UNIVERSIDAD DE LAS FUERZAS ARMADAS-ESPE SEDE SANTO DOMINGO
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN - DCCO-SS
CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

PERIODO : Abril 2023 – Octubre 2023

ASIGNATURA : POO

TEMA : Proyecto

NOMBRES : Ordoñez Eduardo
Kelvin Quezada
Riera Michael
Sanchez Lander

NIVEL-PARALELO : Segundo “A”

DOCENTE : Ing. Cevallos Farias Javier Moyota

FECHA DE ENTREGA : 29/05/2023

Grupo 4

SANTO DOMINGO - ECUADOR

2023

Índice

1. Introducción

En este informe, se presentará un programa desarrollado en el lenguaje de programación Java que integra todos los temas aprendidos en relación a la POO como clases, objetos, herencia, polimorfismo, encapsulación y abstracción.. Además, se incluirán los correspondientes diagramas UML, que proporcionarán una representación visual del diseño y la estructura del programa.

Además se realizó una breve consulta de los temas tratados que nos ayuda a realizar nuestro proyecto de programación Orientado a Objetos como:

Diagramas UML es un lenguaje de modelado visual general, semántica y sintácticamente rico para la arquitectura, el diseño y la utilización de la composición y la conducta de los sistemas de programa complicados.

Encapsulamiento es el proceso de guardar en una misma parte los recursos de una abstracción que componen su composición y comportamiento; se usa para dividir la interfaz contractual de una abstracción de su utilización.

Los Constructores son un factor de clase cuyo identificador corresponde a la clase en cuestión y cuyo objetivo es llevar a cabo y mantener el control de cómo se inicializan las instancias de una clase dada, debido a que Java no posibilita que las cambiantes integrante de novedosas instancias permanezcan inicializadas.

Métodos.- Un método Java es una pieza de código que hace una labor relacionada con un objeto, un procedimiento es prácticamente una funcionalidad que forma parte de un objeto o una clase.

Código Limpio.- El código limpio es un grupo de principios que ayudan a producir un código intuitivo y de forma fácil modificable.

2. Objetivos

2.1.Objetivos Generales

- Desarrollar un programa en Java que integre todos los conceptos y temas aprendidos en programación orientada a objetos, demostrando comprensión y habilidad en la implementación de estos conceptos en un proyecto práctico.

2.2.Objetivos Específicos

- Documentar de manera clara y concisa el código del programa desarrollado, incluyendo comentarios y explicaciones detalladas de su funcionamiento, para facilitar su comprensión.
- Implementar el uso de diagramas UML para mejor entendimiento del programa.
- Usar código limpio como buenas prácticas de programación

3. Marco Teórico

3.1. Programación Orientado a Objeto

Según (Martínez, 2020) la programación orientada a objetos (POO) es un paradigma de programación, es decir, un patrón o estilo de programación que nos dice cómo usarlo. Se basa en los conceptos de clases y objetos. Este tipo de programación se utiliza para estructurar el software como fragmentos simples y reutilizables de planes de código (clases) para crear instancias individuales de objetos.

3.1.2. Diagramas de UML

Mediante el señor (Mancuzo, 2021) nos dice que el lenguaje de modelado unificado (UML) se creó para proporcionar un lenguaje de modelado visual general, semántica y sintácticamente rico para la arquitectura, el diseño y la implementación de la estructura y el

comportamiento de los sistemas de software complejos. UML tiene aplicaciones más allá del desarrollo de software, p. Por ejemplo, en un flujo de proceso en la industria manufacturera.

- **Clases :** Una clase es el elemento principal de un diagrama y, como sugiere su nombre, una clase representa una clase en un paradigma orientado a objetos. Estos tipos de elementos se utilizan a menudo para representar conceptos o entidades "comerciales". Una clase define un conjunto de objetos que comparten propiedades, condiciones y significado comunes.
- **Casos de Uso:** (Cabrera, 2022) nos dice que un diagrama de casos de uso le permite visualizar las posibles interacciones que un usuario o cliente podría tener con el sistema. Sin embargo, los diagramas de casos de uso, utilizados anteriormente en la programación de computadoras, se han vuelto populares en las industrias minorista y de servicio al cliente para explicar las interacciones del cliente con una empresa o negocio.

3.1.3. Encapsulamiento

(Lara, 2015) nos da a conocer que el proceso de almacenar en una misma parte los elementos de una abstracción que conforman su estructura y comportamiento; se utiliza para separar la interfaz contractual de una abstracción de su implementación. Existen tres niveles de acceso para el encapsulamiento, los cuales son:

- público(public)
- protegido(protected)
- privado(private)

3.1.4. Constructores

Un constructor es un elemento de clase cuyo identificador corresponde a la clase en cuestión y cuyo propósito es implementar y controlar cómo se inicializan las instancias de una clase dada, ya que Java no permite que las variables miembro de nuevas instancias permanezcan inicializadas.

ilustración 1. Constructor en java

```
public Administrativos(String dni, String apellidos, String nombres,
   String sexo, int edad,String cargo ) {
   this.dni = dni;
   this.apellidos = apellidos;
   this.nombres = nombres;
   this.sexo = sexo;
   this.edad = edad;
   this.cargo=cargo;
}
```

3.1.5. Métodos

Un método Java es una pieza de código que realiza una tarea relacionada con un objeto, un método es básicamente una función que pertenece a un objeto o una clase.

Set: Un método Java es una pieza de código que realiza una tarea relacionada con un objeto, un método es básicamente una función que pertenece a un objeto o una clase.

Get: El método get es un método público al igual que una colección, pero el método get se encarga de mostrar la propiedad del objeto o el valor de la propiedad encapsulado en la clase correspondiente, es decir, declarado o protegido por la palabra reservada private.

Ilustración 2. Métodos get y set

```
//generamos los metodos get nos devuelve el valor de dni
public String getDni() {
    return dni;
}
//generamos los métodos set no devuelve nada solo establece datos
public void setDni(String dni) {
    this.dni = dni;
}
```

3.1.6. Código Limpio

Según (Paredes, 2022) el código limpio no es un conjunto rígido de reglas, sino un conjunto de principios que ayudan a crear un código intuitivo y fácilmente modificable. Intuitivo en este caso significa que cualquier desarrollador profesional puede entenderlo de inmediato. El código fácilmente personalizable tiene las siguientes características:

- La secuencia de ejecución de todo el programa es lógica y la estructura es simple.
- La relación entre las diferentes partes del código es claramente visible.
- La tarea o función de cada clase, función, método y variable se puede entender de un vistazo.

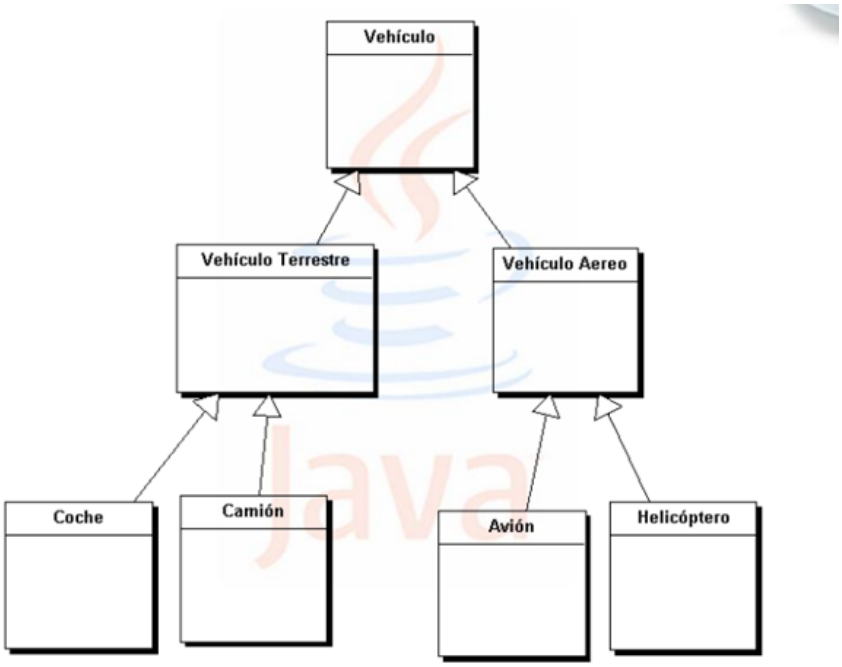
3.2. Generalización/ Especialización

Según (Vazquez) la relación de especialización/generalización (o de herencia) entre dos clases. Esta relación se considera propia de los lenguajes de POO. Una relación de herencia de la clase B (subclase, o clase hija) con respecto a (superclase o clase base) nos permite decir que la clase B obtiene todos los métodos y atributos de la clase A, y que luego puede añadir algunas características propias.

En el caso anterior se supone que utilizaremos la relación de herencia para decir que la clase B hereda de la clase A. Por medio de este mecanismo, la clase B comparte todas las

características (atributos o estado y métodos o comportamiento) de la clase A. Esto no impide que se le pueda añadir a la clase B características adicionales (de nuevo, tanto atributos como métodos), o incluso, se modifique el comportamiento (la definición, no la declaración) de alguno de los métodos heredados.

Ilustración 3. Generalización/especificación (herencia)



La Herencia

Las instancias de una clase incluyen también las de su superclase o superclases. Como resultado, además de los atributos y métodos específicos de la clase, también utilizan los definidos en la superclase.

Esta capacidad se llama herencia, lo que significa que una clase hereda las propiedades y funciones de sus superclases para que sus instancias puedan usarlas. Al colocar el símbolo

delante de los atributos y métodos heredados en las subclases, UML brinda la opción de representarlos.

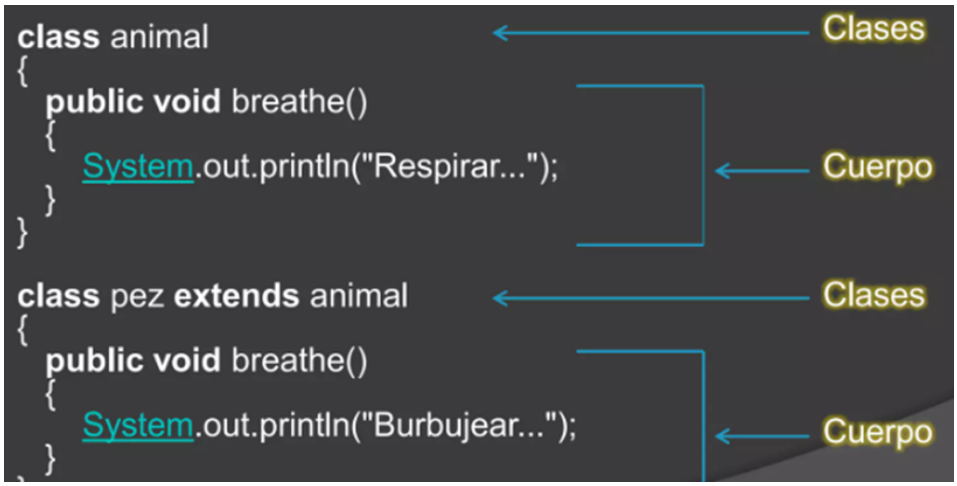
3.2.1. Implementación

Según (Solís, 2015) la programación e implementación de clases trata de amoldarse al modo de pensar del hombre y ajustarse a su forma de analizar y gestionar los problemas que tenga que desarrollar. Las clases son abstracciones que representan a un conjunto de objetos con un comportamiento e interfaz común. La implementación de una clase comprende dos componentes: la declaración y el cuerpo de la clase.

Ejemplo de sintaxis

Definidos los tipos y funciones pasamos a la declaración e implementación de clases con ello podemos identificar la estructura de la programación.

Ilustración 4. Ejemplo de sintaxis



3.3. Gestión de Defecto testing

3.3.1. Verificacion y Validacion

Los procesos de verificación y análisis que aseguran que el software que se está desarrollando está en línea con su especificación y satisface las necesidades de los clientes se denominan verificación y validación. Un proceso de ciclo de vida completo es VandV. Las revisiones de los requisitos vienen primero, luego las revisiones de los diseños y el código, y finalmente las pruebas del producto. En cada etapa del proceso de desarrollo de software, se llevan a cabo actividades de V&V. A pesar de la facilidad con la que pueden confundirse, Boehm (1979) describió sucintamente la diferencia entre verificación y validación de la siguiente manera.

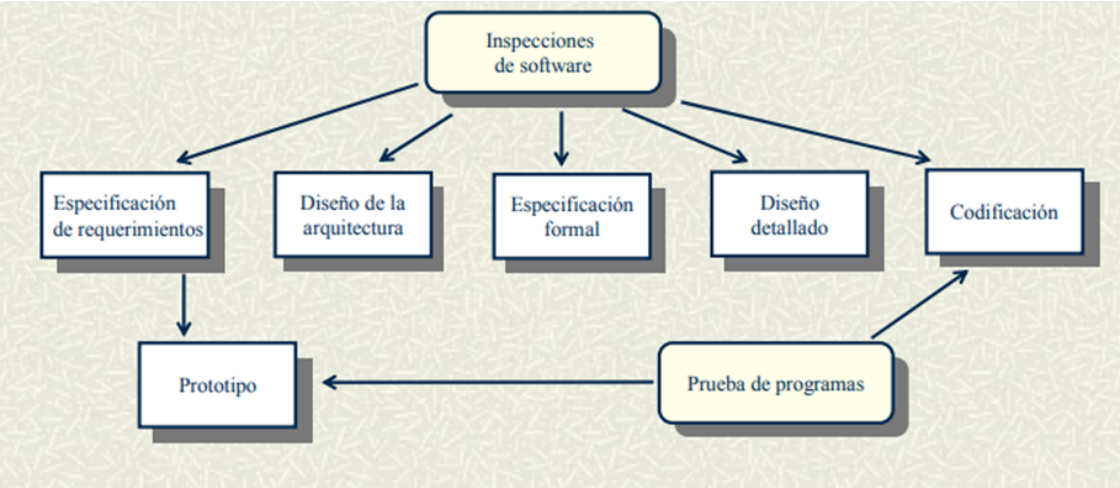
Verificación: las responsabilidades de Verificación incluyen asegurarse de que el software cumpla con sus especificaciones. Se confirma el cumplimiento del sistema con los requisitos funcionales y no funcionales establecidos.

La validación es un proceso que se usa más ampliamente: El software necesita ser revisado para ver si cumple con las expectativas del cliente. El software se prueba para ver si funciona como el usuario espera en lugar de lo especificado, lo que va más allá de determinar si el sistema cumple con sus especificaciones.

Primero, es crucial validar los requisitos del sistema. Es sencillo cometer errores y omisiones durante la fase de análisis de requisitos del sistema y, en tales casos, el software terminado no estará a la altura de las expectativas del cliente. Sin embargo, en realidad, no todos los problemas que presenta una aplicación no se pueden encontrar a través de la validación de

requisitos. Cuando el sistema se haya implementado por completo, es posible que se encuentren algunos errores en los requisitos. (Drake, 2009)

Ilustración 5. Verificación y validación



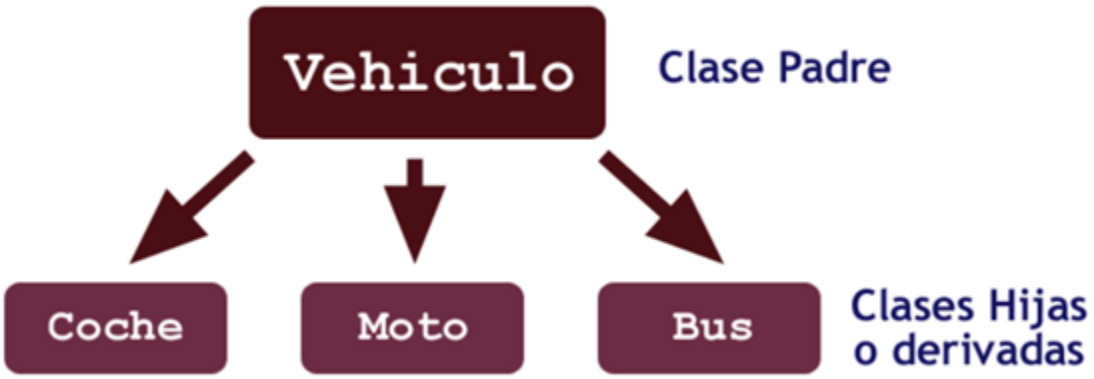
3.3.2. Polimorfismo

Según (Fernandez, s.f.) al crear objetos con comportamientos compartidos, el polimorfismo nos permite procesar objetos de diversas formas. Implica tener la capacidad de mostrar la misma interfaz de usuario para varios formularios o tipos de datos subyacentes. Los objetos pueden reemplazar comportamientos primarios comunes con comportamientos secundarios particulares mediante el uso de la herencia. La sobrecarga de métodos y la anulación de métodos son dos formas en que el polimorfismo permite que el mismo método lleve a cabo varios comportamientos.

Las clases con tipos compatibles se pueden usar en cualquier parte de nuestro código gracias al polimorfismo. La compatibilidad de tipos en Java se refiere a cómo una clase se extiende a otra o cómo una clase implementa una interfaz. Informalmente: Podemos enlazar

referencias de sus hijas a una referencia de tipo padre. Cualquier instancia de una clase que implemente la interfaz se puede conectar a una referencia de tipo de interfaz.

Ilustración 6. Polimorfismo, herencia



3.4. Modelo Vista controlador

3.4.1. El Modelo

El señor (Pantoja, 2004) el modelo es un conjunto de clases que representa la información del mundo real que debe procesar el sistema. Sin tener en cuenta los métodos por los cuales se generarán los datos o cómo se mostrarán, es decir, sin tener una conexión con otra entidad dentro de la aplicación. Las vistas y un controlador están ocultos para el modelo, que no los conoce. SmallTalk sugiere que el modelo en realidad consta de dos submódulos: el modelo de dominio y el modelo de red. Si bien ese enfoque suena intrigante, no es práctico porque debe haber interfaces que permitan que los módulos se comuniquen entre sí. solicitud.

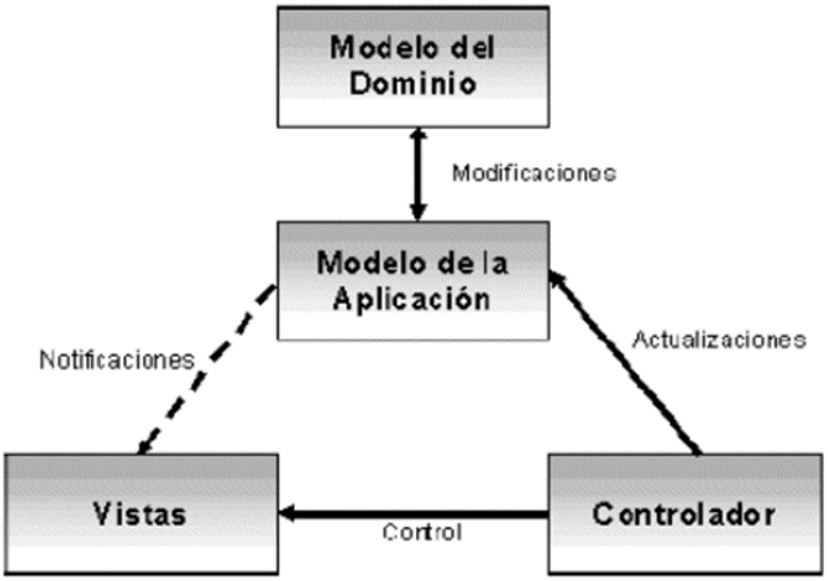
3.4.2. La Vista

Las vistas se encargan de mostrar al usuario los datos del modelo. Las vistas y el modelo tienen una relación de muchos a uno, lo que significa que cada vista tiene un modelo asociado, aunque puede haber varias vistas vinculadas a un solo modelo. De esta forma, por ejemplo, podría tener una vista que muestre la hora del sistema como un reloj analógico y otra vista que muestre la misma información como un reloj digital. La vista solo requiere los datos necesarios del modelo para ejecutar una visualización. La vista se altera a través de notificaciones que produce el modelo de aplicación cada vez que se realiza una acción que implica un cambio en el modelo de dominio. En pocas palabras, cuando se produce una modificación, la representación visual del modelo vuelve a dibujar los componentes necesarios.

3.4.3. El Controlador

Mediante el señor (Hernandez, 2021) la responsabilidad del controlador es extraer, modificar y proporcionar datos al usuario. Esencialmente, el controlador es el enlace entre y el modelo a través de las funciones getter y setter, el controlador extrae datos del modelo e inicializa las vistas si hay alguna actualización desde las vistas, modifica los datos con una función setter.

Ilustración 7. MVC



3.5. Desarrollo

3.5.1. Reglas del Ejercicio

Municipalidad

Área administrativa: El administrativo realiza tareas administrativas y de oficina de acuerdo con los procedimientos establecidos por cada organización. Los administrativos gestionan, organizan, planifican, atienden y realizan tareas administrativas, de soporte y apoyo a la organización

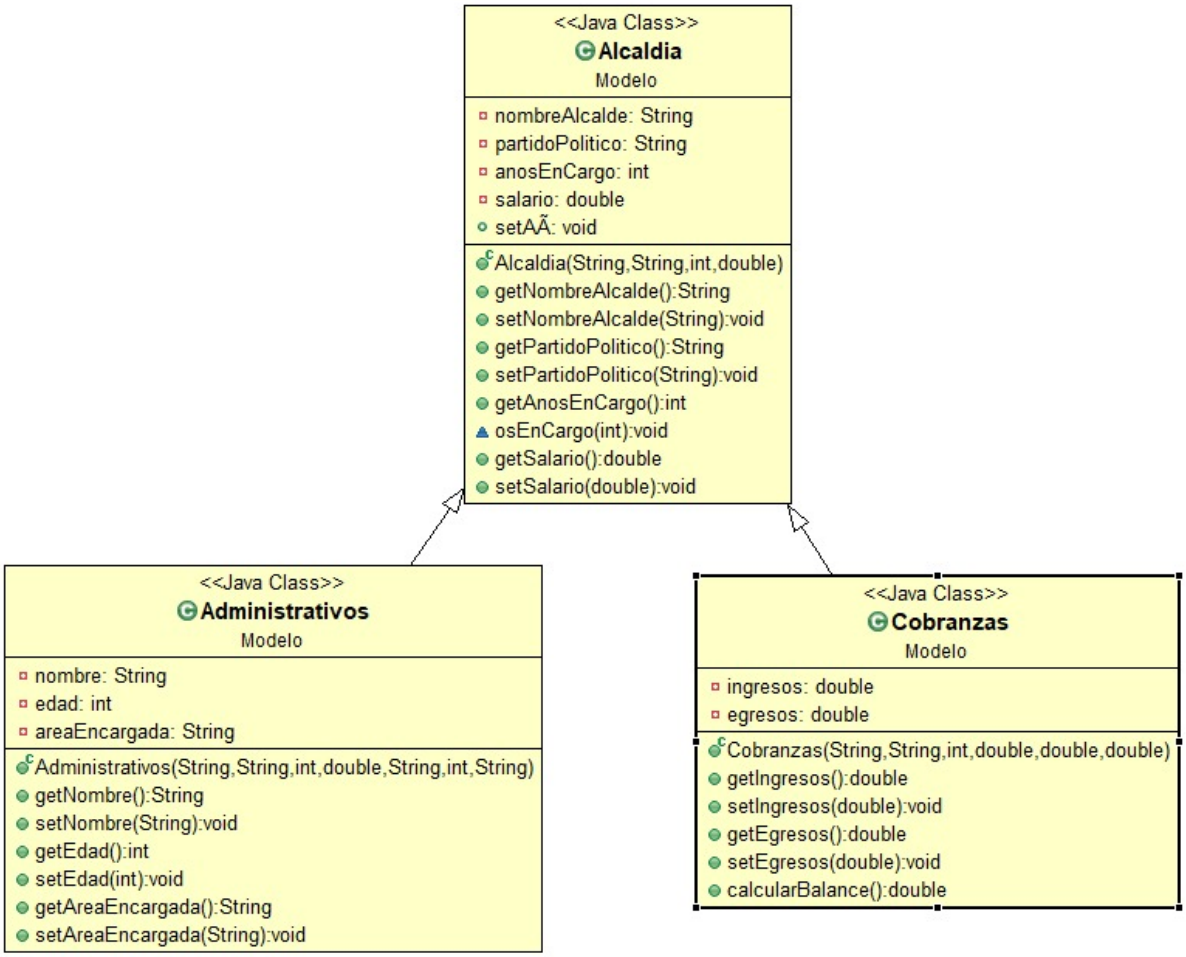
Alcaldía: El alcalde dirige la administración municipal, puede reorganizar los gastos de los servicios ordinarios del municipio, decide en qué gastar el dinero que sobra para cubrir los gastos ordinarios, propone cambios en la ley local, preside y representa al ayuntamiento

Cobranzas: Supervisar las cuentas para identificar las deudas pendientes. Investigar datos históricos de cada deuda o factura, localizar y ponerse en contacto con los clientes para

preguntarles por los pagos vencidos, Tomar medidas para propiciar el pago oportuno de la deuda, Tramitar los pagos y reembolsos.

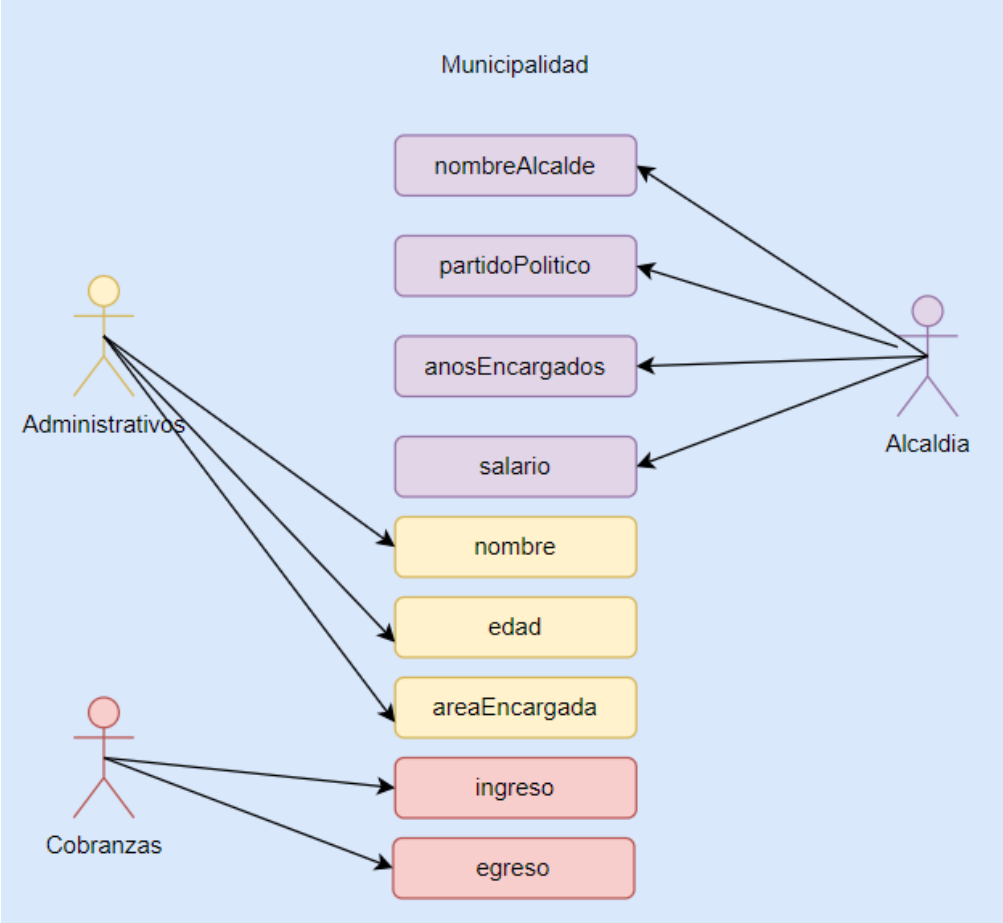
3.6. Anexos.

3.6.1. Diagrama UML



3.6.2. Diagrama de Caso de Uso

Mediante este diagrama de caso uso de la municipalidad daremos a conocer unas áreas que están relacionados que son la alcaldía, administrativos y cobranzas en la cual se va desarrollar ejercicios aplicando herencia, encapsulamiento, constructores, métodos, arreglos en la cual nos ayudará mucho en nuestro ejercicio.



3.6.3. Cuadro estadístico

Alcaldía	
nombreAlcaldia	Wilson Herazo
partidoPolitico	Lista 100

anoEncargado	2018
salario	2000

Administrativos	
nombre	Daniel
edad	33
areaEncargada	Cajero

Cobranzas	
Ingreso	500
Egreso	350
Total	150

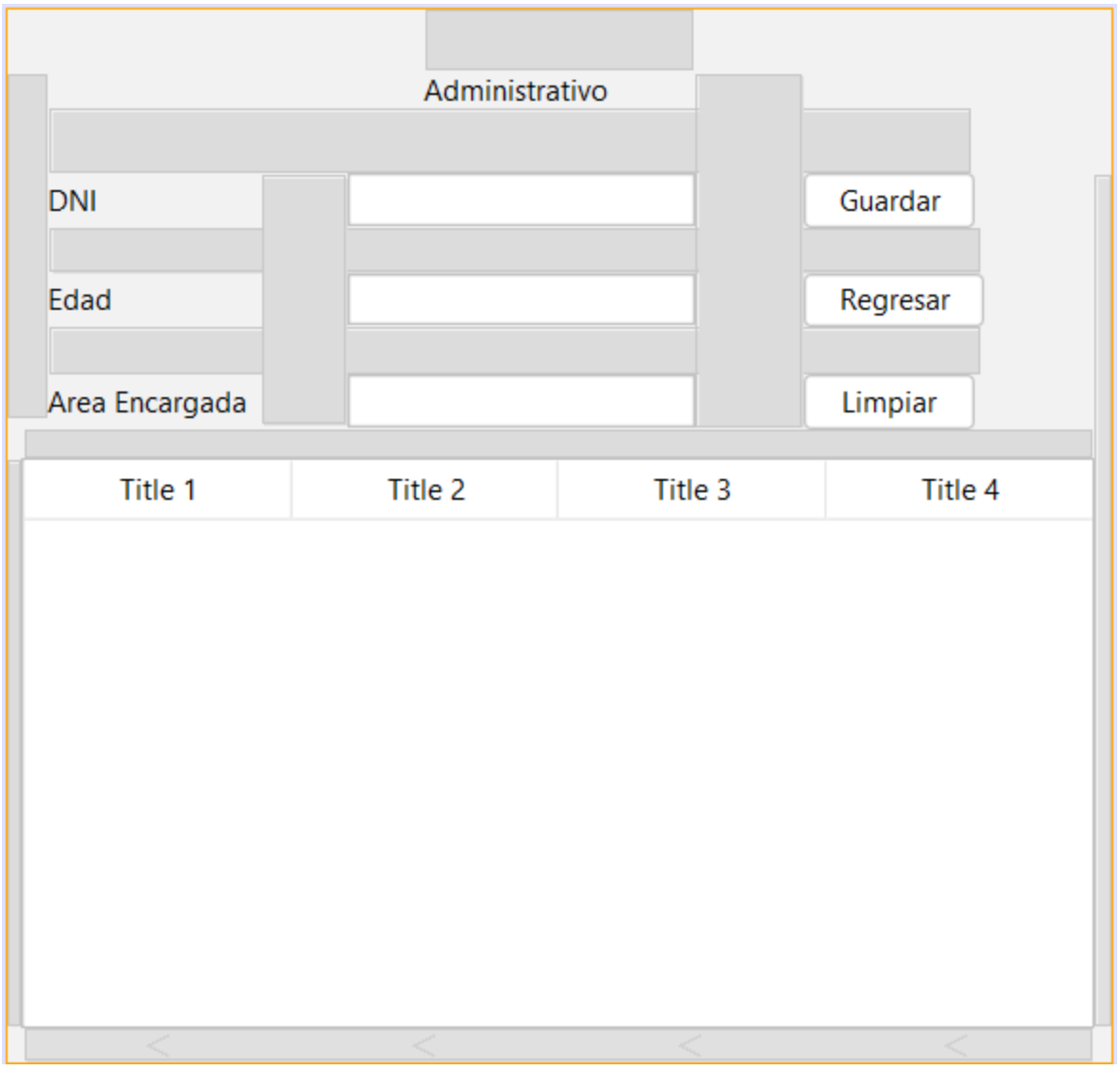
3.6.4. Capture del programa

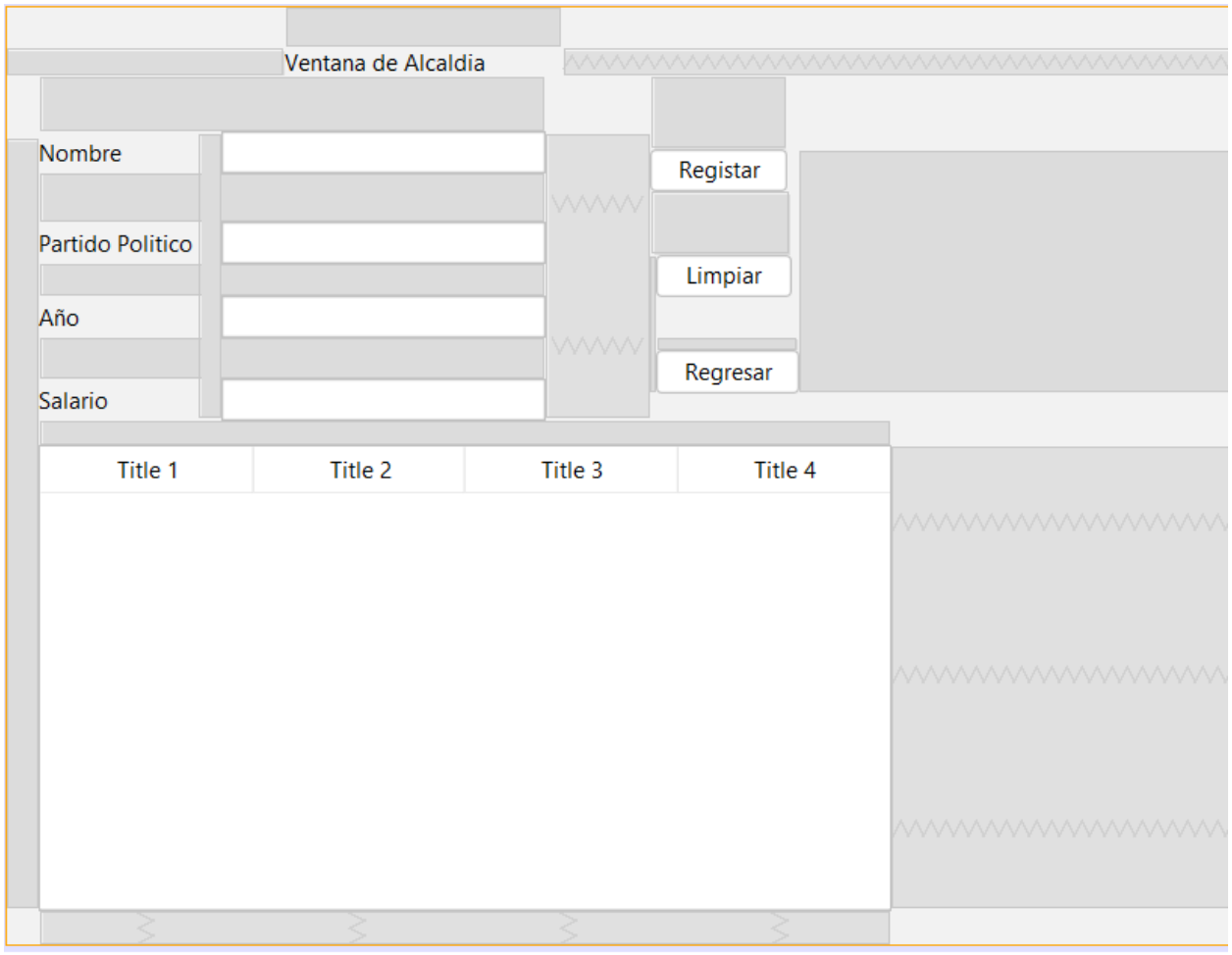
```java
package Modelo;
public class Administrativos extends Alcaldia {
    private String nombre;
    private int edad;
    private String areaEncargada;
    public Administrativos(String nombreAlcalde, String partidoPolitico,
            int anosEnCargo, double salario, String nombre, int edad, String areaEncargada) {
        super(nombreAlcalde, partidoPolitico, anosEnCargo, salario);
        this.nombre = nombre;
        this.edad = edad;
        this.areaEncargada = areaEncargada;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public String getAreaEncargada() {
        return areaEncargada;
    }

    public void setAreaEncargada(String areaEncargada) {
        this.areaEncargada = areaEncargada;
    }
}
```

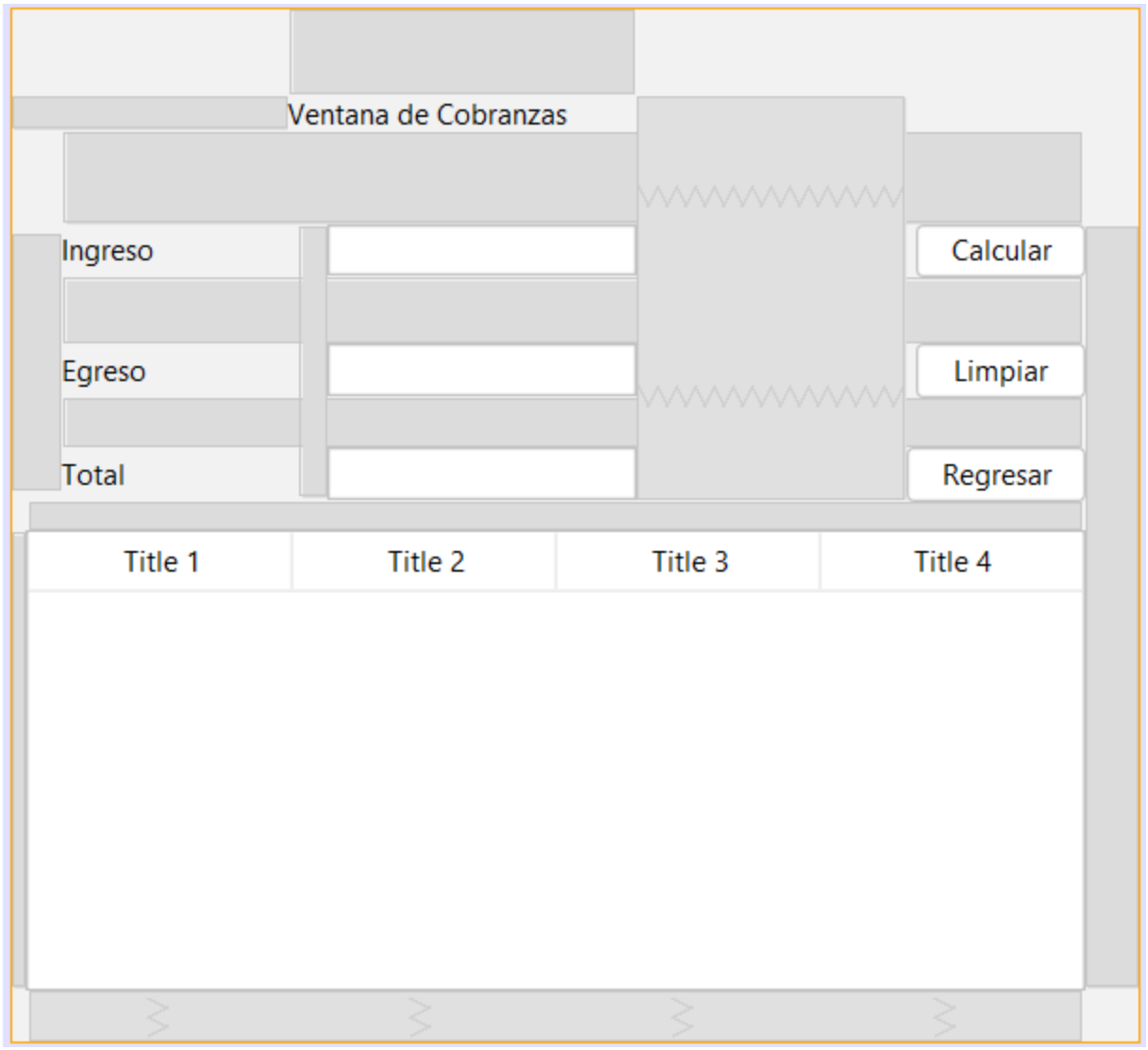
```java
package Modelo;
public class Alcaldia {
 private String nombreAlcalde;
    private String partidoPolitico;
    private int anosEnCargo;
    private double salario;
    public Alcaldia(String nombreAlcalde, String partidoPolitico, int anosEnCargo, double salario) {
        this.nombreAlcalde = nombreAlcalde;
        this.partidoPolitico = partidoPolitico;
        this.anosEnCargo = anosEnCargo;
        this.salario = salario;
    }
    public String getNombreAlcalde() {
        return nombreAlcalde;
    }
    public void setNombreAlcalde(String nombreAlcalde) {
        this.nombreAlcalde = nombreAlcalde;
    }
    public String getPartidoPolitico() {
        return partidoPolitico;
    }
    public void setPartidoPolitico(String partidoPolitico) {
        this.partidoPolitico = partidoPolitico;
    }
    public int getAnosEnCargo() {
        return anosEnCargo;
    }
    public void setAñosEnCargo(int añosEnCargo) {
        this.anosEnCargo = añosEnCargo;
    }
    public double getSalario() {
        return salario;
    }
    public void setSalario(double salario) {
        this.salario = salario;
    }
}
```

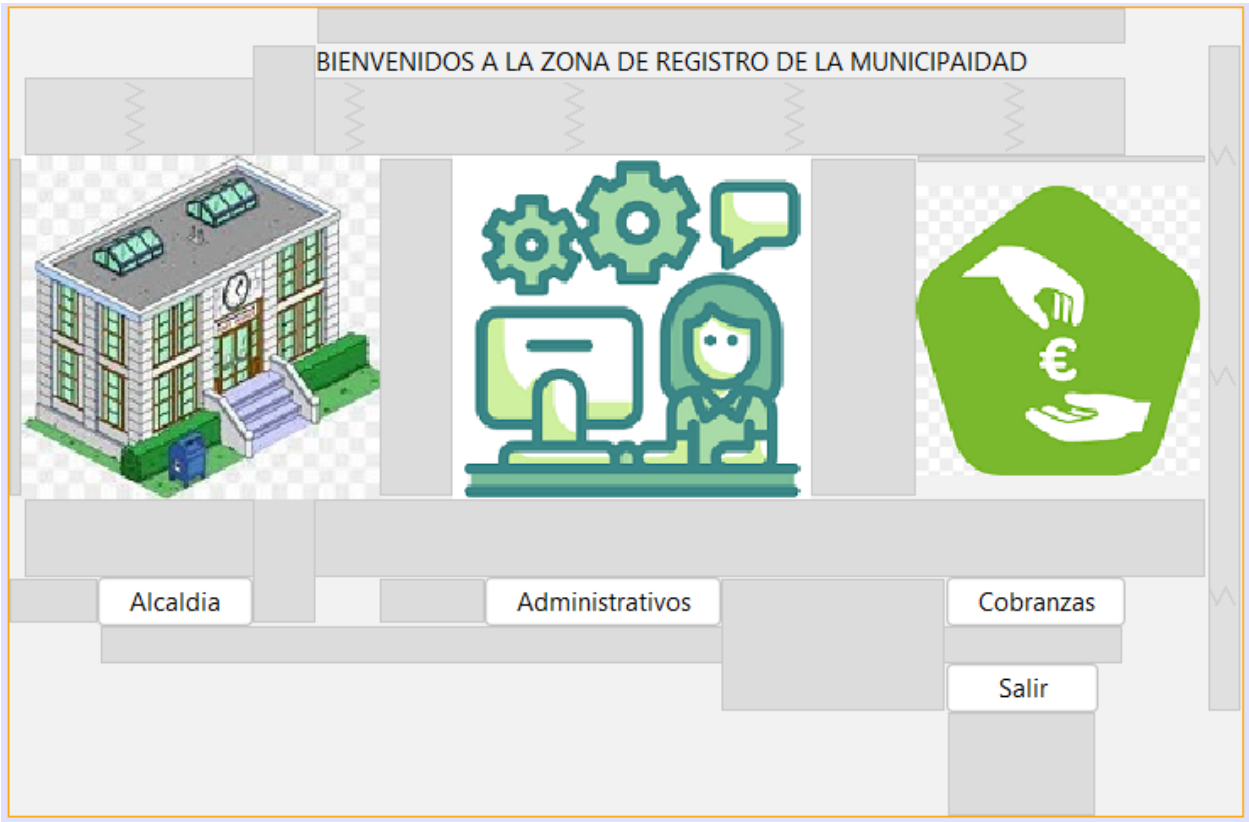
```java
package Modelo;
import java.util.Scanner;

public class Cobranzas extends Alcaldia {
    private double ingresos;
    private double egresos;

    public Cobranzas(String nombreAlcalde, String partidoPolitico, int
            anosEnCargo, double salario, double ingresos, double egresos) {
        super(nombreAlcalde, partidoPolitico, anosEnCargo, salario);
        this.ingresos = ingresos;
        this.egresos = egresos;
    }

    public double getIngresos() {
        return ingresos;
    }

    public void setIngresos(double ingresos) {
        this.ingresos = ingresos;
    }

    public double getEgresos() {
        return egresos;
    }

    public void setEgresos(double egresos) {
        this.egresos = egresos;
    }

    public double calcularBalance() {
        return ingresos - egresos;
    }
}

```


USUARIO DE LA MUNICIPALIDAD
Ingresar


Administrativo
DNI
Edad
Area Encargada
Guardar
Regresar
Limpiar
Title 1
Title 2
Title 3
Title 4


Ventana de Alcaldia
Nombre
Partido Politico
Año
Salario
Registar
Limpiar
Regresar
Title 1
Title 2
Title 3
Title 4


Ventana de Cobranzas
Ingreso
Egreso
Total
Calcular
Limpiar
Regresar
Title 1
Title 2
Title 3
Title 4




```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this
 */
package controlador;

import Vista.VistaAdministrativo;
import Vista.VistaAlcaldia;

/**
 *
 * @author antho
 */
public class AdministrativoControlador {
    public static VistaAdministrativo ventana=new VistaAdministrativo();
    public static void mostrar(){ventana.setVisible(b: true);}
    public static void ocultar(){ ventana.setVisible(b: false);}

}

```

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt t
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit thi
 */
package controlador;

import Vista.VistaAlcaldia;


/**
 *
 * @author antho
 */
public class AlcaldeControlador {

    private VistaAlcaldia objV;
    private AlcaldeControlador objC;


    public static VistaAlcaldia ventana=new VistaAlcaldia();
    public static void mostrar(){ventana.setVisible(b: true);}
    public static void ocultar(){ ventana.setVisible(b: false);}

}

```

```java
import Vista.VistaCobranzas;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JOptionPane;

public class CobranzasControlador  implements ActionListener{
    public static VistaCobranzas ventana=new VistaCobranzas();
    public static void mostrar(){ventana.setVisible(b: true);}
    private Cobranzas objeModelo;
    private VistaCobranzas objCobranzas;
    public CobranzasControlador() {
        this.objeModelo = objeModelo;
        this.objCobranzas = objCobranzas;
        objCobranzas.getBtnCalcular().addActionListener(l: this);
    }
    public void iniciar(){
       objCobranzas.setTitle(title: "USTED VA A SUMAR POR MEDELO DE VISTA CONTROLADOR");
       objCobranzas.setLocationRelativeTo(c: null);
       objCobranzas.getTxtC1().setText(t: String.valueOf(d: objeModelo.getIngresos()));
       objCobranzas.getTxtC2().setText(t: String.valueOf(d: objeModelo.getEgresos()));
       objCobranzas.getTxtC3().setText(t: String.valueOf(d: objeModelo.calcularBalance()));
    }
    @Override
    public void actionPerformed(ActionEvent e) {
    //codigo de la ejecucion
    objeModelo.setIngresos(ingresos: Integer.valueOf(s: objCobranzas.getTxtC1().getText()));
    objeModelo.setEgresos(egresos: Integer.valueOf(s: objCobranzas.getTxtC2().getText()));

    if (e.getSource()==objCobranzas.getBtnCalcular()){
        objeModelo.calcularBalance();
    }
    objCobranzas.getTxtC3().setText(t: String.valueOf(d: objeModelo.calcularBalance()));
    }
}

```

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this
 */
package controlador;

import Vista.VentanaLogin;
import Vista.VistaPrincipal;
import static controlador.PrincipalControlador.ventana;



public class LoginControlador {
    public static VentanaLogin ventana=new VentanaLogin();
    public static void mostrar(){ventana.setVisible(b: true);}
    public static void ocultar(){ ventana.setVisible(b: false);}

    public static void eventoBoton1(){
        String usuario=ventana.getjTextField1().getText();
        String clave=ventana.getjTextField2().getText();

        if (usuario.contains(s: "kelvin")&& clave.equals(anObject: "123")){
            ocultar();
            PrincipalControlador.mostrar();
        }else{
            System.out.println(x: "usuario o clave incorrecta");
        }
    }
}

```

```java
package controlador;
import Modelo.Administrativos;
import Modelo.Cobranzas;
import java.util.Scanner;
public class Principal {
    public static void main(String[] args) {
        LoginControlador.mostrar();
        Scanner leer = new Scanner(source:System.in);
        System.out.println(x: "Ingrese los datos del Alcalde:");
        System.out.print(s: "Nombre: ");
        String nombreAlcalde = leer.nextLine();
        System.out.print(s: "Partido Politico: ");
        String partidoPolitico = leer.nextLine();
        System.out.print(s: "Años en Cargo: ");
        int anosEnCargo = leer.nextInt();
        System.out.print(s: "Salario: ");
        double salario = leer.nextDouble();
        leer.nextLine(); // Consumir el salto de línea después de leer el double
        Cobranzas cobranzas;
        System.out.println(x: "\nIngrese los datos de Cobranzas:");
        System.out.print(s: "Ingresos: ");
        double ingresos = leer.nextDouble();
        System.out.print(s: "Egresos: ");
        double egresos = leer.nextDouble();
        leer.nextLine(); // Consumir el salto de línea después de leer el double
        cobranzas = new Cobranzas(nombreAlcalde, partidoPolitico, anosEnCargo, salario, ingresos, egresos);
        System.out.println(x: "\nIngrese los datos del Area Administrativa:");
        System.out.print(s: "Nombre del Encargado: ");
        String nombreEncargado = leer.nextLine();
        System.out.print(s: "Edad del Encargado: ");
        int edadEncargado = leer.nextInt();
        leer.nextLine(); // Consumir el salto de línea después de leer el int
        System.out.print(s: "Area Encargada: ");
        String areaEncargada = leer.nextLine();
        Administrativos administrativo = new Administrativos(nombreAlcalde, partidoPolitico, anosEnCargo, salario, nombre:nombreEncargado, edad: edadEncargado, areaEncargada);

```

```
37            leer.close(); // Cerrar el Scanner
38
39            // Mostramos los datos ingresados
40            System.out.println(x: "\nDatos del Alcalde:");
41            System.out.println("Nombre: " + cobranzas.getNombreAlcalde());
42            System.out.println("Partido Político: " + cobranzas.getPartidoPolitico());
43            System.out.println("Años en Cargo: " + cobranzas.getAnosEnCargo());
44            System.out.println("Salario: " + cobranzas.getSalario());
45
46            System.out.println(x: "\nRegistro de Cobranzas:");
47            System.out.println("Ingresos: " + cobranzas.getIngresos());
48            System.out.println("Egresos: " + cobranzas.getEgresos());
49            System.out.println("Balance: " + cobranzas.calcularBalance());
50
51            System.out.println(x: "\nDatos del Area Administrativa:");
52            System.out.println("Nombre: " + administrativo.getNombre());
53            System.out.println("Edad: " + administrativo.getEdad());
54            System.out.println("Area Encargada: " + administrativo.getAreaEncargada());
55        }
56    }
57
```

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt t
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit thi
 */
package controlador;

import Vista.VistaPrincipal;

public class PrincipalControlador {
    public static VistaPrincipal ventana=new VistaPrincipal();
    public static void mostrar(){ventana.setVisible(b: true);}
    public static void ocultar(){ ventana.setVisible(b: false);}

    public static void botonoSalir(){
        ocultar();
        LoginControlador.mostrar();
    }

    public static void botonoAlcaldia(){
        ocultar();
        AlcaldeControlador.mostrar();
    }

    public static void botonoCobranzas(){
        ocultar();
        CobranzasControlador.mostrar();
    }

    public static void botonoAdministrativos(){
        ocultar();
        CobranzasControlador.mostrar();
    }
}

```

3.6.5. Código del Programa

Modelo
package Modelo; public class Alcaldia { private String nombreAlcalde; private String partidoPolitico; private int anosEnCargo; private double salario;

```
    public Alcaldia(String nombreAlcalde, String partidoPolitico, int anosEnCargo, double
salario) {
        this.nombreAlcalde = nombreAlcalde;
        this.partidoPolitico = partidoPolitico;
        this.anosEnCargo = anosEnCargo;
        this.salario = salario;
    }

    public String getNombreAlcalde() {
        return nombreAlcalde;
    }

    public void setNombreAlcalde(String nombreAlcalde) {
        this.nombreAlcalde = nombreAlcalde;
    }

    public String getPartidoPolitico() {
        return partidoPolitico;
    }

    public void setPartidoPolitico(String partidoPolitico) {
        this.partidoPolitico = partidoPolitico;
    }

    public int getAnosEnCargo() {
        return anosEnCargo;
    }

    public void setAñosEnCargo(int añosEnCargo) {
        this.anosEnCargo = añosEnCargo;
    }

    public double getSalario() {
        return salario;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }

   }
```

```
package Modelo;
```

```
public class Administrativos extends Alcaldia {
    private String nombre;
    private int edad;
    private String areaEncargada;

    public Administrativos(String nombreAlcalde, String partidoPolitico,
            int anosEnCargo, double salario, String nombre, int edad, String areaEncargada) {
        super(nombreAlcalde, partidoPolitico, anosEnCargo, salario);
        this.nombre = nombre;
        this.edad = edad;
        this.areaEncargada = areaEncargada;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public String getAreaEncargada() {
        return areaEncargada;
    }

    public void setAreaEncargada(String areaEncargada) {
        this.areaEncargada = areaEncargada;
    }
}
```

```
package Modelo;
import java.util.Scanner;

public class Cobranzas extends Alcaldia {
    private double ingresos;
    private double egresos;
```

```
    public Cobranzas(String nombreAlcalde, String partidoPolitico, int
            anosEnCargo, double salario, double ingresos, double egresos) {
        super(nombreAlcalde, partidoPolitico, anosEnCargo, salario);
        this.ingresos = ingresos;
        this.egresos = egresos;
    }

    public double getIngresos() {
        return ingresos;
    }

    public void setIngresos(double ingresos) {
        this.ingresos = ingresos;
    }

    public double getEgresos() {
        return egresos;
    }

    public void setEgresos(double egresos) {
        this.egresos = egresos;
    }

    public double calcularBalance() {
        return ingresos - egresos;
    }
}
```

Controlador

```
package controlador;

import Vista.VistaAdministrativo;
import Vista.VistaAlcaldia;

public class AdministrativoControlador {
    public static VistaAdministrativo ventana=new VistaAdministrativo();
    public static void mostrar(){ventana.setVisible(true);}
    public static void ocultar(){ ventana.setVisible(false);}

}
```

```
package controlador;

import Vista.VistaAlcaldia;

public class AlcaldeControlador {

    public static VistaAlcaldia ventana=new VistaAlcaldia();
    public static void mostrar(){ventana.setVisible(true);}
    public static void ocultar(){ ventana.setVisible(false);}
}
```

```
package controlador;
import Modelo.Cobranzas;
import Vista.VentanaLogin;
import Vista.VistaCobranzas;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JOptionPane;

public class CobranzasControlador  implements ActionListener{

    public static VistaCobranzas ventana=new VistaCobranzas();
    public static void mostrar(){ventana.setVisible(true);}

    private Cobranzas objeModelo;
    private VistaCobranzas objCobranzas;

    public CobranzasControlador() {
        this.objeModelo = objeModelo;
        this.objCobranzas = objCobranzas;
        objCobranzas.getBtnCalcular().addActionListener(this);

    }
    public void iniciar(){
      objCobranzas.setTitle("USTED VA A SUMAR POR MEDELO DE VISTA
CONTROLADOR");
       objCobranzas.setLocationRelativeTo(null);
       objCobranzas.getTxtC1().setText(String.valueOf(objeModelo.getIngresos()));
       objCobranzas.getTxtC2().setText(String.valueOf(objeModelo.getEgresos()));
       objCobranzas.getTxtC3().setText(String.valueOf(objeModelo.calcularBalance()));
    }
    @Override
    public void actionPerformed(ActionEvent e) {
```

```
    //codigo de la ejecucion
    objeModelo.setIngresos(Integer.valueOf(objCobranzas.getTxtC1().getText()));
    objeModelo.setEgresos(Integer.valueOf(objCobranzas.getTxtC2().getText()));

    if (e.getSource()==objCobranzas.getBtnCalcular()){
        objeModelo.calcularBalance();

    }
    objCobranzas.getTxtC3().setText(String.valueOf(objeModelo.calcularBalance()));
    }


}
```

```
package controlador;

import Vista.VentanaLogin;
import Vista.VistaPrincipal;
import static controlador.PrincipalControlador.ventana;

public class LoginControlador {
    public static VentanaLogin ventana=new VentanaLogin();
    public static void mostrar(){ventana.setVisible(true);}
    public static void ocultar(){ ventana.setVisible(false);}

    public static void eventoBoton1(){
        String usuario=ventana.getjTextField1().getText();
        String clave=ventana.getjTextField2().getText();

        if (usuario.contains("kelvin")&& clave.equals("123")){
            ocultar();
            PrincipalControlador.mostrar();
        }else{
            System.out.println("usuario o clave incorrecta");
        }
    }
}
```

```
package controlador;

import Vista.VistaPrincipal;
```

```
public class PrincipalControlador {
   public static VistaPrincipal ventana=new VistaPrincipal();
   public static void mostrar(){ventana.setVisible(true);}
   public static void ocultar(){ ventana.setVisible(false);}

   public static void botonoSalir(){
      ocultar();
      LoginControlador.mostrar();
   }

   public static void botonoAlcaldia(){
      ocultar();
      AlcaldeControlador.mostrar();
   }

   public static void botonoCobranzas(){
      ocultar();
      CobranzasControlador.mostrar();
   }

   public static void botonoAdministrativos(){
      ocultar();
      CobranzasControlador.mostrar();
   }
}
```

```
package controlador;

import Modelo.Administrativos;
import Modelo.Cobranzas;
import java.util.Scanner;

public class Principal {
   public static void main(String[] args) {

      LoginControlador.mostrar();

      Scanner leer = new Scanner(System.in);

      System.out.println("Ingrese los datos del Alcalde:");
      System.out.print("Nombre: ");
      String nombreAlcalde = leer.nextLine();

      System.out.print("Partido Politico: ");
```

```
        String partidoPolitico = leer.nextLine();

        System.out.print("Años en Cargo: ");
        int anosEnCargo = leer.nextInt();

        System.out.print("Salario: ");
        double salario = leer.nextDouble();
        leer.nextLine(); // Consumir el salto de línea después de leer el double

        Cobranzas cobranzas;

        System.out.println("\nIngrese los datos de Cobranzas:");
        System.out.print("Ingresos: ");
        double ingresos = leer.nextDouble();

        System.out.print("Egresos: ");
        double egresos = leer.nextDouble();
        leer.nextLine(); // Consumir el salto de línea después de leer el double

        cobranzas = new Cobranzas(nombreAlcalde, partidoPolitico, anosEnCargo, salario,
ingresos, egresos);

        System.out.println("\nIngrese los datos del Area Administrativa:");
        System.out.print("Nombre del Encargado: ");
        String nombreEncargado = leer.nextLine();

        System.out.print("Edad del Encargado: ");
        int edadEncargado = leer.nextInt();
        leer.nextLine(); // Consumir el salto de línea después de leer el int

        System.out.print("Area Encargada: ");
        String areaEncargada = leer.nextLine();

        Administrativos administrativo = new Administrativos(nombreAlcalde, partidoPolitico,
anosEnCargo, salario, nombreEncargado, edadEncargado, areaEncargada);

        leer.close(); // Cerrar el Scanner

        // Mostramos los datos ingresados
        System.out.println("\nDatos del Alcalde:");
        System.out.println("Nombre: " + cobranzas.getNombreAlcalde());
        System.out.println("Partido Político: " + cobranzas.getPartidoPolitico());
        System.out.println("Años en Cargo: " + cobranzas.getAnosEnCargo());
        System.out.println("Salario: " + cobranzas.getSalario());

        System.out.println("\nRegistro de Cobranzas:");
```

```
        System.out.println("Ingresos: " + cobranzas.getIngresos());
        System.out.println("Egresos: " + cobranzas.getEgresos());
        System.out.println("Balance: " + cobranzas.calcularBalance());

        System.out.println("\nDatos del Area Administrativa:");
        System.out.println("Nombre: " + administrativo.getNombre());
        System.out.println("Edad: " + administrativo.getEdad());
        System.out.println("Area Encargada: " + administrativo.getAreaEncargada());
    }
}
```

4. Conclusiones

- El desarrollo del programa básico en Java se realizó con éxito, además nos permitió familiarizarnos aún más con los conceptos fundamentales de la programación orientada a objetos
- La utilización de diagramas UML fue de gran ayuda para comprender y visualizar la estructura del programa. Así como la implementación de buenas prácticas de programación, como el código limpio, nos ayudó a mejorar la legibilidad y mantenibilidad de nuestro programa

5. Recomendaciones

- Continuar explorando y profundizando en los conceptos y temas de programación orientada a objetos más allá de los requeridos para este proyecto. Esto permitirá seguir fortaleciendo las habilidades y comprensión, lo que se reflejará en proyectos futuros.
- Mantener la práctica de documentar el código de manera clara y concisa. Establecer el hábito de incluir comentarios y explicaciones detalladas de las partes clave del código, así como de su funcionamiento general, facilitará la comprensión del propio código en el futuro.

6. Bibliografía

Cabrera, I. (2022, January 12). *Todo lo que necesitas saber sobre el diagrama de caso de uso*. Venngage. Retrieved May 30, 2023, from https://es.venngage.com/blog/diagrama-de-caso-de-uso/

Lara, D. (2015, July 7). *Encapsulamiento en la programación orientada a objetos*. Styde.net. Retrieved May 30, 2023, from https://styde.net/encapsulamiento-en-la-programacion-orientada-a-objetos/

Mancuzo, G. (2021, June 24). *Qué son los Diagramas de UML? + Tipos + Importancia*. Blog – ComparaSoftware. Retrieved May 30, 2023, from https://blog.comparasoftware.com/diagramas-de-uml-que-significa-esta-metodologia/

Martínez, M. (2020, November 2). *¿Qué es la Programación Orientada a Objetos?* Profile. Retrieved May 30, 2023, from https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/

Paredes, B. (2022, January 26). *¿Qué es código limpio?* LinkedIn. Retrieved May 30, 2023, from https://es.linkedin.com/pulse/qu%C3%A9-es-c%C3%B3digo-limpio-b-parde

Burbeck, S. (2023, January 6). *Programación de Aplicaciones*. Model-View-Controller. Retrieved June 24, 2023, from http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.

Diácono, J. (2023, January 6). *Arquitectura Modelo-Vista-Controlador (MVC)*. Retrieved June 24, 2023, from http://www.jdl.co.uk/briefings/mvc.pdf

Hernandez, R. D. (2021, June 28). *El patrón modelo-vista-controlador: Arquitectura y frameworks explicados*. freeCodeCamp. Retrieved June 24, 2023, from https://www.freecodecamp.org/espanol/news/el-modelo-de-arquitectura-view-controller-pattern/

Pantoja, B. (2004). *El patrón de diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swing*. SciELO Bolivia. Retrieved June 24, 2023, from http://www.scielo.org.bo/scielo.php?pid=S1683-07892004000100005&script=sci_arttext