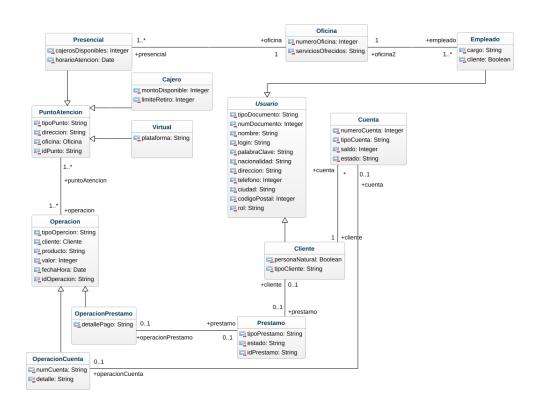
Proyecto – Entrega 3

Grupo 6

1. Documentación de las clases de Java Spring

1.1. Modelo UML



1.2 Clase Usuario:

Descripción: Clase que representa un usuario en el sistema.

Atributos:

Integer id: Identificador único del usuario (auto-generado).

String nombre: Nombre del usuario.

String login: Nombre de usuario para acceso al sistema.

String palabra_clave: Contraseña del usuario.

String nacionalidad: Nacionalidad del usuario.

String direccion: Dirección residencial del usuario.

String telefono: Teléfono de contacto del usuario.

String ciudad: Ciudad de residencia del usuario.

String codigo_postal: Código postal de la residencia del usuario.

String rol: Rol del usuario en el sistema.

Métodos:

Constructor completo para inicialización.

Constructor vacío para JPA.

Getters y setters para cada atributo.

1.3 Repositorio Usuario Repository

Descripción: Interfaz para el manejo de operaciones CRUD y consultas adicionales para usuarios.

Métodos:

Collection<Usuario> darUsuarios(): Retorna todos los usuarios.

Usuario dar Usuario (Integer id): Retorna un usuario específico por ID.

Métodos para insertar, actualizar y eliminar usuarios utilizando operaciones nativas.

1.4 Clase Cliente

Descripción: Clase que representa un cliente en el sistema.

Atributos:

Integer id: Identificador único del cliente.

String tipo_cliente: Tipo de cliente (por ejemplo: individual, corporativo).

Métodos:

Constructor completo para inicialización.

Constructor vacío para JPA.

Getters y setters para cada atributo.

1.5 Repositorio ClienteRepository

Descripción: Interfaz para el manejo de operaciones CRUD y consultas adicionales para clientes.

Métodos:

Collection<Cliente> darClientes(): Retorna todos los clientes.

Cliente darCliente(Integer id): Retorna un cliente específico por ID.

Métodos para insertar, actualizar y eliminar clientes utilizando queries nativas.

Collection<RespuestaInformacionCliente> obtenerInformacionCliente(Integer idCliente): Retorna información detallada de un cliente específico.

1.6 Clase Empleado

Descripción: Clase que representa a un empleado en el sistema.

Atributos:

Integer id: Identificador único del empleado.

String numero_oficina: Número de la oficina a la que está asignado el empleado.

Métodos:

Constructor completo para inicialización.

Constructor vacío para JPA.

Getters y setters para cada atributo.

1.7 Repositorio Empleado Repository

Descripción: Interfaz para el manejo de operaciones CRUD y consultas adicionales para empleados.

Métodos:

Collection<Empleado> darEmpleados(): Retorna todos los empleados.

Empleado dar Empleado (Integer id): Retorna un empleado específico por ID.

Métodos para insertar, actualizar y eliminar empleados utilizando queries nativas.

1.8 Clase Punto_atencion

Descripción: Representa un punto de atención al cliente en el sistema.

Atributos:

Integer id: Identificador único del punto de atención.

String tipo_punto: Tipo del punto de atención (e.g., cajero, presencial, virtual).

String direccion: Ubicación del punto de atención.

Métodos:

Constructores para inicialización.

Métodos estándar de acceso y modificación (getters y setters).

toString: Representación en cadena del objeto.

1.9 Repositorio Punto_atencionRepository

Descripción: Interfaz para operaciones de base de datos relacionadas con puntos de atención.

Operaciones:

Métodos CRUD estándar para Punto_atencion.

Operaciones específicas para insertar y eliminar diferentes tipos de puntos de atención (cajero, presencial, virtual).

1.10 Clase Presencial

Paquete: uniandes.edu.co.proyecto.modelo

Descripción: Representa un punto de atención presencial.

Atributos:

Integer id: Identificador del punto presencial.

Integer cajeros_disponibles: Número de cajeros disponibles en el punto presencial.

LocalTime horario_atencion_inicio: Hora de inicio de atención.

LocalTime horario_atencion_fin: Hora de fin de atención.

Integer numerooficina: Número de oficina asociado al punto presencial.

1.11 Repositorio Presencial Repository

Descripción: Interfaz para operaciones de base de datos relacionadas con puntos de atención presenciales.

Operaciones:

Métodos CRUD estándar para Presencial.

Métodos para insertar y actualizar detalles específicos de un punto presencial.

1. 12 Clase Cajero

Descripción: Modela un cajero automático dentro del sistema, incluyendo su ubicación, límite de retiro, y monto disponible.

Atributos:

Integer id: Identificador único del cajero.

Integer monto_disponible: Monto disponible en el cajero para retiros.

Integer limite_retiro: Límite máximo de retiro por transacción.

Integer numerooficina: Número de oficina donde está ubicado el cajero.

Métodos:

Constructor para inicialización con todos los atributos.

Constructor vacío.

Métodos getter y setter para cada atributo.

toString: Representación en forma de cadena del cajero.

1.13 Repositorio CajeroRepository

Descripción: Interfaz para manejar operaciones de base de datos para los cajeros.

Operaciones:

Métodos CRUD para Cajero.

Métodos para insertar y actualizar cajeros en la base de datos.

1.14 Clase Virtual

Descripción: Representa un punto de atención virtual, como un portal web o una aplicación móvil.

Atributos:

Integer id: Identificador único del punto virtual.

String plataforma: Plataforma del punto de atención virtual (por ejemplo, "Web", "App").

Métodos:

Constructor para inicialización con la plataforma.

Constructor vacío.

Métodos getter y setter para los atributos.

1.15 Repositorio Virtual Repository

Descripción: Interfaz para manejar operaciones de base de datos para los puntos de atención virtual.

Operaciones:

Métodos CRUD para Virtual.

Métodos para insertar, actualizar y eliminar puntos de atención virtual.

1.16 Clase Cuenta

Descripción: Representa una cuenta bancaria de un cliente dentro del sistema.

Atributos:

Integer id: Identificador único de la cuenta.

String tipo: Tipo de cuenta (por ejemplo, corriente, ahorros).

String estado: Estado actual de la cuenta (activo, inactivo).

int saldo: Saldo actual en la cuenta.

int cliente_id: Identificador del cliente propietario de la cuenta.

int oficina: Identificador de la oficina a la que está asociada la cuenta.

Métodos:

Constructores para la inicialización con y sin parámetros.

Métodos getter y setter para cada atributo.

1.17 Repositorio Cuenta Repository

Descripción: Interfaz para la gestión de las operaciones de base de datos asociadas a las cuentas bancarias.

Operaciones:

Métodos CRUD estándar para Cuenta.

Métodos para buscar cuentas por tipo, estado, y saldo.

Métodos para insertar, actualizar y eliminar cuentas en la base de datos.

1.18 Clase Oficina

Descripción: Modela una oficina bancaria, detallando los servicios que ofrece.

Atributos:

Integer numero_oficina: Número identificador de la oficina.

String servicios_ofrecidos: Descripción de los servicios que ofrece la oficina.

Métodos:

Constructores para la inicialización con y sin parámetros.

Métodos getter y setter para cada atributo.

1.19 Repositorio Oficina Repository

Descripción: Interfaz para la gestión de operaciones de base de datos relacionadas con las oficinas.

Operaciones:

Métodos CRUD estándar para Oficina.

Método para buscar oficinas por número.

Métodos para insertar, actualizar y eliminar oficinas en la base de datos.

1.20 Clase Operacion

Descripción: Representa una operación bancaria realizada por un usuario, como transferencias o pagos.

Atributos:

Integer id: Identificador único de la operación.

String tipo: Tipo de operación (por ejemplo, depósito, retiro).

Integer id_usuario: Identificador del usuario que realiza la operación.

String producto: Detalle del producto bancario asociado a la operación.

Integer valor: Monto de la operación.

String fecha_hora: Fecha y hora en la que se realiza la operación.

Métodos:

Constructores para la inicialización con y sin parámetros.

Métodos getter y setter para cada atributo.

1.21 Repositorio Operacion Repository

Paquete: uniandes.edu.co.proyecto.repositorio

Descripción: Interfaz para la gestión de operaciones de base de datos relacionadas con operaciones bancarias.

Operaciones:

Métodos CRUD estándar para Operacion.

Métodos para insertar, actualizar y eliminar operaciones en la base de datos.

1.23 Controladores

Descripción General: Controladores Spring MVC para manejar las peticiones HTTP relacionadas con todas las clases Java Spring.

Funcionalidades:

Métodos para mostrar vistas de listado, creación, edición y eliminación.

Métodos para procesar el guardado y la actualización de datos desde formularios.

Redirecciones apropiadas tras completar acciones como guardar o eliminar.

2. Ejercicio de aislamiento

2.1 READ COMMITTED

2.1.1 Registro de Operaciones

Sesion 1	Tiempo	Sesion 2
commit; Set autocommit 0; SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	t1	
UPDATE cuentas SET saldo = saldo + 1000000 WHERE ID = 101;	t2	
INSERT INTO operaciones_cuentas (id, cuenta, detalle) VALUES (33,101, 'Consignación de \$1.000.000');	t3	
UPDATE cuentas SET saldo = saldo - 50000 WHERE ID = 201;	t4	commit; SET autocommit 0; SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
INSERT INTO operaciones_cuentas (id, cuenta, detalle) VALUES (34, 201, 'Retiro de \$50.000');	t5	
	t6	UPDATE cuentas SET saldo = saldo - 30000 WHERE ID = 101;
COMMIT;	t7	,
	t8	INSERT INTO operaciones_cuentas (id, cuenta, detalle) VALUES (35, 101, 'Cuota de manejo de \$30.000');
	t9	UPDATE cuentas SET saldo = saldo + 5000 WHERE ID = 201;
	t10	INSERT INTO operaciones_cuentas (id, cuenta, detalle) VALUES (36, 201, 'Consignación de intereses de \$5.000');
SELECT * FROM cuentas WHERE ID IN (101, 201);	t11	
	t12	COMMIT;
SELECT * FROM cuentas WHERE ID IN (101, 201);	t13	SELECT * FROM cuentas WHERE ID IN (101, 201);

2.1.2 Explicación

Sesión 1	Tiempo	Sesión 2
Se inicia la sesión con un COMMIT previo para asegurarse de que no hay transacciones pendientes. Se deshabilita el autocommit y se establece el nivel de aislamiento a READCOMITED. Esto permite que las operaciones de lectura vean solo los datos confirmados hasta ese punto.	T1	DECEMBER SETACO SALO CLIPTE_D OFFICINA
Se actualiza el saldo de la cuenta con ID 101, añadiendo 1 millón. Bajo READ COMMITTED, esta fila estará bloqueada para actualizaciones hasta que se ejecute el commit, pero otras transacciones podrían ver el saldo anterior si la lectura ocurre antes del commit.	T2	
10 451 ahorros cerrada 5000 955 10 Se inserta un registro en el log de operaciones. Dado que es una inserción, no afecta a otras transacciones en READ COMMITTED. 10 10 (2000) (10 (200)) (10 (2000)	T3	
Se actualiza el saldo de la cuenta con ID 201, retirando 50,000. Otras sesiones podrán ver este cambio solo después del commit.	T4	La sesión se inicia con el mismo nivel de aislamiento que la sesión 1. La transacción puede leer datos que han sido

1 1 10 10 10 10 10 10		Confirmados antes del inicio de cada operación. ↑ □ ↑ □ ↑ □ ↑ □ ↑ □ ↑ □ ↑ □ ↑ □ ↑ □ ↑
Se inserta el registro del retiro en el log de operaciones. Se Causara Certair	T5	AL 22 JULIORISTANDISTO DE 1200-000 EN DES CONTRA COTTANDO
	T6	Se intenta descontar 30,000 de la cuenta con ID 101 por concepto de cuota de manejo. Si la sesión 1 no ha hecho commit, esta operación esperará a que la sesión 1 termine para ver los datos más recientes. En READ COMMITTED, no se retiene un bloqueo después de que la operación de lectura se completa, por lo que esta sesión no bloquearía la sesión 1.
Se hace commit de los cambios, liberando cualquier bloqueo y haciendo que las actualizaciones sean visibles para otras transacciones.	T7	
	T8	Se inserta el registro de la cuota de manejo en el log de operaciones. Puesto que es una inserción y la sesión 1 ya hizo commit, no hay conflicto.

	T	
		2 51 corriente desactivada 5000 255 2 3 101 AFC activa 972000 955 3
		4 151ahorros desactivada 1500 1005 4 5 201corriente activa 53000 1055 5
		6 251 AFC cerrada 2500 1105 6
		7 301 ahorros activa 4000 1155 7 8 351 corriente desactivada 3500 55 8
		9 401AFC activa 4500 255 9 10 451ahorros cerrada 5000 955 10
		10 451 ahorros cerrada 5000 955 10 (DED) (CENTA (DETALE 4) 1480 08 SERVICES POP SNO, UNU GESSE CHEEKS OF RECEPTOR DESACTIVADA 25 27 201 Transferencia do 2000,000 a cuenta corriente activa
		27 28 251 Depósito de 4400.000 en cuenta AFC cerrada 28 29 301 Retiro de 4150.000 de cuenta de ahorros ectiva
		29 30 351 Transferencia de 6100.000 a cuenta corriente desactivada 30 31 401 Depósito de 6600.000 en cuenta AFC activa
		31 33 101 Commigmación de 81.000.000 32 34 201 Retiro de 550.000 33 35 101 Cuota de manejo de 630.000
		34 2051 51 Abono a la cuenta 35 32 201 Consignación de 6100.000 en una cuenta corriente
	T9	Se actualiza el saldo de la
		cuenta con ID 201,
		añadiendo 5,000 de
		intereses. Como la
		actualización de la sesión 1
		ya se ha confirmado con
		commit, esta operación
		trabajará con el saldo
		actualizado.
		2 51 corriente desactivada 5000 255 2
		3 101AFC activa 972000 955 3 4 151ahorros desactivada 1500 1005 4
		5 201 corriente activa 58000 1055 5
		6 251AFC cerrada 2500 1105 6 7 301ahorros activa 4000 1155 7
		8 351 corriente desactivada 3500 55 8 9 401 AFC activa 4500 255 9
		10 451 ahorros cerrada 5000 955 10
	T10	Se inserta el registro de la
		consignación de intereses
		en el log de operaciones. Al
		igual que antes, como es
		una inserción y la sesión 1
		ya ha confirmado sus
		cambios, no hay conflicto.
		(D) (D) (CLENTA (D) DETALE (D)
		28 29 301 Retiro de s150.000 de cuenta de ahorros activa 29 30 351 Transferencia de \$100.000 a cuenta corriente desactivada 30 31 401 Depósito de \$600.000 en cuenta AFC activa
		31 33 101 Consignación de 61.000.000 32 34 201 Retiro de 650.000
		33 35 101Cuota de manejo de 430.000 34 36 201Consignación de interese de 65.000 35 2051 51Abono a la cuenta
	T4.4	35 2051 51 Abono e la cuenta 36 32 201 Consignación de 6100.000 en una cuenta corriente
Se consulta el saldo de las	T11	
cuentas afectadas para		
confirmar los cambios.		
Estos saldos ya incluyen las		
operaciones confirmadas.		
1 101AFC activa 1002000 955 3 2 201corriente activa 53000 1055 5		
* 201 COLLIGEROR SCELLAR 22000 1022 2	T12	Se hace commit en la
	112	
		sesión 2, lo que hace que
		todos los cambios sean
		definitivos y visibles para
		todas las nuevas
		operaciones en otras
		•
0	T40	transacciones.
Se seleccionan y muestran	T13	Se realiza una consulta para
las cuentas con ID 101 y		confirmar los saldos de las
201 para verificar los saldos		cuentas afectadas. Estos
después de las		saldos reflejarán todos los
transacciones.		cambios de ambas sesiones
แนกงินบบเปกิบิง.	1	

ID () TIPO		∯ SALDO	CLIENTE_ID) OFICINA	que se hayan confirma
101 AFC	activa	972000	955	3	que se nayan commu
201 corrien	te activa	58000	1055	5	hasta este punto.
					1 101 AFC activa 972000 955
					2 201 corriente activa 58000 1055

2.2 SERIALIZABLE

2.2.1 Registro de Operaciones

Sesion 1	Tiempo	Sesion 2
commit; Set autocommit 0; SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	t1	
UPDATE cuentas SET saldo = saldo + 1000000 WHERE ID = 101;	t2	
INSERT INTO operaciones_cuentas (id, cuenta, detalle) VALUES (33,101, 'Consignación de \$1.000.000');	t3	
UPDATE cuentas SET saldo = saldo - 50000 WHERE ID = 201;	t4	commit; SET autocommit 0; SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
INSERT INTO operaciones_cuentas (id, cuenta, detalle) VALUES (34, 201, 'Retiro de \$50.000');	t5	
	t6	UPDATE cuentas SET saldo = saldo - 30000 WHERE ID = 101;
COMMIT;	t7	,
	t8	INSERT INTO operaciones_cuentas (id, cuenta, detalle) VALUES (35, 101, 'Cuota de manejo de \$30.000');
	t9	UPDATE cuentas SET saldo = saldo + 5000 WHERE ID = 201;
	t10	INSERT INTO operaciones_cuentas (id, cuenta, detalle) VALUES (36, 201, 'Consignación de intereses de \$5.000');
SELECT * FROM cuentas WHERE ID IN (101, 201);	t11	
, , , , ,	t12	COMMIT;
SELECT * FROM cuentas WHERE ID IN (101, 201);	t13	SELECT * FROM cuentas WHERE ID IN (101, 201);

2.2.2 Explicación

Sesión 1	Tiempo	Sesión 2
Se inicia la sesión con un	T1	
COMMIT previo para asegurarse		
de que no hay transacciones		
pendientes. Se deshabilita el		
autocommit y se establece el		
nivel de aislamiento a		
SERIALIZABLE. Esto significa		
que cualquier lectura que		
realice esta sesión verá datos		
consistentes como si fueran		
una instantánea en el tiempo, y		
no verá cambios hechos por		
otras transacciones que se		
cometan después de este		
punto.		
0 0 0 TIPO 0 ESTADO 0 SALDO 0 CLIENTE_ID 0 OFICINA 1 1 CONTIENTE CETRADA 10000 55 1		
2 51 corriente desactivada 5000 255 2 3 101 AFC activa 2000 955 3 4 151 ahorros desactivada 1500 1005 4		
5 201 corriente activa 103000 1055 5 6 251 AFC cerrada 2500 1105 6		
7 301 ahorros activa 4000 1155 7 8 351 corriente desactivada 3500 55 8 9 401 AFC activa 4500 255 9		
050 Octobrol Detection 1		
27 25 23) Deplative de 2600,000 en unesta AFT cerrain 28 29 30) Entire de 1250,000 de cuenta de hobrero activa 29 20 31) Transferencia de 1200,000 a cuenta Corriente desactivada 30 31 40) Deplativo de 2600,000 en cuenta AFT estiva		
30 31 401 Depúsito de 6000.000 en cuenta AFC activa 31 2091 51 Abnos a la cuenta 32 32 201 Consignación de 8100.000 en una cuenta corriente		
Se actualiza el saldo de la	T2	
cuenta con ID 101, sumándole 1		
millón. En un nivel de		
aislamiento SERIALIZABLE, esta		
operación adquiere un bloqueo		
exclusivo sobre esta fila,		
previniendo que otras		
transacciones puedan leer o		
modificar esta fila hasta que se		
haga commit		
2 51 corriente desactivada 5000 255 2 3 101 AFC activa 1002000 955 3		
4 151 ahorros desactivada 1500 1005 4 5 201 corriente activa 103000 1055 5		
6 251AFC cerrada 2500 1105 6 7 301ahorros activa 4000 1155 7		
8 351 corriente desactivada 3500 55 8 9 401 AFC activa 4500 255 9		
10 451 ahorros cerrada 5000 955 10 11 601 ahorros cerrada 255 255 10		
Se inserta un registro en	Т3	
operaciones_cuentas con los		
detalles de la consignación. Se		
utiliza un ID fijo (33).		
⊕ 20 ♦ CUENTA ⊕ DETALLE 1 33 101 Consignación de 21.000.000 2 2 51 Retir de ¢200.000 de cuenta corriente desactivada		
3 101Transferencia de 01.000.000 a cuenta AFC activa 4 4 151Pago de servicios por 0150.000 desde cuenta de ahorros desactivada		
Se intenta actualizar el saldo de	T4	Paralelamente en la sesión 2,
la cuenta con ID 201, restándole	1-7	también se inicia una
50,000. Como esta es una		transacción con commit,
operación diferente y no hay		deshabilitación del autocommit
transacciones concurrentes		y establecimiento de nivel de
que la afecten, se ejecuta sin		aislamiento SERIALIZABLE.
		aistaillicittu SENIALIZADLE.
problemas.		

Se inserta otro registro en operaciones_cuentas usando operaciones_seq.NEXTVAL, lo cual es correcto para mantener la unicidad del ID.	T5	
AV 34 2013001110 GB 101.000	T6	Se actualiza el saldo de la cuenta con ID 101, restándole 30,000 por concepto de cuota de manejo. Esto debería bloquearse hasta que la sesión 1 haga commit, ya que ambas sesiones no pueden modificar la misma fila al mismo tiempo en un nivel SERIALIZABLE.
Se hace commit en la sesión 1, liberando todos los bloqueos que tenía y haciendo permanentes los cambios.	T7	
	TO.	Se inserta el registro de la operación en operaciones_cuentas. Si la sesión 1 no ha hecho commit, esta operación tendría que esperar. Pero como ya lo hiso y estamos en uns sistema SERIALIZABLE, no se podra realizar la transaccion hasta que la sesión 2 haga commit.
	Т9	Se actualiza la cuenta con ID 201, sumándole 5,000 por intereses. Dado que hay conflicto porque esta sesión no ha hecho commit, no se realizará la transacción.

	T10	Error que esparsa en la linear 19 del consendo - UNIDATE consense SET salio -
Se seleccionan y muestran las cuentas con ID 101 y 201 para verificar los saldos después de las transacciones, y se ve que no hay cambios aparte de los realizados en esta sesión. DI TPO ESTADO SALDO CLIENTE_ID OFICINA 1 101 APC activa 1002000 955 3 2 201 corriente activa 50000 1055 5	T11	transection.
	T12	Finalmente, se hace commit en la sesión 2, lo que libera cualquier bloqueo que mantuviera y hace permanentes los cambios.
Se seleccionan y muestran las cuentas con ID 101 y 201 para verificar los saldos después de las transacciones. D TIPO ESTADO SALDO CLIENTE_D OFICINA 1 101 AFC activa 1002000 955 3 2 201 corriente activa 53000 1055 5	T13	Se seleccionan y muestran las cuentas con ID 101 y 201 para verificar los saldos después de las transacciones, y solo se ven las transacciones realizadas por la sesión 1.