MANUAL DE PROGRAMADOR

```
private void btnIniciarActionPerformed(java.awt.event.ActionEvent evt) {

    if (esEntero(txtInventarioTiempo.getText()) == false || esEntero(txtInventarioCostos.getText()) == false
            || esEntero(txtProduccionTiempo.getText()) == false || esEntero(txtProduccionCostos.getText()) == false
            || esEntero(txtEmpaquetadoTiempo.getText()) == false || esEntero(txtEmpaquetadoCostos.getText()) == false
            || esEntero(txtSalidaTiempo.getText()) == false || esEntero(txtSalidaCostos.getText()) == false) {
        JOptionPane.showMessageDialog(null, "Hay valores que no son enteros o campos vacios.", "Error", JOptionPane.ERROR_MESSAGE);
    } else {
        monkey[0][0] = Integer.parseInt(txtInventarioTiempo.getText());
        monkey[0][1] = Integer.parseInt(txtInventarioCostos.getText());
        monkey[1][0] = Integer.parseInt(txtProduccionTiempo.getText());
        monkey[1][1] = Integer.parseInt(txtProduccionCostos.getText());
        monkey[2][0] = Integer.parseInt(txtEmpaquetadoTiempo.getText());
        monkey[2][1] = Integer.parseInt(txtEmpaquetadoCostos.getText());
        monkey[3][0] = Integer.parseInt(txtSalidaTiempo.getText());
        monkey[3][1] = Integer.parseInt(txtSalidaCostos.getText());
        JOptionPane.showMessageDialog(null, "Valores guardados", "Exito", JOptionPane.INFORMATION_MESSAGE);
        Simulacion simulacion = new Simulacion(monkey);
        simulacion.setVisible(true);
    }
```

La función btnIniciarActionPerformed es un controlador de eventos que se ejecuta cuando el usuario hace clic en el botón "Iniciar simulación" de la interfaz gráfica. El código dentro de esta función realiza las siguientes acciones:

1. Comprueba si los valores en los campos de texto (txtInventarioTiempo, txtInventarioCostos, txtProduccionTiempo, txtProduccionCostos, txtEmpaquetadoTiempo, txtEmpaquetadoCostos, txtSalidaTiempo y txtSalidaCostos) son números enteros válidos. Para hacer esto, utiliza la función esEntero(), que devuelve true si el valor es un entero y false en caso contrario. Si alguno de los campos de texto no contiene un valor entero válido, se muestra un cuadro de diálogo de error informando al usuario que hay valores incorrectos o campos vacíos.
2. Si todos los campos contienen números enteros válidos, se almacenan estos valores en la matriz bidimensional monkey. La matriz monkey tiene 4 filas y 2 columnas, donde cada fila representa un sector (Inventario, Producción, Empaquetado y Salida) y las columnas representan el tiempo y el costo de ese sector, respectivamente. Los valores se extraen de los campos de texto utilizando Integer.parseInt() para convertirlos de cadenas de caracteres (String) a números enteros (int).
3. Después de almacenar los valores en la matriz monkey, se muestra un cuadro de diálogo de información informando al usuario que los valores se han guardado correctamente.
4. A continuación, se crea una instancia de la clase Simulacion, pasando la matriz monkey como argumento al constructor. Esto permite que la clase Simulacion tenga acceso a los datos almacenados en monkey.
5. Finalmente, se establece la instancia de la clase Simulacion como visible, lo que muestra la ventana de simulación al usuario.

En resumen, este controlador de eventos verifica si los valores de entrada son válidos, almacena estos valores en una matriz y luego inicia la ventana de simulación con los datos almacenados en la matriz monkey.

```
private void lblFlechaInicioMouseClicked(java.awt.event.MouseEvent evt) {
    int contadorInicio = Integer.parseInt(lblContadorInicio.getText());
    int contadorFinal = Integer.parseInt(lblContadorFinal.getText());
    if (contadorInicio == 0) {
        JOptionPane.showMessageDialog(null, "YA NO HAY MÁS PELOTAS", "Información", JOptionPane.INFORMATION_MESSAGE);
    } else {
        hiloPelota = new Pelota(lblPelota, lblInventario_Simulacion, lblSalida_Simulacion, panelSimulacion, lblPrueba,
                lblContadorInventario, lblContadorProduccion, lblContadorEmpaquetado, lblContadorSalida,
                lblContadorInicio, lblContadorFinal, lblProduccion_Simulacion, lblEmpaquetado_Simulacion, monkey,
        btnRegresarMenu_Simulacion, btnReporte_Simulacion);
        hiloPelota.iniciarPelota();
    }
    if (contadorFinal == 30) {
        JOptionPane.showMessageDialog(null, "SIMULACIÓN FINALIZADA", "Información", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

La función lblFlechaInicioMouseClicked es un controlador de eventos que se ejecuta cuando el usuario hace clic en el componente lblFlechaInicio en la interfaz gráfica. El código dentro de esta función realiza las siguientes acciones:

1. Obtiene el valor numérico del contador de inicio (lblContadorInicio) y del contador final (lblContadorFinal) a partir de sus respectivos componentes de texto (JLabel), convirtiendo las cadenas de caracteres (String) a números enteros (int) usando Integer.parseInt().
2. Verifica si el contador de inicio (contadorInicio) es igual a 0, lo que indica que no hay más pelotas disponibles para la simulación. Si es el caso, muestra un cuadro de diálogo de información al usuario indicando que no hay más pelotas.
3. Si aún hay pelotas disponibles, crea una nueva instancia de la clase Pelota, pasando todos los componentes y valores necesarios como argumentos al constructor. Luego, se llama al método iniciarPelota() de la instancia de Pelota creada. Este método inicia el movimiento de la pelota en la simulación.
4. Después, verifica si el contador final (contadorFinal) es igual a 30, lo que indica que la simulación ha finalizado. Si es el caso, muestra un cuadro de diálogo de información al usuario informando que la simulación ha finalizado.

En resumen, este controlador de eventos verifica si hay pelotas disponibles y si la simulación aún no ha finalizado antes de crear una nueva instancia de la clase Pelota y comenzar el movimiento de la pelota en la simulación. También muestra mensajes de información al usuario según las condiciones mencionadas.

```
private void btnReporte_SimulacionActionPerformed(java.awt.event.ActionEvent evt) {
    String nombreCompleto = "Luis Rodolfo Porras García";
    String carnet = "201901462";

    int costoInventario = monkey[0][1], costoProduccion = monkey[1][1], costoEmpaquetado = monkey[2][1], costoSalida = monkey[3][1];
    int tiempoInventario = monkey[0][0], tiempoProduccion = monkey[1][0], tiempoEmpaquetado = monkey[2][0], tiempoSalida = monkey[3][0];

    int costoTotalInventario = costoInventario * tiempoInventario * 30;
    int costoTotalProduccion = costoProduccion * tiempoProduccion * 30;
    int costoTotalEmpaquetado = costoEmpaquetado * tiempoEmpaquetado * 30;
    int costoTotalSalida = costoSalida * tiempoSalida * 30;
    int costoTotal = costoTotalInventario + costoTotalProduccion + costoTotalEmpaquetado + costoTotalSalida;

    String htmlContenido = "<html>\n"
            + "<head><title>Reporte</title></head>\n"
            + "<body>\n"
            + "<h1>Reporte de costos</h1>\n"
            + "<table border='1'>\n"
            + "<tr><th>Sector</th><th>Costo</th></tr>\n"
            + "<tr><td>Inventario</td><td>" + costoTotalInventario + "</td></tr>\n"
            + "<tr><td>Producción</td><td>" + costoTotalProduccion + "</td></tr>\n"
            + "<tr><td>Empaquetado</td><td>" + costoTotalEmpaquetado + "</td></tr>\n"
            + "<tr><td>Salida</td><td>" + costoTotalSalida + "</td></tr>\n"
            + "<tr><td><b>Total</b></td><td><b>" + costoTotal + "</b></td></tr>\n"
            + "</table>\n"
            + "<p>Nombre completo: " + nombreCompleto + "</p>\n"
            + "<p>Carnet: " + carnet + "</p>\n"
            + "</body>\n"
            + "</html>";
```

```
    try {
        File archivo = new File("reporte.html");
        archivo.createNewFile();
        try (FileWriter escritor = new FileWriter(archivo)) {
            escritor.write(htmlContenido);
        }
        Desktop abridor = Desktop.getDesktop();
        abridor.open(archivo);

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

La función btnReporte_SimulacionActionPerformed es un controlador de eventos que se ejecuta cuando el usuario hace clic en el botón btnReporte_Simulacion en la interfaz gráfica. El código dentro de esta función realiza las siguientes acciones:

1. Define variables con los valores de nombre completo y carnet del alumno.
2. Obtiene los costos y tiempos de procesamiento de cada sector (Inventario, Producción, Empaquetado y Salida) desde la matriz monkey.
3. Calcula el costo total de cada sector usando la fórmula Costo de sector = Costo por producto del sector * tiempo de procesamiento de sector * 30. Luego, calcula el costo total sumando los costos totales de cada sector.
4. Crea una cadena de caracteres (String) llamada htmlContenido que contiene el contenido del archivo HTML que se generará. Esta cadena contiene etiquetas HTML para dar formato a la tabla con los costos de cada sector y mostrar los datos del alumno (nombre completo y carnet).
5. Dentro de un bloque try, realiza las siguientes acciones:
   a. Crea un objeto File llamado archivo que representa un archivo llamado "reporte.html".
   b. Llama al método createNewFile() en el objeto archivo para crear el archivo "reporte.html" en el sistema de archivos.
   c. Crea un objeto FileWriter llamado escritor en un bloque try , que garantiza que el FileWriter se cierre automáticamente al final del bloque.

```java
package Codigo;

import javax.swing.JLabel;

public class Tiempo extends Thread {

    private JLabel lblTiempo;
    int min = 00, seg = 00;

    public Tiempo(JLabel lblTiempo) { // Constructor de la clase Tiempo
        this.lblTiempo = lblTiempo;
    }

    public void run() {
        while (true) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println("Hilo interrumpido");
                break;
            }

            seg++;
            if (seg >= 60) {
                seg = 0;
                min++;
            }
            lblTiempo.setText(String.format("%02d:%02d", min, seg));
        }
    }
}
```

La clase Tiempo es una subclase de Thread, lo que permite ejecutar un contador de tiempo en un hilo separado de la interfaz gráfica de usuario. Aquí está el desglose de las líneas de código en la clase:

1. Se define la clase Tiempo que extiende de la clase Thread.
2. Se declara una variable de instancia lblTiempo del tipo JLabel y dos variables enteras min y seg inicializadas en 0.
3. Se define el constructor de la clase Tiempo, que recibe un objeto JLabel como argumento y asigna este objeto a la variable de instancia lblTiempo.
4. Se sobrescribe el método run() de la clase Thread. Este método se ejecutará cuando se inicie el hilo Tiempo.
5. Dentro del método run(), se crea un bucle while (true) que se ejecutará indefinidamente.
6. Dentro del bucle, se utiliza Thread.sleep(1000) para poner en pausa la ejecución del hilo durante 1000 milisegundos (1 segundo). Esto simula el avance del tiempo en el contador.
7. Si la llamada a Thread.sleep(1000) se interrumpe, se captura la excepción InterruptedException, se imprime un mensaje en la consola y se termina el bucle while con la instrucción break.
8. Después de cada segundo, se incrementa la variable seg en 1.
9. Si la variable seg alcanza o supera 60, se reinicia a 0 y se incrementa la variable min en 1. Esto permite llevar el conteo de los minutos y segundos que han pasado.
10. Se actualiza el texto del JLabel lblTiempo con el tiempo transcurrido en minutos y segundos usando String.format() para dar formato al tiempo con dos dígitos para los minutos y dos dígitos para los segundos.

Cuando se instancia y se inicia un objeto Tiempo, este hilo actualizará el texto del JLabel proporcionado con un contador de tiempo en minutos y segundos que avanza cada segundo.

```java
public void aparecerPelota() {
    lblPelota = new JLabel();
    lblPelota.setBounds(x, y, 24, 24);
    // Direccipon computadora trabajo
    //ImageIcon icon = new ImageIcon("C:/Users/sistemas2/Documents/NetBeansProjects/Practica2/src/Imagenes/Feliz.png");

    //Direccion computadora casa
    ImageIcon icon = new ImageIcon("C:/Users/Luis Porras/Documents/NetBeansProjects/Practica2/src/Imagenes/Feliz.png");
    if (icon.getImageLoadStatus() == MediaTracker.COMPLETE) {
        lblPelota.setIcon(icon);
        lblPelota.setOpaque(false);
        panel.add(lblPelota);
        panel.setComponentZOrder(lblPelota, 0);
    } else {
        System.out.println("Error al cargar la imagen");
    }
}
```

El método aparecerPelota() se encarga de crear y mostrar un objeto JLabel llamado lblPelota, que representa una pelota en la interfaz gráfica. Aquí está el desglose de las líneas de código en el método:

1. Se crea un nuevo objeto JLabel y se asigna a la variable lblPelota.
2. Se establece la posición y el tamaño de lblPelota usando el método setBounds(). Las coordenadas x y y, así como las dimensiones 24x24, se pasan como argumentos.
3. Se crea un objeto ImageIcon llamado icon que carga una imagen desde la ruta especificada en el constructor. La ruta de la imagen puede variar según la ubicación del archivo en la computadora que se utiliza.
4. Se verifica si la imagen se ha cargado correctamente mediante el método getImageLoadStatus(). Si la imagen se ha cargado con éxito, su estado será igual a MediaTracker.COMPLETE.
5. Si la imagen se carga correctamente, se configura el ícono de lblPelota con el objeto icon usando el método setIcon().
6. Se establece la opacidad de lblPelota en falso usando el método setOpaque(false), lo que permite que el fondo del JLabel sea transparente.
7. Se añade lblPelota al objeto panel (presumiblemente un JPanel) usando el método add().
8. Se establece el orden Z del componente lblPelota en el panel utilizando el método setComponentZOrder(). El valor 0 se pasa como argumento, lo que indica que lblPelota debe colocarse en la capa superior del panel.
9. Si la imagen no se carga correctamente, se imprime un mensaje de error en la consola utilizando System.out.println().

En resumen, este método crea una instancia de JLabel que representa una pelota con una imagen y la coloca en un panel de la interfaz gráfica de usuario.

```
private void actualizarContador(JLabel label, int valor) {
    int valorActual = Integer.parseInt(label.getText());
    valorActual += valor;
    label.setText(String.valueOf(valorActual));
}
```

El método actualizarContador() se encarga de actualizar el valor numérico mostrado en un objeto JLabel y modificarlo según el valor entero proporcionado como argumento. Aquí está el desglose de las líneas de código en el método:

1. Se recibe como argumentos un objeto JLabel llamado label y un entero llamado valor.
2. Se obtiene el texto del JLabel utilizando el método getText() y se convierte en un entero utilizando Integer.parseInt(). Este entero se asigna a la variable valorActual.
3. Se suma el valor del argumento valor a valorActual.
4. Se convierte el valor actualizado de valorActual en una cadena de texto utilizando String.valueOf() y se establece como el nuevo texto del objeto label con el método setText().

En resumen, este método actualiza el valor numérico mostrado en un JLabel sumándole el valor entero proporcionado como argumento y actualizando su texto para reflejar el nuevo valor.

```
Color colorInventario = lblInventario_Simulacion.getBackground();
Color colorProduccion = lblProduccion_Simulacion.getBackground();
Color colorEmpaquetado = lblEmpaquetado_Simulacion.getBackground();
Color colorSalida = lblSalida_Simulacion.getBackground();
Color colorPanel = panel.getBackground();

while (true) {
    if (moviendoArriba) {
        y -= 3;
        if (y <= 540 && !disminuirInicio) {
            actualizarContador(lblContadorInicio, -1);
            disminuirInicio = true;
        }
        // CONDICIONAL PARA AUMENTAR EL CONTADOR CUANDO LA PELOTA LLEGUE AL "y=459" de lblInventario_Simulacion

        //  -----   -----   -----   -----
        //  |   |   |   |   |   |   |   |
        //  |   |   |   |   |   |   |   |
        //  |   |   |   |   |   |   |   |
        //  --.--   -----   -----   -----
        if (y <= 455 && !aumentarInventario) {
            actualizarContador(lblContadorInventario, 1);
            aumentarInventario = true;
            lblPelota.setBounds(x, y, 30, 30);
            Border newBorder = BorderFactory.createLineBorder(colorEmpaquetado, 3);
            lblPelota.setBorder(newBorder);
            lblPelota.setOpaque(true);
            lblPelota.setBackground(colorEmpaquetado);
        }
        if (x == 128 && y == 270) {
            Thread.sleep(monkey[0][0] * 1000);
        }
        if (y < mitadInicio) {
            moviendoArriba = false;
            moviendoDerecha = true;
        }
```

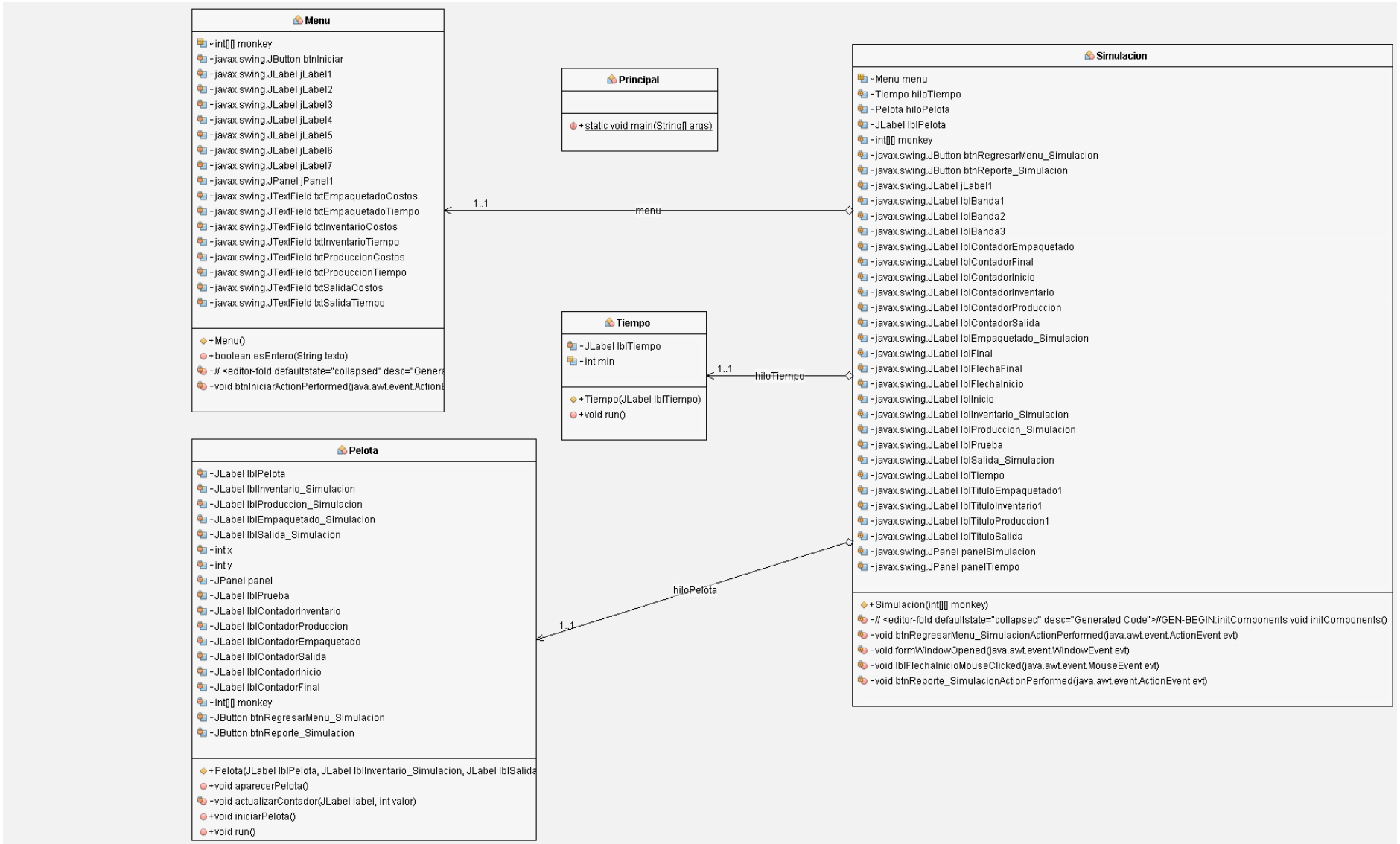
Este bloque de código forma parte del método run() dentro de una clase que extiende de Thread. Se encarga de mover una pelota (representada por un JLabel llamado lblPelota) hacia arriba en la interfaz gráfica y realizar acciones en ciertos puntos específicos.

1. Se inicializan varias variables booleanas para controlar si ciertas acciones han sido realizadas o no.
2. Se establecen variables booleanas (moviendoArriba, moviendoDerecha, moviendoAbajo) para controlar la dirección en la que se mueve la pelota.
3. Se calculan las posiciones mitadSalida y mitadInicio, que representan puntos específicos en los objetos lblSalida_Simulacion y lblInventario_Simulacion.
4. Se obtienen los colores de fondo de los objetos lblInventario_Simulacion, lblProduccion_Simulacion, lblEmpaquetado_Simulacion, lblSalida_Simulacion y panel.
5. Se inicia un bucle while (true) para mover continuamente la pelota.
6. Dentro del bucle, si moviendoArriba es verdadero, la posición **y** de la pelota disminuye en 3 unidades para moverla hacia arriba.
7. Cuando la posición **y** de la pelota es menor o igual a 540 y disminuirInicio es falso, se actualiza el contador lblContadorInicio y se establece disminuirInicio en verdadero.

8. Cuando la posición y de la pelota es menor o igual a 455 y aumentarInventario es falso, se actualiza el contador lblContadorInventario, se cambia el tamaño y el estilo del borde de la pelota, y se establece aumentarInventario en verdadero.
9. Cuando las coordenadas de la pelota coinciden con (128, 270), el hilo se detiene durante un tiempo específico, multiplicando el valor de monkey[0][0] por 1000 milisegundos.
10. Si la posición **y** de la pelota es menor que mitadInicio, se cambia la dirección de la pelota al establecer moviendoArriba en falso y moviendoDerecha en verdadero.

Este bloque de código controla el movimiento hacia arriba de la pelota en la interfaz gráfica y realiza acciones específicas en ciertos puntos durante su desplazamiento.

El código es similar para las demás áreas.


Menu
~ int[][] monkey
- javax.swing.JButton btnIniciar
- javax.swing.JLabel jLabel1
- javax.swing.JLabel jLabel2
- javax.swing.JLabel jLabel3
- javax.swing.JLabel jLabel4
- javax.swing.JLabel jLabel5
- javax.swing.JLabel jLabel6
- javax.swing.JLabel jLabel7
- javax.swing.JPanel jPanel1
- javax.swing.JTextField txtEmpaquetadoCostos
- javax.swing.JTextField txtEmpaquetadoTiempo
- javax.swing.JTextField txtInventarioCostos
- javax.swing.JTextField txtInventarioTiempo
- javax.swing.JTextField txtProduccionCostos
- javax.swing.JTextField txtProduccionTiempo
- javax.swing.JTextField txtSalidaCostos
- javax.swing.JTextField txtSalidaTiempo
+ Menu()
+ boolean esEntero(String texto)
- // <editor-fold defaultstate="collapsed" desc="Gener
- void btnIniciarActionPerformed(java.awt.event.Action
Principal
+ static void main(String[] args)
1..1
menu
Simulacion
~ Menu menu
- Tiempo hiloTiempo
- Pelota hiloPelota
- JLabel lblPelota
- int[][] monkey
- javax.swing.JButton btnRegresarMenu_Simulacion
- javax.swing.JButton btnReporte_Simulacion
- javax.swing.JLabel jLabel1
- javax.swing.JLabel lblBanda1
- javax.swing.JLabel lblBanda2
- javax.swing.JLabel lblBanda3
- javax.swing.JLabel lblContadorEmpaquetado
- javax.swing.JLabel lblContadorFinal
- javax.swing.JLabel lblContadorInicio
- javax.swing.JLabel lblContadorInventario
- javax.swing.JLabel lblContadorProduccion
- javax.swing.JLabel lblContadorSalida
- javax.swing.JLabel lblEmpaquetado_Simulacion
- javax.swing.JLabel lblFinal
- javax.swing.JLabel lblFlechaFinal
- javax.swing.JLabel lblFlechaInicio
- javax.swing.JLabel lblInicio
- javax.swing.JLabel lblInventario_Simulacion
- javax.swing.JLabel lblProduccion_Simulacion
- javax.swing.JLabel lblPrueba
- javax.swing.JLabel lblSalida_Simulacion
- javax.swing.JLabel lblTiempo
- javax.swing.JLabel lblTituloEmpaquetado1
- javax.swing.JLabel lblTituloInventario1
- javax.swing.JLabel lblTituloProduccion1
- javax.swing.JLabel lblTituloSalida
- javax.swing.JPanel panelSimulacion
- javax.swing.JPanel panelTiempo
+ Simulacion(int[][] monkey)
- // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initComponents()
- void btnRegresarMenu_SimulacionActionPerformed(java.awt.event.ActionEvent evt)
- void formWindowOpened(java.awt.event.WindowEvent evt)
- void lblFlechaInicioMouseClicked(java.awt.event.MouseEvent evt)
- void btnReporte_SimulacionActionPerformed(java.awt.event.ActionEvent evt)
Tiempo
- JLabel lblTiempo
~ int min
+ Tiempo(JLabel lblTiempo)
+ void run()
1..1
hiloTiempo
Pelota
- JLabel lblPelota
- JLabel lblInventario_Simulacion
- JLabel lblProduccion_Simulacion
- JLabel lblEmpaquetado_Simulacion
- JLabel lblSalida_Simulacion
- int x
- int y
- JPanel panel
- JLabel lblPrueba
- JLabel lblContadorInventario
- JLabel lblContadorProduccion
- JLabel lblContadorEmpaquetado
- JLabel lblContadorSalida
- JLabel lblContadorInicio
- JLabel lblContadorFinal
- int[][] monkey
- JButton btnRegresarMenu_Simulacion
- JButton btnReporte_Simulacion
+ Pelota(JLabel lblPelota, JLabel lblInventario_Simulacion, JLabel lblSalida
+ void aparecerPelota()
- void actualizarContador(JLabel label, int valor)
+ void iniciarPelota()
+ void run()
hiloPelota
1..1