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 : Tarea 2

NOMBRES : Kelvin Quezada

NIVEL-PARALELO : Segundo “A”

DOCENTE : Ing. Cevallos Farias Javier Moyota

FECHA DE ENTREGA : 17/06/2023

Grupo 4

SANTO DOMINGO - ECUADOR

2023

Ilustración

Ilustración de Figuras

1. Introducción

En este informe, se presentará un programa desarrollado en el lenguaje de programación que integra todos los temas aprendidos en relación a la POO como clases, objetos, herencia, polimorfismo, sobreescritura, encapsulación y abstracción.

Mediante este ejercicio se desarrollo un ejercicio de jerarquía de clases de empleados utilizando herencia, constructores, método get y set y abstracto realizamos lo que es condicionales mediante la sobreescritura.

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. El 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.

Los 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. El 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 nuestro ejercicio de jerarquía de clases.

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.

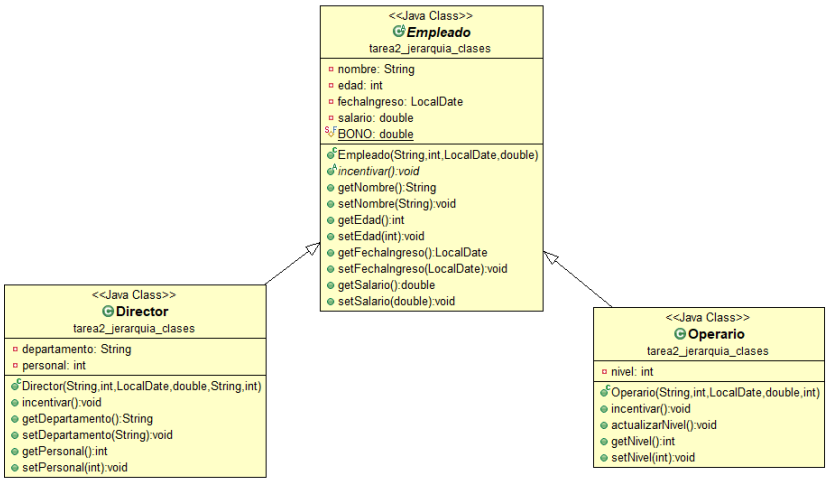

Figura 1. Diagrama UML

- **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.

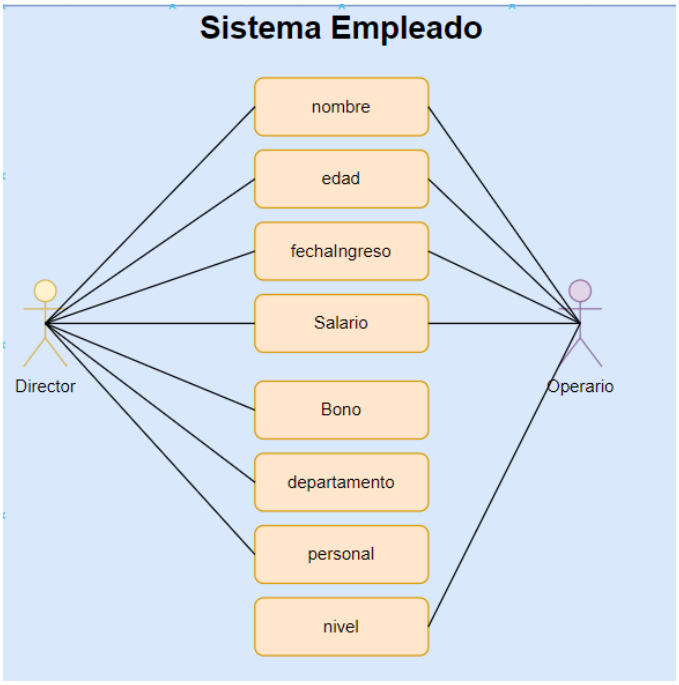

Figura 2. Diagrama de Uso

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)

```
public abstract class Empleado {

private String nombre;
    public int edad;
    private LocalDate fechaIngreso;
    private double salario;
    protected static final double BONO = 2000.0;
```

Figura 3. Encapsulamiento

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 miembros de nuevas instancias permanezcan inicializadas.

```
public Empleado(String nombre, int edad, LocalDate fechaIngreso, double salario) {
    this.nombre = nombre;
    this.edad = edad;
    this.fechaIngreso = fechaIngreso;
    this.salario = salario;
}
```

Figura 4. Constructores

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.

```java
    //el metodo get nos devuelve el valor de Nombre

    public String getNombre() {
        return nombre;
    }
    //el metodo set no devuelve nada solo establece datos
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
```

Figura 5. Métodos get y set

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.

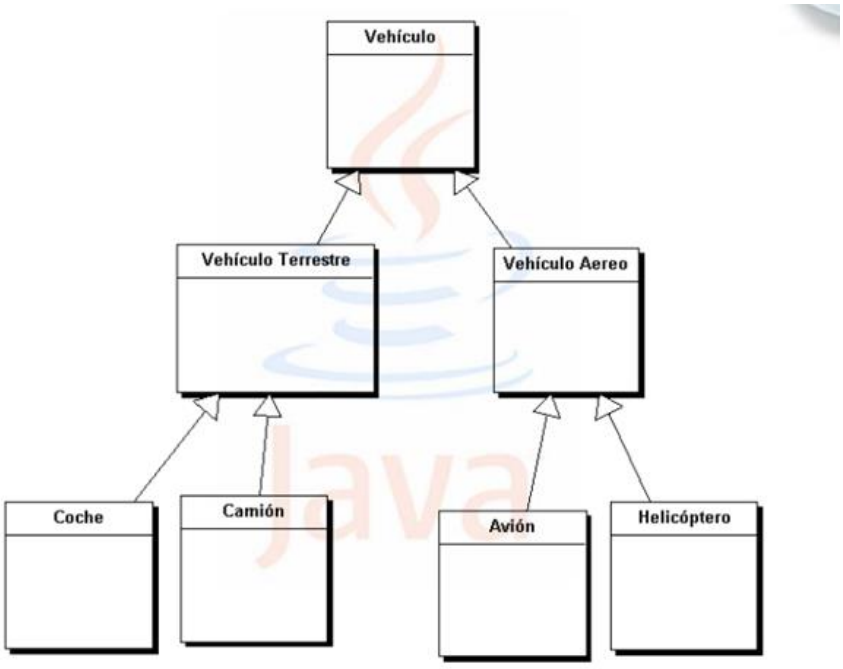

Figura 6. Generalización/ Especialización

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.

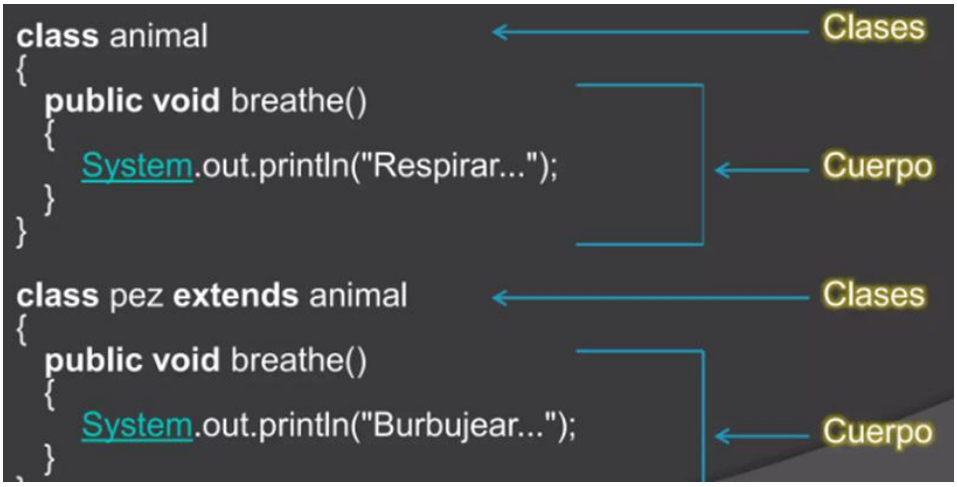

Figura 7. Implementación

3.3. Gestión de Defecto testing

3.3.1. Verificación y Validación

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)

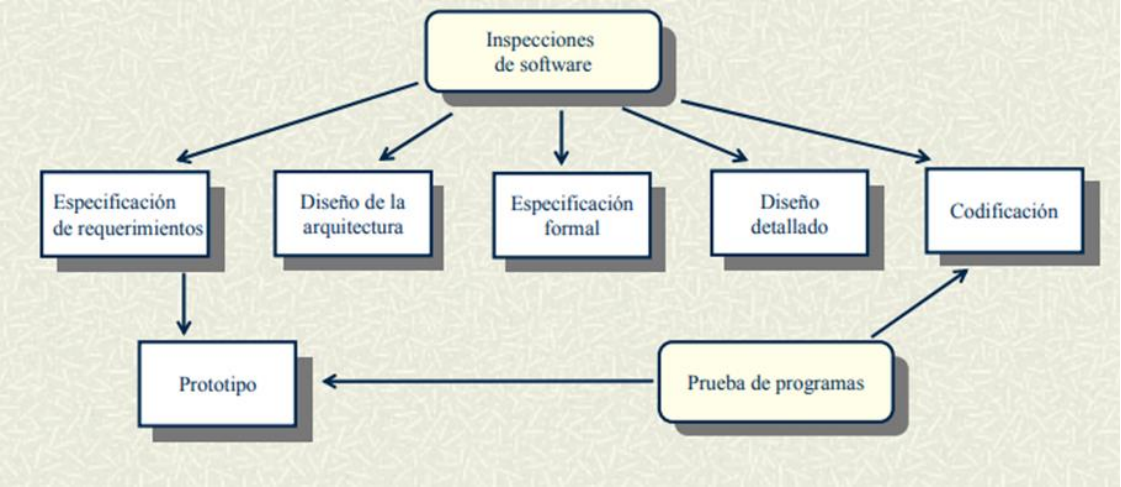

Figura 8. Verificacion y Validacion

3.4. 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.

3.4.1. Sobrecarga de métodos

sobrecarga permite declarar métodos que se llamen igual pero que reciban parámetros diferentes (no puede haber 2 métodos con el mismo nombre y los mismos parámetros), por esta razón lo que define a que método se ingresa, son los argumentos que se envían como parámetros.

```java
public class Rectangulo implements Figura, Dibujar{

    double base;
    double altura;

    public Rectangulo(double base, double altura) {
        this.altura=altura;
        this.base=base;
    }

    @Override
    public void dibujar() {
        System.out.println(x: "vas a dibujar");

    }

    @Override
    public double calcularArea() {
        double resultado=base*altura;
        return resultado;
    }
```

```java
public class Circulo implements Figura, Dibujar, Rotar {

    double radio;

    public Circulo(double radio) {
        this.radio=radio;
    }
    public double calcularArea() {
        double pi=3.1416;
        double resultado=pi*radio*radio;
        return resultado;
    }
    @Override
    public void rodar() {
        System.out.println(x: "vas a rotar");

    }
    @Override
    public void dibujar() {
        System.out.println(x: "vas a dibujar");

    }
```

Figura 9. Sobrecarga de métodos

3.4.2. Sobre escritura de métodos.

es decir, si tengo una clase padre con el método incetivar() yo puedo crear en la clase hija un método que también se llame incetivar () pero implementándolo según lo que necesite (siguiendo obviamente unas reglas.

```
public abstract void incentivar();
```

```
//realizamos una sobre escritura en el metodo incentivar
    @Override
    public void incentivar() {
        //plusMonths() de la clase LocalDate se usa para agregar la cantidad de meses especificados en esta LocalDate
        //isBefore verifica si esta fecha es anterior a la fecha especificada.
        //now obtener la fecha y la hora actuales en una zona horaria especifica
        if (getFechaIngresada().plusMonths(monthsToAdd: 30).isBefore(other: LocalDate.now()) && personal > 20) {
            setSalario(getSalario() + 2 * BONO);
        } else if (getFechaIngresada().plusMonths(monthsToAdd: 30).isBefore(other: LocalDate.now()) || personal > 20) {
            setSalario(getSalario() + BONO);
        }
    }
```

Figura 10. Sobre escritura de métodos

3.5. Anexos

3.5.1. Jerarquía de clases empleados

En este ejercicio deberás crear una jerarquía de clases que representen los empleados de una empresa y su funcionalidad.

Se deberán crear las siguientes clases

Empleado

- Es la clase padre, contiene los atributos nombre.
- Nombre del empleado
- edad. Edad del empleado
- fechaIngreso. Fecha en la que ingresó en la empresa
- salario. Salario anual

Además de los constructores y métodos setter/getter, incluirá un método abstracto llamado incentivar(), que no devolverá ningún resultado ni recibirá ningún parámetro. También incluirá una constante doble llamada BONO, con un valor predefinido.

Director

Será un subtipo de empleado que incluirá los atributos:

- departamento. Nombre del departamento del que es director
- personal. Número de personas a su cargo Sobrescribirá el método incentivar(), de modo que haga lo siguiente:

Si el lleva más de 30 meses en la empresa y tiene más de 20 personas a su cargo, se le incrementará su salario con el doble del bono. Si cumple solo una de las condiciones anteriores, se le incrementará su salario con el bono y si no cumple ninguna, no se hará nada

Operario

Será un subtipo de empleado que incluirá el siguiente atributo:

- nivel. Nivel de seguridad del operario, puede ser un valor entre 1 y 5.

Sobrescribirá el método incentivar(), de modo que haga lo siguiente:

Si tiene más de 30 años y su nivel es superior a 2, se le incrementará su salario con el doble del bono. Si cumple solo una de las condiciones anteriores, se le incrementará su salario con el bono y si no cumple ninguna, no se hará nada. Dispondrá además de un método actualizarNivel(), que en caso de que el empleado lleve más de dos años en la empresa se le subirá un nivel de seguridad. Si ya está en nivel 5 no se hará nada

3.5.2. El diagrama Uml y diagrama de Uso

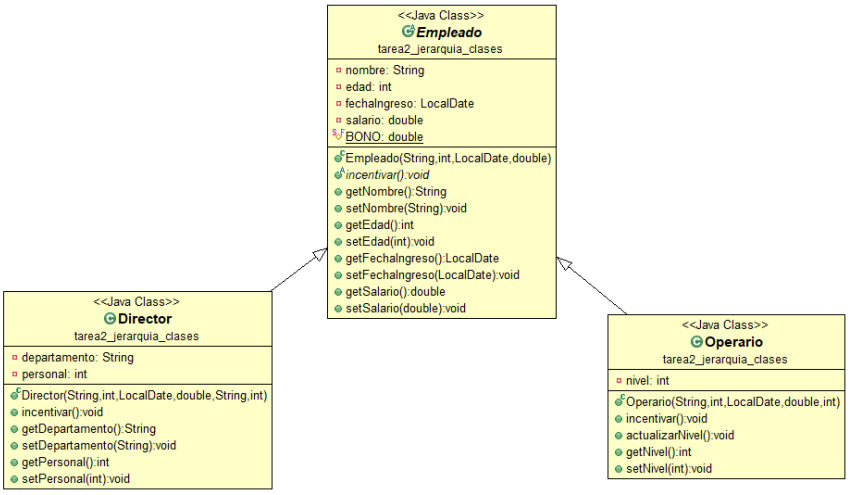

Figura 11. Diagrama uml Sistema Empleado

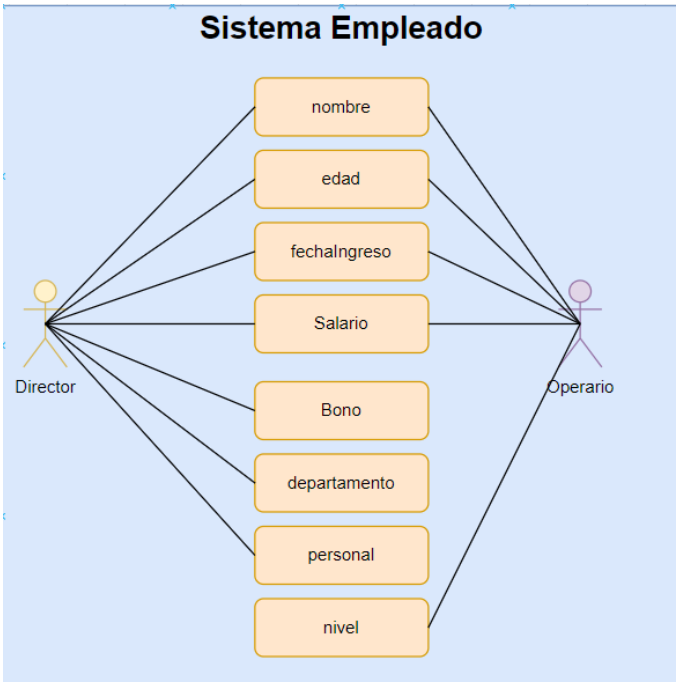

Figura 12. Diagrama de Uso de sistema de Empleado

3.5.3. Código del programa.

```
package tarea2_jerarquia_clases;
//la libreria LocalDate guarda la fecha de una etapa año-mes-dia
```

```
import java.time.LocalDate;
//Clase Padre
public abstract class Empleado {
  //Atributos de la clase
    private String nombre;
    public int edad;
    private LocalDate fechaIngreso;
    private double salario;
    //acceso protegido
    //creamos una constante estatica
    protected static final double BONO = 2000.0;
//creamos los contructores donde llamaremos todos los atributos de la clase Empleado
    public Empleado(String nombre, int edad, LocalDate fechaIngreso, double salario) {
        this.nombre = nombre;
        this.edad = edad;
        this.fechaIngreso = fechaIngreso;
        this.salario = salario;
    }
    //creamos un metodo abstract tipo vacio
    public abstract void incentivar();

    //el metodo get nos devuelve el valor de Nombre

    public String getNombre() {
        return nombre;
    }
    //el metodo set no devuelve nada solo establece datos
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getEdad() {
        return edad;
    }

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

    public LocalDate getFechaIngresada() {
        return fechaIngreso;
    }

    public void setFechaIngreso(LocalDate fechaIngreso) {
        this.fechaIngreso = fechaIngreso;
    }
```

```
    public double getSalario() {
        return salario;
    }

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

```
import java.time.LocalDate;
//tendremos la clases hija que va heredar los atributos de la clase padre
class Director extends Empleado {
    //atritubo de la clase director
    private String departamento;
    private int personal;
//creamos los contructores donde vamos a llamar los atributos de la clase padre y de la clase hija llamada Director
    public Director(String nombre, int edad, LocalDate fechaIngresada, double salario, String departamento, int personal) {
        super(nombre, edad, fechaIngresada, salario);
        this.departamento = departamento;
        this.personal = personal;
    }
//realizamos una sobre escritura en el metodo incentivar
    @Override
    public void incentivar() {
        /*
        Si el lleva más de 30 meses en la empresa y tiene más de 20 personas a su cargo, se le
        incrementará su salario con el doble del bono. Si cumple solo una de las condiciones
        anteriores, se le incrementará su salario con el bono y si no cumple ninguna, no se hará nada
        plusMonths() de la clase LocalDate se usa para agregar la cantidad de meses especificados en esta LocalDate
        isBefore verifica si esta fecha es anterior a la fecha especificada.
        now obtener la fecha y la hora actuales en una zona horaria específica
        */
        if (getFechaIngresada().plusMonths(30).isBefore(LocalDate.now()) && personal > 20) {
            setSalario(getSalario() + 2 * BONO);
        } else if (getFechaIngresada().plusMonths(30).isBefore(LocalDate.now()) || personal > 20) {
            setSalario(getSalario() + BONO);
        }
    }

    // Cramos los metodos Getters y setters
```

```
    public String getDepartamento() {
        return departamento;
    }

    public void setDepartamento(String departamento) {
        this.departamento = departamento;
    }

    public int getPersonal() {
        return personal;
    }

    public void setPersonal(int personal) {
        this.personal = personal;
    }
}
```

```
import java.time.LocalDate;
//tendremos la clases hija que va heredar los atributos de la clase padre
class Operario extends Empleado {
    private int nivel;
    //creamos los contructores donde vamos a llamar los atributos de la clase padre y de la clase hija llamada Operario
    public Operario(String nombre, int edad, LocalDate fechaIngreso, double salario, int nivel) {
        super(nombre, edad, fechaIngreso, salario);
        this.nivel = nivel;
    }
//creamos una sobre escritura incentivar para realizar un metodo
    @Override
    public void incentivar() {
        /*
        Si tiene más de 30 años y su nivel es superior a 2, se le incrementará su salario con el doble del
        bono. Si cumple solo una de las condiciones anteriores, se le incrementará su salario con el
        bono y si no cumple ninguna, no se hará nada.
        */
        //operdor and && asocia dos términos y busca un registro coincidente si ambos términos o frases existen en un registro
        if (getEdad() > 30 && nivel > 2) {
            //daremos a conocer el salario que sumaremos para dos y lo multiplicamos para el bono
            setSalario(getSalario() + 2 * BONO);
            //daremos a conocer la edad de la persona con el operador or || ya que va evalua dos operadores
        } else if (getEdad() > 30 || nivel > 2) {
```

```
            //daremos a conocer el bono del salario
            setSalario(getSalario() + BONO);
        }
    }
//creamos los metodos
    public void actualizarNivel() {
        /*
        Dispondrá además de un método actualizarNivel(), que en caso de que el empleado lleve
más
        de dos años en la empresa se le subirá un nivel de seguridad. Si ya está en nivel 5 no se
hará
        nada
        plusYears es una sintaxi que Devuelve LocalDate después de agregar el número
especificado de años
        isBefore verifica si esta fecha es anterior a la fecha especificada.
        now obtener la fecha y la hora actuales en una zona horaria específica
        operdor and && asocia dos términos y busca un registro coincidente si ambos términos o
frases existen en un registro
        */
        if (getFechaIngresada().plusYears(2).isBefore(LocalDate.now()) && nivel < 5) {
            nivel++;
        }
    }

    // Getters y setters

    public int getNivel() {
        return nivel;
    }

    public void setNivel(int nivel) {
        this.nivel = nivel;
    }
}
```

```
import java.time.LocalDate;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        //El Scanner es para el ingreso de datos
        Scanner leer = new Scanner(System.in);
        //creamos una variabale tipo entera
        int opc;
        //creamos un do-while para la creacion de menu
```

```
      do{
            System.out.println("Bienvenidos al menu de Empleados");
            System.out.println("1. Director");
            System.out.println("2. Operario");
            System.out.println("3. Salir");
            opc=leer.nextInt();
            switch(opc){
               case 1:
                   System.out.println("=======================================");
                  System.out.println("Ingrese los datos del Director:");
                  System.out.println("=======================================");
                  leer.nextLine();
                  System.out.print("Nombre: ");
                  //ingresamos los nextLine para el ingreso de caracteres
                  String nombreDirector = leer.nextLine();
                  System.out.print("Edad: ");
                  //ingresamos los nextInte para el ingreso de numeros enteros
                  int edadDirector = leer.nextInt();
                  System.out.print("Fecha de ingreso (AAAA-MM-DD): ");
                  //ingresamos los LocalDate y el next para el ingreso de datos de fecha
                  LocalDate fechaIngresoDirector = LocalDate.parse(leer.next());
                  System.out.print("ingrese el salario anual: ");
                  //ingresamos el nextDouble para almacenar numeros enteros y decimales
                  double salarioDirector = leer.nextDouble();
                  System.out.print("DEPARTAMENTO: ");
                  String departamentoDirector = leer.next();
                  System.out.print("NUMERO DE PERSONAS: ");
                  int personalDirector = leer.nextInt();
                  //llamamos a nuestra clase Director y cramos un nuevo objeto y llamamos los
atributos de la clase
                  Director director = new Director(nombreDirector, edadDirector,
fechaIngresoDirector,
                        salarioDirector,departamentoDirector, personalDirector);
                  //llamamos a nuestro objeto y a nuesto metodo incentivar
                  director.incentivar();
                  System.out.println("Datos del Director");
                  System.out.println("NOMBRE: "+director.getNombre());
                  System.out.println("EDA: "+director.getEdad());
                  System.out.println("FECHA INGRESO: "+director.getFechaIngresada());
                  System.out.println("SALARIO ANUAL: "+director.getSalario());
                  System.out.println("DEPARTAMENTO: "+director.getDepartamento());
                  System.out.println("PERSONAS A CARGO: "+director.getPersonal());
                  break;
               case 2:
                   System.out.println("iNGRESO DE DATOS OPERARIO:");
                  System.out.print("NOMBRE: ");
```

```
                String nombreOperario = leer.nextLine();
                leer.nextLine();
                System.out.print("EDAD: ");
                int edadOperario = leer.nextInt();
                System.out.print("FECHA DE INGRESO (AAAA-MM-DD): ");
                LocalDate fechaIngresoOperario = LocalDate.parse(leer.next());
                System.out.print("SALARIO ANUAL: ");
                double salarioOperario = leer.nextDouble();
                System.out.print("NIVEL DE SEGURIDAD ENTRE (1 y 5): ");
                int nivelOperario = leer.nextInt();
                ////llamamos a nuestra clase Operario y cramos un nuevo objeto y llamamos los
atributos de la clase
                Operario operario = new Operario(nombreOperario, edadOperario,
                      fechaIngresoOperario, salarioOperario,nivelOperario);
                //llamamos nuestro dos metodos incentivar y actualizarNivel
                operario.incentivar();
                operario.actualizarNivel();
                 //imprimir los resultado
                 System.out.println("Datos del Operario");
                 System.out.println("NOMBRE: "+operario.getNombre());
                 System.out.println("EDA: "+operario.getEdad());
                 System.out.println("FECHA INGRESO: "+operario.getFechaIngresada());
                 System.out.println("SALARIO ANUAL: "+operario.getSalario());
                 System.out.println("NIVEL: "+operario.getNivel());
                break;

             case 3:
                System.out.println("A finalizado");
                break;
             default:
                System.out.println("Esta opcion es invalidad vuelva a intentar");
          }
        }while(opc !=3);









   }
}
```

3.5.4. Capture del código

```java
package tarea2_jerarquia_clases;
//la libreria LocalDate guarda la fecha de una etapa año-mes-dia
import java.time.LocalDate;
//Clase Padre
public abstract class Empleado {
  //Atributos de la clase
    private String nombre;
    public int edad;
    private LocalDate fechaIngreso;
    private double salario;
    //acceso protegido
    //creamos una constante estatica
    protected static final double BONO = 2000.0;
//creamos los contructores donde llamaremos todos los atributos de la clase Empleado
    public Empleado(String nombre, int edad, LocalDate fechaIngreso, double salario) {
        this.nombre = nombre;
        this.edad = edad;
        this.fechaIngreso = fechaIngreso;
        this.salario = salario;
    }
    //creamos un metodo abstract tipo vacio
    public abstract void incentivar();
    //el metodo get nos devuelve el valor de Nombre
    public String getNombre() {
        return nombre;
    }
    //el metodo set no devuelve nada solo establece datos
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public int getEdad() {
        return edad;
    }
```

```java
    public void setEdad(int edad) {
        this.edad = edad;
    }
    public LocalDate getFechaIngresada() {
        return fechaIngreso;
    }
    public void setFechaIngreso(LocalDate fechaIngreso) {
        this.fechaIngreso = fechaIngreso;
    }
    public double getSalario() {
        return salario;
    }
    public void setSalario(double salario) {
        this.salario = salario;
    }
}
```

Figura 13. Clase Padre Empleado

```java
import java.time.LocalDate;
//tendremos la clases hija que va heredar los atributos de la clase padre
class Director extends Empleado {
    //atritubo de la clase director
    private String departamento;
    private int personal;
//creamos los contructores donde vamos a llamar los atributos de la clase padre y de la clase hija llamada Director
    public Director(String nombre, int edad, LocalDate fechaIngresada, double salario, String departamento, int personal) {
        super(nombre, edad, fechaIngreso: fechaIngresada, salario);
        this.departamento = departamento;
        this.personal = personal;
    }
//realizamos una sobre escritura en el metodo incentivar
    @Override
    public void incentivar() {
        /*
        Si el lleva más de 30 meses en la empresa y tiene más de 20 personas a su cargo, se le
        incrementará su salario con el doble del bono. Si cumple solo una de las condiciones
        anteriores, se le incrementará su salario con el bono y si no cumple ninguna, no se hará nada
        plusMonths() de la clase LocalDate se usa para agregar la cantidad de meses especificados en esta LocalDate
        isBefore verifica si esta fecha es anterior a la fecha especificada.
        now obtener la fecha y la hora actuales en una zona horaria específica
        */
        if (getFechaIngresada().plusMonths(monthsToAdd: 30).isBefore(other: LocalDate.now()) && personal > 20) {
            setSalario(getSalario() + 2 * BONO);
        } else if (getFechaIngresada().plusMonths(monthsToAdd: 30).isBefore(other: LocalDate.now()) || personal > 20) {
            setSalario(getSalario() + BONO);
        }
    }
```

```java
    // Cramos los metodos Getters y setters
    public String getDepartamento() {
        return departamento;
    }

    public void setDepartamento(String departamento) {
        this.departamento = departamento;
    }

    public int getPersonal() {
        return personal;
    }

    public void setPersonal(int personal) {
        this.personal = personal;
    }
}
```

Figura 14. Clase hija Director

```java
package tarea2_jerarquia_clases;

import java.time.LocalDate;
//tendremos la clases hija que va heredar los atributos de la clase padre
class Operario extends Empleado {
    private int nivel;
    //creamos los contructores donde vamos a llamar los atributos de la clase padre y de la clase hija llamada Operario
    public Operario(String nombre, int edad, LocalDate fechaIngreso, double salario, int nivel) {
        super(nombre, edad, fechaIngreso, salario);
        this.nivel = nivel;
    }
//creamos una sobre escritura incentivar para realizar un metodo
    @Override
    public void incentivar() {
        /*
        Si tiene más de 30 años y su nivel es superior a 2, se le incrementará su salario con el doble del
        bono. Si cumple solo una de las condiciones anteriores, se le incrementará su salario con el
        bono y si no cumple ninguna, no se hará nada.
        */
        //operdor and && asocia dos términos y busca un registro coincidente si ambos términos o frases existen en un registro
        if (getEdad() > 30 && nivel > 2) {
            //daremos a conocer el salario que sumaremos para dos y lo multiplicamos para el bono
            setSalario(getSalario() + 2 * BONO);
            //daremos a conocer la edad de la persona con el operador or || ya que va evalua dos operadores
        } else if (getEdad() > 30 || nivel > 2) {
            //daremos a conocer el bono del salario
            setSalario(getSalario() + BONO);
        }
    }
//creamos los metodos
    public void actualizarNivel() {
        /*
        Dispondrá además de un método actualizarNivel(), que en caso de que el empleado lleve más
        de dos años en la empresa se le subirá un nivel de seguridad. Si ya está en nivel 5 no se hará
        nada
        plusYears es una sintaxi que Devuelve LocalDate después de agregar el número especificado de años
        isBefore verifica si esta fecha es anterior a la fecha especificada.
        now obtener la fecha y la hora actuales en una zona horaria específica
        operdor and && asocia dos términos y busca un registro coincidente si ambos términos o frases existen en un registro
        */
        if (getFechaIngresada().plusYears(yearsToAdd: 2).isBefore(other: LocalDate.now()) && nivel < 5) {
            nivel++;
        }
    }
    // Getters y setters
    public int getNivel() {
        return nivel;
    }
    public void setNivel(int nivel) {
        this.nivel = nivel;
    }
}
```

Figura 15. Clase hija Operario

```java
import java.time.LocalDate;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        //El Scanner es para el ingreso de datos
        Scanner leer = new Scanner(source:System.in);
        //creamos una variabale tipo entera
        int opc;
        //creamos un do-while para la creacion de menu
        do{
        System.out.println(x: "Bienvenidos al menu de Empleados");
        System.out.println(x: "1. Director");
        System.out.println(x: "2. Operario");
        System.out.println(x: "3. Salir");
        opc=leer.nextInt();
        switch(opc){
            case 1:
                System.out.println(x: "=========================================");
                System.out.println(x: "Ingrese los datos del Director:");
                System.out.println(x: "=========================================");
                leer.nextLine();
                System.out.print(s: "Nombre: ");
                //ingresamos los nextLine para el ingreso de caracteres
                String nombreDirector = leer.nextLine();
                System.out.print(s: "Edad: ");
                //ingresamos los nextInte para el ingreso de numeros enteros
                int edadDirector = leer.nextInt();
                System.out.print(s: "Fecha de ingreso (AAAA-MM-DD): ");
                //ingresamos los LocalDate y el next para el ingreso de datos de fecha
                LocalDate fechaIngresoDirector = LocalDate.parse(text: leer.next());
                LocalDate fechaIngresoDirector = LocalDate.parse(text: leer.next());
                System.out.print(s: "ingrese el salario anual: ");
                //ingresamos el nextDouble para almacenar numeros enteros y decimales
                double salarioDirector = leer.nextDouble();
                System.out.print(s: "DEPARTAMENTO: ");
                String departamentoDirector = leer.next();
                System.out.print(s: "NUMERO DE PERSONAS: ");
                int personalDirector = leer.nextInt();
                //llamamos a nuestra clase Director y cramos un nuevo objeto y llamamos los atributos de la clase
                Director director = new Director(nombre:nombreDirector, edad: edadDirector, fechaIngresada: fechaIngresoDirector,
                        salario: salarioDirector,departamento: departamentoDirector, personal: personalDirector);
                //llamamos a nuestro objeto y a nuesto metodo incentivar
                director.incentivar();
                System.out.println(x: "=========================================");
                System.out.println(x: "Datos del Director");
                System.out.println(x: "=========================================");
                System.out.println("NOMBRE: "+director.getNombre());
                System.out.println("EDA: "+director.getEdad());
                System.out.println("FECHA INGRESO: "+director.getFechaIngresada());
                System.out.println("SALARIO ANUAL: "+director.getSalario());
                System.out.println("DEPARTAMENTO: "+director.getDepartamento());
                System.out.println("PERSONAS A CARGO: "+director.getPersonal());
                break;
            case 2:
                System.out.println(x: "=========================================");
                 System.out.println(x: "iNGRESO DE DATOS OPERARIO:");
                 System.out.println(x: "=========================================");
                System.out.print(s: "NOMBRE: ");
                String nombreOperario = leer.nextLine();
                leer.nextLine();
                System.out.print(s: "EDAD: ");
                int edadOperario = leer.nextInt();
                System.out.print(s: "FECHA DE INGRESO (AAAA-MM-DD): ");
                LocalDate fechaIngresoOperario = LocalDate.parse(text: leer.next());
```

```
                System.out.print(s: "SALARIO ANUAL: ");
                double salarioOperario = leer.nextDouble();
                System.out.print(s: "NIVEL DE SEGURIDAD ENTRE (1 y 5): ");
                int nivelOperario = leer.nextInt();
                ////llamamos a nuestra clase Operario y cramos un nuevo objeto y llamamos los atributos de la clase
                Operario operario = new Operario(nombre:nombreOperario, edad: edadOperario,
                        fechaIngreso: fechaIngresoOperario, salario: salarioOperario,nivel: nivelOperario);
                //llamamos nuestro dos metodos incentivar y actualizarNivel
                operario.incentivar();
                operario.actualizarNivel();
                 //imprimir los resultado
                System.out.println(x: "=======================================");
                 System.out.println(x: "Datos del Operario");
                 System.out.println(x: "=======================================");
                 System.out.println("NOMBRE: "+operario.getNombre());
                 System.out.println("EDA: "+operario.getEdad());
                 System.out.println("FECHA INGRESO: "+operario.getFechaIngresada());
                 System.out.println("SALARIO ANUAL: "+operario.getSalario());
                 System.out.println("NIVEL: "+operario.getNivel());
                break;

            case 3:
                System.out.println(x: "A finalizado");
                break;
            default:
                System.out.println(x: "Esta opcion es invalidad vuelva a intentar");
        }
    }while(opc !=3);
```

Figura 16. Main de Menu

```
run:
Bienvenidos al menu de Empleados
1. Director
2. Operario
3. Salir
1
=======================================
Ingrese los datos del Director:
=======================================
Nombre: kelvin
Edad: 22
Fecha de ingreso (AAAA-MM-DD): 2018-04-15
ingrese el salario anual: 500
DEPARTAMENTO: admi
NUMERO DE PERSONAS: 25
=======================================
Datos del Director
=======================================
NOMBRE: kelvin
EDA: 22
FECHA INGRESO: 2018-04-15
SALARIO ANUAL: 4500.0
DEPARTAMENTO: admi
PERSONAS A CARGO: 25
```

```
Bienvenidos al menu de Empleados
1. Director
2. Operario
3. Salir
2
=======================================
iNGRESO DE DATOS OPERARIO:
=======================================
NOMBRE: peter
EDAD: 26
FECHA DE INGRESO (AAAA-MM-DD): 2020-05-09
SALARIO ANUAL: 650
NIVEL DE SEGURIDAD ENTRE (1 y 5): 3
=======================================
Datos del Operario
=======================================
NOMBRE:
EDA: 26
FECHA INGRESO: 2020-05-09
SALARIO ANUAL: 2650.0
NIVEL: 4
```

Figura 17. Ejecución de Director Y Operario

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.
- Aprendimos lo que es jerarquía de clases que nos ayuda a realizar lo que es herencia, conocer más sobre la sobreescritura aprendimos a conocer lo que es abstracciones de clase.
- 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

- Tuvimos que buscar más información para crear la fecha para eso utilizamos la librería LocalDate que nos ayuda mucho para el ingreso de las fechas.
- Mediante este ejercicio también tuvimos problemas para realizar el ejercicio para realizar utilizar las condicionales y ser validar nuestro ejercicio.

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

Link del video

https://www.youtube.com/watch?v=mRnOIMYVfqE

https://www.youtube.com/watch?v=nmZvK_LwmFI

https://www.youtube.com/watch?v=T1YD4Hl6gI8