Tecnológico de Costa Rica

Programación orientada a objetos

SoC

Estudiantes:

Diego Castillo Fallas

David Fernández Torres

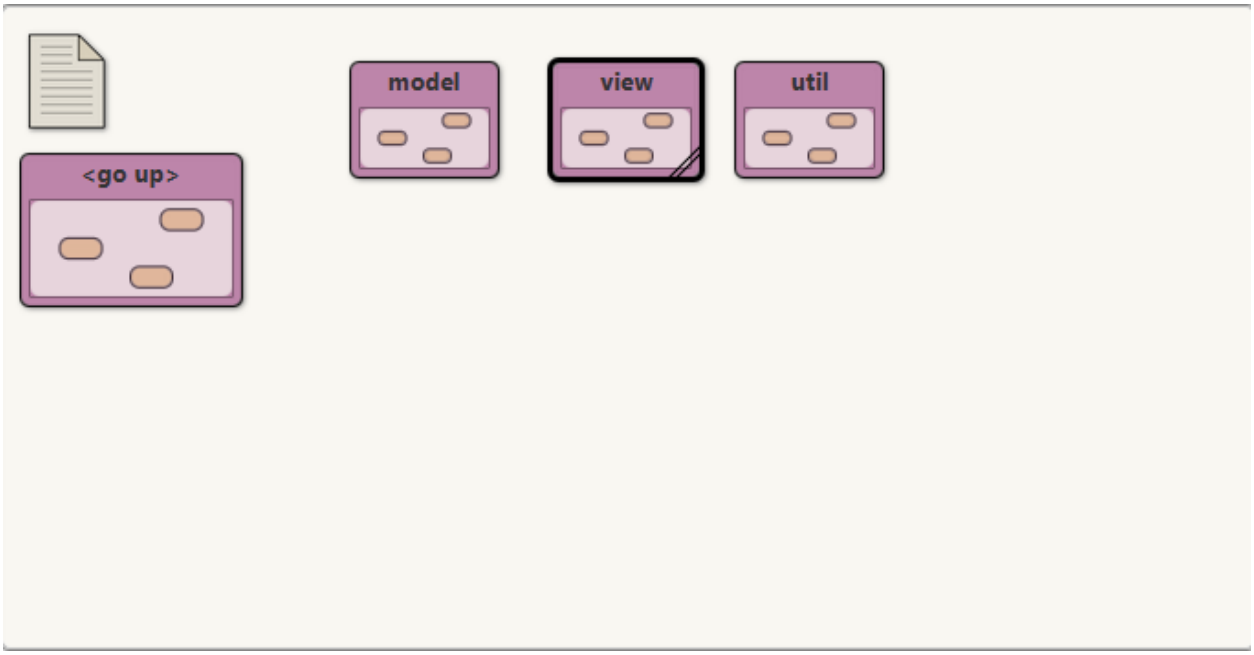
Josimar Spencer Suarez

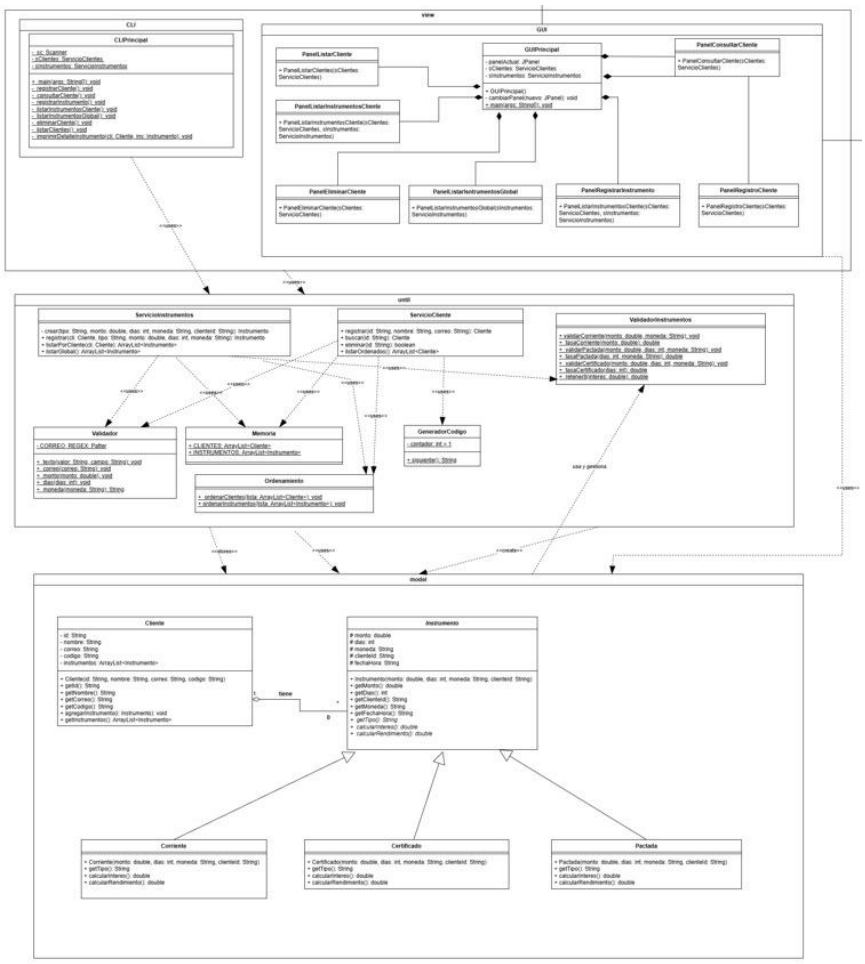
Profesor

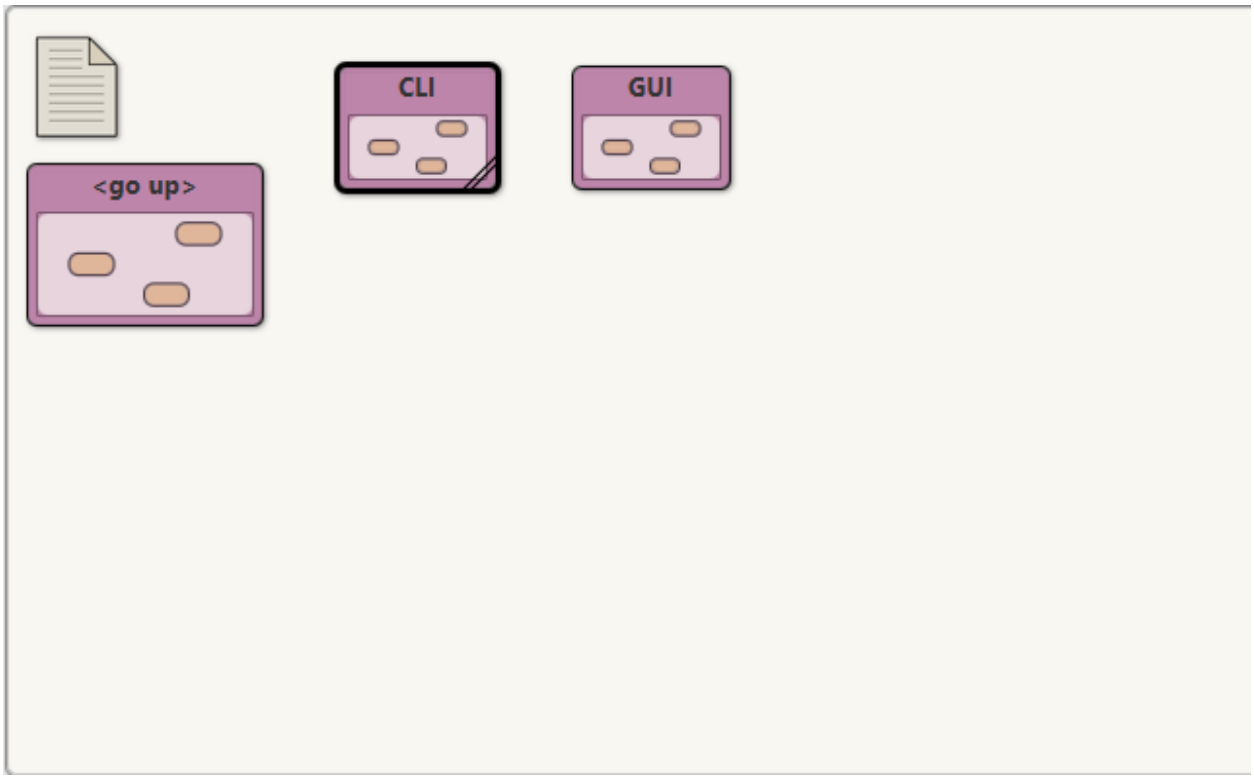
Luis Javier Chavarría Sánchez

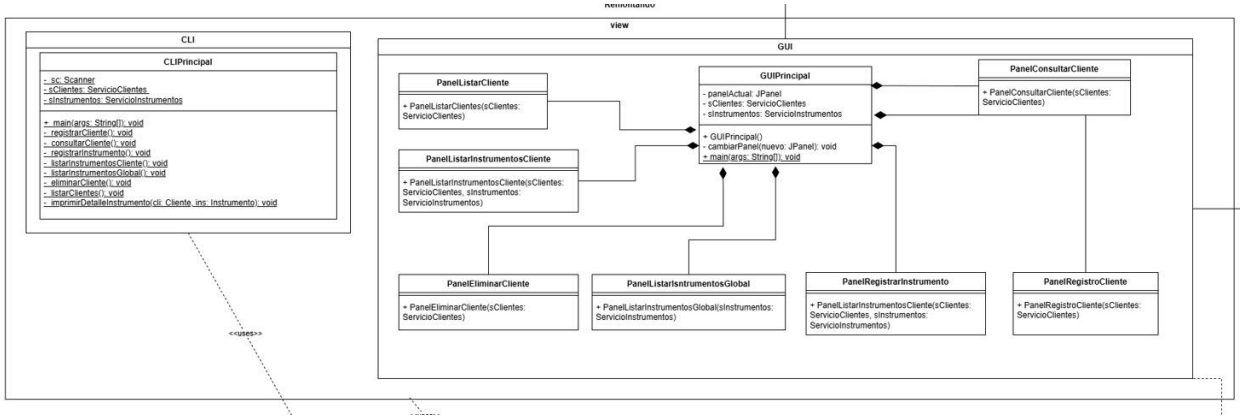
Semestre 2

Año 2025

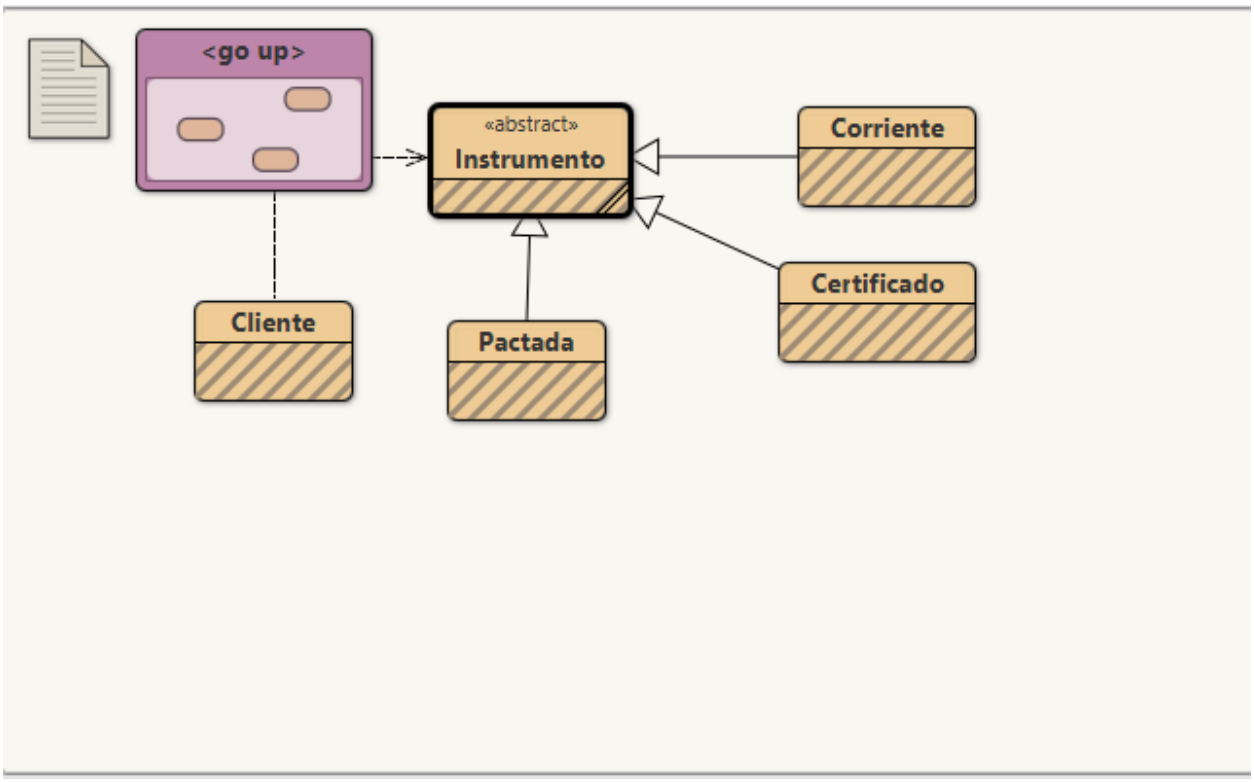

<go up>
model
view
util

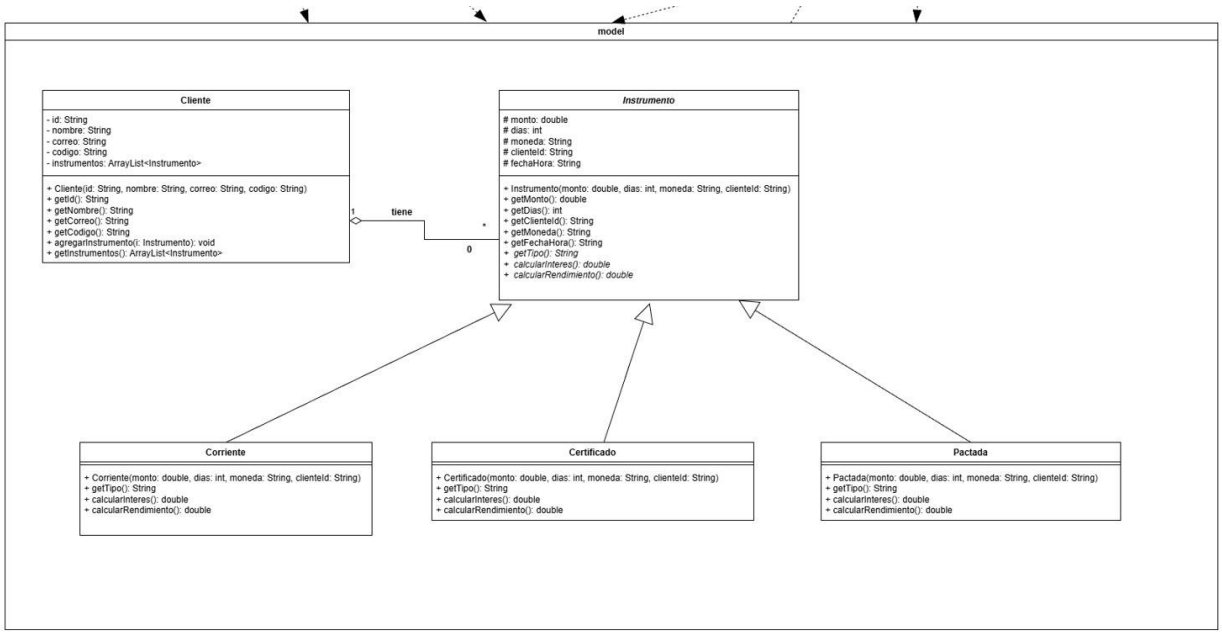

view
CLI
CLIPrincipal
- sc: Scanner
- iCClientes: IServicioClientes
- iInstrumentos: IServicioInstrumentos
+ main(args: String[]): void
- registrarCliente(): void
- consultarCliente(): void
- registrarInstrumento(): void
- listarInstrumentosCliente(): void
- listarInstrumentosGlobal(): void
- eliminarCliente(): void
- listarClientes(): void
- imprimirDetalleInstrumento(ci: Cliente, ins: Instrumento): void
GUI
GUIPrincipal
- panelActual: JPanel
- iClientes: IServicioClientes
- iInstrumentos: IServicioInstrumentos
+ GUIPrincipal()
- cambiarPanel(nuevo: JPanel): void
+ main(args: String[]): void
PanelListarCliente
+ PanelListarCliente(cc:IClientes: ServicioClientes)
PanelConsultarCliente
+ PanelConsultarCliente(cc:IClientes: ServicioClientes)
PanelListarInstrumentosCliente
+ PanelListarInstrumentos(Cliente(cc:IClientes: ServicioClientes, cins:Instrumentos: ServicioInstrumentos)
PanelEliminarCliente
+ PanelEliminarCliente(cc:IClientes: ServicioClientes)
PanelListarInstrumentosGlobal
+ PanelListarInstrumentosGlobal(cins:Instrumentos: ServicioInstrumentos)
PanelRegistrarInstrumento
+ PanelListarInstrumentos(Cliente(cc:IClientes: ServicioClientes, cins:Instrumentos: ServicioInstrumentos)
PanelRegistroCliente
+ PanelRegistroCliente(cc:IClientes: ServicioClientes)
<<uses>>
util
ServicioInstrumentos
- crearObjeto: String, monto: double, dias: int, moneda: String, clienteId: String): Instrumento
+ registrar(ci: Cliente, tipo: String, monto: double, dias: int, moneda: String): Instrumento
+ listarPorCliente(ci: Cliente): ArrayList<Instrumento>
+ listarGlobal(): ArrayList<Instrumento>
ServicioCliente
+ registrar(id: String, nombre: String, correo: String): Cliente
+ buscar(id: String): Cliente
+ eliminar(id: String): boolean
+ listarOrdenados(): ArrayList<Cliente>
ValidadorInstrumentos
+ validarCorriente(monto: double, moneda: String): void
+ tasaCorriente(monto: double): double
+ validarPactada(monto: double, dias: int, moneda: String): void
+ tasaPactada(dias: int, moneda: String): double
+ validarCertificado(monto: double, dias: int, moneda: String): void
+ tasaCertificado(dias: int): double
+ retener(interes: double): double
Validador
- CORREO: REGEX: Pattern
+ leerInput(prompt: String, campo: String): void
+ correo(correo: String): void
+ monto(monto: double): void
+ dias(dias: int): void
+ moneda(moneda: String): String
Memoria
+ CLIENTES: ArrayList<Cliente>
+ INSTRUMENTOS: ArrayList<Instrumento>
GeneradorCodigo
- contador: int = 1
+ siguiente(): String
Ordenamiento
+ ordenarClientes(lista: ArrayList<Cliente>): void
+ ordenarInstrumentos(lista: ArrayList<Instrumento>): void
usa y gestiona
<<creates>>
model
Cliente
- id: String
- nombre: String
- correo: String
- codigo: String
- instrumentos: ArrayList<Instrumento>
+ Cliente(id: String, nombre: String, correo: String, codigo: String)
+ getId(): String
+ getNombre(): String
+ getCorreo(): String
+ getCodigo(): String
+ agregarInstrumento(i: Instrumento): void
+ getInstrumentos(): ArrayList<Instrumento>
1
tiene
0..*
Instrumento
monto: double
dias: int
moneda: String
clienteId: String
fechaHora: String
+ Instrumento(monto: double, dias: int, moneda: String, clienteId: String)
+ getMonto(): double
+ getDias(): int
+ getClienteId(): String
+ getMoneda(): String
+ getFechaHora(): String
+ getTipo(): String
+ calcularInteres(): double
+ calcularRendimiento(): double
Corriente
+ Corriente(monto: double, dias: int, moneda: String, clienteId: String)
+ getTipo(): String
+ calcularInteres(): double
+ calcularRendimiento(): double
Certificado
+ Certificado(monto: double, dias: int, moneda: String, clienteId: String)
+ getTipo(): String
+ calcularInteres(): double
+ calcularRendimiento(): double
Pactada
+ Pactada(monto: double, dias: int, moneda: String, clienteId: String)
+ getTipo(): String
+ calcularInteres(): double
+ calcularRendimiento(): double

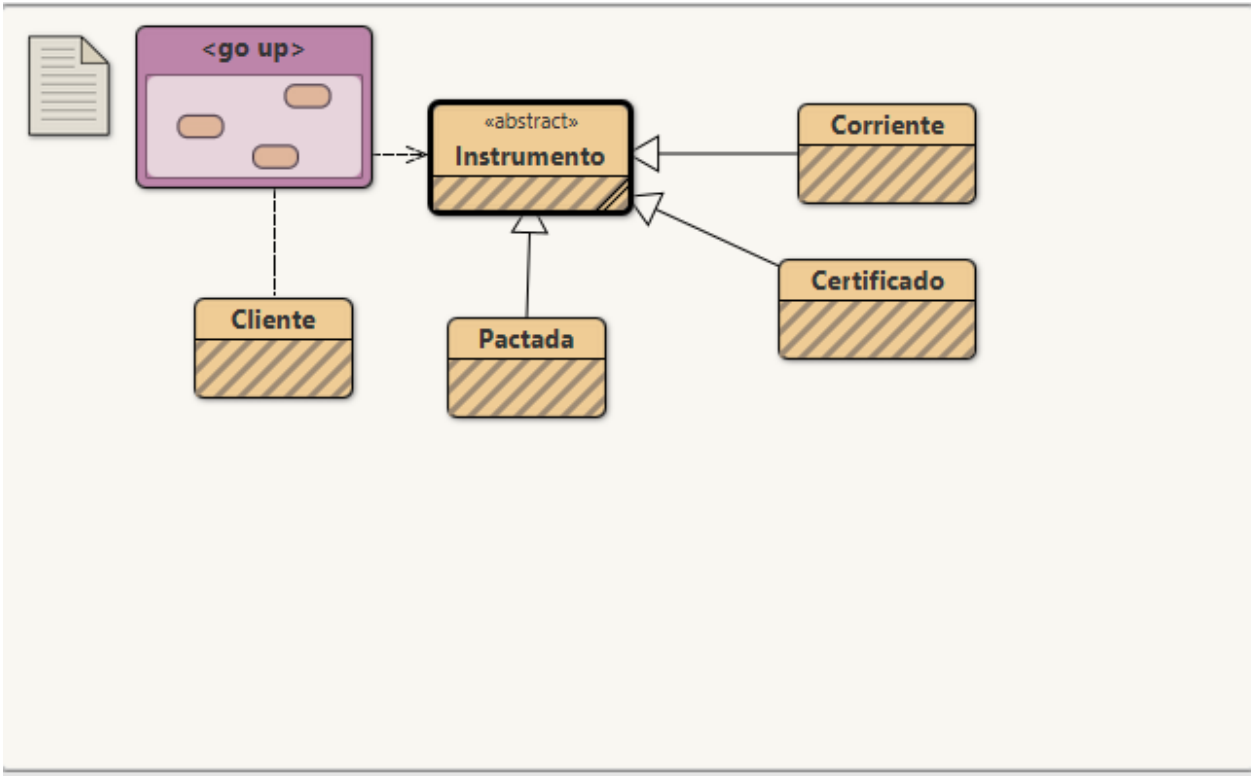


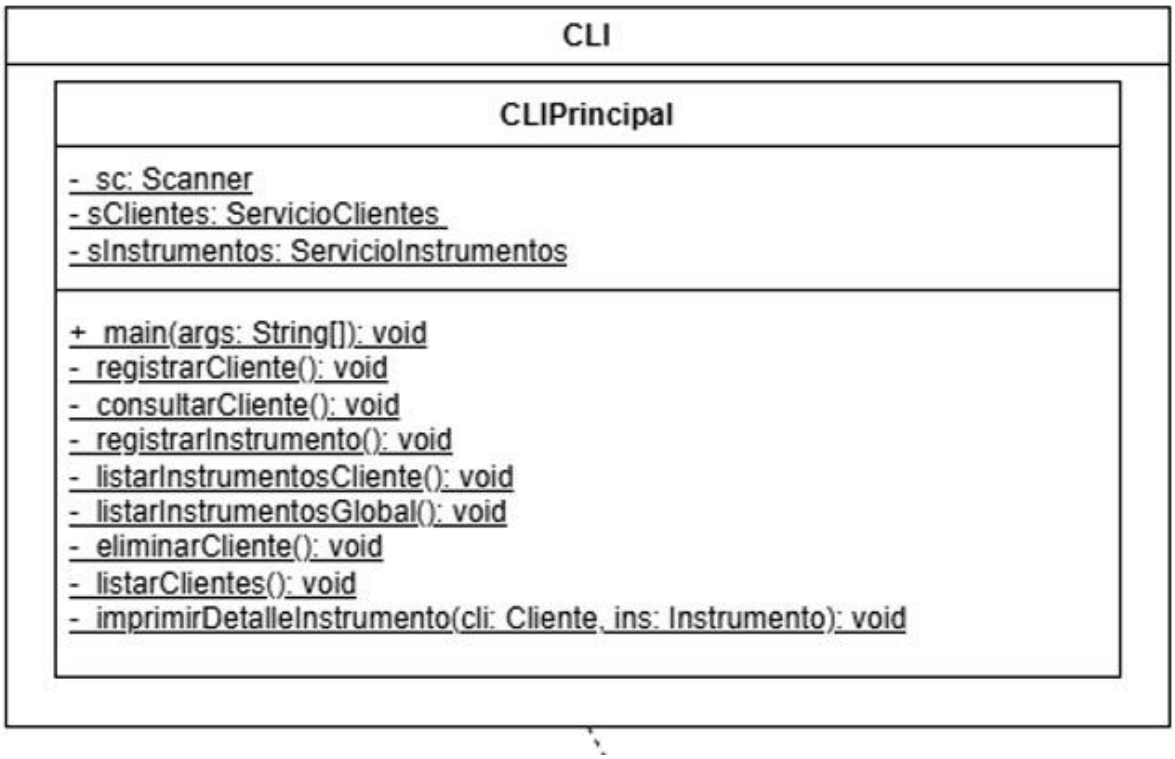


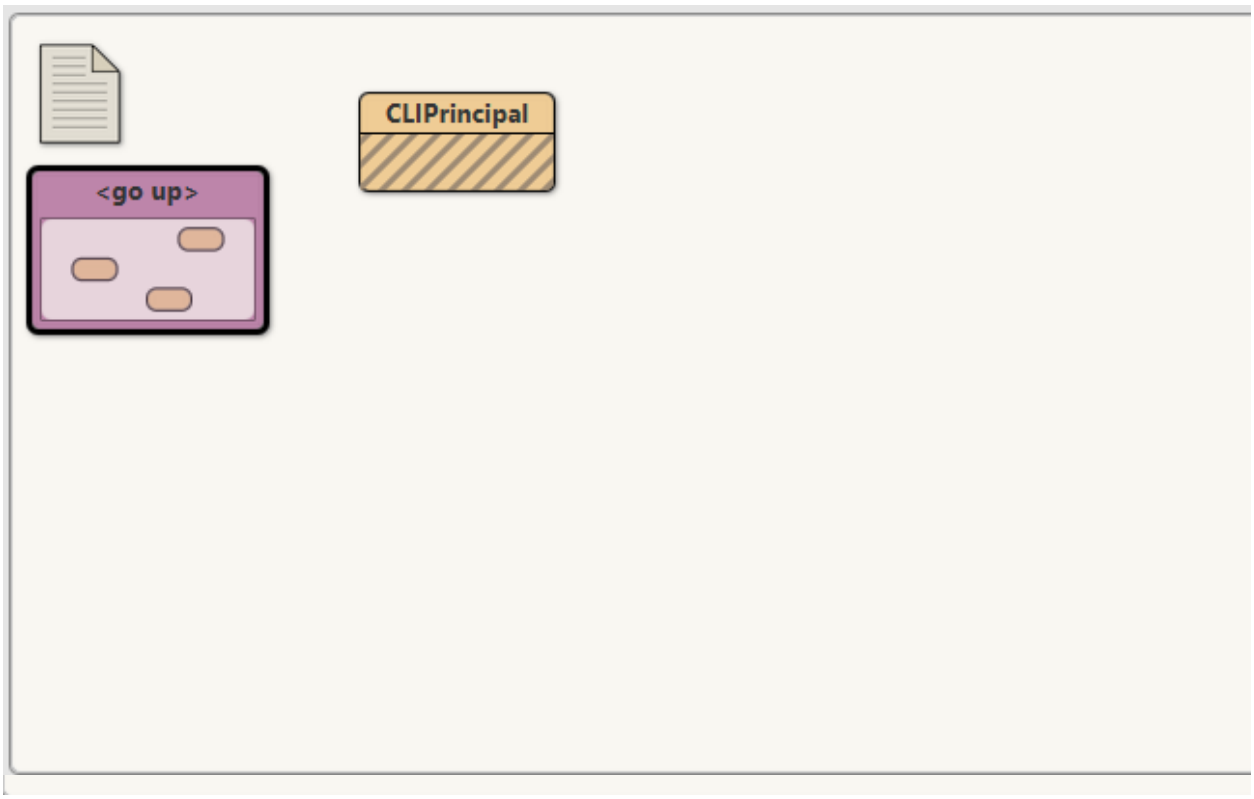
La solución separa toda la lógica del sistema en paquetes, donde “model” contiene el modelo del negocio, “util” agrupa validadores, memoria y servicios, y “view” contiene exclusivamente la capa de presentación, dividida en CLI y GUI. Esta separación física cumple con el principio SoC.

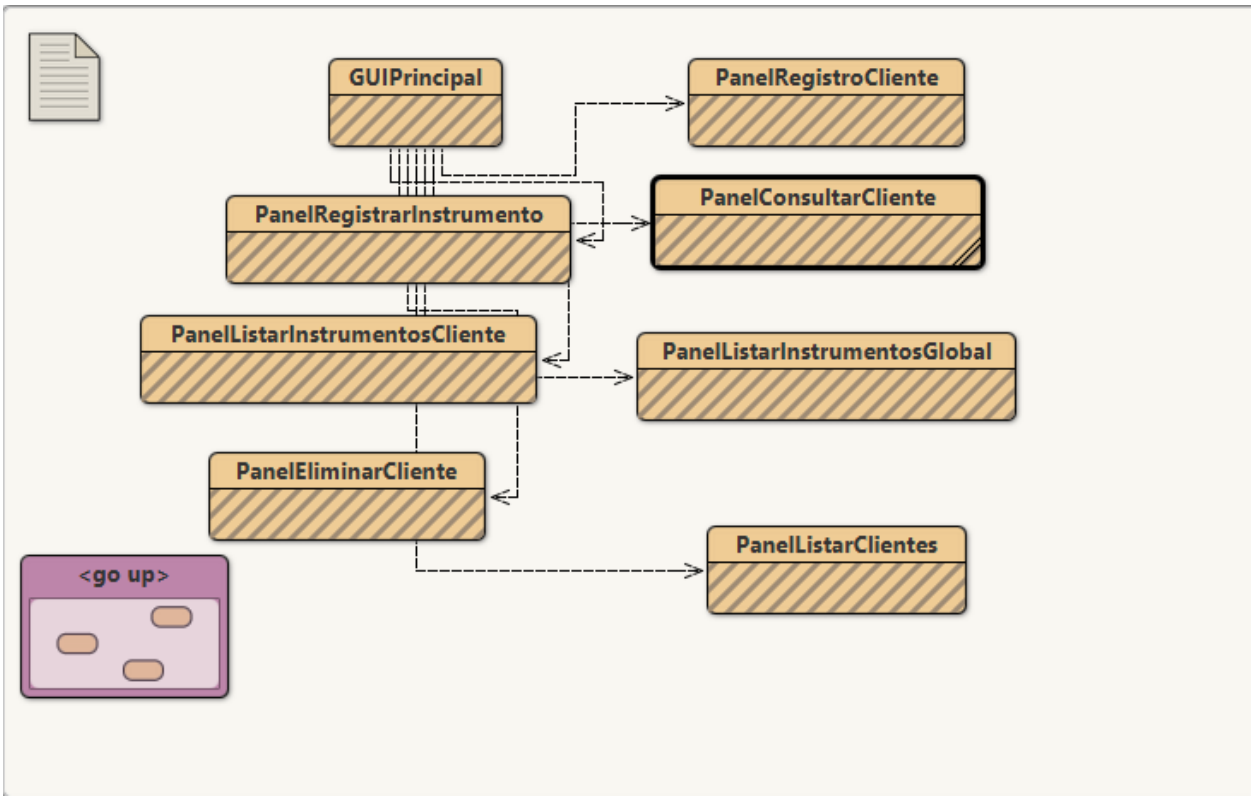

<go up>
«abstract»
Instrumento
Corriente
Certificado
Cliente
Pactada

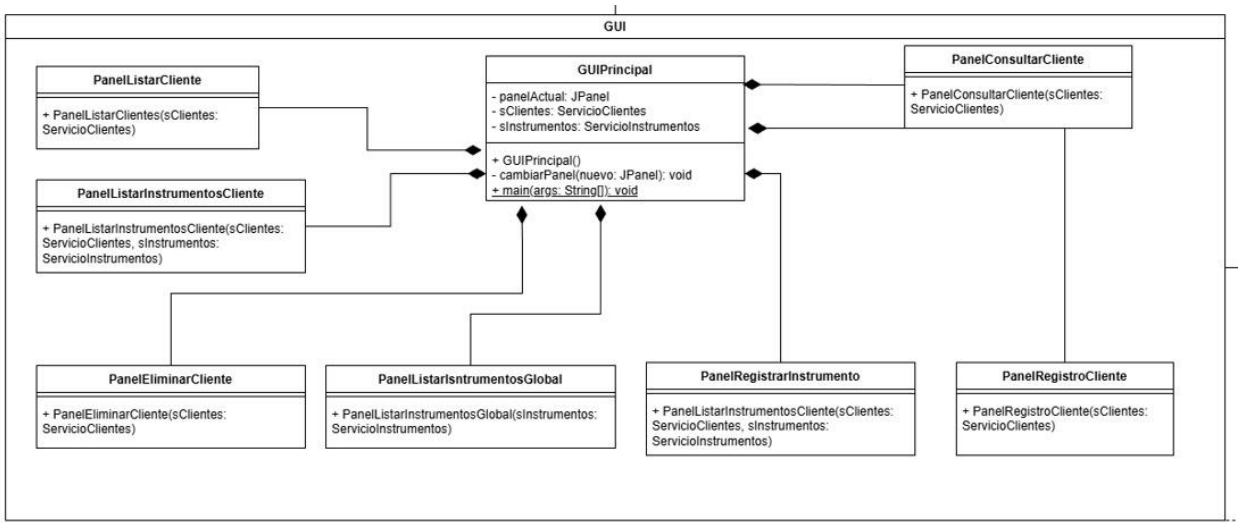

model
Cliente
- id: String
- nombre: String
- correo: String
- codigo: String
- instrumentos: ArrayList<Instrumento>
+ Cliente(id: String, nombre: String, correo: String, codigo: String)
+ getId(): String
+ getNombre(): String
+ getCorreo(): String
+ getCodigo(): String
+ agregarInstrumento(i: Instrumento): void
+ getInstrumentos(): ArrayList<Instrumento>
1
tiene
*
0
Instrumento
monto: double
dias: int
moneda: String
clienteId: String
fechaHora: String
+ Instrumento(monto: double, dias: int, moneda: String, clienteId: String)
+ getMonto(): double
+ getDias(): int
+ getClienteId(): String
+ getMoneda(): String
+ getFechaHora(): String
+ getTipo(): String
+ calcularInteres(): double
+ calcularRendimiento(): double
Corriente
+ Corriente(monto: double, dias: int, moneda: String, clienteId: String)
+ getTipo(): String
+ calcularInteres(): double
+ calcularRendimiento(): double
Certificado
+ Certificado(monto: double, dias: int, moneda: String, clienteId: String)
+ getTipo(): String
+ calcularInteres(): double
+ calcularRendimiento(): double
Pactada
+ Pactada(monto: double, dias: int, moneda: String, clienteId: String)
+ getTipo(): String
+ calcularInteres(): double
+ calcularRendimiento(): double

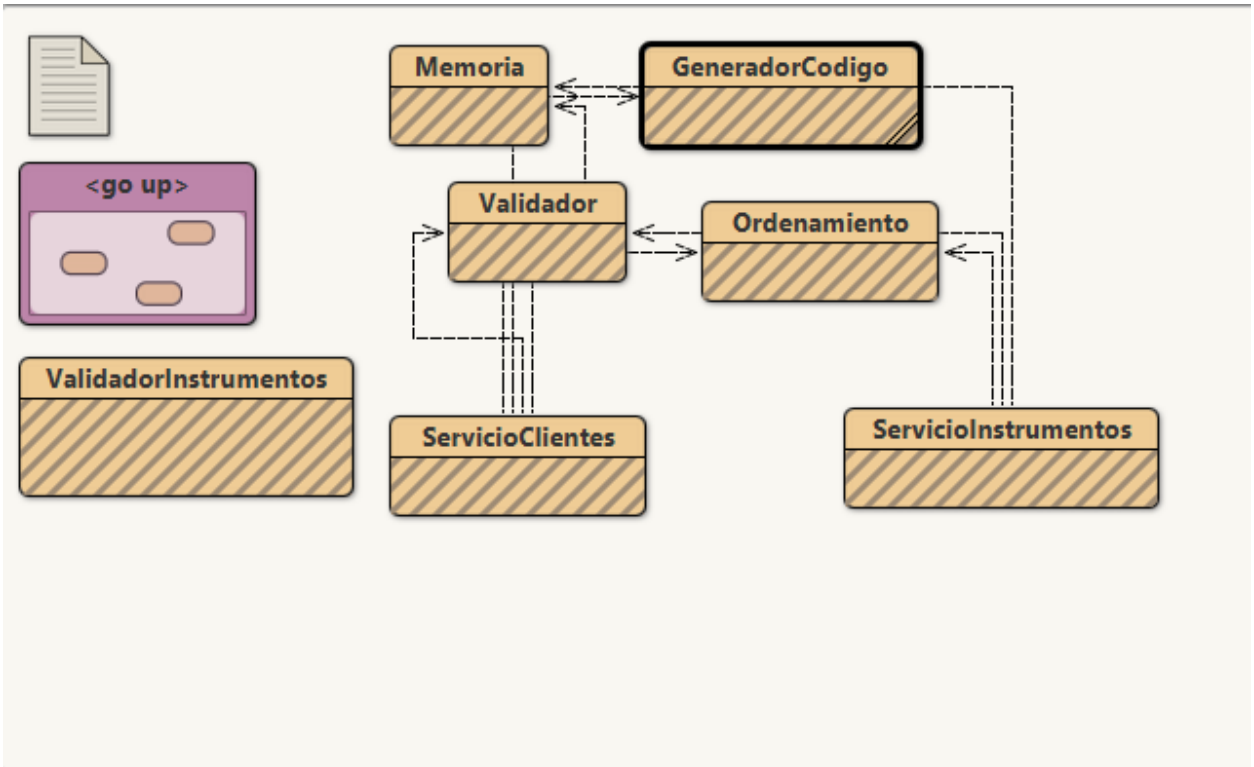

<go up>
«abstract»
Instrumento
Corriente
Certificado
Cliente
Pactada

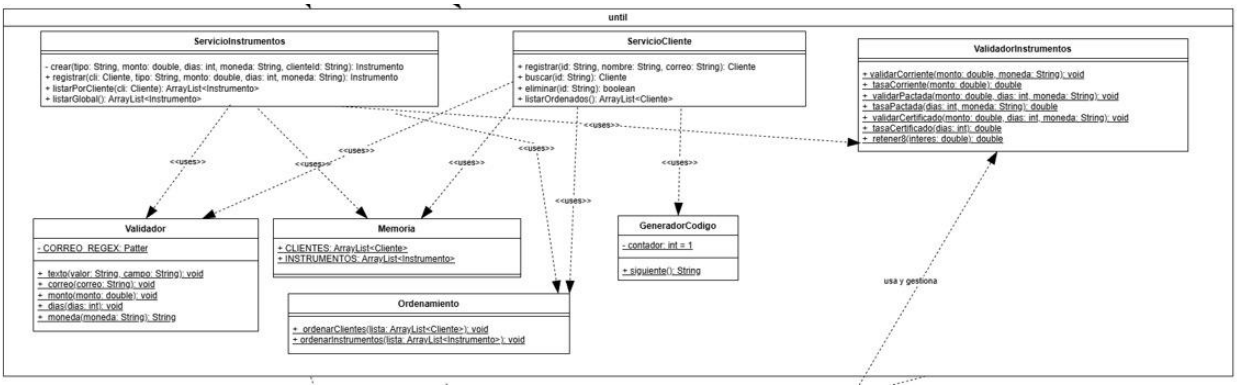

CLI
CLIPrincipal
- sc: Scanner
- sClientes: ServicioClientes
- sInstrumentos: ServicioInstrumentos
+ main(args: String[]): void
- registrarCliente(): void
- consultarCliente(): void
- registrarInstrumento(): void
- listarInstrumentosCliente(): void
- listarInstrumentosGlobal(): void
- eliminarCliente(): void
- listarClientes(): void
- imprimirDetalleInstrumento(cli: Cliente, ins: Instrumento): void


CLIPrincipal
<go up>


GUIPrincipal
PanelRegistroCliente
PanelConsultarCliente
PanelRegistrarInstrumento
PanelListarInstrumentosCliente
PanelListarInstrumentosGlobal
PanelEliminarCliente
PanelListarClientes
<go up>

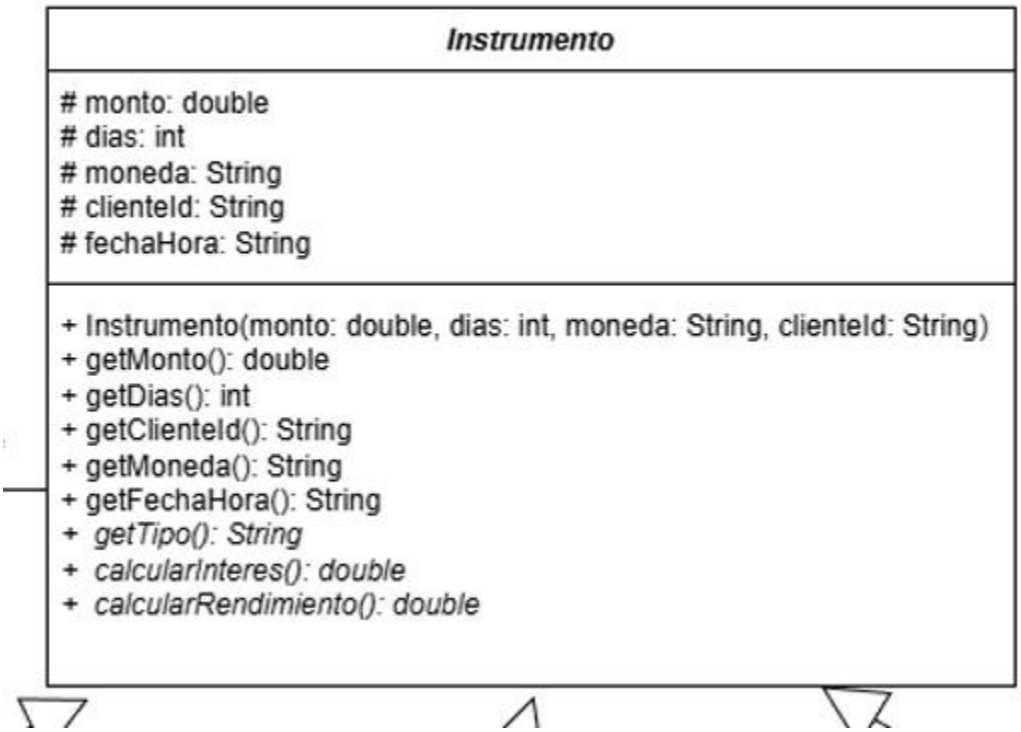

GUI
PanelListarCliente
+ PanelListarClientes(sClientes: ServicioClientes)
GUIPrincipal
- panelActual: JPanel
- sClientes: ServicioClientes
- sInstrumentos: ServicioInstrumentos
+ GUIPrincipal()
- cambiarPanel(nuevo: JPanel): void
+ main(args: String[]): void
PanelConsultarCliente
+ PanelConsultarCliente(sClientes: ServicioClientes)
PanelListarInstrumentosCliente
+ PanelListarInstrumentosCliente(sClientes: ServicioClientes, sInstrumentos: ServicioInstrumentos)
PanelEliminarCliente
+ PanelEliminarCliente(sClientes: ServicioClientes)
PanelListarIsntrumentosGlobal
+ PanelListarInstrumentosGlobal(sInstrumentos: ServicioInstrumentos)
PanelRegistrarInstrumento
+ PanelListarInstrumentosCliente(sClientes: ServicioClientes, sInstrumentos: ServicioInstrumentos)
PanelRegistroCliente
+ PanelRegistroCliente(sClientes: ServicioClientes)



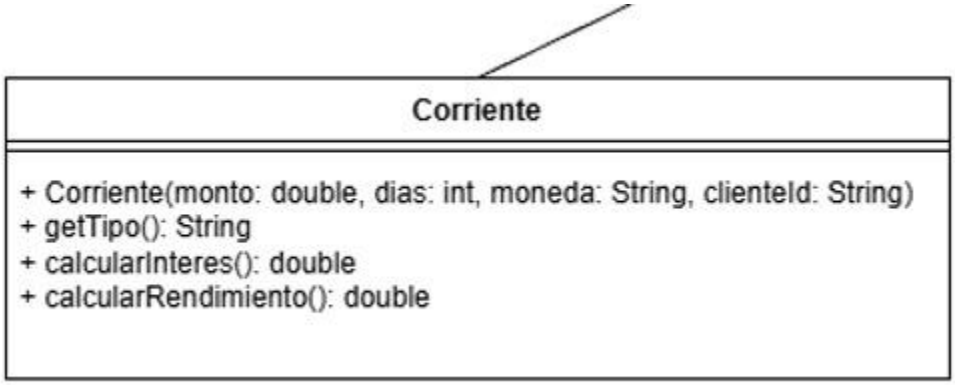


El diseño UML muestra la separación conceptual entre modelo, vista y servicios. Cada clase pertenece a una capa única y no se mezclan las responsabilidades.La clase Instrumento pertenece al paquete model y representa una abstracción del negocio.

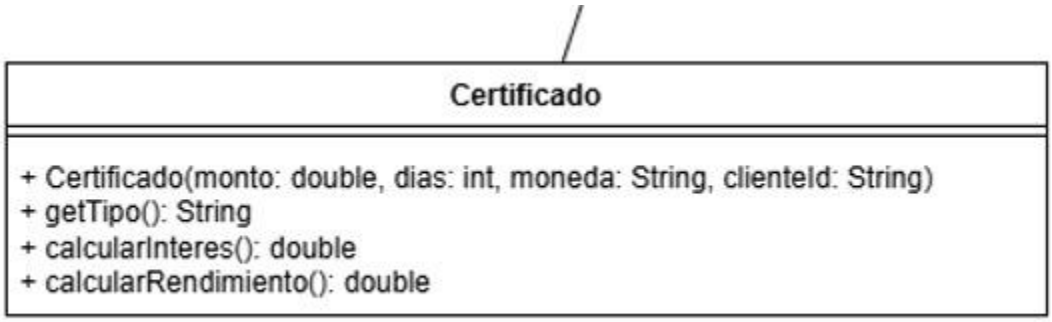
```java
package Remontando.model;

public abstract class Instrumento {

    protected double monto;
    protected int dias;
    protected String moneda;
    protected String clienteId;
    protected String fechaHora;

    public Instrumento(double monto, int dias, String moneda, String clienteId) {
        this.monto = monto;
        this.dias = dias;
        this.moneda = moneda;
        this.clienteId = clienteId;

        // Fecha y hora del registro
        this.fechaHora = java.time.LocalDateTime.now().format(
            java.time.format.DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")
        );
    }

    public double getMonto() { return monto; }
    public int getDias() { return dias; }
    public String getMoneda() { return moneda; }
    public String getClienteId() { return clienteId; }
    public String getFechaHora() { return fechaHora; }
```



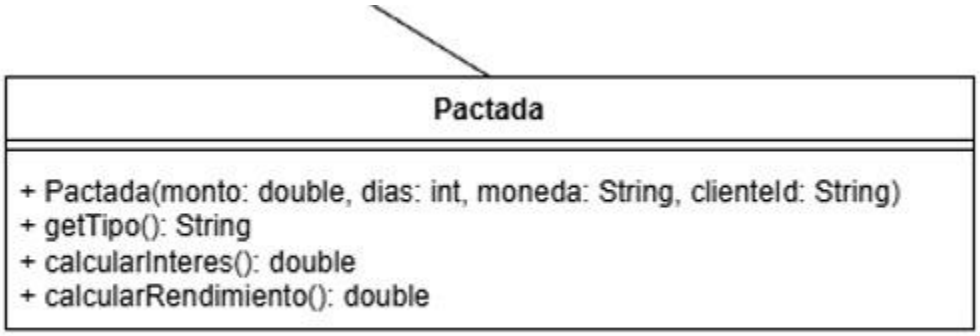
```java
package Remontando.model;

import Remontando.util.ValidadorInstrumentos;

public class Corriente extends Instrumento {

    public Corriente(double monto, int dias, String moneda, String clienteId) {
        super(monto, dias, moneda, clienteId);

        // VALIDACIONES OFICIALES
        ValidadorInstrumentos.validarCorriente(monto, moneda);
    }

    public String getTipo() { return "corriente"; }
```



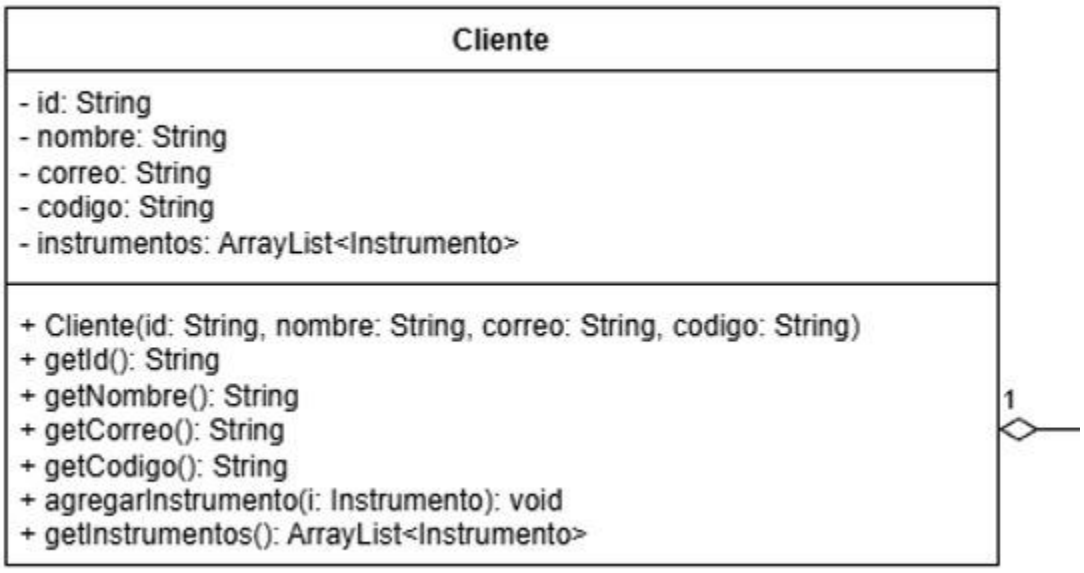
```java
package Remontando.model;

import Remontando.util.ValidadorInstrumentos;

public class Certificado extends Instrumento {

    public Certificado(double monto, int dias, String moneda, String clienteId) {
        super(monto, dias, moneda, clienteId);

        // VALIDACIONES OFICIALES
        ValidadorInstrumentos.validarCertificado(monto, dias, moneda);
    }

    public String getTipo() { return "certificado"; }
```




```java
package Remontando.model;

public class Pactada extends Instrumento {

    private static final double TASA = 0.008;

    public Pactada(double monto, int dias, String moneda, String clienteId) {
        super(monto, dias, moneda, clienteId);
    }

    public String getTipo() { return "pactada"; }
```



```java
package Remontando.model;

import java.util.ArrayList;

public class Cliente {

    private String id;
    private String nombre;
    private String correo;
    private String codigo;
    private ArrayList<Instrumento> instrumentos = new ArrayList<>();

    public Cliente(String id, String nombre, String correo, String codigo) {
        this.id = id;
        this.nombre = nombre;
        this.correo = correo;
        this.codigo = codigo;
    }

    public String getId() { return id; }
    public String getNombre() { return nombre; }
    public String getCorreo() { return correo; }
    public String getCodigo() { return codigo; }
```

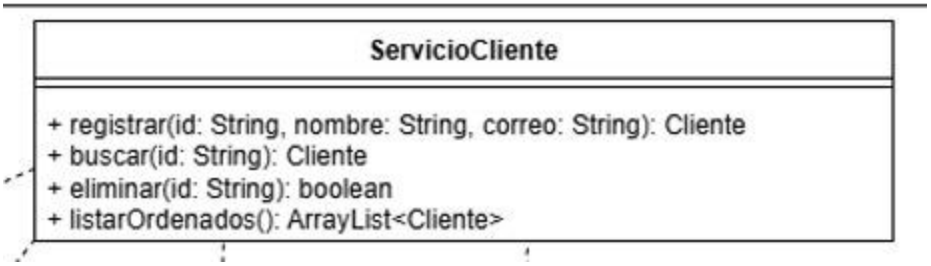


La clase Instrumento pertenece al paquete model y representa una abstracción del negocio. Contiene únicamente atributos privados, getters y métodos propios del dominio. Corriente, Pactada y Certificado heredan de ella, demostrando herencia y polimorfismo. No contiene código de interfaz ni validaciones, cumpliendo estrictamente el principio de SoC.

```
public Cliente registrar(String id, String nombre, String correo) {

    Validador.texto(id, "ID");
    Validador.texto(nombre, "nombre");
    Validador.correo(correo);

    if (buscar(id) != null)
        throw new IllegalArgumentException("Ese ID ya existe");

    String codigo = GeneradorCodigo.siguiente();

    Cliente c = new Cliente(id, nombre, correo, codigo);
    Memoria.CLIENTES.add(c);

    return c;
}
```

```
public Cliente buscar(String id) {
    for (Cliente c : Memoria.CLIENTES)
        if (c.getId().equals(id)) return c;
    return null;
}
```

```
public boolean eliminar(String id) {

    Cliente c = buscar(id);

    if (c == null) {
        return false; // no existe ese cliente
    }

    // eliminarlo de memoria
    return Memoria.CLIENTES.remove(c);
}
```

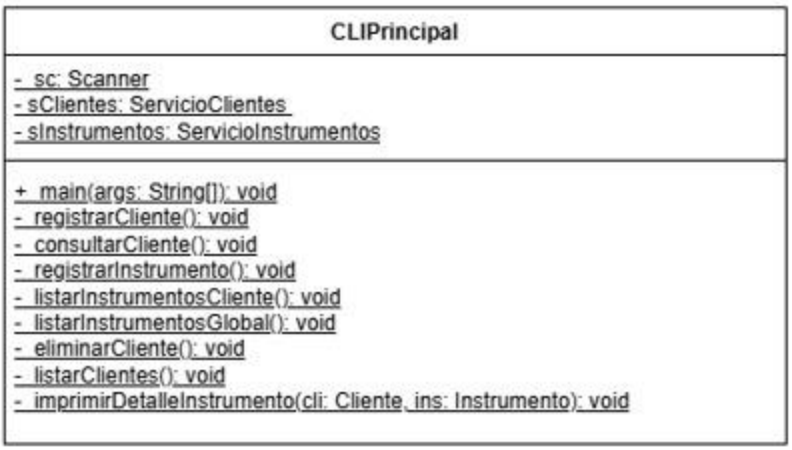


Las clases del paquete util contienen la lógica del negocio, como registrar, buscar y eliminar entidades, separando completamente la vista del procesamiento real.

```java
package Remontando.util;

import java.util.regex.Pattern;

public class Validador {

    private static final Pattern CORREO_REGEX =
        Pattern.compile("^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$");

    public static void texto(String valor, String campo) {
        if (valor == null || valor.trim().isEmpty()) {
            throw new IllegalArgumentException("Debe ingresar " + campo);
        }
    }

    public static void correo(String correo) {
        if (!CORREO_REGEX.matcher(correo).matches()) {
            throw new IllegalArgumentException("Correo inválido");
        }
    }

    public static void monto(double monto) {
        if (monto <= 0) throw new IllegalArgumentException("Monto inválido");
    }

    public static void dias(int dias) {
        if (dias <= 0) throw new IllegalArgumentException("Días inválidos");
    }

    public static String moneda(String moneda) {
        String m = moneda.toUpperCase();
        if (!m.equals("CRC") && !m.equals("USD")) {
            throw new IllegalArgumentException("Moneda inválida");
        }
        return m;
    }
}
```

La validación de datos se encuentra en una capa independiente (util.Validador), separada de las clases del modelo y la vista, cumpliendo la robustez y separación de responsabilidades exigida.

```java
import Remontando.util.ServicioClientes;
import Remontando.util.ServicioInstrumentos;

import Remontando.model.Cliente;
import Remontando.model.Instrumento;

public class CLIPrincipal {

    private static Scanner sc = new Scanner(System.in);
    private static ServicioClientes sClientes = new ServicioClientes();
    private static ServicioInstrumentos sInstrumentos = new ServicioInstrumentos();

    public static void main(String[] args) {

        while (true) {
            System.out.println("\n===== MENU PRINCIPAL =====");
            System.out.println("1. Registrar cliente");
            System.out.println("2. Consultar cliente por ID");
            System.out.println("3. Registrar instrumento");
            System.out.println("4. Listar instrumentos del cliente");
            System.out.println("5. Listar instrumentos globales");
            System.out.println("6. Eliminar cliente");
            System.out.println("7. Listar clientes");
            System.out.println("0. Salir");
            System.out.print("Seleccione: ");

            String opcion = sc.nextLine();

            switch (opcion) {

                case "1": registrarCliente(); break;
                case "2": consultarCliente(); break;
                case "3": registrarInstrumento(); break;
                case "4": listarInstrumentosCliente(); break;
                case "5": listarInstrumentosGlobal(); break;
                case "6": eliminarCliente(); break;
                case "7": listarClientes(); break;

                case "0":
                    System.out.println("Saliendo...");
                    return;

                default:
                    System.out.println("Opción inválida.");
            }
        }
    }
```



La vista CLI solo controla interacción con usuario y delega toda la lógica a los servicios, cumpliendo SoC

```java
package Remontando.view.GUI;


import javax.swing.*;
import java.awt.*;

import Remontando.util.ServicioClientes;
import Remontando.util.ServicioInstrumentos;
import Remontando.model.Cliente;
import Remontando.model.Instrumento;

public class PanelRegistrarInstrumento extends JPanel {
    public PanelRegistrarInstrumento(ServicioClientes sClientes,
                                     ServicioInstrumentos sInstrumentos) {

        setLayout(new BorderLayout());

        // FORMULARIO
        JPanel form = new JPanel(new GridLayout(6, 2, 5, 5));

        JTextField txtId = new JTextField();
        JTextField txtTipo = new JTextField();
        JTextField txtMonto = new JTextField();
        JTextField txtDias = new JTextField();
        JTextField txtMoneda = new JTextField();

        form.add(new JLabel("ID del cliente:"));
        form.add(txtId);

        form.add(new JLabel("Tipo (corriente/pactada/certificado):"));
        form.add(txtTipo);

        form.add(new JLabel("Monto:"));
        form.add(txtMonto);

        form.add(new JLabel("Días:"));
        form.add(txtDias);

        form.add(new JLabel("Moneda (CRC/colones o USD/dólares):"));
        form.add(txtMoneda);

        JButton btn = new JButton("Registrar instrumento");
        form.add(btn);

        add(form, BorderLayout.NORTH);

        // ÁREA DE TEXTO DE SALIDA
        JTextArea salida = new JTextArea();
        salida.setEditable(false);
        salida.setFont(new Font("Monospaced", Font.PLAIN, 13));
        JScrollPane scroll = new JScrollPane(salida);
        add(scroll, BorderLayout.CENTER);

        // ACCIÓN DEL BOTÓN
```

```java
// ACCIÓN DEL BOTÓN
btn.addActionListener(e -> {
    try {
        Cliente cli = sClientes.buscar(txtId.getText());

        if (cli == null) {
            JOptionPane.showMessageDialog(this,
                    "No existe un cliente con ese ID. No se puede registrar un instrumento.");
            return;
        }

        double monto = Double.parseDouble(txtMonto.getText());
        int dias = Integer.parseInt(txtDias.getText());

        Instrumento ins = sInstrumentos.registrar(
                cli,
                txtTipo.getText(),
                monto,
                dias,
                txtMoneda.getText()
        );

        // Formato de moneda
        String nombreMoneda = ins.getMoneda().equals("CRC") ? "colones" : "dólares";

        // Formatos numéricos
        String montoTxt   = String.format("%,.0f", ins.getMonto());
        String interesTxt = String.format("%,.2f", ins.calcularInteres());
        String saldoTxt   = String.format("%,.2f", ins.calcularRendimiento());

        // Nombre formal del tipo
        String nombreSistema;
        double tasaAnual;

        switch (ins.getTipo().toLowerCase()) {
            case "corriente":
                nombreSistema = "Cuenta corriente";
                tasaAnual = 2.0;
                break;

            case "pactada":
                nombreSistema = "Cuenta pactada";
                tasaAnual = 4.0;
                break;

            default:
                nombreSistema = "Certificado a plazo";
                tasaAnual = 6.0;
                break;
        }

        // Construir salida igual al ejemplo del PDF:
        StringBuilder sb = new StringBuilder();

        sb.append("--- Datos del cliente y registro de su instrumento de ahorro o inversión ---\n");
        sb.append("Cliente:  ").append(cli.getNombre()).append("\n");
        sb.append("Código de Cliente ").append(cli.getCodigo())
                .append(", ID ").append(cli.getId())
                .append(", correo: ").append(cli.getCorreo()).append("\n\n");

        sb.append("Monto de ahorro e inversión:\t").append(montoTxt)
```

```
    switch (ins.getTipo().toLowerCase()) {
        case "corriente":
            nombreSistema = "Cuenta corriente";
            tasaAnual = 2.0;
            break;

        case "pactada":
            nombreSistema = "Cuenta pactada";
            tasaAnual = 4.0;
            break;

        default:
            nombreSistema = "Certificado a plazo";
            tasaAnual = 6.0;
            break;
    }

    StringBuilder sb = new StringBuilder();

    sb.append("--- Datos del cliente y registro de su instrumento de ahorro o inversión ---\n");
    sb.append("Cliente:  ").append(cli.getNombre()).append("\n");
    sb.append("Código de Cliente ").append(cli.getCodigo())
            .append(", ID ").append(cli.getId())
            .append(", correo: ").append(cli.getCorreo()).append("\n\n");

    sb.append("Monto de ahorro e inversión:\t").append(montoTxt)
            .append("  ").append(nombreMoneda).append("\n");
    sb.append("Plazo de la inversión días:\t").append(ins.getDias()).append(" días\n");
    sb.append("Sistema de ahorro e inversión:\t").append(nombreSistema).append("\n");
    sb.append(String.format("Interés anual correspondiente:\t%.2f %%\n\n", tasaAnual));

    sb.append("Rendimiento\n");
    sb.append("Plazo en días   Monto de ahorro e inversión   Intereses ganados   Saldo Final\n");
    sb.append(String.format(
            "%12d   %25s   %16s   %12s\n",
            ins.getDias(), montoTxt, interesTxt, saldoTxt
    ));

    sb.append("-- Última línea --\n");

    salida.setText(sb.toString());
} catch (Exception ex) {
    JOptionPane.showMessageDialog(this, ex.getMessage());
}
```

La GUI solo muestra información y envía datos a los servicios. No contiene cálculos ni validaciones complejas, cumpliendo SoC.

Estrategias que se tomaron en consideración para el cumplimiento de la separación de responsabilidades:

. Capas Arquitectónicas Claras

Se planteo la solución por medio de este codigo que se dividió el sistema en tres capas independiente

. Delegación Total de Lógica a Servicios

Toda operación significativa (buscar cliente, registrar instrumento, calcular intereses) se realiza en los servicios. La vista solo invoca métodos del servicio y muestra el resultado.

. Modelo

Las clases del modelo solo tienen atributos y getters. No realizan validaciones, cálculos ni tienen dependencias con interfaces.