Memoría Escrita Del Proyecto Mis Finanzas - Python

Programación Orientada A Objetos

Juan Pablo Mejia Gomez , Jorge Humberto Gaviria Botero ,
Leonard David Vivas Dallos , Jose Daniel Moreno Ceballos y
Tomas Escobar Rivera .

Docente: Ph.D. Jaime Alberto Guzman Luna.

Universidad Nacional De Colombia

Sede Medellín

Facultad De Minas

Medellín, 2023

Tabla de Contenido

1. ***Ventana de inicio:***

1.1. Brindar un saludo de bienvenida al sistema (Izquierda superior: P3)

Imagen 1.1 Saludo de bienvenida

Imagen 1.2 Código saludo de bienvenida

1.2. Breve hoja de vida de cada desarrollador:

Cada hoja de vida cambia por clic del ratón sobre la región del texto de la hoja de vida. (Derecha superior: P5): Por cada clic sobre la región del texto de la hoja de vida se llama a ejecución al método change_button_text, que primero verifica el primer carácter del texto actual del botón utilizando button_developers_text.get()[0:1]. Si el primer carácter es "1", se cambia el texto del botón a "2. Leonard David Vivas Dallos." y se aplica un estilo particular al botón. Si el primer carácter es "2", se realiza una acción similar a la anterior, pero se cambia el texto del botón a "3. José Daniel Moreno Ceballos.". Este proceso se repite para los casos en los que el primer carácter sea "3", "4" o "5", cambiando el texto del botón y aplicando un estilo particular para cada caso y volviendo al principio cada vez. Al final de la función, se llama a la función update_image() para actualizar las imágenes mostradas en la interfaz gráfica.

Imagen 2. Código hoja de vida de los desarrolladores

1.3. Cada vez que se cambie la hoja de vida de cada desarrollador como se describe en el apartado anterior, se deberá mostrar en P6 4 fotos de cada uno de los desarrolladores haciendo uso del posicionamiento Grid dentro de P6:

Se llama al método *update_image* cada vez que se da click sobre el apartado de hoja de vida. Allí se crea una lista llamada *image_paths* que contiene las rutas de las imágenes en un orden específico. Luego, se borran todos los subframes existentes en un marco llamado *bottom_right_frame*. Posteriormente se divide el marco *bottom_right_frame* en una cuadrícula de 2 filas y 2 columnas. Se crean subframes para mostrar las imágenes superiores. Se itera dos veces y en cada iteración se realiza lo siguiente: Se calcula el índice de la imagen actual utilizando el valor de *image_index*. Por otra parte, se carga la imagen actual utilizando la ruta correspondiente obtenida de *image_paths* y se crea un subframe para mostrar la imagen en el marco *bottom_right_frame*. Se crea un label que contiene la imagen y se muestra en el subframe. Se crean subframes para mostrar las imágenes inferiores. El siguiente proceso es similar al paso anterior, pero se itera otras dos veces y se utiliza un desplazamiento en el cálculo del índice para obtener las imágenes correctas. Se incrementa el valor de *image_index* en 4 para asegurarse de que las siguientes imágenes a mostrar sean diferentes a las actuales. Esto se hace utilizando el operador módulo (%) para asegurarse de que el índice no exceda el tamaño de la lista de imágenes.

Imagen 3. Código de fotos de desarrolladores

1.4. Permitir el ingreso al sistema por medio de un botón que al dar click irá a la siguiente ventana (Ventana Principal de cualquier usuario) (Deberá ocupar la parte inferior de P4)

Imagen 4.1. Inicio de sesión

Imagen 4.2. Código inicio de sesión

Imagen 4.3. Código inicio de sesión

1.5. Imágenes asociadas al sistema. Se podrán cambiar por un evento del ratón al pasar sobre la misma región de la foto. Presentar 5 imágenes. (La imagen deberá ocupar la parte superior de P4):

Se utiliza nonlocal para indicar que la variable *current_image_index* es una variable no local que está definida en un ámbito superior al de la función *change_system_image*(). Esto permite acceder y modificar la variable desde la función. Se incrementa el valor de *current_image_index* en 1 utilizando el operador módulo (%) para asegurarse de que el índice no exceda el tamaño de la lista *image_paths*. Esto permite cambiar al siguiente índice de imágen en cada llamada a la función. Se obtiene la imagen actual utilizando el índice actualizado. La imagen se extrae de la lista images utilizando *current_image_index*. Finalmente se configura la imagen en un widget de etiqueta llamado *system_image_label* utilizando el método config(image=*current_image*). Esto actualiza la imagen mostrada en el widget cada vez que se pasa el cursor por encima de la imagen.

Imagen 5. Imágenes asociadas al sistema

2. Ventana principal del usuario:

2.1. Distribución ventana del usuario:

Imagen 6. Distribución ventana del usuario

2.2. Menú superior (Zona 1 de la interfaz):

Imagen 7. Menú superior

2.3. Zona de interacción usuario (Zona 2 de la interfaz):

2.3.1.1. Los diálogos de texto:

Para llevar a cabo la impresión de diálogos de texto se hace uso de un objeto de la clase *FieldFrame*. Esta última recibe como parámetros a los elementos exigidos durante la práctica, además, recibe parámetro llamado frame de tipo list que contiene la información necesaria para configurar el formato de creación del frame *field_frame* (Dícese de parámetros como el master y su ubicación en el administrador de esquemas).

Imagen 8. Diálogos de texto

2.3.1.2. Muestra de resultados de procesos y/o consultas:

Los resultados de procesos y/o consultas se muestran centrados en la parte inferior de la ventana principal como es requerido en el instructivo de la práctica. Para esto se utiliza el administrador de esquemas grid para ubicar correctamente cada label y button, de la misma manera, se utiliza el método de instancia columnconfigure de los objetos frame para asegurar que los label abarcan todo el espacio de ventana disponible.

Imagen 9.1 Muestra de resultados de procesos y/o consultas

Imagen 9.2 Muestra de resultados de procesos y/o consultas

3. Manejo de errores:

El programa Mis Finanzas plantea numerosos casos de error en función de las acciones tomadas por el usuario en el sistema. Dentro del paquete *excepciones* se encuentran los módulos *accountsException*, *banksException*, *genericException*, *suscriptionException* y *usersException*. Excepciones creadas por los desarrolladores como *genericException.ValueNotFoundException*, entre otras, tienen la función de evitar que el programa deje de funcionar si no se encuentra el valor insertado por el usuario en los campos generados por la clase *FieldFrame*. Lo anterior puede verse en la línea 539 de la clase *App*, en la cual se utiliza la palabra clave *raise* para invocar esta excepción en caso tal de que no se encuentre el nombre de la instancia *Banco* insertada por el usuario.
Puede ver otros ejemplos de manejo de excepciones creadas por los desarrolladores en el siguiente enlace:

Imagen 10. Manejo de errores

4. Manual de usuario:

Importante: Lo que está en cursiva son las preguntas que se le hacen al usuario y que debe responder ingresando datos por consola.

Luego de ejecutar el programa aparecerá una ventana de inicio con los elementos exigidos en el enunciado de la práctica.

Instrucciones para Iniciar Sesión:

1. Dirigirse al apartado de la parte inferior derecha de la ventana de inicio cuyo enunciado es *"Ingresa tus datos para iniciar sesión:"*.
2. En el campo Usuario/Correo ingresar "**Jaime Guzman**". (Textualmente respetado el case sensitive)
3. En el campo Contraseña ingresar "**12345**".
4. Presionar el botón *"Ingresar"*.

Instrucciones del Menú Procesos y Consultas:

Imagen 11. Menú procesos y consultas

1. Ejecutar las instrucciones para iniciar sesión.

4.1.1. Crear una cuenta:

4.1.1.1. Se verifica que no se hayan creado ya el máximo de cuentas permitidas por el nivel de suscripción del usuario. Si se ha alcanzado el máximo puede escoger "*Yes*" en la ventana emergente para modificar su nivel de suscripción, por otro lado, si escoge *"No"*

entonces será regresado a la pantalla inicial. Si su elección fue *"Yes"* será redirigido a la opción modificar mi suscripción.

4.1.1.2. Deberá insertar el nombre del banco al que desea asociar la cuenta respetando el formato y el case sensitive.

4.1.1.3. Deberá insertar la palabra Ahorros o Corriente en función de su deseo de elección. La cuenta de Ahorros maneja un saldo, mientras que la cuenta Corriente maneja un cupo y un disponible.

4.1.1.4. Deberá ingresar la clave de la cuenta, el formato es libre y puede escoger cualquiera.

4.1.1.5. Deberá insertar el valor de la divisa en la que va a manejar el dinero de la cuenta respetando el formato y el case sensitive.

4.1.1.6. Deberá insertar el nombre de la cuenta.

4.1.1.7. Deberá presionar el botón *"Continuar".*

4.1.1.8. Espere el mensaje de retorno.

4.1.1.9. Finalmente, deberá presionar el botón *"Volver al menú principal"* para volver al menú principal.

4.1.2. Ver mis cuentas:

4.1.2.1. Una vez entra a esta opción se muestran las cuentas del usuario.

4.1.2.2. Finalmente, deberá presionar el botón *"Volver al menú principal"* para volver al menú principal.

4.1.3. Modificar mi suscripción:

4.1.3.1. Deberá seleccionar del menú desplegable el nombre de un banco asociado al usuario para realizar la comprobación del nivel de suscripción del mismo (puede seleccionar cualquiera de entre los bancos disponibles), luego debe presionar *"Aceptar"*.

4.1.3.2. Deberá seleccionar el botón *"Si"* para continuar con el proceso, de lo contrario puede seleccionar el botón *"N"* para regresar al menú principal.

4.1.3.3. Deberá seleccionar del menú desplegable el nivel de suscripción que desea asignarle al usuario(puede seleccionar cualquiera de entre los niveles disponibles), luego debe presionar *"Aceptar"*.

4.1.3.4. Espere el mensaje de retorno.

4.1.3.5. Finalmente, deberá presionar el botón *"Volver al menú principal"* para volver al menú principal.

4.1.4. Invertir saldo en mi cuenta:

4.1.4.1. Deberá seleccionar del menú desplegable el nombre de una cuenta de ahorros asociada al usuario para realizar la inversión del saldo de la misma (puede seleccionar cualquiera de entre las cuentas disponibles), luego debe presionar *"Aceptar"*.

4.1.4.2. Se verifica que la cuenta seleccionada tenga saldo. Si la cuenta seleccionada no tiene saldo puede escoger "*Yes"* en la ventana emergente para consignarle saldo, por otro lado, si escoge *"No"* entonces será regresado a la pantalla inicial. Si su elección fue *"Yes"* será redirigido a la opción consignar saldo a mi cuenta.

4.1.4.3. Espere el mensaje de retorno.

4.1.4.4. Finalmente, deberá presionar el botón *"Volver al menú principal"* para volver al menú principal.

4.1.5. Consignar saldo en mi cuenta:

4.1.5.1. Deberá seleccionar del menú desplegable el nombre de una cuenta de ahorros asociada al usuario para realizar la consignación del saldo (puede seleccionar cualquiera de entre las cuentas disponibles), luego debe presionar *"Aceptar"*.

4.1.5.2. Deberá ingresar el saldo que le desea consignar a la cuenta, el formato es numérico y puede escoger cualquiera.

4.1.5.3. Espere el mensaje de retorno.

4.1.5.4. Finalmente, deberá presionar el botón *"Volver al menú principal"* para volver al menú principal.

4.1.6. Transferir saldo entre mis cuentas:

4.1.6.1. Deberá seleccionar el botón correspondiente a una de las dos opciones para transferir el saldo: seleccione *"2. Cuenta externa"* para

transferir a una cuenta de ahorros externa al usuario o seleccione *"1. Cuenta propia"* para transferir a una cuenta de ahorros propia al usuario.

4.1.6.2. Si la opción escogida es *"1. Cuenta propia*, entonces deberá seleccionar del menú desplegable el nombre de una cuenta de ahorros asociada al usuario para realizar la transferencia del saldo, esta última fungirá como orígen (puede seleccionar cualquiera de entre las cuentas disponibles), luego debe presionar *"Aceptar"*.

4.1.6.2.1. Deberá seleccionar del menú desplegable el nombre de una cuenta de ahorros asociada al usuario para realizar la transferencia del saldo, esta última fungirá como destino (puede seleccionar cualquiera de entre las cuentas disponibles), luego debe presionar *"Aceptar"*.

4.1.6.2.2. Deberá ingresar el saldo que desea transferir desde la cuenta origen a la cuenta destino, el formato es numérico y puede escoger cualquiera, luego debe presionar *"Continuar"*.

4.1.6.2.3. Espere el mensaje de retorno.

4.1.6.2.4. Finalmente, deberá presionar el botón *"Volver al menú principal"* para volver al menú principal.

4.1.6.3. Si la opción escogida es *2. Cuenta externa"*, entonces deberá seleccionar del menú desplegable el nombre de una cuenta de ahorros asociada al usuario para realizar la transferencia del saldo, esta última fungirá como orígen (puede seleccionar cualquiera de entre las cuentas disponibles), luego debe presionar *"Aceptar"*.

4.1.6.3.1. Deberá seleccionar del menú desplegable el nombre de una cuenta de ahorros para realizar la transferencia del saldo, esta última fungirá como destino (puede seleccionar cualquiera de entre las cuentas disponibles), luego debe presionar *"Aceptar"*.

4.1.6.3.2. Deberá ingresar el saldo que desea transferir desde la cuenta origen a la cuenta destino, el formato es numérico y puede escoger cualquiera, luego debe presionar *"Continuar"*.

4.1.6.3.3. Espere el mensaje de retorno.

4.1.6.3.4. Finalmente, deberá presionar el botón *"Volver al menú principal"* para volver al menú principal.

4.1.7. Compra con cuenta corriente:

4.1.7.1.

4.1.8. Gestionar mis préstamos:

Para gestionar tus préstamos tienes dos opciones en el menú, pagar un préstamo o pedir un préstamo .

Pedir Prestamo

4.1.8.1. Diríjase al menú desplegable a "Gestionar Préstamos" y seleccione la opción "Pedir Prestamo"

4.1.8.2. Seleccione la cuenta de AHORROS con la que desea pedir un préstamo

4.1.8.3. Ingrese la cantidad de dinero que desea pedir prestado sin superar la máxima cantidad que le presta el banco.

4.1.8.4. Presione el botón aceptar. Se realizará el préstamo y se le ingresara el dinero a la cuenta seleccionada anteriormente.

Pagar Presta

4.1.8.5. Diríjase al menú desplegable a "Gestionar Préstamos" y seleccione la opción "Pagar Préstamo"

4.1.8.6. Seleccione el préstamo que desea pagar

4.1.8.7. Ingrese la cantidad del préstamo que desea pagar

4.1.8.8. Presione el botón aceptar

4.1.9. Asesoramiento de inversiones:

4.1.9.1. Dar click en el botón "Comenzar".

4.1.9.2. Ingresar un valor del menú "Tolerancia a Riesgos" y darle click a "Siguiente".

4.1.9.3. Ingresar un número entero en “¿Cuánto deseas invertir?” y darle click a “Siguiente”.
4.1.9.4. Seleccionar si desea cambiar la fecha o no. Se recomienda dar click en Sí para ver todo lo que ofrece la funcionalidad.
4.1.9.5. Si la respuesta anterior fue Sí, ingrese la nueva fecha de la meta en el formato dd/mm/yyyy (por ejemplo 01/01/2025) y de click en el botón “Continuar”. De lo contrario continúe al siguiente paso.
4.1.9.6. Darle click a “Siguiente”.
4.1.9.7. Seleccionar si desea crear la meta o no. Se recomienda dar click en Sí para ver todo lo que ofrece la funcionalidad.
4.1.9.8. Confirme el resultado y darle click a “Siguiente”.
4.1.9.9. Darle click a “Siguiente”.
4.1.9.10. Seleccionar si desea hacer el préstamo o no. Se recomienda dar click en Sí para ver todo lo que ofrece la funcionalidad.
4.1.9.11. Si su respuesta anterior fue Sí, darle click a “continuar”. De lo contrario continúe al paso 14.
4.1.9.12. Ingrese el monto que desea solicitar prestado y de click en el botón “Continuar”.
4.1.9.13. Confirmar el resultado y darle click a “Siguiente”.
4.1.9.14. Si desea volver a comenzar de click en el botón “Reiniciar”.

4.1.10. Compra de cartera:

4.1.10.1. En la lista que se despliega y con base en la información de la tabla, escoger la cuenta a la cual desea aplicar el mecanismo financiero.
4.1.10.2. Confirmar o rechazar la elección hecha con cada botón. De ser negativa la respuesta se vuelve a la impresión de las cuentas (4.1.10.1), de lo contrario se continúa.
4.1.10.3. De la lista que se despliega y con base en la información de la tabla, escoger la cuenta a la cual queremos enviar la deuda.
4.1.10.4. Confirmar o rechazar la elección hecha con cada botón. De ser negativa la respuesta se vuelve a la impresión de las cuentas (4.1.10.3), de lo contrario se continúa.
4.1.10.5. Escoger entre mantener la periodicidad del pago de la deuda o no según se desee, se recomienda darle no para ver la totalidad de la función.
4.1.10.6. De la lista que se despliega escoger la cantidad de cuotas que más se ajuste a nuestro gusto.
4.1.10.7. Confirmar o rechazar la elección hecha con cada botón. De ser negativa la respuesta se vuelve a la impresión de las cuentas (4.1.10.6), de lo contrario se continúa.
4.1.10.8. Escoger entre mantener pagar intereses en el primer mes o no, no hay diferencia en la ejecución de uno u otro.
4.1.10.9. Confirmar o denegar la realización del movimiento.

4.1.11. Cambio de divisa

4.1.11.1. Dar click a comenzar
4.1.11.2. Dar “aceptar” en el mensaje entrante
4.1.11.3. Se dará la bienvenida al servicio
4.1.11.4. Se pregunta qué tipo de cambio desea efectuar. Escoja libremente cualquiera de los dos.
4.1.11.5. Aparecerán dos ‘combobox’, se les está pidiendo escoja una divisa en cada uno. Por favor escoja divisas distintas en cada caja, de lo contrario un error del usuario se hará cargo de la excepción.
4.1.11.6. Aparecerá un mensaje entrante. Dos opciones, no hay bancos posibles para el cambio, sí hay bancos posibles para el cambio. Le aparecerá el segundo.
4.1.11.7. Podría aparecerle un mensaje de error si usted no tiene cuentas con la divisa de origen escogida.
4.1.11.8. Se pregunta si desea seguir con el proceso
4.1.11.9. Si la respuesta anterior es positiva. Se da a escoger entre sus cuentas que cumplan con utilizar la divisa de destino, para que usted reciba el dinero cambiado.

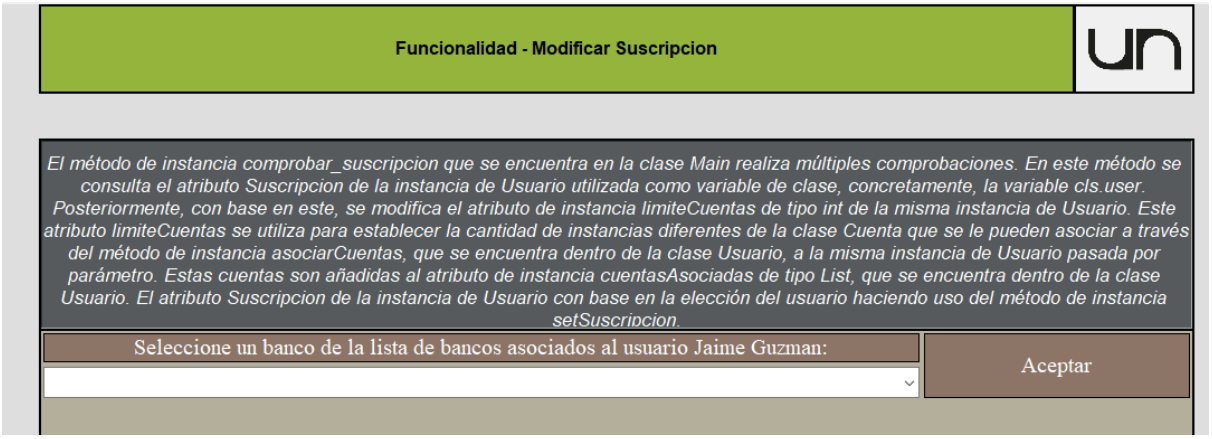
4.1.11.10. Una vez hecha la selección. Se verifica usted cuente con suficientes fondos en la cuenta de origen, sino se sale de la funcionalidad.

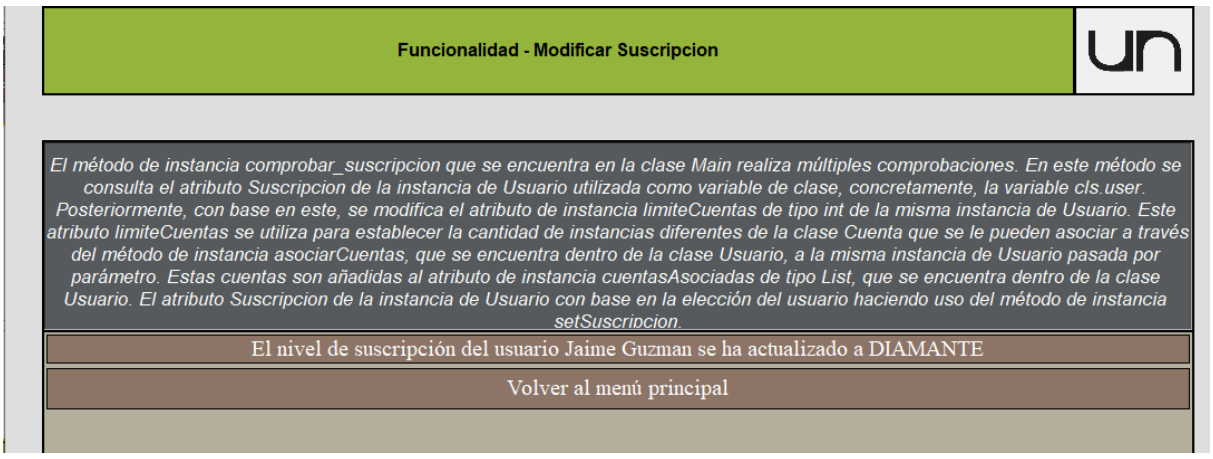
4.1.11.11. Finalmente se efectúa el cambio, aparece un mensaje de información concluyendo que el cambio fue efectuado exitosamente.

5. *Descripción de cada una de las 5 funcionalidades implementadas (incluye la descripción de la funcionalidad, que objetos intervienen en su implementación con un breve modelo de la secuencia del proceso, y por último, Incorporar una captura de pantalla con los resultados que presenta al usuario):*

5.1. Funcionalidad de Suscripciones de Usuarios:

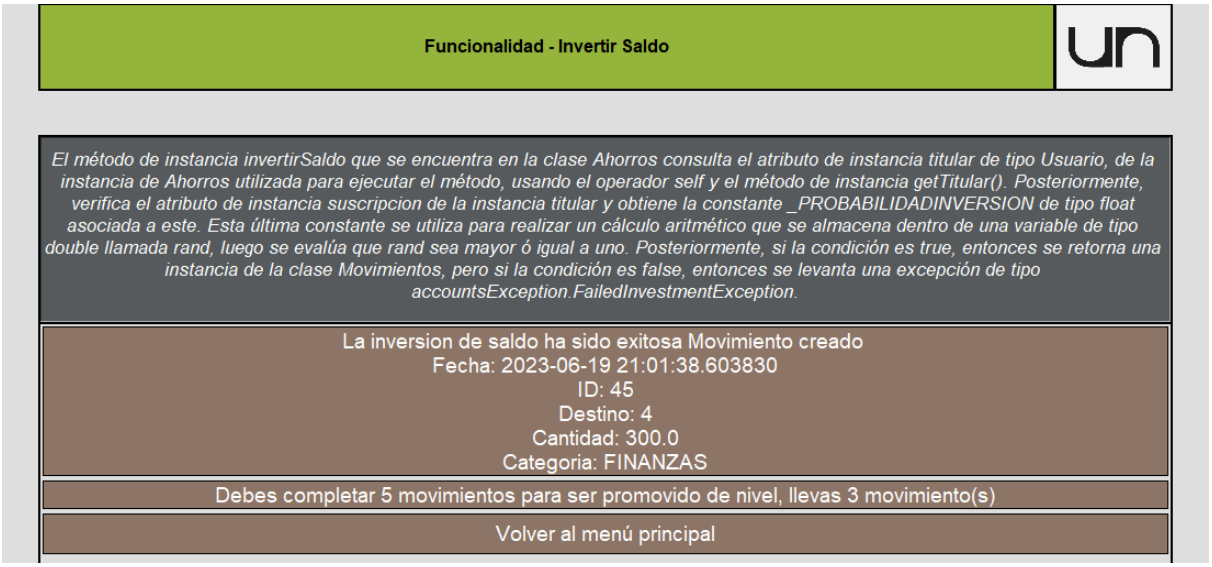
5.1.1. **Método comprobarSuscripcion**: El método de instancia *comprobar_suscripcion* que se encuentra en la clase *Main* realiza múltiples comprobaciones. En este método se consulta el atributo *Suscripcion* de la instancia de *Usuario* utilizada como variable de clase, concretamente, la variable cls.user. Posteriormente, con base en este, se modifica el atributo de instancia *limiteCuentas* de tipo *int* de la misma instancia de *Usuario*. Este atributo *limiteCuentas* se utiliza para establecer la cantidad de instancias diferentes de la clase *Cuenta* que se le pueden asociar a través del método de instancia *asociarCuentas,* que se encuentra dentro de la clase *Usuario*, a la misma instancia de *Usuario* pasada por parámetro. Estas cuentas son añadidas al atributo de instancia *cuentasAsociadas* de tipo *List*, que se encuentra dentro de la clase *Usuario*. El atributo *Suscripcion* de la instancia de *Usuario* con base en la elección del usuario haciendo uso del método de instancia *setSuscripcion*.





5.1.2. **Método crearMovimiento**: El método estático *crearMovimiento* que se encuentra en la clase *Movimientos* recibe como parámetros dos instancias de la clase *Ahorros* llamadas *destino* y *origen*, un enum de *Categoria*, un dato de tipo *double* llamado *cantidad* y un objeto de tipo *datetime* llamado *fecha*. Este método verifica que origen no sea None, de ser así, entonces se retorna una instancia de la clase *Movimientos* que es luego asociada a la instancia de *Usuario* pasada por parámetro usando el método de instancia *asociarMovimiento* de la clase *Usuario*. Por otra parte, en el método estático *CrearMovimiento* se consultan los atributos *_tasa_impuestos* y *comision*. El primero está asociado a la instancia de *Estado*, ésta última se obtiene usando el método *getEstadoAsociado()* usando la instancia de *Banco* asociada a la cuenta de *Ahorros* destino pasada por parámetro, esta instancia de *Banco* se obtiene usando el método de instancia *getBanco()* de la clase *Cuenta*. El segundo se obtiene haciendo uso de la la instancia de *Banco* asociada a la cuenta de *Ahorros* destino pasada por parámetro, llamando al método *getComision()* de la clase *Banco*.

5.1.3. **Método invertirSaldo**: El método de instancia *invertirSaldo* que se encuentra en la clase *Ahorros* consulta el atributo de instancia *titular* de tipo *Usuario*, de la instancia de *Ahorros* utilizada para ejecutar el método, usando el operador *self* y el método de instancia *getTitular()*. Posteriormente, verifica el atributo de instancia *suscripcion* de la instancia *titular* y obtiene la constante *_PROBABILIDADINVERSION* de tipo *float* asociada a este. Esta última constante se utiliza para realizar un cálculo aritmético que se almacena dentro de una variable de tipo double llamada *rand*, luego se evalúa que rand sea mayor ó igual a uno. Posteriormente, si la condición es true, entonces se retorna una instancia de la clase *Movimientos*, pero si la condición es false, entonces se levanta una excepción de tipo *accountsException.FailedInvestmentException*.



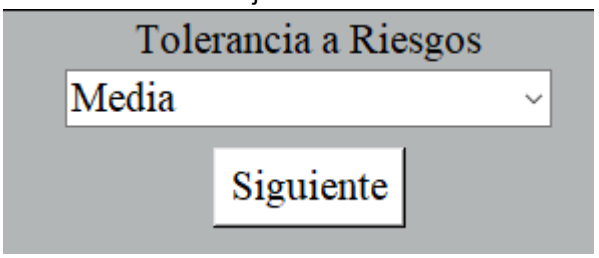
5.2. Funcionalidad Asesoramiento de inversiones:

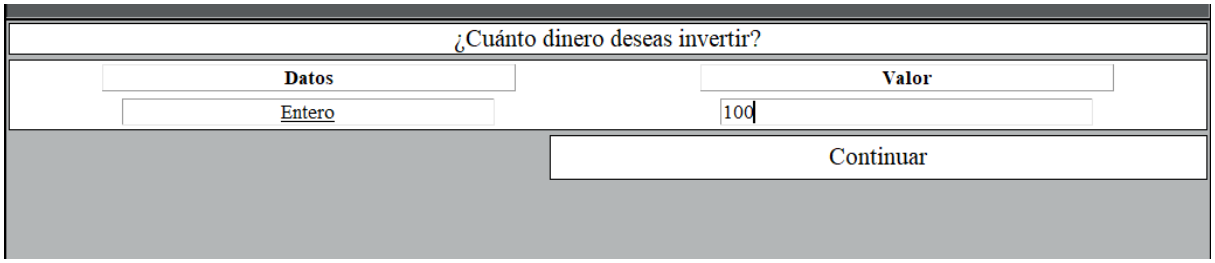
Diagrama de la funcionalidad:

Diagrama de la funcionalidad (Debe crear una cuenta en LucidChart para revisar el diagrama desde ésta página) **ó** *Ver figura de la tabla*

Funcionamiento:
La funcionalidad da una recomendación de un portafolio de inversiones en base a las preferencias y características del usuario, como las fechas de sus metas y sus movimientos o el dinero que hay en sus cuentas. Además, provee herramientas que pretenden mejorar aún más la inversión para la satisfacción del usuario.

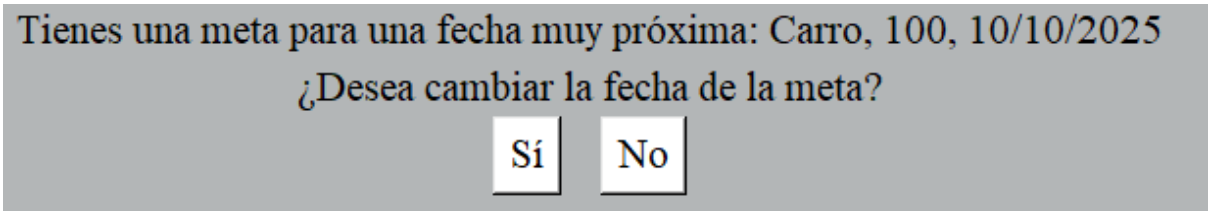
Al dar click en el botón “Comenzar” se le pide al usuario que ingrese su tolerancia al riesgo y la cantidad que desea invertir. Cuando el usuario da click en el botón “Siguiente” antes de continuar se revisa que el contenido de los anteriores dos campos sea correcto. Para esto se usa el manejo de errores, que van a confirmar que el valor de tolerancia a riesgos sea ‘Bajo’, ‘Medio’ o ‘Alto’ y que el valor de la cantidad que desea invertir sea un número entero. Luego de hacer estas confirmaciones se ejecutan estos métodos:



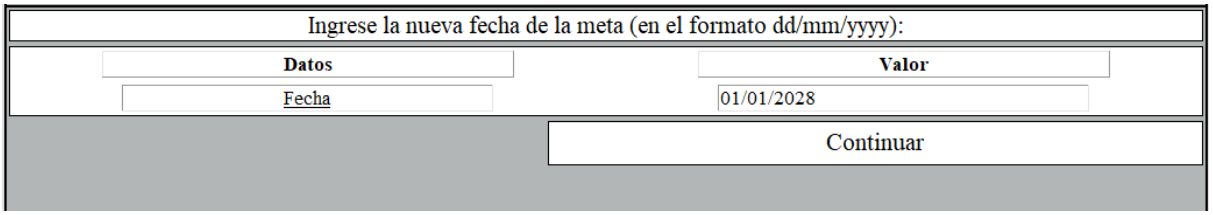


revision_metas():Ubicado en la clase Metas, su objetivo es encontrar la próxima meta asociada al usuario que tenga una fecha establecida. El método itera sobre todas las metas asociadas al usuario y busca la meta con la fecha más cercana en el futuro haciendo comparaciones entre estas.

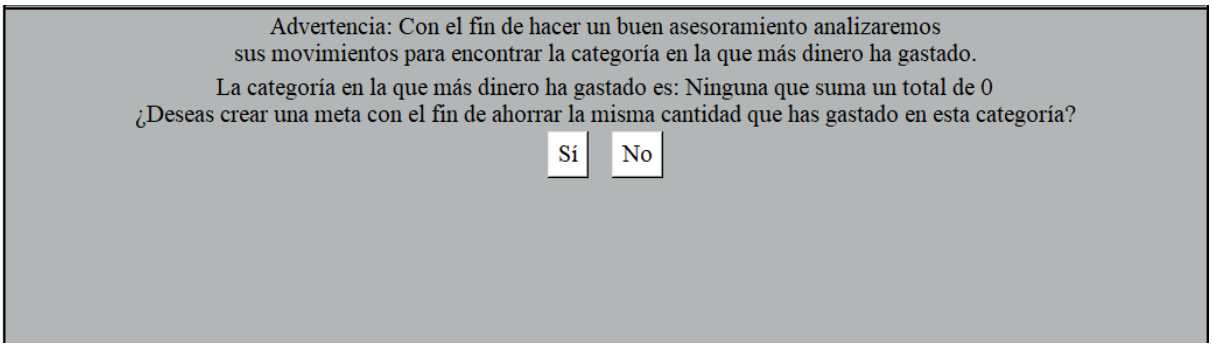
*Se le pregunta al usuario si desea cambiar la fecha de la meta, si su respuesta es que si se ejecuta el siguiente método



cambio_fecha(): Ubicado en la clase Metas, se encarga de cambiar la fecha de la meta haciendo un setFecha().



determinar_plazo(): Ubicado en la clase Metas, su objetivo es determinar el plazo de una meta en función de su fecha. El método compara la fecha de la meta con dos fechas de referencia: date1 y date2, utilizando el módulo datetime para convertir las cadenas de fecha en objetos de fecha y hora.
analizar_categoria(): Ubicado en la clase Movimientos, analiza los movimientos de un usuario y determina la categoría principal según la cantidad de ocurrencias de cada categoría. Luego, establece el nombre de la categoría principal y calcula la cantidad total en esa categoría. Además, determina una fecha recomendada para la categoría principal en función del plazo dado. Finalmente, crea una nueva meta con la categoría principal y la asocia al usuario, y luego prioriza las metas del usuario según la nueva meta.

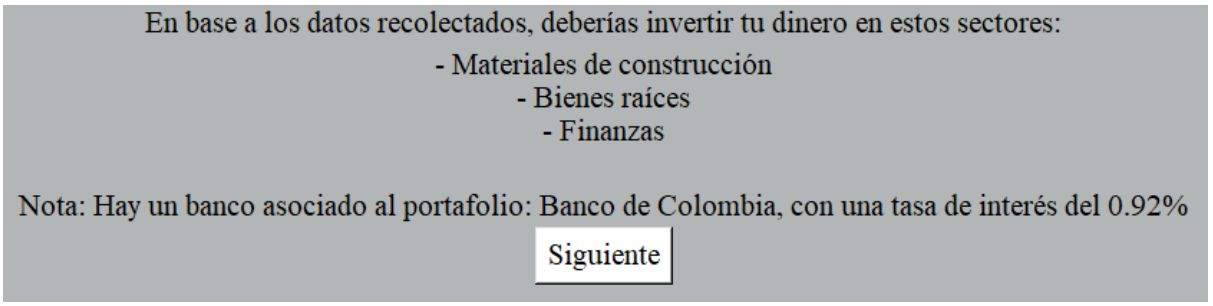


prioridad_metas(): Ubicado en la clase Metas, se encarga de poner la lista que recibe como parámetro de primera en la lista de metas asociadas al usuario

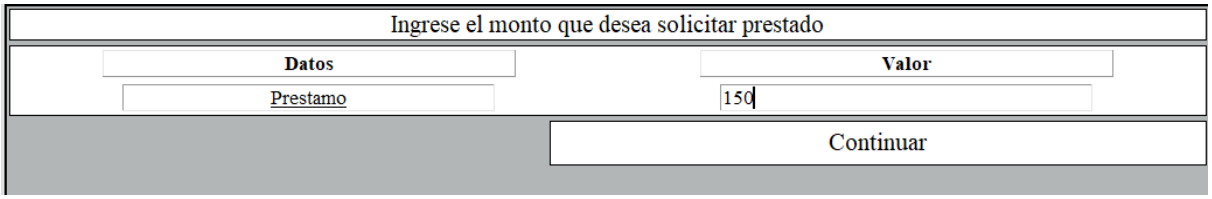
retorno_portafolio(): Ubicado en la clase Banco, el método determina el nivel de riesgo, calcula el monto a cobrar según el saldo/disponible de la cuenta, crea un movimiento de inversión, aplica impuestos al movimiento y devuelve un código numérico que representa el resultado de la inversión.

impuestos_movimiento(): Ubicado en la clase Movimientos, el método crea una cuenta de impuestos, verifica si los bancos de origen y destino son iguales y luego crea un movimiento de impuestos con una cantidad modificada. Luego, elimina el movimiento y la cuenta de impuestos de las listas correspondientes. Finalmente, devuelve un valor booleano para indicar si se aplicaron los impuestos al movimiento o no.

banco_portafolio(): Ubicado en la clase Banco, el método busca el banco asociado al usuario en su portafolio financiero. Si el usuario tiene solo un banco asociado, devuelve ese banco. Si el usuario tiene más de un banco asociado, devuelve el primer banco que sea diferente al banco anterior en la lista de bancos asociados.



gota_gota(): Ubicado en la clase Cuenta, el método busca la cuenta con el saldo o disponible más alto del usuario y crea una transacción de préstamo utilizando esa cuenta como destino. Luego, remueve la transacción de préstamo de la lista de movimientos y devuelve la cuenta seleccionada.
vaciar_cuenta(): Ubicado en la clase Cuenta, y en sus clases hijas Ahorros y Corriente, dependiendo del tipo de cuenta con el que se llame al método se creará un movimiento entre la cuenta del usuario y la del gota gota, por la cantidad de saldo, o bien disponible, que tenga la cuenta. Así, la cuenta del usuario se quedará sin dinero.



Finalmente aparece un mensaje de despedida y un botón “Reiniciar” por si se quiere volver a comenzar la funcionalidad.

5.3. Funcionalidad Préstamos y Deudas:

Esta Funcionalidad tiene la intención de crear un sistema de préstamos que el usuario pueda realizar en caso de que lo necesite.En esta funcionalidad intervienen 5 clases diferentes, la clase Usuario, la clase Ahorros, la clase Deuda, la clase Movimiento,la clase Banco; estas clases interactúan entre sí de diferentes maneras

- **Usuario:** es el que interactúa con la interfaz y es al que se le asigna la deuda y es el dueño de las cuentas, además nos dice el tipo de suscripción que nos da las condiciones para realizar préstamos. Esta clase contiene el método ComprobarConfiabiliad() en la línea 183 el cual comprueba si el usuario puede realizar préstamos. Para hacerlo es necesario que tenga más de una cuenta de ahorros asociadas, que los bancos de estas cuentas presten dinero y su suscripción le permita realizar una suscripción más.
- **Banco:** Con su atributo préstamo nos dice la cantidad máxima que se puede prestar en cada una de las cuentas dependiendo del banco al que está asociado la cuenta.
- **Ahorros:** Son las cuentas del usuario con las que puede hacer préstamos, estas deben ser de esta clase ahorros. En esta clase se encuentra el método comprobarPrestamo() el cual se encuentra en la línea 51. Este método consigue las cuentas del usuario y consigue el banco asociado a cada cuenta, luego comprueba que el atributo préstamo del banco sea diferente de 0, y retorna una lista con las cuentas las cuales su préstamo es diferente de 0.
- **Movimientos:** En esta clase se encuentran dos metodos RealizarPrestamo() en la línea 207 y PagarDeuda() en la línea 215. RealizarPrestamo() se encarga de hacer las últimas comprobaciones antes de efectuar el préstamo y crea un movimiento

ingresando el dinero a la cuenta; el método PagarDeuda() realiza un movimiento restando el dinero de la cuenta y disminuyendo la deuda, en caso de que se pague el total de la cuenta se elimina esta.

- **Deuda:**Las deudas son instancias de la clase deuda,al crear un prestamo se crea una instancia de esta clase, además de contar con el metodo conseguirDeuda() de la línea 11 que busca las deudas correspondientes al usuario debido a que es necesario en varias ocasiones para el correcto funcionamiento de la funcionalidad.

Ver diagrama de la funcionalidad Figura 24

El usuario ingresa al menú despegable a “Gestionar Prestamos”,tiene dos opciones,Pedir Préstamo y Pagar Préstamo.
Pedir Préstamo: En caso de elegir la primera se ejecuta el método comprobarConfiabilidad() comprueba que el usuario tenga cuentas de ahorros creadas, en caso de que no, se le muestra el error al usuario y se termina la funcionalidad, consigue las deudas del usuario con conseguirDeudas() y se comprueba que su suscripción le permite realizar como mínimo un préstamo, en caso de que no pueda realizar un préstamo se le muestra el error.

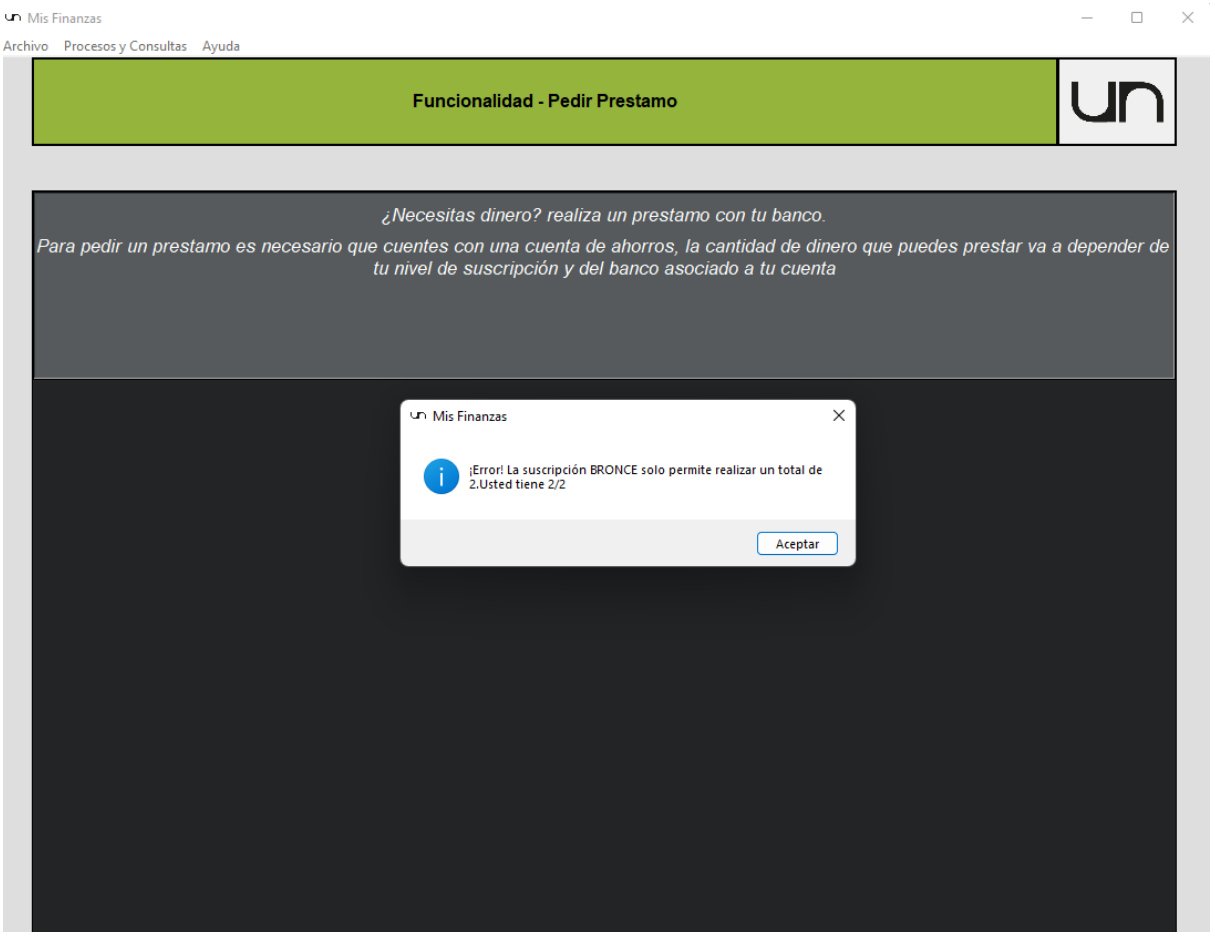
En caso de que las dos anteriores comprobaciones sean correctas se retorna una lista con las cuentas y se ejecuta el método (2) ComprobarPrestamo() este método recibe las cuentas y como se puede dar que algún banco no realice préstamos es decir que su atributo préstamo sea 0, se consiguen las cuentas que realizan préstamos y se agregan a un arreglo, en caso de que este arreglo sea de tamaño 0, se le muestra al usuario el error de que ninguno de sus bancos realiza préstamos, en el caso contrario se le muestra al usuario las cuentas con las que puede realizar préstamos y se le pide que seleccione una.

(3) Al momento de seleccionar una cuenta se le pide que ingrese la cantidad del préstamo que desea realizar y se ejecuta el metodo realizarPréstamo() el cual recibe la cuenta y la cantidad, comprueba que la cantidad ingresada por el usuario no exceda la cantidad permitida por el banco de la cuenta y el multiplicador de su suscripción, en caso que de no exceda este valor se crea la instancia de la deuda y se realiza el movimiento, en caso contrario se le muestra el error al usuario.

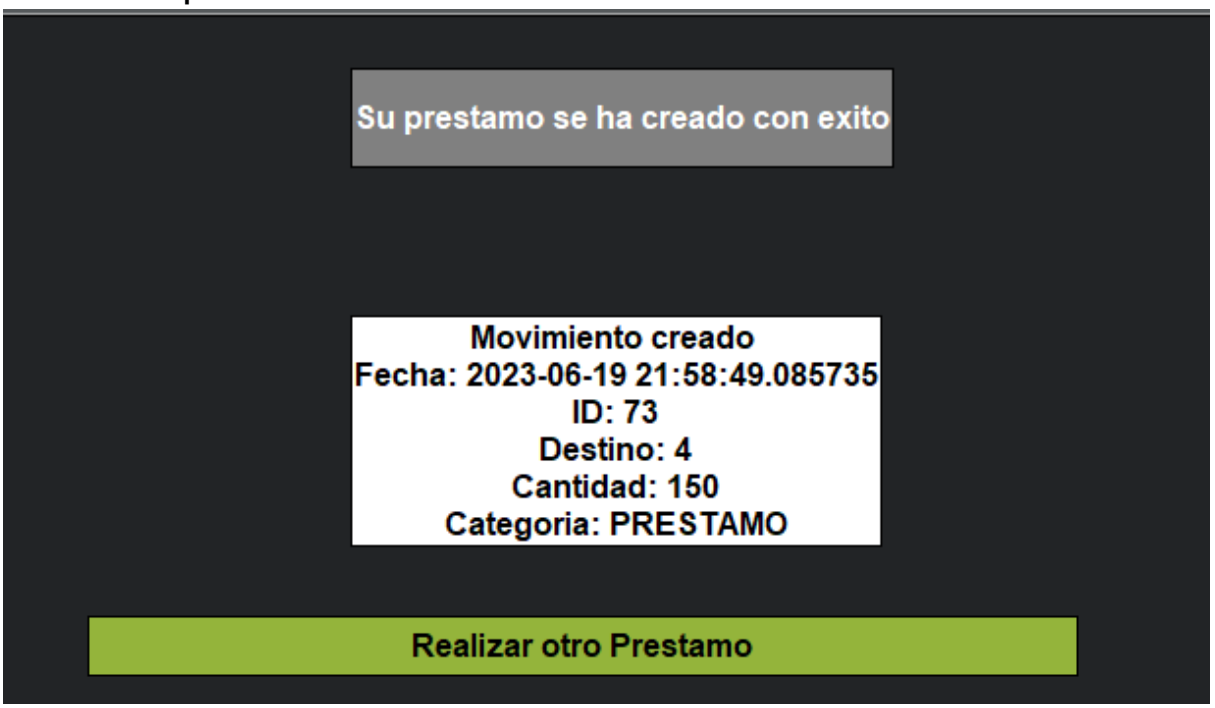
Pagar Préstamo: (4)Con el método de la clase deuda ConseguirDeudas() se buscan las deudas que corresponden al usuario, en caso de que no tenga deudas asociadas se le imprime al usuario el mensaje “Usted no tiene deudas por pagar”, en caso de que si tenga deudas, se le muestran al usuario estas deudas y se le pide al usuario que seleccione la que desee pagar y la cantidad a pagar,(5) se le pasa la cantidad con la deuda y el usuario al método pagarDeuda(), se comprueba la cantidad

Resultados del usuario

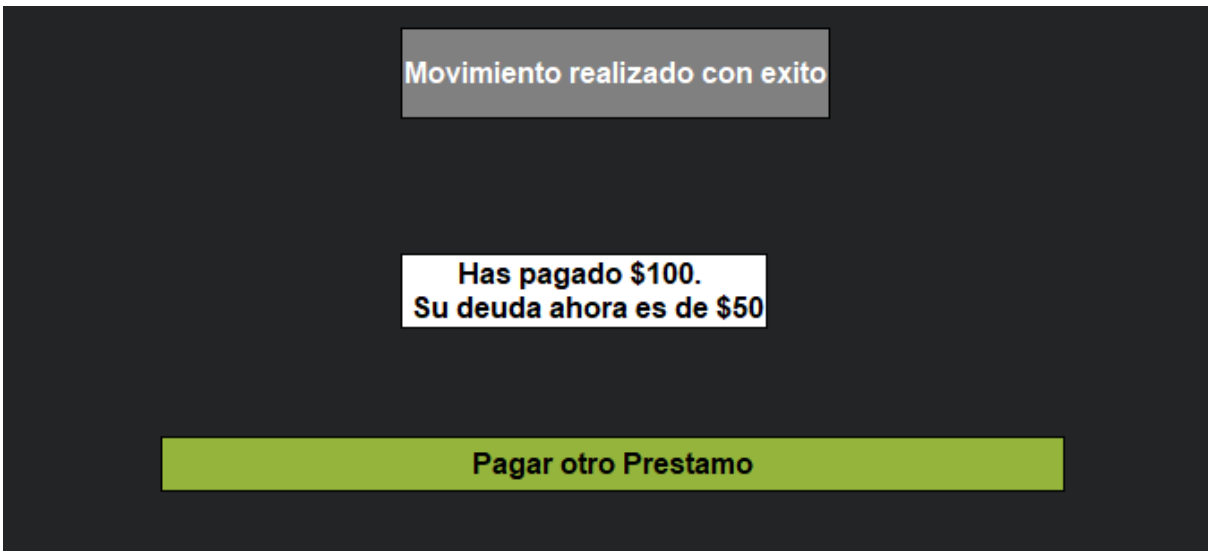
- **Caso 1: El usuario no puede pedir más préstamos**



- **Caso 2: El préstamo se realiza de manera exitosa**



- **Caso 3:Pagar Deuda**



5.4. Funcionalidad Compra de Cartera:

Diagrama de la funcionalidad:

[Diagrama de la funcionalidad] **(Debe crear una cuenta en LucidChart para revisar el diagrama desde ésta página) ó** **ver figura 25 de la Tabla**

Planteamiento:
Lo que se plantea con esta funcionalidad es emular el mecanismo financiero de la Compra de Cartera usado para aliviar momentáneamente el bolsillo de los usuarios que hacen uso de esta, mecanismo que lo que hace es transferir una deuda de una cuenta a otra con la posibilidad de recibir mejores intereses y/o con la posibilidad de cambiar el plazo de pago de dicha deuda. De esta manera, lo que se busca es que el usuario escoja una de las cuentas Corriente de las cuales tiene una deuda, luego realice el mismo proceso con una capaz de recibir dicha deuda verificando los intereses bancarios con los cuales quedaría para posteriormente realizar las comprobaciones correspondientes por parte del usuario y ejecutar los cambios necesarios en las cuentas correspondientes.

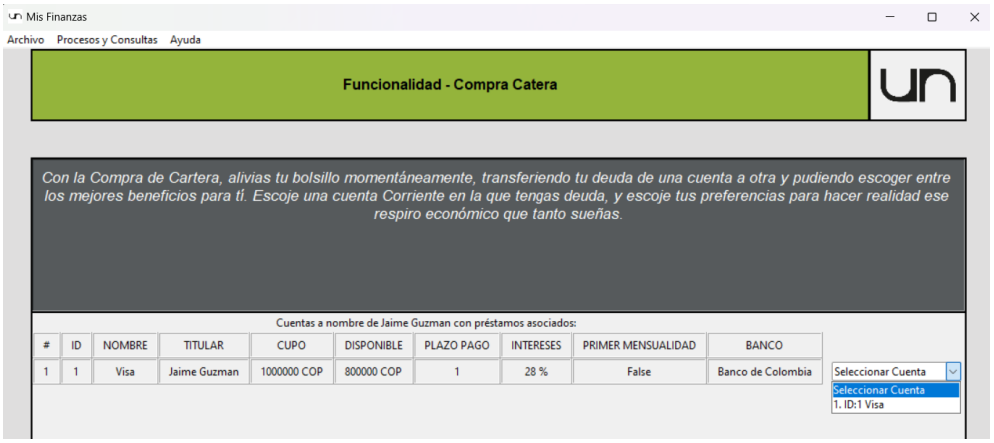
Funcionamiento:
La funcionalidad realiza los cambios necesarios a las cuentas escogidas por parte del usuario con base a sus preferencias en términos de cuotas e intereses. Además de esto, se brinda al usuario la posibilidad de una revisión completa de la información necesaria de la cuenta destino.

La funcionalidad comienza haciendo unas comprobaciones, con el fin de que la funcionalidad se pueda ejecutar. Estas primeras comprobaciones se basan en la verificación de la posesión por parte del usuario de mínimo dos cuentas corrientes y que al menos una de estas posea deuda alguna. Para la primera verificación se hace uso del getter del atributo CuentasCorrienteAsociadas y se evalúa que hayan por lo menos dos cuentas en este arreglo. Para la segunda comprobación se hace uso del siguiente método:

retornarDeudas(): Verifica cuales de las cuentas que posee un usuario tiene alguna deuda (lo hacer mediante la comparación entre los cupos y los disponibles de cada cuenta) para devolver un arreglo con las cuentas que satisfagan la condición.

Una vez obtenidas las cuentas con Deuda, se realiza la verificación de la existencia de al menos una deuda (con el tamaño del arreglo). Si no se pasa alguna de estas dos comprobaciones se otorga el aviso correspondiente al caso y se finaliza la funcionalidad.

Se realiza la impresión de las cuentas con deuda para que el usuario escoja a cual de estas cuentas con deuda desea aplicar el mecanismo financiero, elección realizada con un Combobox. Se realiza la impresión de la información de la cuenta escogida y se pide la confirmación de que dicha cuenta es justamente a la que se desea aplicar la funcionalidad. En caso de que se exprese que no es la cuenta escogida por el usuario, vuelve a la posible elección de la cuenta.

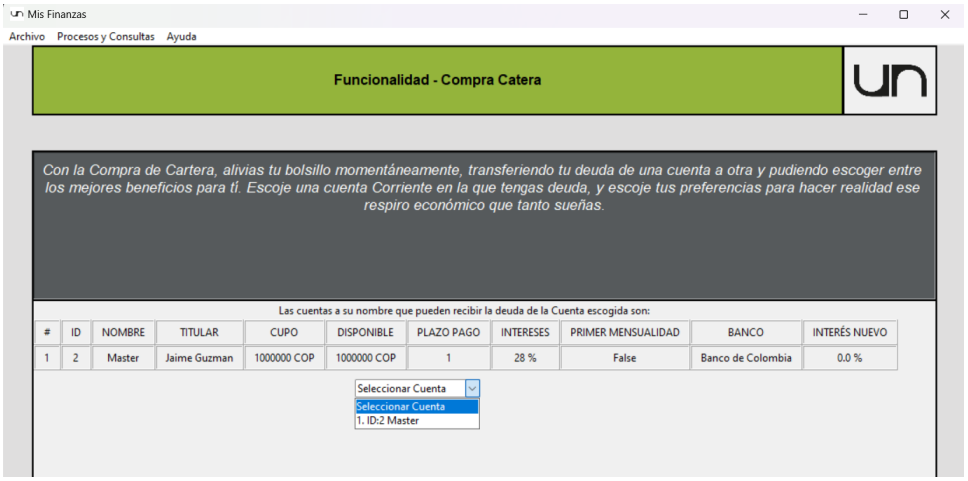


Se envía un arreglo con las cuentas asociadas al usuario (menos la cuenta escogida) y la cuenta escogida al método *Capacidad_Endeudamiento*, que validará las cuentas propias capaces de recibir la deuda de la cuenta escogida, luego de esto se enviará el arreglo recibido por este método a uno llamado *verificarTasasdeInteres* que devolverá las tasas de interes que cobrará cada uno de los bancos a las cuentas por recibir esta nueva deuda. A continuación, una explicación más detallada de los dos métodos usados:

capacidad_endeudamiento(Cuenta[], Corriente): Este método tiene como objetivo principal realizar las validaciones necesarias de las cuentas corriente asociadas a un usuario con el fin de devolver un arreglo con las cuentas que sean capaces de recibir la deuda de una cuenta ingresada.
verificar_tasas_de_interes(Usuario, Corriente[]): Este método de la clase Banco tiene como objetivo principal realizar la verificación de las tasas de interés de un conjunto de cuentas y retornarlas de manera ordenada. La idea de la validación se basa en que cada uno de los bancos brindará determinados descuentos a cada cuenta y a cada usuario, según su suscripción y la cantidad de movimientos que tenga asociados a un banco. Así pues, el método recibe dos parámetros, el primero de ellos un Usuario del cual se tomará el valor de Suscripción, el segundo un arreglo de cuentas Corriente sobre las cuales se determinará la tasa de interés. En este método se hace una iteración de las cuentas Corriente que vienen en el arreglo y las envía al método verificar_tasas_de_interes_1 que se describe más adelante, pero que devuelve la tasa de intereses a cobrar a cada cuenta, para luego añadirlo al *arreglo* que devuelve el método.

verificar_tasas de interes_1(Suscripción, Corriente): Este método de la clase Banco tiene como objetivo principal realizar la validación de la tasa de interés a aplicar de una cuenta corriente con base en su Suscripción y en su descuento por parte del Banco. Para realizarlo se reciben dos parámetros, el primero de ellos con la Suscripción del Usuario que será usada como casuística y el segundo de ellos una cuenta Corriente de la cual se extraerá el banco asociado con el fin de validar los descuentos necesarios.

Luego de esta serie de métodos, se verifica que el arreglo devuelto no esté vacío (esto nos indicará de que hay por lo menos una cuenta capaz de recibir la deuda), de no ser pasada la verificación se muestra el mensaje correspondiente y se finaliza la funcionalidad. De ser pasada la verificación, se continúa el programa con la impresión de las cuentas del arreglo en mención de manera que el usuario escoja la cuenta a la cual desea enviar la deuda escogida anteriormente. Dicha elección se almacena en el entero *cuenta_Destino*.



Luego de escoger la cuenta destino de la deuda, comienza el proceso en el cual el usuario escoge las preferencias con las cuales va a realizar la compra de cartera. Primero, se le pide al Usuario que escoja si desea mantener la periodicidad del pago de la cuenta que tiene (cantidad de cuotas en la que se salda la deuda de la cuenta destino). De ser afirmativa la respuesta se asigna al atributo *eleccion_periodicidad* de tipo Cuotas el atributo *plazo_Pago* de la cuenta destino.. Empero, de ser negativa esta respuesta se da la opción al usuario de que escoja la nueva periodicidad de la cuenta, esto se realiza mediante la impresión de las posibles cuotas y la elección del usuario. Posteriormente, según esta selección se asigna al atributo *eleccion_periodicidad* el valor correspondiente a la elección.

Una vez tenemos la preferencia del usuario frente a la periodicidad de su pago, se procede a realizar la vista previa de cómo quedaría la cuenta una vez se hayan realizado los cambios según las elecciones hechas por el usuario, para este fin, se guardan los valores antiguos de las cuentas y se realizan los cambios correspondientes.

A continuación, se realiza la validación de la cantidad de cuotas que fueron escogidas por el usuario, almacenadas en el atributo *elección_periodicidad* y si la cantidad de cuotas de este atributo (tomadas con *getCantidad_Cuotas()*) es mayor que 1, se brinda la posibilidad al usuario de que pueda realizar el pago de intereses en el primer mes del pago de la deuda, o si no desea realizarlo pero teniendo en cuenta de que tendrá que pagar dicho valor correspondiente en el segundo mes de cuota. Se realizan dichas validaciones y por tanto se tendrán dos casos, según la elección ya descrita.

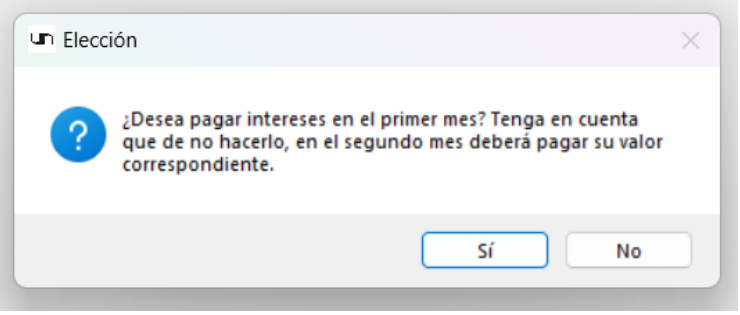
En el primero de ellos haremos uso del método *retornoCuotaMensual*, al que enviaremos la deuda que recibe el mecanismo financiero con el fin de retornar la primer cuota a pagar tomando en cuenta que se realizará pago de intereses en dicho mes. En el segundo caso

usaremos una sobrecarga emulada del mismo método, al cual enviaremos también la deuda a recibir y un 1, como referencia de que será el mes uno a pagar, este método sobrecargado toma en cuenta el caso en el cual no se pagarán intereses en el primer mes y obtendremos así la cuota del mismo. Además, según sea la opción se asignará *true* o *false* según sea el caso al atributo *primerMensualidad* de la cuenta asociada a la *vistaPrevia*, que almacena justamente la decisión ya descrita. A continuación se explican en detalle los métodos mencionados anteriormente.

retornoCuotaMensual (double): Este método de la clase Corriente tiene como objetivo principal hacer el cálculo de los valores a pagar en una cuota con la condición de que en el primer mes sí se hace el pago de intereses correspondientes. Es un método que recibe como parámetro un double que almacena la DeudaActual con la que llega el usuario para el cálculo de la cuota independiente del mes en cuestión. (Notar que el hecho de que la DeudaActual sea la DeudaTotal significa que se habla del primer mes de pago de cuota), y que devuelve la información de la cuota correspondiente. Este inicia creando un Array de tipo double, Array llamado *cuotaMensual* donde se añadirán los valores y la información a devolver por parte del método. Luego de esto, se llama al método estático de la clase *Corriente* llamado *calculoInteresNominalMensual*, al cual se le envían los intereses de la cuenta desde la cual se llama el método en el que estamos. Este método me devuelve el interés nominal mensual calculado a partir de los intereses de la cuenta (que hacen acolación al atributo *intereses* pero que referencia la tasa efectiva de interés anual de la cuenta. Este método se explica en detalle más adelante. Luego de tener la tasa de interés a usar se hacen cálculos de *interes*, *abono_capital* y *cuotaMensualFinal* de la cuota en cuestión, agregándolos en orden a las posiciones del Array a devolver y haciendo uso de los atributos de la cuenta de instancia correspondiente al método. Una vez hecho esto, se devuelve este Array con la información de la cuotaMensual.

retornoCuotaMensual(double, int): Este método es una réplica del método anterior pero en la condición en que en el primer mes no pagaremos intereses. Para esto, hacemos uso del segundo atributo que es ingresado como parámetro, en donde hacemos un condicional que cubre tres casos, el primero de ellos en donde estamos en el mes 1 y no pagamos intereses, el segundo en el cual se sumarán los intereses del mes 1 con los del mes 2 y el tercero para el mes 3 en adelante, en donde se copia el del método anterior.

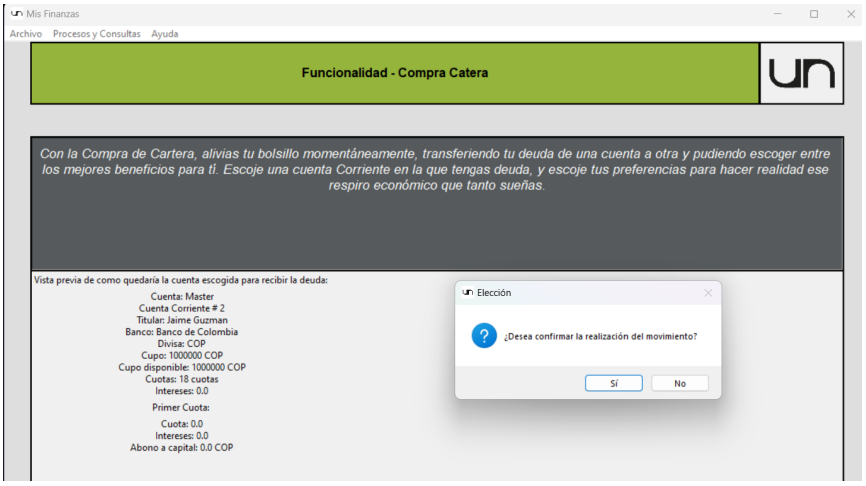
calculoInteresNominalMensual(double): Método estático de la clase *Corriente* que se encarga de hacer la conversión de la tasa efectiva anual (recibido por parámetro) por la tasa de interés nominal mensual. Son cálculos matemáticos basados en la realidad del cálculo de este tipo de datos.



Luego de recibir cuál será la primer cuota a pagar por parte del usuario según su elección, se procede a realizar la impresión de la *vista Previa* de la cuenta y de la primer cuota con el fin de que el usuario tenga claridad en su decisión y escoger lo mejor para su proyección monetaria.

Luego de este proceso, se procede con la validación de la realización del movimiento, para esto se da la opción al usuario de confirmar el deseo de realizar el movimiento, de ser afirmativa la respuesta, con los setters se reiniciarán los datos necesarios de la cuenta origen (disponible y plazo_Pago), y se muestra el mensaje de éxito.

De ser la respuesta negativa, se revierten los cambios y se dejan las cuentas que tenía el usuario tal cual como las tenía en un inicio. Una vez realizada la confirmación finaliza la funcionalidad.



5.5. Funcionalidad de Cambio de Divisa:

Planteamiento:
La funcionalidad de cambio de divisa se desarrolló con el propósito de solventar las disparidades de valor que existen entre diversas monedas durante un proceso de intercambio monetario. En un entorno económico, donde las monedas poseen valores fluctuantes, resulta esencial contar con una solución que permita equilibrar estas diferencias. El cambio de divisa proporciona la capacidad de convertir una moneda en otra, lo cual simplifica y agiliza las transacciones. Esta funcionalidad busca minimizar las barreras y dificultades asociadas con las disparidades de valor, lo que a su vez promueve una mayor eficiencia y estabilidad en los intercambios monetarios del programa. Además, el cambio de divisa también es una herramienta, al permitir a los individuos y empresas diversificar sus inversiones y activos en diferentes monedas.

Funcionamiento:

El método principal es “cambio_divisa()”. Al ser llamado, muestra un mensaje de bienvenida al servicio de cambio de divisa. Si la variable booleana novato es verdadera, se muestran explicaciones adicionales sobre los tipos de cambio disponibles: convencional y exacto.

A continuación, se solicita al usuario que elija el tipo de cambio deseado, en caso de escoger exacta se establece la variable booleana “exacta” como verdadera. Luego se solicita al usuario que seleccione la divisa desde la cual desea hacer el cambio. Se muestra la lista de divisas disponibles en el sistema y se lee la elección del usuario. La divisa seleccionada se almacena en la variable “divisa_origen”. A continuación, se buscan las cuentas de ahorro asociadas al usuario que tengan la misma divisa que “divisa_origen” y se almacenan en la lista “ahorrosPosibles”. Se muestra nuevamente la lista de divisas disponibles, excluyendo la previamente escogida, y se solicita al usuario que seleccione la divisa del destino. La divisa correspondiente se guarda en “divisa_destino”.

Se verifica si el cambio es exacto o convencional, y se muestra un mensaje para solicitar el monto a cambiar. El valor ingresado por el usuario se almacena en la variable “monto”.

A continuación, se crea un objeto de la clase Movimientos llamado “cambioDiv”, que representa el cambio de divisa a realizar. Se invoca el método estático “facilitarInformación()” de la clase Movimientos para buscar bancos que permitan realizar el cambio de divisa especificado. El resultado se almacena en la lista “existeCambio”.
facilitar_información() recibe un objeto movimiento con la divisa de origen, la de destino y el usuario. De los bancos asociados al usuario, les cambia el atributo asociado a true.

Luego para los bancos asociados al usuario, recorre los tipos de cambio de divisa del atributo “dic”, si encuentra un tipo de cambio igual al pedido, anota el banco en la lista “existe_cambio”.

Se verifica si existe al menos un banco disponible para el cambio en la lista “existe_cambio”. Si no hay bancos, se muestra un mensaje y se termina el proceso. Si hay bancos, se muestra la cantidad y se continúa. Luego se muestra una lista de las cuentas de ahorro disponibles en “ahorrosPosibles”, es decir, las cuentas del usuario con la divisa como la desde donde se va a hacer el cambio . Si no hay cuentas disponibles, se muestra un mensaje y se termina el proceso.

Se muestra una lista de cotizaciones posibles para el cambio de divisa y se utiliza el método estático “cotizar_taza()” de la clase Banco para obtener la información de cotización de los bancos disponibles. Las cotizaciones se almacenan en la lista “imprimir”.
cotizar_tasa() recibe el usuario, la lista “existe_cambio”, “cadena” que es la concatenación de los nombres de las divisas escogidas, y la lista “ahorrosPosibles”. Para cada ahorro de “ahorrosPosibles”, para cada banco de “existe cambio” se obtiene el valor de la tasa de cambio y se le hacen descuentos si es en el propio banco y se se está asociado.La cuota de manejo depende de la suscripción. Todos estos datos se almacenan en “imprimir” que es también un objeto de tipo movimientos, a modo de cotización.

A continuación, se muestra la lista de cotizaciones y se solicita al usuario que elija una opción. La cotización seleccionada se almacena en la variable “escogencia” que es un objeto de la clase Movimientos.

Se solicita al usuario que confirme si desea continuar con el proceso. Si la respuesta es negativa, se termina el proceso. A continuación, se solicita al usuario que elija la cuenta receptora del dinero en la divisa objetivo, mostrando una lista de opciones. La cuenta seleccionada se guarda en la variable “cuenta_destino”.

Si el cambio es exacto o no, se verifica si el usuario tiene suficientes fondos en la cuenta de origen para realizar el cambio, esto se hace mediante el método sobrecargado “comprobarSaldo()” . Si no hay suficientes fondos, se muestra un mensaje y se termina el proceso. Si hay suficientes fondos, se llama al método sobrecargado, para “exacta” o no, “hacer_cambio()” de la clase Cuenta para realizar el cambio de divisa con el monto especificado.
hacer_cambio() recibe la escogencia, el monto del cambio, la cuenta que lo recibe y el usuario. su multiplica por los valores escogidos en la cotización de “escogencia” y ser crea el objeto movimiento “m”, que esta vez sí efectúa cambios en los saldos de las dos cuentas implicadas.

Finalmente, se muestra una tabla indicando cómo quedan las cuentas después del cambio de divisa.

Diagrama Funcionalidad (Necesita de una cuenta de Lucidchart).

6. *Figuras:*.

Figura 1.1 Saludo de bienvenida

Bienvenidos al sistema de gestión financiera Mis Finanzas programado por:
->Juan Pablo Mejía Gómez.
->Leonard David Vivas Dallos.
->José Daniel Moreno Ceballos.
->Tomás Escobar Rivera.
->Jorge Humberto García Botero.

Figura 1.2 Código saludo de bienvenida

```
# ------------------------------------------------
# -------Texto de bienvenida(P3 - upper_left_frame)
--------------------
welcome_label_text_variable = "Bienvenidos al sistema de gestión
financiera Mis Finanzas programado por: \n->Juan Pablo Mejía
Gómez.\n->Leonard David Vivas Dallos.\n->José Daniel Moreno
Ceballos.\n->Tomás Escobar Rivera.\n->Jorge Humberto García
Botero."
welcome_label = tk.Text(upper_left_frame, cursor="cross",
fg="black", bg="white", font=(
    "Alegreya Sans", 12), wrap="word", spacing1=8, border=0)
welcome_label.insert(tk.INSERT, welcome_label_text_variable)
welcome_label.tag_configure("justifying", justify="center")
welcome_label.tag_add("justifying", "1.0", tk.END)
welcome_label.config(state="disabled")
welcome_label.pack(expand=True, fill="both",
                   anchor="s", padx=1, pady=20)
# ------------------------------------------------
```

Figura 2. Hoja de vida desarrollador

```
# -------Hoja de vida de los desarrolladores(P5 - upper_right_frame)--------------------
# Crear el botón y asociar la función change_button_text con él
button_developers_text = tk.StringVar(
    upper_right_frame, "1. Tomas Escobar Rivera.\n Soy un apasionado programador con experiencia en múltiples lenguajes de programación.
button_developers = Button(upper_right_frame, textvariable=button_developers_text, bg="white", command=change_button_text, font=(
    "Alegreya Sans", 12), activebackground="gray", activeforeground="white", border=1, relief="groove", cursor="cross", wraplength=450)
style = font.Font(family="Times New Roman", size=12)
button_developers.config(
    font=style, bg="#f8e5c7", border=2, relief="raised")
button_developers.pack(expand=True, fill="both")
# ------------------------------------------------
```

Figura 3. Código de fotos de desarrolladores

```
# -------Fotos de los desarrolladores(P6 - bottom_right_frame)----------------------
image_label = tk.Label(bottom_right_frame, cursor="cross")
image_label.pack(expand=True, fill="both")
update_image()
# ------------------------------------------------
```

Figura 4.1. Inicio de sesión

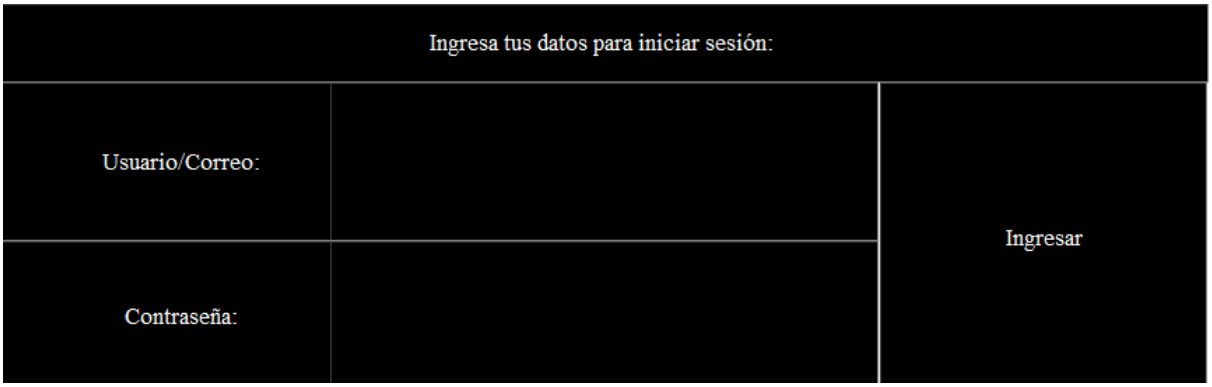


Figura 4.2. Código inicio de sesión

```
# -------Interfaz de acceso al sistema(P4 - bottom_left_frame)--------------------
# Crear un label para inicio de sesión.
style = font.Font(family="Times New Roman", size=13)
login_label = tk.Label(bottom_left_frame, text="Ingresa tus datos para iniciar sesión: ",
                       fg="white", bg="black", border=1, relief="sunken", font=style)
login_label.place(anchor="n", relheight=.1,
                  relwidth=.99, relx=0.5, rely=0.51)
# Crear un label con el usuario ó el correo.
user_email_label = tk.Label(bottom_left_frame, text="Usuario/Correo: ",
                            fg="white", bg="black", border=1, relief="sunken", font=style)
user_email_label.place(anchor="n", relheight=.20,
                       relwidth=.3, relx=0.156, rely=0.61)
# Crear un entry para recibir el usuario ó el correo del usuario.
user_email_entry = tk.Entry(
    bottom_left_frame, fg="white", bg="black", border=1, relief="sunken", font=style)
user_email_entry.place(anchor="n", relheight=.20,
                       relwidth=.45, relx=0.5, rely=0.61)
# Crear un label con el usuario ó el correo.
password_label = tk.Label(bottom_left_frame, text="Contraseña: ",
                          fg="white", bg="black", border=1, relief="sunken", font=style)
password_label.place(anchor="n", relheight=.19,
                     relwidth=.3, relx=0.156, rely=0.81)
# Crear un entry para recibir la contraseña del usuario.
password_entry = tk.Entry(
    bottom_left_frame, fg="white", bg="black", border=1, relief="sunken", font=style)
password_entry.place(anchor="n", relheight=.19,
                     relwidth=.45, relx=0.5, rely=0.81)
# Crear un botón para iniciar sesión.
login_button = tk.Button(bottom_left_frame, fg="white", bg="black", border=1, relief="sunken",
                         font=style, text="Ingresar", activebackground="gray", activeforeground="black", cursor="cross")
login_button.place(anchor="s", relheight=.39,
                   relwidth=.269, relx=0.860, rely=0.9999999)
login_button.bind("<Button-1>", login)
```

Figura 4.3. Código inicio de sesión

```
def login(event):
    name_email_user = str(user_email_entry.get())
    password_user = str(password_entry.get())
    try:
        possible_user = Usuario.verificarCredenciales(name_email_user, password_user)
    except usersException.NoUserFoundException:
        confirmation = messagebox.askretrycancel("Mis finanzas", usersException.NoUserFoundException.show_message())
        if confirmation:
            user_email_entry.delete(0, tk.END)
            password_entry.delete(0, tk.END)
        else:
            exit_initial_window()
    else:
        cls.user = possible_user
        exit_initial_window()
        App.start_main_window()
```

Figura 5. Imágenes asociadas al sistema

```python
# -------Imágenes asociadas al sistema(P4 - bottom_left_frame)----------
# Crear un label para mostrar la imagen.
system_image_label = tk.Label(
    bottom_left_frame, border=2, relief="groove", cursor="cross")
system_image_label.place(
    anchor="n", relheight=.5, relwidth=.99, relx=0.5, rely=0.01)

# Lista de las imágenes asociadas al sistema
image_paths = [
    route_logo,
    route_image,
    route_logo,
    route_image,
    route_logo
]

# Cargar las imágenes
images = [tk.PhotoImage(file=image_path) for image_path in image_paths]

# Mostrar la imagen inicial
current_image_index = 0
current_image = images[current_image_index]
system_image_label.config(image=current_image)

# Vincular el evento de poner el ratón sobre el label de la imagen
system_image_label.bind("<Enter>", change_system_image)
# ------------------------------------------------
```

Figura 6. Distribución ventana del usuario

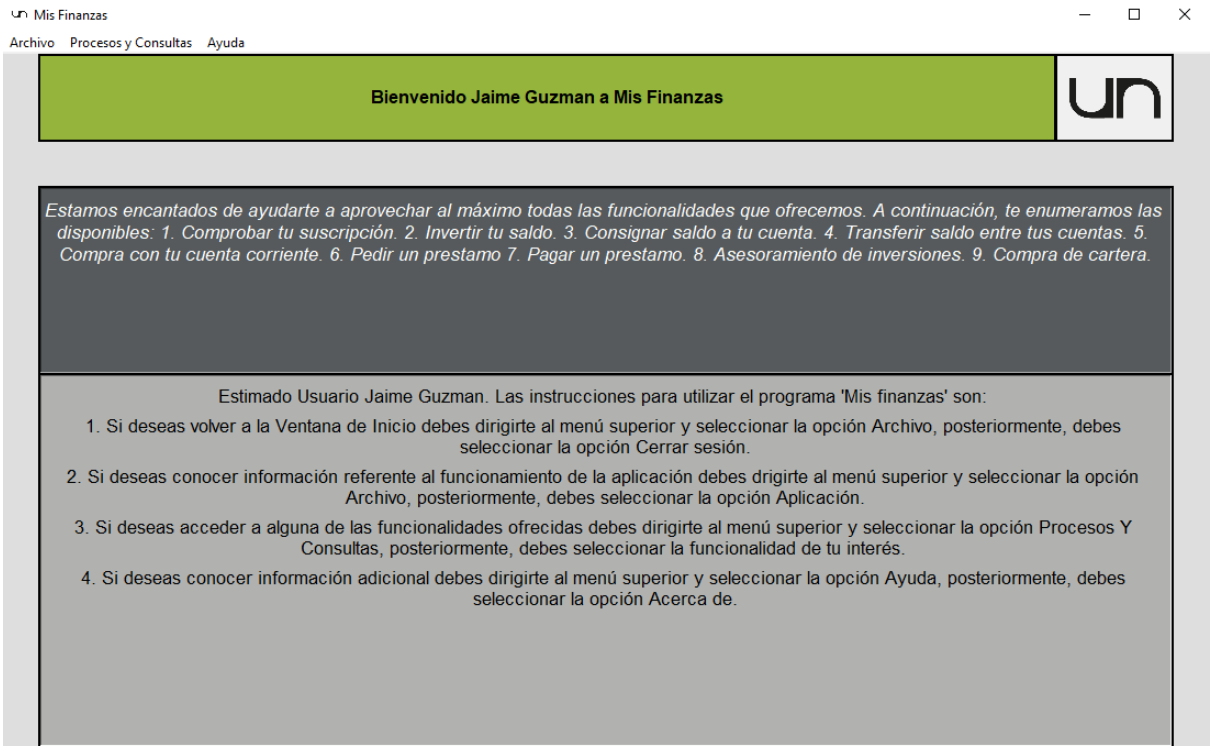


Figura 7. Menú superior

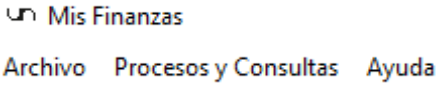


Figura 8. Diálogos de texto

```
# ------ FIELD FRAME PARA DIÁLOGOS DE TEXTO --------
class FieldFrame(tk.Frame):
    def __init__(self, tituloCriterios, criterios, tituloValores, frame,
    **kwargs):
        self.setTituloCriterios(tituloCriterios)
        self.setCriterios(criterios)
        self.setTituloValores(tituloValores)
        self.valores = []
        self.habilitado = []

        for key in kwargs:
            if key == "valores":
                self.setValores(kwargs[key])
            if key == "habilitado":
                self.setHabilitado(kwargs[key])
        self.field_frame = tk.Frame(frame[0], bg="white", borderwidth=1,
        relief="solid")
        if frame[0].winfo_name() == "subframe_main":
            self.field_frame.place(relheight=0.75, relwidth=0.6, rely=0.
            25, relx=0.2)
        else:
            self.field_frame.grid(row=frame[1], column=frame[2],
            columnspan=frame[3], rowspan=frame[4], padx=2, pady=2,
            sticky="NSEW")
            frame[0].columnconfigure(0, weight=1)

        title_style = font.Font(family="Times New Roman", size=13,
        weight="bold")
        criteria_style = font.Font(family="Times New Roman", size=13,
        underline=1)
        entry_style = font.Font(family="Times New Roman", size=13)

        title_criteria = tk.Label(master=self.field_frame, textvariable
        = self.tituloCriterios, width=35, bg="white", fg="black"
```

Figura 9.1 Muestra de resultados de procesos y/o consultas

```
balance_consign_frame.columnconfigure(0, weight=1)
balance_consign_frame.columnconfigure(1, weight=0)
balance_consign_frame.columnconfigure(2, weight=0)
consign_movement = Movimientos.crearMovimiento(selected_account,
selected_balance, Categoria.OTROS, datetime.now())
cls.user.asociarMovimiento(consign_movement)
label_consign_result = tk.Label(balance_consign_frame, text="La
consignación de saldo ha sido exitosa: \n" + str
(consign_movement), font=style_consign_balance, cursor="cross",
border=1, relief="solid", bg="#8C7566", fg="white")
label_consign_result.grid(row=0, column=0, sticky="NSEW", padx=2,
pady=2)
label_movements_result = tk.Label(balance_consign_frame, text=cls.
user.verificarContadorMovimientos(), font=style_consign_balance,
cursor="cross", border=1, relief="solid", bg="#8C7566",
fg="white")
label_movements_result.grid(row=1, column=0, sticky="NSEW",
padx=2, pady=2)
button_result = tk.Button(balance_consign_frame, text="Volver al
menú principal", font=style_consign_balance,
command=back_menu_main, activebackground="gray",
activeforeground="black", cursor="cross", border=1,
relief="solid", bg="#8C7566", fg="white")
button_result.grid(row=2, column=0, sticky="NSEW", padx=2, pady=2)
```

Figura 9.2 Muestra de resultados de procesos y/o consultas

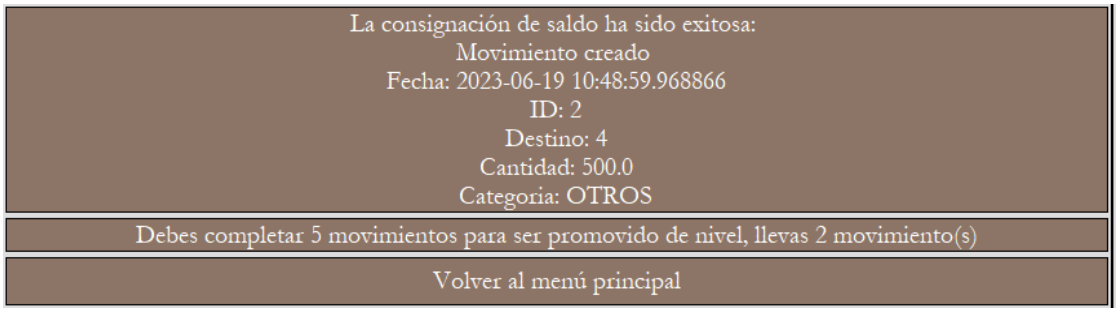


Figura 10. Manejo de errores

```
185            try:
186                possible_user = Usuario.verificarCredenciales(name_email_user, password_user)
187            except usersException.NoUserFoundException:
188                confirmation = messagebox.askretrycancel("Mis finanzas", usersException.NoUserFoundException.show_message())
189                if confirmation:
190                    user_email_entry.delete(0, tk.END)
191                    password_entry.delete(0, tk.END)
192                else:
193                    exit_initial_window()
194            else:
```

```
544                 try:
545                     for inserted_value_entry in inserted_values_entries:
546                         if(inserted_value_entry.winfo_name() == "nombredelbanco"):
547                             selected_bank = inserted_value_entry.get()
548                             c = True
549                             try:
550                                 for bank in Banco.getBancosTotales():
551                                     if(selected_bank == bank.getNombre()):
552                                         selected_bank = bank
553                                         c = False
554                                         break
555                                 if c:
556                                     raise genericException.ValueNotFoundException()
557                             except genericException.ValueNotFoundException:
558                                 confirmation = messagebox.askyesno("Mis finanzas", genericException.ValueNotFoundException.show_message())
559                                 if confirmation:
560                                     create_account_user()
561                                     break
562                                 else:
563                                     back_menu_main()
564                                     break
```

```
645             try:
646                 if(len(cls.user.getCuentasAsociadas()) >= cls.user.getLimiteCuentas()):
647                     raise accountsException.MaxLimitAccountsReached(cls.user)
648             except accountsException.MaxLimitAccountsReached:
649                 confirmation = messagebox.askyesno("Mis finanzas", accountsException.MaxLimitAccountsReached(cls.user).show_message())
650                 if confirmation:
651                     back_menu_main()
652                     comprobar_suscripcion()
653                 else:
654                     back_menu_main
655             else:
```

```
678             try:
679                 asociated_accounts_user = cls.user.mostrarCuentasAhorroAsociadas()
680             except accountsException.NoSavingAccountsAssociatedException:
681                     messagebox.showerror("Mis finanzas", accountsException.NoSavingAccountsAssociatedException(cls.user).show_message())
682                     back_menu_main()
683             else:
```

```
699             try:
700                 accounts_total = Ahorros.getCuentasAhorrosTotales()
701             except accountsException.NoSavingAccountsAssociatedException:
702                     messagebox.showerror("Mis finanzas", accountsException.NoSavingAccountsAssociatedException(cls.user).show_message())
703                     back_menu_main()
704             else:
```

```
720               try:
721                   asociated_banks_user = cls.user.mostrarBancosAsociados()
722               except banksException.NoBanksAssociatedException:
723                   messagebox.showerror("Mis finanzas", banksException.NoBanksAssociatedException(cls.user).show_message())
724                   back_menu_main()
725               else:
```

```
767                     try:
768                         if(selected_suscription is None or selected_suscription == ""):
769                             raise suscriptionException.NoSuscriptionSelectedException
770                         else:
771                             selected_suscription = Suscripcion.__getitem__(selected_suscription)
772                             if(selected_suscription.getLimiteCuentas() < len(cls.user.getCuentasAsociadas())):
773                                 raise suscriptionException.UnderAccountsLimitException
774                     except suscriptionException.NoSuscriptionSelectedException:
775                         confirmation = messagebox.askretrycancel("Mis finanzas", suscriptionException.NoSuscriptionSelectedException.show_message())
776                         if confirmation:
777                             yes_no_confirmation()
778                         else:
779                             back_menu_main()
780                     except suscriptionException.UnderAccountsLimitException:
781                         messagebox.showerror("Mis finanzas", suscriptionException.UnderAccountsLimitException(selected_suscription, cls.user).show_message())
782                         back_menu_main()
783                     else:
```

```
815                     try:
816                         if(selected_bank == "" or selected_bank is None):
817                             raise banksException.NoBankSelectedException
818                     except banksException.NoBankSelectedException:
819                         confirmation = messagebox.askretrycancel("Mis finanzas", banksException.NoBankSelectedException.show_message())
820                         if confirmation:
821                             start_functionality()
822                         else:
823                             back_menu_main()
824                     else:
```

```
                try:
                    if(selected_account == "" or selected_account is None):
                        raise accountsException.NoAccountSelectedException
                    for account in Ahorros.getCuentasAhorrosTotales():
                        if(selected_account == account.getNombre()):
                            selected_account = account
                    c = selected_account.invertirSaldo()
                except accountsException.FailedInvestmentException:
                    messagebox.showwarning("Mis finanzas", accountsException.FailedInvestmentException(cls.user).show_message())
                    back_menu_main()
                except accountsException.NoAccountSelectedException:
                    confirmation = messagebox.askretrycancel("Mis finanzas", accountsException.NoAccountSelectedException.show_message())
                    if confirmation:
                        start_functionality()
                    else:
                        back_menu_main()
                else:
```

```
                    try:
                        if (selected_balance is None or selected_balance == ""):
                            raise genericException.NoValueInsertedException(int)
                        selected_balance = int(selected_balance)
                    except ValueError:
                        confirmation = messagebox.askretrycancel("Mis finanzas", "Debes insertar un número. ¿Deseas intentarlo de nuevo? ")
                        if confirmation:
                            functionality_logic()
                        else:
                            back_menu_main()
                    except genericException.NoValueInsertedException:
                        confirmation = messagebox.askretrycancel("Mis finanzas", genericException.NoValueInsertedException(int).show_message())
                        if confirmation:
                            functionality_logic()
                        else:
                            back_menu_main()
                    else:
```

```
                try:
                    if(selected_account == "" or selected_account is None):
                        raise accountsException.NoAccountSelectedException
                    for account in Ahorros.getCuentasAhorrosTotales():
                        if(selected_account == account.getNombre()):
                            selected_account = account
                except accountsException.NoAccountSelectedException:
                    confirmation = messagebox.askretrycancel("Mis finanzas", accountsException.NoAccountSelectedException.show_message())
                    if confirmation:
                        start_functionality()
                    else:
                        back_menu_main()
                else:
```

```
                        try:
                            if (selected_balance is None or selected_balance == ""):
                                raise genericException.NoValueInsertedException(int)
                            selected_balance = int(selected_balance)
                        except ValueError:
                            confirmation = messagebox.askretrycancel("Mis finanzas", "Debes insertar un número. ¿Deseas intentarlo de nuevo? ")
                            if confirmation:
                                own_account_logic()
                            else:
                                back_menu_main()
                        except genericException.NoValueInsertedException:
                            confirmation = messagebox.askretrycancel("Mis finanzas", genericException.NoValueInsertedException(int).show_message())
                            if confirmation:
                                own_account_logic()
                            else:
                                back_menu_main()
                        else:
```

```
                try:
                    if(selected_account_origin.getSaldo() == 0):
                        raise accountsException.NoBalanceinSavingAccountException(selected_account_origin)
                except accountsException.NoBalanceinSavingAccountException:
                    confirmation = messagebox.askyesno("Mis finanzas", accountsException.NoBalanceinSavingAccountException(selected_account_origin).show_message())
                    if(confirmation):
                        back_menu_main()
                        consignar_saldo()
                    else:
                        own_account_functionality_logic()
                else:
```

```
                try:
                    if(len(cls.user.getCuentasAhorroAsociadas()) < 2):
                        raise accountsException.NotEnoughSavingAccountsException(cls.user)
                except accountsException.NotEnoughSavingAccountsException:
                    confirmation = messagebox.askyesno("Mis finanzas", accountsException.NotEnoughSavingAccountsException(cls.user).show_message())
                    if(confirmation):
                        back_menu_main()
                        create_account_user()
                    else:
                        back_menu_main()
                else:
```

```
                        try:
                            if (selected_balance is None or selected_balance == ""):
                                raise genericException.NoValueInsertedException(int)
                            selected_balance = int(selected_balance)
                        except ValueError:
                            confirmation = messagebox.askretrycancel("Mis finanzas", "Debes insertar un número. ¿Deseas intentarlo de nuevo? ")
                            if confirmation:
                                own_account_logic()
                            else:
                                back_menu_main()
                        except genericException.NoValueInsertedException:
                            confirmation = messagebox.askretrycancel("Mis finanzas", genericException.NoValueInsertedException(int).show_message())
                            if confirmation:
                                own_account_logic()
                            else:
                                back_menu_main()
                        else:
```

```
                    try:
                        if(selected_account_origin.getSaldo() == 0):
                            raise accountsException.NoBalanceinSavingAccountException(selected_account_origin)
                    except accountsException.NoBalanceinSavingAccountException:
                        confirmation = messagebox.askyesno("Mis finanzas", accountsException.NoBalanceinSavingAccountException(selected_account_origin).show_message())
                        if(confirmation):
                            back_menu_main()
                            consignar_saldo()
                        else:
                            another_account_functionality_logic()
                    else:
```

```
                    try:
                        if(len(cls.user.getCuentasAhorroAsociadas()) < 2):
                            raise accountsException.NotEnoughSavingAccountsException(cls.user)
                    except accountsException.NotEnoughSavingAccountsException:
                        confirmation = messagebox.askyesno("Mis finanzas", accountsException.NotEnoughSavingAccountsException(cls.user).show_message())
                        if(confirmation):
                            back_menu_main()
                            create_account_user()
                        else:
                            back_menu_main()
                    else:
```

```
                    try:
                        monto_inversion == int(monto_inversion)
                    except ValueError:
                        messagebox.showerror("Error", "El campo '¿Cuánto dinero deseas invertir?' debe ser un número entero.")
                        return
```

```
                        try:
                            datetime.strptime(entry_fecha.get(), "%d/%m/%Y")
                        except ValueError:
                            messagebox.showerror("Error", "La fecha debe estar en el formato dd/mm/yyyy.")
                            return
```

```
                        try:
                            entry_monto.get() == int(entry_monto.get())
                        except ValueError:
                            messagebox.showerror("Error", "Debes ingresar un número entero.")
                            return
```

```
                    try:
                        if(cantidad is None or cantidad == ""):
                            raise genericException.NoValueInsertedException(int)
                        cantidad = int(cantidad)
                        if cantidad <=0 or cantidad>cuentaSeleccionada.getBanco().getPrestamo():
                            raise genericException.ValuePrestamoException(cantidad,cuentaSeleccionada.getBanco().getPrestamo())
                    except ValueError:
                        error= "continuar"
                        confirmation = messagebox.askretrycancel("Mis finanzas", "Debes insertar un número. ¿Deseas intentarlo de nuevo? (Y/N): ")
                        if confirmation:
                            error ="continuar"
                        else:
                            error = "cancelar"

                    except genericException.ValuePrestamoException:
                        error= "continuar"
                        confirmation =messagebox.askretrycancel("Mis finanzas", genericException.ValuePrestamoException(cantidad,cuentaSeleccionada.getBanco().getPrestamo()).show_message())
                        if confirmation:
                            error ="continuar"
                        else:
                            error = "cancelar"

                    except genericException.NoValueInsertedException:
                        error= "continuar"
                        confirmation = messagebox.askretrycancel("Mis finanzas", genericException.NoValueInsertedException(int).show_message())
                        if confirmation:
                            error ="continuar"

                        else:
                            error = "cancelar"
```

Figura 11. Menú procesos y consultas

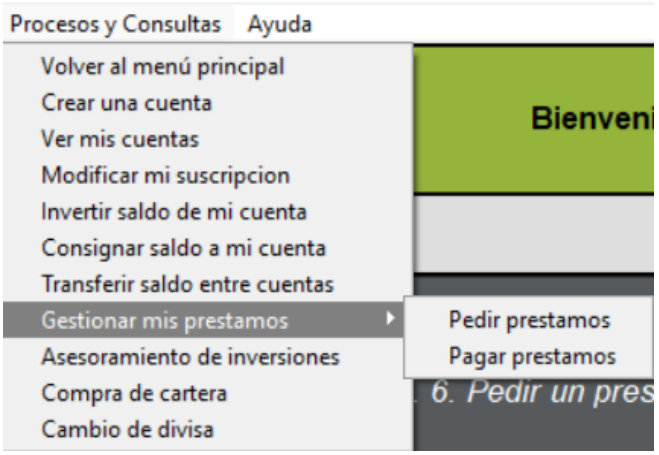


Figura Organización de paquetes y clases

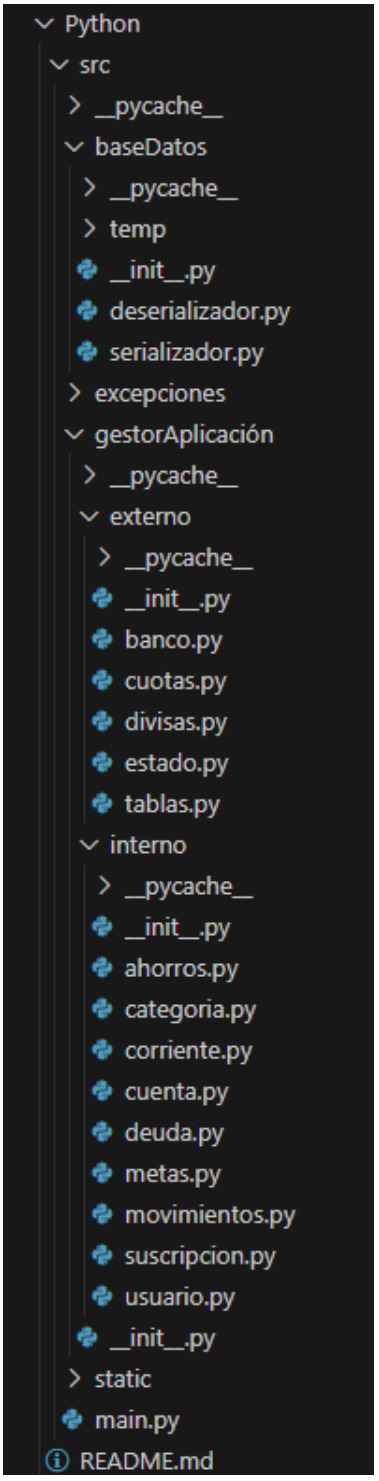


Figura Diagrama UML

Actividad En Equipo POO UNAL.pdf

[Diagrama de la funcionalidad asesor inversiones]

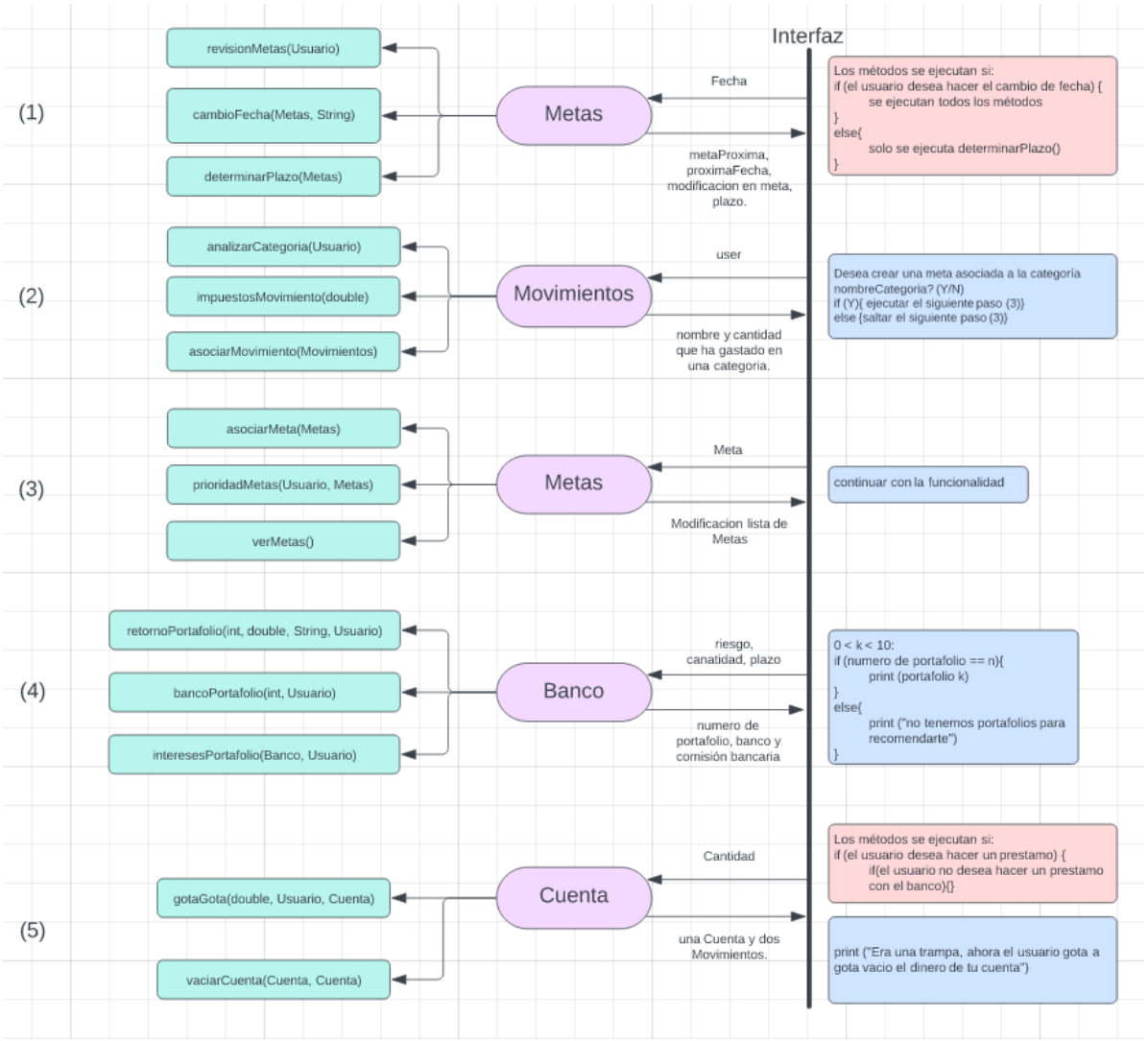


Figura 24. Diagrama de la funcionalidad Préstamos

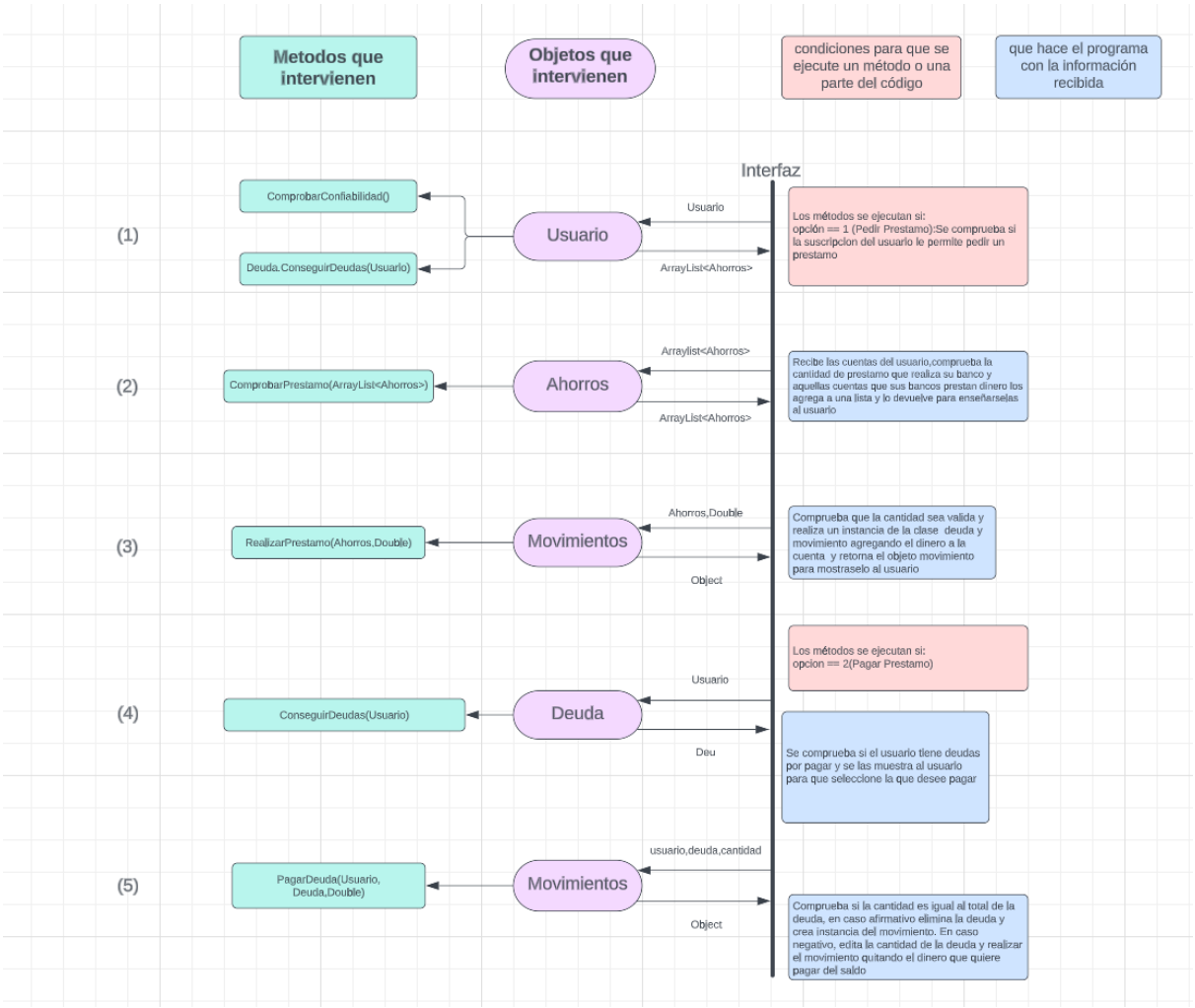
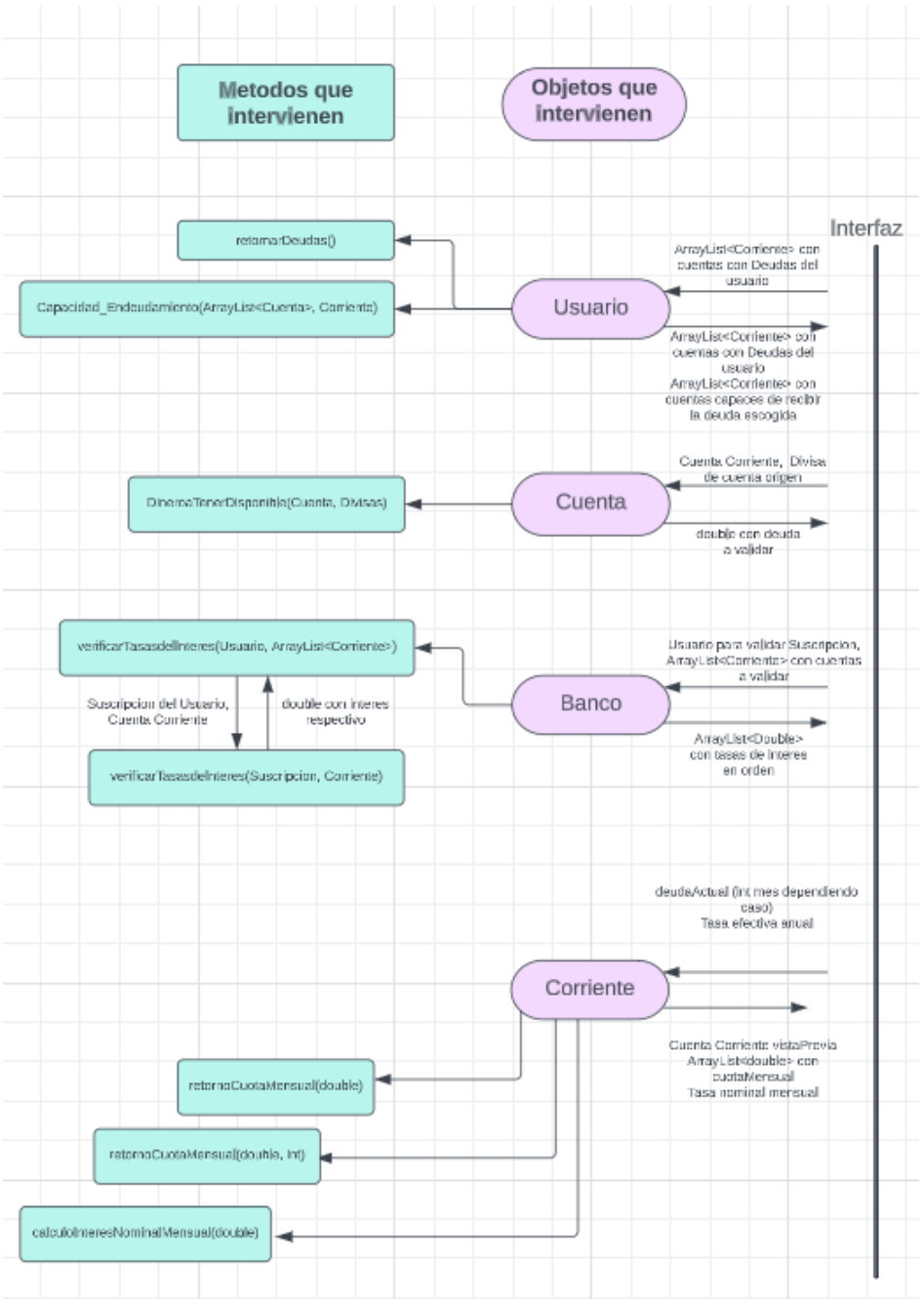


Figura 25. Diagrama de la funcionalidad Compra de Cartera


Metodos que intervienen
Objetos que intervienen
Interfaz
retornarDeudas()
Capacidad_Endeudamiento(ArrayList<Cuenta>, Corriente)
Usuario
ArrayList<Corriente> con cuentas con Deudas del usuario
ArrayList<Corriente> con cuentas con Deudas del usuario
ArrayList<Corriente> con cuentas capaces de recibir la deuda escogida
DineroTenerDisponible(Cuenta, Divisas)
Cuenta
Cuenta Corriente, Divisa de cuenta origen
double con deuda a validar
verificarTasasdeInteres(Usuario, ArrayList<Corriente>)
Suscripcion del Usuario, Cuenta Corriente
double con interes respectivo
verificarTasasdeInteres(Suscripcion, Corriente)
Banco
Usuario para validar Suscripcion, ArrayList<Corriente> con cuentas a validar
ArrayList<Double> con tasas de interes en orden
deudaActual (int mes dependiendo caso)
Tasa efectiva anual
Corriente
Cuenta Corriente vistaPrevia
ArrayList<double> con cuotaMensual
Tasa nominal mensual
retornoCuotaMensual(double)
retornoCuotaMensual(double, int)
calculoInteresNominalMensual(double)