OPCION 2

****Introducción:****
En este tutorial, vamos a crear una aplicación en Java que simula una carrera de caballos. Utilizaremos Java Swing para crear una interfaz gráfica de usuario con barras de progreso para los caballos y animaciones de sprites. La carrera comienza cuando el usuario hace clic en un botón y el primer caballo en llegar al 100% gana.

****Paso 1: Configuración Inicial:****
Comencemos creando la ventana principal y configurando la estructura básica de la aplicación.

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

****Paso 2: Creación de la Ventana y Componentes:****
Dentro de nuestra clase principal `CarreraCaballos2`, creamos la ventana y los componentes necesarios, como las barras de progreso, las etiquetas de los caballos y el botón de inicio.

```java
public class CarreraCaballos2 extends JFrame {
    private JProgressBar[] progressBars;
    private JLabel[] caballoLabels;
    private JButton btnIniciar;
    private Timer animationTimer;
    // Otras variables de control...
```

****Paso 3: Constructor:****
En el constructor de la clase, configuramos la apariencia inicial de la ventana y los componentes.

```java
public CarreraCaballos2() {
    setTitle("Carrera de Caballos");
    setSize(600, 400);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLocationRelativeTo(null);
```

Paso 4: Configuración de Componentes:
Continuamos configurando los componentes, como las barras de progreso y las etiquetas de los caballos, asignando imágenes a cada caballo.

```java
    progressBars = new JProgressBar[4];
    caballoLabels = new JLabel[4];
    caballos = new Thread[4];
    caballoImages = new int[4]; // Índices de imágenes de caballos
    String[] imagePaths = {"horse1.png", "horse2.png", "horse3.png", "horse4.png"};

    for (int i = 0; i < 4; i++) {
        progressBars[i] = new JProgressBar(0, 100);
        progressBars[i].setValue(0);
        progressBars[i].setStringPainted(true);
        caballoLabels[i] = new JLabel(createImageIcon(imagePaths[i]));
        caballoImages[i] = i; // Asigna una imagen a cada caballo
    }
```

Paso 5: Botón de Inicio y Animación:
Creamos un botón de inicio que permite al usuario comenzar la carrera y configuramos una animación que alterna entre imágenes de caballos para dar la ilusión de movimiento.

```java
    btnIniciar = new JButton("Iniciar Carrera");
    btnIniciar.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (!carreraEnCurso) {
                iniciarCarrera();
            }
        }
    });

    animationTimer = new Timer(150, new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (carreraEnCurso) {
                alternarAnimacion();
            }
        }
    });
```

Paso 6: Diseño de la Interfaz:
Creamos un diseño de interfaz utilizando un panel con una cuadrícula para organizar los componentes. También aplicamos márgenes para mejorar el aspecto.

```java
    JPanel panel = new JPanel(new GridLayout(5, 1));
    for (int i = 0; i < 4; i++) {
        panel.add(new JLabel("Caballo " + (i + 1)));
        panel.add(caballoLabels[i]);
        panel.add(progressBars[i]);
        JPanel horsePanel = new JPanel();
        horsePanel.setLayout(new FlowLayout(FlowLayout.CENTER));
        horsePanel.add(caballoLabels[i]);
        panel.add(horsePanel);
    }
    panel.add(btnIniciar);
    panel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

    add(panel);
```

Paso 7: Lógica de Carrera:
Implementamos la lógica de la carrera en la clase interna `Caballo`. Cada caballo avanza de manera aleatoria y se detiene cuando alcanza el 100%. Si un caballo gana, se muestra un mensaje y la carrera se reinicia.

```java
class Caballo implements Runnable {
    private int caballoId;

    public Caballo(int caballoId) {
        this.caballoId = caballoId;
    }

    @Override
    public void run() {
        Random rand = new Random();
        while (progressBars[caballoId].getValue() < 100 && ganador == -1) {
            int avance = rand.nextInt(15) + 1;
            int newValue = progressBars[caballoId].getValue() + avance;
            if (newValue >= 100) {
                newValue = 100;
                ganador = caballoId;
                animationTimer.stop();
                mostrarGanador();
            }
            progressBars[caballoId].setValue(newValue);
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                return;
            }
        }
    }
}
```

Paso 8: Mostrar al Ganador y Reiniciar:
Cuando se encuentra un ganador, se muestra un mensaje y se reinicia la carrera.

```java
private void mostrarGanador() {
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            JOptionPane.showMessageDialog(null, "¡Caballo " + (ganador + 1) + " ha ganado!");
            reiniciarCarrera();
        }
    });
}

private void reiniciarCarrera() {
    carreraEnCurso = false;
    ganador = -1;
    // Detener los hilos y restaurar las imágenes de los caballos...
}
```

Paso 9: Métodos Auxiliares:
Implementamos métodos auxiliares, como `alternarAnimacion` para cambiar la animación de los caballos y `createImageIcon` para cargar imágenes.

Paso 10: Ejecución Principal:
En el método `main`, creamos una instancia de `CarreraCaballos2` y la hacemos visible.

```java
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            CarreraCaballos2 carrera = new CarreraCaballos2();
            carrera.setVisible(true);
        }
    });
}
```

Conclusión:
Hemos creado una aplicación de carrera de caballos en Java Swing con animaciones y una interfaz de usuario amigable. El usuario puede iniciar la carrera, y el primer caballo en llegar al 100