MÓDULO 5: INTERFAZ GRÁFICA

Unión de la interfaz con el mundo

Esta guía es la continuación del reto 3 de este módulo. Aquí encontrará los pasos que siguen a lo que usted ya realizó.

ETAPA 1: PREPARACIÓN DEL ENTORNO DE DESARROLLO

Para preparar el entorno de desarrollo del reto siga la siguiente instrucción:

1. Abra en eclipse el proyecto en el que usted avanzó en el reto anterior (Reto: Agregue todos los elementos de interacción)

ETAPA 2: ASOCIACIÓN DE LAS ACCIONES DEL USUARIO CON EVENTOS DE LA APLICACIÓN

Antes de empezar el desarrollo de esta etapa, recordemos los 3 pasos que se explicaron en la presentación teórica para asociar una acción del usuario con un evento de la aplicación (ver vídeo: Acciones de usuario y eventos de la aplicación):

1. Dar un nombre al evento y asociarlo a un botón.
2. Atender el evento.
3. Decir que el pánel (al cual pertenece el botón) es el encargado de atender el evento.

1. El primer paso consiste entonces en darle nombre a todos los eventos posibles que se pueden generar como consecuencia de las acciones del usuario y asociarlos a los botones correspondientes. En cada panel que contiene botones, cree constantes que representen el nombre de cada evento y asocie este nombre a cada botón en el método constructor de la clase respectiva.

En la declaración de la clase PanelDatosPaciente:

```
public class PanelDatosPaciente  extends JPanel implements ActionListener
{

    /**
     * El comando para el boton de calcular la edad del paciente
     */
    private final static String CALCULAR_EDAD = "CALCULAR EDAD";
```

Figura 26. Declaración de la constante para el evento que se genera cuando el usuario presiona el botón "Calcular Edad" del panel de datos del paciente

En el método constructor de la clase PanelDatosPaciente:

```
/**
 * Se crea el botón para calcular la edad del paciente
 * y se le asocia el comando respectivo (la constante)
 */
butEdad = new JButton( "Calcular Edad" );
butEdad.setActionCommand( CALCULAR_EDAD );
```

Figura 27. Asociación del comando CALCULAR_EDAD al botón butEdad

En la declaración de la clase PanelDatosMuestra:

```
/**
 * El comando para la opción en ayunas
 */
private final static String AYUNAS = "En ayunas";

/**
 * El comando para el boton de calcular el hematocrito
 */
private final static String CALCULAR_HEMATOCRITO = "CALCULAR HEMATOCRITO";

/**
 * El comando para el boton de calcular los leucocitos
 */
private final static String CALCULAR_LEUCOCITOS = "CALCULAR LEUCOCITOS";
```

Figura 28. Declaración de las constantes para los eventos que se generan cuando el usuario presiona los botones "Calcular Hematocrito" y "Calcular Leucocitos" del panel de datos de la muestra

En el método constructor de la clase PanelDatosMuestra:

```
/**
 * Se crea el botón para calcular el valor del hematocrito
 * y se le asocia el comando respectivo (la constante)
 */
butHematocrito = new JButton( );
butHematocrito.setText( "Calcular Hematocrito" );
butHematocrito.setActionCommand( CALCULAR_HEMATOCRITO );

/**
 * Se crea el botón para calcular los leucocitos
 * y se le asocia el comando respectivo (la constante)
 */
butLeucocitos = new JButton( );
butLeucocitos.setText( "Calcular Leucocitos" );
butLeucocitos.setActionCommand( CALCULAR_LEUCOCITOS );

/**
 * Se crea el botón de chequeo para indicar si la muestra
 * está en ayunas y se le asocia el comando respectivo
 * (la constante)
 */
cbAyunas = new JCheckBox("Ayunas");
cbAyunas.setActionCommand(AYUNAS);
```

Figura 29. Asociación de los comandos CALCULAR_HEMATOCRITO, CALCULAR_LEUCOCITOS y AYUNAS a los botones butHematocrito y butLeucocitos y al botón de chequeo cbAyunas, respectivamente

En la declaración de la clase PanelExtensiones:

```java
public class PanelExtensiones extends JPanel
{

    //-----------------------------------------------------
    // Constantes
    //-----------------------------------------------------

    /**
     * El comando para el botón avanzar
     */
    private static final String AVANZAR = "AVANZAR";

    /**
     * El comando para el botón retroceder
     */
    private static final String RETROCEDER = "RETROCEDER";

    /**
     * El comando para el botón 1
     */
    private final String OPCION_1 = "opcion1";

    /**
     * El comando para el botón 2
     */
    private final String OPCION_2 = "opcion2";
```

Figura 30. Declaración de las constantes para los eventos que se generan cuando el usuario presiona los botones "Avanzar", "Retroceder", "Opción 1" y "Opción 2" del panel de extensión

En el método constructor de la clase PanelExtensiones:

```java
/**
 * Se crean los botones para avanzar y retroceder y
 * se le asocia los comandos respectivo (las constantes)
 */
btnAvanzar = new JButton( ">" );
btnAvanzar.setActionCommand( AVANZAR );

btnRetroceder = new JButton( "<" );
btnRetroceder.setActionCommand( RETROCEDER );

/**
 * Se crean los botones de opcion1 y opcion2
 * se le asocia los comandos respectivo (las constantes)
 */
butOpcion1 = new JButton( "Opción 1" );
butOpcion1.setActionCommand( OPCION_1 );

butOpcion2 = new JButton( "Opción 2" );
butOpcion2.setActionCommand( OPCION_2 );
```

Figura 31. Asociación de los comandos AVANZAR, RETROCEDER, OPCION_1 y OPCION_2 a los botones btnAvanzar, btnRetroceder, butOpcion1 y butOpcion2 respectivamente

2. El segundo paso para asociar una acción del usuario con un evento de la aplicación consiste en atender el evento. Para esto:
   - Las clases de los páneles que reciben acciones del usuario deben implementar la interfaz de Java ActionListener.
   - Estas mismas clases deben implementar el método actionPerformed de la interfaz ActionListener.

Usted debe:

1. Modificar el encabezado de cada clase de panel que contiene botones para añadir "implements ActionListener" (para que pueda "percibir" las acciones del usuario). Para que la interfaz ActionListener sea reconocida, usted debe importar las librerías java.awt.event.ActionEvent y java.awt.event.ActionListener. A continuación se ilustra el ejemplo sobre la clase PanelDatosPaciente. Usted debe replicar los mismos cambios sobre PanelDatosMuestra y PanelExtensiones.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class PanelDatosPaciente extends JPanel implements ActionListener
{
```

Figura 32. Modificación del encabezado de la clase PanelDatosPaciente para que implemente la interfaz ActionListener e importación de las librerías necesarias

2. Implementar en la clase del panel que contiene botones, el método especial ActionPerformed. A continuación se ilustra el ejemplo sobre la clase PanelDatosMuestra. Usted debe replicar los mismos cambios sobre PanelDatosPaciente y PanelExtensiones. Por el momento, solo es necesario que escriba el esqueleto del método sin hacer ningún llamado a métodos de la ventana principal.

```java
/**Método actionPerformed que se ejecuta cada vez que se
 * genera un evento como consecuencia de una acción del usuario
 * @param evento Evento generado
 */
public void actionPerformed(ActionEvent evento)
{
    String comando = evento.getActionCommand( );
    if(comando.equals( CALCULAR_HEMATOCRITO ))
    {
        //reación al evento CALCULAR_HEMATOCRITO;
    }
    else if(comando.equals( CALCULAR_LEUCOCITOS ))
    {
        //reación al evento CALCULAR_LEUCOCITOS;
    }
    else if (comando.equals(AYUNAS))
    {
        //reación al evento AYUNAS;
    }
}
```

Figura 33. Esqueleto del método actionPerformed de la clase PanelDatosMuestra

3. El tercer paso para asociar una acción del usuario con un evento de la aplicación consiste en decir que el pánel (al cual pertenecen los botones) es el encargado de atender el evento. Para esto: haga el llamado al método addActionListener de la clase JButton (o JCheckBox según corresponda) sobre todos los botones de los páneles de su aplicación. Este método recibe como parámetro el panel que contiene al botón (es decir, **this**). A continuación se ilustra el ejemplo sobre el método constructor de la clase PanelDatosMuestra. Usted debe replicar los mismos cambios sobre los métodos constructores de las clases PanelDatosPaciente y PanelExtensiones.

```java
// Llamado al método addActionListener sobre los botones
// para indicar que el panel es el encargado de atender el
// evento que se genere cuando se presiona el botón
// Note que this en este caso es el panel mismo
butHematocrito.addActionListener( this );
butLeucocitos.addActionListener( this );
cbAyunas.addActionListener( this );
```

Figura 34. Llamado al método AddActionListener sobte los botones y la caja de chequeo de la clase PanelDatosMuestra

ETAPA 3: CREACIÓN DE LOS MÉTODOS DE LAS CLASES DE LOS PÁNELES

Recordemos que el rol del método actionPerformed de los páneles es identificar cuál fue el evento que se disparó como consecuencia de una acción del usuario (como oprimir un botón del pánel) y responder al mismo, llamando a un método de la ventana principal. Es por lo tanto indispensable que los páneles que implementan este método actionPerformed, conozcan a la ventana principal.

1. Añada a la clase PanelDatosMuestra un nuevo atributo que represente la interfaz principal como se muestra en la siguiente figura. Haga lo mismo sobre PanelDatosPaciente y PanelExtensiones.

```
public class PanelDatosMuestra extends JPanel implements ActionListener
{

    //-----------------------------------------------------------------
    // Atributos
    //-----------------------------------------------------------------

    /**
     *  Interfaz principal de la aplicación
     */
    private InterfazSistemaPacientes principal;
```

Figura 35. Declaración de un nuevo atributo en la clase PanelDatosMuestra para guardar la asociación con la ventana principal

2. Modifique el método constructor de la clase PanelDatosMuestra para que reciba como parámetro la interfaz principal y asigne este valor al atributo creado en el punto anterior, como se muestra en la siguiente figura. Haga lo mismo sobre PanelDatosPaciente y PanelExtensiones.

```
//-----------------------------------------------------------------
// Método constructor
//-----------------------------------------------------------------
public PanelDatosMuestra(InterfazSistemaPacientes v)
{
    principal=v;
```

Figura 36. Modificación del método constructor de la clase PanelDatosMuestra para que reciba como parámetro la asociación a la ventana principal y la guarde en el nuevo atributo

3. Modifique la instrucción de creación del panelDatosMuestra en el método constructor de la clase principal de la ventana principal, pasándole como parámetro dicha ventana

principal (**this**), como se muestra en la siguiente figura. Haga lo mismo sobre panelDatosPaciente y panelExtensiones.

```
public InterfazSistemaPacientes()
{
    setTitle( "Sistema de Pacientes" );
    setSize( 700, 450 );
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    setLayout( new BorderLayout( ) );

    // Crea el panel de datos de la muestra
    panelDatosMuestra = new PanelDatosMuestra( this );
```

Figura 37. Modificación de la instrucción de creación del panelDatosMuestra pasándole como parámetro la asociación a la ventana principal (this)

4. Ahora que la clase PanelDatosMuestra conoce a la ventana principal, complete el esqueleto del método actionPerfomed que usted mismo creó en la etapa anterior (ver figura 33) con los llamados a los métodos de la ventana principal, en función del evento que se haya generado y que haya llegado como parámetro a este método actionPerformed. La siguiente figura ilustra cómo llevar a cabo estos llamados. Es normal que en este punto aparezcan los errores que se muestran en la figura ya que dichos métodos no han sido aún implementados en la clase de la ventana principal.

```
227     /**Método actionPerformed que se ejecuta cada vez que se
228      * genera un evento como consecuencia de una acción del usuario
229      * @param evento Evento generado
230      */
231     public void actionPerformed(ActionEvent evento)
232     {
233         String comando = evento.getActionCommand( );
234         if(comando.equals( CALCULAR_HEMATOCRITO ))
235         {
236             principal.calcularHematocrito();
237         }
238         else if(comando.equals( CALCULAR_LEUCOCITOS ))
239         {
240             principal.calcularLeucocito();
241         }
242         else if (comando.equals(AYUNAS))
243         {
244             principal.cambiarEnAyunas();
245         }
246     }
```

Figura 38. Modificación del método actionPerformed de la clase PanelDatosMuestra con los llamados a los métodos de la ventana principal en función del evento que se haya generado

5. Repita el paso anterior en la clase PanelDatosPaciente. Tenga en cuenta que el panel de los datos del paciente cuenta con un único botón (el que permite calcular la edad). Por consiguiente, su método actionPerformed sólo debe llamar a un método de la ventana principal. Este es calcularEdad.

6. Repita el paso anterior en la clase PanelExtensiones. Tenga en cuenta que este panel cuenta con cuatro botones (avanzar, retroceder, opción1 y opción 2). Los métodos de la ventana principal que deben ser llamados desde el actionPerformed del panel son: avanzar, retroceder, reqFuncOpcion1 y reqFuncOpcion2.

Cuando los páneles hacen llamados a métodos de la ventana principal como respuesta a una acción del usuario, esta se comunica con el mundo, quien es el responsable de ejecutar los requerimientos funcionales de la aplicación. El mundo responde a dicho llamado, ya sea modificando el estado de sus objetos, ya sea devolviendo algún valor a la interfaz o ambos. Cuando la interfaz recibe información del mundo, como resultado de la ejecución de alguno de sus métodos, debe mostrarla al usuario a través de las zonas de texto que se encuentran

en los páneles. Es por lo tanto indispensable que los páneles cuenten con métodos que permitan mostrar dicha información.

7. Para cada uno de los JTextField y para el JCheckBox creados en la clase PanelDatosMuestra, cree un método que permita establecer el valor del campo de texto/caja de chequeo con un valor dado como parámetro, como se ilustra en la siguiente figura:

```
/**
 * Método que muestra el valor de hematocrito
 * recibido por parámetro en el campo de texto
 * txtHematocrito
 * @param pHematocrito Cantidad hematocrito
 */
public void mostrarHematocrito( String pHematocrito )
{
    txtHematocrito.setText( pHematocrito +"" );
}

/**
 * Método que muestra el valor de leucocito
 * recibido por parámetro en el campo de texto
 * txtLeucocitos
 * @param pLeucocito Cantidad leucocito
 */
public void mostrarLeucocito( String pLeucocito )
{
    txtLeucocitos.setText( pLeucocito+"" );
}

/**
 * Método que muestra el valor de "en ayunas"
 * de acuerdo al recibido por parámetro
 * en la caja de chequeo cbAyunas
 * @param enAyunas Indica true o false si la muestra es en ayunas o no
 */
public void mostrarAyunas( boolean pEnAyunas )
{
    cbAyunas.setSelected( pEnAyunas );
}
```

Figura 39. Creación de los métodos que permiten desplegar información en las zonas de texto y en la caja de chequeo de la clase PanelDatosMuestra

8. Repita el paso anterior en la clase PanelDatosPaciente para el campo de texto de la edad.

   De forma análoga, es necesario que la ventana principal pueda obtener la información que el usuario haya ingresado en un momento dado en los campos de texto que se encuentran en los páneles. Esta información normalmente constituye las entradas de los requerimientos funcionales y debe ser suministrada al mundo como parámetros de los

métodos que sean llamados por la ventana principal. Es por lo tanto indispensable que los páneles cuenten con métodos que permitan conocer dicha información.

9. Para cada uno de los JTextField de la clase PanelDatosMuestra, cree un método que devuelva el valor del campo de texto como un String, como se ilustra en la siguiente figura:

```
/**
 * Método que devuelve el volumen de la muestra que haya ingresado
 * el usuario en el campo de texto txtVolumenMuestra
 * @return el volumen de la muestra
 */
public String darVolumenMuestra()
{
    String rta=txtVolumenMuestra.getText();
    return rta;
}

/**
 * Método que devuelve el volumen de eritrocitos que haya ingresado
 * el usuario en el campo de texto txtVolumenEritrocitos
 * @return el volumen de eritrocitos
 */
public String darVolumenEritrocitos()
{
    String rta=txtVolumenEritrocitos.getText();
    return rta;
}
```

```
/**
 * Método que devuelve el conteo de leucocitos que haya ingresado
 * el usuario en el campo de texto txtConteoLeucocitos
 * @return el conteo de leucocitos
 */
public String darConteoLeucocitos()
{
    String rta=txtConteoLeucocitos.getText();
    return rta;
}

/**
 * Método que devuelve el conteo de plaquetas que haya ingresado
 * el usuario en el campo de texto txtPlaquetas
 * @return el conteo de plaquetas
 */
public String darConteoPlaquetas()
{
    String rta=txtConteoLeucocitos.getText();
    return rta;
}
```

Figura 40. Creación de los métodos que permiten capturar la información de las zonas de texto de la clase PanelDatosMuestra

10. Repita el paso anterior para el JCheckBox de la clase PanelDatosMuestra, como se ilustra en la siguiente figura:

```
/**
 * Método que devuelve el valor (true/false que indica seleccionada/no seleccionada)
 * de la caja de chequeo cbAyunas
 * @return true o false
 */
public boolean estaEnAyunas()
{
    return cbAyunas.isSelected();
}
```

Figura 41. Creación del método que permite capturar la información de la caja de chequeo de la clase PanelDatosMuestra

11. Repita el paso 9 en la clase PanelDatosPaciente para el campo de texto de la edad.

    Puede ser útil contar con métodos que “limpien” los campos de texto, por ejemplo si se fuera posible crear un nuevo paciente y se quisiera tener la ventana sin información.

12. En la clase PanelDatosMuestra cree un método que limpie los campos de texto del hematocrito y del conteo de leucocitos, como se muestra en la siguiente figura:

```java
/**
 * Limpia los campos de texto del panel.
 */
public void limpiarCampos( )
{
    txtHematocrito.setText( "" );
    txtLeucocitos.setText( "" );
}
```

Figura 42. Creación del método limpiarCampos de la clase PanelDatosMuestra

13. Repita el paso anterior en la clase PanelDatosPaciente para limpiar el campos de texto de la edad.

Cuando se navega entre los pacientes (avanzando o retrocediendo) es necesario actualizar todos los campos de información con los datos del paciente que se está visualizando. Es entonces interesante contar con métodos en los páneles que actualicen todos sus elementos gráficos, evitando así sin tener que hacerlo uno por uno.

14. En la clase PanelDatosMuestra cree un método que le permita actualizar todos los campos con los valores dados como parámetro, como se ilustra en la siguiente figura:

```java
/**
 * Actualiza los campos del panel con la información que entra como parámetro
 * @param pFechaMuestra: fecha de toma de la muestra
 * @param pEnAyunas: indica si la muestra es en ayunas o no
 * @param pVolumenMuestra: volumen de la muestra
 * @param pVolumenEritrocitos: volumen de eritrocitos
 * @param pConteoLeucocitos: conteo de leucocitos
 * @param pConteoPlaquetas: conteo de Plaquetas
 */
public void actualizarCampos(String pFechaMuestra, boolean pEnAyunas,
        double pVolumenMuestra, double pVolumenEritrocitos,
        int pConteoLeucocitos, int pConteoPlaquetas)
{
    txtFTomaMuestra.setText(pFechaMuestra);
    txtVolumenMuestra.setText(pVolumenMuestra + "");
    txtVolumenEritrocitos.setText(pVolumenEritrocitos + "");
    txtConteoLeucocitos.setText(pConteoLeucocitos + "");
    txtConteoPlaquetas.setText(pConteoPlaquetas + "");
    cbAyunas.setSelected(pEnAyunas);
}
```

Figura 43. Creación del método actualizarCampos de la clase PanelDatosMuestra

15. Repita el paso anterior en la clase PanelDatosPaciente para actualizar los campos del nombre, apellido, sexo, fecha de nacimiento e imagen, como se ilustra en la siguiente figura. Note cómo se asocia una imagen a una etiqueta, a través del método setIcon de la clase JLabel.

```java
/**
 * Actualiza los campos del panel con la información del paciente.
 * @param pNombre: nombre del paciente
 * @param pApellido: apellido del paciente
 * @param pSexo: sexo del paciente
 * @param pFechaN: fecha de nacimiento
 * @param imagen: ruta donde se encuentra la imagen (foto)
 */
public void actualizarCampos( String pNombre, String pApellido,
                              String pSexo, String pFechaN, String pImagen )
{
    txtNombre.setText( pNombre );
    txtApellido.setText( pApellido );
    txtSexo.setText( pSexo );
    txtFNacimiento.setText( pFechaN );
    labImagen.setIcon( new ImageIcon( pImagen ) );
    txtEdad.setText("");
}
```

Figura 44. Creación del método actualizarCampos de la clase PanelDatosPaciente

ETAPA 4: CREACIÓN DE LOS MÉTODOS DE LA CLASE DE LA VENTANA PRINCIPAL

En esta última etapa de la guía de trabajo, usted creará en la clase InterfazSistemaPacientes, los métodos necesarios para responder a los requerimientos funcionales de la aplicación. Recuerde que estos métodos de la ventana principal son aquellos llamados por los métodos actionPerformed de las clases PanelDatosMuestra, PanelDatosPaciente y PanelExtensiones.

A continuación se muestran a manera de ejemplo las implementaciones de los siguientes métodos: actualizarInfoPaciente, avanzar, calcularHematocrito, cambiarEnAyunas y reqFuncOpcion1. Usando estos métodos como guía, usted debe implementar los métodos que quedarían pendientes para terminar por completo la aplicación. Estos son: retroceder, calcularEdad, calcularLeucocito y reqFuncOpcion2.

Nota: Tenga en cuenta las verificaciones de formatos, los mensajes de error y las actualizaciones necesarias por cada evento.

1. Método actualizarInfoPaciente: este método es utilizado por la misma ventana principal, cada vez que se navega (avanza o retrocede) entre los pacientes.

```
/**
 * Actualiza la información del paciente dado por parámetro.
 * @param pPaciente Paciente del cual se mostrarón los datos.
 */
public void actualizarInfoPaciente( Paciente pPaciente )
{
    String nombre = pPaciente.darNombre( );
    String apellido = pPaciente.darApellido( );

    String sexo = "F";
    int iSexo = pPaciente.darSexo( );
    if( iSexo == 2 )
        sexo = "M";

    String fechaTomaMuestra = pPaciente.darFechaTomaMuestra( );
    String fechaN = pPaciente.darFechaNacimiento( );
    String imagen = pPaciente.darImagen( );
    double volumenMuestra = pPaciente.darVolumenMuestra();
    double volumenEritrocitos =pPaciente.darVolumenEritrocitos();
    int conteoLeucocitos =pPaciente.darConteoLeucocitos();
    int conteoPlaquetas =pPaciente.darConteoPlaquetas();
    boolean enAyunas = pPaciente.darEnAyunas();

    panelDatosPaciente.actualizarCampos( nombre, apellido, sexo, fechaN, imagen );
    panelDatosMuestra.actualizarCampos(fechaTomaMuestra, enAyunas, volumenMuestra,
                                       volumenEritrocitos, conteoLeucocitos,
                                       conteoPlaquetas);
    panelDatosMuestra.limpiarCampos( );
}
```

Figura 45. Método actualizarInfoPaciente de la clase InterfazSistemaPacientes

2. Método avanzar: este método es llamado por el PanelExtensiones cuando el usuario oprime el botón de avanzar.

```
/**
 * Avanza al siguiente paciente y actualiza la información de la interfaz
 * @param pPaciente Paciente del cual se mostrarón los datos.
 */
public void avanzar()
{
    Paciente actual = sistemaPacientes.darPacienteSiguiente( );
    actualizarInfoPaciente( actual );
}
```

Figura 46. Método avanzar de la clase InterfazSistemaPacientes

3. Método calcularHematocrito: este método es llamado por el PanelDatosMuestra cuando el usuario oprime el botón “Calcular Hematocrito”.

```
/**
 * Calcula y muestra el valor de hematocrito del paciente.
 */
public void calcularHematocrito()
{
    String pVolumenMuestra=panelDatosMuestra.darVolumenMuestra();
    String pVolumenEritrocitos=panelDatosMuestra.darVolumenEritrocitos();

    if(pVolumenMuestra.trim().equals("") || pVolumenEritrocitos.trim().equals("") )
    {
        JOptionPane.showMessageDialog( this, "Debe ingresar los datos.",
                                       "Calcular Hematocrito", JOptionPane.ERROR_MESSAGE );

    }
    else if(pVolumenMuestra.trim().matches( "[0123456789.]*" ) &&
            pVolumenEritrocitos.trim().matches( "[0123456789.]*" ) )
    {

        double volumenMuestra = Double.parseDouble( pVolumenMuestra.trim( ) );
        double volumenEritrocitos = Double.parseDouble( pVolumenEritrocitos.trim( ) );

        sistemaPacientes.darPacienteActual().cambiarVolumenMuestra(volumenMuestra);
        sistemaPacientes.darPacienteActual().cambiarVolumenEritrocitos(volumenEritrocitos);

        double hematocrito=sistemaPacientes.darPacienteActual( ).calcularHematocrito();
        hematocrito = Math.round(hematocrito * 100.0) / 100.0;
        panelDatosMuestra.mostrarHematocrito(" " + hematocrito);
    }
    else
    {
        JOptionPane.showMessageDialog( this, "Los datos ingresados no son correctos.",
                                       "Calcular Hematocrito", JOptionPane.ERROR_MESSAGE );
    }
}
```

Figura 47. Método calcularHematocrito de la clase InterfazSistemaPacientes

4. Método cambiarEnAyunas: este método es llamado por el PanelDatosMuestra cuando el usuario oprime la caja de chequeo “En ayunas”:

```java
/**
 * Muestra un mensaje indicando la muestra está en ayunas o no
 * dependiendo de la selección o deselección que haya hecho el
 * usuario en el panel de los datos de la muestra
 */
public void cambiarEnAyunas()
{
    if ( panelDatosMuestra.estaEnAyunas())
    {
        JOptionPane.showMessageDialog(this, "En ayunas", "Estado", JOptionPane.INFORMATION_MESSAGE);
    }
    else
    {
        JOptionPane.showMessageDialog(this, "No en ayunas", "Estado", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

Figura 48. Método cambiarEnAyunas de la clase InterfazSistemaPacientes

5. Método reqFuncOpcion1: este método es llamado por el PanelExtensiones cuando el usuario oprime el botón “Opción 1”:

```java
/**
 * Metodo para la extensión 1
 */
public void reqFuncOpcion1( )
{
    String resultado = sistemaPacientes.metodo1( );
    JOptionPane.showMessageDialog( this, resultado, "Respuesta", JOptionPane.INFORMATION_MESSAGE );
}
```

Figura 49. Método reqFuncOpcion1 de la clase InterfazSistemaPacientes

6. Implemente los métodos retroceder, calcularEdad, calcularLeucocito y reqFuncOpcion2 de la clase de la ventana principal.

7. Ejecute el programa y compruebe que funciona correctamente.