Universidad de las Fuerzas Armadas

“Sistema de Gestión de Parking”

Integrantes:

1. Alan Nicolas Neira Guevara

Curso:

1323

Asignatura De Programación orientada a objetos

Docente:

LUIS ENRIQUE JARAMILLO MONTAÑO

05 de diciembre de 2024

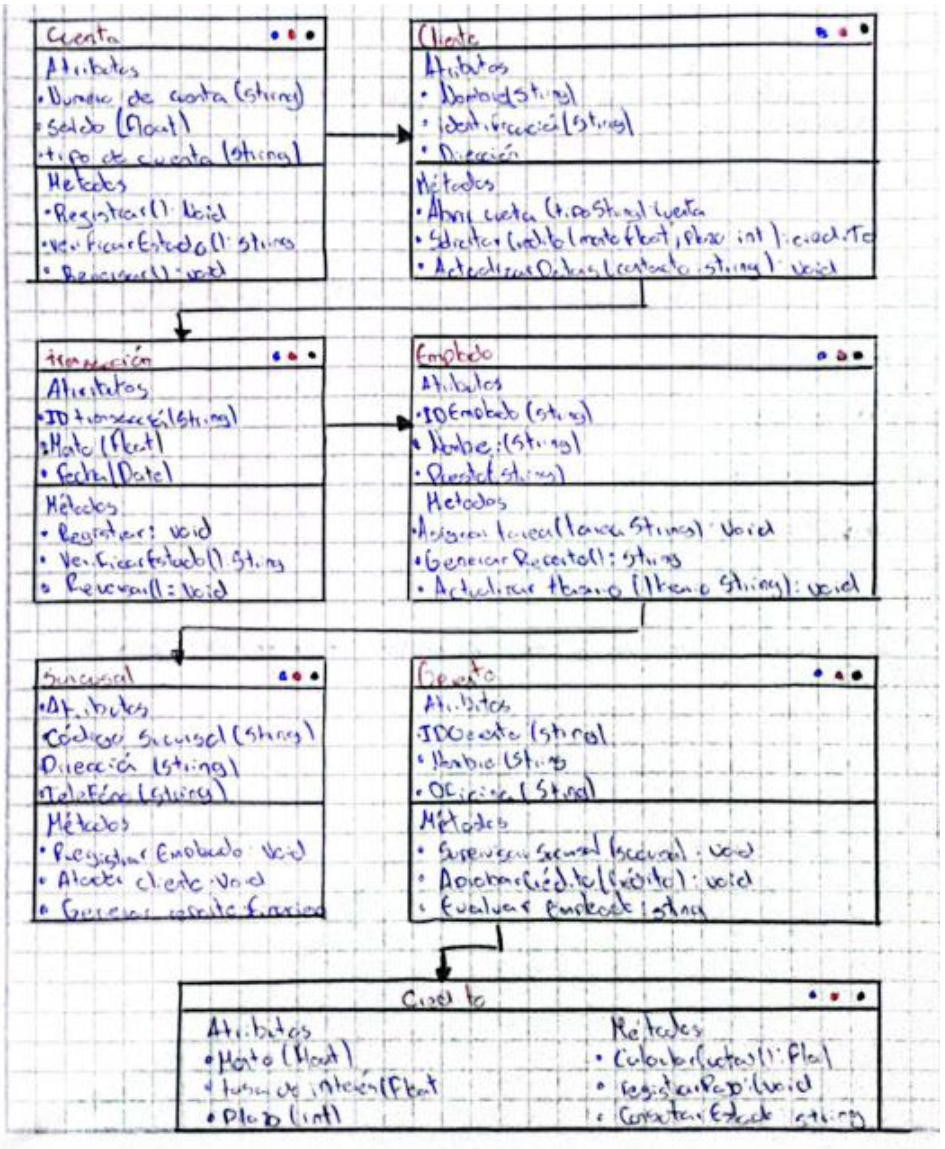
1. Introduccion

El objetivo fue diseñar **5 objetos** representados como clases con su respectivo diagrama UML y sus relaciones. Este diseño siguió los principios de **POO** para lograr un sistema estructurado y modular, enfocado en una simulación de entidades relacionadas.

2. Desarrollo

Se inició identificando las entidades principales del sistema (Cuenta, Cliente, Transacción, Empleado y Sucursal). Luego, se definieron las clases con sus respectivos atributos y métodos, asegurando que cada clase represente las características y acciones de las entidades. Posteriormente, se establecieron las relaciones entre las clases, como la asociación entre Cliente y Cuenta, y la conexión entre Sucursal, Empleados y Clientes. Finalmente, se construyó un diagrama UML en papel cuadriculado, mostrando las clases, atributos, métodos y sus relaciones de manera clara y ordenada.

3. UMLS


Cuenta
Atributos
Cliente
Atributos
Métodos
Transacción
Atributos
Métodos
Empleado
Atributos
Métodos
Sucursal
Atributos
Métodos
Gerente
Atributos
Métodos
Atributos
Métodos

4. Relaciones

Cuenta ←→ Cliente

La relación es Asociación

Un cliente puede poseer uno o mas cuentas bancarias

Transacción ←→ Cuenta

Relación : Agregación

Una transacción afecta a uno o mas cuentas, pero las cuentas existen de manera independiente

Empleado ←→ Sucursal

Relación : Composición

Un empleado depende de la sucursal a la que esta asignado

Gerente ←→ empleado

Relación: Asociación

Un gerente supervisa a multiples empleados

Crédito ←→ Cliente

Relación : Asociación

Un cliente puede tener uno o mas créditos otorgados por la entidad financiera.

5. Código

```
// Clase Cliente
public class Cliente {
    private String nombre;
    private String identificacion;
    private String direccion;
   public Cliente(String nombre, String identificacion, String direccion) {
        this.nombre = nombre;
        this.identificacion = identificacion;
        this.direccion = direccion;
    }
    public Cuenta abrirCuenta(String tipo) {
        System.out.println("Cuenta de tipo " + tipo + " abierta para el cliente " + nombre);
        return new Cuenta(tipo);
    }
    public Credito solicitarCredito(double monto, int plazo) {
        System.out.println("Crédito de " + monto + " solicitado a " + plazo + " meses por el cliente " + nombre);
        return new Credito(monto, plazo);
    }
    public void actualizarDatos(String nuevaDireccion) {
        this.direccion = nuevaDireccion;
        System.out.println("Datos actualizados. Nueva dirección: " + direccion);
    }
}
// Clase Cuenta
public class Cuenta {
    private String numeroCuenta;
    private double saldo;
    private String tipoCuenta;

    public Cuenta(String tipoCuenta) {
        this.numeroCuenta = "AUTO_GENERADO";
        this.saldo = 0.0;
        this.tipoCuenta = tipoCuenta;
    }
  public void retirar(double monto) {
        if (monto <= saldo) {
            saldo -= monto;
            System.out.println(monto + " retirados. Saldo actual: " + saldo);
        } else {
            System.out.println("Saldo insuficiente.");
        }
    }

    public void consultarSaldo() {
        System.out.println("Saldo disponible: " + saldo);
    }
}

// Clase Credito
public class Credito {
    private double monto;
    private double tasaInteres;
    private int plazo;

    public Credito(double monto, int plazo) {
        this.monto = monto;
        this.plazo = plazo;
        this.tasaInteres = 0.05; // Ejemplo de tasa fija
    }

    public double calcularCuotas() {
        double cuota = (monto * (1 + tasaInteres)) / plazo;
        System.out.println("Cuota mensual calculada: " + cuota);
        return cuota;
    }

    public void registrarPago(double montoPago) {
        monto -= montoPago;
        System.out.println("Pago de " + montoPago + " registrado. Monto restante: " + monto);
    }
```

Conclusión

El diseño presentado modela un sistema bancario simple, estructurado en torno a objetos y relaciones. Se logra un diagrama UML claro y funcional que respeta los principios de POO, permitiendo escalabilidad y mantenimiento en sistemas complejos.