

# Monitorización de carteras de opciones del IBEX35

Diseño de Interfaces de Usuario

Jose David Santana Rocha  
CURSO 2015/2016

---

## ÍNDICE

---

1. Características.....	2
2. Casos de uso.....	2
3. Diagrama de flujo.....	3
4. Diseño conceptual.....	4
5. Decisiones adoptadas.....	6
6. Detalles de implementación.....	10
6.1. Eventos.....	12
6.2. Funciones.....	21
6.3. Clase cartera.....	26
6.4. Clase FiltroCartera.....	34
7. Material entregado.....	35

## 1. Características

La aplicación cuenta con una cómoda visualización de la cotización puntual del IBEX35, contratos de futuros del MINI-IBEX35 y las opciones del IBEX gestionadas por MEFF, que se va actualizando en un periodo de tiempo.

También permite al usuario la creación y gestión de carteras personales, con las que podrá realizar el seguimiento de las opciones añadidas, y analizar el posible beneficio en el intervalo de tiempo que haya pasado desde que fueron añadidas a la cartera.

Todo ello ofreciendo una interfaz de fácil manejo con la que el usuario se sentirá cómodo, y que podrá percibir de manera intuitiva.

## 2. Casos de uso

El usuario puede realizar diversas actividades en la aplicación, las presentamos a continuación:

**Visualizar opciones:** Se muestran varias tablas en la pantalla principal de la aplicación que muestran información relativa a la cotización puntual, a los contratos de futuros y a las opciones del IBEX35.

**Crear cartera:** Es posible la creación de una cartera personal a la que el usuario podrá poner un nombre de su elección y también añadirle opciones.

**Abrir cartera:** Permite cargar las opciones de una cartera ya existente para visualizarla o modificarla.

**Guardar cartera:** Se ofrece la posibilidad de almacenar en un fichero todos los cambios realizados en la cartera.

**Cerrar cartera:** Si el usuario lo desea, puede descartar una cartera ya abierta del gestor de carteras para una mayor comodidad.

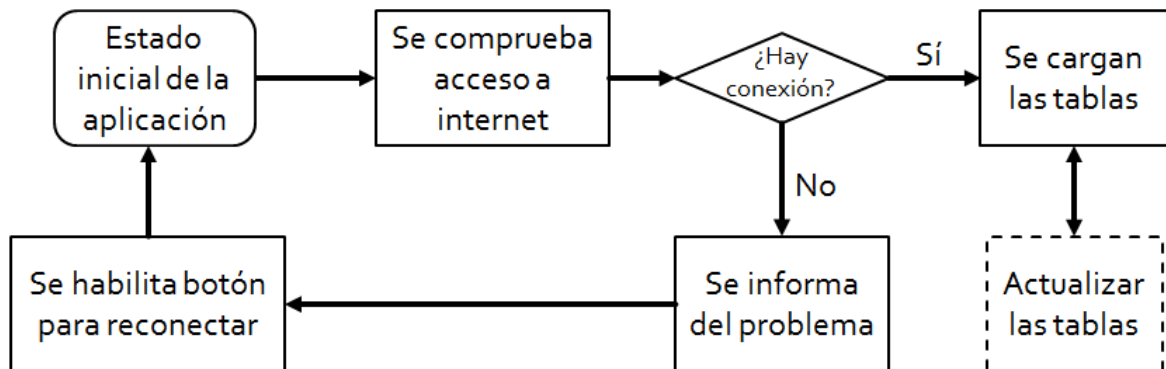
**Añadir opción a la cartera:** Si el usuario tiene al menos una cartera activa, podrá añadir la opción seleccionada a la cartera.

**Eliminar opción de la cartera:** Asimismo, el usuario será capaz de borrar la opción seleccionada de la cartera.

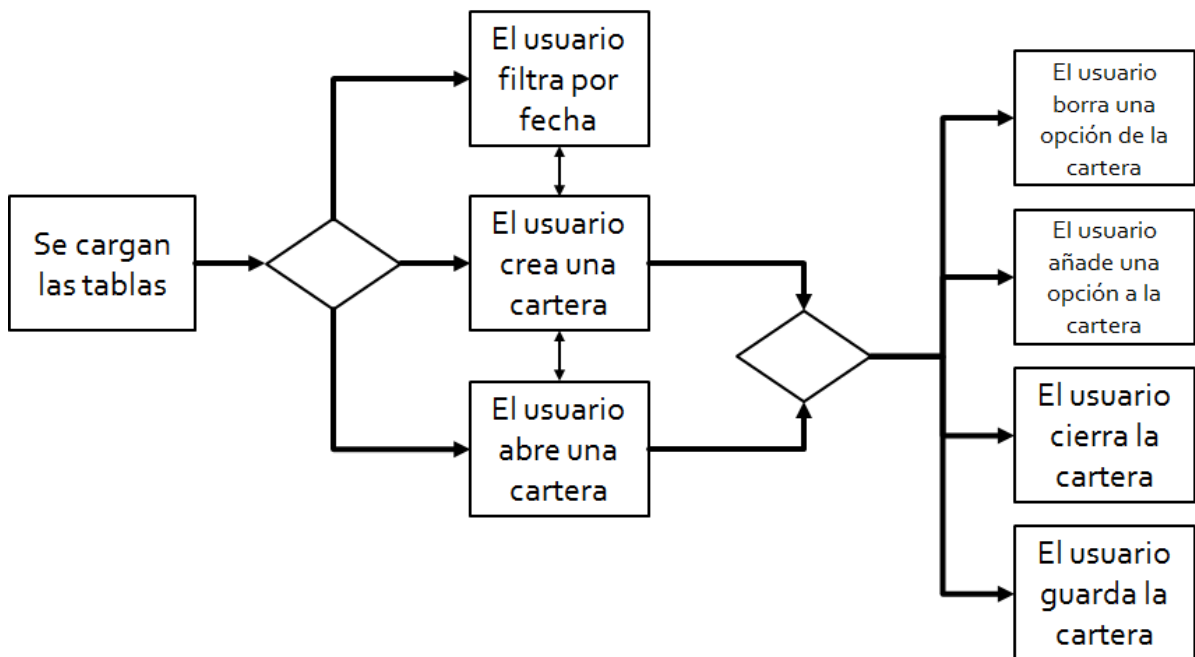
**Filtrar opciones por fecha:** Para visualizar solamente las opciones cuya fecha de vencimiento coincide con la seleccionada en la lista desplegable.

### 3. Diagrama de flujo

Actividades realizadas automáticamente al iniciar la aplicación:

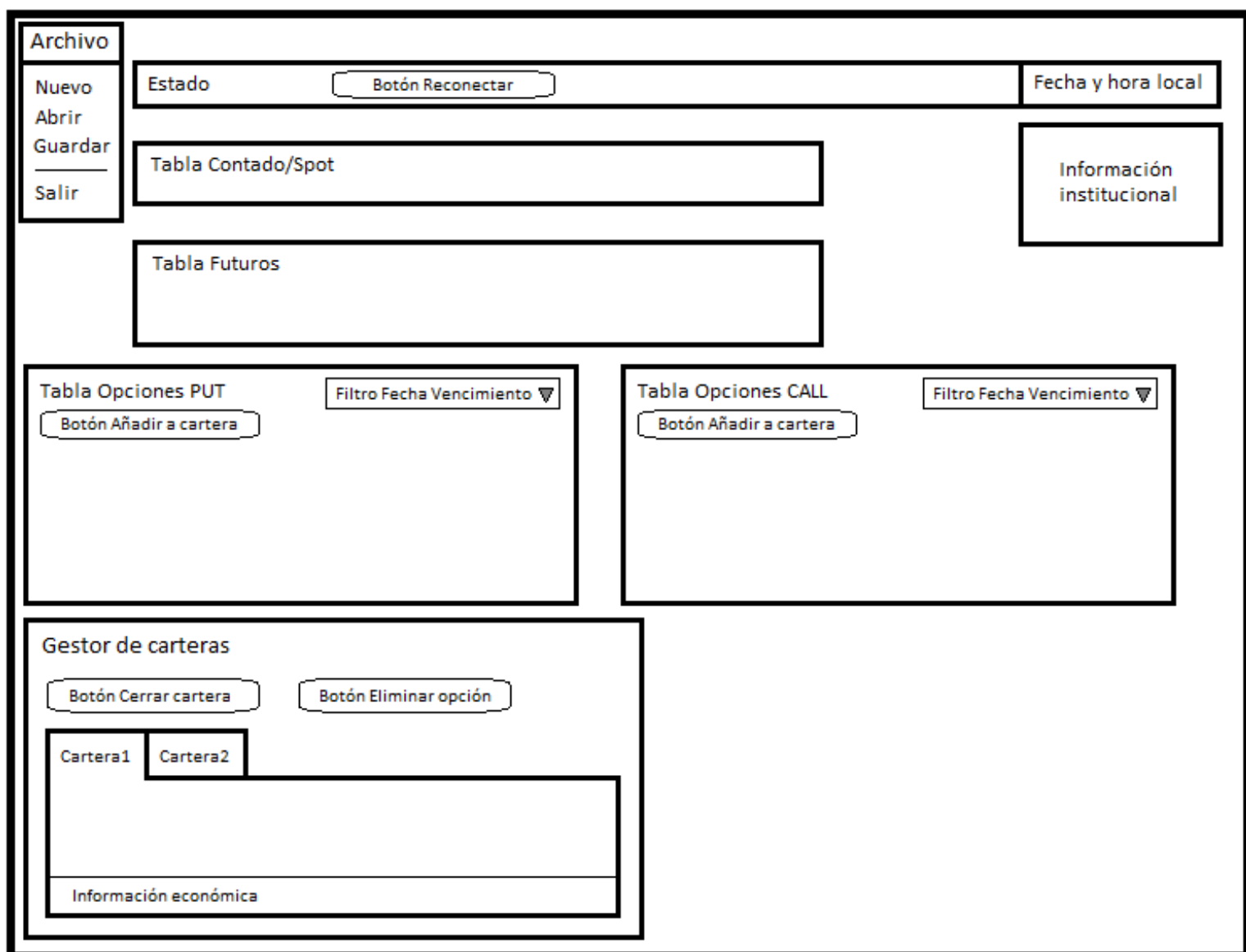


Actividades realizadas por el usuario una vez la aplicación ha iniciado:



#### 4. Diseño conceptual

Modelo esquemático de la aplicación:




Prototipo funcional realizado con *Matisse*:

Archivo
Estado: Cartera sin cargar
Reconectar

Contado/Spot

Último	Diferencia(%)	Anterior	Máximo	Mínimo	Fecha	Hora (Madrid)



Universidad de Las Palmas de Gran Canaria  
Escuela de Ingeniería Informática  
Diseño de Interfaces de Usuario  
Jose David Santana Rocha

Futuros (MINI) IBEX35

Vencimiento	Vol. Compra	P. Compra	P. Venta	Vol. Venta	Último	Volumen	Apertura	Máximo	Mínimo	Anterior	Hora (Ma...

Opciones IBEX35 - PUT

+ Añadir opción a la cartera
Fecha de Vencimiento: Todas

Ejercicio	Vol. Compra	P. Compra	P. Venta	Vol. Venta	Último	Volumen	Hora (Mad...

Opciones IBEX35 - CALL

+ Añadir opción a la cartera
Fecha de Vencimiento: Todas

Ejercicio	Vol. Compra	P. Compra	P. Venta	Vol. Venta	Último	Volumen	Hora (Ma...

Gestor de carteras

X Cerrar cartera
Eliminar opción de la cartera

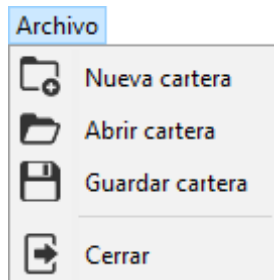
Importe invertido:
Valoración actual:

**Ganancia total:**

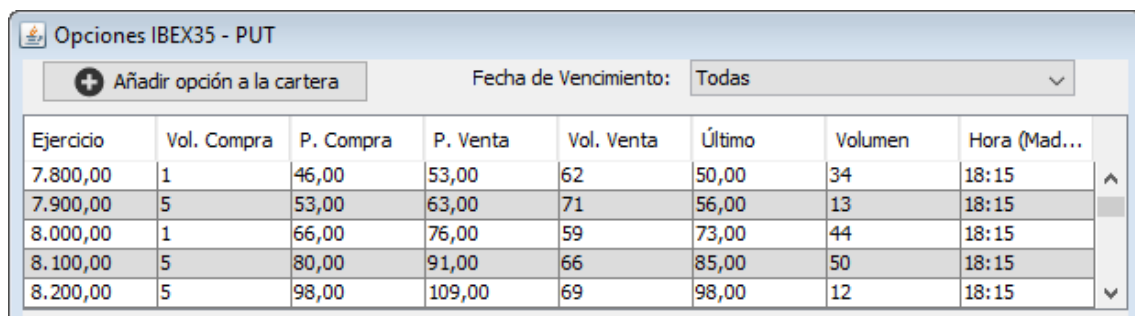
## 5. Decisiones adoptadas

En el menú Archivo se ha añadido una barra separadora que aparta las opciones de crear, abrir o guardar cartera de la opción de cerrar para evitar que el usuario haga clic erróneamente y para que el usuario perciba que se trata de una funcionalidad completamente diferente a las otras.

También se han añadido iconos a cada una de las opciones que reflejan de manera más intuitiva la funcionalidad de esa opción.

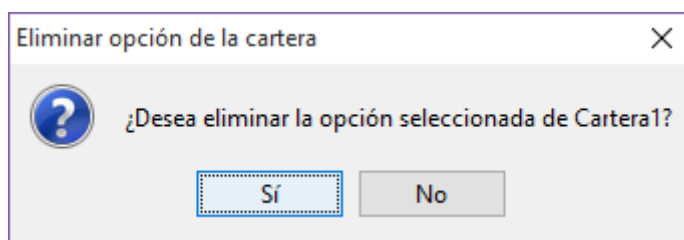


Se ha optado por colorear las filas de las tablas de colores alternados para facilitar la visualización al usuario y para que dé la sensación de que la fila es una unidad, ya que corresponden a una opción puntual.

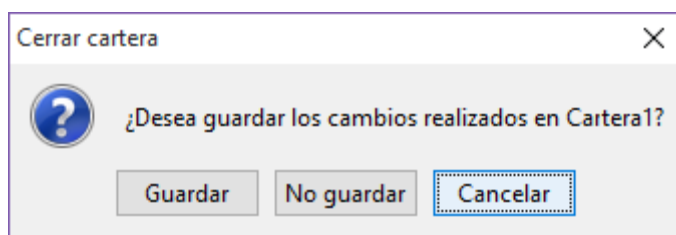


+ Añadir opción a la cartera		Fecha de Vencimiento:		Todas			
Ejercicio	Vol. Compra	P. Compra	P. Venta	Vol. Venta	Último	Volumen	Hora (Mad...
7.800,00	1	46,00	53,00	62	50,00	34	18:15
7.900,00	5	53,00	63,00	71	56,00	13	18:15
8.000,00	1	66,00	76,00	59	73,00	44	18:15
8.100,00	5	80,00	91,00	66	85,00	50	18:15
8.200,00	5	98,00	109,00	69	98,00	12	18:15

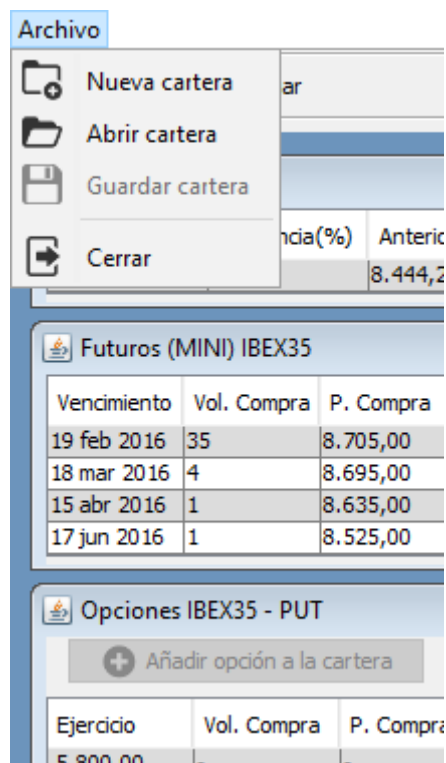
Se ha añadido un mensaje de confirmación cuando el usuario pulsa sobre el botón Eliminar opción de la cartera. Esto evitará que el usuario lamente equivocarse.



También se ha añadido un mensaje de confirmación si el usuario va a cerrar la aplicación o la cartera sin haber guardado los cambios, que le preguntará si desea guardar el fichero.

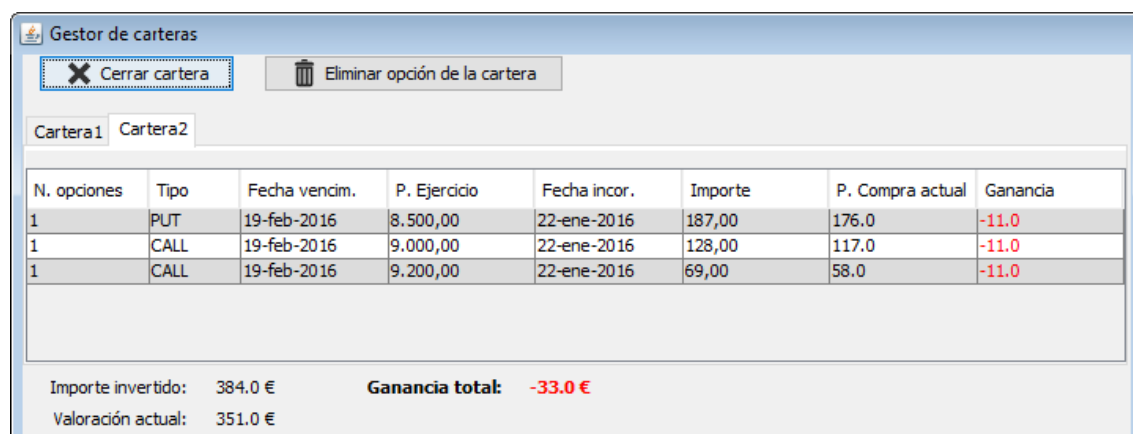


Se ha decidido dejar inactivas inicialmente la opción de Guardar y los botones para Añadir opción a la cartera, hasta que el usuario cargue una cartera.

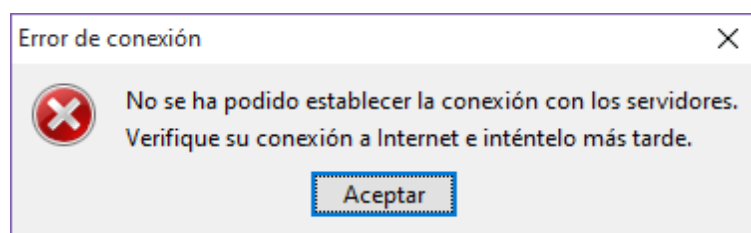


Si el usuario cierra todas las carteras del gestor, las opciones anteriores se volverán inactivas de nuevo.

Cuando el usuario carga o crea una nueva cartera, ésta pasa a ser la cartera activa en el gestor de carteras para una mayor comodidad.

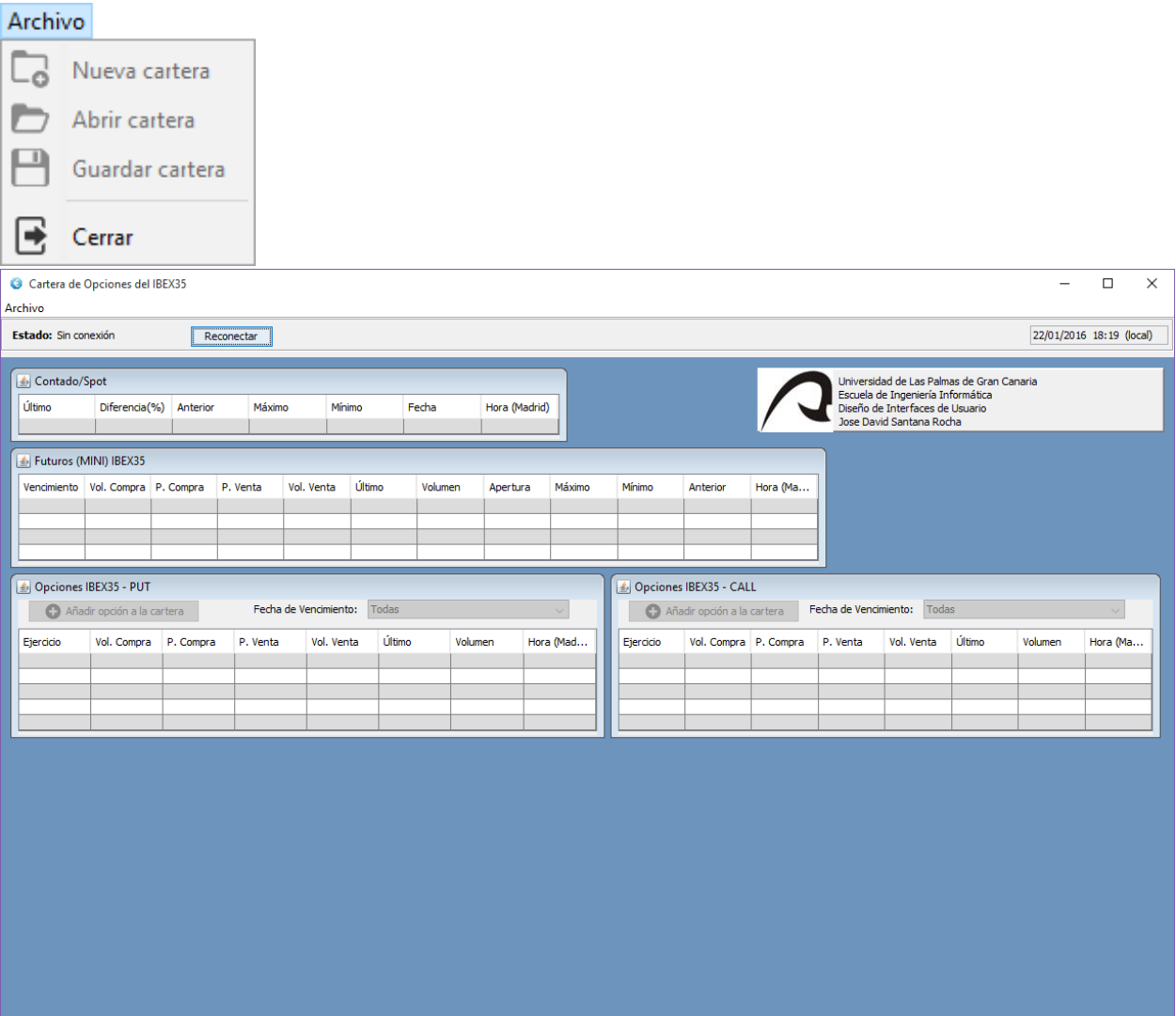


Si se pierde la conexión con la red, al iniciar la aplicación se mostrará un mensaje que informará de ello al usuario.



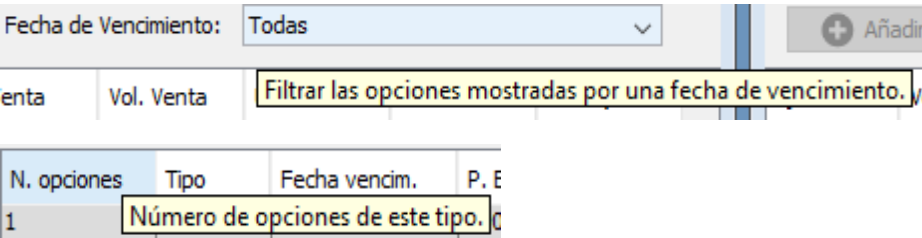


Seguidamente, se mostrará la interfaz de usuario completamente inactiva, y se mostrará un botón para realizar la reconexión.

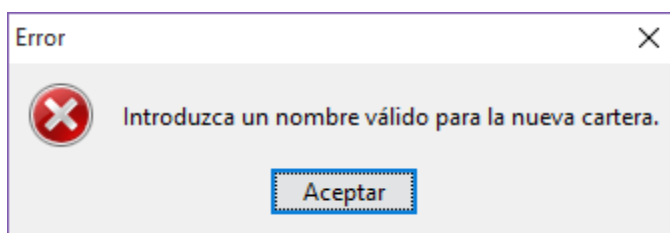
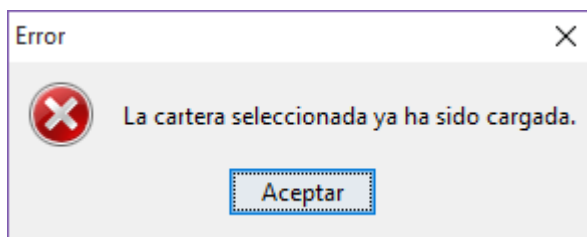
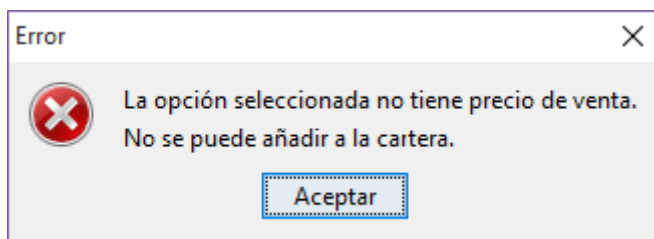
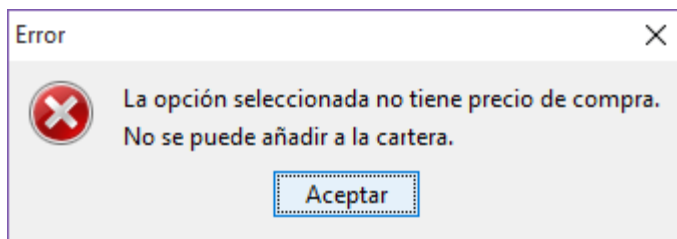
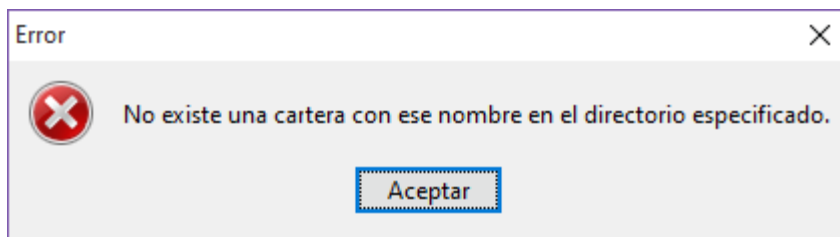
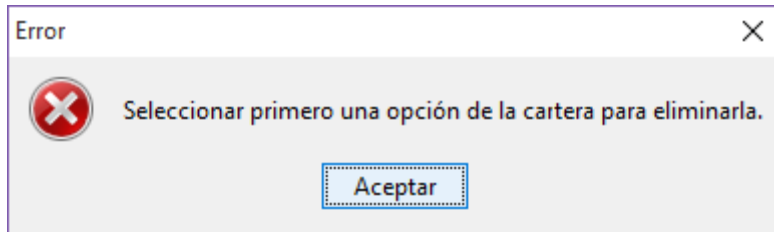
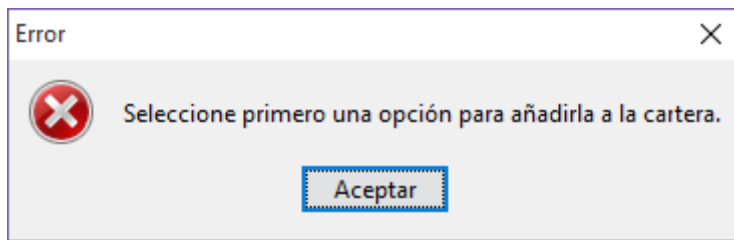


Se ha añadido un panel de estado para que el usuario conozca en todo momento el estado en el que se encuentra la aplicación.

Se ha añadido descripción emergente (tooltip) a cada botón y funcionalidad de la aplicación para que el usuario conozca más detalladamente las características de cierto aspecto que desconozca.



Cada vez que el usuario realiza alguna actividad anómala, se mostrará un mensaje de error.



## 6. Detalles de implementación

En el constructor de la ventana principal tenemos lo siguiente:

```
public MainFrame() {
    initComponents();

    // Coloreamos las tablas
    setCellRender(TablaContado);
    setCellRender(TablaFuturos);
    setCellRender(TablaOpcionesCALL);
    setCellRender(TablaOpcionesPUT);

    // Deshabilitamos inicialmente las opciones que no están disponibles
    opcionGuardar.setEnabled(false);
    botonAddPut.setEnabled(false);
    botonAddCall.setEnabled(false);

    // Permitimos selección simple en las tablas
    TablaContado.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    TablaFuturos.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    TablaOpcionesCALL.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    TablaOpcionesPUT.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

    // Hacemos invisible la ventana de las carteras y el botón de reconectar
    ventanaCartera.setVisible(false);
    botonReconectar.setVisible(false);

    // Centramos las ventanas en la pantalla
    setLocationRelativeTo(null);
    dialogoNuevaCartera.setLocationRelativeTo(null);

    setDate();
    Timer timDate = new Timer(20000, new ActionListener() { // 20 segundos
        @Override
        public void actionPerformed(ActionEvent evt) {
            setDate();
        }
    });
    timDate.start();

    CollectData();
    Timer timData = new Timer(180000, new ActionListener() { // 3 minutos
        @Override
        public void actionPerformed(ActionEvent evt) {
            CollectData();
        }
    });
    timData.start();
}
```

La función *setCellRender* establece el color a cada tabla.

```
// Designar el CellRenderer a cada una de las columnas de la tabla
public void setCellRender(JTable table) {
    Enumeration<TableColumn> en = table.getColumnModel().getColumns();
    while (en.hasMoreElements()) {
        TableColumn tc = en.nextElement();
        tc.setCellRenderer(new CellRenderer());
    }
}
```

- Clase CellRenderer

La clase que colorea las tablas es la siguiente:

```
public class CellRenderer extends DefaultTableCellRenderer implements TableCellRenderer {

    @Override
    public Component getTableCellRendererComponent(JTable table,
        Object value, boolean isSelected, boolean hasFocus, int row, int column) {
        // Color del fondo
        setBackground(null);

        // Constructor de la clase DefaultTableCellRenderer
        super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);

        // Establecemos las filas que queremos cambiar el color.
        boolean oddRow = (row % 2 == 0);

        // Creamos un color para las filas.
        Color c = new Color(220, 220, 220);

        // Si las filas son pares, se cambia el color a gris
        if (!isSelected){
            if (oddRow) {
                setBackground(c);
            }
        }
        return this;
    }
}
```

## 6.1. Eventos

Las opciones del menú Archivo tienen asociado un manejador de eventos.

- Opción Nueva cartera:

```
private void opcionNuevaActionPerformed(java.awt.event.ActionEvent evt) {  
    dialogoNuevaCartera.setVisible(true);  
}
```

Este diálogo de Nueva cartera tiene dos botones, Aceptar y Cancelar, que también tienen asociados manejadores de eventos.

- Botón Aceptar:

```
private void botonAceptarNuevaCarteraActionPerformed(java.awt.event.ActionEvent evt) {  
    // Eliminamos espacios al comienzo y al final del nombre introducido  
    String nombre = campoNombre.getText().trim();  
    // Comprobamos que esté compuesto de caracteres válidos  
    if (nombre.matches("[a-zA-Z0-9\\s]+")) {  
        File archivo = new File(nombre + ".car");  
        JFileChooser fileChooser = new JFileChooser() {  
            @Override  
            public void approveSelection() {  
                File f = getSelectedFile();  
                if (f.exists() && getDialogType() == SAVE_DIALOG) {  
                    int result = JOptionPane.showConfirmDialog(this,  
                        "El fichero seleccionado ya existe, ¿desea sobrescribirlo?",  
                        "Sobrescribir fichero", JOptionPane.YES_NO_CANCEL_OPTION);  
                    switch (result) {  
                        case JOptionPane.YES_OPTION:  
                            super.approveSelection();  
                            return;  
                        case JOptionPane.NO_OPTION:  
                            return;  
                        case JOptionPane.CLOSED_OPTION:  
                            return;  
                        case JOptionPane.CANCEL_OPTION:  
                            cancelSelection();  
                            return;  
                    }  
                }  
                super.approveSelection();  
            }  
        };  
        fileChooser.setSelectedFile(archivo);  
        fileChooser.setCurrentDirectory(null);  
        FiltroCartera filtro = new FiltroCartera();  
        fileChooser.addChoosableFileFilter(filtro);  
        fileChooser.setFileFilter(filtro);  
        fileChooser.setDialogTitle("Seleccione una ubicación");  
        int userSelection = fileChooser.showSaveDialog(this);  
    }  
}
```

```

if (userSelection == JFileChooser.APPROVE_OPTION) {
    File fileToSave = fileChooser.getSelectedFile();
    String ubicacion = fileToSave.getAbsolutePath();
    String extension = ubicacion.substring(ubicacion.lastIndexOf(".") + 1, ubicacion.length());
    if (!extension.equals("car")){
        ubicacion = ubicacion + ".car";
    }
    PrintWriter writer;
    try {
        // Guardamos el fichero con el nombre introducido
        writer = new PrintWriter(ubicacion, "UTF-8");
        writer.println(nombre);
        writer.close();
    } catch (FileNotFoundException | UnsupportedEncodingException ex) {
        Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE, null, ex);
    }
    // Ocultamos el dialogo de nueva cartera y mostramos el gestor de carteras
    dialogoNuevaCartera.setVisible(false);
    ventanaCartera.setVisible(true);
    Cartera cartera = new Cartera(nombre, ubicacion);
    // Añadimos la cartera al gestor de carteras
    gestorCarteras.add(nombre, cartera);
    // Hacemos activa la última cartera añadida
    gestorCarteras.setSelectedIndex(gestorCarteras.getComponentCount() - 1);
}
} else {
    JOptionPane.showMessageDialog(null,
        "Introduzca un nombre válido para la nueva cartera.",
        "Error", JOptionPane.ERROR_MESSAGE);
}
campoNombre.setText("");
opcionGuardar.setEnabled(true);
botonAddPut.setEnabled(true);
botonAddCall.setEnabled(true);
}
}

```

- Botón Cancelar:

```

private void botonCancelarNuevaCarteraActionPerformed(java.awt.event.ActionEvent evt) {
    dialogoNuevaCartera.setVisible(false);
}

```

- Opción Abrir cartera:

```
private void opcionAbrirActionPerformed(java.awt.event.ActionEvent evt) {  
    // Creamos el selector de archivos y le aplicamos el filtro de archivos  
    JFileChooser fileChooser = new JFileChooser();  
    fileChooser.setCurrentDirectory(null);  
    FiltroCartera filtro = new FiltroCartera();  
    fileChooser.addChoosableFileFilter(filtro);  
    fileChooser.setFileFilter(filtro);  
    int retorno = fileChooser.showOpenDialog(this);  
    if (retorno == JFileChooser.APPROVE_OPTION) {  
        File file = fileChooser.getSelectedFile();  
        // Comprobamos que el archivo introducido existe  
        if (file.exists()) {  
            // Hacemos visible la ventana de carteras  
            ventanaCartera.setVisible(true);  
  
            // Leemos el archivo de cartera  
            Charset charset = Charset.forName("US-ASCII");  
            try (BufferedReader reader = Files.newBufferedReader(Paths.get(file.getCanonicalPath()), charset)) {  
                // Leemos el nombre de la cartera  
                String nombre = reader.readLine();  
                Boolean cargada = false;  
                for (int i = 0; i < gestorCarteras.getComponentCount(); i++) {  
                    // Comprobamos el nombre de las carteras cargadas en el gestor  
                    Cartera c = (Cartera) gestorCarteras.getComponentAt(i);  
                    String n = c.getWalletName();  
                    if (nombre.equals(n)) {  
                        // La cartera ya está cargada  
                        cargada = true;  
                        break;  
                    }  
                }  
                if (!cargada) {  
                    estado.setText("Cargando cartera...");  
                    String ubicacion = file.getAbsolutePath();  
                    Cartera cartera = new Cartera(nombre, ubicacion);  
                    gestorCarteras.add(nombre, cartera);  
                    gestorCarteras.setSelectedIndex(gestorCarteras.getComponentCount() - 1);  
  
                    String line;  
                    // Leemos el resto del archivo para rellenar las filas de la tabla  
                    while ((line = reader.readLine()) != null) {  
                        cartera.addRow(line);  
                    }  
                }  
            }  
        }  
    }  
}
```

```

        // Comprobamos si hay fechas de vencimiento caducadas en alguna opción
        if (cartera.deleteOldDates()) {
            JOptionPane.showMessageDialog(null,
                "En la cartera cargada habian opciones cuya fecha de vencimiento ha caducado."
                + "\nHan sido eliminadas de la cartera.",
                "Advertencia", JOptionPane.INFORMATION_MESSAGE);
        }
        // Calculamos el importe, la valoracion y el beneficio
        if (!sinConexion) {
            cartera.calculate(importe, valoracion, ganancia);
        }
        estado.setText("Listo");
    } else {
        JOptionPane.showMessageDialog(null,
            "La cartera seleccionada ya ha sido cargada.",
            "Error", JOptionPane.ERROR_MESSAGE);
    }
} catch (IOException x) {
    System.err.format("IOException: %s\n", x);
}
// Habilitamos la opción de Guardar y los botones para añadir opciones
opcionGuardar.setEnabled(true);
botonAddPut.setEnabled(true);
botonAddCall.setEnabled(true);
} else {
    // Si el fichero no existe se muestra en mensaje de error
    JOptionPane.showMessageDialog(null,
        "No existe una cartera con ese nombre en el directorio especificado.",
        "Error", JOptionPane.ERROR_MESSAGE);
}
}
}

```

- Opción Guardar:

Se muestra un mensaje para confirmar que el usuario desea guardar los cambios.

```

private void opcionGuardarActionPerformed(java.awt.event.ActionEvent evt) {
    Cartera cartera = (Cartera) gestorCarteras.getSelectedComponent();
    int retorno = JOptionPane.showConfirmDialog(null,
        "¿Desea guardar los cambios de "
        + cartera.getWalletName()
        + "?", "Guardar cambios de la cartera",
        JOptionPane.YES_NO_OPTION,
        JOptionPane.QUESTION_MESSAGE);
    if (retorno == 0) {
        cartera.save();
        cartera.setChanges(false);
    }
}
}

```



- Opción Cerrar:

La opción Cerrar y el evento al cerrar la ventana se manejan de la misma forma, por lo que sólo se explicará uno.

```
private void opcionCerrarActionPerformed(java.awt.event.ActionEvent evt) {
    // Cambiamos el estado de la aplicación
    cerrandoAplicacion = true;
    // Variable que controla si el usuario ha cancelado el proceso de cerrar
    boolean cancelado = false;
    int numeroCarteras = gestorCarteras.getComponentCount();
    while (numeroCarteras > 0){
        Cartera c = (Cartera) gestorCarteras.getComponentAt(0);
        // Comprobamos si se han producido cambios en esa cartera
        if (c.checkChanges()) {
            Object[] textoBotones = {"Guardar", "No guardar", "Cancelar"};
            int retorno = JOptionPane.showOptionDialog(null,
                "¿Desea guardar los cambios realizados en "
                + c.getWalletName()
                + "?", "Salir",
                JOptionPane.YES_NO_OPTION,
                JOptionPane.QUESTION_MESSAGE,
                null, textoBotones, textoBotones[2]);
            if (retorno == 0) {
                // Guardamos los cambios en la cartera y la cerramos
                c.save();
                gestorCarteras.remove(c);
                numeroCarteras--;
            }
            if (retorno == 1) {
                // Cerramos la cartera sin guardar
                gestorCarteras.remove(c);
                numeroCarteras--;
            }
            if (retorno == 2) {
                // Se cancela el proceso de cerrado
                cancelado = true;
                break;
            }
        } else {
            // Cerramos la cartera si no se han producido cambios
            gestorCarteras.remove(c);
            numeroCarteras--;
        }
    }
    if (!cancelado){
        // Si no se ha cancelado el proceso de cerrado,
        // se finaliza la ejecución
        System.exit(0);
    }
}
```

- Botón Añadir a la cartera:

Hay dos botones de este tipo, uno que añade las opciones de la tabla PUT a la cartera, y otro que hace exactamente lo mismo, pero refiriéndose a la tabla CALL. Por lo tanto sólo se explicará uno.

```
private void botonAddPutActionPerformed(java.awt.event.ActionEvent evt) {
    // Obtenemos la opción que se desea añadir a la cartera
    int row = TablaOpcionesPUT.getSelectedRow();
    Opcion opcion = listaPutFiltrada.get(row);

    // Comprobamos que la opción tenga un precio de compra válido
    if (opcion.Compra_Precio.equals("- ")){
        JOptionPane.showMessageDialog(null,
            "La opción seleccionada no tiene precio de compra. "
            + "\nNo se puede añadir a la cartera.",
            "Error", JOptionPane.ERROR_MESSAGE);
    } else {
        // Comprobamos que la opción tenga un precio de venta válido
        if (opcion.Venta_Precio.equals("- ")){
            JOptionPane.showMessageDialog(null,
                "La opción seleccionada no tiene precio de venta. "
                + "\nNo se puede añadir a la cartera.",
                "Error", JOptionPane.ERROR_MESSAGE);
        } else {
            Cartera cartera = (Cartera)gestorCarteras.getSelectedComponent();
            // Comprobamos en qué fila de la tabla está la opción pasada
            int fila = cartera.include(opcion);
            // Si la opción no está, se añade a la cartera
            if (fila == -1){
                String fechaActual = campoFecha.getText();
                fechaActual = adjustDate(fechaActual);
                String datos = "1 " + opcion.Tipo + " " +
                    opcion.Vencimiento.replaceAll(" ", "-") + " "
                    + opcion.Ejercicio.trim() + " "
                    + fechaActual + " " + opcion.Venta_Precio;
                // Añadimos la información a la cartera actual
                cartera.addRow(datos);
                cartera.calculate(importe, valoracion, ganancia);
            } else {
                // Si la opción ya estaba en la cartera,
                // aumentamos el número de opciones de ese tipo
                cartera.increaseCount(fila);
                cartera.calculate(importe, valoracion, ganancia);
            }
            cartera.setChanges(true);
        }
    }
}
```

- Lista desplegable de fechas de vencimiento:

Hay dos listas desplegables con las fechas de vencimiento de cada tabla, una para las opciones PUT y otra para las opciones CALL, con manejador de eventos de cambio de estado. Ambos manejadores de eventos están implementados de la misma forma, así que sólo se explicará uno.

```
private void comboBoxFechasPutItemStateChanged(java.awt.event.ItemEvent evt) {
    // Comprobamos el número de fechas de vencimiento de la lista desplegable
    if (comboBoxFechasPut.getItemCount() > 1) {
        int nOpciones = opciones.listaPut.size();
        // Limpiamos la lista PUT filtrada por fecha
        listaPutFiltrada.clear();
        String fechaEscogida = (String) comboBoxFechasPut.getSelectedItem();
        int coincidencias = 0;
        // Comprobamos el número de opciones cuya fecha de vencimiento coincide
        for (int i = 0; i < nOpciones; i++) {
            Opcion opcionPut = opciones.listaPut.get(i);
            if (fechaEscogida.equals("Todas") || fechaEscogida.equals(opcionPut.Vencimiento)) {
                coincidencias++;
            }
        }
        DefaultTableModel tableModelPut = (DefaultTableModel) TablaOpcionesPUT.getModel();
        tableModelPut.setRowCount(coincidencias);

        int fila = 0; // Fila de la tabla en la que introducir los datos
        for (int i = 0; i < nOpciones; i++) {
            Opcion opcionPut = opciones.listaPut.get(i);
            // Comprobamos si la fecha escogida en la lista desplegable coincide
            if (fechaEscogida.equals("Todas") || fechaEscogida.equals(opcionPut.Vencimiento)) {
                // Rellenamos la tabla con esa opción
                TablaOpcionesPUT.setValueAt(opcionPut.Ejercicio, fila, 0);
                TablaOpcionesPUT.setValueAt(opcionPut.Compra_Vol, fila, 1);
                TablaOpcionesPUT.setValueAt(opcionPut.Compra_Precio, fila, 2);
                TablaOpcionesPUT.setValueAt(opcionPut.Venta_Precio, fila, 3);
                TablaOpcionesPUT.setValueAt(opcionPut.Venta_Vol, fila, 4);
                TablaOpcionesPUT.setValueAt(opcionPut.Ultimo, fila, 5);
                TablaOpcionesPUT.setValueAt(opcionPut.Volumen, fila, 6);
                TablaOpcionesPUT.setValueAt(opcionPut.Hora, fila, 7);
                // Añadimos a la lista filtrada todas las opciones cuya fecha coincide
                listaPutFiltrada.add(opcionPut);
                fila++;
            }
        }
    }
}
```

- Gestor de carteras

Se ha implementado un manejador de evento de cambio de estado para el gestor de carteras, que es un *JTabbedPane*. Así, cada vez que se cambie la cartera activa, se recalcula la información relativa al importe invertido, la valoración actual de todas las opciones y los beneficios totales.

```
private void gestorCarterasStateChanged(javax.swing.event.ChangeEvent evt) {
    // Comprobamos si se no ha iniciado el proceso de cerrado
    if (!cerrandoAplicacion) {
        Cartera cartera = (Cartera) gestorCarteras.getSelectedComponent();
        if (cartera != null) {
            // Cada vez que se cambie de pestaña activa, se recalcula la información económica
            cartera.calculate(importe, valoracion, ganancia);
        }
    }
}
```

- Botón Cerrar cartera:

```
private void botonCloseActionPerformed(java.awt.event.ActionEvent evt) {
    Cartera cartera = (Cartera) gestorCarteras.getSelectedComponent();
    // Comprobamos si se han realizado cambios en la cartera seleccionada
    if (cartera.checkChanges()) {
        Object[] textoBotones = {"Guardar", "No guardar", "Cancelar"};
        int retorno = JOptionPane.showOptionDialog(null,
            "¿Desea guardar los cambios realizados en "
            + cartera.getWalletName()
            + "?", "Cerrar cartera",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null, textoBotones, textoBotones[2]);
        if (retorno == 0) {
            // Guardamos los cambios en la cartera y la cerramos
            cartera.save();
            gestorCarteras.remove(cartera);
        }
        if (retorno == 1) {
            // Cerramos la cartera
            gestorCarteras.remove(cartera);
        }
        // Si no quedan más carteras abiertas en el gestor de carteras,
        // se oculta el gestor y se deshabilitan los botones oportunos
        if (gestorCarteras.getComponentCount() == 0) {
            ventanaCartera.setVisible(false);
            opcionGuardar.setEnabled(false);
            botonAddPut.setEnabled(false);
            botonAddCall.setEnabled(false);
            estado.setText("Cartera sin cargar");
        }
    } else {
        // Si no se han realizado cambios, se cierra la cartera
        gestorCarteras.remove(cartera);
        if (gestorCarteras.getComponentCount() == 0) {
            // Si no quedan más carteras abiertas en el gestor de carteras,
            // se oculta el gestor y se deshabilitan los botones oportunos
            ventanaCartera.setVisible(false);
            opcionGuardar.setEnabled(false);
            botonAddPut.setEnabled(false);
            botonAddCall.setEnabled(false);
            estado.setText("Cartera sin cargar");
        }
    }
}
```

- Botón Eliminar opción de la cartera:

```
private void botonRemoveActionPerformed(java.awt.event.ActionEvent evt) {
    Cartera cartera = (Cartera)gestorCarteras.getSelectedComponent();
    // Comprobamos si el usuario ha seleccionado una fila
    if (cartera.getSelectedRow() == -1){
        JOptionPane.showMessageDialog(null,
            "Debe seleccionar una opción de la cartera.",
            "Error", JOptionPane.ERROR_MESSAGE);
    } else {
        int retorno = JOptionPane.showConfirmDialog(null,
            "¿Desea eliminar la opción seleccionada de "
            + cartera.getWalletName()
            + "?", "Eliminar opción de la cartera",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE);
        if (retorno == 0) {
            // Eliminamos la fila seleccionada
            cartera.removeRow(cartera.getSelectedRow());
            // Recalculamos la información económica
            cartera.calculate(importe, valoracion, ganancia);
        }
        // Establecemos que se han producido cambios en la cartera
        cartera.setChanges(true);
    }
}
```

- Botón Reconectar:

```
private void botonReconectarActionPerformed(java.awt.event.ActionEvent evt) {
    CollectData();
}
```

## 6.2. Funciones

Ahora se explicarán las funciones utilizadas a lo largo de todo el proceso.

- CollectData

Se han realizado unos pequeños cambios y adiciones en esta función.

```
private void CollectData() {
    // Actualiza la tabla de contado
    if (!contado.getSpot()) {
        JOptionPane.showMessageDialog(null,
            "No se ha podido establecer la conexión con los servidores."
            + "\nVerifique su conexión a Internet e inténtelo más tarde.",
            "Error de conexión", JOptionPane.ERROR_MESSAGE);
        // Si ha ocurrido un problema de conexión, se deshabilitan todas
        // las funciones de la aplicación y se muestra el botón de reconexión
        botonReconectar.setVisible(true);
        botonAddPut.setEnabled(false);
        botonAddCall.setEnabled(false);
        botonRemove.setEnabled(false);
        opcionAbrir.setEnabled(false);
        opcionNueva.setEnabled(false);
        comboBoxFechasCall.setEnabled(false);
        comboBoxFechasPut.setEnabled(false);
        estado.setText("Sin conexión");
        // Se cambia el estado de la aplicación
        sinConexion = true;
        return;
    } else {
        // Si hay conexión, comprobamos el número de carteras cargadas
        if (gestorCarteras.getComponentCount() > 0) {
            estado.setText("Listo");
            // Habilitamos los botones para añadir o eliminar opciones
            botonAddPut.setEnabled(true);
            botonAddCall.setEnabled(true);
            botonRemove.setEnabled(true);
        } else {
            estado.setText("Cartera sin cargar");
        }
        // Ocultamos el botón de reconexión
        botonReconectar.setVisible(false);
        // Habilitamos las opciones para crear o abrir carteras
        opcionAbrir.setEnabled(true);
        opcionNueva.setEnabled(true);
        // Habilitamos listas desplegables de fechas de vencimiento
        comboBoxFechasCall.setEnabled(true);
        comboBoxFechasPut.setEnabled(true);
        // Cambia el estado de la aplicación
        sinConexion = false;
    }
}
```

```

TableModel model = TablaContado.getModel();
TablaContado.setValueAt(contado.Spot, 0, 0);
TablaContado.setValueAt(contado.Anterior, 0, 2);
TablaContado.setValueAt(contado.Maximo, 0, 3);
TablaContado.setValueAt(contado.Minimo, 0, 4);
TablaContado.setValueAt(contado.Fecha, 0, 5);
TablaContado.setValueAt(contado.Hora, 0, 6);

Float diferencia = toFloat(contado.Diferencia);
// Coloreamos el texto dependiendo de la diferencia
if (diferencia > 0) {
    model.setValueAt("<html><font color='green'>" + diferencia + "</font></html>", 0, 1);
} else if (diferencia < 0) {
    model.setValueAt("<html><font color='red'>" + diferencia + "</font></html>", 0, 1);
} else {
    model.setValueAt("<html><font color='black'>" + diferencia + "</font></html>", 0, 1);
}

futuros.getFutures();
int nfuturos = futuros.Futuros.size();

DefaultTableModel tablemodel = (DefaultTableModel) TablaFuturos.getModel();
tablemodel.setRowCount(nfuturos);

for (int i = 0; i < nfuturos; i++) {
    Futuro f = futuros.Futuros.get(i);
    TablaFuturos.setValueAt(f.Vencimiento, i, 0);
    TablaFuturos.setValueAt(f.Compra_Vol, i, 1);
    TablaFuturos.setValueAt(f.Compra_Precio, i, 2);
    TablaFuturos.setValueAt(f.Venta_Precio, i, 3);
    TablaFuturos.setValueAt(f.Venta_Vol, i, 4);
    TablaFuturos.setValueAt(f.Ultimo, i, 5);
    TablaFuturos.setValueAt(f.Volumen, i, 6);
    TablaFuturos.setValueAt(f.Apertura, i, 7);
    TablaFuturos.setValueAt(f.Maximo, i, 8);
    TablaFuturos.setValueAt(f.Minimo, i, 9);
    TablaFuturos.setValueAt(f.Anterior, i, 10);
    TablaFuturos.setValueAt(f.Hora, i, 11);
}

```



```

// Opciones PUT
opciones.getOptions();
int nOpciones = opciones.listaPut.size();
DefaultTableModel tableModelPut = (DefaultTableModel) TablaOpcionesPUT.getModel();
tableModelPut.setRowCount(nOpciones);
String[] fechas = new String[0];

for (int i = 0; i < nOpciones; i++) {
    Opcion opcionPut = opciones.listaPut.get(i);
    TablaOpcionesPUT.setValueAt(opcionPut.Ejercicio, i, 0);
    TablaOpcionesPUT.setValueAt(opcionPut.Compra_Vol, i, 1);
    TablaOpcionesPUT.setValueAt(opcionPut.Compra_Precio, i, 2);
    TablaOpcionesPUT.setValueAt(opcionPut.Venta_Precio, i, 3);
    TablaOpcionesPUT.setValueAt(opcionPut.Venta_Vol, i, 4);
    TablaOpcionesPUT.setValueAt(opcionPut.Ultimo, i, 5);
    TablaOpcionesPUT.setValueAt(opcionPut.Volumen, i, 6);
    TablaOpcionesPUT.setValueAt(opcionPut.Hora, i, 7);

    // Inicializamos la lista filtrada
    listaPutFiltrada.add(opcionPut);

    // Comprobamos fechas para incluirlas a la lista desplegable
    Boolean aparece = false;
    for (String fecha : fechas) {
        if (opcionPut.Vencimiento.equals(fecha)) {
            aparece = true;
        }
    }
    if (!aparece) {
        if (fechas.length == 0) {
            fechas = new String[]{opcionPut.Vencimiento};
        } else {
            String[] fechasAux = new String[fechas.length + 1];
            System.arraycopy(fechas, 0, fechasAux, 0, fechas.length);
            fechasAux[fechas.length] = opcionPut.Vencimiento;
            fechas = fechasAux;
        }
    }
}

// Limpiamos la lista desplegable y lo actualizamos con las fechas
comboBoxFechasPut.removeAllItems();
comboBoxFechasPut.addItem("Todas");
// Rellenamos la lista desplegable con las fechas de vencimiento
for (String fecha : fechas) {
    comboBoxFechasPut.addItem(fecha);
}
comboBoxFechasPut.removeItemAt(1); // Borramos el item vacio

```



```

// Opciones CALL
nOpciones = opciones.listaCall.size();
DefaultTableModel tableModelCall = (DefaultTableModel) TablaOpcionesCALL.getModel();
tableModelCall.setRowCount(nOpciones);
fechas = new String[0];

for (int i = 0; i < nOpciones; i++) {
    Opcion opcionCall = opciones.listaCall.get(i);
    TablaOpcionesCALL.setValueAt(opcionCall.Ejercicio, i, 0);
    TablaOpcionesCALL.setValueAt(opcionCall.Compra_Vol, i, 1);
    TablaOpcionesCALL.setValueAt(opcionCall.Compra_Precio, i, 2);
    TablaOpcionesCALL.setValueAt(opcionCall.Venta_Precio, i, 3);
    TablaOpcionesCALL.setValueAt(opcionCall.Venta_Vol, i, 4);
    TablaOpcionesCALL.setValueAt(opcionCall.Ultimo, i, 5);
    TablaOpcionesCALL.setValueAt(opcionCall.Volumen, i, 6);
    TablaOpcionesCALL.setValueAt(opcionCall.Hora, i, 7);

    // Inicializamos la lista filtrada
    listaCallFiltrada.add(opcionCall);

    Boolean aparece = false;
    // Comprobamos si la fecha de vencimiento de la opción actual
    // está incluida en el vector de fechas
    for (String fecha : fechas) {
        if (opcionCall.Vencimiento.equals(fecha)) {
            aparece = true;
        }
    }
    if (!aparece) {
        // Si la fecha de vencimiento de la opción actual no
        // está incluida en el vector, la incluimos
        if (fechas.length == 0) {
            fechas = new String[]{opcionCall.Vencimiento};
        } else {
            String[] fechasAux = new String[fechas.length + 1];
            System.arraycopy(fechas, 0, fechasAux, 0, fechas.length);
            fechasAux[fechas.length] = opcionCall.Vencimiento;
            fechas = fechasAux;
        }
    }
}

// Limpiamos el comboBox y lo actualizamos con las fechas
comboBoxFechasCall.removeAllItems();
comboBoxFechasCall.addItem("Todas");
// Rellenamos el comboBox con las fechas de vencimiento
for (String fecha : fechas) {
    comboBoxFechasCall.addItem(fecha);
}
comboBoxFechasCall.removeItemAt(1); // Borramos el item vacio
}

```

- adjustDate

Esta función toma una fecha como *String* en formato DD/MM/AAAA y la convierte a formato DD-MMM-AAAA. Por ejemplo, la fecha 23/01/2016 la convierte a 23-ene-2016. Se utiliza para mostrar la fecha en la tabla.

```
public String adjustDate(String date) {
    String fecha = date.substring(0, date.indexOf(" "));
    // Cogemos el mes de la fecha
    String mes = fecha.split("/")[1];
    if (mes.equals("01")) {
        mes = "ene";
    }
    if (mes.equals("02")) {
        mes = "feb";
    }
    if (mes.equals("03")) {
        mes = "mar";
    }
    if (mes.equals("04")) {
        mes = "abr";
    }
    if (mes.equals("05")) {
        mes = "may";
    }
    if (mes.equals("06")) {
        mes = "jun";
    }
    if (mes.equals("07")) {
        mes = "jul";
    }
    if (mes.equals("08")) {
        mes = "ago";
    }
    if (mes.equals("09")) {
        mes = "sep";
    }
    if (mes.equals("10")) {
        mes = "oct";
    }
    if (mes.equals("11")) {
        mes = "nov";
    }
    if (mes.equals("12")) {
        mes = "dic";
    }
    // Concatenamos todo separando con guiones
    fecha = fecha.split("/")[0] + "-" + mes + "-" + fecha.split("/")[2];
    return fecha;
}
```

### 6.3. Clase Cartera

Esta clase es la que almacena la tabla en la que están guardadas las opciones que el usuario irá almacenando, y la que contiene todos los métodos aplicables a la cartera.

- Variables y constructor:

```
public class Cartera extends javax.swing.JPanel {

    private javax.swing.JScrollPane scroll;
    private javax.swing.JTable tablaCartera;
    private final String nombre;
    private final String ubicacion;
    private int ultimaFila;
    private boolean cambiosRealizados;
    private final MEFF_Opciones opciones = new MEFF_Opciones();

    public Cartera(String n, String u) {
        nombre = n;
        ubicacion = u;
        ultimaFila = 0;
        cambiosRealizados = false;
        initComponents();
    }
}
```

La variable **scroll** es el panel de barras de desplazamiento que incluirá la tabla **tablaCartera**.

La variable **nombre** contiene el nombre que el usuario asignó a la cartera, **ubicacion** es la ubicación donde está almacenado el fichero de la cartera en el momento de cargarla o crearla.

La variable **ultimaFila** indica cual es la posición del primer espacio vacío de la tabla.

La variable **cambiosRealizados** indica el estado de la cartera para saber si se debe guardar o no.

- initComponents

Aquí creamos los objetos necesarios y les aplicamos el estilo de la interfaz.

```
private void initComponents() {
    scroll = new javax.swing.JScrollPane();
    // Creamos un vector con las pistas que nos gustaría añadir a la cabecera de la tabla
    String[] columnToolTips = {
        "Número de opciones de este tipo.",
        "Tipo de opción.",
        "Fecha de vencimiento.",
        "Precio de Ejercicio.",
        "Fecha de incorporación de la opción a la cartera.",
        "Importe de compra en el mercado al que estaba la opción al añadirla a la cartera.",
        "Precio de compra actual de la opción.",
        "Ganancia individual de la opción."};
    // Creamos la tabla y le añadimos las pistas informativas
    tablaCartera = new javax.swing.JTable(new DefaultTableModel()) {
        @Override
        protected JTableHeader createDefaultTableHeader() {
            return new JTableHeader(columnModel) {
                @Override
                public String getToolTipText(MouseEvent e) {
                    String tip = null;
                    java.awt.Point p = e.getPoint();
                    int index = columnModel.getColumnIndexAtX(p.x);
                    int realIndex
                        = columnModel.getColumn(index).getModelIndex();
                    return columnToolTips[realIndex];
                }
            };
        }
    };

    // Creamos el modelo de la tabla y le añadimos las columnas deseadas
    DefaultTableModel model = new DefaultTableModel();
    model.addColumn("N. opciones");
    model.addColumn("Tipo");
    model.addColumn("Fecha vencim.");
    model.addColumn("P. Ejercicio");
    model.addColumn("Fecha incor.");
    model.addColumn("Importe");
    model.addColumn("P. Compra actual");
    model.addColumn("Ganancia");
    tablaCartera.setModel(model);
}
```

```

// Cambiamos el ancho de las dos primeras columnas
tablaCartera.getColumnModel().getColumn(0).setPreferredWidth(60);
tablaCartera.getColumnModel().getColumn(1).setPreferredWidth(40);

// Coloreamos las filas de la tabla con colores alternados
setCellRender(tablaCartera);
// Sólo permitimos selección simple de filas
tablaCartera.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
add(scroll);
scroll.setViewportView(tablaCartera);

// Diseño que debe adoptar el scroll al añadirse al gestor de carteras
javax.swing.GroupLayout carteraLayout = new javax.swing.GroupLayout(this);
this.setLayout(carteraLayout);
carteraLayout.setHorizontalGroup(carteraLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, carteraLayout.createSequentialGroup()
        .addComponent(scroll, javax.swing.GroupLayout.DEFAULT_SIZE, 557, Short.MAX_VALUE)
    ));
carteraLayout.setVerticalGroup(carteraLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, carteraLayout.createSequentialGroup()
        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        .addGroup(carteraLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(scroll, javax.swing.GroupLayout.DEFAULT_SIZE, 557, Short.MAX_VALUE))
    ));
}

```

- addRow

```

// Añade la línea a la tabla
public void addRow(String line) {
    // Comprobamos campos y los vamos introduciendo en la tabla
    String[] campos = line.split(" ");
    DefaultTableModel model = (DefaultTableModel) tablaCartera.getModel();
    model.addRow(campos);
    ultimaFila++;
}

```

- removeRow

```

// Elimina la fila indicada de la tabla
public void removeRow(int indice) {
    DefaultTableModel model = (DefaultTableModel) tablaCartera.getModel();
    model.removeRow(indice);
    ultimaFila--;
}

```

- getSelectedRow

```

// Devuelve la fila seleccionada de la tabla
public int getSelectedRow() {
    return tablaCartera.getSelectedRow();
}

```

- setCellRender

```
// Colorea las filas de la tabla
public void setCellRender(JTable table) {
    Enumeration<TableColumn> en = table.getColumnModel().getColumns();
    while (en.hasMoreElements()) {
        TableColumn tc = en.nextElement();
        tc.setCellRenderer(new CellRenderer());
    }
}
```

- getLastRow

```
// Devuelve la primera fila de la tabla en la que no hay datos
public int getLastRow() {
    return ultimaFila;
}
```

- getWalletName

```
// Devuelve el nombre de la cartera
public String getWalletName() {
    return nombre;
}
```

- save

```
// Guarda los cambios de la cartera en el fichero
public void save() {
    int row = tablaCartera.getRowCount();
    String line = "";
    PrintWriter writer;
    try {
        // Creamos un nuevo fichero y escribimos el nombre
        writer = new PrintWriter(ubicacion, "UTF-8");
        writer.println(nombre);
        // Leemos toda la tabla
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < 6; j++) {
                line = line + tablaCartera.getValueAt(i, j) + " ";
            }
            // Una vez haya leído toda la fila, la guardamos en el fichero
            writer.println(line);
            line = "";
        }
        writer.close();
    } catch (FileNotFoundException | UnsupportedEncodingException ex) {
        Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

- checkChanges

```
// Comprueba si se han producido cambios en la cartera
public boolean checkChanges() {
    return cambiosRealizados;
}
```

- setChanges

```
// Establece que se han producido cambios en la cartera
public void setChanges(boolean b) {
    cambiosRealizados = b;
}
```

- include

```
// Comprueba en qué fila de la tabla está la opción pasada
public int include(Opcion opcion) {
    for (int i = 0; i < tablaCartera.getRowCount(); i++) {
        // Comprobamos si la opcion ya está dentro de la cartera
        if (opcion.Ejercicio.trim().equals(tablaCartera.getValueAt(i, 3))) {
            if (opcion.Vencimiento.replaceAll(" ", "-").equals(tablaCartera.getValueAt(i, 2))) {
                if (opcion.Tipo.equals(tablaCartera.getValueAt(i, 1))) {
                    return i;
                }
            }
        }
    }
    // No se encontró dentro de la cartera
    return -1;
}
```

- increaseCount

```
// Incrementa la cantidad de opciones de ese tipo de una fila dada
public void increaseCount(int row) {
    int num = Integer.parseInt((String) tablaCartera.getValueAt(row, 0));
    num++;
    String dato = Integer.toString(num);
    tablaCartera.setValueAt(dato, row, 0);
}
```

- checkDate

```
// Comprueba si la fecha pasada es anterior o posterior a la actual
private boolean checkDate(String date) {
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
    Date currentDate = new Date();
    String current = dateFormat.format(currentDate);
    return sortDate(date).compareTo(current) < 0;
}
```

- deleteOldDates

```
// Borra las filas de la tabla cuya fecha de vencimiento ha caducado e informa de ello
public boolean deleteOldDates() {
    for (int i = 0; i < tablaCartera.getRowCount(); i++) {
        if (checkDate((String) tablaCartera.getValueAt(i, 2))) {
            removeRow(i);
            return true;
        }
    }
    return false;
}
```

- sortDate

Esta función es la inversa de la función *adjustDate*.

```
// Dada una fecha en formato "DD-MMM-AAAA" nos la ordena a formato "AAAA-MM-DD"
private String sortDate(String date) {
    String mes = date.split("-")[1];
    switch (mes) {
        case "ene":
            mes = "01";
            break;
        case "feb":
            mes = "02";
            break;
        case "mar":
            mes = "03";
            break;
        case "abr":
            mes = "04";
            break;
        case "may":
            mes = "05";
            break;
        case "jun":
            mes = "06";
            break;
        case "jul":
            mes = "07";
            break;
        case "ago":
            mes = "08";
            break;
        case "sep":
            mes = "09";
            break;
        case "oct":
            mes = "10";
            break;
        case "nov":
            mes = "11";
            break;
        case "dic":
            mes = "12";
            break;
        default:
            break;
    }
    String fecha = date.split("-")[2] + "-" + mes + "-" + date.split("-")[0];
    return fecha;
}
```



- calculate

Esta función calcula la información económica de interés para el usuario. Más concretamente, obtiene el precio de compra actual de todas las opciones, el beneficio resultante de cada opción, el importe invertido en la adquisición de la cartera, la valoración actual de toda la cartera y el beneficio total.

```
public void calculate(JLabel labelImporte, JLabel labelValoracion, JLabel labelGanancia) {
    // Calculamos los beneficios individuales para cada opción
    for (int i = 0; i < tablaCartera.getRowCount(); i++) {
        // Obtenemos el precio de compra
        float precioCompra = Float.parseFloat(((String) tablaCartera.getValueAt(i, 5)).replace(".", "").replace(",", "."));
        opciones.getOptions();
        // Comprobamos si la opción es de tipo PUT
        if (((String) tablaCartera.getValueAt(i, 1)).equals("PUT")) {
            for (Opcion opcionPut : opciones.listaPut) {
                // Comprobamos que el precio de ejercicio y la fecha de vencimiento sean iguales
                if (opcionPut.Ejercicio.trim().equals(tablaCartera.getValueAt(i, 3))) {
                    if (opcionPut.Vencimiento.replaceAll(" ", "-").equals(tablaCartera.getValueAt(i, 2))) {
                        // Obtenemos el precio de compra actual
                        float precioActual = Float.parseFloat((opcionPut.Compra_Precio).replace(".", "").replace(",", "."));
                        String actual = precioActual + "";
                        // Lo introducimos en la tabla
                        tablaCartera.setValueAt(actual, i, 6);
                        // Calculamos el beneficio obtenido
                        float beneficio = precioActual - precioCompra;
                        String dato = beneficio + "";
                        // Coloreamos el texto dependiendo del valor del beneficio
                        if (beneficio > 0) {
                            tablaCartera.setValueAt("<html><font color='green'>" + dato + "</font></html>", i, 7);
                        } else if (beneficio < 0) {
                            tablaCartera.setValueAt("<html><font color='red'>" + dato + "</font></html>", i, 7);
                        } else {
                            tablaCartera.setValueAt("<html><font color='black'>" + dato + "</font></html>", i, 7);
                        }
                    }
                }
            }
        }
    }
}

} else {
    // Comprobamos si la opción es de tipo CALL
    if (((String) tablaCartera.getValueAt(i, 1)).equals("CALL")) {
        for (Opcion opcionCall : opciones.listaCall) {
            // Comprobamos que el precio de ejercicio y la fecha de vencimiento sean iguales
            if (opcionCall.Ejercicio.trim().equals(tablaCartera.getValueAt(i, 3))) {
                if (opcionCall.Vencimiento.replaceAll(" ", "-").equals(tablaCartera.getValueAt(i, 2))) {
                    // Obtenemos el precio de compra actual
                    float precioActual = Float.parseFloat((opcionCall.Compra_Precio).replace(".", "").replace(",", "."));
                    String actual = precioActual + "";
                    // Lo introducimos en la tabla
                    tablaCartera.setValueAt(actual, i, 6);
                    // Calculamos el beneficio obtenido
                    float beneficio = precioActual - precioCompra;
                    String dato = beneficio + "";
                    // Coloreamos el texto dependiendo del valor del beneficio
                    if (beneficio > 0) {
                        tablaCartera.setValueAt("<html><font color='green'>" + dato + "</font></html>", i, 7);
                    } else if (beneficio < 0) {
                        tablaCartera.setValueAt("<html><font color='red'>" + dato + "</font></html>", i, 7);
                    } else {
                        tablaCartera.setValueAt("<html><font color='black'>" + dato + "</font></html>", i, 7);
                    }
                }
            }
        }
    }
}
}
```

```

// Calculamos el importe invertido, la valoración total de la cartera y la ganancia total
float invertido = 0;
float valoracion = 0;
float ganancia = 0;
for (int i = 0; i < tablaCartera.getRowCount(); i++) {
    // Importe invertido en la adquisición de la cartera que será la suma
    // de los precios de compra multiplicado respectivamente por el número de opciones
    invertido = invertido + (Float.parseFloat(
        ((String) tablaCartera.getValueAt(i, 5)).replace(".", "").replace(",", ".")
        * Float.parseFloat((String) tablaCartera.getValueAt(i, 0))));
    if (tablaCartera.getValueAt(i, 6) != null) {
        // Valoración actual de toda la cartera
        valoracion = valoracion + Float.parseFloat((String) tablaCartera.getValueAt(i, 6));
    } else {
        tablaCartera.setValueAt("<html><font color='blue'>"
            + "Error al buscar esta opción</font></html>", i, 6);
    }
    // Ganancia total
    ganancia = valoracion - invertido;
}
// Escribimos en las etiquetas los valores obtenidos
labelImporte.setText(Float.toString(invertido) + " €");
labelValoracion.setText(Float.toString(valoracion) + " €");

// Coloreamos el texto dependiendo si el beneficio es positivo o negativo
if (ganancia > 0) {
    labelGanancia.setText("<html><font color='green'><b>"
        + Float.toString(ganancia) + " €</b></font></html>");
} else if (ganancia < 0) {
    labelGanancia.setText("<html><font color='red'><b>"
        + Float.toString(ganancia) + " €</b></font></html>");
} else {
    labelGanancia.setText("<html><font color='black'><b>"
        + Float.toString(ganancia) + " €</b></font></html>");
}
}
}

```

#### 6.4. Clase FiltroCartera

Esta clase sirve para que el selector de ficheros sólo muestre los ficheros con una determinada extensión.

```
public class FiltroCartera extends FileFilter {

    @Override
    public boolean accept(File f) {
        if (f.isDirectory()) {
            return true;
        }
        String filename = f.getName();
        int dot = filename.lastIndexOf('.');
        String extension = filename.substring(dot + 1);

        if (extension != null) {
            if (extension.equals("car")) {
                return true;
            } else {
                return false;
            }
        }
        return false;
    }

    @Override
    public String getDescription() {
        return "Ficheros de cartera (*.car)";
    }
}
```

## 7. *Material entregado*

Junto a este informe, se adjunta en el archivo *.zip* la carpeta con el proyecto realizado en **Netbeans** y el archivo ejecutable *.jar* en la carpeta *dist* para una fácil y rápida ejecución, además del documento de presentación de la defensa pública realizada.