Proceso de Consulta y Actualización de Cotizaciones de Criptomonedas

1. Inicio de la Aplicación

- La aplicación comienza creando una instancia de PantallaCotizaciones y ControladorCotizaciones.

```
PantallaCotizaciones vista = new PantallaCotizaciones();
ControladorCotizaciones controlador = new ControladorCotizaciones(vista, usuarioId);
```

2. Inicialización del Controlador

- El constructor de ControladorCotizaciones llama al método inicializar.

```
public ControladorCotizaciones(PantallaCotizaciones vista, int usuarioId) {
    this.vista = vista;
    this.usuarioId = usuarioId;
    inicializar();
}
```

3. Configuración de la Vista

- En el método inicializar, se configuran algunos aspectos de la vista:
    - Se establece el ID del usuario en la vista.
    - Se añade un listener al botón "Volver" para cerrar la ventana cuando se haga clic.
    - Se cargan las cotizaciones iniciales.

```
private void inicializar() {
    vista.setUsuario(usuarioId);
    vista.addVolverListener(e -> vista.dispose());
    cargarCotizacionesIniciales();
}
```

4. Carga de Cotizaciones Iniciales

- El método cargarCotizacionesIniciales obtiene los precios de las criptomonedas llamando al método estático obtenerPrecios de ConsultarPrecioCripto.
- Luego, actualiza la base de datos con estos precios y los muestra en la vista.

```java
private void cargarCotizacionesIniciales() {
    try {
        Map<String, Double> datos = ConsultarPrecioCripto.obtenerPrecios();
        actualizarBaseDeDatos(datos);
        vista.cargarCriptos(datos);
    } catch (Exception e) {
        JOptionPane.showMessageDialog(vista, "Error al cargar cotizaciones iniciales: " + e.getMessage(),
                "Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

5. Obtención de Precios de Criptomonedas

- El método obtenerPrecios en ConsultarPrecioCripto realiza una solicitud HTTP a la API de CoinGecko.
- Si la respuesta es exitosa (código de estado 200), se parsea el cuerpo de la respuesta JSON para extraer los precios y se devuelve un HashMap con los datos.

```java
public static Map<String, Double> obtenerPrecios() throws IOException, InterruptedException {
    HttpClient cliente = HttpClient.newHttpClient();
    HttpRequest solicitud = HttpRequest.newBuilder()
            .uri(URI.create(URL_API))
            .GET()
            .build();

    HttpResponse<String> respuesta = cliente.send(solicitud, HttpResponse.BodyHandlers.ofString());
    if (respuesta.statusCode() == 200) {
        return parsearPrecios(respuesta.body());
    } else {
        throw new IOException("Error al consultar precios. Código de estado: " + respuesta.statusCode());
    }
}
```

6. Actualización de la Base de Datos

- El método actualizarBaseDeDatos en ControladorCotizaciones actualiza los precios de las criptomonedas en la base de datos SQLite.
- Utiliza una conexión a la base de datos y un PreparedStatement para ejecutar las actualizaciones.

```java
private void actualizarBaseDeDatos(Map<String, Double> precios) {
    String query = "UPDATE Moneda SET valor = ? WHERE nombre = ?";
    String url = "jdbc:sqlite:projectDatabase.db";
    try (Connection conexion = DriverManager.getConnection(url);
         PreparedStatement stmt = conexion.prepareStatement(query)) {
        for (Map.Entry<String, Double> entry : precios.entrySet()) {
            stmt.setDouble(1, entry.getValue()); // Establece el precio
            stmt.setString(2, entry.getKey());   // Establece el nombre de la moneda
            stmt.executeUpdate();                // Ejecuta la actualización
        }
        System.out.println("Precios actualizados en la base de datos.");
    } catch (SQLException e) {
        System.err.println("Error al actualizar los precios en la base de datos: " + e.getMessage());
    }
}
```

7. Carga de Criptomonedas en la Vista

- El método cargarCriptos en PantallaCotizaciones actualiza la interfaz gráfica para mostrar las criptomonedas y sus precios.
- Se crean paneles para cada criptomoneda y se añaden al panel principal.

```
public void cargarCriptos(Map<String, Double> datosCriptos) {
    panelCriptos.removeAll();
    etiquetasPrecios.clear();

    datosCriptos.forEach((nombre, precio) -> {
        JPanel panelCripto = new JPanel(new GridBagLayout());
        panelCripto.setBackground(Color.WHITE);
        panelCripto.setBorder(BorderFactory.createMatteBorder(0, 0, 1, 0, Color.LIGHT_GRAY));

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(5, 10, 5, 10);
        gbc.fill = GridBagConstraints.HORIZONTAL;

        gbc.gridx = 0;
        gbc.weightx = 0.5;
        gbc.anchor = GridBagConstraints.WEST;
        JLabel lblCripto = new JLabel(nombre);
        lblCripto.setFont(new Font("Arial", Font.BOLD, 14));
        panelCripto.add(lblCripto, gbc);

        gbc.gridx = 1;
        gbc.weightx = 0.5;
        gbc.anchor = GridBagConstraints.CENTER;
        JLabel lblPrecio = new JLabel(String.format("$%.2f", precio));
        lblPrecio.setFont(new Font("Arial", Font.PLAIN, 14));
        panelCripto.add(lblPrecio, gbc);

        etiquetasPrecios.put(nombre, lblPrecio);
        panelCriptos.add(panelCripto);
    });
    panelCriptos.revalidate();
    panelCriptos.repaint();
}
```

Descripción de Clases Adicionales

ControladorCotizaciones.java

- **Manejo de Excepciones en Java**:
    - Se utiliza en los métodos cargarCotizacionesIniciales y actualizarBaseDeDatos para manejar posibles errores al obtener precios y actualizar la base de datos.

```
try {
    Map<String, Double> datos = ConsultarPrecioCripto.obtenerPrecios();
    actualizarBaseDeDatos(datos);
    vista.cargarCriptos(datos);
} catch (Exception e) {
    JOptionPane.showMessageDialog(vista, "Error al cargar cotizaciones iniciales: " + e.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
}
```

- **Manejo de Eventos en GUI**:
    - Se utiliza para manejar el evento de clic en el botón "Volver".

```
vista.addVolverListener(e -> vista.dispose());
```

- **Threads y Concurrencia en Java**:
    - Aunque está comentado, el método iniciarActualizacionCotizaciones muestra cómo se puede usar un Timer para actualizar las cotizaciones periódicamente en un hilo separado.

```
new Timer(5000, e -> actualizarCotizaciones()).start();
```

PantallaCotizaciones.java

- **Manejo de Eventos en GUI**:
    - Se utiliza para manejar el evento de clic en el botón "Volver".

```
public void addVolverListener(ActionListener listener) {
    btnVolver.addActionListener(listener);
}
```

- **AWT y Swing**:
    - Se utilizan para crear y manejar componentes gráficos como JFrame, JPanel, JButton, JLabel, etc.

```
public PantallaCotizaciones() {
    etiquetasPrecios = new HashMap<>();
    setTitle("Billetera Virtual - Cotizaciones");
    setSize(700, 450);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setResizable(false);
    setLayout(new BorderLayout());
    // ... más código para configurar la interfaz gráfica
}
```

- **Framework de Colecciones**:
    - Se utiliza un HashMap para almacenar las etiquetas de precios de las criptomonedas.

```
private Map<String, JLabel> etiquetasPrecios;
```

ConsultarPrecioCripto.java

- **Entrada/Salida (I/O) en Java**:
    - Se utiliza para realizar una solicitud HTTP a la API de CoinGecko y obtener la respuesta.

```
HttpClient cliente = HttpClient.newHttp
```