



Universidad de San Carlos de Guatemala  
Facultad de Ingeniería

**PROYECTO #1**  
**MANUAL TÉCNICO**  
**USAC BANK**

**Introducción a la Programación 1**  
Auxiliar Sebastian Alejandro Velasquez Bonilla

Guatemala, marzo de 2025

## **Introducción**

Este manual técnico documenta la arquitectura, componentes y funcionamiento interno del sistema bancario USAC Bank desarrollado para el curso IPC1. El sistema permite la gestión de clientes, cuentas y transacciones bancarias mediante una interfaz gráfica de usuario.

## **Arquitectura del Sistema**

- Patrón MVC

El sistema utiliza el patrón de diseño Modelo-Vista-Controlador (MVC)

1. Modelo: Clases que representan entidades del negocio (Cliente, Cuenta, Transaccion, Bitacora)
2. Vista: Interfaces gráficas que muestran la información al usuario
3. Controlador: Clases que gestionan la lógica de negocio (ClienteController, TransaccionController)

## **Estructura de Paquetes**

- **Componentes Principales**

### ***Modelos:***

Cliente, representa un cliente del banco con atributos como CUI, nombre y apellido.

```
Cliente.java x
src > main > java > com > usacbank > model > Cliente.java > Cliente > cui

6 public class Cliente {
7     private String cui;
8     private String nombre;
9     private String apellido;
10    private List<Cuenta> cuentas;
11
12    public Cliente(String cui, String nombre, String apellido) {
13        this.cui = cui;
14        this.nombre = nombre;
15        this.apellido = apellido;
16        this.cuentas = new ArrayList<>();
17    }
18
19    public String getCui() {
20        return cui;
21    }
22
23    public String getNombre() {
24        return nombre;
25    }
26
27    public String getApellido() {
28        return apellido;
29    }
30
31    public List<Cuenta> getCuentas() {
32        return cuentas;
33    }
34
35    public void agregarCuenta(Cuenta cuenta) {
36        cuentas.add(cuenta);
37    }
38
39    @Override
40    public String toString() {
41        return cui + " - " + nombre + " " + apellido;
42    }
43 }
44
```

Cuenta, representa una cuenta bancaria asociada a un cliente.

```
Cuenta.java M X
src > main > java > com > usacbank > model > Cuenta.java > Cuenta

3 public class Cuenta {
4     private static int contadorCuentas = 1;
5     private String id;
6     private Cliente cliente;
7     private double saldo;
8
9     public Cuenta(Cliente cliente) {
10         this.id = generarId();
11         this.cliente = cliente;
12         this.saldo = 0.0;
13     }
14
15     private String generarId() {
16         return "208825" + contadorCuentas++;
17     }
18
19     public String getId() {
20         return id;
21     }
22
23     public Cliente getCliente() {
24         return cliente;
25     }
26
27     public double getSaldo() {
28         return saldo;
29     }
30
31     public void depositar(double monto) {
32         saldo += monto;
33     }
34
35     public boolean retirar(double monto) {
36         if (monto > saldo)
37             return false;
38         saldo -= monto;
39         return true;
40     }
41 }
```

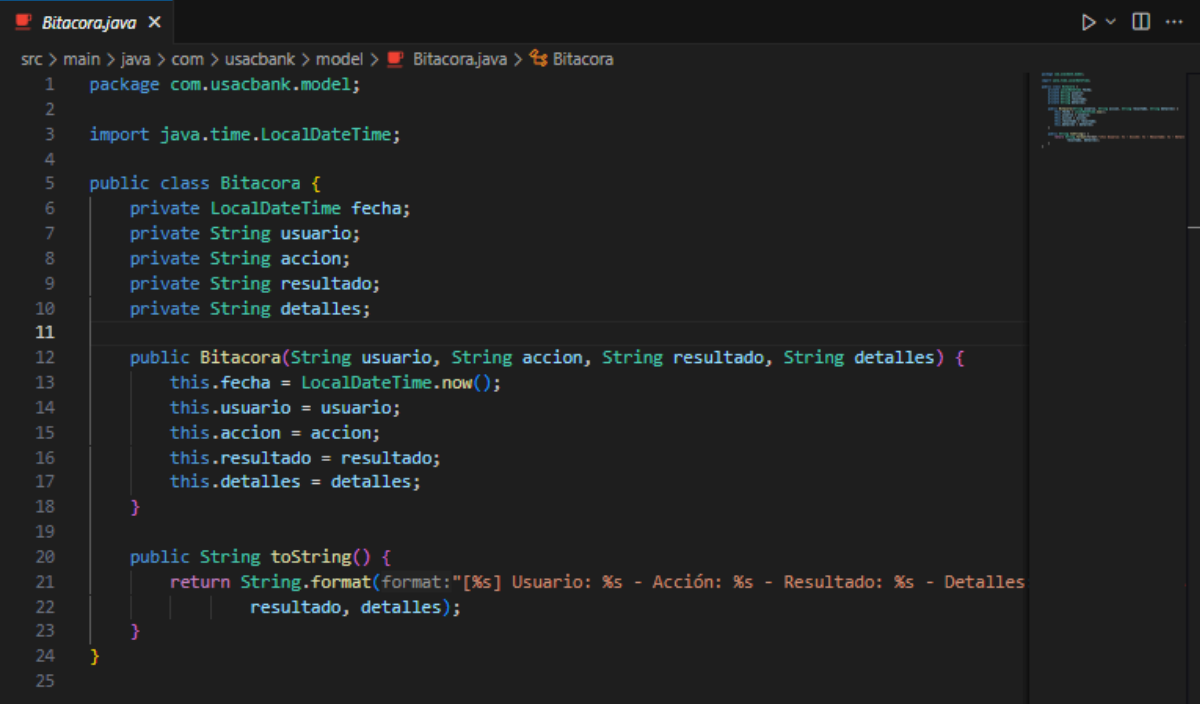
Transacción, registra operaciones de depósito o retiro realizadas en las cuentas.

```
Transaccion.java M X
src > main > java > com > usacbank > model > Transaccion.java > Transaccion > montoDebito

6 public class Transaccion {
7     private static int contadorTransacciones = 1;
8     private int id;
9     private Cuenta cuenta;
10    private Date fecha;
11    private String detalle;
12    // Para retiros
13    private double montoDebito;
14    // Para depósitos
15    private double montoCredito;
16    private double saldoResultante;
17
18    public Transaccion(Cuenta cuenta, double monto, String tipo) {
19        this.id = contadorTransacciones++;
20        this.cuenta = cuenta;
21        this.fecha = new Date();
22
23        if (tipo.equals(anObject:"DEPOSITO")) {
24            this.detalle = "Depósito";
25            this.montoCredito = monto;
26            this.montoDebito = 0;
27        } else if (tipo.equals(anObject:"RETIRO")) {
28            this.detalle = "Retiro";
29            this.montoDebito = monto;
30            this.montoCredito = 0;
31        }
32
33        this.saldoResultante = cuenta.getSaldo();
34    }
35
36    public int getId() {
37        return id;
38    }
39
40    public Cuenta getCuenta() {
41        return cuenta;
42    }
43
44    public Date getFecha() {
```

```
45        return fecha;
46    }
47
48    public String getDetalle() {
49        return detalle;
50    }
51
52    public double getMontoDebito() {
53        return montoDebito;
54    }
55
56    public double getMontoCredito() {
57        return montoCredito;
58    }
59
60    public double getSaldoResultante() {
61        return saldoResultante;
62    }
63
64    public String getFechaFormateada() {
65        SimpleDateFormat sdf = new SimpleDateFormat(pattern:"dd/MM/yyyy HH:mm:ss");
66        return sdf.format(fecha);
67    }
68 }
```

Bitácora, registra eventos y operaciones del sistema para control en desarrollo.



```
src > main > java > com > usacbank > model > Bitacora.java > Bitacora
1  package com.usacbank.model;
2
3  import java.time.LocalDateTime;
4
5  public class Bitacora {
6      private LocalDateTime fecha;
7      private String usuario;
8      private String accion;
9      private String resultado;
10     private String detalles;
11
12     public Bitacora(String usuario, String accion, String resultado, String detalles) {
13         this.fecha = LocalDateTime.now();
14         this.usuario = usuario;
15         this.accion = accion;
16         this.resultado = resultado;
17         this.detalles = detalles;
18     }
19
20     public String toString() {
21         return String.format(format:"[%s] Usuario: %s - Acción: %s - Resultado: %s - Detalles
22         resultado, detalles);
23     }
24 }
25
```

- **Controladores**

#### *ClienteController*

Gestiona la creación y consulta de clientes del banco.

Métodos principales:

crearCliente(Cliente cliente): Registra un nuevo cliente.

crearCliente(String cui, String nombre, String apellido): Sobrecarga del método anterior.

existeCUI(String cui): Verifica si un CUI ya está registrado.

getClienteporCui(String cui): Busca un cliente por su CUI.

#### *TransaccionController*

Administra depósitos, retiros y consulta de transacciones.

Límite de transacciones: 25 por cuenta

Métodos principales:

registrarDeposito(Cuenta cuenta, double monto): Registra un depósito.

registrarRetiro(Cuenta cuenta, double monto): Registra un retiro.

getTransaccionesPorCuenta(Cuenta cuenta): Obtiene las transacciones de una cuenta.

limiteTransaccionesAlcanzado(Cuenta cuenta): Verifica si una cuenta alcanzó su límite.

## Flujo de Datos

- **Registro de Clientes**

1. La vista solicita datos del cliente,
2. El controlador válida:
  - CUI no está duplicado.
  - Límite de clientes no alcanzado.
3. Si las validaciones son exitosas, el cliente es registrado

- **Depósito**

1. La vista solicita datos: cuenta y monto.
2. El controlador válida:
  - Monto positivo.
  - Límite de transacciones no alcanzado.
3. Se actualiza el saldo de la cuenta.
4. Se registra la transacción.
5. Se genera entrada en bitácora.

- **Retiro**

1. La vista solicita datos: cuenta y monto.
2. El controlador válida:
  - Monto positivo.
  - Saldo suficiente.
  - Límite de transacciones no alcanzado.
3. Se actualiza el saldo de la cuenta
4. Se registra la transacción
5. Se genera entrada en bitácora

## Restricciones del Sistema

- Límite de clientes: Configurado en la clase [ClienteController](#).
- Límite de transacciones por cuenta de 25 transacciones.
- Validaciones de montos:
  - Depósitos: monto > 0.
  - Retiros: monto > 0 y monto ≤ saldo.

## Compilación y Ejecución

### **Requisitos:**

- JDK 11 o superior
- IDE compatible con Java (Eclipse, IntelliJ IDEA, etc.)

## Compilación y Ejecución

1. Ingresar a la consola el comando de compilación:  
*javac com/usacbank/\*\*/\*.java*
2. Ingresar a la consola el comando para la ejecución:  
*java com.usacbank.Main*