Manual Técnico

Introducción

Este documento describe el funcionamiento interno del sistema BancoMVC, desarrollado en Java siguiendo el patrón Modelo-Vista-Controlador (MVC). Está dirigido a desarrolladores y técnicos que necesiten comprender la estructura del código y sus funcionalidades principales.

BancoMVC está diseñado para gestionar operaciones bancarias de manera eficiente y estructurada. Su arquitectura modular permite una fácil integración con bases de datos y otros sistemas, facilitando la escalabilidad y mantenimiento del software. Este manual proporciona una visión detallada del código, explicando sus componentes y la interacción entre ellos.

Requerimientos del Sistema

- Lenguaje de programación: Java
- Entorno de desarrollo: NetBeans o cualquier IDE compatible con Java
- JDK recomendado: Java SE 8 o superior
- Base de datos: (Si aplica, especificar)
- Dependencias externas: (Si aplica, especificar)

Explicación Detallada de Funciones/Métodos

Paquete Modelo

- **Banco.java**: Clase que representa el banco y sus operaciones principales.
- **Cuenta.java**: Modelo de cuenta bancaria con atributos como número de cuenta, saldo, titular.
- **Usuario.java**: Representa un usuario del sistema, con credenciales y datos personales.
- **Transaccion.java**: Maneja las operaciones como depósitos y retiros.

Paquete Controlador

- **Controlador_banco.java**: Administra la lógica principal del banco.
- **Controlador_login.java**: Gestiona la autenticación de usuarios.
- **Controlador_reportes.java**: Genera informes sobre transacciones y cuentas.

Paquete Vista

- **Login_vista.java**: Interfaz gráfica para el inicio de sesión.
- **Crear_cuenta_usuarios.java**: Permite la creación de cuentas nuevas.
- **Retiros_usuarios.java** y **Depositos_usuarios.java**: Manejan las interfaces de retiro y depósito.
- **Historial_transacciones.java**: Muestra las transacciones realizadas.

1. Arquitectura General

Descripción del patrón MVC (Modelo-Vista-Controlador):

- **Modelo (Model):** En este componente se gestiona la lógica de negocio y el acceso a datos. Las clases Banco, Cuenta, Usuario, y Transaccion son ejemplos de clases del modelo. Estas clases contienen las reglas y procesos necesarios para las operaciones bancarias.
- **Vista (View):** Se encarga de la interfaz gráfica del usuario. Las clases Login_vista, Crear_cuenta_usuarios, Retiros_usuarios, Depositos_usuarios, y Historial_transacciones están relacionadas con la presentación de datos al usuario y la interacción con él.
- **Controlador (Controller):** Actúa como un intermediario entre el modelo y la vista. Las clases Controlador_banco, Controlador_login, y Controlador_reportes son responsables de manejar la lógica de los eventos del usuario y la actualización de la vista según los resultados del modelo.

Flujo de trabajo general: El usuario interactúa con la vista (por ejemplo, realiza un inicio de sesión en Login_vista.java). El controlador correspondiente, como Controlador_login.java, procesa la solicitud, verifica las credenciales del usuario a través del modelo (Usuario.java), y actualiza la vista en consecuencia.

2. Detalles de Implementación

Paquete Modelo**:**

- **Banco.java:** Esta clase es el corazón de las operaciones bancarias. Podría contener métodos como:

- agregarCuenta(Cuenta cuenta): Añade una nueva cuenta al banco.
- realizarTransaccion(Transaccion transaccion): Ejecuta una transacción (por ejemplo, un depósito o retiro).
- obtenerCuenta(String numeroCuenta): Retorna una cuenta específica.

Cuenta.java: Define las propiedades y métodos asociados a una cuenta bancaria, como:

- numeroCuenta: Atributo único para cada cuenta.
- saldo: Monto actual en la cuenta.
- Métodos para realizar depósitos, retiros y consultar el saldo.

Usuario.java: Gestiona la información del usuario, con métodos como:

- verificarCredenciales(String usuario, String contrasena): Verifica las credenciales del usuario.
- registrarUsuario(String nombre, String correo, String contrasena): Registra un nuevo usuario.

Transaccion.java: Encapsula las transacciones, con métodos como:

- `realizarDeposito(double monto)`: Realiza un depósito en la cuenta.
- `realizarRetiro(double monto)`: Realiza un retiro, si el saldo lo permite.

Paquete Controlador:

Controlador_banco.java: Este controlador orquesta las operaciones bancarias entre las vistas y el modelo, como:

- procesarDeposito(Cuenta cuenta, double monto): Llama al modelo para realizar un depósito.
- procesarRetiro(Cuenta cuenta, double monto): Llama al modelo para realizar un retiro.

Controlador_login.java: Encargado de manejar la autenticación de los usuarios. Puede contener métodos como:

- autenticarUsuario(String usuario, String contrasena): Verifica las credenciales del usuario y redirige a la vista correspondiente.

Controlador_reportes.java: Maneja la generación de informes de transacciones y cuentas. Por ejemplo:

- generarReporteTransacciones(Cuenta cuenta): Genera un informe detallado de todas las transacciones de una cuenta.

Paquete Vista:

- **Login_vista.java:** Interfaz gráfica de inicio de sesión. Se comunica con Controlador_login.java para autenticar al usuario.
- **Crear_cuenta_usuarios.java:** Permite a los usuarios crear nuevas cuentas. Se conecta con Controlador_banco.java para agregar la cuenta.
- **Retiros_usuarios.java y Depositos_usuarios.java:** Permiten realizar transacciones de retiro o depósito y envían las acciones al controlador para ejecutar la operación correspondiente.
- **Historial_transacciones.java:** Muestra las transacciones realizadas, utilizando los datos proporcionados por el controlador para generar una vista actualizada del historial.

3. Interacciones y Flujo de Datos

El flujo de datos sigue este patrón:

- El usuario interactúa con la vista, por ejemplo, realiza un inicio de sesión.
- La vista envía los datos al controlador, como el nombre de usuario y la contraseña.
- El controlador procesa la solicitud llamando a métodos en el modelo, como verificarCredenciales.
- El modelo interactúa con la base de datos o realiza operaciones lógicas, como validar las credenciales del usuario.
- El controlador actualiza la vista con los resultados (por ejemplo, redirigiendo a una pantalla principal si la autenticación es exitosa).

4. Seguridad

Si el sistema maneja información sensible como contraseñas o detalles bancarios, es fundamental incluir prácticas de seguridad, tales como:

- Encriptación de contraseñas: Asegúrate de almacenar contraseñas de forma segura usando técnicas de encriptación como BCrypt.
- Validación de entradas: Asegúrate de que todas las entradas del usuario sean validadas y sanitizadas para evitar inyecciones SQL y otros ataques comunes.

5. Pruebas

El sistema debería ser probado para garantizar que las operaciones bancarias se realizan correctamente. Las pruebas unitarias para cada clase del modelo y los controladores son esenciales. Puedes utilizar bibliotecas como JUnit para crear pruebas automatizadas que verifiquen que los métodos de las clases Banco, Cuenta, Usuario, y Transaccion funcionan según lo esperado.

6. Escalabilidad y Mantenimiento

El diseño modular de BancoMVC facilita el mantenimiento y escalabilidad del sistema. Si se desea agregar nuevas funcionalidades (por ejemplo, transferencias entre cuentas o gestión de préstamos), se pueden agregar nuevas clases en el modelo, actualizar los controladores correspondientes, y extender las vistas sin afectar el funcionamiento general del sistema.

Conclusión

BancoMVC es un sistema robusto y flexible, construido sobre el patrón MVC, lo que asegura una estructura clara y escalable. El código modular y bien organizado permite que desarrolladores y técnicos puedan entender y extender fácilmente el sistema según sea necesario. Con un enfoque en seguridad, pruebas y mantenimiento, BancoMVC está preparado para manejar operaciones bancarias de manera eficiente.

Si deseas más detalles sobre alguna de las secciones mencionadas o ejemplos de código específicos, no dudes en pedírmelo.