Instituto Politécnico Nacional
Unidad Profesional
Interdisciplinaria De Ingeniería
Campus Zacatecas

Ingeniería en Sistemas Computacionales

Práctica 5 - Uso de Agregación

Alumna: Vanessa Melenciano Llamas

Boleta: 2020670081

Profesora: Monreal Mendoza Sandra Mireya

Materia: Programación Orientada a Objetos

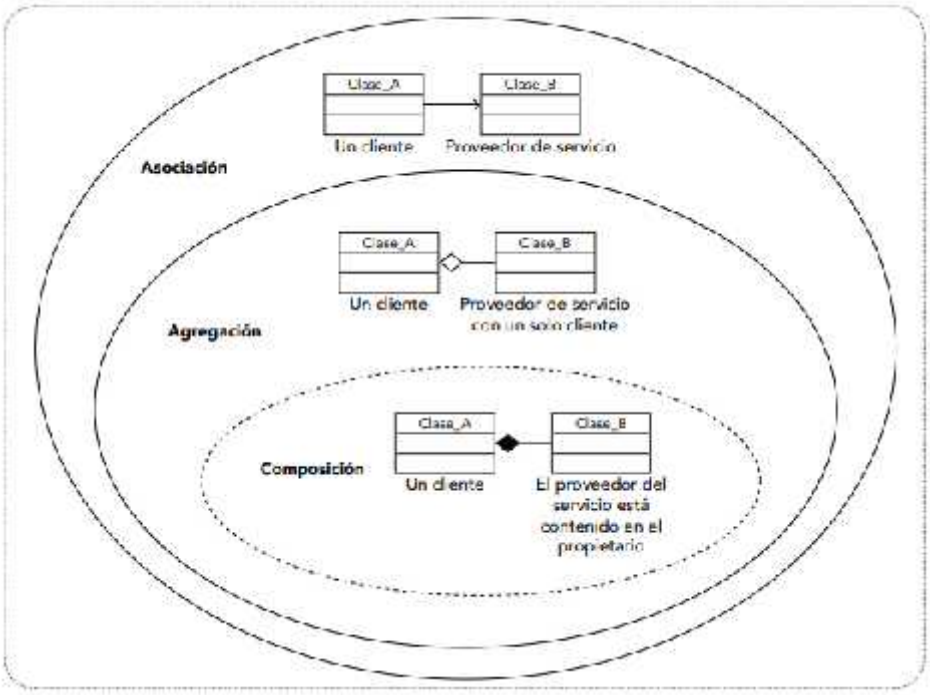
Grupo: 2CM2

Fecha: 16 de noviembre de 2020

Índice

Introducción

Se dice que un objeto de clase A está asociado con uno de clase B cuando el objeto de clase A hace uso de algún método del objeto de clase B. Existen tres niveles de asociación entre dos clases: la asociación simple, la agregación y la composición. En la figura se ilustran estos tres niveles, que se explican en cada una de las siguientes subsecciones:

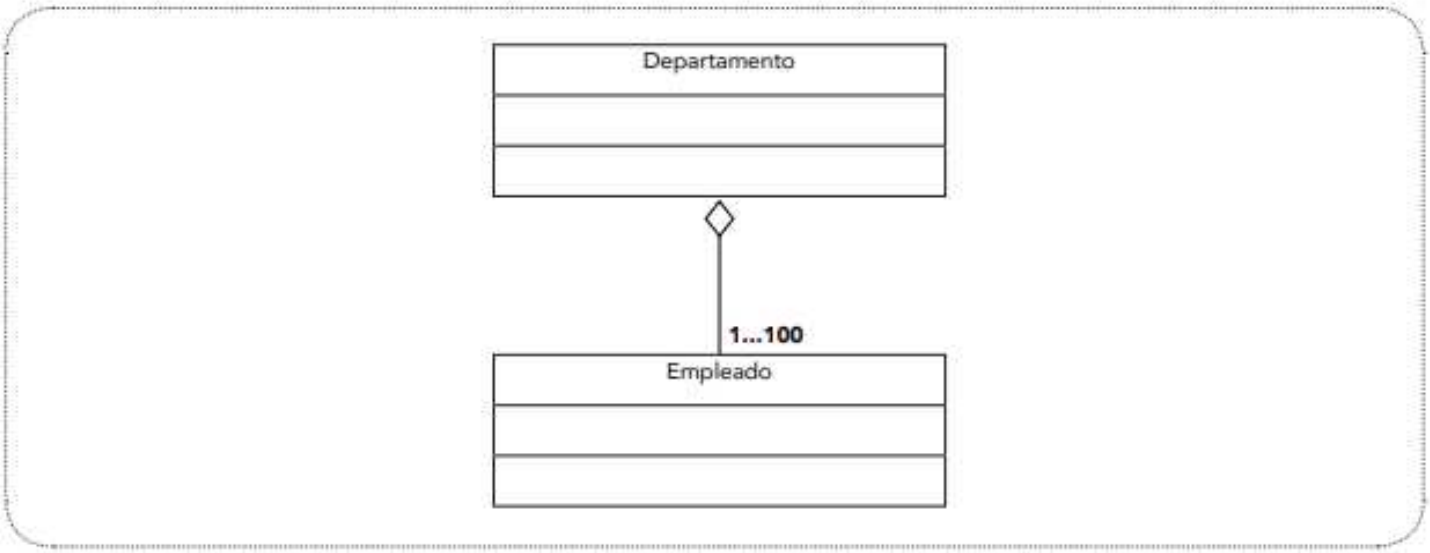


La agregación

Cuando una clase B está agregada a una clase A se entiende que uno o varios objetos de la clase B le dan servicio exclusivo a un objeto de la clase A. En este caso, cada objeto de clase B agregado a uno de clase A puede reasignarse a otro objeto también de clase A. Si el objeto de clase A desaparece, el objeto de clase B puede seguir existiendo, y éste debe agregarse a otro objeto de clase A para que su existencia tenga sentido.

La agregación de un objeto de clase B a un objeto de clase A se hace mediante un método de la clase A, el cual recibe como parámetro la referencia al objeto que se le va a agregar, que ya fue creado previamente.

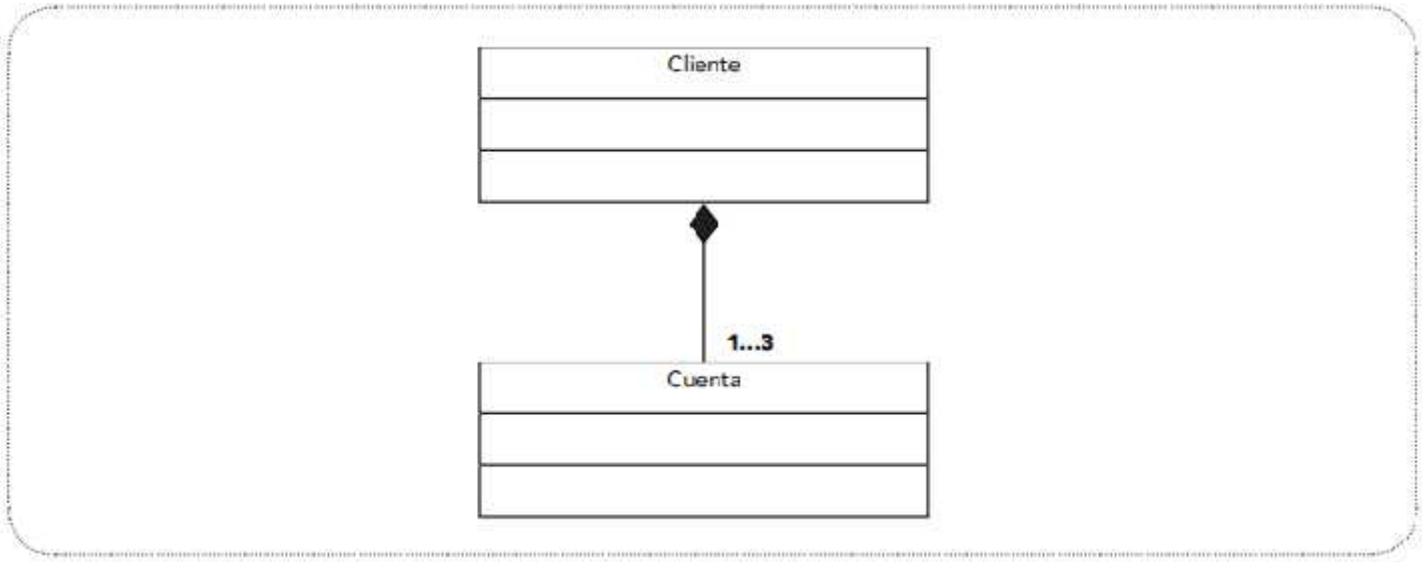
En la figura se muestra la representación en UML de la relación de agregación entre las clases



La composición

En una relación de composición entre A y B, en la que los objetos de la clase A tienen como componentes uno o más objetos de clase B. Los objetos de clase B son dependientes de la clase A ya que no pueden existir sin ser componentes de un objeto de clase A. Así, cuando desaparece el objeto de clase A, desaparecen también los objetos de clase B, porque no tiene sentido la existencia de B sin el objeto del que son componentes. (Cervantes, Del Carmen, Gonzáles, & García, 2016)

En la figura se muestra la representación en UML de la relación de composición entre las clases.

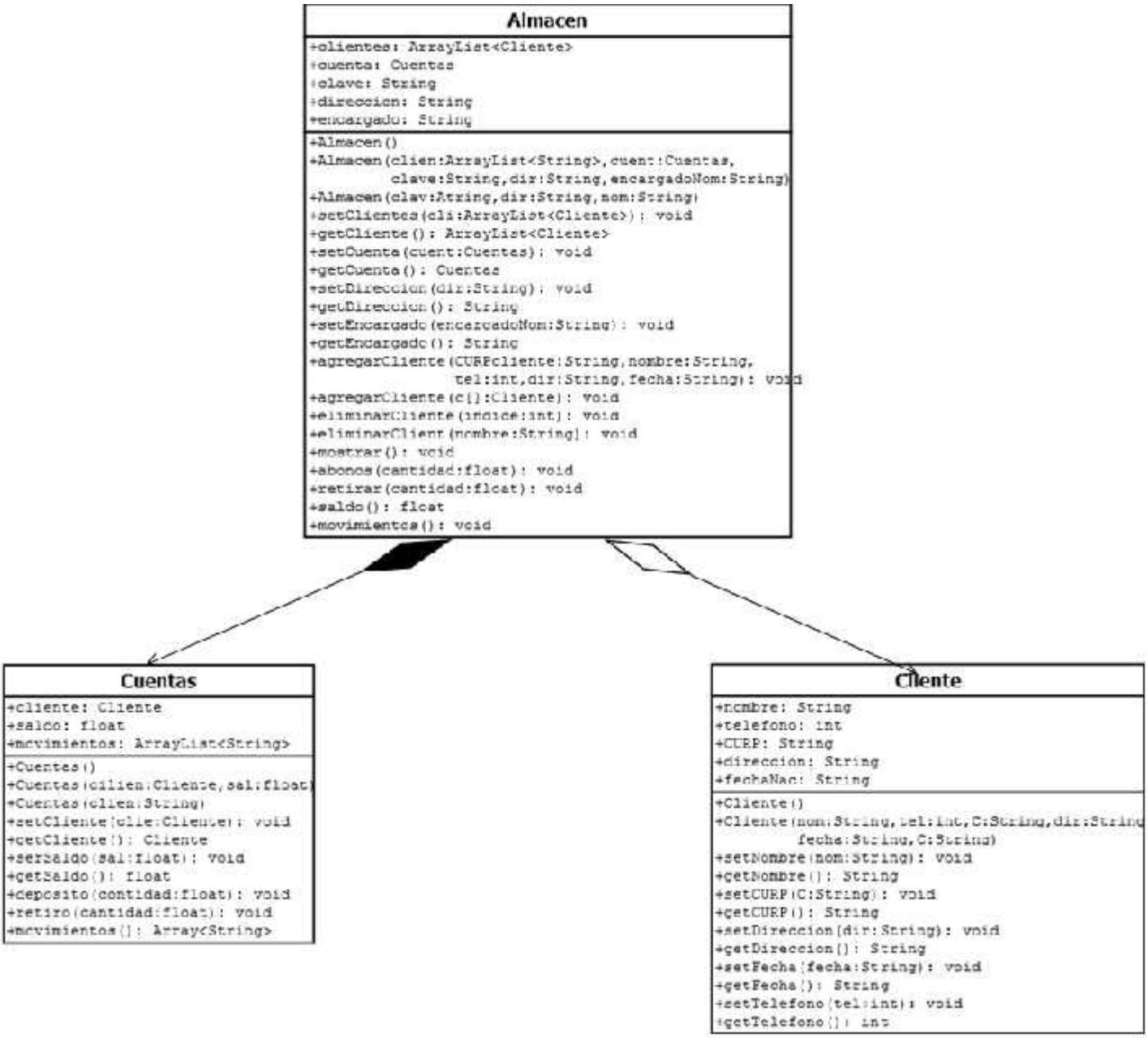


Objetivos

Realizar un programa donde se emplee la composición y la agregación como parte de las relaciones entre objetos.

Desarrollo

Diseño con diagrama ULM



Código Fuente

Para la realización de la práctica, se usaron cuatro clases:

- **Cliente**

Esta clase consiste en almacenar los datos de cada cliente, con sus respectivos getters y setters

```
package ejercicio1;

/**
 *
 * @author Vanessa
 */
public class Cliente {
    public String nombre;
    public String telefono;
    public String CURP;
    public String direccion;
    public String fechaNac;
    Cliente(){
    }
    Cliente(String nom, String tel, String dir, String fecha, String C){
        nombre = nom;
        telefono = tel;
        CURP = C;
        direccion = dir;
        fechaNac = fecha;
    }
```

```
    public void setNombre(String nom){
        nombre = nom;
    }
    public String getNombre(){
        return nombre;
    }
    public void setCURP(String C){
        CURP = C;
    }
    public String getCURP(){
        return CURP;
    }
    public void setDireccion(String dir){
        direccion = dir;
    }
    public String getDireccion(){
        return direccion;
    }
    public void setFechaNac(String fecha){
        fechaNac = fecha;
    }
    public String getFechaNac(){
        return fechaNac;
    }
    public void setTelefono(String tel){
        telefono = tel;
    }
    public void setTelefono(String tel){
        telefono = tel;
    }
    public String getTelefono(){
        return telefono;
    }

}
```

❖ Cuentas

Esta clase almacena todo lo relacionado con el saldo, retiro, deposito y movimiento de las cuentas.
La clase tiene tres atributos, los dos mencionados en la práctica, el nombre del cliente y el saldo, adicionalmente, agregue el atributo de movimientos, como un ArrayList tipo String, en el que se va almacenando los retiros y depósitos realizados, para poder mostrarlos más fácilmente al ser solicitados.
Tiene tres atributos con diferentes parámetros cada uno, tiene sus respetivos getters y setters, y por último tiene tres métodos, uno para los depósitos, donde se le suma al saldo actual el depósito, y este movimiento se guarda en el ArrayList de movimientos, después una función similar, pero en lugar de sumar, resta la cantidad al saldo actual, y por último una función que retorna los movimientos realizados.

```
package ejercicio1;
import java.util.ArrayList;

/**
 * Nobre: Vanessa Melenciano Llamas
 * Tema del programa: Relaciones entre ob
 * Descripcion: Relacion entre objetos de
 * Fecha: 03/10/20
 */
public class Cuentas {
    public String cliente;
    public float saldo;
    public ArrayList<String> movimientos;

    Cuentas(){
        movimientos = new ArrayList();
        saldo=89768;
    }
    Cuentas(String clien, float sal){
        movimientos = new ArrayList();
        cliente = clien;
        saldo=sal;
    }
    Cuentas(String clien){
        movimientos = new ArrayList();
        cliente = clien;
        saldo=56984;
    }
```

```
    public void setCliente(String clie){
        cliente = clie;
    }
    public String getCliente(){
        return cliente;
    }
    public void setSaldo(float sal){
        saldo = sal;
    }
    public float getSaldo(){
        return saldo;
    }
    public void deposito(float cantidad){
        saldo+=cantidad;
        String mov = "Deposito de " + cantidad;
        movimientos.add( e: mov);
    }
    public void retiro(float cantidad){
        saldo-=cantidad;
        String mov ="Retiro de " + cantidad;
        movimientos.add( e: mov);
    }
    public ArrayList<String> movimiento(){
        return movimientos;
    }

}
```

- **Almacén**

Esta función usa las dos anteriores. Tiene cinco atributos, tres de ellos String, la clave del almacén, dirección y encargado. Los otros atributos, son del tipo de otras clases, el primero es tipo ArrayList<Cliente>, que se usa para guardar una lista de clientes para cada almacén, el segundo atributo es de tipo Cuentas, y es para la cuenta de cada almacén.

Tiene tres constructores, cada uno con diferentes parámetros, los getter y setter de cada atributo. Los métodos siguientes son para realizar actividades en específico, una consiste en agregar clientes, de tipo Cliente al almacén, y de igual manera, el método para eliminar clientes. Después una función para mostrar la lista de clientes que tiene el almacén.

Los siguientes cuatro métodos, llaman a la clase Cuentas, el primero es para abonar a la cuenta, el siguiente para retirar, y el tercero para mostrar el saldo actual. El último método muestra los movimientos realizados.

```
package ejercicio1;
import java.util.ArrayList;

/**
 * Nobre: Vanessa Melenciano Llamas
 * Tema del programa: Relaciones entre objetos
 * Descripcion: Relacion entre objetos del mismo tipo
 * Fecha: 03/10/20
 */
public class Almacen {
    public ArrayList<Cliente> clientesList;
    public Cuentas cuenta;
    public String clave;
    public String direccion;
    public String encargado;

    Almacen(){
       clientesList = new ArrayList();
       cuenta = new Cuentas( clien: "Mariano Perea",  sal: 89278);
    }
    Almacen(ArrayList<Cliente> clien, Cuentas cuent, String clav, String dir, String nom){
        clientesList = new ArrayList();
        clientesList = clien;
        cuenta = cuent;
        clave = clav;
        direccion = dir;
        encargado = nom;
    }
    Almacen(String clav, String dir, String ncm){
        clientesList = new ArrayList();
        clave = clav;
        direccion = dir;
        encargado = nom;
        cuenta = new Cuentas( clien: "Mariano Perea",  sal: 89278);
    }

    }
```

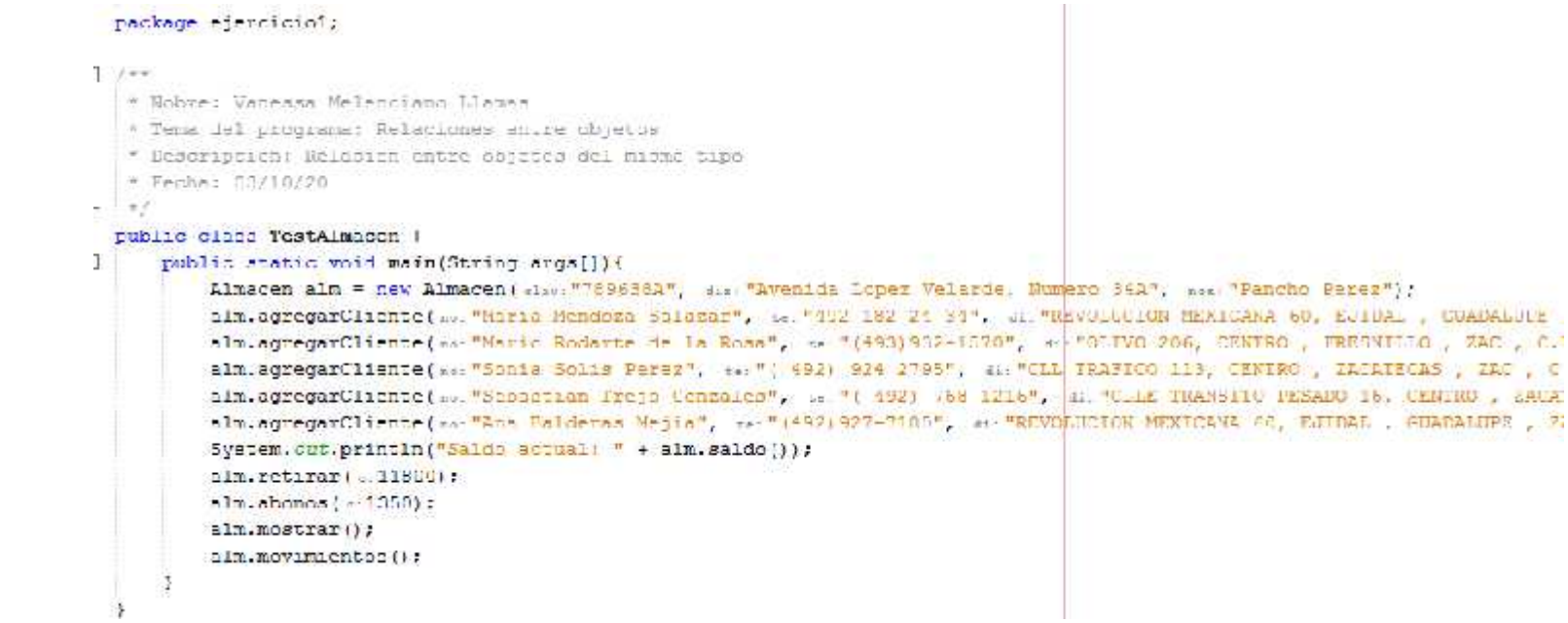
```java
public void setClientes(ArrayList<Cliente> cli){
    clientesList = cli;
}
public ArrayList<Cliente> getClientes(){
    return clientesList;
}
public void setCuenta (Cuentas cuent){
    cuenta = cuent;
}
public void setCuenta(float saldo){
    cuenta = new Cuentas( clien:"Mariano Perea",  sal:saldo);
}
public void setCuenta(){
    cuenta = new Cuentas( clien:"Mariano Perea",  sal:89278);
}
public Cuentas getCuenta(){
    return cuenta;
}
public void setDireccion (String dir){
    direccion = dir;
}
public String getDireccion(){
    return direccion;
}
public void setEncargado (String encargadoNom){
    encargado = encargadoNom;
}
public String getEncargado(){
    return encargado;
}
```

```
public void agregarCliente(String no, String te, String di, String fechaN, String CU){
    Cliente cliente = new Cliente( nombre: no,  telefono: te,  direccion: di,  fechaNacimiento: fechaN,  curp: CU);
    clientesList.add( e: cliente);
}

public void eliminarCliente(int indice){
    clientesList.remove( index: indice);

}
public void eliminarCliente(String nom, String CU){
    for(Cliente c:clientesList){
        if(c.getNombre().equals( anObject: nom) && c.getCURP().equals( anObject: CU)){
            clientesList.remove( o: c);
            break;
        }
    }
}
public void mostrar(){
    System.out.println( x: "\n");
    for(int i=0; i<clientesList.size(); i++){
       if(clientesList.get( index: i) !=null){
            System.out.println("Nombre: " + clientesList.get( index: i).getNombre() + "   CURP:  " + clientesList.get( index: i).ge
            System.out.println("      Fecha de nacimiento: " + clientesList.get( index: i).getFechaNac() + "    Telefono:  " + c
        }
    }
}
```

```
    }
    public void abonos(int c){
        cuenta.deposito( cantidad: c);
        System.out.println("\nSe ha depositado " + c + " a la cuenta");
        System.out.println("Saldo actual: " + saldo());
    }
    public void retirar(float c){
        cuenta.retiro( cantidad: c);
        System.out.println("\nSe ha retirados " + c + " de la cuenta");
        System.out.println("Saldo actual: " + saldo());
    }
    public float saldo(){
        return cuenta.getSaldo();
    }
    public void movimientos(){
        System.out.println( x: "\n ---- Movimientos ---- ");
        for(int i=0; i<cuenta.movimientos.size(); i++){
            if(cuenta.movimientos.get( index: i) !=null){
                System.out.println( x: cuenta.movimientos.get( index: i));
            }
        }
        System.out.println("Saldo actual: " + saldo());
    }
}
```

- **TestAlmacen**

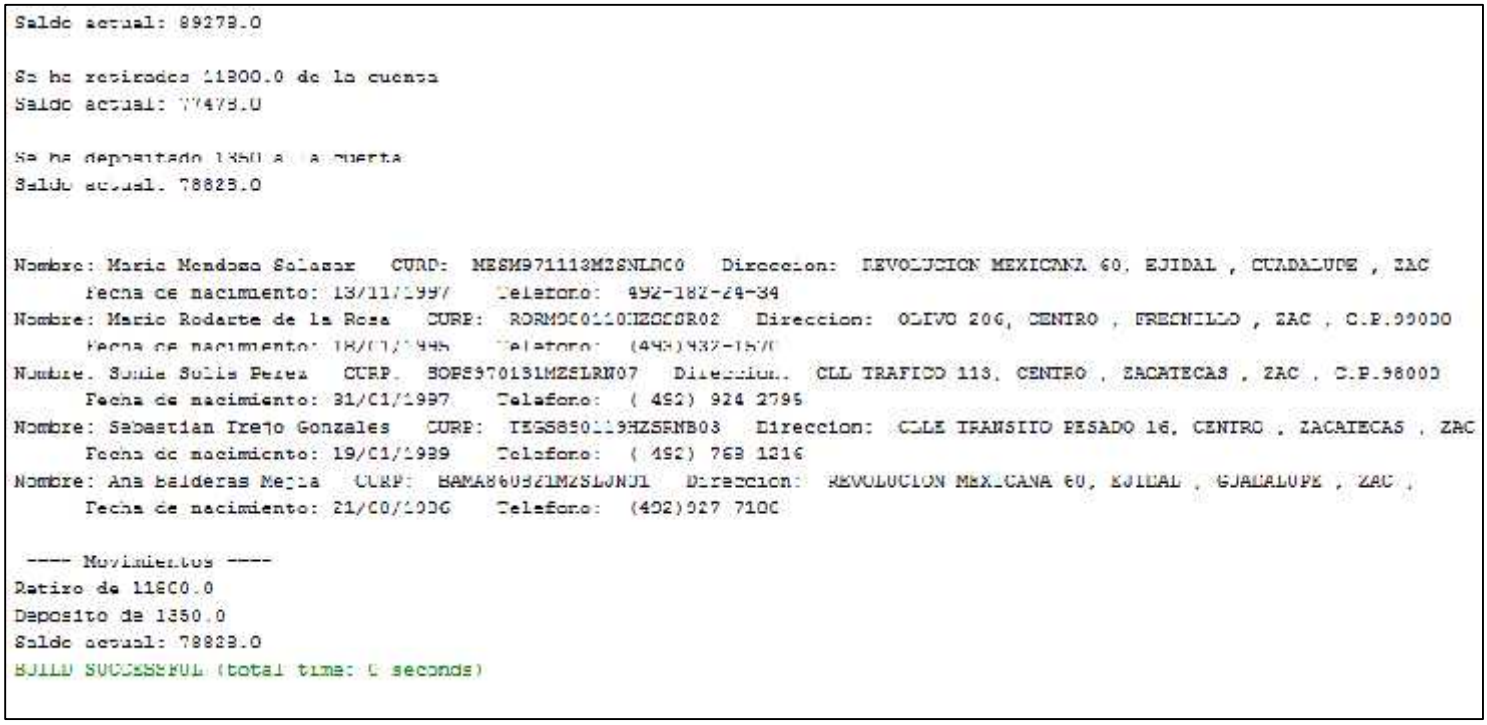
La última clase solo contiene el TestAlmacen para las impresiones en pantalla y consultas ordenadas.



```
package ejercicio1;

/**
 * Nobre: Vanessa Melanciano Llanas
 * Tema del programa: Relaciones entre objetos
 * Descripcion: Relacion entre objetos del mismo tipo
 * Fecha: 03/10/20
 */
public class TestAlmacen {
    public static void main(String args[]){
        Almacen alm = new Almacen("789638A", "Avenida Lopez Velarde, Numero 3423", "Pancho Perez");
        alm.agregarCliente("Mario Mendoza Salazar", "222 182 21 21", "REVOLUCION MEXICANA 60, EJIDAL , GUADALUPE
        alm.agregarCliente("Maric Roberto de la Rosa", "(493)932-1070", "OLIVO 206, CENTRO , TRINIDAD , ZAC , C
        alm.agregarCliente("Sonia Solis Perez", "( 492) 924 2795", "CLL TRAFICO 113, CENTRO , ZACATECAS , ZAC , C
        alm.agregarCliente("Sebastian Trejo Gonzalez", "( 492) 768 1216", "CALLE TRANSITO PESADO 16, CENTRO , ZACA
        alm.agregarCliente("Ana Calderas Mejia", "(492)927-2100", "REVOLUCION MEXICANA 60, EJIDAL , GUADALUPE , Z
        System.out.println("Saldo actual: " + alm.saldo());
        alm.retirar(11800);
        alm.abonos(1000);
        alm.mostrar();
        alm.movimientos();
    }
}
```

Funcionamiento

- Por NetBeans



La parte del main, consiste en crear un objeto tipo almacén, el cual se usa para llamar a las funciones para hacer las posteriores funciones.

Después agregar cinco objetos tipo Cliente, usando el objeto de almacén. De la cuenta, se retira una cantidad, y se muestra el saldo, posteriormente, abona a la cuenta, y de igual manera muestra el sueldo actual.

Debajo de esto, muestra la lista de los clientes, cada uno con su información correspondiente.

Por último, sale la lista de movimientos que se han hecho, junto con el saldo actual.

Errores detectados

Al realizar la práctica, lo primero que se complicó, fue comprender el contexto del problema para poder realizar lo mejor posible el ULM, y con base a este, proseguir a crear las clases en NetBeans.

Una vez hechas las clases en java, un pequeño problema que surgió, fue al no crear correctamente los atributos de una clase, que era de tipo de otra clase, al no poner su creación en todos los constructores, y dar error al compilar, pero después de comprender la causa del problema y poder atacarla, todo funcionó correctamente.

Posibles mejoras

Lo que se puede mejorar, es practicar un poco más en los problemas de este tipo para poder dominar la agregación y composición de clases.

Conclusión

La agregación es cuando un objeto es propietario o responsable de otro objeto Relación es parte de Implica que ambos objetos tienen el mismo tiempo de vida. Por otra parte, la asociación es cuando un objeto conoce otro (tiene una referencia) y el objeto puede solicitar una operación en otro objeto, pero no es responsable de él. La relación de asociación es más débil que la de agregación. La diferencia es más de intención que de implementación. Normalmente hay menos agregaciones que asociaciones, pero son más duraderas. Hay asociaciones que sólo existen durante la ejecución de un método

Se logro el objetivo de la práctica al comprender la diferencia entre la composición y la agregación, y poder llevarlo a la práctica, resolviendo el ejercicio planteado.

Referencia

Botero Tabares, R. d., Castro Castro, C. A., & Parra Castrillón, J. E. (2006). Método integrado de programación secuencial y programación orientada a objetos para el análisis diseño y elaboración de algoritmos - MIPSOO. *Revista Virtual Universidad Católica del Norte*.

Cervantes, J., Del Carmen, M., Gonzáles, P., & García, A. (2016). *Introduccion a la programacion orientada a objetos.* México: Casa abierta al tiempo.