UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE OCCIDENTE
DIVISIÓN DE CIENCIAS DE LA INGENIERÍA
INTRODUCCIÓN A LA PROGRAMACION 2

MANUAL TECNICO

DIEGO ALESSANDRO GONZALEZ BRAVO / 202230939

Herramientas de desarrollo.
Las herramientas (IDES, lenguajes de programación) utilizadas para el desarrollo de la aplicacion Manejo de tarjetas son:

- JAVA
    - Version openjdk version "21.0.4"
- Apache NetBeans
    - Version 22
- Sistema operativo de desarrollo
    - Ubuntu 24.04 LTS

Conceptos aplicados:
Al desarrollar una aplicación de movimiento de tarjetas en Java que interactúa con una base de datos, lee archivos .txt y genera archivos .html, es fundamental comprender varios conceptos clave para asegurar una implementación eficiente y robusta. Aquí te presento una descripción de los conceptos involucrados:

1. Conexión a MySQL desde Java
    - Driver JDBC de MySQL: Para conectar Java con MySQL, necesitas el controlador JDBC de MySQL (mysql-connector-java). Este driver permite que la aplicación Java se comunique con la base de datos MySQL utilizando el protocolo adecuado.

    - Establecer la Conexión: Utilizas DriverManager.getConnection con la URL de conexión adecuada (jdbc:mysql://host:port/database) junto con las credenciales de usuario y contraseña para establecer una conexión con la base de datos MySQL.
2. Consultas SQL en MySQL
    - Consultas Parametrizadas: Es importante usar consultas parametrizadas con PreparedStatement para evitar inyecciones SQL. Por ejemplo:
      String query = "SELECT * FROM movimientos WHERE tarjeta_id = ?";
      PreparedStatement pstmt = conn.prepareStatement(query);
      pstmt.setInt(1, tarjetaId);
      ResultSet rs = pstmt.executeQuery();

    - Manejo de Fechas y Tiempos: MySQL tiene tipos de datos específicos para manejar fechas y horas (DATE, TIME, DATETIME, TIMESTAMP). Es importante formatear y manejar adecuadamente estos tipos desde Java.
3. Transacciones en MySQL
    - Gestión de Transacciones: En operaciones críticas, como los movimientos de tarjetas, es fundamental gestionar las transacciones. Puedes hacerlo manualmente con conn.setAutoCommit(false), seguido de conn.commit() o conn.rollback() dependiendo del resultado de las operaciones.
    - Bloqueo de Filas: Para evitar condiciones de carrera y asegurar la consistencia, es posible que necesites utilizar bloqueos (LOCK IN SHARE MODE, FOR UPDATE) en las consultas SQL cuando trabajes con datos críticos como el saldo de una tarjeta.
4. Optimización de Consultas
    - Índices: MySQL permite crear índices para mejorar el rendimiento de las consultas, especialmente en tablas con un gran volumen de datos, como una

tabla de movimientos de tarjetas. Asegúrate de tener índices en columnas que se utilizan frecuentemente en WHERE o JOIN.
    - Explain Plan: Utiliza la sentencia EXPLAIN para entender cómo MySQL está ejecutando tus consultas y optimizarlas si es necesario.
5. Seguridad en MySQL
    - Usuarios y Permisos: Configura usuarios específicos en MySQL con permisos mínimos necesarios para la aplicación. Esto reduce el riesgo en caso de una vulnerabilidad.
    - Cifrado de Datos: Considera cifrar datos sensibles, como números de tarjetas, tanto en la base de datos como en las transmisiones entre la aplicación y MySQL.
6. Exportación e Importación de Datos
    - Carga de Archivos .txt: Podrías usar la funcionalidad de MySQL para importar archivos .txt directamente a tablas con el comando LOAD DATA INFILE, pero también puedes manejar esto desde Java si necesitas procesar los datos antes de insertarlos.
    - Exportación de Resultados: MySQL permite exportar resultados de consultas directamente a archivos .csv o .txt, que luego pueden ser procesados o transformados en .html desde Java.

APLICANDO LOS CONCEPTOS

Conexion base de datos

En el siguiente ejemplo podemos ver como realizar una conexion a la base de datos mediante java, utilizando la librería Java.SQL en donde necesitamos el url de mysql, el usuario y la contraseña para ingresar a la base de datos.

```
private static final String URL_MYSQL = "jdbc:mysql://localhost:3306/practica1";
private static final String USER = "root";
private static final String PASSWORD = "26359";
public Connection connection;
public boolean solicitudExistente;
public boolean solicitudEnviada;

public conexionDB() {
    conectar();
}

private void conectar() {
    // Creamos la conexion con la base de datos
    try {
        connection = DriverManager.getConnection(URL_MYSQL, USER, PASSWORD);
    } catch (SQLException e) {
        System.out.println("Error al connectar a la DB");
    }
}
```

Consultas

En el siguiente código se muestra como se puede realizar una consulta en la base de datos aplicando los conocimientos, se almacena la consulta en un String ya que es una cadena de caracteres y luego con la conexion podemos realizar el executeQuery lo cual realiza la consulta deseada.

```java
public boolean verificarSolicitud(autoTarjetas tarjeta) {
    String consulta = "SELECT 1 FROM autorizacion WHERE solicitud = ? LIMIT 1";

    try (PreparedStatement statementConsulta = connection.prepareStatement(consulta)) {
        int numeroSolicitud = tarjeta.getNumeroSolicitud();

        // Establecer el número de solicitud en la consulta
        statementConsulta.setInt(1, numeroSolicitud);

        // Ejecutar la consulta
        ResultSet resultSet = statementConsulta.executeQuery();

        // Si se encuentra al menos un resultado, la solicitud existe
        if (resultSet.next()) {
            // Actualizar el estado de la tarjeta
            tarjeta.setAprovado(false);
            solicitudEnviada = false;
            return false; // La solicitud ya existe
        }
    } catch (SQLException e) {
        System.out.println("Error al verificar la existencia de la solicitud");
    }

    return true; // Retornar true si no se encontró la solicitud
}
```

Verificación de datos

A la hora de realizar movimientos en la tarjeta necesitamos varias verificaciones tanto en el frontend y en el backend para que no se pierda información o no se ingrese información no deseada

Ejemplo de verificaciones en frontend:

```java
private void btnSolicitarActionPerformed(java.awt.event.ActionEvent evt) {
    if (txtSolicitud.getText().length() == 0) {
        JOptionPane.showMessageDialog(null, "NO SE INGRESO UNA SOLICITUD");
        return;
    } else if (txtNombre.getText().length() == 0) {
        JOptionPane.showMessageDialog(null, "NO SE INGRESO UN NOMBRE");
        return;
    } else if (txtSalario.getText().length() == 0) {
        JOptionPane.showMessageDialog(null, "NO SE INGRESO UN SALARIO");
        return;
    } else if (txtDireccion.getText().length() == 0) {
        JOptionPane.showMessageDialog(null, "NO SE INGRESO UNA DIRECCION");
        return;
    } else if (txtFecha.getText().length() == 0) {
        JOptionPane.showMessageDialog(null, "NO SE INGRESO UNA FECHA");
        return;
    }

    try {
        // Intentar convertir el texto a un número entero
        solicitudId = Integer.parseInt(txtSolicitud.getText());

    } catch (NumberFormatException ex) {
        // Si la conversión falla, muestra un mensaje de error
        JOptionPane.showMessageDialog(null, "POR FAVOR INGRESE UNA SOLICITUD VALIDA", "ERROR", JOptionPane.ERROR_MESSAGE);
        return;
    }

    try {
        // Intentar convertir el texto a un número entero
        salario = Double.parseDouble(txtSalario.getText());

        // Si la conversión es exitosa, puedes continuar con el procesamiento
    } catch (NumberFormatException ex) {
        // Si la conversión falla, muestra un mensaje de error
        JOptionPane.showMessageDialog(null, "POR FAVOR INGRESE UN SALARIO VALIDO", "ERROR", JOptionPane.ERROR_MESSAGE);
        return;
    }

    if (txtNombre.getText().length() >= 100) {
        JOptionPane.showMessageDialog(null, "NOMBRE DEMASIADO LARGO", "ERROR", JOptionPane.ERROR_MESSAGE);
        return;
    }
    if (txtDireccion.getText().length() >= 150) {
        JOptionPane.showMessageDialog(null, "DIRECCION DEMASIADA LARGA", "ERROR", JOptionPane.ERROR_MESSAGE);
        return;
    }

    solicitud = new crearSolicitud(solicitudId, txtNombre.getText(), salario, txtDireccion.getText(), txtFecha.getText(),
            cbTarjetas.getSelectedItem().toString());
    crearSolicitudDB solicitudDB = new crearSolicitudDB();
    solicitudDB.crearSolicitud(solicitud, this);

}
```

En este caso el frontend verificamos si nuestras casillas de texto contienen algo, si no contienen nada verificar el error.
Ejemplo de verificaciones en backen:

```
public void movimientoValido(String tipoCargo, double monto, movimientos mensajes) {
    // Verifica que tipo de movimiento realiza y si se puede efectuar
    if (tipoCargo.equals("ABONO")) {
        if (tarjeta.getSaldo() == 0) {
            mensajes.mensajeAlDia();
        } else {
            abono = tarjeta.getSaldo() - monto;
            tarjeta.setSaldo(abono);
            mensajes.mensajeAbono(tarjeta.getSaldo());
        }
    } else if (tipoCargo.equals("CARGO")) {
        if (monto > tarjeta.getLimite()) {
            mensajes.mensajeCargoMax();
        } else {
            cargo = tarjeta.getSaldo() + monto;
            if (cargo > tarjeta.getLimite()) {
                mensajes.mensajeLimite();
            } else {
                tarjeta.setSaldo(cargo);
                mensajes.mensajeCargo(tarjeta.getSaldo(), tarjeta.getLimite());
            }

        }
    }
}
```

En este caso verificamos el objeto de movimientos, en el cual a la hora de realizar un movimiento si es un cargo o un abono, si es cargo que el monto del cargo no sobrepase el límite de la tarjeta y que el saldo de la tarjeta no sobrepase el límite de la tarjeta

Diagramas
Diagrama de entidad relación E/R

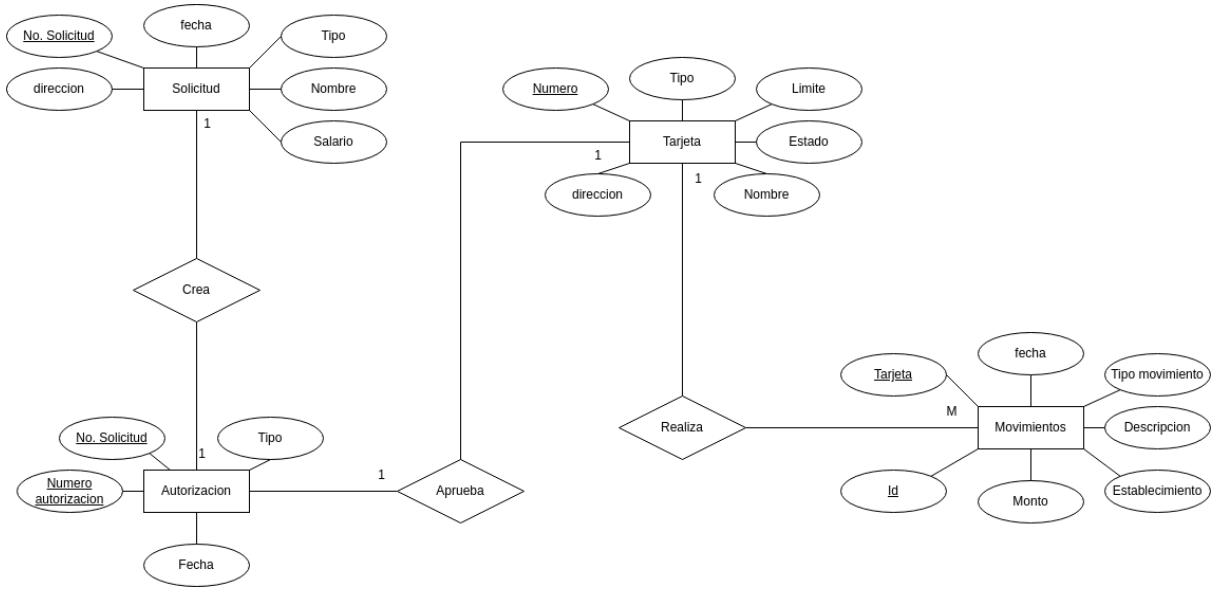

Diagrama de tablas

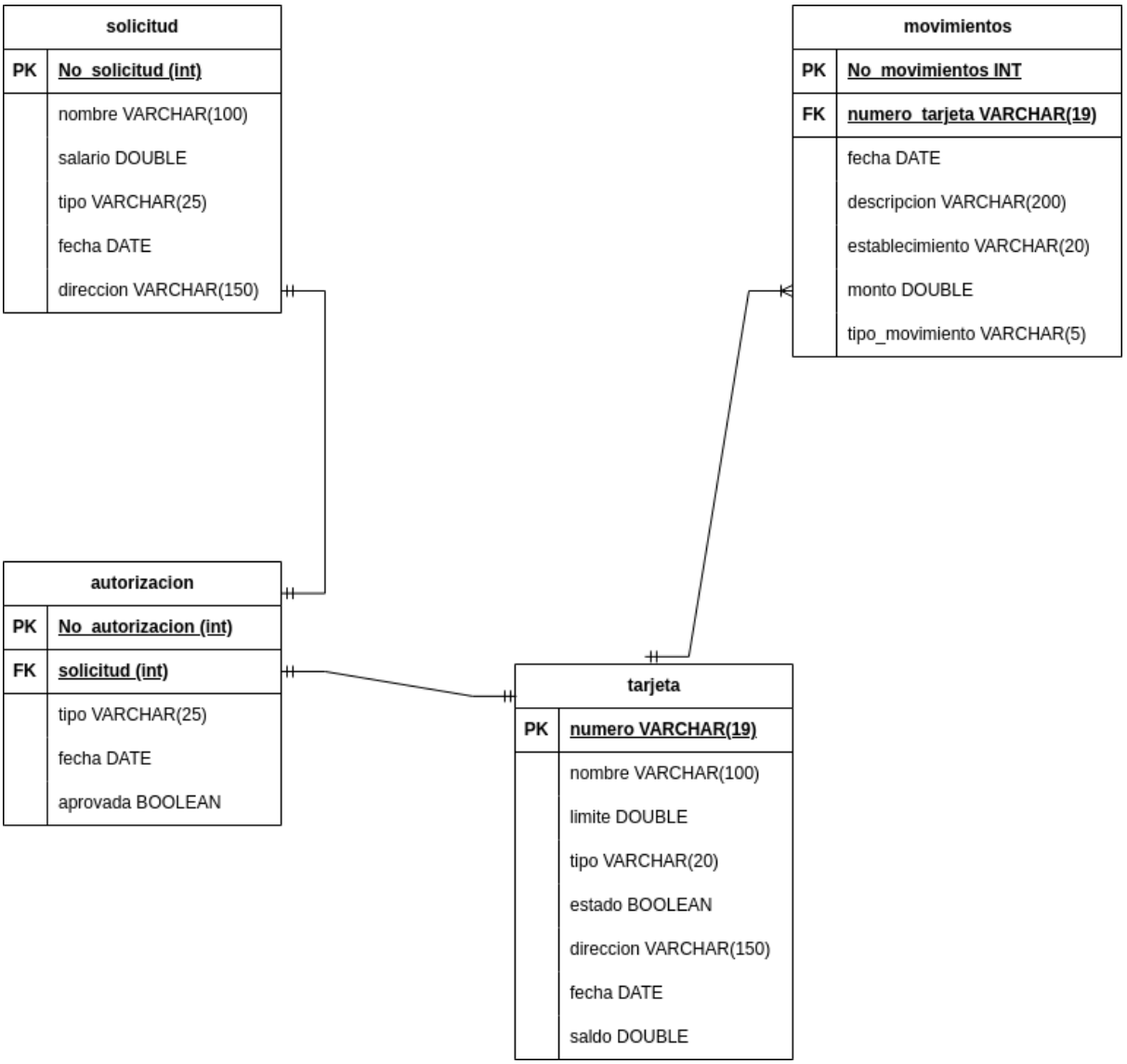

Diagrama de clases