# Universidad Nacional Autónoma de México

## Facultad de ingeniería

## Sistemas operativos

**Grupo: 6**

**My feelings**

**Profesor: Gunnar Eyal Wolf Iszaevich**

**Alumnos:**

**Vera Morales Juan Pablo**

**López Estrada Carlos**

**Semestre 2025-2**

# Feelings

## Introducción

Durante nuestro día a día, se nos presentan diversos problemas, tales que a cada una de ellas reaccionamos de manera diferente, la problemática en que relacionamos este proyecto con nuestra vida cotidiana es que en particular hay momentos que no solamente nos sentimos enojados o felices, hay una combinación de sentimientos. Todos nosotros experimentamos una gama de emociones que se activan o desactivan dependiendo de lo que vivimos. Por ejemplo, podemos sentirnos felices por una comida rica, pero al mismo tiempo frustrados porque se nos cayó al suelo. En otras ocasiones, la ansiedad y la tristeza pueden coexistir en una misma situación (como un mal resultado académico, aun sabiendo que trabajamos duro).

Estas emociones se disparan casi al mismo tiempo, conviven, se influyen mutuamente, y cambian su intensidad a lo largo del tiempo. Esta dinámica es un claro ejemplo de concurrencia: varias "entidades" (emociones) están activas, modificándose y afectándose a la vez, de forma no secuencial.

Queremos modelar este sistema mediante un programa en Java que refleje cómo estas emociones interactúan, usando hilos, semaforización, barreras y otros mecanismos de sincronización.

## Consecuencias nocivas en la concurrencia

En la vida real, la concurrencia emocional descontrolada puede tener consecuencias claras, la primera sería conflictos entre emociones, por ejemplo, sentir alegría y tristeza a la vez puede generar confusión, contradicciones internas o fatiga emocional. La interferencia, una emoción puede suprimir a otra, por ejemplo, la ansiedad puede opacar completamente una emoción positiva. Y finalmente una sobrecarga emocional, si demasiadas emociones se intensifican al mismo tiempo, puede haber un “desbordamiento” mental que lleva a decisiones impulsivas o desconexión

En términos computacionales o que vimos en clase  eso sería como tener condiciones de carrera, bloqueos o recursos saturados.

## Eventos concurrentes y control necesarios

Los siguientes eventos ocurren de manera concurrente, la aparición de una emoción en respuesta a estímulos, leer una frase que expresa alegría o enojo.
La decadencia gradual de una emoción cuando ya no hay estímulo que la mantenga, la interacción entre emociones, y una visualización gráfica sincronizada mediante colores relacionados a dicha emoción, mediante barras de intensidad.

Para evitar resultados incoherentes o conflictos, es necesario controlar cuántas emociones pueden actualizarse al mismo tiempo usando semáforos. Que ciertas emociones solo se activan juntas, con ello usaremos usando barreras.
Que no se solapen accesos concurrentes a la interfaz o los mismos valores (evitando condiciones de carrera).

**Eventos concurrentes cuyo ordenamiento no importa**

Hay momentos en los que el orden relativo de aparición o atenuación de emociones no es importante, tenemos que, si alegría y sorpresa aparecen al mismo tiempo por una buena noticia, el orden exacto en que se actualicen sus barras no cambia la experiencia emocional.
Ahora, sí tristeza y enojo se activan por un mismo evento negativo, el orden en que decaen no afecta el resultado, mientras ambas lo hagan de forma sincronizada con la lógica emocional.

Esto permite aplicar estructuras o grupos de hilos sin necesidad de ordenarlos estrictamente.

Este problema es un reflejo directo de nuestro día a día emocional, y representa un campo interesante para aplicar conceptos de concurrencia, sincronización y visualización interactiva. Además, va más allá de un problema clásico de múltiples actores, ya que los actores (emociones) están en nuestra mente, interactúan de manera compleja, y nos afectan de manera constante.

Este programa en Java busca simular este proceso, usando hilos y estructuras de concurrencia para mostrar de forma elegante cómo estas emociones evolucionan, se regulan y conviven.

**Ahora sí, vamos al código.**

## Análisis del código

Como mencionamos al inicio la realización de este proyecto fue en lenguaje Java, empezamos entonces por el main del mismo:

```java
import javax.swing.*;

public class Pri {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(EmocionesApp::new);
    }
}
```

Lo primero que vemos es el import a la librería swing que será utilizada a lo largo del código para darle interfaz a nuestro proyecto, en nuestro main llamamos a la función de utilities “invokeLater” sobre la clase EmocionesApp, esta función sirve para asegurarse de que la GUI se ejecute en el hilo de swing para manejar eventos, pasamos entonces a EmocionesApp.

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

1 usage
public class EmocionesApp extends JFrame {
    1 usage
    public EmocionesApp() {
        super( title: "Feelings");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize( width: 500,  height: 400);
        setLayout(new BorderLayout());

        EmocionesP emocionesP = new EmocionesP();
        add(emocionesP, BorderLayout.CENTER);

        JPanel inputPanel = new JPanel(new BorderLayout());
        JLabel label = new JLabel( text: "¿Cómo te sientes ahora mismo?");
        JTextField entrada = new JTextField();
        entrada.addActionListener(e -> {
            emocionesP.procesarEntrada(entrada.getText());
            entrada.setText("");
        });

        inputPanel.add(label, BorderLayout.NORTH);
        inputPanel.add(entrada, BorderLayout.CENTER);
        add(inputPanel, BorderLayout.SOUTH);

        setVisible(true);
    }
}
```

Esta cuenta con la estructura anterior y su función principal es la de conformar la ventana que contiene la interfaz grafica.

```
public EmocionesApp() {
    super( title: "Feelings");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setSize( width: 500,  height: 400);
    setLayout(new BorderLayout());
```

Definimos el titulo de la ventana y el método para cerrar la ventana además de las dimensiones de la misma.

```
EmocionesP emocionesP = new EmocionesP();
add(emocionesP, BorderLayout.CENTER);
```

Aquí, creamos una instancia de EmocionesP, clase que veremos mas adelante pero que en resumen es un panel de nuestra interfaz gráfica, posteriormente lo añadimos al centro de nuestra ventana.

```
JPanel inputPanel = new JPanel(new BorderLayout());
JLabel label = new JLabel( text: "¿Cómo te sientes ahora mismo?");
JTextField entrada = new JTextField();
entrada.addActionListener(e -> {
    emocionesP.procesarEntrada(entrada.getText());
    entrada.setText("");
});

inputPanel.add(label, BorderLayout.NORTH);
inputPanel.add(entrada, BorderLayout.CENTER);
add(inputPanel, BorderLayout.SOUTH);
```

Aquí, en la parte inferior de la interfaz añadimos una cuadro de texto donde el usuario podrá introducir la emoción que esta sintiendo en ese momento, este llama a uno de los métodos de emocionesP “procesarEntrada” que realizara la acción pertinente en la interfaz basado en la emoción que el usuario introduzca, finalmente añadimos este cuadro al fondo de la ventana.

Como vimos las funcionalidades de la ventana se encuentran en EmocionesApp vamos entonces a EmocionesP encargada de procesar y representar las entradas del usuario.

```
public class EmocionesP extends JPanel {
    3 usages
    private final Map<String, JProgressBar> barras;
    8 usages
    private final Map<String, Emocion> emociones;
    4 usages
    private final ExecutorService executor;
    9 usages
    private final Lock lockEmociones;
    3 usages
    private final Condition emocionesCompatibles;
    1 usage
    private final Semaphore semaforoGlobal = new Semaphore( permits: 3,  fair: true);

    2 usages
    private final Set<String> grupo1 = Set.of("alegria", "tristeza", "enojo");
    2 usages
    private final Set<String> grupo2 = Set.of("ansiedad", "calma");
```

Estos son los atributos que la definen, la primera corresponde a la barra de progreso que representa el como sentimos esa emoción, nos va llenando y eventualmente va mermando, el segundo corresponde a como va sintiendo esas emociones el usuario, tiene el tipo “String, Emocion” siendo emoción un tipo creado por nosotros que revisaremos después, executor de tipo executorService que gestiona los hilos de cada emoción, lockEmociones de tipo lock que sincroniza el acceso a las emociones y el poder activarlas, el semáforoGlobal limita a que haya solo 3 emociones activas al mismo tiempo.

```
private final Color[] coloresArcoiris = {
        new Color( r: 255,  g: 50,  b: 50,  a: 40), // Rojo
        new Color( r: 255,  g: 150,  b: 50,  a: 40), // Naranja
        new Color( r: 255,  g: 255,  b: 50,  a: 40), // Amarillo
        new Color( r: 50,  g: 255,  b: 50,  a: 40), // Verde
        new Color( r: 50,  g: 150,  b: 255,  a: 40), // Azul
        new Color( r: 100,  g: 50,  b: 255,  a: 40), // Índigo
        new Color( r: 200,  g: 50,  b: 255,  a: 40) // Violeta
};
```

Esta sección forma parte de la estilización de la interfaz como parte del header.

```
private JPanel crearHeaderArcoiris() {
    JPanel headerPanel = new JPanel(new BorderLayout()) {
        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            Graphics2D g2d = (Graphics2D) g;

            // Degradado horizontal arcoíris
            float[] fractions = { 0f, 0.16f, 0.33f, 0.49f, 0.66f, 0.82f, 1f };
            Color[] colors = {
                    new Color( r: 255,  g: 0,  b: 0), // Rojo
                    new Color( r: 255,  g: 165,  b: 0), // Naranja
                    new Color( r: 255,  g: 255,  b: 0), // Amarillo
                    new Color( r: 0,  g: 128,  b: 0), // Verde
                    new Color( r: 0,  g: 0,  b: 255), // Azul
                    new Color( r: 75,  g: 0,  b: 130), // Índigo
                    new Color( r: 238,  g: 130,  b: 238) // Violeta
            };

            LinearGradientPaint gradient = new LinearGradientPaint(
                    startX: 0,  startY: 0, getWidth(),  endY: 0, fractions, colors);

            g2d.setPaint(gradient);
            g2d.fillRect( x: 0,  y: 0, getWidth(), getHeight());
        }
    };
```

Aquí con esos mismos colores generamos el header.

```
public EmocionesP() {
    setLayout(new BorderLayout());
    setOpaque(false);
    add(crearHeaderArcoiris(), BorderLayout.NORTH);
    String[] nombres = { "alegria", "tristeza", "enojo", "ansiedad", "calma" };
    Map<String, Color> coloresEmociones = Map.of(
            k1: "alegria", new Color( r: 30,  g: 144,  b: 255),
            k2: "tristeza", new Color( r: 70,  g: 40,  b: 100),
            k3: "enojo", new Color( r: 220,  g: 20,  b: 60),
            k4: "ansiedad", new Color( r: 204,  g: 85,  b: 0),
            k5: "calma", new Color( r: 50,  g: 205,  b: 50));

    barras = new HashMap<>();
    emociones = new HashMap<>();
    executor = Executors.newFixedThreadPool(nombres.length);
    lockEmociones = new ReentrantLock();
    emocionesCompatibles = lockEmociones.newCondition();

    JPanel panelBarras = new JPanel(new GridLayout(nombres.length,  cols: 1,  hgap: 10,  vgap: 10));
    panelBarras.setOpaque(false);
```

Posteriormente inicializamos todo el panel, agregamos gráficamente las barras y se crean y lanzan todas las instancias de emoción mediante el siguiente for.

```
for (String nombre : nombres) {
    JProgressBar barra = new JProgressBar(SwingConstants.VERTICAL, min: 0, max: 100) {
        @Override
        protected void paintComponent(Graphics g) {
            Graphics2D g2d = (Graphics2D) g.create();

            g2d.setColor(new Color( r: 240, g: 240, b: 240, a: 220));
            g2d.fillRoundRect( x: 0, y: 0, getWidth(), getHeight(), arcWidth: 15, arcHeight: 15);

            int altura = (int) (getHeight() * (getValue() / 100.0));
            g2d.setColor(getForeground());
            g2d.fillRoundRect( x: 3, y: getHeight() - altura, width: getWidth() - 6, altura, arcWidth: 10, arcHei

            g2d.setColor(getForeground().darker().darker());
            g2d.drawRoundRect( x: 0, y: 0, width: getWidth() - 1, height: getHeight() - 1, arcWidth: 15, arcHeight:

            g2d.setColor(Color.BLACK);
            g2d.setFont(new Font( name: "Arial", Font.BOLD, size: 12));
            FontMetrics fm = g2d.getFontMetrics();
            String texto = getString();
            int x = (getWidth() - fm.stringWidth(texto)) / 2;
            int y = (getHeight() + fm.getAscent()) / 2 - 2;
            g2d.drawString(texto, x, y);

            g2d.dispose();
        }
    };
```

Finalmente configuramos y registramos la barra de progreso personalizada y lanzamos el hilo correspondiente para animarla.

```
barra.setStringPainted(true);
barra.setForeground(coloresEmociones.get(nombre));
barra.setString(nombre);
barra.setBorder(BorderFactory.createEmptyBorder( top: 5, left: 10, bottom: 5, right: 10));
barras.put(nombre, barra);
panelBarras.add(barra);

Emocion emocion = new Emocion(nombre, barra, lockEmociones, emocionesCompatibles,
        grupo1, grupo2, panel: this);
emociones.put(nombre, emocion);
executor.submit(emocion);
```

Mediante los atributos de cada emoción se le asigna a cada barra las características correspondientes y finalmente creamos una instancia de moción que contiene la lógica de como cambia la emoción, consulta su compatibilidad con el resto de emociones ejecutándose y actualiza su barra con los nuevos valores.

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

    Graphics2D g2d = (Graphics2D) g.create();
    int width = getWidth();
    int height = getHeight();
    int bandHeight = height / coloresArcoiris.length;

    for (int i = 0; i < coloresArcoiris.length; i++) {
        g2d.setColor(coloresArcoiris[i]);
        g2d.fillRect( x: 0,  y: i * bandHeight, width, bandHeight);

        if (i < coloresArcoiris.length - 1) {
            GradientPaint gp = new GradientPaint(
                    x1: 0,  y1: i * bandHeight + (bandHeight / 2), coloresArcoiris[i],
                    x2: 0,  y2: (i + 1) * bandHeight, coloresArcoiris[i + 1]);
            g2d.setPaint(gp);
            g2d.fillRect( x: 0,  y: i * bandHeight + (bandHeight / 2), width,  height: bandHeight / 2);
        }
    }
    g2d.dispose();
```

El siguiente método es un override del método paintComponent original y fue el utilizado para generar el degradado en nuestro header lo que hace es dividir el fondo en franjas y generar un degradado de un color a otro.

```java
public Set<String> getEmocionesActivas() {
    lockEmociones.lock();
    try {
        return emociones.keySet().stream()
                .filter(this::estaEmocionActiva)
                .collect(Collectors.toSet());
    } finally {
        lockEmociones.unlock();
    }
}
```

El siguiente método nos retorna el set de emociones que se encuentran activas en el momento, este utiliza un Lock para evitar problemas de concurrencia y seguir las incompatibilidades entre emociones o la presencia de 3 o más emociones activas a la vez.

```java
public boolean sonCompatibles(Set<String> emocionesAVerificar) {
    // Verificar incompatibilidades dentro del grupo1
    long countGrupo1 = emocionesAVerificar.stream().filter(grupo1::contains).count();
    if (countGrupo1 > 1)
        return false;

    // Verificar incompatibilidades dentro del grupo2
    long countGrupo2 = emocionesAVerificar.stream().filter(grupo2::contains).count();
    if (countGrupo2 > 1)
        return false;

    // Verificar reglas cruzadas
    if (emocionesAVerificar.contains("alegria") && emocionesAVerificar.contains("ansiedad"))
        return false;
    if (emocionesAVerificar.contains("enojo") && emocionesAVerificar.contains("ansiedad"))
        return false;
    if (emocionesAVerificar.contains("enojo") && emocionesAVerificar.contains("calma"))
        return false;
    if (emocionesAVerificar.contains("tristeza") && emocionesAVerificar.contains("calma"))
        return false;

    return true;
}
```

El método sonCompatibles realiza dicha comparación verificando que las emociones pertenezcan a el mismo grupo, este grupo de emociones es el mismo definido al principio de emocionesP definiendo a que emociones tiene sentido que una persona presente al mismo tiempo.

```java
public void procesarEntrada(String texto) {
    texto = texto.toLowerCase();
    lockEmociones.lock();
    try {
        Set<String> emocionesAActivar = new HashSet<>();

        // Detectar todas las emociones a activar
        if (texto.contains("feliz") || texto.contains("alegria"))
            emocionesAActivar.add("alegria");
        if (texto.contains("triste") || texto.contains("llorando"))
            emocionesAActivar.add("tristeza");
        if (texto.contains("enojo") || texto.contains("molesto") || texto.contains("enojado"))
            emocionesAActivar.add("enojo");
        if (texto.contains("ansioso") || texto.contains("estresado"))
            emocionesAActivar.add("ansiedad");
        if (texto.contains("calmado") || texto.contains("tranquilo"))
            emocionesAActivar.add("calma");
```

Procesar entrada es el método mencionado al inicio encargado de manejar la entrada del usuario, lo primero que hace es bloquear las emociones.

Este compara distintas palabras asignadas por nosotros y que se referirían a ciertas emociones especificas, después de catalogar la palabra propuesta por el usuario se realiza los siguiente:

```
// Verificar compatibilidad
if (!this.sonCompatibles(emocionesAActivar)) {
    mostrarErrorIncompatibilidad();
    return;
}

// Desactivar incompatibles y activar las nuevas
for (String emocion : emocionesAActivar) {
    desactivarIncompatibles(emocion);
    emociones.get(emocion).activar();
}

// Actualizar barras
for (String emocion : emociones.keySet()) {
    barras.get(emocion).setValue(estaEmocionActiva(emocion) ? 100 : 0);
}
```

Se verifica compatibilidad con el método mencionado anteriormente, se desactivan aquellas incompatibles y finalmente se va actualizando el estado de la barra.

```
    } finally {
        lockEmociones.unlock();
    }
}
```

Echo esto, se desbloquean las emociones.

```
private void desactivarIncompatibles(String emocionActivada) {
    // Desactivar todas las emociones incompatibles
    for (String emo : emociones.keySet()) {
        if (!emo.equals(emocionActivada)) {
            Set<String> conjunto = Set.of(emocionActivada, emo);
            if (!sonCompatibles(conjunto)) {
                emociones.get(emo).desactivarForzado();
            }
        }
    }
}
```

El siguiente método desactivarIncompatibles hace uso del método desactivarForazado de Emocion para parar la ejecución de una de las emociones cuando se encuentra con alguna incompatible.

```
public void mostrarErrorIncompatibilidad() {
    SwingUtilities.invokeLater(() -> JOptionPane.showMessageDialog( parentComponent: this,
            message: "¿Estas seguro de sentirte así?",  title: "Error",
            JOptionPane.ERROR_MESSAGE));
}
```

Para esta misma funcionalidad se activa mostrar error cuando alguien no introduce una serie de emociones coherentes.

```
public boolean estaEmocionActiva(String nombreEmocion) {
    lockEmociones.lock();
    try {
        Emocion emocion = emociones.get(nombreEmocion);
        return emocion != null && emocion.activa;
    } finally {
        lockEmociones.unlock();
    }
}
```

Emoción activa retorna un booleano confirmando si la emoción esta o no activa.

```
public void liberarPermiso() {
    semaforoGlobal.release();
}
```

Por ultimo con respecto a los metodos de Emocion P se encuentra liberarPermiso que se encarga de liberar el semaforo global, con esto podemos pasar a la ultima de las clases que definen a nuestro proyecto: Emocion.

```
public class Emocion implements Runnable {
    14 usages
    public volatile boolean activa;
    8 usages
    private volatile boolean forzarDesactivar;
    10 usages
    private int intensidad;
    4 usages
    private final String nombre;
    3 usages
    private final JProgressBar barra;
    11 usages
    private final Lock lock;
    4 usages
    private final Condition emocionesCompatibles;
    1 usage
    private final Set<String> grupo1;
    1 usage
    private final Set<String> grupo2;
    7 usages
    private final EmocionesP panel;
```

Las instancias de Emocion implementan a Runnable puesto que se espera que sean ejecutadas por un hilo y poseen los atributos enlistados arriba.

```
public Emocion(String nombre, JProgressBar barra, Lock lock, Condition emocionesCompatibles,
        Set<String> grupo1, Set<String> grupo2, EmocionesP panel) {
    this.nombre = nombre;
    this.barra = barra;
    this.lock = lock;
    this.emocionesCompatibles = emocionesCompatibles;
    this.grupo1 = grupo1;
    this.grupo2 = grupo2;
    this.panel = panel;
    this.activa = false;
    this.forzarDesactivar = false;
    this.intensidad = 0;
}
```

El constructor de estos objetos esta definido de esta forma, que como pudimos ver son todos aquellos atributos que desde EmocionP utilizabamos para definir un nuevo objeto Emocion para cada una de las emociones que escogimos.

```java
public void activar() {
    lock.lock();
    try {
        if (!activa) {
            activa = true;
            forzarDesactivar = false;
            emocionesCompatibles.signalAll();
        }
    } finally {
        lock.unlock();
    }
}
```

El primero de los metodos de esta clase es activar que se encarga de activar la emocion, primero bloqueamos la emocion para asegurarnos de que unicamente un hilo se encargue de actualizarla protegiendo activa y forzarDesactivar, luego despetamos a los hilos que verifican emocionesCompatibles echo esto desbloqueamos la emocion.

```java
public void desactivarForzado() {
    lock.lock();
    try {
        if (activa) {
            forzarDesactivar = true;
            emocionesCompatibles.signalAll();
        }
    } finally {
        lock.unlock();
    }
}
```

El metodo desactivarForzado tiene la funcion opuesta, igualmente bloquela la emocion, no cambia su estado ya sea que fuera activo o inactivo y llama a desactiarForzado y notifica al resto de emociones que se le desactivo.

```java
public void run() {
    while (!Thread.currentThread().isInterrupted()) {
        try {
            if (!lock.tryLock( time: 100, TimeUnit.MILLISECONDS)) {
                continue;
            }
            try {
                while (!esCompatible()) {
                    if (activa) {
                        activa = false;
                        panel.liberarPermiso();
                    }
                    emocionesCompatibles.await( time: 100, TimeUnit.MILLISECONDS);
                }
```

Pasamos entonces al metodo run, el condicional while del inicio indica que mientras no se le interrumpa el hilo seguira corriendo, el try intenta en esa vuelta del bucle acceder al hilo si no puede lo intenta en la siguiente iteración del bucle.

El try posterior a esto solo se activa si la emocion no es compatible a una de las demas corriendo de ser asi pone en false el atributo activa del hilo terminando con su ejecución.

```
    if (forzarDesactivar) {
        if (activa) {
            panel.liberarPermiso();
            activa = false;
        }
        forzarDesactivar = false;
        continue;
    }

    if (activa) {
        aumentarIntensidad();
    } else {
        disminuirIntensidad();
    }
} finally {
    lock.unlock();
}
```

Continuando ocn el metodo, si el atributo forzarDesactivar se hace true en alguna parte del bucle igualmente seteamos activar en false y regresamos forzar a false enotnces contiunamos, si el hilo se encuentra activo aumentamos la intensidad de la emocion, de no ser asi la disminuimos con el metodo correspondiente, por ultimo desbloqueamos la emocion.

```
private void aumentarIntensidad() throws InterruptedException {
    for (int i = intensidad; i <= 100 && !forzarDesactivar; i += 5) {
        intensidad = i;
        actualizarBarra();
        Thread.sleep( millis: 50);
    }

    if (!forzarDesactivar && intensidad >= 100) {
        lock.lock();
        try {
            activa = false;
            panel.liberarPermiso();
        } finally {
            lock.unlock();
        }
    }
}
```

El metodo aumentarIntensidad es el que se presenta en el bucle cuando la emocion este activa, este va aumentando el porcentaje de lo presente que esta la emocion hasta llegar al 100%.

```java
private void disminuirIntensidad() {
    if (intensidad > 0) {
        intensidad = Math.max(0, intensidad - 2);
        actualizarBarra();

        if (intensidad == 0) {
            lock.lock();
            try {
                if (activa) {
                    activa = false;
                    panel.liberarPermiso();
                }
            } finally {
                lock.unlock();
            }
        }
    }
}
```

Por el contrario, disminuir intensidad es la que se presenta cuando el hilo ha sido desactivado por, este va disminuyenddo la presencia de la emocion poco a poco, el funcionamiento mas rapido de su contraparte se debe a que las emociones usualmente se presentan de golpe ante nosotros al suceder algo y por el contrario la desaparación de esa emoción se da poco a poco hasta que pasamos a un estado neutral.

```java
private void actualizarBarra() {
    SwingUtilities.invokeLater(() -> {
        barra.setValue(intensidad);
        barra.setString(nombre + " (" + intensidad + "%)");
    });
}
```

Este es el metodo llamado dentro de ambos para actualizar en la interfaz el porcentaje de la barra.

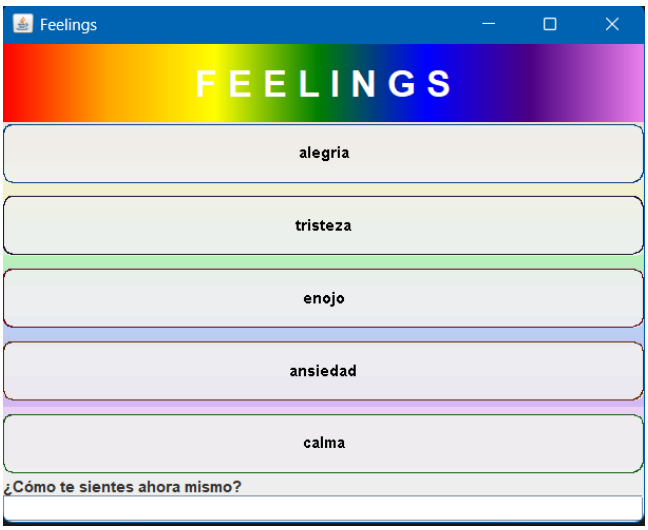
```java
private boolean esCompatible() {
    if (!activa && !forzarDesactivar) {
        return true;
    }

    Set<String> emocionesActivas = panel.getEmocionesActivas();
    emocionesActivas.remove(nombre);

    Set<String> conjuntoAVerificar = new HashSet<>(emocionesActivas);
    conjuntoAVerificar.add(nombre);

    return panel.sonCompatibles(conjuntoAVerificar);
}
```

Por ultimo es compatible que verifica que emociones se encuentran activas al momento de un conjunto de emociones a verificar según las introducidas por el usuario al momento.

Estas son las clases, metodos y objetos que definen a nuestro proyecto una ves explicados pasamos a una serie de pruebas.

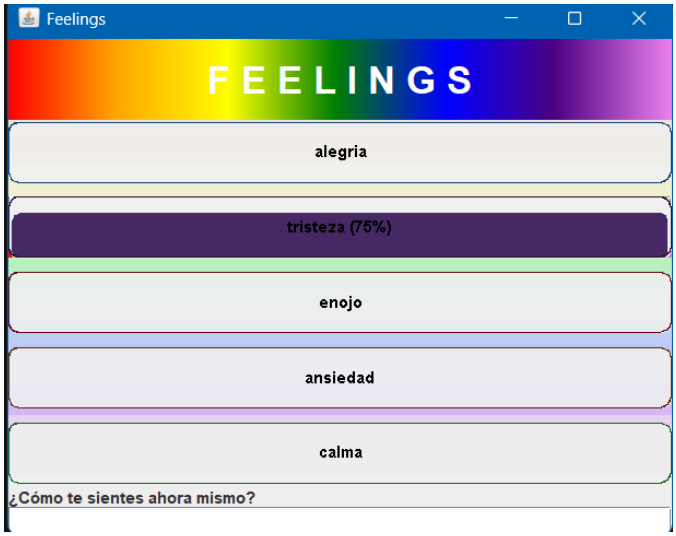
## Pruebas

Las pruebas de nuestro codigo se realizaran en la IDE inteliJ IDEA, se recomienda el uso de esta IDE para cualquier prueba.

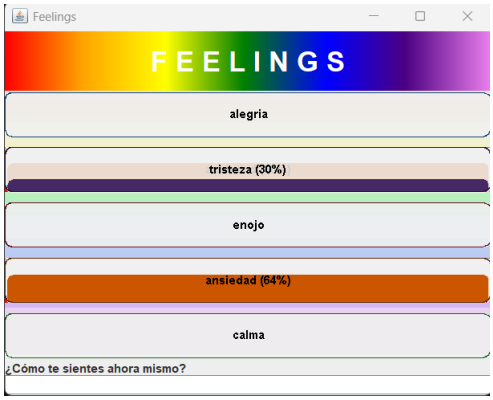



Correcta ejecucion del codigo generando todas las partes de la interfaz de la manera correcta.


Pruebas para el input de usuario “triste”.




Correcto llenado de la barra de tristeza y posterior vaciado.




Para el input triste y ansioso correcto llenado de ambas emociones, posteriormente según el orden en el que fueron introducidas se vacia una antes que otra.


Pruebas para tres inputs de emociones a la vez con las emociones del grupo 1, correcto llenado y vaciado de las emociones.

## Instrucciones de uso

Seleccionar cualquiera de las siguientes palabras para desatar el hilo de la emoción correspondiente.

| Emoción | Palabras clave |
|---|---|
| alegría | feliz, alegria |
| tristeza | triste, llorando |
| enojo | enojo, molesto, enojado |
| ansiedad | ansioso, estresado |
| calma | calmado, tranquilo |