangle with the specified `width` and `height` in the current background color. The top-left corner of the rectangle has the coordinate (x, y).
`public void drawRoundRect( int x, int y, int width, int height, int arcWidth, int arcHeight )`	
	Draws a rectangle with rounded corners in the current color with the specified `width` and `height`. The `arcWidth` and `arcHeight` determine the rounding of the corners (see Fig. 12.15).
`public void fillRoundRect( int x, int y, int width, int height, int arcWidth, int arcHeight )`	
	Draws a solid rectangle with rounded corners in the current color with the specified `width` and `height`. The `arcWidth` and `arcHeight` determine the rounding of the corners (see Fig. 12.15).

Method Description	
`public void draw3DRect( int x, int y, int width, int height, boolean b )`	
	Draws a three-dimensional rectangle in the current color with the specified `width` and `height`. The top-left corner of the rectangle has the coordinates (`x`, `y`). The rectangle appears raised when `b` is true and lowered when `b` is false.
`public void fill3DRect( int x, int y, int width, int height, boolean b )`	
	Draws a filled three-dimensional rectangle in the current color with the specified `width` and `height`. The top-left corner of the rectangle has the coordinates (`x`, `y`). The rectangle appears raised when `b` is true and lowered when `b` is false.
`public void drawOval( int x, int y, int width, int height )`	
	Draws an oval in the current color with the specified `width` and `height`. The bounding rectangle's top-left corner is at the coordinates (`x`, `y`). The oval touches all four sides of the bounding rectangle at the center of each side (see Fig. 12.16).
`public void fillOval( int x, int y, int width, int height )`	
	Draws a filled oval in the current color with the specified `width` and `height`. The bounding rectangle's top-left corner is at the coordinates (`x`, `y`). The oval touches all four sides of the bounding rectangle at the center of each side (see Fig. 12.16).

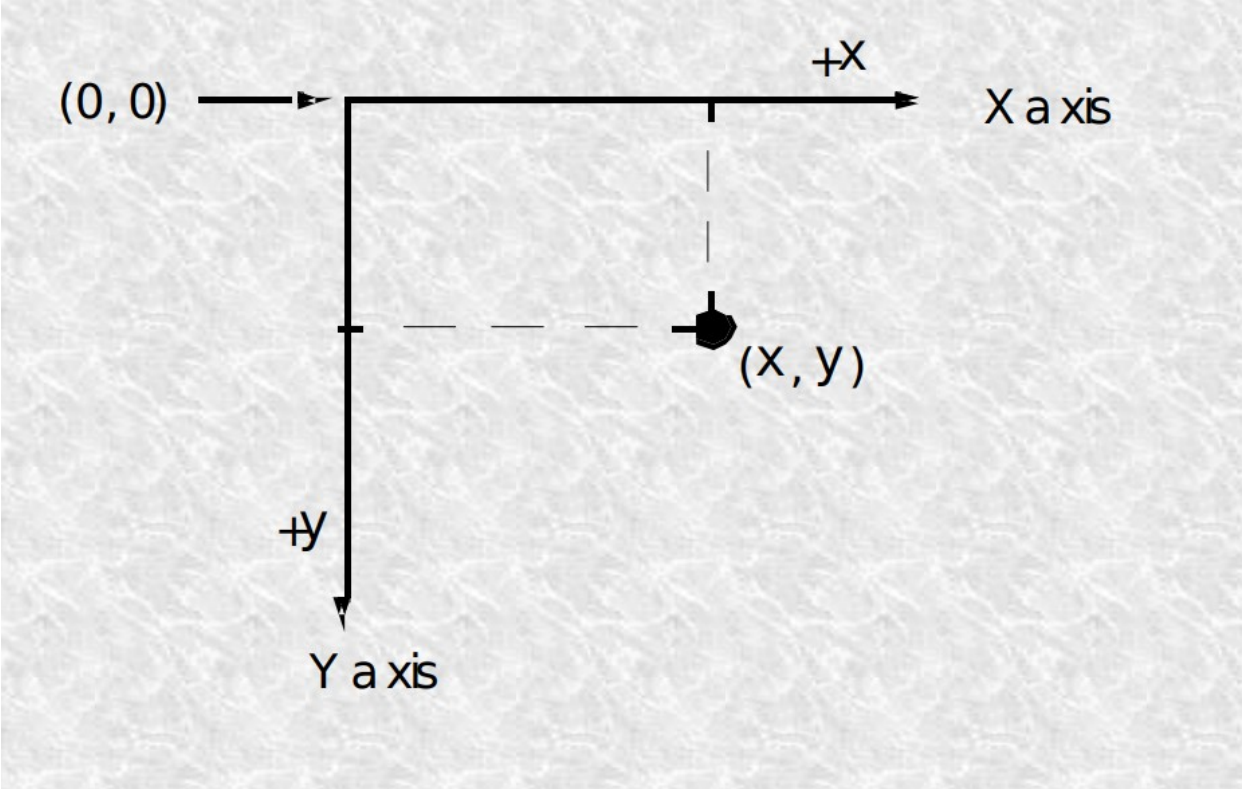
EJEMPLO1:

Por ejemplo, vamos a dibujar una cara en un marco. Un marco es un elemento de la clase JFrame y para dibujar en él procedemos así:

```
public void paint (Graphics g){
        super.paint(g);
         //Dibujo el contorno de la cara
        g.setColor(Color.BLACK);
        g.fillOval(105, 70, 100, 100);
        //Dibujo de los ojos
        g.setColor(Color.GREEN);
        g.fillOval(125, 100, 10, 10);
        g.fillOval(175, 100, 10, 10);
        //Dibujo de la nariz
        g.drawLine(150, 100, 150, 130);

        //Dibujo de la boca
        g.drawArc(118, 120, 75, 30, 180, 180);
}
```

Para entender lo que ha hecho el método anterior hay que saber que:

- El sistema de coordenadas de un contenedor tiene su origen en su esquina superior izquierda.
- Las abscisas se incrementan hacia la derecha y las ordenadas hacia abajo.



- Cada punto es un píxel.
- En general, el dibujo de una figura (rectángulo, elipse, rectángulo redondo, etc.) se realiza dando las coordenadas de la esquina superior izquierda de un rectángulo imaginario que la contiene.

Algunos métodos de la clase Graphics para dibujar figuras son:

- **drawLine(x1,y1,x2,y2):** dibuja una línea recta desde el punto (x1,y1) al punto (x2,y2)
- **fillRect(x,y,ancho,alto):** rellena el rectángulo que tiene su esquina superior izquierda en (x,y) y el ancho y largo dados
- **drawOval(x,y,ancho,alto):** dibuja una elipse contenida en un rectángulo imaginario cuya esquina superior izquierda está en (x,y) y tiene el ancho y largo dados
- **fillOval(x,y,ancho,alto)**: rellena la elipse especificada pordrawOval(x,y,ancho,largo)
- **drawArc(x,y,ancho,alto, inicioAngulo,barridoAngulo):** dibuja parte de una elipse dentro de un rectángulo imaginario cuya esquina superior izquierda está en (x,y), tiene el largo y ancho dados, empieza a dibujar en el ángulo inicioAngulo y hace un barrido de barridoAngulo
- **setColor(Color.red):** cambia la “tinta” del objeto g y la pone de color rojo. La clase Color se importa de awt:

```
import java.awt.*;
```

JPanel se puede usar para dibujar. Para dibujar en un panel se crea una clase derivada de JPanel y se redefine el método **paintComponent()** que le indica al panel como dibujar.

Cuando utilizamos el método **paintComponent()** para dibujar en un contexto gráfico g, ese contexto es un ejemplar de una subclase concreta de la clase Graphics para la plataforma específica.

El método **paintComponent()** es llamado la primera vez y cada vez que es necesario redibujar el componente. Al hacer **super.paintComponent(g)** nos aseguramos de que el área visualizada se limpia antes de volver a dibujar.

EJEMPLO2:

```
class MiPanel extends JPanel {
        public void paintComponent(Graphics g) {
                super.paintComponent(g);
                g.drawString("Interfaz gráfica", 40, 40); }
}
```

1. Dibujar sobre paneles:

- Cuando se actualiza un componente, se borra su aspecto actual y se invoca **paint(g)**. El borrado previo puede producir parpadeo (flickering) ,por lo que a veces el método **paint(g)** evita hacerlo.
- Sin embargo, la actualización puede necesitar hacer el borrado previo(para actualizar el fondo del componente, por ejemplo). **En estos casos se invoca el método paintComponent(g) de la clase JComponent, que permite hacer un barrido previo, pero usando la técnica del doblebuffer para eliminar el parpadeo**.
- En casos como los anteriores **lo que se hace es redefinir el método paintComponent(g) en lugar del método paint(g)**.

Por ejemplo, para dibujar la cara que pintamos antes, pero sobre un Jpanel en lugar de sobre un marco, hacemos lo siguiente:

- Se declara la clase PanelCara que extiende la clase Jpanel, como clase (privada) de la clase MarcoCara que habíamos creado antes
- Se redefine el método paintComponent(g) usando las mismas instrucciones que antes, pero llamando inicialmente a super.paintComponent(g)

```
public void paintComponent(Graphics g) {
        super.paintComponent(g);…}
```

Cuando el contenido de un marco o panel cambia, el método **repaint()** se encarga de actualizar el contenedor y mostrarlo por pantalla, invocando implícitamente el método **paint(g)** o **paintComponent(g)**.

Por ejemplo, para añadir un botón al marco que mostraba una cara sonriente de forma que, cuando se pulse, la cara cambie a una cara eenojada, basta hacer lo siguiente:

- Se añade a contentPane el botón cuyo efecto cambiará la sonrisa de la cara
- Se añade un atributo booleano al marco que indica si la cara sonríe o no

```
private boolean sonrie=true;
```

- Dentro del actionPerformed del boton codeamos lo siguiente:

```
sonrie=!sonrie;
repaint();
```

- El método paintComponent(g) para pintar el panel se redefine por el siguiente:

```
public void paintComponent(Graphics g) {
        super.paintComponent(g);
        //Dibujo de la cara igual que antes, salvo la boca
        ...
        //Dibujo de la boca
        if (sonrie)
                g.drawArc(118, 125, 75, 30, 180, 180);
        else
                g.drawArc(118, 125, 75, 30, 180, -180);
        }
```

2. Dibujar texto:

La clase Graphics permite “dibujar” texto, como alternativa al texto mostrado en los componentes JLabel, JTextField y JTextArea. El método que permite graficar texto sobre el JFrame es:

```
drawString(String str, int x, int y);
```

3. Clase Color:

La clase java.awt.Color encapsula colores utilizando el formato RGB (Red, Green, Blue). Las componentes de cada color primario en el color resultante se expresan con números enteros entre 0 y 255, siendo 0 la intensidad mínima de ese color y 255 la máxima. En la clase Color existen constantes para colores predeterminados de uso frecuente: black, white, green, blue, red, yellow, magenta, cyan, orange, pink, gray, darkGray, lightGray.

4. Presentación de imágenes:

Java permite incorporar imágenes de tipo GIF y JPEG definidas en ficheros. Se dispone para ello de la clase java.awt.Image. Para cargar una imagen hay que indicar la localización del archivo y cargarlo mediante el método **getImage()**. Este método existe en las clases **java.awt.Toolki**t.

Entonces, para cargar una imagen hay que comenzar creando un objeto (o una referencia) **Image** y llamar al método **getImage()** (de Toolkit); Una vez cargada la imagen, hay que representarla, para lo cual se redefine el método **paint()** para llamar al método

drawImage() de la clase Graphics. Los objetos Graphics pueden mostrar imágenes a través del método: **drawImage()**. Dicho método admite varias formas, aunque casi siempre hay que incluir el nombre del objeto imagen creado.

5. Clase Image:

Una imagen es un objeto gráfico rectangular compuesto por pixels coloreados. Cada pixel en una imagen describe un color de una particular localización de la imagen.

A continuación, algunos métodos de la clase Image:

La clase Graphics provee el método drawImage() para dibujar imagenes; este método admite varias formas:

- ***drawImage (Image i, int x, int y, ImageObserver o)***
- ***drawImage (Image i,int x,int y,int width,int height,ImageObserver o)***

Un ejemplo del metodo paint(Graphics G) para traer una imagen de la carpeta raiz es el siguiente:

```
public void paint (Graphics g)
{
        super.paint(g);
        Toolkit t = Toolkit.getDefaultToolkit ();
        Image imagen = t.getImage ("imagen1.jpg");
        g.drawImage (imagen, 0, 0, this);
}
```

EJEMPLO3:Usar la clase Graphics en un canvas

```
public class EjemploCanvas extends Canvas { // se puede añadir a un frame para visualizarlo
        String cad = "Escrito en canvas";
        // este metodo se ejecuta automaticamente cuando Java necesita mostrar la ventana

        public void paint(Graphics g) { // obtener el color original
                Color colorOriginal = g.getColor();
                // escribir texto grafico en la ventana y recuadrarlo
                g.drawString(cad, 40, 20);
                g.drawRect(35, 8, (cad.length()*7), 14);
                // dibujo de algunas lineas
                for (int i=20; i< 50; i= i+3)
                        { if ((i % 2) == 0)
                                g.setColor(Color.blue);
                        else
                                g.setColor(Color.red);
                                g.drawLine(40, (90-i), 120, 25+i); }
                // dibujo y relleno de un óvalo
                g.drawOval(40, 95, 120, 20);
                g.fillOval(40, 95, 120, 20);
                g.setColor(colorOriginal); }
}
```

6. La clase FontMetrics:

La clase *FontMetrics* nos permite conocer las características de una fuente de texto.

Desde el contexto gráfico *g*, llamamos a la función ***getFontMetrics*** para obtener un objeto ***fm*** de la clase ***FontMetrics*** que nos describe las características de una fuente determinada o de la fuente actualmente seleccionada. En el primer caso escribimos:

```
Font fuente=new Font("Dialog", Font.BOLD, 36);
FontMetrics fm=g.getFontMetrics(fuente);
```

En el segundo caso, escribimos

```
Font fuente=new Font("Courier", Font.BOLD, 36);
g.setFont(fuente);
FontMetrics fm=g.getFontMetrics();
```

Para obtener la altura de una fuente de texto, llamamos a la función ***getHeight*** miembro de ***FontMetrics***. La altura de una fuente de texto, es la distancia entre dos líneas base (baseline) consecutivas, y es la suma de el *ascent*, *descent* y *leading*.

Tres funciones que comienzan por *get* devuelven los valores de estos tres atributos de una fuente de texto: ***getAscent***, ***getDescent***, y ***getLeading***.

Para escribir dos líneas de texto, una debajo de otra escribimos

```
FontMetrics fm=g.getFontMetrics();
String texto="La cigüeña vendrá";
g.drawString(texto, 10, 50);
int hFont=fm.getHeight();
texto=new String("serÁ en primavera");
g.drawString(texto, 10, 50+hFont);
```

La primera línea de texto, se sitúa en el punto (10, 50), la ordenada 50, señala la posición vertical de la línea base de la fuente de texto, véase la figura. La segunda línea de texto tiene una línea base cuya posición vertical se obtiene sumando a 50 la altura *hFont* de la fuente de texto.

Otro valor interesante, es la anchura de un texto, que se obtiene mediante la función miembro ***stringWidth***, y se le pasa el texto. Por ejemplo, para centrar horizontalmente un texto en el applet escribimos.

```
String texto="La cigüeña vendrá";
int ancho=fm.stringWidth(texto);
g.drawString(texto, (anchoApplet-ancho)/2, 50);
```

La función ***getSize().width*** obtiene la anchura del componente (applet), y la variable *ancho*, guarda la anchura del string *texto*.

Un poco más difícil es centrar un texto verticalmente, en una determinada posición. Teniendo en cuenta, que las coordendas que se le pasan a la función *drawString* se refieren a la línea base del primer carácter, tal como se ve en la figura. La fórmula de centrado vertical en un punto de ordenada *y* sería: la ordenada *y* de la línea base menos

descent más la mitad de la altura de los caracteres *hFont*. Se ha de tener en cuenta que la ordenada *y* aumenta de arriba hacia abajo.

```
g.drawLine(0, y, anchoApplet, y);
g.setColor(Color.red);
texto="Centrado: a, p, ñ, á, Á ";
g.drawString(texto, 10, y+hFont/2-descent);
```

Como los caracteres pueden estar o no acentuados, escritos en mayúsculas o minúsculas, etc, la fórmula para mostrar un texto centrado verticalmente no es única, se sugiere probar estas dos dadas por otros autores

```
g.drawString(texto, 10, y+ascent/4);
g.drawString(texto, 10, y-hFont/2+ascent);
```

EJEMPLO4:

```
import javax.swing.*;
import java.awt.*;

import static java.awt.Font.PLAIN;

public class Ejemplo19 extends JFrame {

   JPanel panel;

   public Ejemplo19(){

      initPanel();
      initPantalla();

   }

   void initPanel(){

      panel = new JPanel();
      add(panel); //Añado el panel al JFrame
      panel.setPreferredSize(new Dimension(800,600)); //Dimensiones del panel

   }

   @Override
   public void paint(Graphics g){
      super.paint(g);

      //Para poder modificar más propiedades con Graphics 2d
      Graphics2D g2d = (Graphics2D) g;

      //Línea
      g2d.setColor(Color.BLUE);
      g2d.setStroke(new BasicStroke(5));
      g2d.drawLine(30, 70, 770, 70);

      //Rectángulo (relleno y borde)
      g2d.setColor(Color.BLUE);
      g2d.fillRect(30, 100, 350, 60);
      g2d.setColor(Color.BLACK);
      g2d.drawRect(30, 100, 350, 60);
```

```
        //Rectángulo redondeado
        g2d.setColor(Color.CYAN);
        g2d.drawRoundRect (420, 100, 350, 60, 50, 50);

        //Arco
        g2d.setColor(Color.PINK);
        g2d.drawArc(30, 200, 100, 100, 180, -90);

        //Círculo
        g2d.setColor(Color.RED);
        g2d.drawOval(100, 200, 100, 100);

        //Óvalo (con relleno y borde)
        g2d.setColor(Color.YELLOW);
        g2d.fillOval(240, 200, 150, 100);
        g2d.setColor(Color.BLACK);
        g2d.drawOval(240, 200, 150, 100);

        //Polígono (3 lados)
        int [] triangulo_x = {450, 510, 570};
        int [] triangulo_y = {300, 200, 300};
        g2d.setColor(Color.ORANGE);
        g2d.drawPolygon (triangulo_x, triangulo_y, 3);

        //Polígono (5 lados con relleno y borde)
        int [] pentagono_x = {670, 650, 700, 750, 730};
        int [] pentagono_y = {300, 245, 200, 245, 300};
        g2d.setColor(Color.MAGENTA);
        g2d.fillPolygon (pentagono_x, pentagono_y, 5);
        g2d.setColor(Color.BLACK);
        g2d.drawPolygon (pentagono_x, pentagono_y, 5);

        //Texto
        g2d.setColor(Color.BLACK);
        g2d.setFont(new Font("ARIAL",PLAIN,32));
        g2d.drawString("Esto es un texto", 30, 400);

        //Imagen
        Toolkit t = Toolkit.getDefaultToolkit();
        Image imagen = t.getImage ("src/img/smile.png");
        g2d.drawImage(imagen, 30, 450, this);

        //Degradado
        GradientPaint gp = new GradientPaint(400, 350, Color.RED, 770, 550, Color.GREEN);
        g2d.setPaint(gp);
        g2d.fillRect(400, 350, 370, 200);
    }

    private void initPantalla(){

        setTitle("Ejemplo 19");
        setSize(800,600);
        setResizable(false);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
```

```
    }

    public static void main(String[] args) {
        new Ejemplo19();
    }
}
```

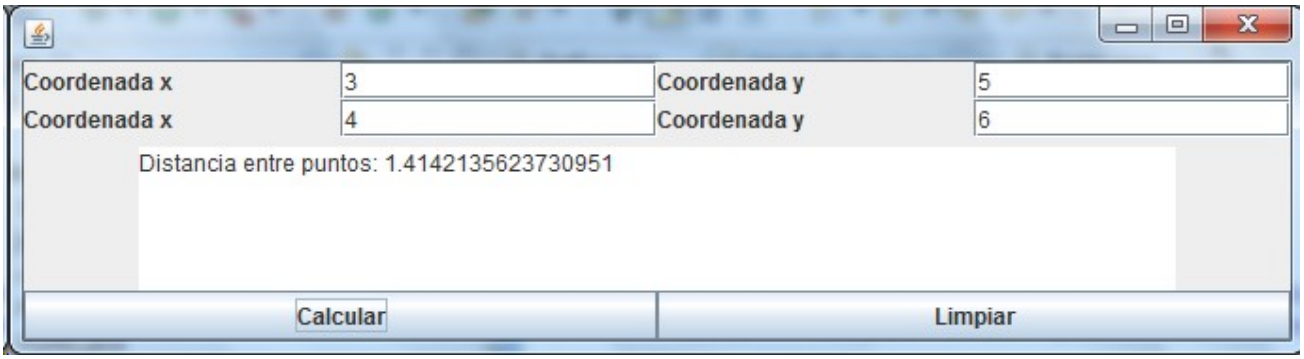
Secuencia/Desarrollo:

1. ***Ejercicio1:*** *Implementa la siguiente interface e implementa la funcionalidad utilizando para ello la clase* ***Punto****. Añade a la clase Punto el método* ***dibujar****, que dibuje un cuadrado creado entre los dos puntos.*

```
public class punto {
  //Atributos
            private double x;
            private double y;
 //Constructores
            public punto() {}

            public punto(double coordenadaX, double coordenadaY) {
              x = coordenadaX;
              y = coordenadaY;
            }
 //Resto de métodos

            /**
             * Devuelve la coordenada x
             */
            public double obtenerX() {
              return x;
            }
             /**
             * Devuelve la coordenada y
             */
            public double obtenerY() {
              return y;
            }

            public void asignarX(double x){
                    this.x=x;
            }

            public void asignarY(double y){
                    this.y=y;
            }

            /**
             * Devuelve la distancia entre 2 puntos: el punto que recibe el mensaje
             * y el punto que recibe como argumento
             */
            public double calcularDistancia(punto p) {
              double disX, disY;
              disX = p.obtenerX() - x;
              disY = p.obtenerY() - y;
```

```
                return (Math.sqrt(disX * disX + disY * disY));
            }
    }
```

Añadir un nuevo botón Dibujar y una lista desplegable que permita cambiar el color de relleno del cuadrado creado.



Requisitos:

- ***Clase Principal (método main)***
- ***Clase Ventana (hereda de JFrame) y construye la interfaz***
- ***Clase Punto***
- ***Clase PanelPunto (hereda de JPanel) y es donde se va a dibujar el cuadrado***
- ***Clase Manejador que implementa la funcionalidad de los botones***