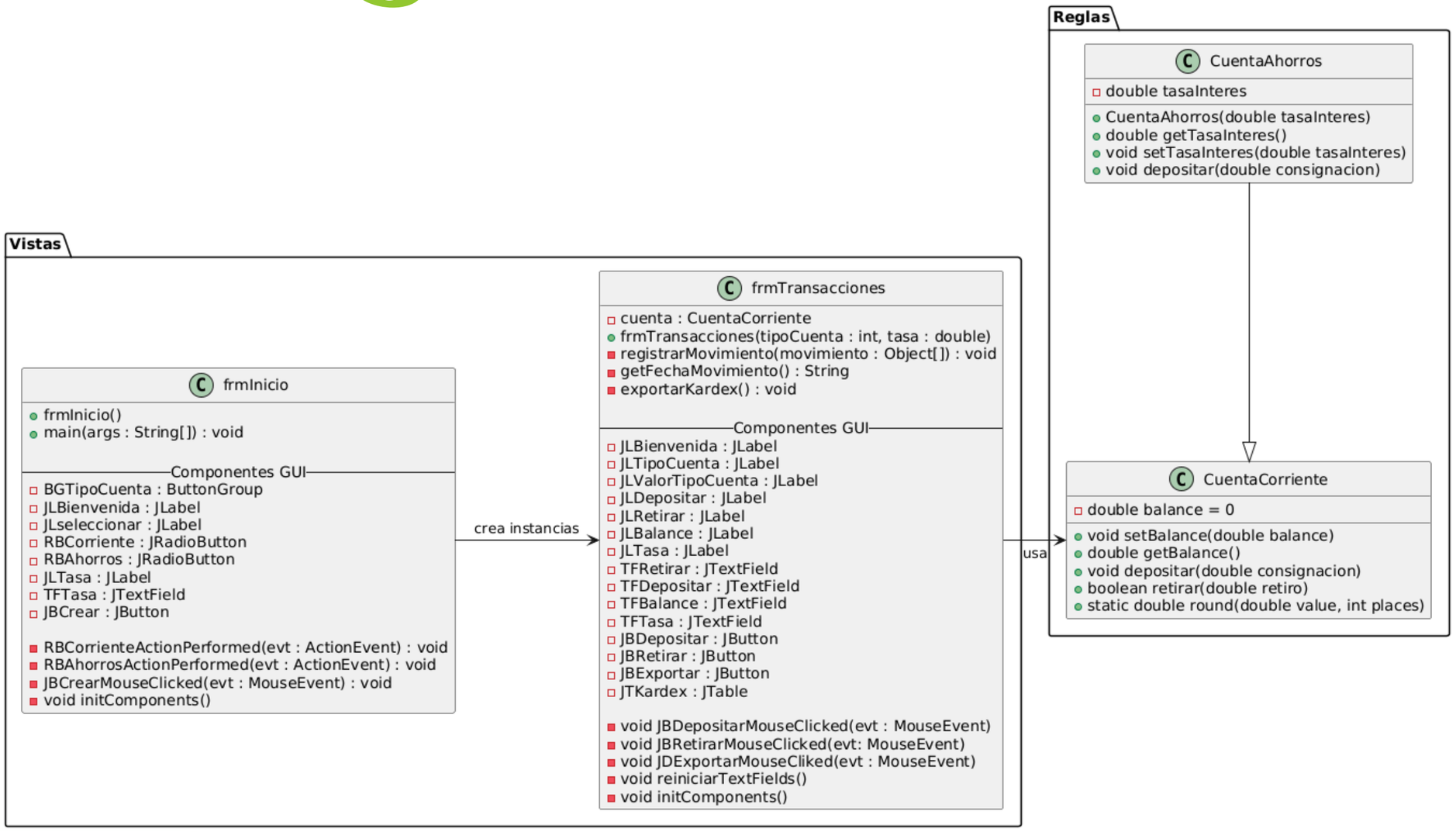
Ejercicio De Reingeniería

Integrantes:
Andres Felipe Muñoz Aguilar - 2210087
Juan Diego Roa Porras - 2210086
Miguel Fernando Pimiento Escobar - 2210054

Universidad Industrial de Santander

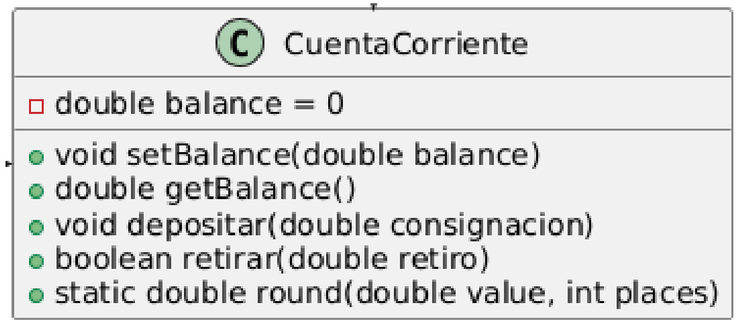
76 Aniversario UIS 1948 - 2024

Diagrama De Clases



Descripción De Metodos

```
public boolean retirar(double retiro) {
    if(balance >= retiro){
     //balance = Math.round(balance - retiro);
        balance = round(balance - retiro, 5);
        return true;
    }
    else{
        JOptionPane.showMessageDialog(null,
                "¡No hay suficiente saldo en tu cuenta para retirar $" + String.valueOf(retiro) +"!" ,
                "Saldo insuficiente",
                JOptionPane.ERROR_MESSAGE);
        return false;
    }
}
```



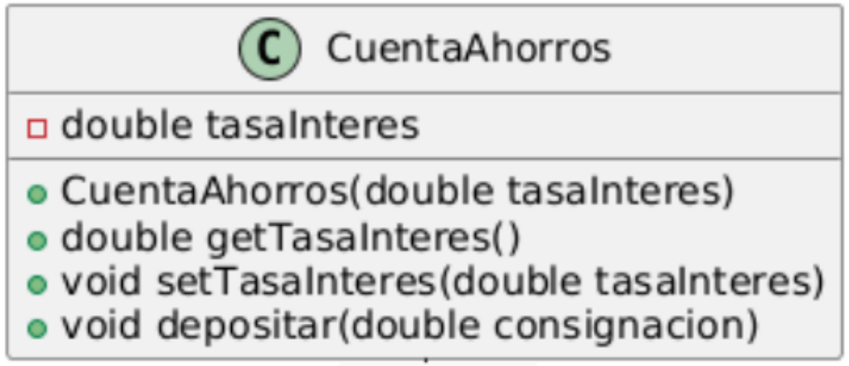
En la función ***retirar*** de la clase ***“CuentaCorriente”***, se implementa un mecanismo que permite verificar si el balance actual de una cuenta es suficiente para realizar una operación de retiro. El proceso consiste en comparar el monto a retirar con el saldo disponible en la cuenta. Si el saldo es mayor o igual al monto solicitado, el retiro se realiza, actualizando el balance mediante una función de redondeo a cinco decimales para mayor precisión. Posteriormente, se retorna el valor ***“true”***, lo que indica que la operación fue exitosa. En caso contrario, si el saldo es insuficiente, se despliega un mensaje de error al usuario utilizando un cuadro de diálogo (“JOptionPane”), notificando la falta de fondos, y la función retorna el valor ***“false”***.

El permitir que la función retorne un valor booleano, facilita que se compruebe el estado de la transacción desde la clase ***“frmTransacciones.java”*** y se haga un manejo de errores, con la funcionalidad de ***“JOptionPane”***, de ser necesario

Descripción De Metodos

```java
@Override
public void depositar(double consignacion) {
    //double newBalance = this.getBalance() + (consignacion*(1+(tasaInteres/100))); // Da error: Proba
    double newBalance = this.getBalance() + consignacion + (consignacion * (tasaInteres / 100));
    this.setBalance(newBalance);
}
```

Este método es una sobrescritura del método original ***"depositar"*** que se encuentra en la clase ***"CuentaCorriente"***. En la clase base, el comportamiento del método simplemente consiste en sumar el valor del depósito al balance actual de la cuenta. Sin embargo, en ***esta implementación*** sobrescrita, en la clase ***“CuentaAhorros”***, no solo se añade el valor de la consignación al balance, sino que también se le ***agrega una tasa de interés***, lo cual incrementa el dinero del usuario cada vez que se realiza un depósito. Este diseño cumple con los requerimientos de la aplicación, estipulados por el cliente: Urbano.



Descripción De Metodos

```java
private void JBCrearMouseClicked(java.awt.event.MouseEvent evt) {
    try {
        if (TFTasa.getText().matches("\\d+(\\.\\d+)?\\s*%?")) {
            //Obtenemos la tasa
            String tasaPorcentaje = TFTasa.getText().replace("%", "").trim();
            Double tasa = Double.valueOf(tasaPorcentaje);

            //Cerramos la pestaña actual
            this.dispose();

            //En base a los datos creamos la ventana del Kardex con su respectiva cuenta:
            if (RBCorriente.isSelected()) {
                new frmTransacciones(0, tasa).setVisible(true); //0 = Cuenta Corriente
            } else {
                new frmTransacciones(1, tasa).setVisible(true); //1 = Cuenta Ahorros
            }
```

Este método, perteneciente a la clase "frmInicio.java", que forma parte de la interfaz encargada de permitir la selección del tipo de cuenta con una tasa de interés personalizable, cumple una función específica en la validación y procesamiento de los datos ingresados. ***En primer lugar, verifica*** si el usuario ha ingresado un ***número válido*** en el campo destinado a la ***tasa de interés*** de una cuenta de ahorro. Esta validación se realiza mediante una ***regex*** que asegura que el formato del valor ingresado cumpla con estos criterios: Un grupo de ***dígitos*** inicial, seguido de un ***punto decimal***, otro grupo de ***dígitos*** cualesquiera, ***posibles espacios***, y finalmente, un ***símbolo de porcentaje opcional***.

Si la entrada cumple con los ***requisitos*** de formato, el sistema procede a ***cerrar la ventana*** de selección de cuenta. A continuación, se evalúa si el usuario ha optado por una cuenta de ahorros o una cuenta corriente, para lo cual ***se generan diferentes tablas (Kardex)*** según el tipo de cuenta. En el caso de la cuenta de ahorros, se toma en cuenta el valor de la tasa de interés, mientras que para la cuenta corriente, dicho valor no es considerado.

Por otro lado, si se detecta que el campo de la tasa de interés contiene un valor inválido, el método activa un mensaje de error utilizando un ***"JOptionPane"***, informando al usuario sobre la necesidad de ingresar un valor correcto antes de continuar con el proceso. Este mecanismo garantiza una interacción fluida y validada entre el usuario y la interfaz.

Descripción De Metodos

```
private void JBDepositarMouseClicked(java.awt.event.MouseEvent evt) {
    try {
        if (TFDepositar.getText().matches("\\d+(\\.\\d+)?") && Double.parseDouble(TFDepositar.getText()) > 0) {
            cuenta.depositar(Double.parseDouble(TFDepositar.getText()));
            //Generamos la estructura del movimiento:
            Object[] movimiento = {"Depositar", getFechaMovimiento(), Double.valueOf(TFDepositar.getText()),
                cuenta.getBalance(), TFTasa.getText()};
            registrarMovimiento(movimiento);
        } else {
            JOptionPane.showMessageDialog(null,
                    "Error: El valor a depositar no es un número válido. Por favor, ingrese un número valido.",
                    "Deposito Invalido",
                    JOptionPane.ERROR_MESSAGE);
        }
```

Este método se utiliza para manejar la acción de depósito realizada por el usuario al hacer clic en el botón correspondiente. Primero, verifica si el valor ingresado en el campo de texto de depósito es un número válido, utilizando una expresión regular que admite dígitos, un punto decimal, posibles espacios y un símbolo de porcentaje opcional. Si el valor es válido y mayor que cero, se realiza el depósito a la cuenta, y se registra el movimiento en el sistema con detalles como el tipo de transacción, la fecha, el monto depositado, el balance actualizado y la tasa de interés. En caso contrario, o si ocurre una excepción de formato, se muestra un mensaje de error para notificar al usuario que el valor ingresado no es válido.

Descripción De Metodos

```
private void JBRetirarMouseClicked(java.awt.event.MouseEvent evt) {
    try {
        if (TFRetirar.getText().matches("\\d+(\\.\\d+)?") && Double.parseDouble(TFRetirar.getText()) > 0) {
            if (cuenta.retirar(Double.parseDouble(TFRetirar.getText())) == true) {
                //Generamos la estructura del movimiento:
                Object[] movimiento = {"Retirar", getFechaMovimiento(), Double.valueOf(TFRetirar.getText()),
                    cuenta.getBalance(), TFTasa.getText()};
                registrarMovimiento(movimiento);
            }
        } else {
            JOptionPane.showMessageDialog(null,
                    "Error: El valor a retirar no es un número válido. Por favor, ingrese un número valido.",
                    "Retiro Invalido",
                    JOptionPane.ERROR_MESSAGE);
        }
```

Este método se emplea para manejar la ***acción de retiro***. Primero, se valida si el valor ingresado es un ***número válido*** mediante una ***expresión regular*** y se comprueba que sea ***mayor que cero***. Si la validación es exitosa y el saldo es suficiente, se realiza el retiro y ***se registra el movimiento***, haciendo uso de un ***arreglo de objetos*** que se ***añade como una fila al kardex*** que se muestra en el ***“frmTransacciones”***, incluyendo detalles como el tipo de transacción, la fecha, el monto retirado, el balance actualizado y la tasa de interés. Esto se ampliará más en la diapositiva siguiente. Si el valor no es válido o si ocurre una excepción, se muestra un mensaje de error notificando al usuario sobre el ingreso incorrecto.

Descripción De Metodos

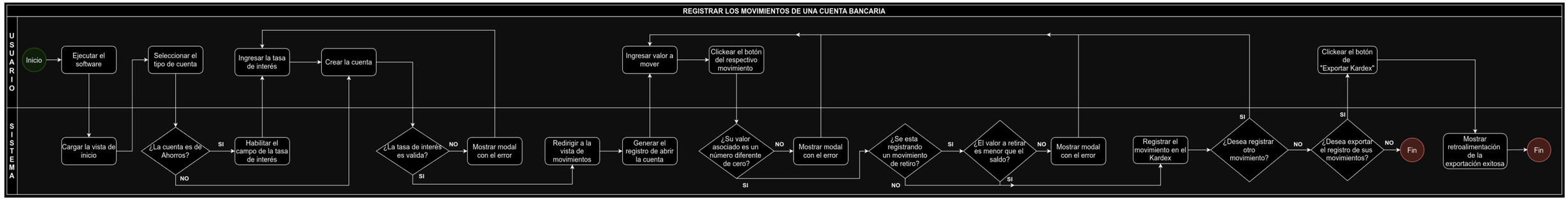
```
private void registrarMovimiento(Object[] movimiento) {
    //Obtenemos el modelo de la tabla:
    DefaultTableModel model = (DefaultTableModel) JTKardex.getModel();
    //Añadimos el movimiento a la tabla
    model.addRow(movimiento);

    //Actualizamos el Balance y los valores a Depositar o Retirar.
    TFBalance.setText(String.valueOf(cuenta.getBalance()));
    reiniciarTextFields();
}

private void reiniciarTextFields() {
    TFRetirar.setText("0");
    TFDepositar.setText("0");
}
```

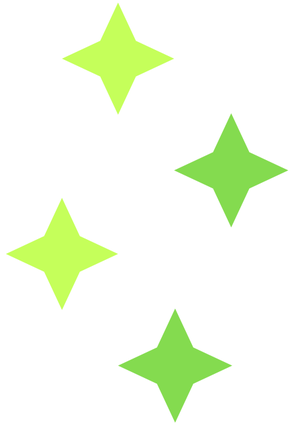
Este método se utiliza para registrar y visualizar un movimiento financiero en la gestión de cuentas bancarias. Al ejecutarse, obtiene el modelo de la tabla de movimientos, añade la nueva transacción, con la función ***“model.addRow()”***; haciendo uso del ***“setText”*** actualiza el balance de la cuenta y restablece los campos de entrada, usando las propiedades que los ***TextField*** proporcionan para la administración del texto dentro de ellos. De esta forma, facilita la actualización en tiempo real de los movimientos y el saldo de la cuenta en la interfaz gráfica.

Diagrama De Actividades

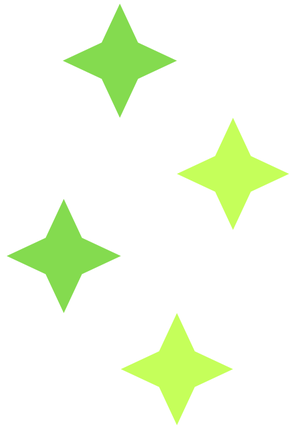


ENLACE AL DIAGRAMA (MEJOR CALIDAD)

Prueba En Vivo



Ubuntu 22.04 LTS
"Jammy Jellyfish"



Codigo Fuente

GitHub

Repositorio Del Ejercicio

Video Explicativo

Video Explicativo

Muchas Gracias