MoMiMeef (Monitoring MiniIbex35 from Meff web)

LAURA DEL PINO DÍAZ Y AITOR DEL PINO SAAVEDRA HERNÁNDEZ – GRADO EN INGENIERÍA INFORMÁTICA – MENCIÓN EN COMPUTACIÓN

Métodos numéricos para la computación.

índice

|  |  |
| --- | --- |
| Objetivo | 2 |
| Diseño conceptual | 3 |
| Diálogo de inicio | 5 |
| Menú superior | 7 |
| Pestañas estáticas | 12 |
| Pestañas de cartera |  |
| Conclusión | **11** |

oBJETIVO

El objetivo de este trabajo de curso es implementar una aplicación que monitorice los cambios de valor de los distintos productos financieros que aparecen en la página del MEFF, a saber: opciones del tipo CALL y PUT, futuros y contado.

Además se permitirá agrupar las opciones en carteras para que el usuario pueda tener separadas las opciones que ha adquirido y un resumen de los principales factores de la misma como pueden ser la inversión, la ganancia y el valor actual.

La aplicación la hemos bautizado como MoMiMeef, que viene de la descripción en inglés de la misma ***Mo****nitoring* ***Mi****niIbex-35 from* ***Me****ff WebSite using WebMining*  donde la última parte ha sido una licencia artística que nos hemos tomado para hacer del nombre más divertido y que se le quede al usuario.

diseño conceptual

El diseño preliminar, el cual estaba basado en un escritorio interior con múltiples ventanas, lo hemos sustituido por uno con un gestor de pestañas porque nos basamos en el conocimiento de que el ser humano solo puede concentrarse en una sola cosa por ello hemos agrupado la información y la hemos puesto cada una en una pestaña.

El diseño conceptual es el siguiente:

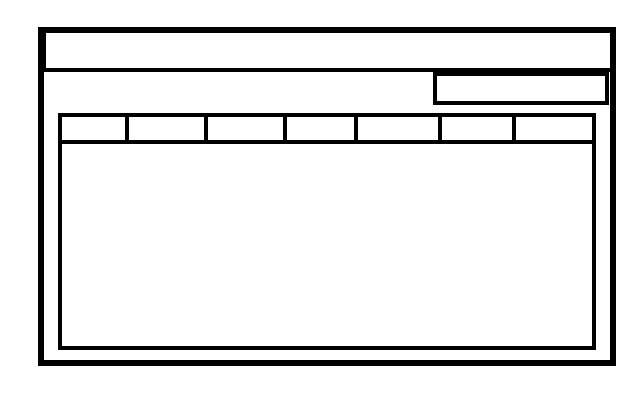


Ilustración 1: Diseño conceptual

Que tras implementarse queda con el siguiente aspecto:

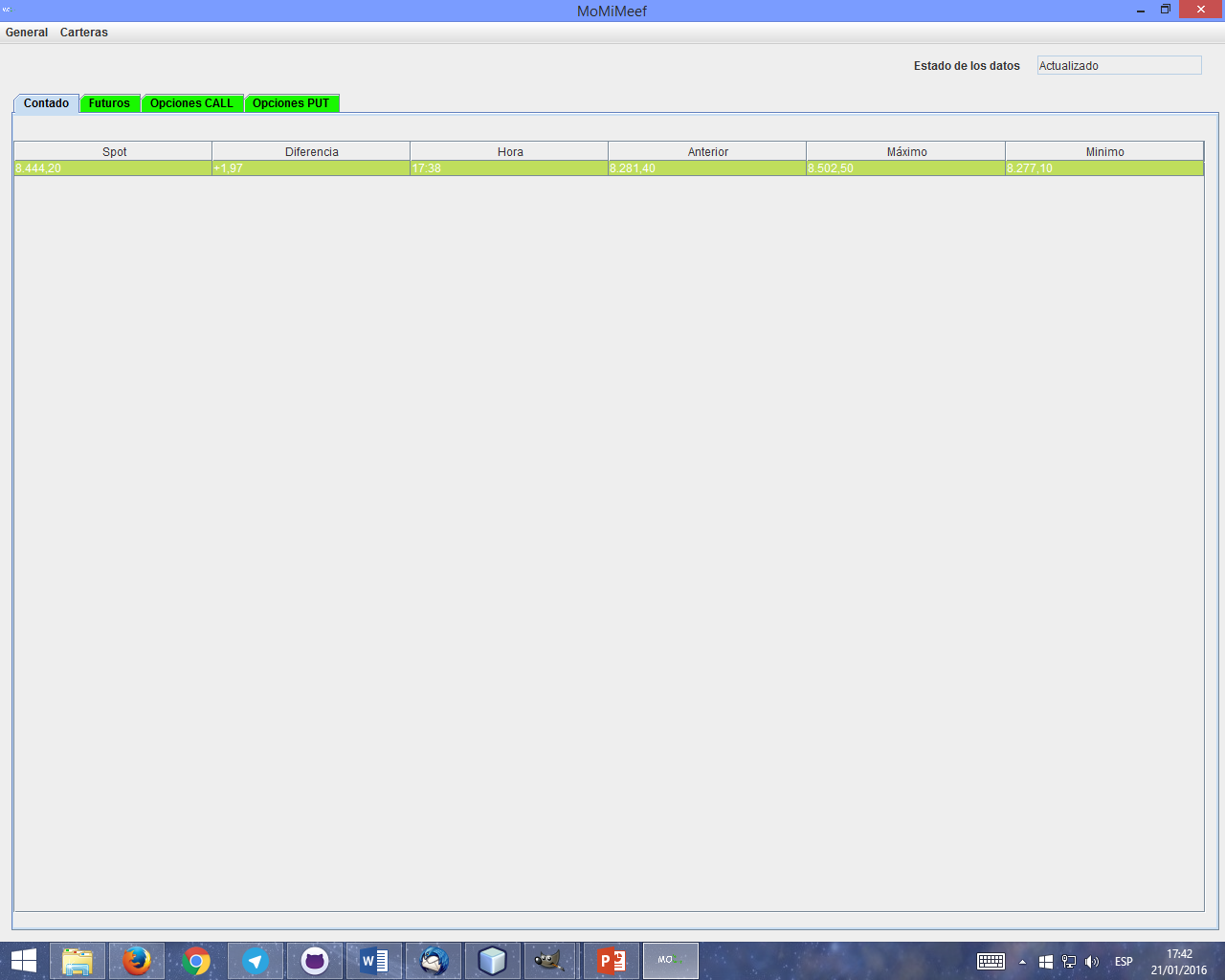


Ilustración 2: Pantalla principal tras la implementación.

Además de esta pantalla principal, la aplicación tiene tres diálogos adicionales: uno de inicio mientras se cargan los datos, otro de gestión de las carteras y otro de información de los creadores con una breve descripción de los objetivos de la aplicación.

diálogo de inicio

El diálogo de inicio es el primer componente de la interfaz de usuario que se muestra, tiene el siguiente aspecto:

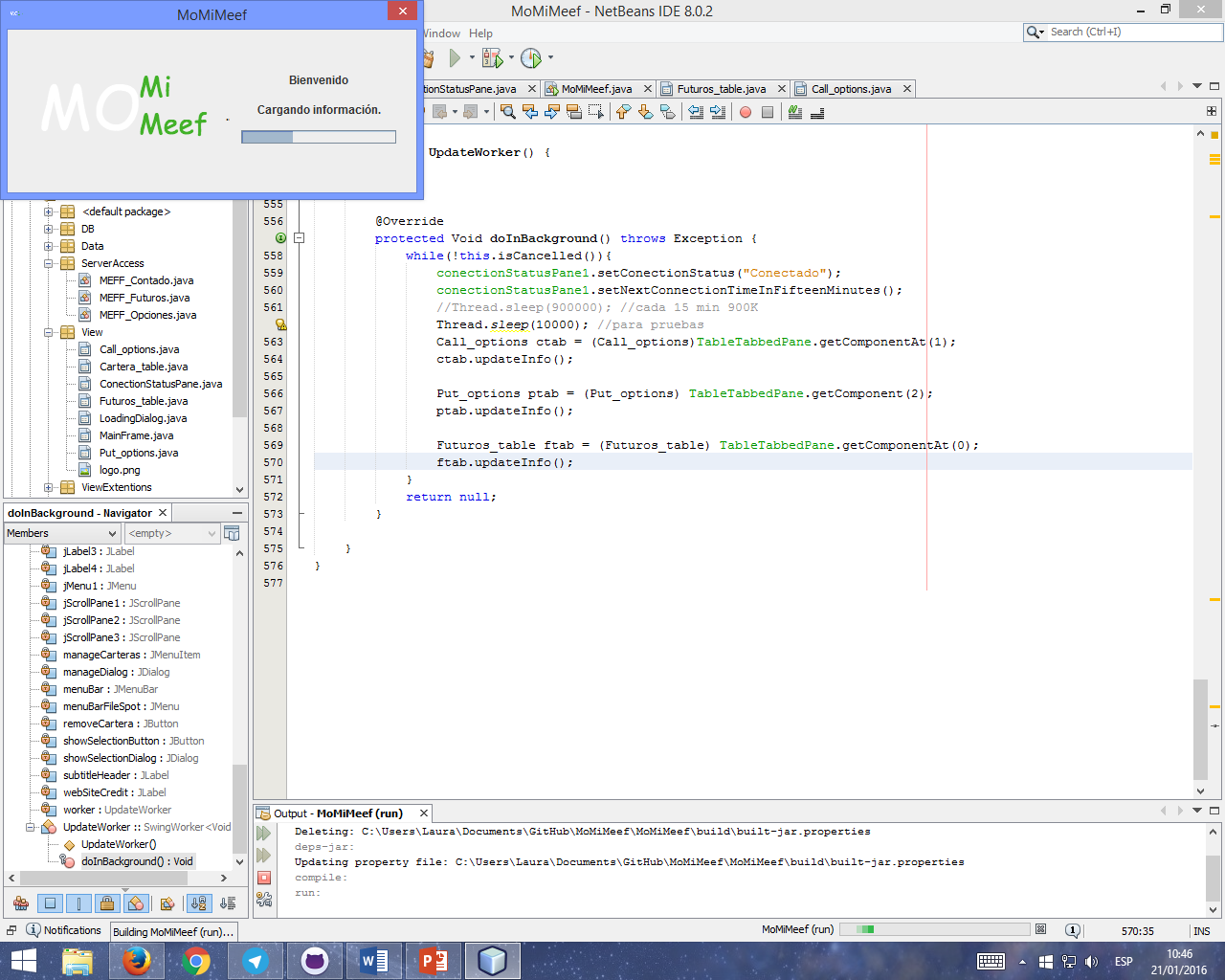


Ilustración 3: Diálogo de inicio

Su función es la de indicar al usuario que la aplicación se ha iniciado y que se está cargando la información, así como lo que resta por cargar para que pueda ser utilizada.

El diseño de este diálogo lo encontramos en la clase “LoadingDialog” que tiene las siguientes funciones:

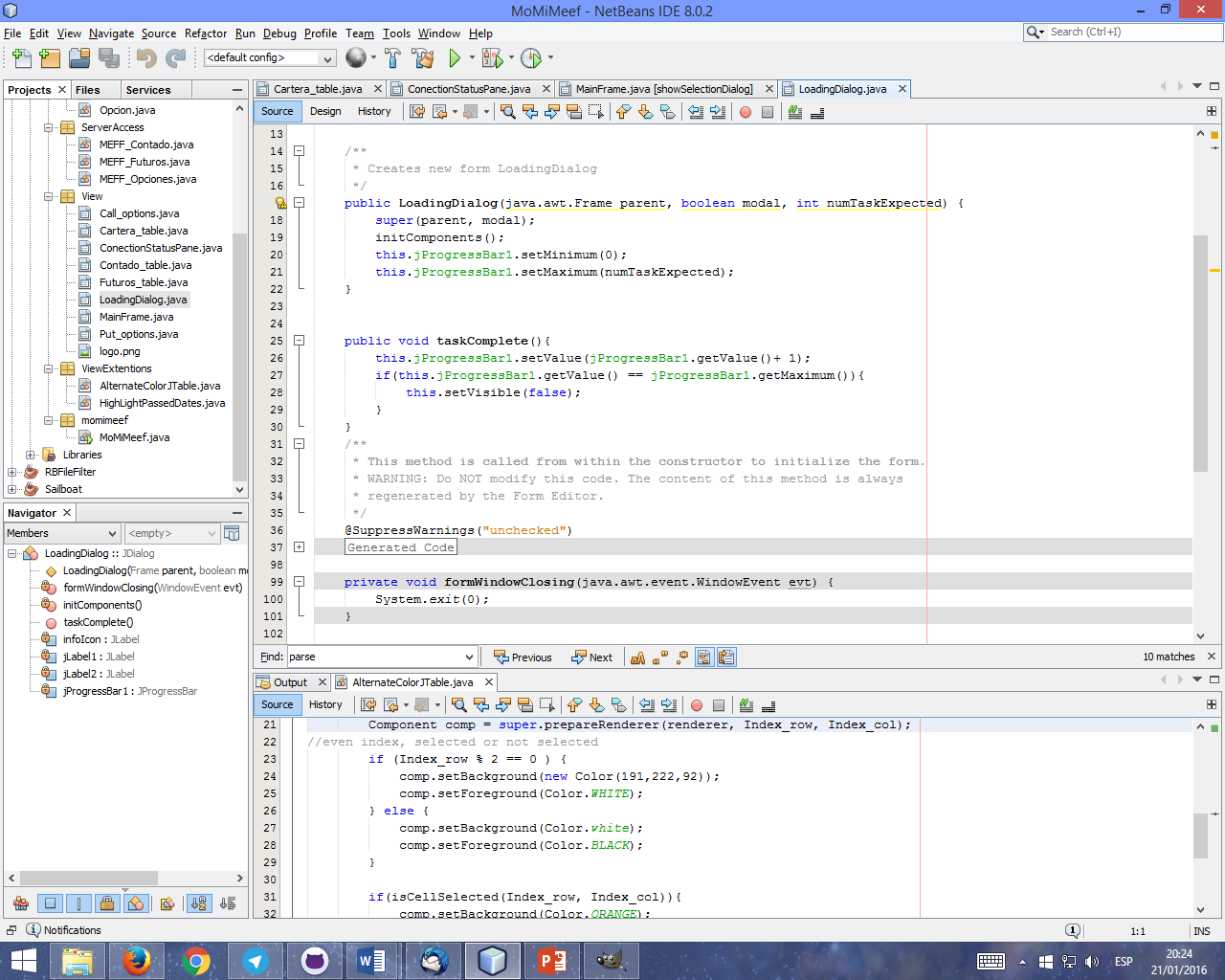


Ilustración 4: Clase dialogo de carga

* Constructor
* Método taskComplete: que sirve para indicarle que ha habido progresos que tienen que ser representados en la barra de progresos.
* Manejador de evento del botón de salida.

Al constructor se le pasan tres parámetros: el componente de la interfaz que es su padre, si es modal, es decir que solo se puede hacer clic en sus componentes y el número de tareas que se espera representar con la barra de progreso. Con este último parámetro se establece el límite superior de la barra de progresos.

El método “taskComplete” incrementa en una unidad a la barra de tareas y en caso de que está completa deja de mostrar el diálogo de inicio.

Por último la única tarea del manejador del evento de cierre del diálogo, cierra la aplicación en caso de que se aborte la carga.

Este elemento se genera en el constructor del marco principal de la aplicación el que indica que tarea ya ha realizado, entendiendo como tarea la carga de una pestaña estática, que es como hemos denominado a aquellas pestañas que contienen las tablas de los tipos de información básica de la aplicación. Veámoslo en el código:

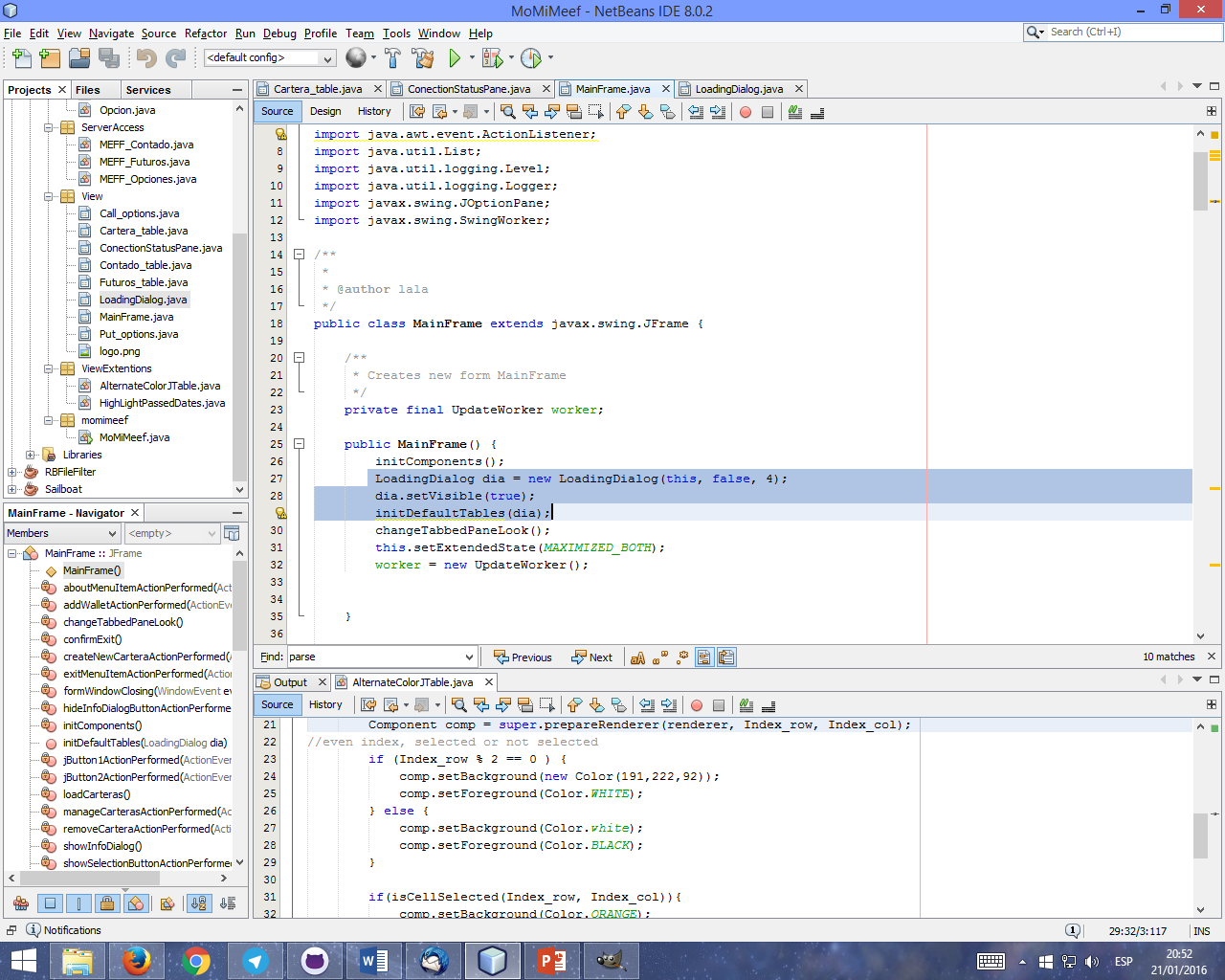


Ilustración 5: Constructor del marco principal.

En la parte resaltada se muestra la generación del diálogo y la llamada al método en el que se carga las pestañas estáticas como se muestra a continuación:

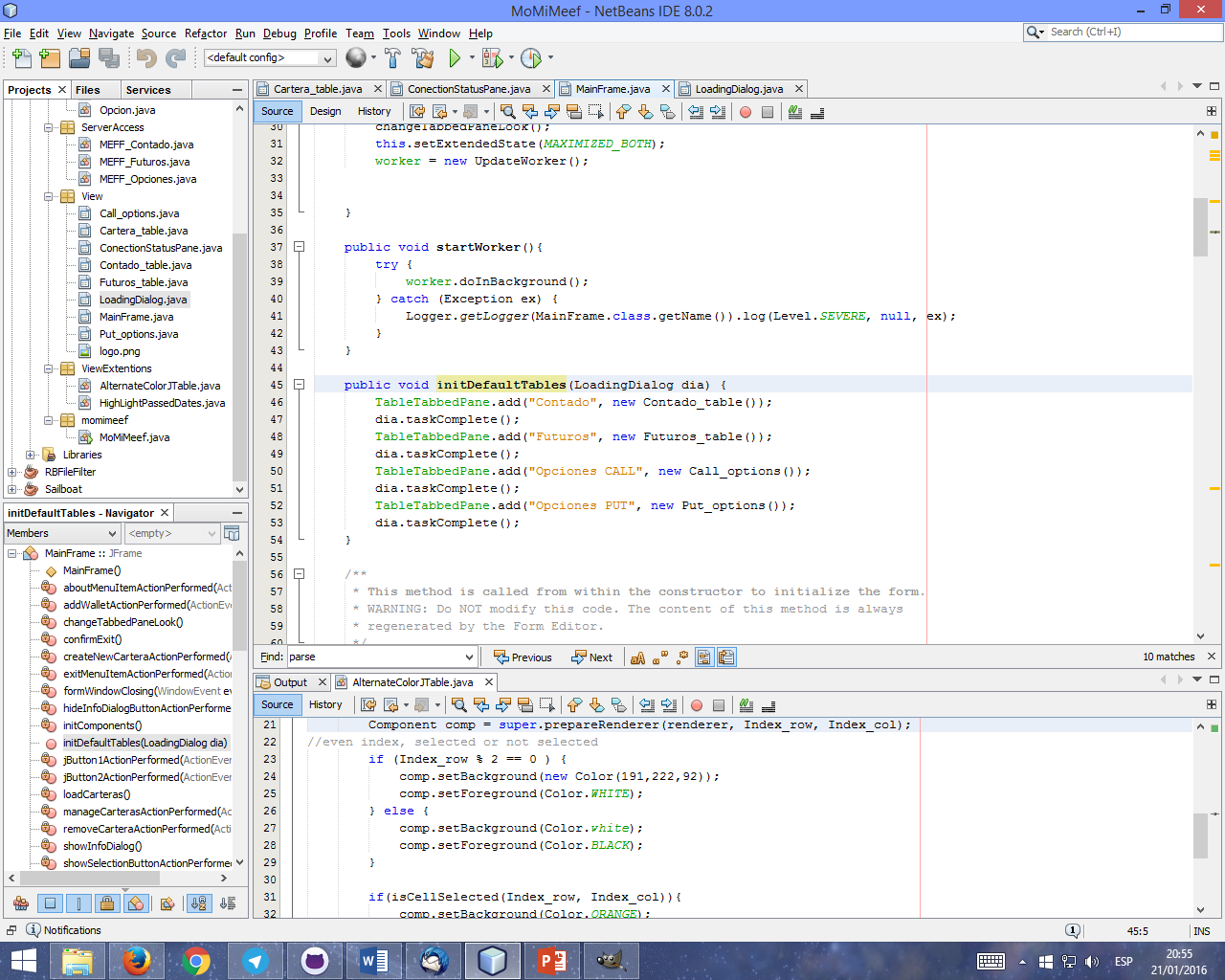


Ilustración 6: Carga de las pestañas.

Menú superior

MoMiMeef tiene un menú superior con dos elementos desplegables “General” y “Carteras”.

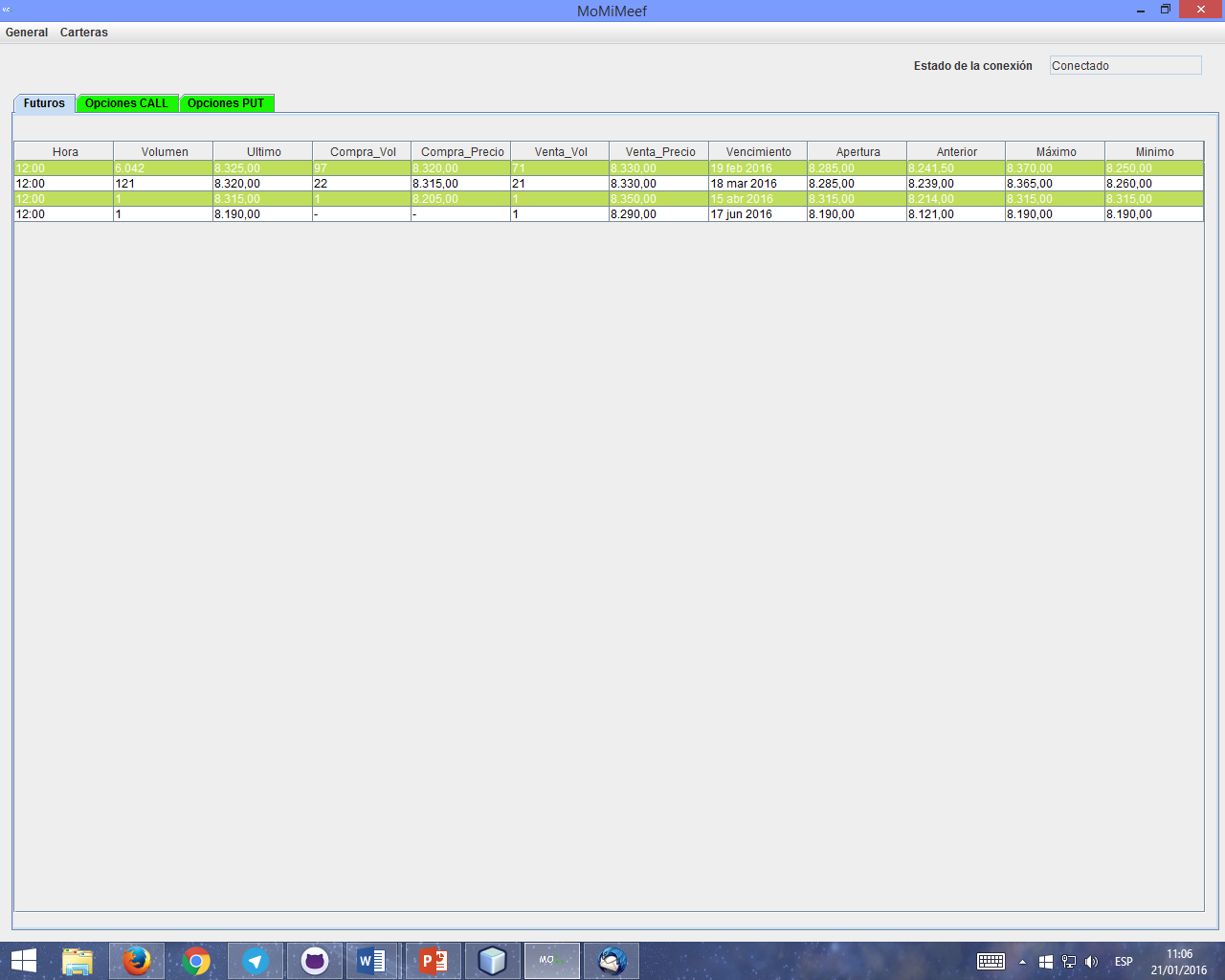


Ilustración 7: Menú superior.

“General” reúne dos elementos “Sobre MoMiMeef” que despliega un diálogo que muestra la información sobre nuestro grupo de desarrollo una breve descripción del objetivo de la aplicación así como el enlace al repositorio donde está el código. Tiene el siguiente aspecto:

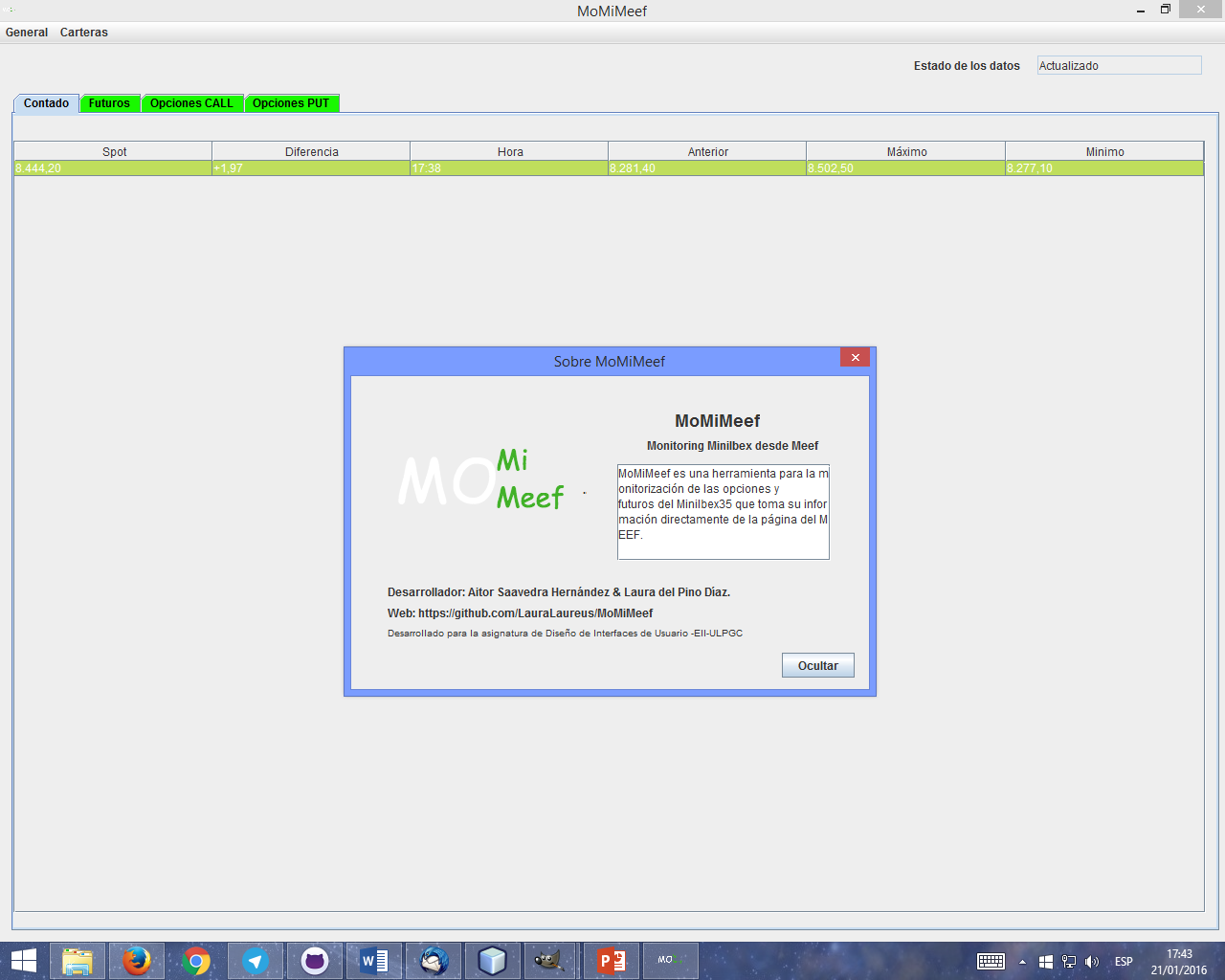


Ilustración 8: Diálogo "Sobre MoMiMeef"

El segundo elemento de este primer elemento desplegable es un elemento de salida que termina con la aplicación. Como bien hemos aprendido en las prácticas de la asignatura al hacer clic en este elemento pedimos la confirmación de la salida con el siguiente JOptionPane:

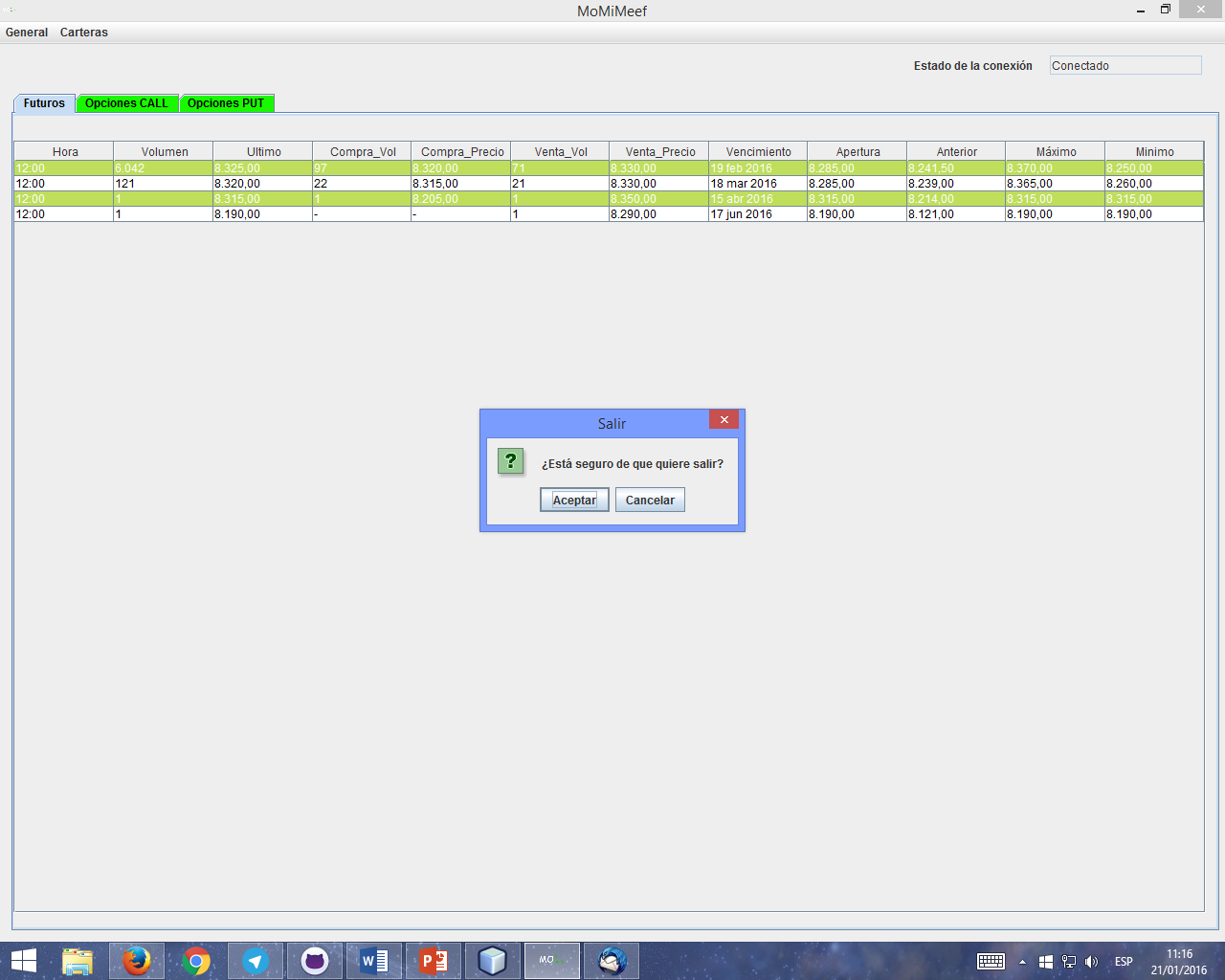


Ilustración 9: JOptionPane de confirmación de salida.

Esto lo hemos implementado con los siguientes manejadores de eventos:



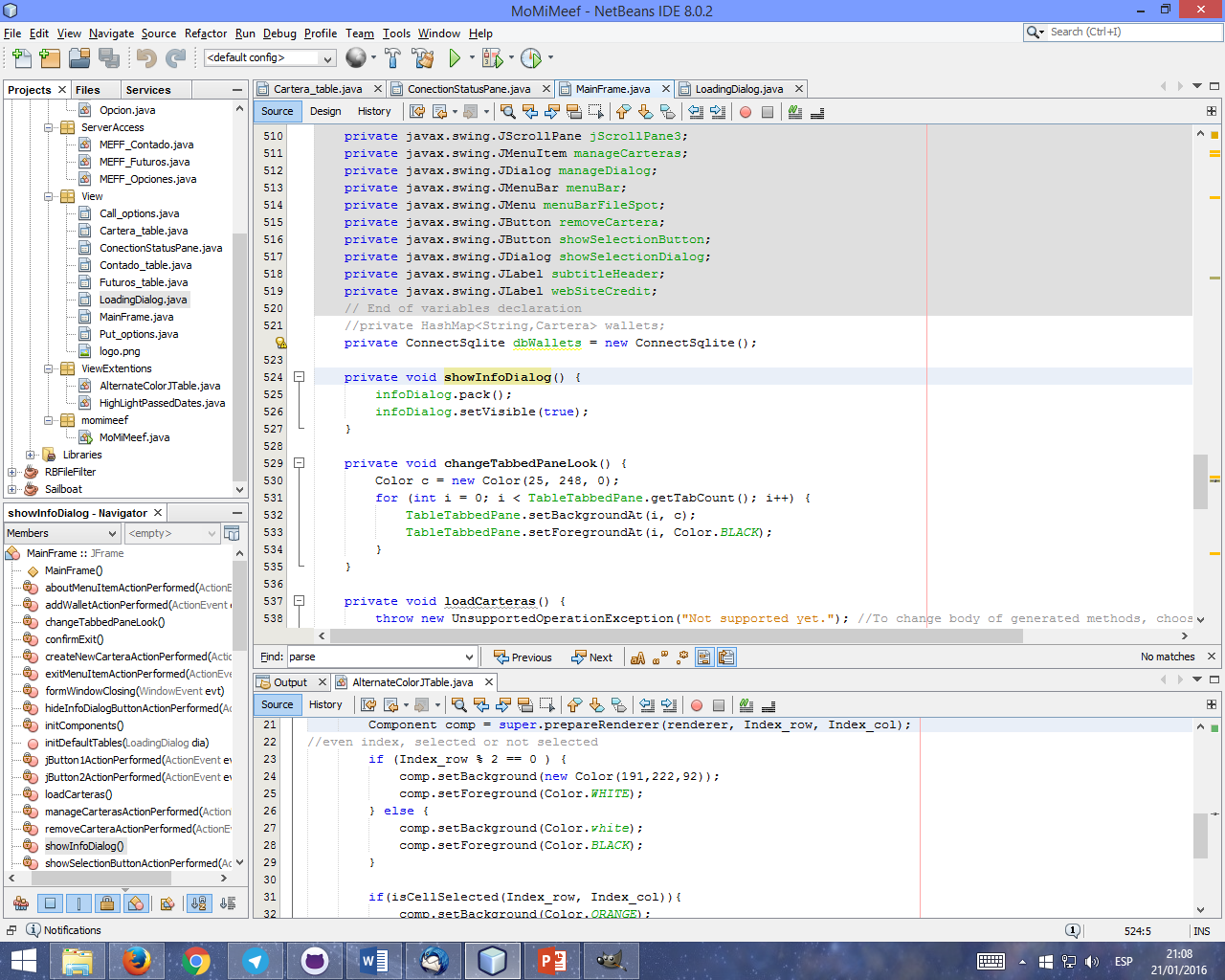


Ilustración 10: Implementación del diálogo de información.



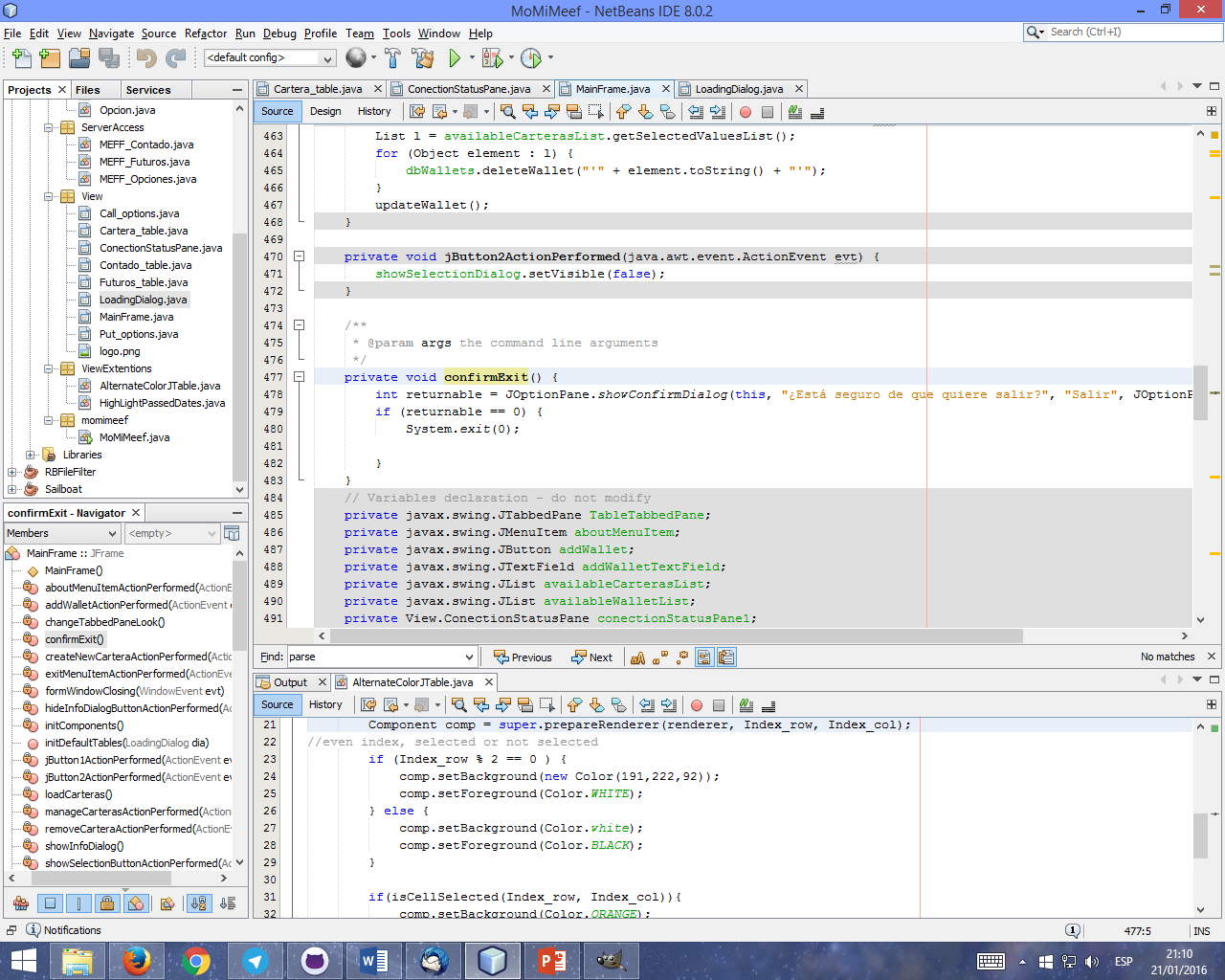


Ilustración 11: Implementación de la confirmación de salida

Ahora que hemos concluido con los elementos de este menú desplegable pasamos al menú desplegable de las carteras. Éste solamente tiene un único elemento “Gestionar carteras” que despliega un diálogo en el que podemos realizar las tres acciones posibles sobre una cartera: crear una cartera, mostrar una cartera en el gestor de pestañas y eliminar una cartera.

Es con este elemento con el que hacemos el primer contacto con la base de datos SQLite asociada a la aplicación puesto que tenemos en ella un listado de todas las carteras que el usuario ha creado.

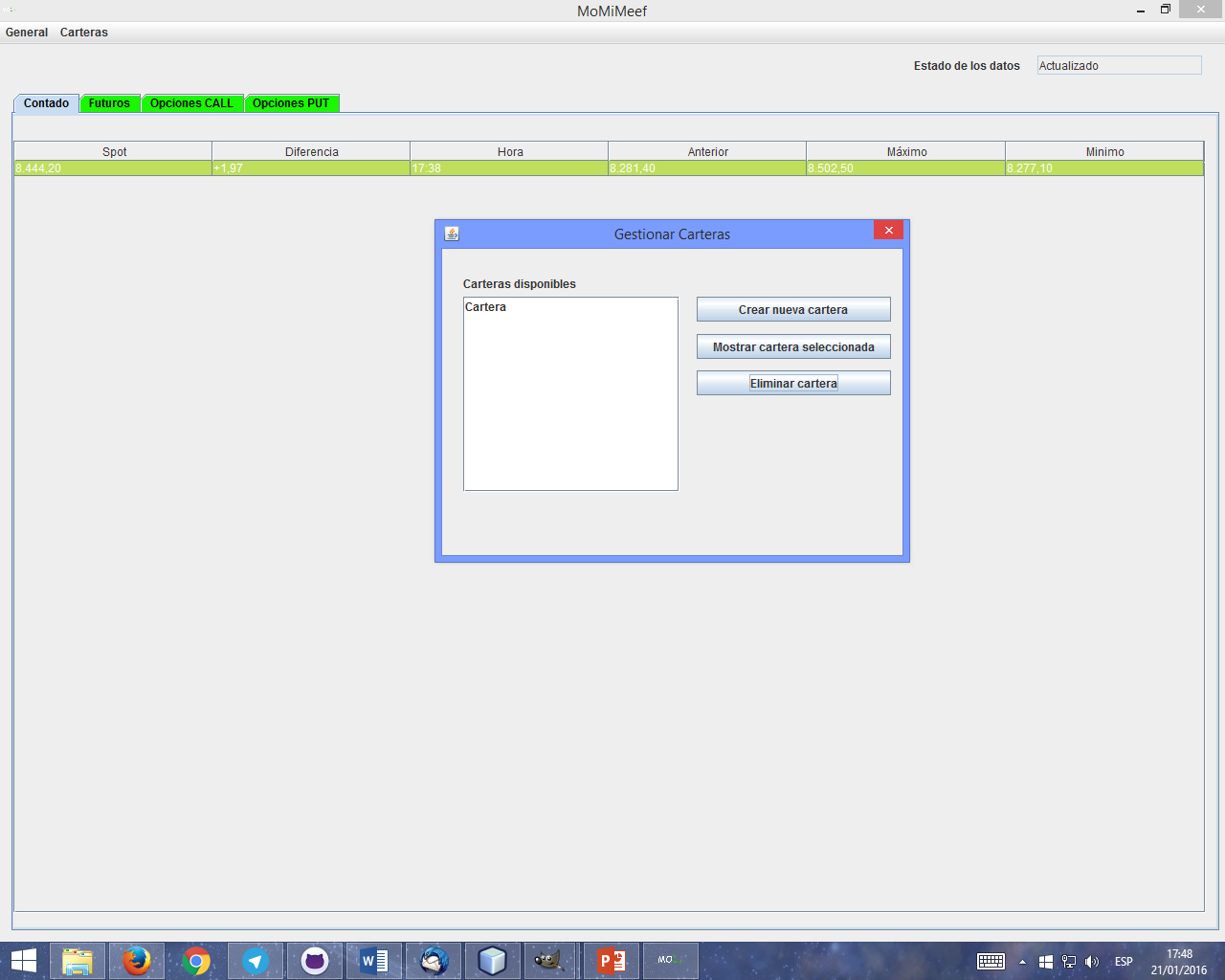


Ilustración 12: Diálogo de gestionar carteras.

Como veníamos diciendo en este diálogo podemos crear una cartera haciendo clic en el botón “Crear una nueva cartera” que nos abrirá un nuevo diálogo en el que introduciremos un nombre de cartera. Se realiza un control de errores sobre este nuevo nombre, de forma que si el nuevo nombre ya está en la base de datos se muestra un mensaje de error como el que se muestra a continuación:

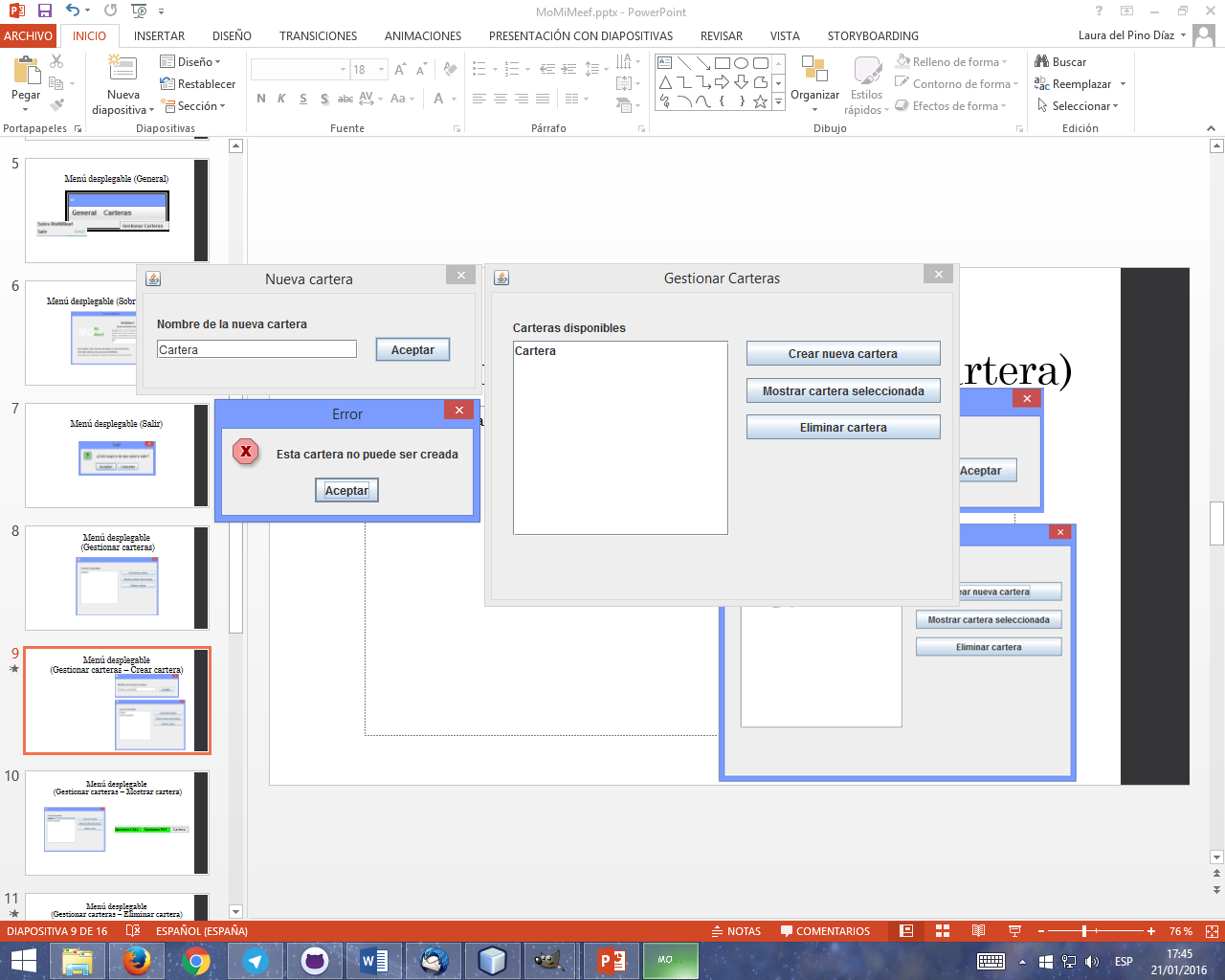


Ilustración 13: Control de errores sobre los nuevos nombres de carteras

Y esto lo hemos implementado con el siguiente código:

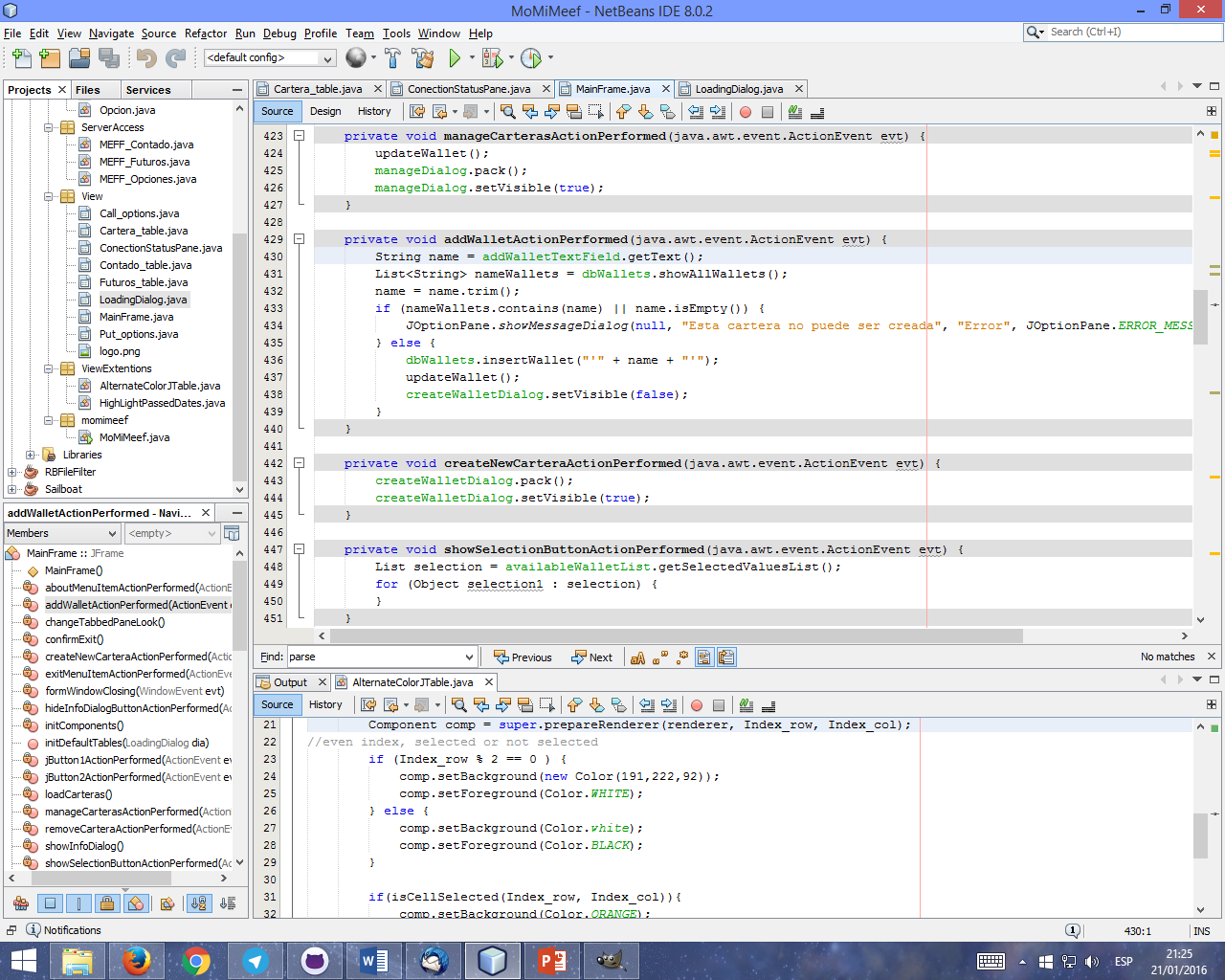


Ilustración 14: Manejador del botón de aceptar el nombre de la nueva cartera.

La segunda acción que podemos realizar en este menú es mostrar una cartera. La cartera tiene que haber sido seleccionado previamente en la lista que tenemos a la izquierda. El modo de selección de esta lista es SINGLE, de forma que solo se puede elegir un elemento.

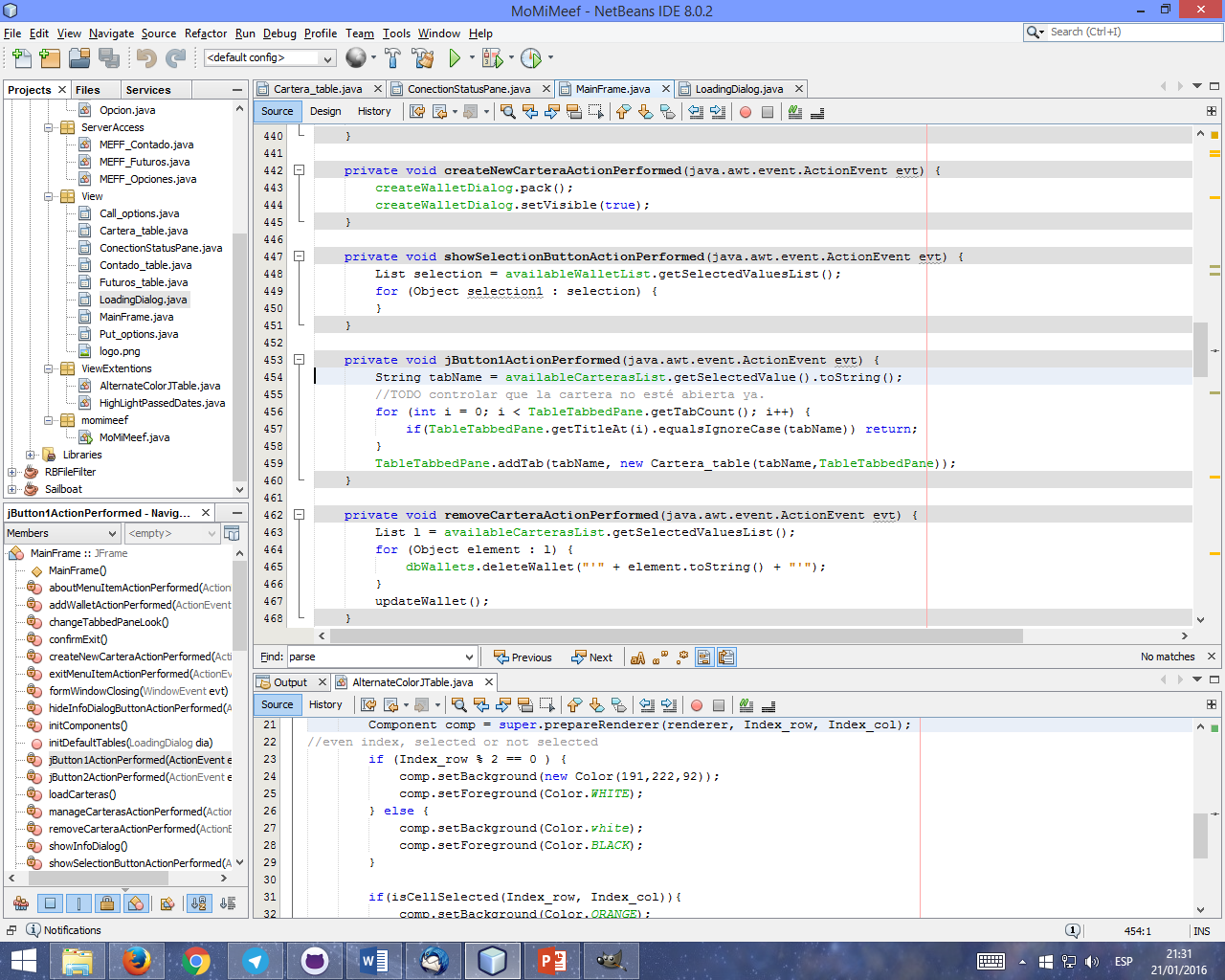


Ilustración 15: Manejador del botón que muestra las carteras.

Y por último eliminar una cartera, al igual que el método de mostrar una cartera, la cartera afectada tiene que estar seleccionada para realizar la operación.

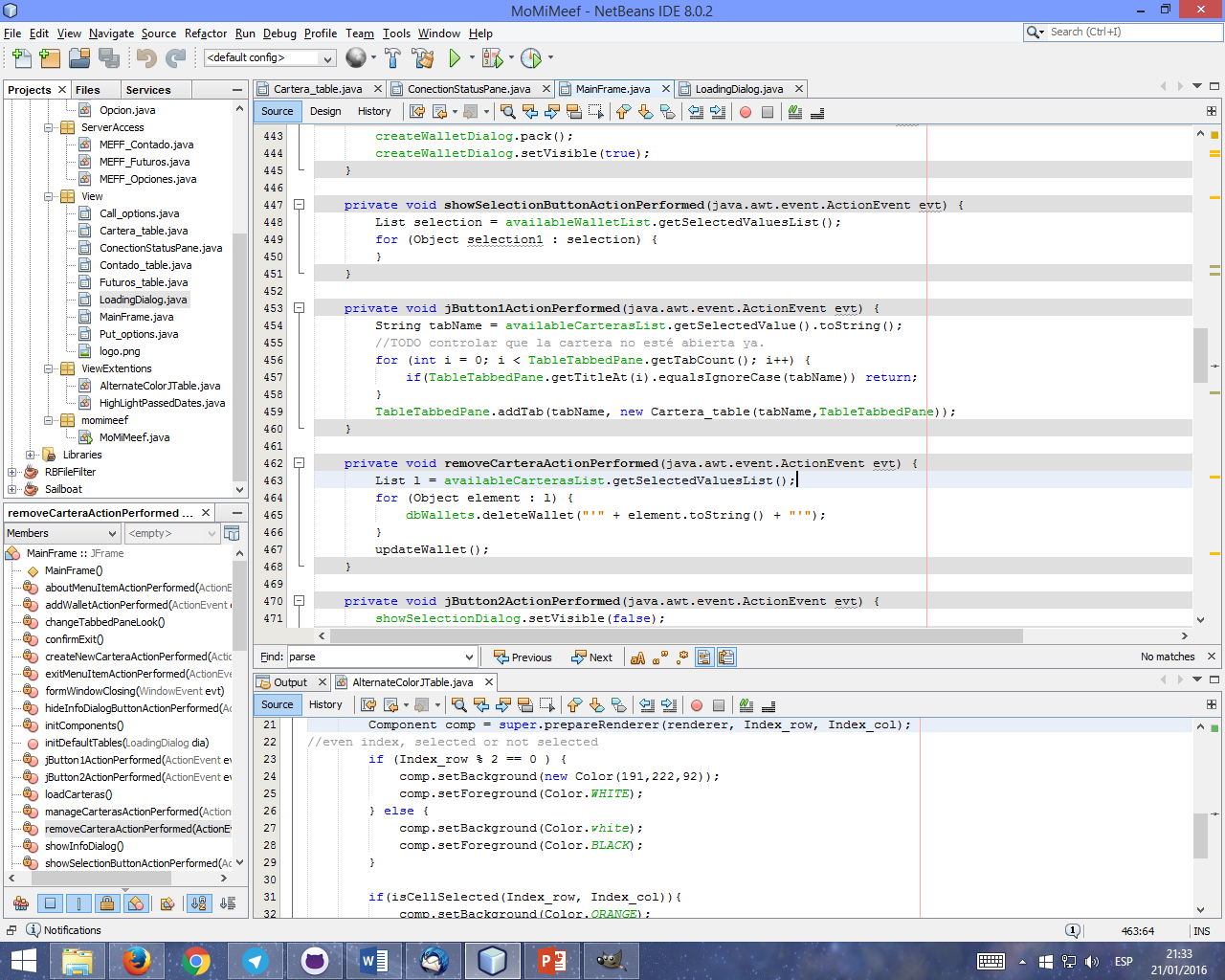


Ilustración 16: Manejador del botón que elimina una cartera.

Para terminar esta sección nos queda tocar un elemento que no es que esté en el menú superior pero está muy próximo a él que es el elemento que nos notifica cuál es el estado en el que están los datos que vemos en pantalla. Al igual que el diálogo de carga lo hemos implementado en una clase aparte “ConectionStatusPane”. Junto con la funcionalidad de monitorizar el estado de los datos que estamos viendo, este componente nos permite saber en qué momento el Worker, o proceso en segundo plano, va a actualizar los datos con un Tooltip como se muestra en la siguiente imagen.

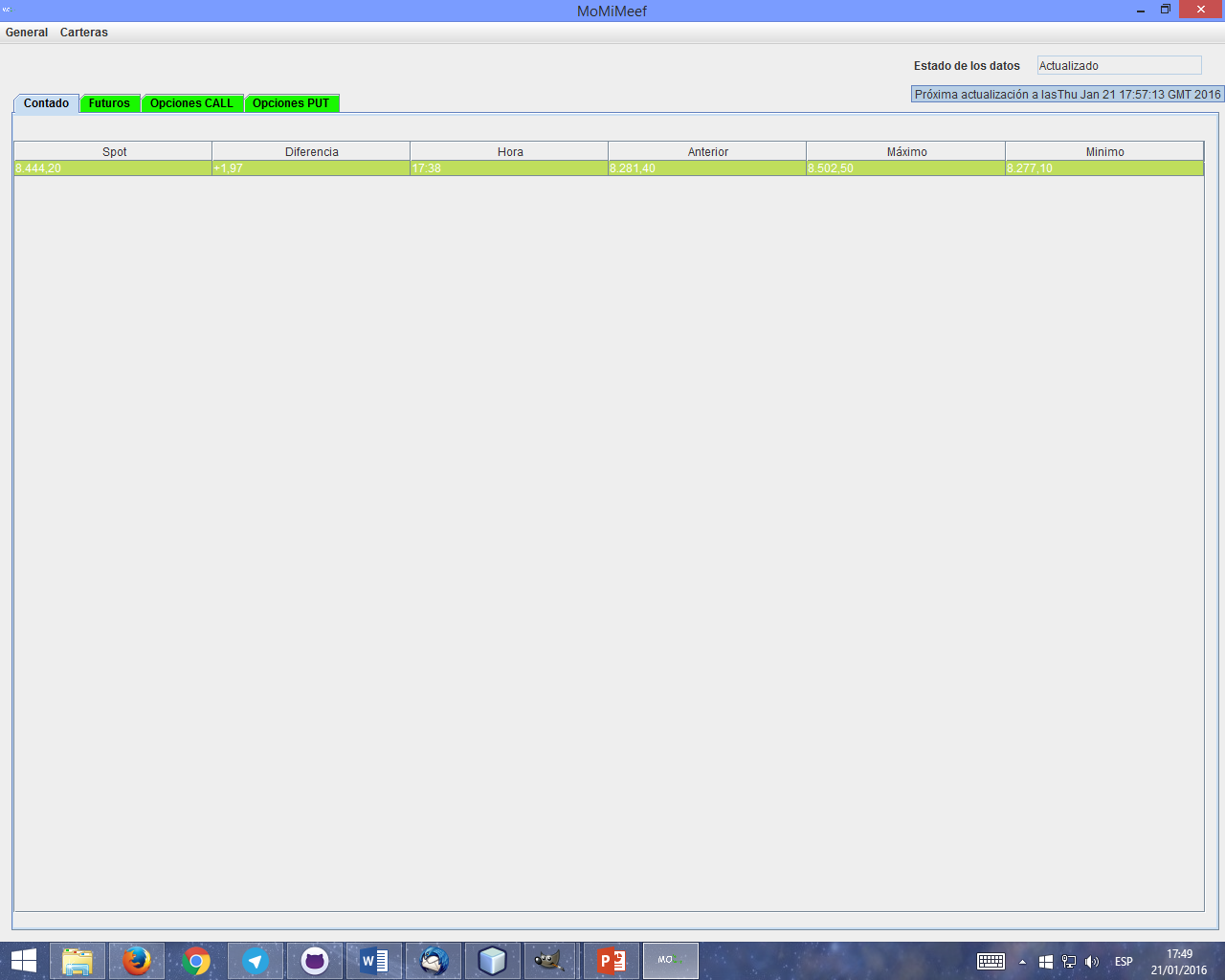


Ilustración 17: Elemento de monitorización del estado de datos, con la fecha en la que se actualizarán los mismos.

El worker que actualiza este componente tiene la siguiente implementación:



Ilustración 18: Implementación del proceso en segundo plano.

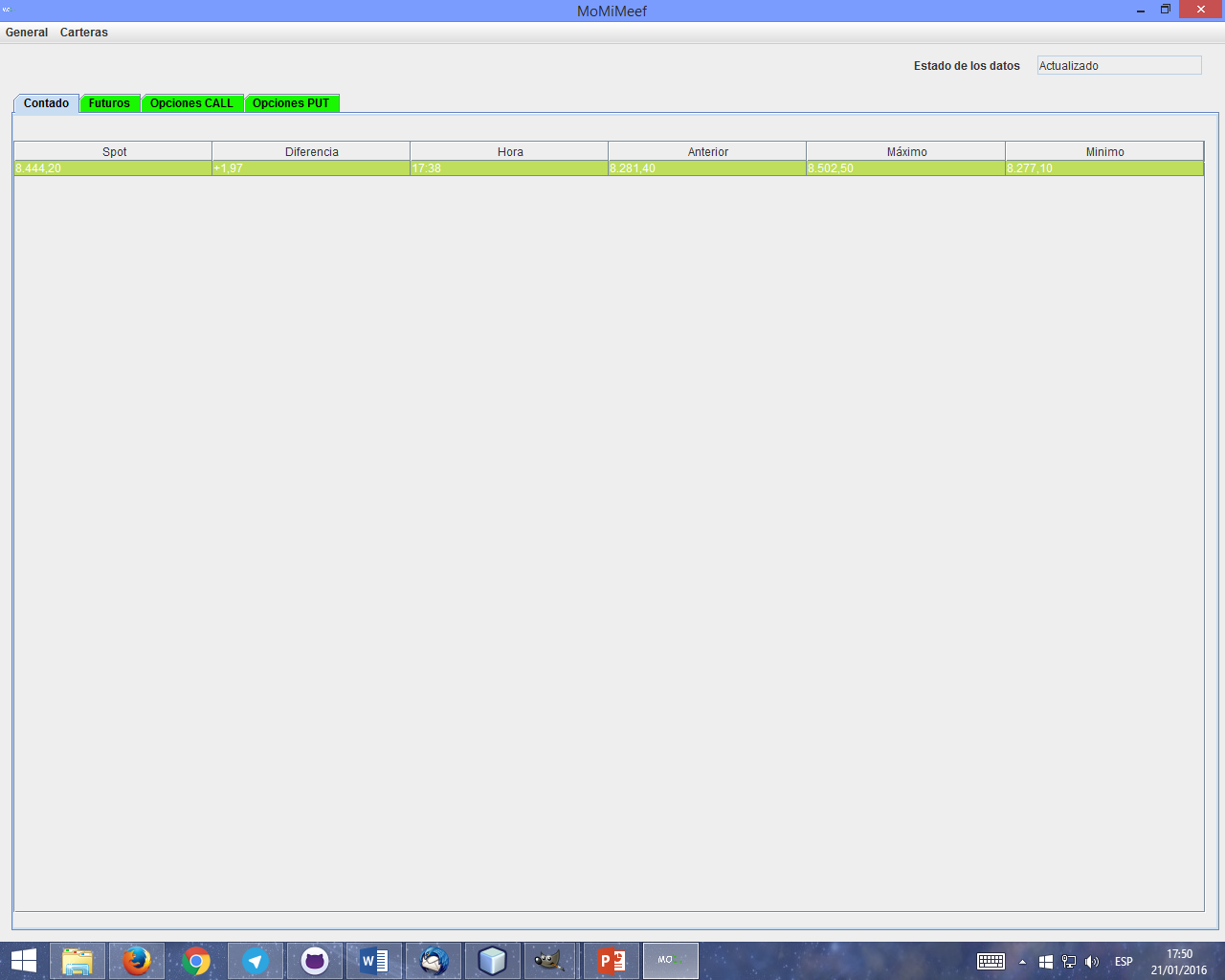
Este proceso en segundo plano no comparte información con el hilo de evento o con el hilo principal de forma directa mediante los métodos que ya conocemos sino nos aprovechamos del acceso a las variables del hilo principal que tenemos por el hecho de ser una clase privada anidada en el cuerpo del marco principal de la aplicación puede acceder a la información necesaria, en este caso realizar actualizar el estado del componente que presentábamos antes y decirle al mismo que la fecha de actualización es en quince minutos, dormir dicho tiempo tras el cual llamará a las funciones de actualización de cada una de las pestañas que tenemos por defecto en la aplicación.

Se ha predispuesto que se actualice cada quince minutos porque sabemos que la página web del MEFF se actualiza en ese intervalo.

pestañas estáticas

Denominamos pestañas estáticas al conjunto de las pestañas que se cargan al principio del programa y que el usuario no puede cerrar. Éstas contienen la información del contado, la tabla de futuros y las opciones tanto del tipo put como del tipo call.

Las pestañas de contienen la información tanto del contado como de los futuros se limitan solamente a mostrar dicha información sin aportar ningún tipo de herramienta extra.



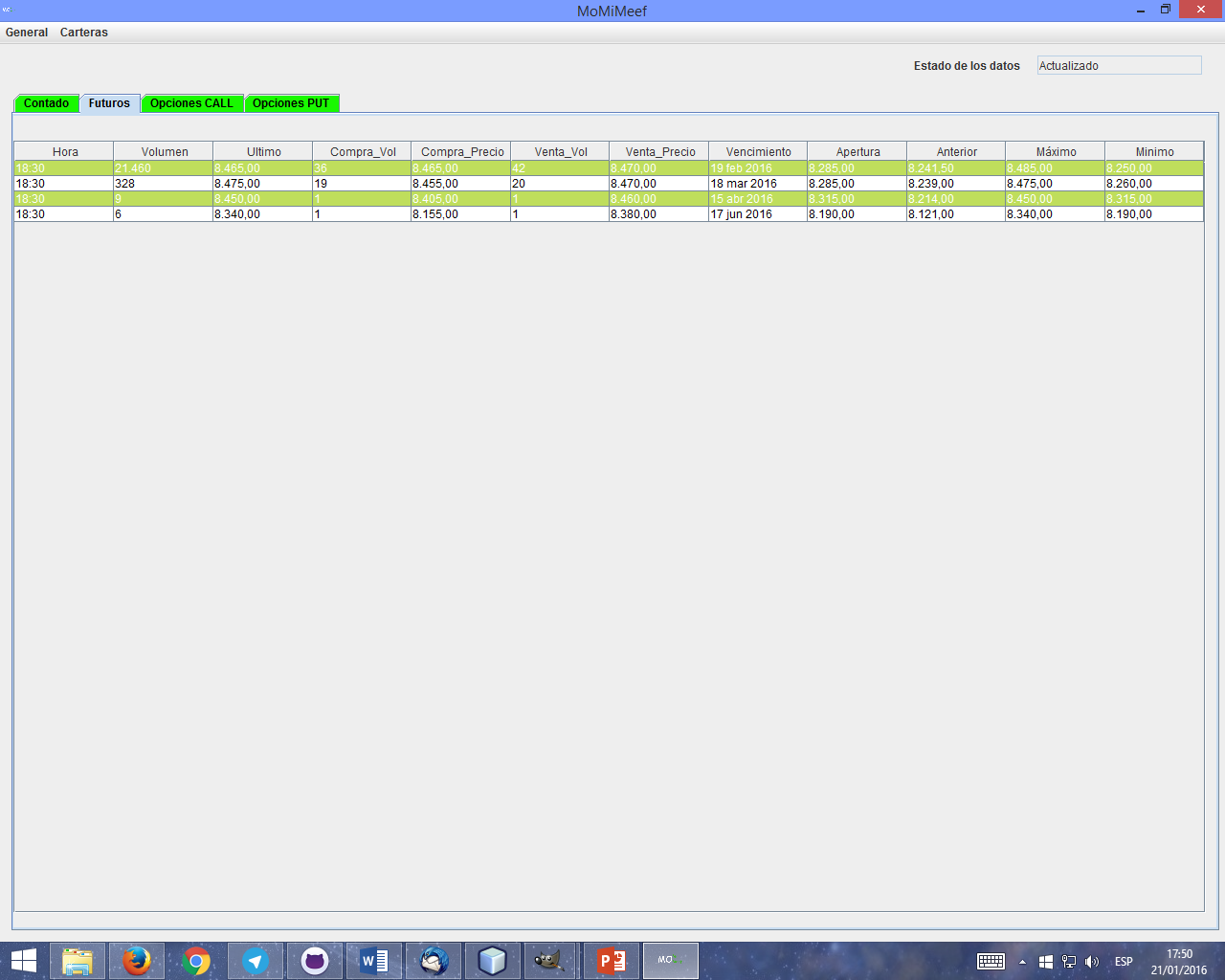
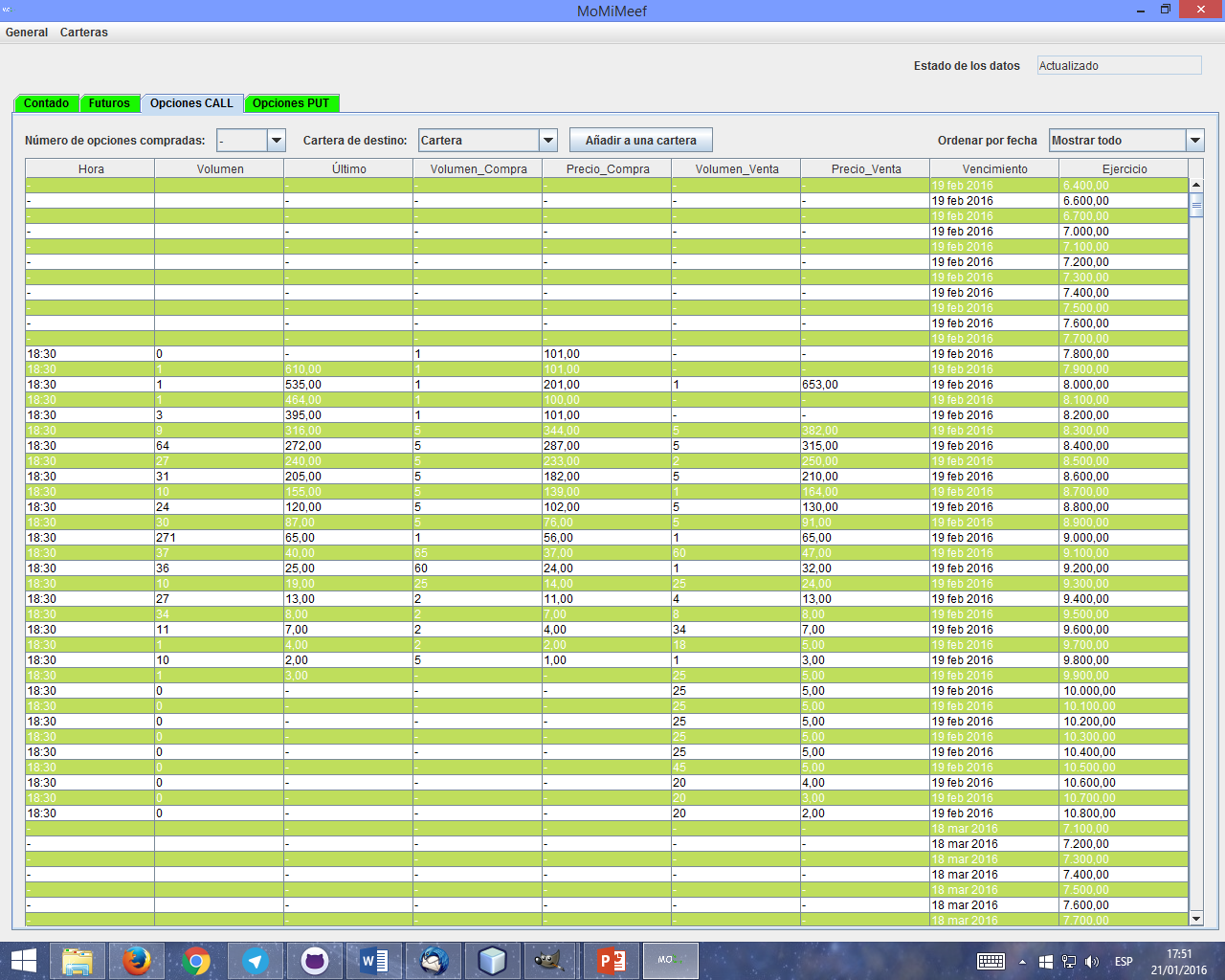


Ilustración 19: Captura de pantalla de las pestañas estáticas de contado y futuro.

Sin embargo las pestañas de las opciones sí que tienen herramientas adicionales que permiten al usuario introducirlas en la cartera que deseen y filtrarlas por fecha de vencimiento:



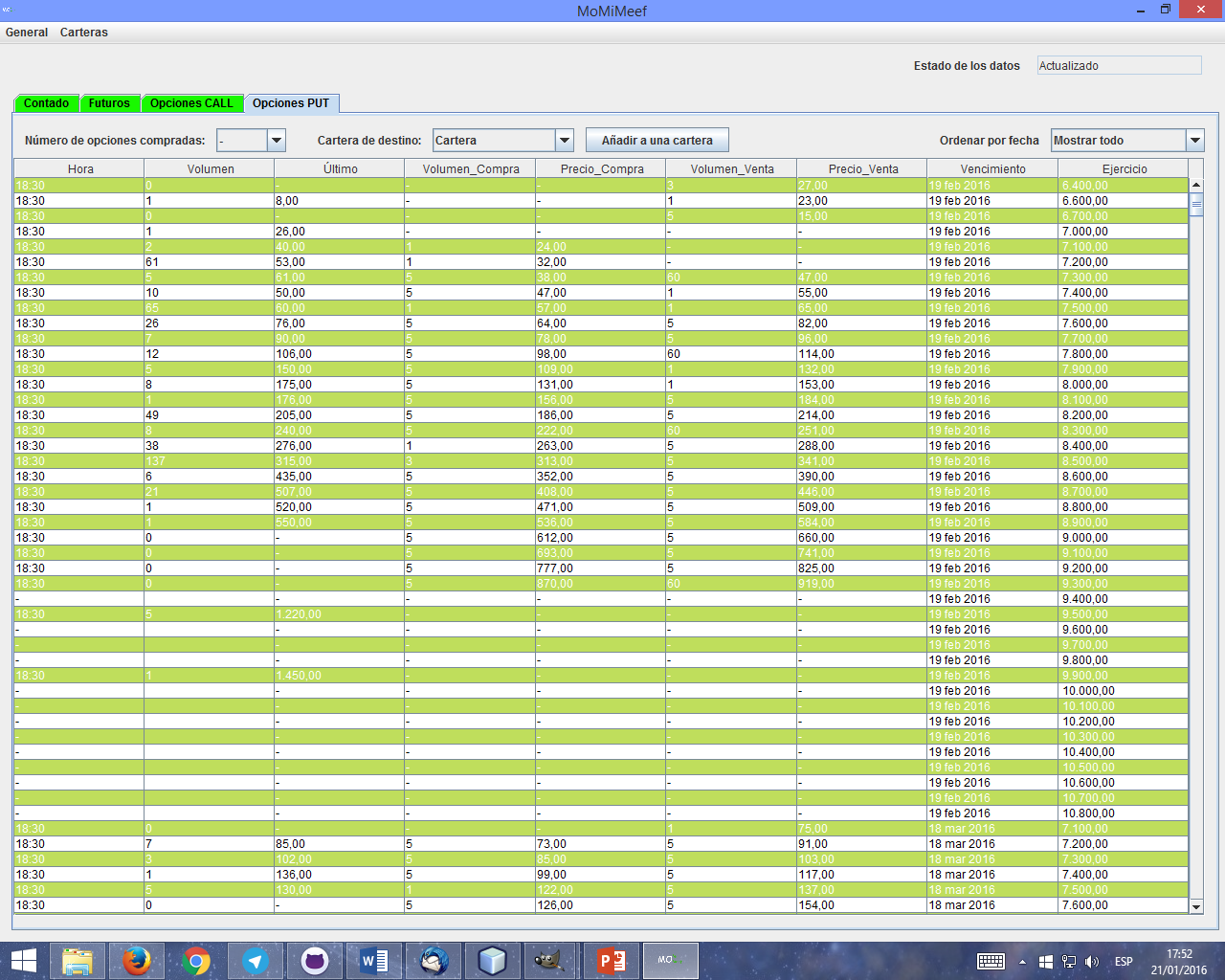


Ilustración 20: Captura de pantalla de las pestañas estáticas de opciones con su menú adicional.

Analicemos los componentes uno por uno, empecemos por el combobox que filtra las opciones por la fecha. Esta acción se implementa con el manejador del evento “itemStateChanged”:

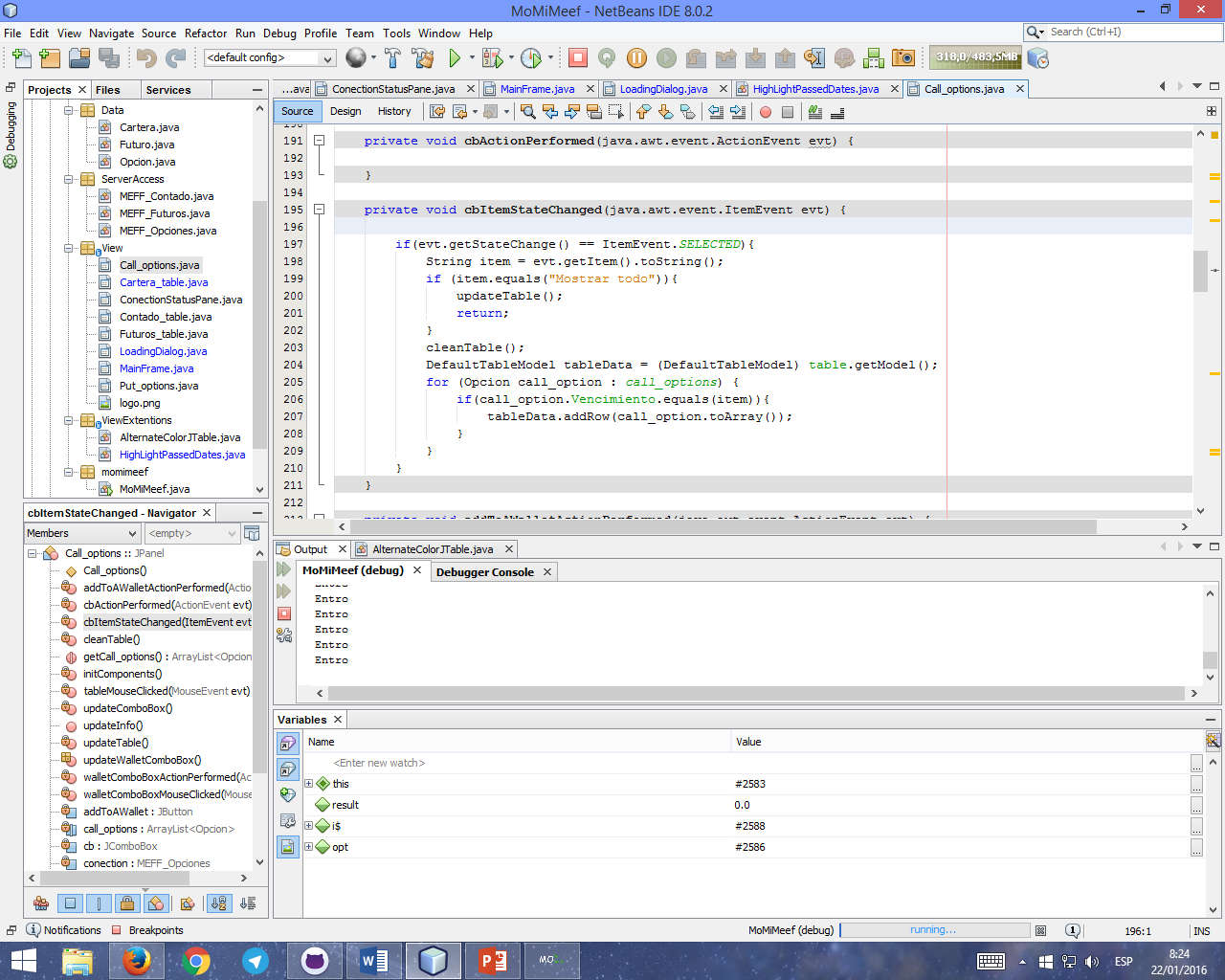


Ilustración 21:manejados del evento itemStateChanged

Este evento nos permite actuar en el momento en el que se selecciona una de las fechas disponibles, que son todas las que aparecen en la tabla de opciones sin repetición. El combobox está por defecto en “Mostrar todo” que muestra el total de las opciones.

Además este evento nos permite saber que elemento del combobox fue el seleccionado con la llamada “evt.getItem()”.

El manejador del evento lo que hace es que si el evento lanzado fue porque se seleccionó un elemento entonces comprueba si el elemento seleccionado es “Mostrar todo” que mostrará todos los elementos o alguna de las fechas, en este caso lo que hará es borrar la tabla por completo y rellenarla con aquellos elementos que tienen la fecha seleccionada.

Acompañando al combobox tenemos una herramienta para añadir opciones a una cartera compuesta de un combobox que permite al usuario elegir el número de opciones que quiere guardar en la cartera que está acompañado de otro comobox que contiene el nombre de todas las carteras que existen en la base de datos. Para añadir una opción a una cartera lo único que tiene que hacer el usuario es seleccionar una opción en la lista y darle al botón de “Agregar a una cartera”.

El manejador de dicho botón es el siguiente:

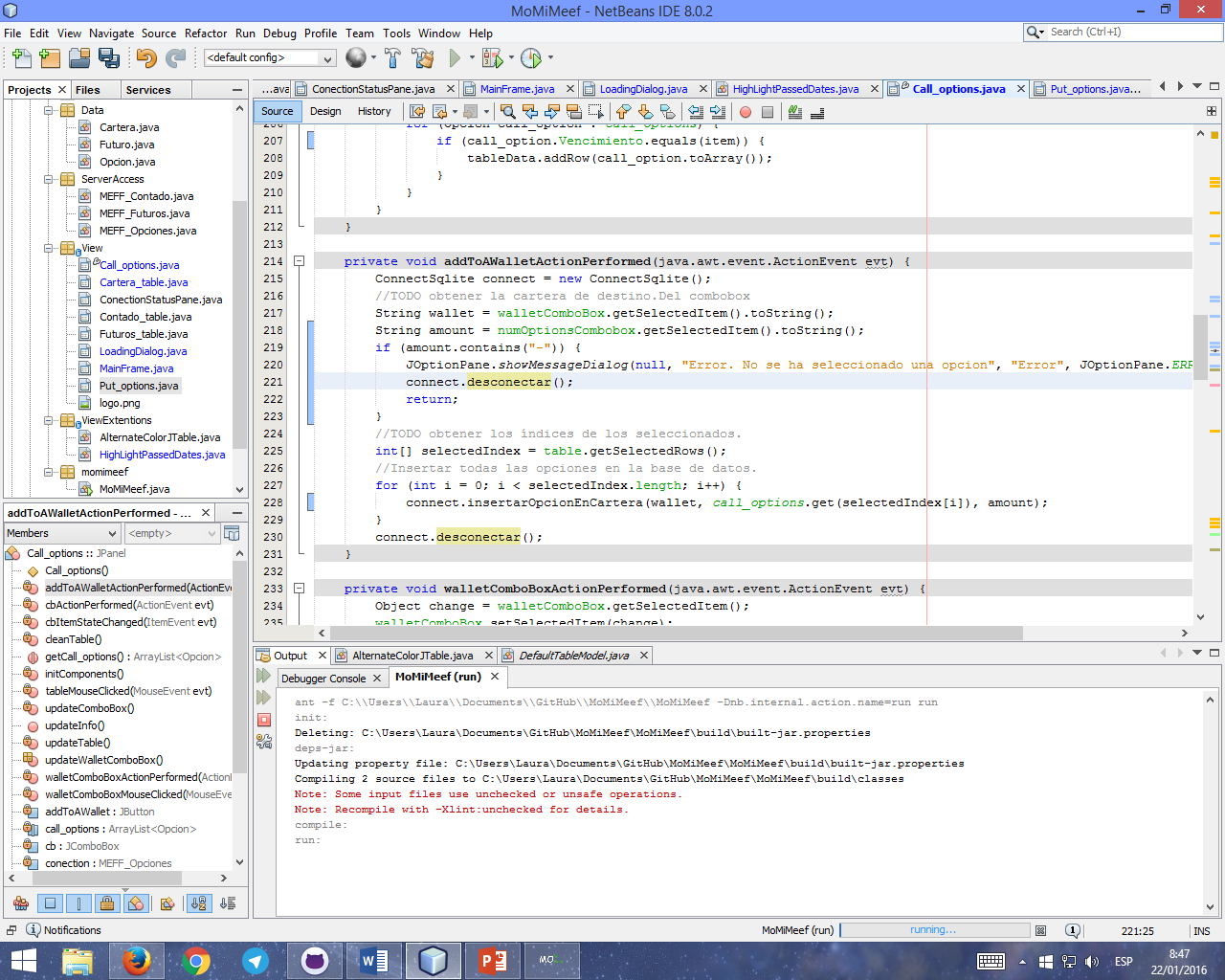


Ilustración 22: Manejador del botón que añade una opción a una cartera.

En este manejador nos conectamos con la base de datos. Obtenemos el nombre de la cartera de destino y el número opciones que se van a añadir. A continuación controlamos el caso de que no se haya seleccionado ningún número de opciones, en este caso mostramos un mensaje de error y cerramos la base de datos. En caso de que sí se haya seleccionado el número de opciones pues se toma el índice correspondiente para pasarle a la base de datos el nombre de la cartera, los datos completos de la opción y la cantidad de ellas para que la inserte.

Todas las operaciones sobre la base de datos están reunidas en la clase “ConnectSQLite” que conecta con la base de datos que utiliza la tecnología SQLite. Esta base de datos tiene dos tablas una de ella contiene una lista con todos los nombres de las carteras del usuario y la segunda contiene todos los campos de las opciones que se muestran incluyendo el tipo de opción y la cartera a la que pertenece.

La operación que utilizamos en la ilustración 22 inserta una opción en la segunda tabla descrita y se implementa con la siguiente función:

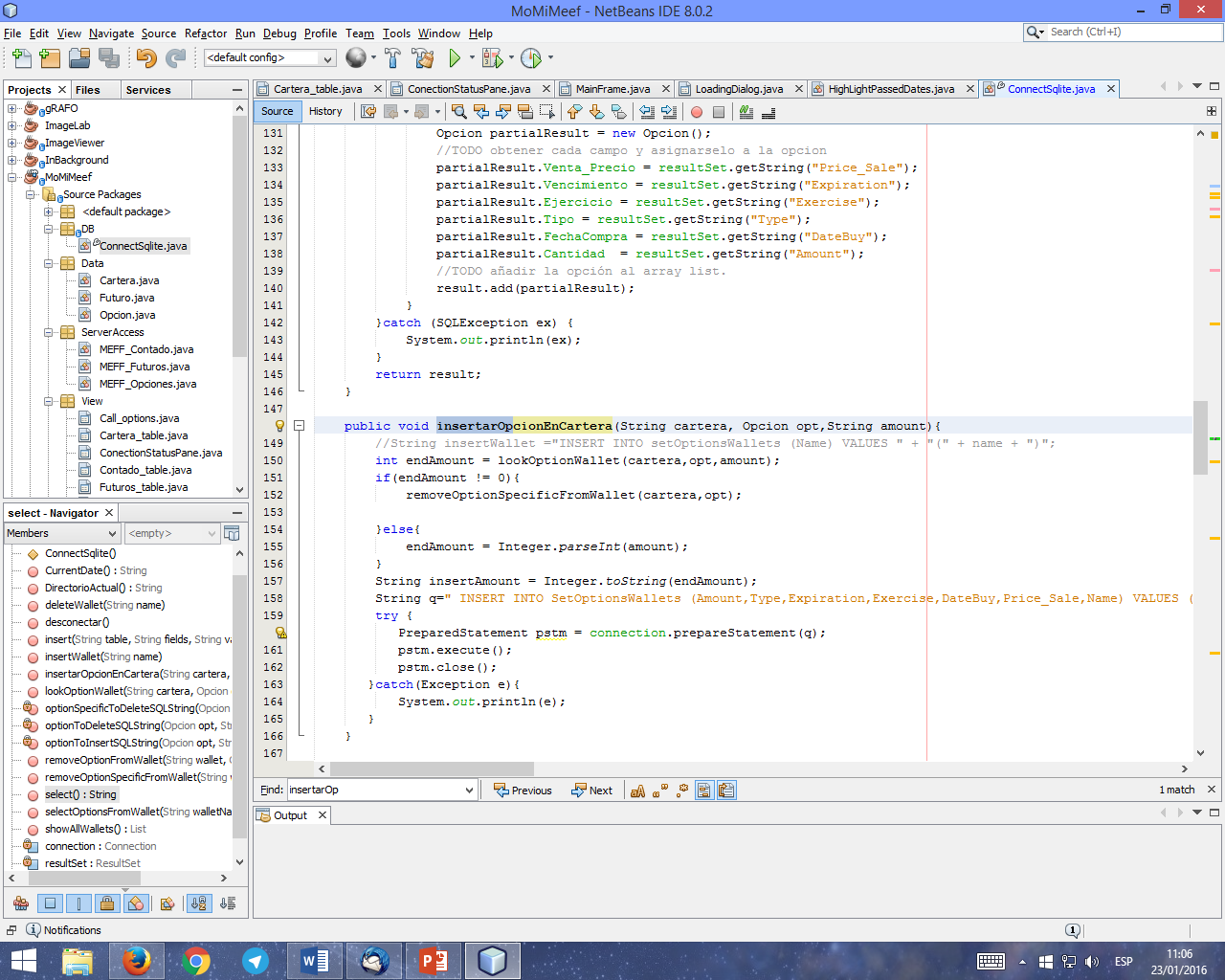


Ilustración 23: Implementación de la inserción de una opción en una cartera.

Lo primero que se mira es si la opción ya existe en la cartera y si existe con una cierta cantidad se elimina la opción. En caso de que no exista la opción entonces se determina que la cantidad con la que se inserta es la que se pasa por parámetro. Tras ello se parsea el número de opciones a string para utilizarlo dentro de la sentencia SQL que construimos a continuación y que ejecutamos.

pestañas de cartera

Las pestañas de cartera contienen la información referente a las opciones que contiene de forma individual y una información global a todas ellas. Se presenta de la siguiente forma al usuario:

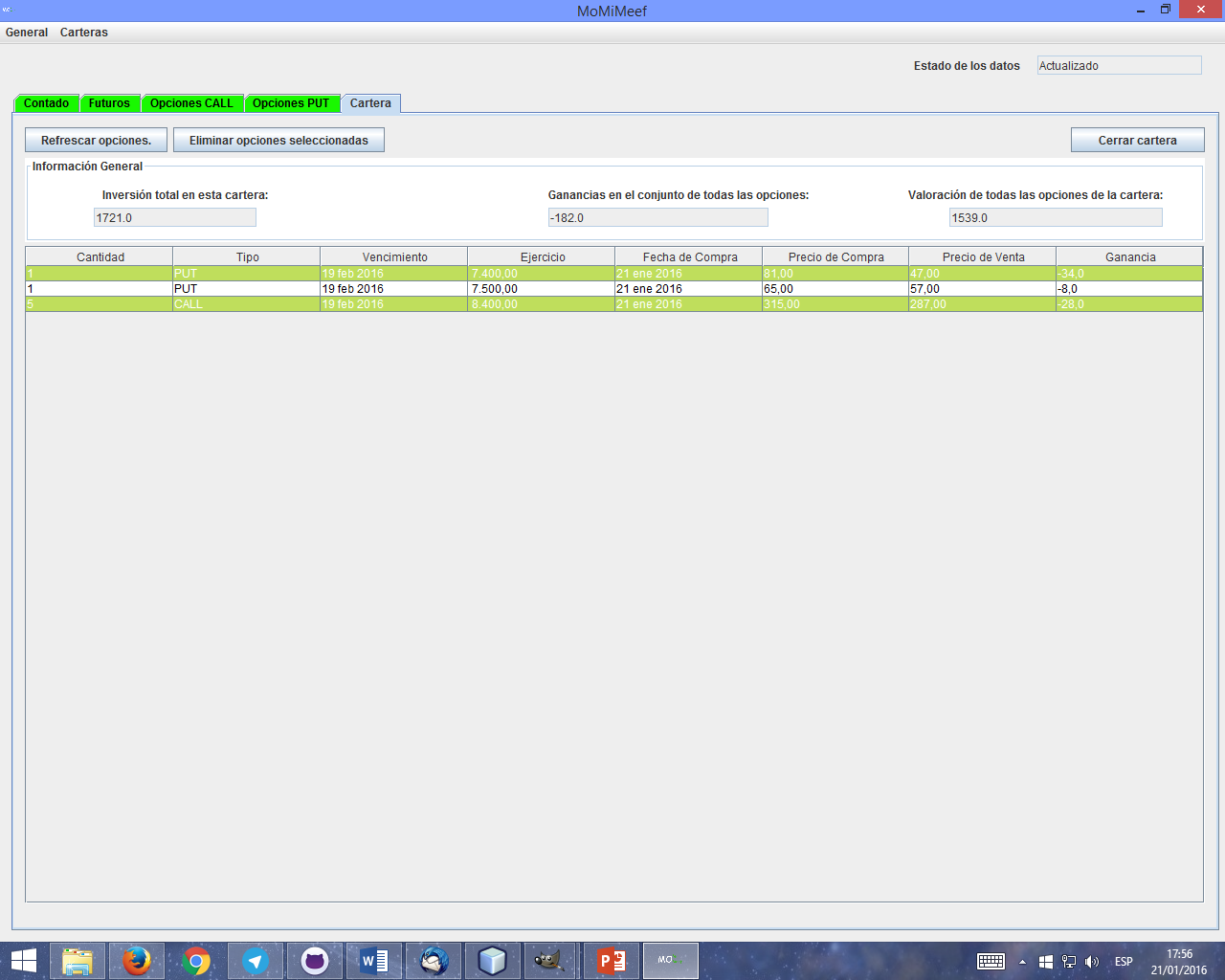


Ilustración 24: Pestaña en el que se muestre una cartera.

El precio de compra que se muestra en esta pestaña es el precio al que compró el usuario una opción de este tipo y el precio de venta es el precio en el que actualmente se vende en el mercado también por unidad.

Encima de esta tabla de opciones encontramos un panel con la información general de todas las opciones que tenemos considerando la cantidad que tenemos de cada tipo.

Las interacciones que puede realizar el usuario en esta pestaña son las siguientes:

* Refrescar las opciones: en caso de que haya abierto la pestaña antes de añadir una nueva opción, necesitará esta acción para ver el conjunto completo de opciones de la cartera actualizado.
* Eliminar opciones seleccionadas: permite eliminar aquellas opciones que están seleccionadas.
* Cerrar pestaña: cierra la pestaña de esta cartera.

Para llevar a cabo dichas opciones se implementan los siguientes manejadores de evnetos en la clase “cartera\_table”:

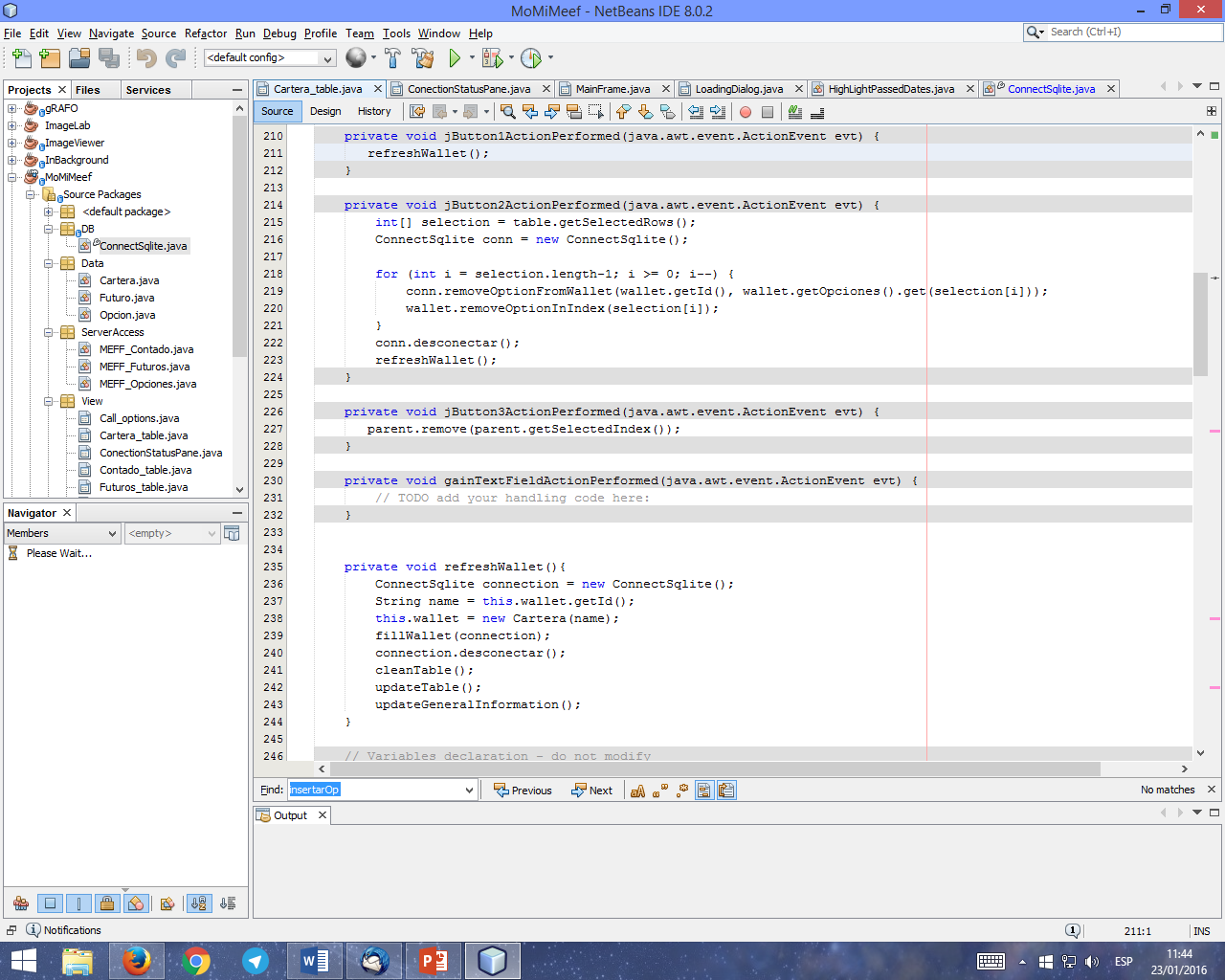


Ilustración 25: Manejador del botón de refresco de la cartera.

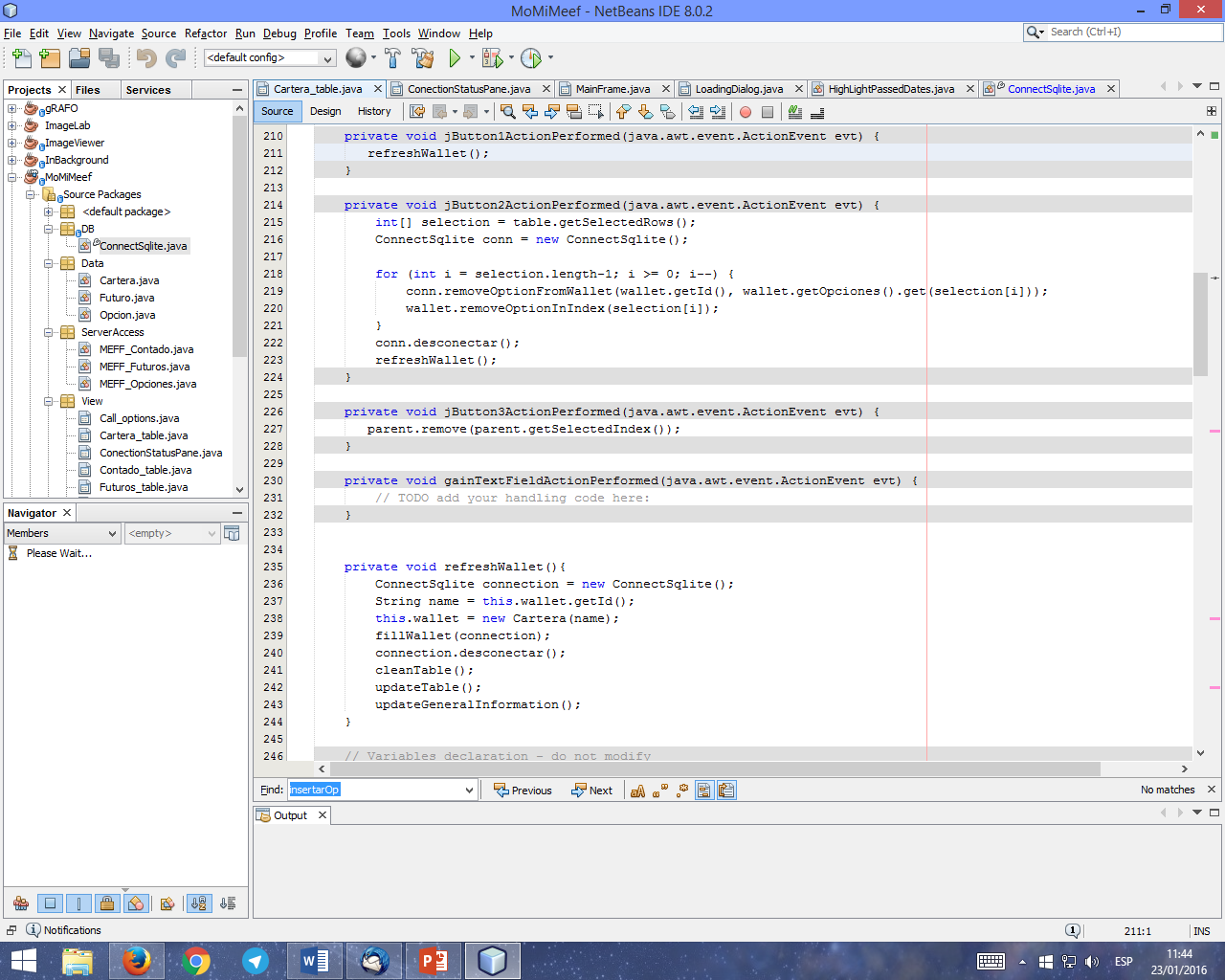


Ilustración 26: Función de refresco de la cartera.

En esta función de refresco se conecta con la base de datos y se regenera la estructura de memoria dinámica que contiene la información de la cartera, implementada con la clase “Cartera”, utilizando la función “fillWallet”. Tras lo cual se vacía la tabla y se muestran los datos nuevos y se actualizan los datos relativos al conjunto de las opciones.

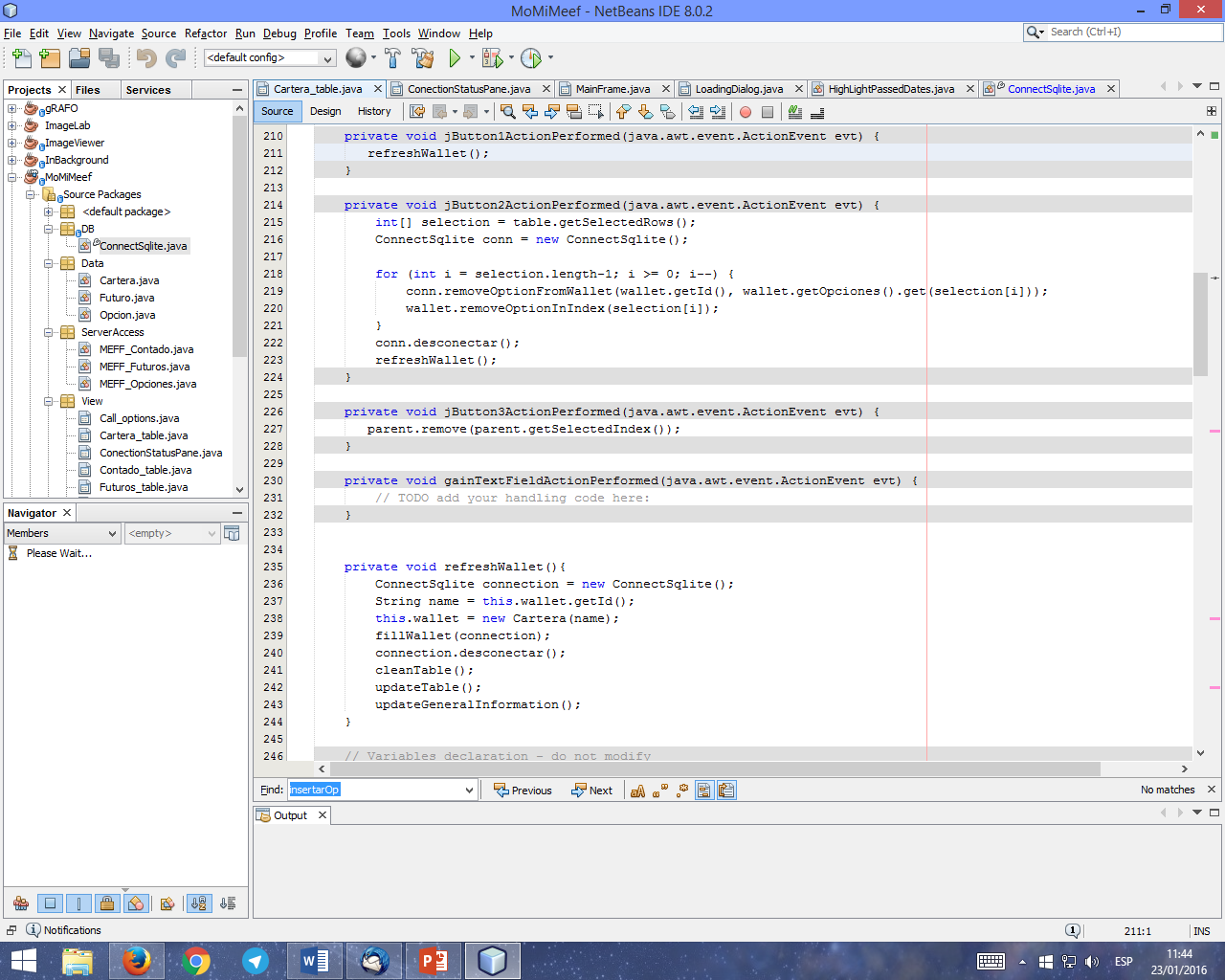


Ilustración 27: Manejador del botón de eliminar las opciones seleccionadas.

Este segundo manejador elimina las opciones seleccionadas, para ello obtiene de la tabla los índices de todas las filas seleccionadas y se conecta a la base de datos para eliminar de ella la opción a la misma vez que la elimina de la estructura de memoria dinámica que tenemos. Por último refresca la cartera en la interfaz gráfica.

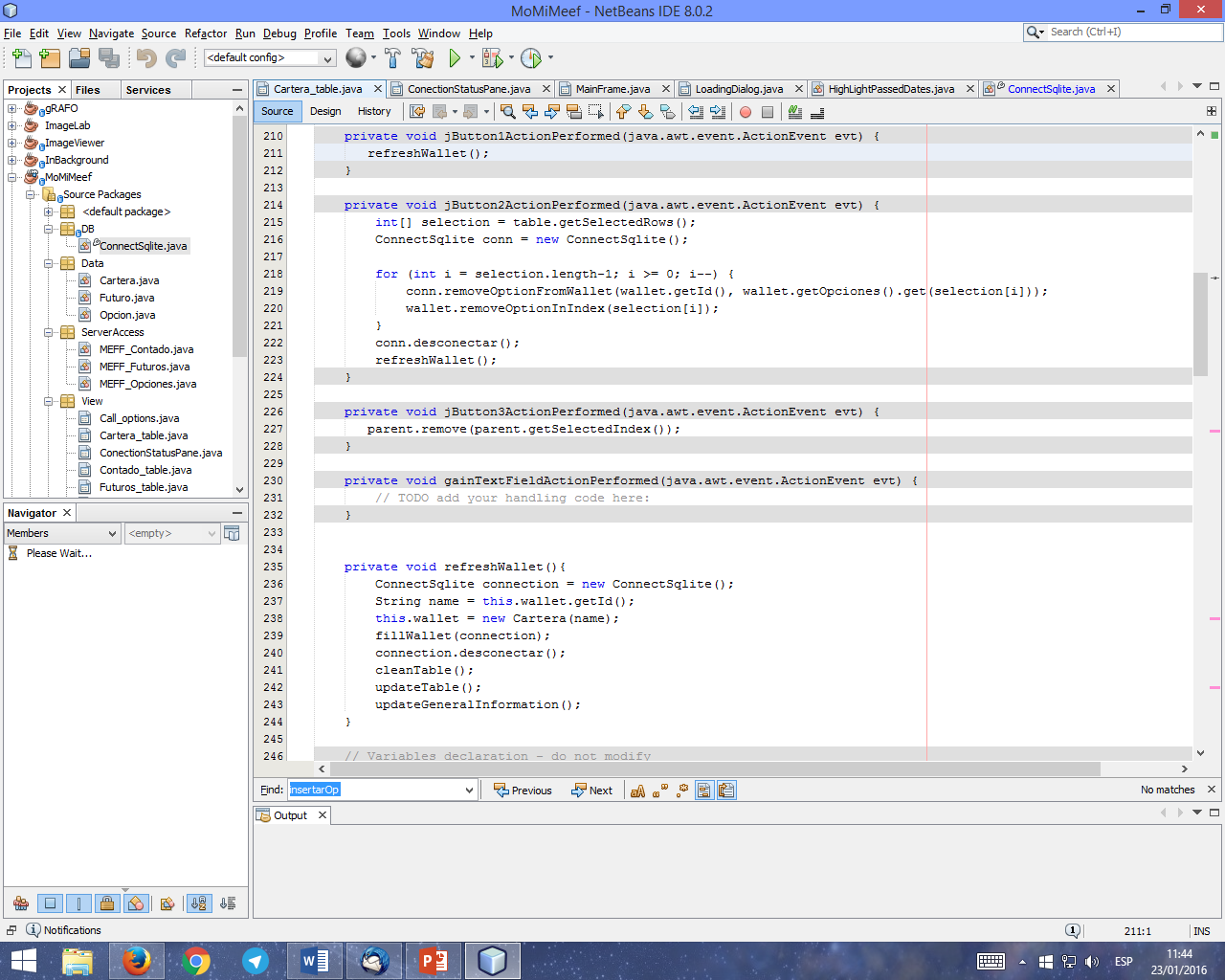


Ilustración 28: Manejador del botón de cerrar pestaña.

Por último el botón de cerrar pestaña llama al componente padre que le hemos pasado en el constructor para que elimine la pestaña en el índice seleccionado que es la pestaña en la que estamos puesto que el botón está dentro de la pestaña.

conclusión

En este trabajo de curso hemos puesto en práctica los siguientes conocimientos adquiridos en las sesiones de laboratorio:

* Manejadores de eventos.
* Barras de progreso
* Etiquetas con imágenes.
* Listas
* Utilización de diferentes modelos de selección (multi-intervalo para la eliminación de opciones dentro de una cartera y opción única en las listas y tablas).
* Menús desplegables.
* Creación de componentes prediseñados con el elemento de monitorización de los datos.
* Hilos en segundo plano para la actualización periódica de los datos.