UNIVERSIDAD CATÓLICA DE SANTA MARÍA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

SESIÓN N° 10:

Java Swing Avanzado

I

OBJETIVOS

- Conocer los conceptos generales de Bases de Datos Relaciones y de las Clases y Métodos de JDBC en JAVA para manipular los datos de la misma.
- Implementar ejemplo del uso de Clases y Métodos de JDBC en JAVA para manipular los datos de una Base de Datos
- Valorar la importancia de JDBC en Java para manipular una Base de Datos.

II

TEMAS A TRATAR

- Binding de Datos
- Gráficos
- Multimedia
- Resumen

III

MARCO TEORICO

1. BINDING DE DATOS

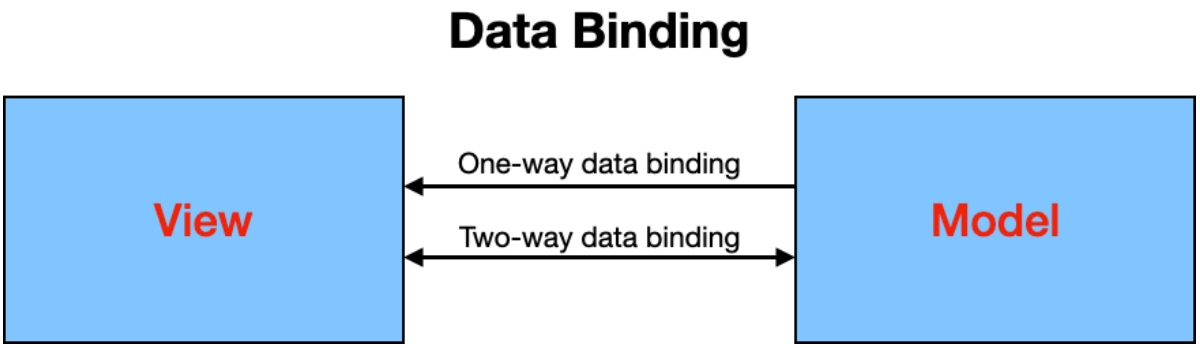
Binding de Datos (o **Data Binding**) es un concepto que se refiere a la sincronización de datos entre el modelo (lógica de negocio) y la vista (interfaz de usuario) de una aplicación. En el contexto de aplicaciones gráficas, especialmente en **Java Swing**, el binding de datos permite que los cambios en el modelo de datos se reflejen automáticamente en los componentes de la interfaz y viceversa, sin que el desarrollador tenga que escribir mucho código para actualizar manualmente estos valores.

A. Concepto de Binding de Datos

El binding de datos conecta un modelo de datos (como una clase que representa una entidad del negocio) con los componentes de la interfaz gráfica (como campos de texto, etiquetas, botones, etc.). Este mecanismo garantiza que:

- **Cambio en el modelo de datos**: Cuando se modifica un valor en el modelo, el componente de la interfaz gráfica asociado refleja automáticamente este cambio.
- **Cambio en la interfaz gráfica**: Cuando el usuario modifica un valor en un componente de la interfaz, el modelo de datos se actualiza automáticamente.

La idea de binding de datos es fundamental en el diseño de aplicaciones que requieren una **sincronización constante**entre la interfaz gráfica y la lógica de la aplicación, facilitando la **separación de preocupaciones** entre la lógica del negocio y la interfaz de usuario.



B. Binding de Datos en el Modelo-Vista-Controlador (MVC)

El binding de datos es especialmente útil en arquitecturas de diseño como **Modelo-Vista-Controlador (MVC)**, donde se desea que el modelo y la vista estén sincronizados sin intervención directa del controlador. En este patrón:

- **Modelo**: Representa los datos de la aplicación y su lógica de negocio.
- **Vista**: Muestra los datos del modelo y permite la interacción del usuario.
- **Controlador**: Coordina la interacción entre la vista y el modelo.

El binding de datos ayuda a implementar MVC de manera eficiente, al reducir la carga del controlador, ya que los cambios en el modelo o en la vista se reflejan automáticamente en el otro.

C. Implementación de Binding de Datos en Java

Java Swing, aunque es una biblioteca potente para crear interfaces gráficas, no incluye soporte nativo para el binding de datos de manera directa. Por esta razón, se utilizan bibliotecas adicionales como **Beans Binding (JSR 295)** y **JGoodies Binding**, que permiten la sincronización entre el modelo de datos y los componentes de la interfaz gráfica. A continuación, se explican estas opciones:

D. Estrategias de Actualización en Binding de Datos

Existen diferentes estrategias para actualizar los datos entre el modelo y la vista, lo cual afecta cómo se sincronizan los datos. Las estrategias más comunes incluyen:

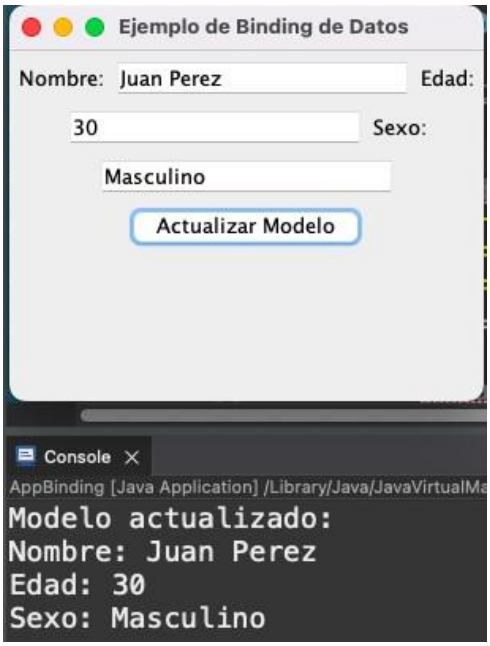
- **READ**: Los datos fluyen solo desde el modelo hacia la vista. Útil cuando el componente de la interfaz solo debe mostrar información sin modificarla.
- **WRITE**: Los datos fluyen solo desde la vista hacia el modelo. Se usa cuando el componente de la interfaz actúa como un controlador, por ejemplo, cuando solo se necesita capturar la entrada del usuario.
- **READ_WRITE**: Los datos fluyen en ambas direcciones, lo cual permite una sincronización completa. Es la estrategia más común en aplicaciones con interfaces interactivas, donde se espera que los cambios en la interfaz reflejen en el modelo, y viceversa.

E. Ejemplo básico de binding de datos en Java Swing

1. La interfaz gráfica muestra los valores iniciales de un objeto Persona en campos de texto.
2. El usuario puede modificar los valores en los campos de texto.
3. Al presionar el botón "Actualizar Modelo", los cambios realizados en la interfaz se sincronizan manualmente con el modelo Persona.
4. Los valores actualizados se imprimen en consola para verificar que el binding de datos manual funcionó correctamente.

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class App {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Ejemplo de Binding de Datos");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 250);

        // Modelo de datos
        Persona persona = new Persona("John Doe", 30, "Masculino");

        // Componentes de interfaz para cada atributo
        JTextField nombreField = new JTextField(persona.getNombre(), 15);
        JTextField edadField = new JTextField(String.valueOf(persona.getEdad()), 15);
        JTextField sexoField = new JTextField(persona.getSexo(), 15);
        JButton button = new JButton("Actualizar Modelo");

        // Listener para actualizar el modelo cuando el botón es presionado
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                persona.setNombre(nombreField.getText());
                try {
                    int edad = Integer.parseInt(edadField.getText());
                    persona.setEdad(edad);
                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(frame, "Por favor, ingrese un número válido para la
edad.");
                }
                persona.setSexo(sexoField.getText());

                // Mostrar los valores actualizados en consola
                System.out.println("Modelo actualizado:");
                System.out.println("Nombre: " + persona.getNombre());
                System.out.println("Edad: " + persona.getEdad());
                System.out.println("Sexo: " + persona.getSexo());
            }
        });

        frame.setLayout(new java.awt.FlowLayout());
        frame.add(new JLabel("Nombre:"));
        frame.add(nombreField);
        frame.add(new JLabel("Edad:"));
        frame.add(edadField);
        frame.add(new JLabel("Sexo:"));
        frame.add(sexoField);
        frame.add(button);

        frame.setVisible(true);
    }
}

// Modelo de datos actualizado con nombre, edad y sexo
class Persona {
    private String nombre;
    private int edad;
    private String sexo;

    public Persona(String nombre, int edad, String sexo) {
        this.nombre = nombre;
```

```
        this.edad = edad;
        this.sexo = sexo;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public String getSexo() {
        return sexo;
    }

    public void setSexo(String sexo) {
        this.sexo = sexo;
    }
}
```



2. GRÁFICOS

El **manejo de gráficos en Java Swing** es una técnica que permite crear y manipular elementos visuales dentro de aplicaciones de interfaz gráfica en Java. Swing, como biblioteca de GUI (interfaz gráfica de usuario), facilita el desarrollo de aplicaciones que incluyen componentes visuales estándar (como botones, campos de texto y menús), y también ofrece soporte para crear gráficos personalizados, como dibujos, formas geométricas, animaciones y visualizaciones de datos.

A. Fundamentos de los Gráficos en Java Swing

Java Swing utiliza el modelo de gráficos de **AWT (Abstract Window Toolkit)**, pero con una estructura más moderna y orientada a objetos. En Swing, el manejo de gráficos se basa en la clase Graphics, una

abstracción que permite manipular elementos gráficos en un componente visual. Para crear gráficos personalizados, los desarrolladores extienden componentes de Swing, como JPanel, y sobrescriben el método paintComponent(Graphics g).

- **Componente de Dibujo**: La mayoría de los gráficos en Swing se implementan dentro de un componente que extiende JPanel. Este panel actúa como un lienzo sobre el cual se pueden dibujar gráficos personalizados.
- **Método paintComponent**: Este método se invoca automáticamente cuando el sistema necesita redibujar el componente (por ejemplo, cuando se minimiza y se restaura la ventana). Al sobrescribir paintComponent, se puede acceder a un objeto Graphics que permite realizar dibujos en el panel.

B. La Clase Graphics y Graphics2D

La clase Graphics es el núcleo del dibujo en Java. Permite dibujar líneas, rectángulos, óvalos, polígonos, texto e imágenes en un componente. En versiones posteriores de Java, se introdujo Graphics2D, una extensión de Graphicsque proporciona más opciones para gráficos avanzados, como trazos personalizados, rotación, escalado y antialiasing (suavizado de bordes).

B.1 Métodos de Graphics

Algunos de los métodos más comunes de Graphics incluyen:

Método	Descripción
drawLine(int x1, int y1, int x2, int y2)	Dibuja una línea entre los puntos (x1, y1) y (x2, y2).
drawRect(int x, int y, int width, int height)	Dibuja un rectángulo con la esquina superior izquierda en (x, y) y las dimensiones especificadas por width y height.
fillRect(int x, int y, int width, int height)	Dibuja un rectángulo relleno con la esquina superior izquierda en (x, y) y las dimensiones especificadas.
drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)	Dibuja un rectángulo con esquinas redondeadas; arcWidth y arcHeight especifican el diámetro de los arcos de las esquinas.
fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)	Dibuja un rectángulo relleno con esquinas redondeadas.
drawOval(int x, int y, int width, int height)	Dibuja un óvalo dentro de un rectángulo delimitador con esquina superior izquierda en (x, y) y dimensiones especificadas.
fillOval(int x, int y, int width, int height)	Dibuja un óvalo relleno dentro del rectángulo delimitador especificado.
drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)	Dibuja un arco dentro de un rectángulo delimitador, comenzando en startAngle y abarcando arcAngle grados.
fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)	Dibuja un arco relleno en el rectángulo delimitador.
drawPolygon(int[] xPoints, int[] yPoints, int nPoints)	Dibuja un polígono con vértices definidos por los arrays xPoints e yPoints y el número de puntos nPoints.
fillPolygon(int[] xPoints, int[] yPoints, int nPoints)	Dibuja un polígono relleno con los vértices definidos por los arrays xPoints e yPoints.

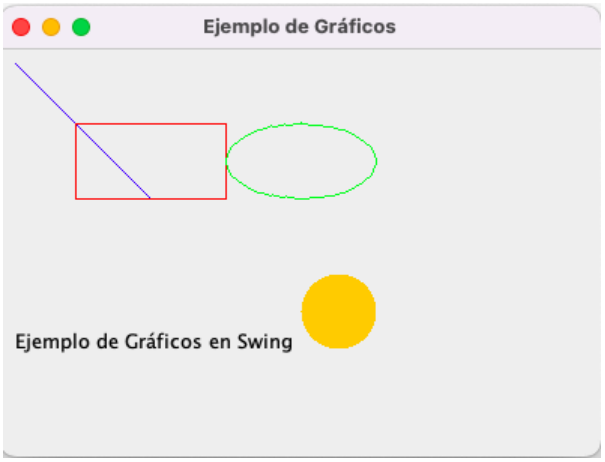
drawPolyline(int[] xPoints, int[] yPoints, int nPoints)	Dibuja una serie de líneas conectadas que forman una polilínea, usando los puntos definidos en xPoints e yPoints.
drawString(String str, int x, int y)	Dibuja el texto especificado (str) comenzando en la posición (x, y).
drawChars(char[] data, int offset, int length, int x, int y)	Dibuja un subconjunto de caracteres de un array data, comenzando en offset y de longitud length, en la posición (x, y).
drawImage(Image img, int x, int y, ImageObserver observer)	Dibuja una imagen en la posición (x, y) utilizando un ImageObserverpara observar el estado de carga de la imagen.
drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)	Dibuja una imagen escalada en la posición (x, y)con el tamaño especificado.
drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver observer)	Dibuja una región específica de una imagen (definida por sx1, sy1, sx2, sy2) en el área destino (dx1, dy1, dx2, dy2).
clearRect(int x, int y, int width, int height)	Limpia un rectángulo (lo pinta con el color de fondo actual) con la esquina superior izquierda en (x, y) y el tamaño especificado.
copyArea(int x, int y, int width, int height, int dx, int dy)	Copia una región rectangular del área de origen (x, y, width, height) y la pega desplazada por (dx, dy).

Ejemplo básico para manejar gráficos en Java Swing

1) **Crear un componente personalizado**: Crea una clase que extienda *JPanel* o cualquier otro componente de Swing.
2) **Sobrescribir el método paintComponent**: Dentro de esta clase, sobrescribe el método *paintComponent(Graphics g)* para definir lo que se va a dibujar.
3) **Usar el objeto Graphics**: Usa el objeto *Graphics*, que recibe el método *paintComponent*, para dibujar líneas, formas, texto o imágenes.
4) **Agregar el panel al marco**: Agrega el componente a un *JFrame* u otro contenedor para mostrar el gráfico.

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

public class AppGraficos extends JPanel {

    // Sobrescribe el método paintComponent para realizar dibujos
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);  // Llama al método paintComponent de la superclase

        // Establece el color de dibujo
        g.setColor(Color.BLUE);

        // Dibuja una línea
        g.drawLine(10, 10, 100, 100);

        // Dibuja un rectángulo
        g.setColor(Color.RED);
        g.drawRect(50, 50, 100, 50);

        // Dibuja un óvalo
        g.setColor(Color.GREEN);
        g.drawOval(150, 50, 100, 50);

        // Dibuja un círculo lleno
        g.setColor(Color.ORANGE);
```

```
        g.fillOval(200, 150, 50, 50);

        // Dibuja un texto
        g.setColor(Color.BLACK);
        g.drawString("Ejemplo de Gráficos en Swing", 10, 200);
    }

    // Método principal para ejecutar el ejemplo
    public static void main(String[] args) {
        JFrame frame = new JFrame("Ejemplo de Gráficos");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);

        // Crea una instancia de nuestro panel personalizado y la agrega al marco
        AppGraficos panel = new AppGraficos();
        frame.add(panel);

        frame.setVisible(true);
    }
}
```



B.2 Métodos Avanzados de Graphics2D

Graphics2D hereda de Graphics y añade varias características avanzadas:

- **Transformaciones**: Permiten rotar, escalar o trasladar gráficos con métodos como rotate, scale y translate.
- **Trazos personalizados**: Los contornos de las formas pueden ajustarse con un grosor específico usando setStroke(new BasicStroke(grosor)).
- **Antialiasing**: Para gráficos más suaves, se puede activar el antialiasing con setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON).

Método	Descripción
draw(Shape s)	Dibuja el contorno de una forma (Shape) especificada, como un rectángulo, un óvalo, o una línea.
fill(Shape s)	Dibuja una forma rellena (Shape) especificada, como un rectángulo o un óvalo.
drawImage(Image img, AffineTransform xform, ImageObserver obs)	Dibuja una imagen transformada por un objeto AffineTransform (escalado, rotación, traslación, etc.).
drawRenderedImage(RenderedImage img, AffineTransform xform)	Dibuja una imagen RenderedImage, aplicando una transformación AffineTransform.

drawRenderableImage(RenderableImage img, AffineTransform xform)	Dibuja una imagen RenderableImage, aplicando una transformación AffineTransform.
drawString(String str, float x, float y)	Dibuja un texto (String) en la posición (x, y) usando coordenadas de punto flotante.
drawString(AttributedCharacterIterator iterator, float x, float y)	Dibuja un texto con atributos en la posición (x, y)utilizando un AttributedCharacterIterator.
drawGlyphVector(GlyphVector g, float x, float y)	Dibuja un GlyphVector (conjunto de glifos de texto) en la posición (x, y).
setStroke(Stroke s)	Establece el trazo (Stroke) para dibujar el contorno de las formas; permite personalizar el grosor y el estilo de las líneas.
setPaint(Paint paint)	Establece el color o patrón (Paint) para dibujar o rellenar las formas; acepta objetos de color sólido o gradientes.
setComposite(Composite comp)	Establece el modo de composición para combinar colores al dibujar (transparencias, superposiciones, etc.).
setRenderingHint(RenderingHints.Key hintKey, Object hintValue)	Configura una pista de renderizado (RenderingHint), como antialiasing o calidad de renderizado, para mejorar la apariencia gráfica.
setTransform(AffineTransform Tx)	Establece una transformación (AffineTransform) para rotar, escalar, o trasladar gráficos en el contexto gráfico.
transform(AffineTransform Tx)	Aplica una transformación adicional (AffineTransform) a la transformación actual, combinándolas.
rotate(double theta)	Rota el contexto gráfico en theta radianes alrededor del origen.
rotate(double theta, double x, double y)	Rota el contexto gráfico en theta radianes alrededor del punto (x, y).
scale(double sx, double sy)	Escala el contexto gráfico en sx y sy a lo largo de los ejes x e y, respectivamente.
translate(int x, int y)	Desplaza el contexto gráfico en (x, y) unidades enteras.
translate(double tx, double ty)	Desplaza el contexto gráfico en (tx, ty) unidades de punto flotante.
shear(double shx, double shy)	Aplica un sesgado (shear) en shx y shy al contexto gráfico.
setBackground(Color color)	Establece el color de fondo del contexto gráfico para las operaciones de limpieza (clearRect).
clearRect(int x, int y, int width, int height)	Limpia un área rectangular, llenándola con el color de fondo actual.
drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver observer)	Dibuja una subimagen especificada de img en un área destino, aplicando un escalado.
drawImage(BufferedImage img, BufferedImageOp op, int x, int y)	Dibuja una imagen BufferedImage aplicando una operación de filtro (BufferedImageOp) en la posición (x, y).

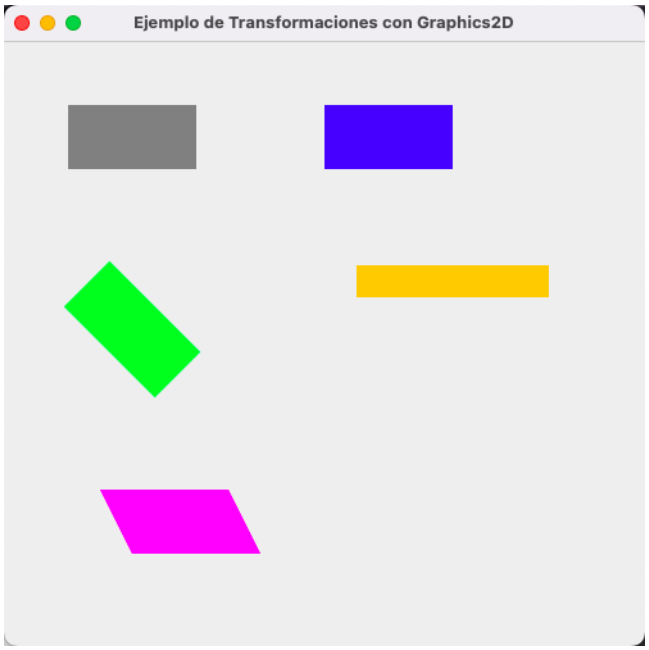
Ejemplo básico para manejar gráficos 2D en Java Swing

1) **Antialiasing**: Activamos el antialiasing para suavizar los bordes de las formas y mejorar la apariencia visual.

2) **Dibujo del Rectángulo Original**: Dibujamos un rectángulo gris en (50, 50) sin ninguna transformación, para tener un punto de referencia.
3) **Transformaciones Aplicadas**:
    a. **Traslación**: Usamos translate(200, 0) para mover el contexto gráfico 200 píxeles a la derecha, y luego dibujamos un rectángulo azul. La traslación se aplica a todas las operaciones de dibujo hasta que se restablezca la transformación.
    b. **Rotación**: Aplicamos rotate(Math.toRadians(45), 100, 75) para rotar 45 grados alrededor del centro del rectángulo (100, 75). Dibujamos un rectángulo verde para ver el efecto de rotación.
    c. **Escalado**: Usamos scale(1.5, 0.5) para escalar el contexto gráfico 1.5 veces en el eje X y 0.5 veces en el eje Y, lo que afecta las dimensiones del rectángulo naranja.
    d. **Sesgado (Shear)**: Aplicamos shear(0.5, 0) para sesgar el rectángulo horizontalmente, distorsionando la forma del rectángulo magenta.
4) **Restauración de la Transformación Original**: Después de cada transformación, se usa setTransform(originalTransform) para restablecer el contexto gráfico a su estado original y evitar que las transformaciones se acumulen. Esto permite aplicar transformaciones independientes para cada rectángulo.

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

public class AppGraficos2D extends JPanel {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        // Convertimos Graphics a Graphics2D para aplicar transformaciones
        Graphics2D g2d = (Graphics2D) g;

        // Activamos el antialiasing para suavizar los bordes
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

        // Definimos el color y dibujamos un rectángulo original en gris
        g2d.setColor(Color.GRAY);
        int rectWidth = 100;
        int rectHeight = 50;
        g2d.fillRect(50, 50, rectWidth, rectHeight);

        // Guardamos el estado original de la transformación
        AffineTransform originalTransform = g2d.getTransform();

        // Traslación
        g2d.translate(200, 0); // Traslada 200 píxeles a la derecha
        g2d.setColor(Color.BLUE);
        g2d.fillRect(50, 50, rectWidth, rectHeight);

        // Restauramos la transformación original
        g2d.setTransform(originalTransform);

        // Rotación
        g2d.translate(0, 150); // Movemos hacia abajo para evitar superposición
        g2d.rotate(Math.toRadians(45), 100, 75); // Rota 45 grados alrededor del centro del rectángulo
        g2d.setColor(Color.GREEN);
        g2d.fillRect(50, 50, rectWidth, rectHeight);

        // Restauramos la transformación original
        g2d.setTransform(originalTransform);

        // Escalado
        g2d.translate(200, 150); // Movemos hacia la derecha y abajo
```

```
        g2d.scale(1.5, 0.5); // Escala 1.5x en X y 0.5x en Y
        g2d.setColor(Color.ORANGE);
        g2d.fillRect(50, 50, rectWidth, rectHeight);

        // Restauramos la transformación original
        g2d.setTransform(originalTransform);

        // Sesgado (Shear)
        g2d.translate(0, 300); // Movemos hacia abajo para evitar superposición
        g2d.shear(0.5, 0); // Aplica un sesgado horizontal
        g2d.setColor(Color.MAGENTA);
        g2d.fillRect(50, 50, rectWidth, rectHeight);

        // Restauramos la transformación original al final para no afectar futuras operaciones
        g2d.setTransform(originalTransform);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Ejemplo de Transformaciones con Graphics2D");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(500, 500);

        // Añadimos el panel de dibujo al frame
        frame.add(new AppGraficos2D());
        frame.setVisible(true);
    }
}
```



B.3 Aplicaciones Avanzadas de los Gráficos en Java Swing

El manejo de gráficos en Java Swing es útil para muchas aplicaciones más avanzadas:

- **Animación:** La animación se puede lograr actualizando el contenido del panel en intervalos de tiempo regulares, generalmente utilizando un javax.swing.Timer que ejecuta un repaint() del componente en cada intervalo.
- **Visualización de Datos:** Para crear gráficos de datos (como gráficas de líneas o barras), se puede utilizar Graphics2D para trazar los puntos y líneas necesarios y generar gráficos informativos.

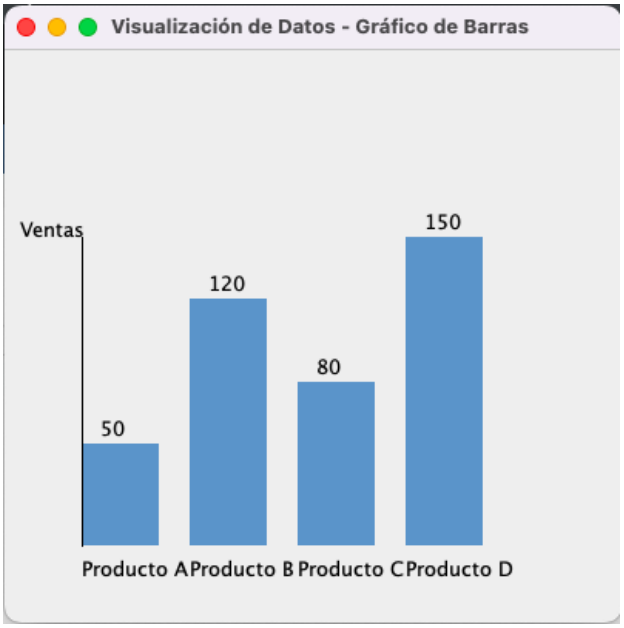
- **Interactividad:** Se puede combinar el manejo de gráficos con eventos del ratón (MouseListener) y del teclado (KeyListener) para crear aplicaciones interactivas, como editores de gráficos o juegos.

Ejemplo básico para manejar gráficos avanzados en Java Swing

1. **Definición de los Datos**: Los datos de ejemplo (productos y ventas) representan nombres de productos y la cantidad de ventas de cada uno.
2. **Escalado de Barras**: Calculamos el valor máximo en ventas para escalar proporcionalmente las alturas de las barras en función del valor máximo (alturaMaxima).
3. **Dibujo del Gráfico de Barras**:
    - Para cada producto, calculamos la posición y altura de su barra basándonos en el valor de ventas y el máximo valor encontrado.
    - Usamos fillRect para dibujar la barra y drawString para agregar etiquetas con el valor de ventas encima de cada barra.
    - Las etiquetas de productos se dibujan debajo de cada barra.
4. **Dibujo del Eje Y**: Se dibuja una línea vertical a la izquierda como eje Y, con el título "Ventas" para darle contexto al gráfico.

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

public class AppGraficoBarras extends JPanel {
    // Datos de ejemplo
    private String[] productos = {"Producto A", "Producto B", "Producto C", "Producto D"};
    private int[] ventas = {50, 120, 80, 150}; // Ventas en unidades

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        // Configuración de Graphics2D y antialiasing
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

        // Configuración de las dimensiones y colores del gráfico
        int anchoBarra = 50;
        int espacioBarra = 20;
        int margenIzquierdo = 50;
        int margenInferior = 50;
        int alturaMaxima = 200;

        // Encontrar el valor máximo de ventas para escalar las barras
        int maxVenta = 0;
        for (int venta : ventas) {
            if (venta > maxVenta) {
                maxVenta = venta;
            }
        }

        // Dibujar las barras
        for (int i = 0; i < productos.length; i++) {
            int alturaBarra = (int) ((double) ventas[i] / maxVenta * alturaMaxima);
            int x = margenIzquierdo + i * (anchoBarra + espacioBarra);
            int y = getHeight() - margenInferior - alturaBarra;

            // Dibujar la barra
            g2d.setColor(new Color(100, 150, 200)); // Color de la barra
            g2d.fillRect(x, y, anchoBarra, alturaBarra);

            // Dibujar la etiqueta de ventas encima de cada barra
            g2d.setColor(Color.BLACK);
```

```
            g2d.drawString(String.valueOf(ventas[i]), x + (anchoBarra / 4), y - 5);

            // Dibujar el nombre del producto debajo de cada barra
            g2d.drawString(productos[i], x, getHeight() - margenInferior + 20);
        }

        // Dibujar el eje y y su título
        g2d.drawLine(margenIzquierdo, getHeight() - margenInferior, margenIzquierdo, getHeight() -
margenInferior - alturaMaxima);
        g2d.drawString("Ventas", margenIzquierdo - 40, getHeight() - margenInferior - alturaMaxima);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Visualización de Datos - Gráfico de Barras");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 400);

        // Agregar el panel de gráficos al frame
        AppGraficoBarras grafico = new AppGraficoBarras();
        frame.add(grafico);

        frame.setVisible(true);
    }
}
```



3. MULTIMEDIA

El manejo de multimedia en Java Swing se refiere a la capacidad de integrar elementos como audio, video e imágenes en aplicaciones de interfaz gráfica. Aunque Swing es principalmente una biblioteca para crear interfaces de usuario en 2D, Java proporciona varias herramientas y bibliotecas de terceros para integrar contenido multimedia, lo cual amplía sus aplicaciones en áreas como reproductores multimedia, visualización de datos y aplicaciones interactivas.

A. Fundamentos del Manejo de Multimedia en Java

Java es un lenguaje orientado a objetos que, desde sus primeras versiones, ofrece soporte para manipular imágenes y audio. Sin embargo, Swing no proporciona soporte nativo directo para reproducir

video o audio de forma avanzada. Para manejar multimedia en Java Swing, se recurre a clases básicas para imágenes y audio, y a bibliotecas externas para una integración más avanzada y robusta de video.

B. Manejo de Audio en Java Swing

Para el manejo de audio, Java proporciona la biblioteca javax.sound.sampled, que permite reproducir, grabar y manipular archivos de audio en formatos como WAV y AU.

Principales clases y métodos para manejar audio:

- **Clip**: Es una interfaz que permite cargar y reproducir un archivo de audio. Es ideal para reproducir sonidos de corta duración (como efectos de sonido).
- **AudioInputStream**: Proporciona un flujo de datos de audio para reproducir archivos de audio desde una fuente, como un archivo o un recurso en red.
- **AudioSystem**: Es una clase de utilidad que permite acceder al sistema de audio y cargar archivos de sonido, y facilita la conversión de formatos y la reproducción.

```
import javax.sound.sampled.*;
import java.io.File;

File audioFile = new File("ruta/a/archivo.wav");
AudioInputStream audioStream = AudioSystem.getAudioInputStream(audioFile);
Clip clip = AudioSystem.getClip();
clip.open(audioStream);
clip.start(); // Reproduce el audio
```

Para reproducir otros formatos de audio, como MP3, es necesario utilizar bibliotecas de terceros, ya que javax.sound.sampled no soporta estos formatos. Una biblioteca popular para MP3 es **JavaZoom JLayer**.

Ejemplo básico de reproducción de un archivo WAV:

1. **AudioInputStream**: Se usa para leer el archivo de audio.
2. **Clip**: Representa una secuencia de audio que se puede reproducir, pausar y detener.
3. **playAudio**: Abre y reproduce el archivo de audio cuando se presiona el botón.

```
import javax.sound.sampled.*;
import javax.swing.*;
import java.io.File;
import java.io.IOException;

public class AppAudio {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Reproducción de Audio");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 200);

        JButton playButton = new JButton("Reproducir Audio");
        playButton.addActionListener(e -> playAudio("sample-3s.wav"));
```

```
        frame.add(playButton);
        frame.setVisible(true);
    }

    public static void playAudio(String filePath) {
        try {
            File audioFile = new File(filePath);
            AudioInputStream audioStream = AudioSystem.getAudioInputStream(audioFile);
            Clip clip = AudioSystem.getClip();
            clip.open(audioStream);
            clip.start();
        } catch (UnsupportedAudioFileException | IOException | LineUnavailableException e) {
            e.printStackTrace();
        }
    }
}
```

C. Manejo de Video en Java Swing

El manejo de video es más complejo que el de audio y no tiene soporte nativo en Java Swing. Sin embargo, hay bibliotecas de terceros que permiten reproducir videos dentro de una aplicación Swing:
Principales bibliotecas para manejar video:

- **JavaFX**: A partir de Java 8, JavaFX se puede integrar con Swing para reproducir video mediante el uso de MediaPlayer y MediaView.
- **VLCJ**: Una biblioteca que utiliza el reproductor VLC para proporcionar soporte de video en Java, permitiendo reproducir una amplia variedad de formatos.
- **Xuggler (descontinuada)**: Esta biblioteca de código abierto permite la manipulación de video, pero ya no se actualiza activamente.

Ejemplo básico de reproducción de un video MP4:

Integración de JavaFX en Swing para reproducir video

1. Asegúrate de usar al menos Java 8.

2. Configura JavaFX en Eclipse (puedes mirarlo en el siguiente enlace) https://youtu.be/nz8P528uGjk?si=_lu_2rC7XHntgdIa

3. Usa JFXPanel, que permite integrar componentes JavaFX en una aplicación Swing.

a) **JFXPanel**: Permite incluir componentes JavaFX en Swing.
b) **Platform.runLater**: Asegura que el código JavaFX se ejecute en el hilo adecuado.
c) **Media y MediaPlayer**: Clases de JavaFX que permiten reproducir video.
d) **MediaView**: Componente de JavaFX para mostrar el video.

```
import javafx.application.Platform;
import javafx.embed.swing.JFXPanel;
import javafx.scene.Scene;
import  javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;

import javax.swing.*;
import java.io.File;

public class VideoExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Reproducción de Video");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(800, 600);

        JFXPanel jfxPanel = new JFXPanel();
        frame.add(jfxPanel);
        frame.setVisible(true);

        Platform.runLater(() -> {
            String videoPath = new File("ruta_del_video.mp4").toURI().toString();
            Media media = new Media(videoPath);
            MediaPlayer mediaPlayer = new MediaPlayer(media);
            MediaView mediaView = new MediaView(mediaPlayer);

            jfxPanel.setScene(new Scene(new javafx.scene.Group(mediaView)));
            mediaPlayer.play();
        });
    }
}
```

4. RESUMEN

El Binding de Datos en Java Swing es una técnica que permite sincronizar automáticamente los datos entre un modelo de datos (lógica de negocio) y la interfaz gráfica, facilitando que los cambios en el modelo se reflejen en la vista y viceversa. El manejo de gráficos en Java Swing utiliza las

clases Graphics y Graphics2D para dibujar y manipular visualmente elementos como formas, líneas, texto y gráficos personalizados, permitiendo crear visualizaciones interactivas y dinámicas. El manejo de multimedia en Java Swing permite la integración de contenido como imágenes, audio y video en aplicaciones de interfaz gráfica; aunque Swing incluye soporte básico para imágenes y audio en formatos como WAV, el uso de bibliotecas externas como JavaFX o VLCJ es común para la reproducción avanzada de audio y video en formatos populares como MP3 y MP4.

IV

(La práctica tiene una duración de 4 horas) ACTIVIDADES

1. EXPERIENCIA DE PRÁCTICA:

Primero prueben ejecutar cada uno de los ejemplos vistos anteriormente, analicen las clases involucradas y la secuencia de pasos para ejecutar cada aplicación.

Luego creen 2 nuevas ventanas por integrante donde se realicen las siguientes acciones:

- Binding de Datos manual con Swing
- Creación de gráficos simples con Swing
- Creación de gráficos avanzados con Swing
- Creación de aplicaciones que utilicen elementos multimedia con Swing

Las ventanas pueden hacer uso combinado de una o más de las acciones vistas anteriormente.

V

EJERCICIOS PROPUESTOS

Trabajo Grupal (3 a 4 Integrantes)

1. Primero deberán realizar al menos 2 de las siguientes 4 aplicaciones propuestas:
   - Crea una aplicación que gestione la información de un producto en una tienda. La aplicación debe permitir ingresar el **nombre del producto**, **precio**, **cantidad en stock** y **categoría**. Al hacer clic en el botón "Actualizar Producto", los datos deben actualizarse en el modelo de datos y mostrarse en una etiqueta en la interfaz. Crear una clase *Producto* con los atributos *nombre*, *precio*, *cantidadStock*, y *categoria*. Diseñar una interfaz en Swing con campos de texto para cada atributo del producto, un botón "Actualizar Producto" y una etiqueta *JLabel* para mostrar la información actualizada. Al presionar el botón, el modelo de datos Producto debe actualizarse y la etiqueta debe mostrar la información del producto deforma estructurada.
   - Crea una aplicación para registrar y visualizar la temperatura diaria de una semana medianteun gráfico de líneas. La aplicación debe permitir al usuario ingresar la temperatura para cada día de la semana y mostrar una línea que conecte los puntos correspondientes a cada día en elgráfico. Diseñar una interfaz con: Siete campos de texto para ingresar la temperatura de cada día (de lunes a domingo). Un botón "Mostrar

Gráfico" que dibuje el gráfico de líneas. Crear un panel donde se dibuje el gráfico de líneas, con puntos para cada día y una línea que conecte los puntos. Al presionar "Mostrar Gráfico", el gráfico debe actualizarse y mostrar las temperaturas para cada día.

- Crea una aplicación de Swing que funcione como un reproductor de efectos de sonido. La aplicación debe permitir reproducir diferentes sonidos al hacer clic en botones específicos. Cada botón representa un efecto de sonido distinto (por ejemplo, "Aplausos", "Campana" y "Explosión"). Diseñar una interfaz con varios botones, cada uno representando un efecto de sonido. Botones como "Aplausos", "Campana" y "Explosión". Al hacer clic en un botón, el sonido correspondiente debe reproducirse. Utilizar la clase Clip de *javax.sound.sampled* parareproducir archivos de sonido en formato WAV.
- Crea una aplicación de Java Swing que permita al usuario reproducir, pausar y reanudar una pista de música. La aplicación debe tener botones de "Reproducir", "Pausar" y "Reanudar" que controlen la reproducción del audio. Diseñar una interfaz con tres botones: "Reproducir", "Pausar" y "Reanudar". Al presionar "Reproducir", el audio comienza a reproducirse desde el inicio. Al presionar "Pausar", el audio se detiene temporalmente, y al presionar "Reanudar" continúa desde donde se detuvo. Utilizar *Clip* de *javax.sound.sampled* para cargar y controlar lareproducción de audio.

2. Deben crear una aplicación de temática libre que utilice el patrón **Modelo-Vista-Controlador (MVC)** y la librería **Swing** para la interfaz gráfica. La aplicación debe interactuar con una **base de datos** para gestionar y almacenar información de acuerdo con las necesidades de la aplicación elegida, tomando en cuenta las siguientes consideraciones:
   - Antes de comenzar a programar, deberán realizar un diseño en papel (mockup) de la interfaz gráfica de su aplicación. Este diseño debe incluir las pantallas principales de la aplicación, como los formularios, tablas o menús, y debe mostrar cómo será la disposición de los elementos en la ventana.
   - El diseño en papel será evaluado para asegurar que tienen una visión clara de la estructura visual y de la interacción con el usuario.
   - La interfaz debe ser desarrollada utilizando la librería **Swing** de Java. La interfaz debe ser clara, fácil de usar y con una navegación lógica y amigable.
   - Deben integrar una base de datos para almacenar y gestionar la información de tu aplicación. Puede ser una base de datos local como SQLite o una base de datos MySQL
   - La aplicación debe seguir el patrón **Modelo-Vista-Controlador (MVC)** para asegurar una correcta separación de responsabilidades

V
CUESTIONARIO

1. ¿Cómo se puede sincronizar un modelo de datos y una interfaz gráfica en Swing de forma manual?
2. ¿Qué métodos de los componentes de Swing se utilizan comúnmente para obtener y establecer valores al realizar binding de datos manual?
3. ¿Cómo se maneja la validación de datos en un formulario que utiliza binding de datos manual?

4. ¿Cómo podrías implementar un binding de datos manual para una lista de objetos y mostrar los datos en un componente JTable?
5. ¿Cómo se pueden crear gráficos interactivos que respondan a eventos de ratón y teclado en un JPanel en Swing?
6. ¿Cómo podrías implementar un gráfico de datos (barras o líneas) utilizando Graphics en Swing para visualizar valores numéricos y permitir que se actualicen dinámicamente?
7. ¿Cómo funciona el método paintComponent(Graphics g) y cuándo se llama automáticamente?
8. ¿Qué es antialiasing y cómo se aplica en gráficos creados con Graphics2D en Java Swing?
9. ¿Qué clases de Java estándar permiten trabajar con audio en una aplicación Swing y cuáles son sus principales métodos?
10. ¿Cuál es la diferencia entre los formatos de audio soportados nativamente en Java (WAV, AU) y otros formatos populares como MP3? ¿Cómo se pueden reproducir estos formatos en Java Swing?
11. ¿Cómo se usa la clase Clip en javax.sound.sampled para cargar y reproducir archivos de audio en Java Swing?
12. ¿Cómo se pueden pausar, detener y reanudar archivos de audio en una aplicación de Java Swing utilizando la clase Clip?

VI
BIBLIOGRAFIA Y REFERENCIAS

1. https://docs.oracle.com/javase/
2. Libro: "Core Java Volume I – Fundamentals" de Cay S. Horstmann y Gary Cornell
3. https://docs.oracle.com/javase/tutorial/uiswing/

LENGUAJES DE PROGRAMACIÓN III
RUBRICA DE EVALUACIÓN: PRACTICA N° 10: JAVA SWING AVANZADO

% / CRITERIOS	Excelente (100%)	Bueno (75%)	Suficiente (50%)	Insuficiente (0%)
Puntualidad en la presentación del informe (10%)	El informe se entregó en la fecha y hora especificadas.	El informe se entregó con un ligero retraso (menos de 24 horas).	El informe se entregó con un retraso significativo (más de 24 horas).	El informe no se entregó.
Trabajo en equipo (10%)	El equipo demostró una excelente colaboración y comunicación durante todo el proyecto. Todos los miembros contribuyeron de manera equitativa y se apoyaron mutuamente.	El equipo trabajó bien en general, pero hubo algunos problemas menores de comunicación o distribución de tareas.	La colaboración en equipo fue aceptable, pero hubo algunos conflictos o falta de participación de algunos miembros.	El equipo no demostró una buena colaboración. Hubo conflictos significativos o falta de participación de varios miembros.
Completitud del informe (10%)	El informe incluye todos los elementos requeridos y responde a todas las preguntas planteadas en la guía de la práctica.	El informe incluye la mayoría de los elementos requeridos y responde a la mayoría de las preguntas, pero puede faltar algún detalle o explicación.	El informe incluye algunos de los elementos requeridos y responde a algunas preguntas, pero faltan varios detalles importantes o explicaciones.	El informe está incompleto y no responde a las preguntas clave de la guía de la práctica.
Binding de Datos (20%)	Cumple con todos los siguientes criterios: 1) Utiliza clases modelo correctamente 2) Maneja Interfaces de escucha asociada a controles 3) Maneja los métodos get y set vinculados a interfaces de escucha	Cumple con 2 de los siguientes criterios: 1) Utiliza clases modelo correctamente 2) Maneja Interfaces de escucha asociada a controles 3) Maneja los métodos get y set vinculados a interfaces de escucha	Cumple con 1 de los siguientes criterios: 1) Utiliza clases modelo correctamente 2) Maneja Interfaces de escucha asociada a controles 3) Maneja los métodos get y set vinculados a interfaces de escucha	No Cumple con ninguno de los siguientes criterios: 1) Utiliza clases modelo correctamente 2) Maneja Interfaces de escucha asociada a controles 3) Maneja los métodos get y set vinculados a interfaces de escucha
Manejo de Gráficos (30%)	Cumple con todos los siguientes criterios: 1) Maneja los métodos de la Clase Graphics 2) Maneja los métodos de la clase Graphics2D 3) Maneja visualización de datos	Cumple con 2 de los siguientes criterios: 1) Maneja los métodos de la Clase Graphics 2) Maneja los métodos de la clase Graphics2D 3) Maneja visualización de datos	Cumple con 1 de los siguientes criterios: 1) Maneja los métodos de la Clase Graphics 2) Maneja los métodos de la clase Graphics2D 3) Maneja visualización de datos	No Cumple con ninguno de los siguientes criterios: 1) Maneja los métodos de la Clase Graphics 2) Maneja los métodos de la clase Graphics2D 3) Maneja visualización de datos
Manejo de Multimedia (20%)	Cumple con todos los siguientes criterios: 1) Maneja las clases y métodos para manipular Audio 2) Maneja las clases y métodos para manipular Video 3) Maneja librerías externas para manipular video	Cumple con 2 de los siguientes criterios: 1) Maneja las clases y métodos para manipular Audio 2) Maneja las clases y métodos para manipular Video 3) Maneja librerías externas para manipular video	Cumple con 1 de los siguientes criterios: 1) Maneja las clases y métodos para manipular Audio 2) Maneja las clases y métodos para manipular Video 3) Maneja librerías externas para manipular video	No Cumple con ninguno de los siguientes criterios: 1) Maneja las clases y métodos para manipular Audio 2) Maneja las clases y métodos para manipular Video 3) Maneja librerías externas para manipular video