

**FACULTAD DE INGENIERIA**

**CARRERA DE INGENIERIA DE SISTEMAS - EPE**

**CICLO 2023-02**

**Fundamentos de Programación 2 (IS275-2302-E21B)**

**TRABAJO FINAL**

**PROFESOR DEL CURSO:**

**Jymmy Stuwart Dextre Alarcon**

**TRABAJO PRESENTADO POR LOS ALUMNOS**

|  |
| --- |
| **Marquina Chamorro, Benjamín Flaviano**  **Gutiérrez Puma, José Andrés**  **Aquije Dulanto, Alberto Daniel**  **Jara Pinto, Alessandro Paris**  **Echegaray Vilcatoma, Estefany Naylu** |

**UPC, diciembre del 2023**

**ÍNDICE**

[I. INTRODUCCIÓN 3](#_Toc152828112)

[II. CAPÍTULO 1: SITUACIÓN ACTUAL 5](#_Toc152828113)

[II.I. ANÁLISIS DEL PROBLEMA 5](#_Toc152828114)

[II.I.I. ANTECEDENTES: 5](#_Toc152828115)

[II.I. II. RECOGO DE DATOS: 5](#_Toc152828116)

[II.I. III. PROPUESTA INNOVADORA: 7](#_Toc152828117)

[II.II. OBJETIVO DEL SISTEMA 8](#_Toc152828118)

[II.II.I OBJETIVO GENERAL 8](#_Toc152828119)

[II.II.II. OBJETIVO ESPECÍFICO 8](#_Toc152828120)

[II.III. LISTA DE FUNCIONALIDADES O HISTORIAS DE USUARIO O CASOS DE USO, PRIORIZADS 8](#_Toc152828121)

[II.III.I. FUNCIONALIDAD 8](#_Toc152828122)

[II.III.I.I. IMPRIMIR INFORMACIÓN DE PROYECTOS 8](#_Toc152828123)

[II.III.I.II. IMPRIMIR CLIENTES POR TIPO DE PROYECTO 8](#_Toc152828124)

[II.III.I.III. IMPRIMIR ENCARGADOS POR TIPO DE PROYECTO 9](#_Toc152828125)

[II.III.I.IV. IMPRIMIR PROYECTOS POR ENCARGADO 9](#_Toc152828126)

[II.III.I.V. IMPRIMIR DETALLES POR CÓDIGO DE PROYECTO 9](#_Toc152828127)

[II.III.I.VI. IMPRIMIR PROYECTO CON MAYOR COSTO 10](#_Toc152828128)

[II.III.II. HISTORIAS DE USUARIOS 10](#_Toc152828129)

[II.III.II.I IMPRIMIR INFORMACIÓN DE PROYECTOS 10](#_Toc152828130)

[II.III.II.II. IMPRIMIR CLIENTES POR TIPO DE PROYECTO 10](#_Toc152828131)

[II.III.II.III. IMPRIMIR ENCARGADOS POR TIPO DE PROYECTO 11](#_Toc152828132)

[II.III.II.IV. IMPRIMIR PROYECTOS POR ENCARGADO 11](#_Toc152828133)

[II.III.II.V. IMPRIMIR DETALLES POR CÓDIGO DE PROYECTO 11](#_Toc152828134)

[II.III.II.VI. IMPRIMIR PROYECTO CON MAYOR COSTO 12](#_Toc152828135)

[II.III.III. CASOS DE USO 12](#_Toc152828136)

[II.IV. CRONOGRAMA Y ASIGNACIÓN DE ACTIVIDADES 13](#_Toc152828137)

[III. CAPÍTULO 2: PROPUESTA DE INNOVACIÓN 15](#_Toc152828138)

[III.I Diseño de Pantallas 15](#_Toc152828139)

[III.I.I Controlador 16](#_Toc152828140)

[**III.I.I.ICliente:** 16](#_Toc152828141)

[**III.I.I.II Empresa:** 19](#_Toc152828142)

[**Voids:** 20](#_Toc152828143)

[**III.I.I.III Encargado:** 25](#_Toc152828144)

[**III.I.I.IV Proyecto:** 26](#_Toc152828145)

[III.I.II Modelo 30](#_Toc152828146)

[III.I.II.I Main 30](#_Toc152828147)

[**A.** **Vista** 40](#_Toc152828148)

[III.II. Vistas de Pantalla 41](#_Toc152828149)

[III.I.III Vista 46](#_Toc152828150)

[III.II. Diagrama de Clase Modelo 52](#_Toc152828151)

[Estructura de la clase Modelo: 57](#_Toc152828152)

[Modelo y Sistema Operativo: 59](#_Toc152828153)

[Patrones de diseño implementados: 59](#_Toc152828154)

[IV. CONCLUSIONES: 61](#_Toc152828155)

[V. Recomendaciones: 62](#_Toc152828156)

[VI. Objetivo del estudiante (Student Outcome): 62](#_Toc152828157)

[IV. Bibliografía: 65](#_Toc152828158)

# INTRODUCCIÓN

El presente trabajo tiene como finalidad implementar una solución a los diversos problemas que provoca la falta de implementación de gestión de la información en las empresas. En los últimos años se ha encontrado que el problema que afecta a la organización de las empresas es la falta de implementar a su sistema una gestión de información, ya que esta implementación es la forma más eficiente de recopilar, almacenar, organizar y utilizar información.

Beton Decken S.A.C, reconocida en el ámbito de la construcción por su compromiso con la calidad en la fabricación y comercialización de productos de concreto, enfrenta desafíos inherentes a la falta de una gestión eficiente de la información. A pesar de su reputación y liderazgo en el mercado desde su fundación en 2005, la ausencia de un sistema integral de gestión de datos ha presentado obstáculos en la toma de decisiones, eficiencia operativa y adaptación ágil a las demandas del mercado en constante cambio.

La falta de gestión de información ha provocado dificultades en la optimización de recursos, la planificación de la producción, el seguimiento de inventarios y la coordinación de los procesos internos. Estos desafíos han resaltado la necesidad apremiante de implementar un sistema de programación que permita a la empresa superar estas barreras, agilizar sus operaciones y mejorar la toma de decisiones estratégicas.

En este documento, se abordará el impacto de la carencia de gestión de información en la empresa Beton Decken S.A.C, así como las oportunidades y beneficios potenciales que conlleva la implementación de un sistema de programación. Se explorarán estrategias, tecnologías y consideraciones clave para la implementación exitosa de este sistema, ofreciendo soluciones que podrían resolver los desafíos actuales, aumentar la eficiencia operativa y posicionar a la empresa para un crecimiento sostenible en un mercado competitivo. La integración de un sistema de programación nos permitirá crear una oportunidad para transformar los procesos internos, mejorar la gestión de información y potenciar la capacidad de la empresa Beton Decken S.A.C para afrontar los desafíos futuros con mayor agilidad y efectividad, como también, mejorar en la toma de decisiones fundamentadas y una ventaja competitiva sostenible en el mercado de la construcción.

# CAPÍTULO 1: SITUACIÓN ACTUAL

# II.I. ANÁLISIS DEL PROBLEMA

El obstáculo que la empresa de Ingeniería Civil Beton Decken S.A.C. debe superar, es la deficiencia en sus sistemas en el área comercial, desde los inicios de la empresa se ha usado diversos programas para solucionar este problema, el cual, en los últimos meses se ha detectado pérdidas de información por el cambio de programas usados, pero sin ningún resultado efectivo para el problema.

# II.I.I. ANTECEDENTES:

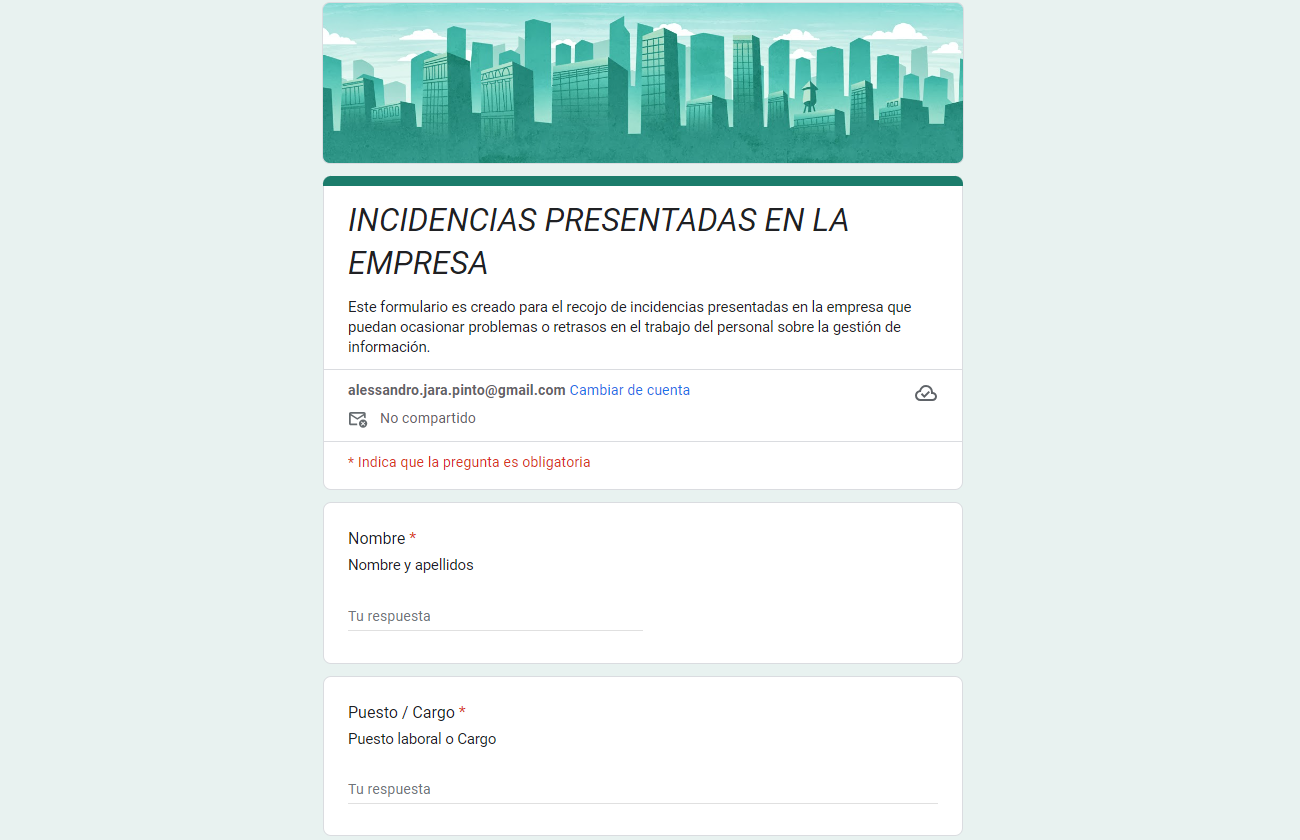
La empresa Beton Decken SAC, actualmente enfrenta un desafío significativo para la gestión de su documentación e información al depender exclusivamente de Excel como principal herramienta. La dependencia de esta herramienta para la gestión de datos y documentación plantea una serie de desafíos que afectan negativamente la eficiencia y la efectividad de la organización. Los problemas más significativos que se enfrentan son:

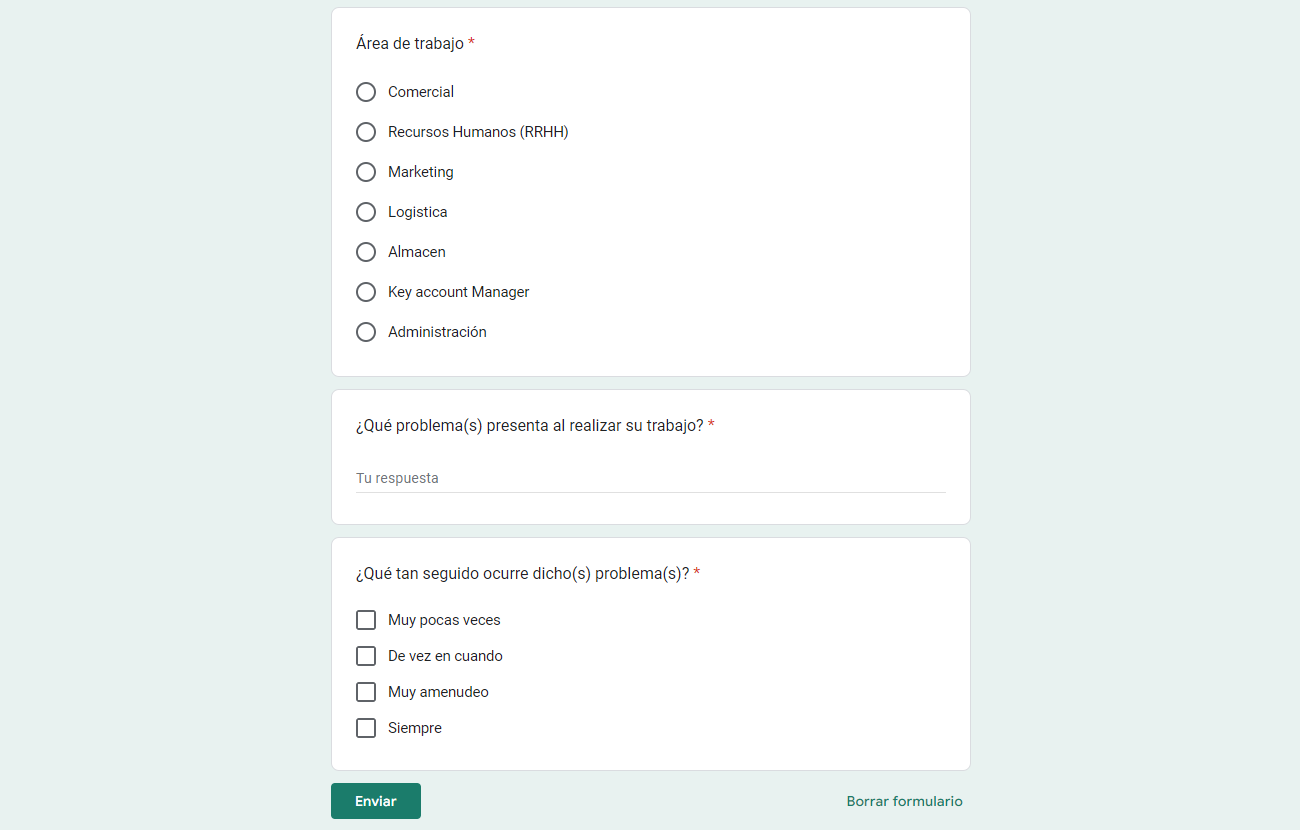
**Dificultades en la segmentación y personalización:** Un sistema deficiente puede no proporcionar datos precisos sobre los clientes, lo que llevará a dificultar la segmentación y la personalización de las estrategias de marketing y ventas.

**Riesgo de errores y pérdida de datos:** Un sistema inadecuado puede aumentar el riesgo de errores humanos y pérdida de datos críticos, lo que puede afectar la confiabilidad de la información comercial.

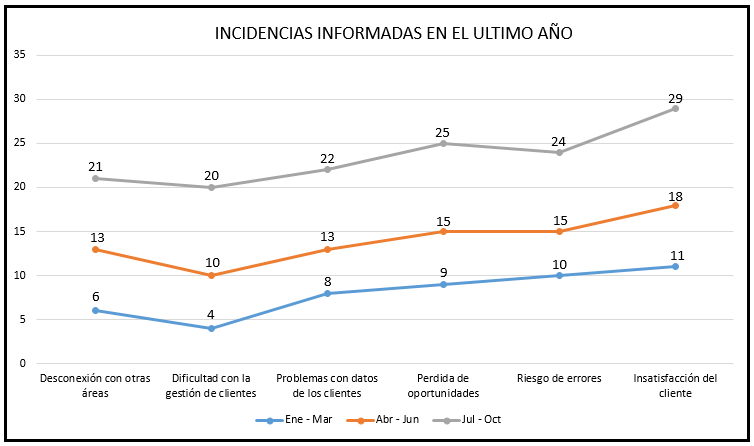
# II.I. II. RECOGO DE DATOS:

Se solicitó a las áreas de la empresa el completar un cuestionario para obtener una gráfica estadística sobre los problemas derivados de la falta de gestión de la información que pueden estar pasando. Al recopilar las respuestas de todo el personal de manera sistemática, se logrará obtener una visión integral de los problemas existentes. Esto nos permitirá visualizar patrones, áreas críticas y tendencias, facilitando así la toma de decisiones informadas para implementar mejoras en la gestión de la información y, en última instancia, optimizar el rendimiento global de la empresa.





Luego de analizar las respuestas obtenidas en el formulario. Se procedió a recopilar las incidencias en grupos de 3 meses. Pasando a plasmarlo en un diagrama que nos ayudara a determinar el aumento de incidencias ocurridas en cada 3 meses.



# II.I. III. PROPUESTA INNOVADORA:

La empresa tiene como objetivo a corto plazo realizar la implementación de un sistema de programación en las áreas afectadas para mejorar la gestión de información. Se podría considerar la creación de un programa personalizado en Intellij, una plataforma de desarrollo de software de alto rendimiento. Este programa estaría diseñado específicamente para la gestión de información comercial y la optimización de procesos. Además, el programa ofrecerá herramientas de análisis avanzado que permitirán a la empresa generar informes y análisis efectivos para una toma de decisiones basada en datos. La personalización de esta solución en Intellij garantiza la eficiencia en la gestión de clientes, segmentación y personalización, así como la automatización de tareas para evitar la pérdida de oportunidades de ventas

# II.II. OBJETIVO DEL SISTEMA

# II.II.I OBJETIVO GENERAL

* + Desarrollar un sistema de programación en Intellij de lenguaje Java que mejorará la gestión de información de la empresa Beton Decken SAC.

# II.II.II. OBJETIVO ESPECÍFICO

* + Diseñar pantallas del sistema de programación en Intellij de lenguaje Java que mejorará la gestión de información de la empresa Beton Decken SAC.
  + Programar pantallas del sistema de programación en Intellij de lenguaje Java que mejorará la gestión de información de la empresa Beton Decken SAC.
  + Realizar pruebas de las pantallas del sistema de programación en Intellij de lenguaje Java que mejorará la gestión de información de la empresa Beton Decken SAC.

# II.III. LISTA DE FUNCIONALIDADES O HISTORIAS DE USUARIO O CASOS DE USO, PRIORIZADS

# II.III.I. FUNCIONALIDAD

# II.III.I.I. IMPRIMIR INFORMACIÓN DE PROYECTOS

La tarea 'Imprimir información de proyecto' implica desarrollar un proceso que sistemáticamente recorra una lista o base de datos que almacena la información detallada de los clientes en un proyecto. Este proceso se diseña de manera eficiente y estructurada para presentar de forma ordenada los detalles asociados a cada cliente.

Para llevar a cabo esta tarea, se implementará un bucle o iterador que recorra la lista de clientes, extrayendo y mostrando diversos atributos como nombres, números de cliente, direcciones y otros datos relevantes. La ejecución de este proceso proporcionará una salida organizada y comprensible que resume la información clave de cada cliente en el contexto del proyecto

# II.III.I.II. IMPRIMIR CLIENTES POR TIPO DE PROYECTO

La tarea 'Imprimir clientes por tipo de proyecto' implica la creación de un proceso especializado que filtre y presente la información detallada de los clientes según el tipo de proyecto en el que están involucrados (PRELOSA, PREVIGA, PARAPETO, FRISO). Este proceso se diseña de manera eficiente y estructurada para proporcionar una visión organizada de los clientes asociados a cada tipo de proyecto.

Para llevar a cabo esta tarea, se implementará un mecanismo de filtrado que identifique y agrupe a los clientes según el tipo de proyecto en el que participan. Posteriormente, se utilizará un bucle o iterador para recorrer cada grupo de clientes y mostrar sus respectivos detalles, incluyendo nombres, números de cliente, direcciones y otros atributos relevantes. La ejecución de este proceso facilitará una presentación clara y específica de los clientes en función de los distintos tipos de proyecto

# II.III.I.III. IMPRIMIR ENCARGADOS POR TIPO DE PROYECTO

La tarea "Imprimir encargados por tipo de proyecto" implica desarrollar un proceso especializado que permita filtrar y presentar la información detallada de los encargados según el tipo de proyecto en el que están asignados. Este proceso se diseña de manera eficiente y estructurada para ofrecer una visión organizada de los encargados asociados a cada tipo de proyecto.

Para llevar a cabo esta tarea, se implementará un mecanismo de clasificación que identifique y agrupe a los encargados según el tipo de proyecto al que están asignados. Posteriormente, se utilizará un bucle o iterador para recorrer cada grupo de encargados y mostrar sus respectivos detalles, incluyendo nombres, roles, información de contacto y otros atributos relevantes. La ejecución de este proceso facilitará una presentación clara y específica de los encargados en función de los distintos tipos de proyecto.

# II.III.I.IV. IMPRIMIR PROYECTOS POR ENCARGADO

La tarea "Imprimir proyectos por encargado" implica desarrollar un proceso especializado que permita filtrar y presentar la información detallada de los proyectos en los que está involucrado cada encargado. Este proceso se diseña de manera eficiente y estructurada para ofrecer una visión organizada de los proyectos asociados a cada encargado.

Para llevar a cabo esta tarea, se implementará un mecanismo de clasificación que identifique y agrupe los proyectos según el encargado responsable. Posteriormente, se utilizará un bucle o iterador para recorrer cada grupo de proyectos y mostrar sus respectivos detalles, incluyendo nombres de proyectos, tipos, fechas de inicio y finalización, y otros atributos relevantes. La ejecución de este proceso facilitará una presentación clara y específica de los proyectos en función de los encargados asignados.

# II.III.I.V. IMPRIMIR DETALLES POR CÓDIGO DE PROYECTO

La tarea "Imprimir detalles por código de proyecto" implica desarrollar un proceso especializado que permita filtrar y presentar la información detallada de un proyecto en particular identificado por su código único. Este proceso se diseña de manera eficiente y estructurada para ofrecer una visión organizada de los detalles asociados a un proyecto específico.

Para llevar a cabo esta tarea, se implementará un mecanismo de búsqueda que identifique el proyecto según su código único. Posteriormente, se imprimirán los detalles específicos de ese proyecto, que pueden incluir nombre, tipo, fecha de inicio, fecha de finalización, encargados asignados, y otros atributos relevantes. La ejecución de este proceso proporcionará una presentación clara y detallada de la información asociada a un proyecto específico identificado por su código único.

# II.III.I.VI. IMPRIMIR PROYECTO CON MAYOR COSTO

La tarea "Imprimir proyecto con mayor costo" implica desarrollar un proceso especializado que permita identificar y presentar la información detallada del proyecto que tiene el mayor costo entre todos los proyectos. Este proceso se diseña de manera eficiente y estructurada para ofrecer una visión organizada de los detalles asociados al proyecto con el costo más alto.

Para llevar a cabo esta tarea, se implementará un mecanismo de comparación que evalúe el costo de cada proyecto y determine cuál tiene el valor más alto. Posteriormente, se imprimirán los detalles específicos de ese proyecto, que pueden incluir nombre, tipo, fecha de inicio, fecha de finalización, encargados asignados, y otros atributos relevantes. La ejecución de este proceso proporcionará una presentación clara y detallada de la información asociada al proyecto con el mayor costo.

# II.III.II. HISTORIAS DE USUARIOS

# II.III.II.I IMPRIMIR INFORMACIÓN DE PROYECTOS

Como usuario que trabaja con el sistema día a día, quiero poder echar un vistazo rápido y bien organizado a la lista completa de clientes; para tener una idea clara de quiénes son y, al mismo tiempo, quiero poder hacer clic fácilmente para manejar las facturas de cada uno. La idea es que la experiencia sea fluida y sin complicaciones; no solo quiero ver quiénes son los clientes, sino también tener la capacidad de lidiar con sus facturas de manera directa. La idea es simplificar mis tareas diarias, haciendo que la gestión de clientes y las operaciones relacionadas con las facturas sean tan fáciles como tomar un café por la mañana.

# II.III.II.II. IMPRIMIR CLIENTES POR TIPO DE PROYECTO

En mi rol como usuario comprometido con la gestión de proyectos, necesito la funcionalidad de imprimir una lista organizada de clientes clasificados por tipo de proyecto. Esta característica me permitiría obtener una visión rápida y estructurada de los clientes asociados a proyectos específicos, lo que facilitaría la identificación de relaciones y la comprensión de cómo se distribuyen los clientes en diferentes categorías de proyectos. Al imprimir esta información, espero contar con un recurso tangible o digital que pueda utilizar para análisis detallados y discusiones estratégicas con el equipo. La meta es mejorar la eficiencia en la toma de decisiones al tener acceso rápido y fácil a datos clave sobre la relación entre clientes y tipos de proyectos.

# II.III.II.III. IMPRIMIR ENCARGADOS POR TIPO DE PROYECTO

En mi rol como usuario comprometido con la gestión de proyectos, necesito la funcionalidad de imprimir una lista organizada de encargados clasificados por tipo de proyecto. Esta característica me permitiría obtener una visión rápida y estructurada de los encargados asociados a proyectos específicos, facilitando la identificación de responsabilidades y la comprensión de cómo se asignan los encargados a diferentes categorías de proyectos. Al imprimir esta información, espero contar con un recurso tangible o digital que pueda utilizar para análisis detallados y discusiones estratégicas con el equipo. La meta es mejorar la eficiencia en la asignación de tareas y la toma de decisiones al tener acceso rápido y fácil a datos clave sobre la relación entre encargados y tipos de proyectos.

# II.III.II.IV. IMPRIMIR PROYECTOS POR ENCARGADO

Como usuario activo en la gestión de proyectos, busco la capacidad de imprimir una lista organizada de proyectos clasificados por encargado. Esta función sería esencial para obtener una visión rápida y bien estructurada de los proyectos asignados a cada encargado, permitiéndome evaluar la carga de trabajo individual y la distribución equitativa de responsabilidades. Al imprimir esta información, deseo contar con un recurso tangible o digital que pueda utilizar para análisis detallados y discusiones estratégicas con el equipo. El objetivo es mejorar la eficiencia en la asignación de tareas y la toma de decisiones al tener acceso rápido y fácil a datos clave sobre la relación entre proyectos y encargados.

# II.III.II.V. IMPRIMIR DETALLES POR CÓDIGO DE PROYECTO

Como usuario dedicado a la gestión de proyectos, necesito la funcionalidad de imprimir detalles específicos organizados por código de proyecto. Esta característica sería esencial para obtener una representación impresa o digital de información detallada sobre un proyecto en particular, incluyendo hitos, recursos asignados, plazos y cualquier otro detalle relevante. Al imprimir esta información, espero contar con un recurso fácilmente accesible que me permita revisar y compartir detalles cruciales del proyecto de manera eficiente. El propósito es agilizar la comunicación y facilitar la toma de decisiones informadas al tener una representación tangible de los detalles clave asociados a un código de proyecto específico.

# II.III.II.VI. IMPRIMIR PROYECTO CON MAYOR COSTO

Como usuario involucrado en la gestión de proyectos, busco la capacidad de imprimir la información detallada del proyecto con el mayor costo. Esta función sería esencial para obtener una visión rápida y bien estructurada de los detalles del proyecto que incurre en el costo más significativo. Al imprimir esta información, deseo contar con un recurso tangible o digital que me permita analizar a fondo los aspectos financieros del proyecto y facilitar la toma de decisiones relacionadas con la gestión de costos. El objetivo es optimizar la eficiencia en la revisión de proyectos al tener acceso rápido y fácil a los detalles clave de aquellos con mayores implicaciones económicas.

# II.III.III. CASOS DE USO

* La empresa requiere que el sistema debe permitir visualizar una lista de opciones que para luego permitir la selección de una de ellas que sea válida, en cambio, si se escoge una opción no válida el sistema se terminará o si se selecciona la opción 0.
* Para el buen desarrollo de la empresa, la empresa requiere que el sistema debe recopilar los datos de los clientes almacenados en una lista que se haya creado, para luego permitir listarlos y mostrar una impresión de la lista de todos los clientes.
* La empresa requiere que el sistema realice un listado de los proyectos que se hayan realizado a cada producto desde los inicios de la empresa, para luego solicitar una búsqueda de los proyectos de cada producto de la empresa, y luego se imprima una lista simplificada de lo buscado.
* La empresa requiere que el sistema pueda solicitar al usuario ingresar los datos de un cliente, y esto realice una búsqueda en la lista de los clientes, para luego el sistema permita la modificación de sus datos (nombre, apellido, edad, etc) por si se requiere realizar la modificación del usuario del cliente.
* La empresa requiere que el sistema pueda crear una lista para almacenar y luego realizar una búsqueda en la lista de los clientes a través del RUC, para que luego muestre los datos del cliente para el área de Comercial de la empresa.
* La empresa requiere que el sistema pueda solicitar al usuario el ingreso de un dato específico del producto que es vendido, para que luego pueda imprimir esa misma lista de productos simplificado en la cantidad restante y arroje el dato de mayor venta del producto solicitado.

# II.IV. CRONOGRAMA Y ASIGNACIÓN DE ACTIVIDADES



* **Semana 1-2**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

En la semana 1 y la semana 2 se definirá los objetivos y alcances proporcionándonos una guía clara sobre lo que queremos lograr, para ello recolectaremos requisitos, se realizará entrevista identificando los objetivos generales y dando una priorización.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

La documentación es fundamental ya que nos va proporcionar una guía clara, se realizará entrevista funcionales, la documentación de funciones será principal ya que explicara el propósito de los parámetros esperados, se usará gráficos y diagramas para representar la arquitectura del sistema y se deberá ser actualizada a medida se va evolucionando el programa.

* **Semana 2 – 4**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Entre la semana 2 a la semana 4 se iniciará con la planificación esto será crucial ya que nos ayudará a entender los requisitos del proyectos, se dividirá las tareas esta descomposición ayudará a estimar mejor el tiempo, se realizará cronograma de desarrollo identificando hitos claves en el proyecto

* **Semana 2 – 5**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Entre la semana 2 a la semana 5 se comenzará a codificar que se implementará los datos que se deben manejar con restricciones, se incluirá un entorno de desarrollo configurando correctamente las herramientas y bibliotecas que usaremos, se digitará el código según la planificación y el diseño, se hará revisiones de códigos para obtener comentarios y ver si se cumple con el requerimiento previsto

* **Semana 2 – 6**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Entre la semana 2 a la semana 6 se dará con la creación del diseño , identificando componentes de disposición de los elementos en la navegación y la apariencia visual asegurando que la experiencia sea intuitiva y eficiente, se diseñara un modelo de datos donde definiremos las tablas, relaciones y restricciones necesarias, se usará diagramas para representar la arquitectura como el UML .

* **Semana 7**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

En la semana 7 se dará se realizará pruebas finales para asegurar el funcionamiento del software, obteniendo la aprobación formal del cliente , se completará y se revisará toda la documentación del proyecto , incluyendo manuales de usuario, documentación técnica y por ultimo se elaborará un informe final que resuma el proyecto, logros y desafíos.

# CAPÍTULO 2: PROPUESTA DE INNOVACIÓN

# III.I Diseño de Pantallas

Dispondremos de 6 estructuras de diseño de pantalla, 3 Packege y 4 Clases y 2 Main (un Modelo y una Vista), para nuestro proyecto:

Ejecutado para nuestro trabajo final

Desplegado desde la composición de los tres package (controlador, modelo, y vista)

Pantalla de computadora

Descripción generada automáticamente

# III.I.I Controlador

Ejercerá el papel de la estructura de todos los proyecto, clientes, empresas y encargo que ofrece Beton Decken en su catálogo de producto, por lo que se implementará del debido registro y uso correspondiente de sus atributos, constructor, getter and setter y creación de operaciones de tipo public void.

Todos los Java class estarán directamente asociado con los Proyectos(extends), en representación a la base de datos con la que trabajaremos, permitiendo maximizar la visualización de todo los registros ofrecidos para la implemtación del Sistema de Gestión de la Información.

## **III.I.I.ICliente:**

La clase Cliente nos va ayudar a representar la información asociada de un cliente hacia el sistema



Indica que la clase Cliente pertenecerá al paquete **“org.example.Contralador”** la cual nos ayudará a que las clases sean relacionadas con el controlador y que la interacción entre el usuario y la lógica sea coordinada.



El código **“import java. util.ArrayList”** se usará para importar clases y así poder usarlas en el código



Iniciaremos declarando una clase la cual se llamará **“cliente”** y será de acceso público

Texto

Descripción generada automáticamente

A la clase **Cliente** le asignaremos 5 **Atributos** la cual será privados de la clase.

Usaremos la variable **String** ya que almacenaremos información tipo texto

Los 5 atributos son los siguientes:

**Private String dirección:** se pedirá la dirección del cliente para poder realizar entregas, se usará la variable **String** ya que es una información textual y que puede incluir números, letras y caracteres especiales. Esta variable se agregará para tener un registro de los clientes y de facturas.

**Private String tipoDocumento:** en este caso se almacenará DNI, pasaporte, cedula, etc. Se usará **String** ya que se almacenará palabras y/o abreviaturas. Esta variable se agregará para saber que tipo de documento tiene

**Private String numeroDocumento:** se almacenará el número de los documentos, esto servirá para tener una información más acertada del cliente que buscamos y/o agregamos. Se usará **String** ya que se está tomando como texto ya que no se realizará ningún tipo de operación aritmética. Esta variable se agregará para tener un registro de los compradores y tengan un único código.

**Private List<Proyecto> proyectosSolicitados:**  en este caso se usará para almacenar una lista de objetos de la clase **Proyecto**, el tipo **List** indicará que puede contener aun varios elementos y servirá para mantener una colección de proyectos a cliente.

****

Este código se corresponde al constructor de la clase **Cliente,** se llamará cuando se cree una nueva instancia de la clase Cliente y será usado para iniciar los valores proporcionados.

**Texto

Descripción generada automáticamente**

Se usará la palabra clave **this** para poder hacer una referencia a los tributos de la instancia actual de la clase, cada objeto que se creará tendrá su propio atributo.

La asignación de valores a los atributos dentro del constructor asegura que los objetos **Cliente** tengan la información necesaria tan pronto como se crean.

**this. nombre = nombre:** Asigna el valor del parámetro nombre (que es el nombre pasado al constructor) al atributo nombre de la instancia actual de la clase Cliente. this. nombre se refiere al atributo nombre de la instancia actual.

**this.direccion = direccion:** Similar al anterior, asigna el valor del parámetro direccion al atributo direccion de la instancia actual de la clase Cliente.

**this.tipoDocumento = tipoDocumento:** Asigna el valor del parámetro tipoDocumento al atributo tipoDocumento de la instancia actual de la clase Cliente.

**this.numeroDocumento = numeroDocumento:** Asigna el valor del parámetro numeroDocumento al atributo numeroDocumento de la instancia actual de la clase Cliente.



Crearemos una nueva instancia de la clase “**ArrayList”** se le asignara el atributo **proyectosSolicitados** para que se pueda garantizar la lista de proyectos y está lista para su uso

Texto

Descripción generada automáticamente

**getNombre:** será un método que no toma ningún tipo de argumento y devolverá un valor de tipo **String**

**Return nombre:**  esto nos indica que cuando llame al método **getNombre** nos devolverá el valor actual del atributo que vendría a ser **nombre** de la instancia

## **III.I.I.II Empresa:**

El Java classs Empresa, ejerce de contenedor para la habilitación de las listas asociadas con las Clases Proyecto y Cliente, Permitiendo la primera generación de registro de datos contenidos, atravez de la operación de agregar nuevos registros a las listas (private List<Proyecto> proyectos; private List<Cliente> clientes)

Captura de pantalla de computadora

Descripción generada automáticamente

La funcionalidad de la Clase Empresa, contiene atributos de tipo String y uso de List<> permitiendo emplearlo dentro de las siguientes operaciones Void. Se hace del uso de generador de constructor dentro del código.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## **Voids:**

Todo proceso se requiere del adecuado manejo y comprensión de la relevancia de los public voids.

**1er Proceso** de agregado en listas de Proyecto y Cliente:

Texto

Descripción generada automáticamente

**2do Proceso**

Imprenta multidinamica dependiendo de la necesidad:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Impresión masiva habilitada para todos los proyectos. Solicitando representación directa con la Clase Proyecto, realizando un llamado get. a todos atributos de proyecto definido si es que están contenidos dentro de la lista habilitada en su registro, caso contrario será adicionado. Definimos un List<String> proyectosImpresos, gestionando “for” la recopilación de todos los registros validos de la lista proyectos, y siguiendo con el proceso requeriremos de un atributo nuevo que cumple el plan de representación **String proyectoInfo,** su funciónno solo tiene el papel de devolver los valores p.getting de la Clase Proyecto, sino que cumple de operación denominante dentro de la siguiente condicional “if”, y seguido del uso aplicado de souf que permitirá obtener todos los resultados solicitados del proyecto.

Texto

Descripción generada automáticamente

Tipo de funcionalidad de imprimir los registros habilitado por tipo de proyecto, generando una lista para este proceso, siguiendo las mismas funcionalidad de la primera impresión excepto que se realizara un get. a los atributos p,.getNombre() y p.getCliente().getNombre(). La funcionalidad del proceso de secuencia for se debe a la asociación de todo contenido referente con el Proyecto, lista y atributos, directamente correlacionado con la sintaxis que permite devolver la información reflejado dentro de los proyectos clasificador, por su tipo de producto ofrecido.

Texto

Descripción generada automáticamente

Operación de impresión aplicado en encargados por tipo de proyecto, implementando funcionalidades similares con los anteriores procesos, permitiendo devolver la información reflejada en dentro de la clase Proyecto, lista y atributos directamente asociados con el Encargado.

* Si no se encuentra ningún proyecto con ese tipo, muestra un mensaje indicando que no se encontró el tipo de proyecto especificado.
* El análisis paso a paso del método **imprimirEncargadosPorTipoProyecto:**
* Parámetro de entrada**: tipoProyecto** es el tipo de proyecto que se utilizará como criterio de búsqueda.
* Creación de variables locales:
* **encargadosImpresos:** Un **ArrayList** que almacena información sobre los encargados de los proyectos ya impresos para evitar imprimir duplicados.
* **proyectoEncontrado:** Se establece en verdadero si se encuentra al menos un proyecto con el tipo especificado.
* Bucle **for:** Recorre la lista de proyectos (proyectos), donde cada Proyecto se representa con la variable **p.**
* Condición **if** dentro del bucle: Verifica si el tipo de proyecto del proyecto actual (p) coincide con el **tipoProyecto**

Texto

Descripción generada automáticamente

Imprimir registro directamente asociados con nombre del encargado.

Este método en Java, **imprimirProyectosPorEncargado,** toma un parámetro **encargadoEncontrado** y busca en una lista de **Proyecto (proyectos).**

También maneja algunas excepciones como **NullPointerException.**

El proceso aplicado del método refleja:

**Parámetro de entrada:** **nombreEncargado** es el nombre del encargado que se utilizará como criterio de búsqueda.

**Creación de variables locales:**

* **encargadoEncontrado:** Una bandera booleana que se inicializa en falso y se establece en verdadero si se encuentra al menos un proyecto relacionado
* **Bloque try-catch:** Este método utiliza un bloque try-catch para manejar posibles excepciones que puedan ocurrir durante la ejecución del código. En particular, maneja la excepción **NullPointerException.**
* **Bucle for:** Recorre la lista de proyectos (proyectos), donde cada Proyecto se representa con la variable p.
* **Condición if** dentro del bucle: Verifica si el encargado del proyecto actual (p) no es nulo,

Texto

Descripción generada automáticamente

Impresión habilitada en búsquedas asociadas con el proyecto de mayor costo de venta.

Habilitando operaciones for and if además de las listas asociadas con proyecto y atributos contenidos en la clase.

Este código en Java implementa un método llamado **imprimirProyectoConMayorCosto()**, que recorre una lista de proyectos **(proyectos)** para encontrar el proyecto con el mayor costo de venta, basándose en un método **calcularCostoVenta()** que devuelve el costo de venta de cada proyecto.

Texto

Descripción generada automáticamente

Motor de búsqueda asociado con los registros por código, especificando multiples de impresiones aplicadas a la clase Proyecto, y componentes asociados permitiedno estructurar los multiples escenarios y pruebas de documentación.

**Variables locales:**

* **codigosImpresos: Un ArrayList** que almacena los códigos de los proyectos ya impresos para evitar mostrar información duplicada.
* **Bloque try-catch:** Este método utiliza un bloque try-catch para manejar posibles excepciones que puedan ocurrir durante la ejecución del código.
* **Bucle for:** Recorre la lista de **Proyecto (proyectos),** donde cada Proyecto se representa con la variable proyecto.
* Condición dentro del bucle: Verifica si el código del proyecto actual **(proyecto.getCodigo())** coincide con el **codigoBuscado** proporcionado como argumento y si este código no ha sido impreso previamente **(!codigosImpresos.contains(codigoBuscado)).**

## **III.I.I.III Encargado:**

Esta clase nos representa a la persona a cargo del proyecto, tendremos información de su nombre, cargo y el tipo de proyecto que se le asignará



Esta declaración nos especifica que la clase **Encargado** pertenece al paquete **org.example.Controlador**



Crearemos la clase llamado **Encargado** de forma **publica** para que sea accesible desde cualquier otra clase.

Texto

Descripción generada automáticamente

Se creará los atributos de clase privada, el atributo nombre nos indicará el nombre del encargado, se le asignará un cargo y a qué tipo de proyecto este asignado.

Texto

Descripción generada automáticamente

El siguiente código es un constructor de la clase llamado **Encargado,** iniciará los atributos de un objeto.

**public Encargado (String nombre, String cargo, String tipoProyecto):**

es una declaración pública lo que significa que otra clase puede acceder, se tomará tres parámetros de tipo **String**

Texto

Descripción generada automáticamente

El método **getnombre** proporcionara para el acceso público para obtener el valor del atributo **nombre.**

La instrucción **Return**  se usará para devolver el valor del atributo **nombre**

## **III.I.I.IV Proyecto:**

La clase proyecto nos servirá para saber la obtención del factor proyecto y el calculo del costo de venta, también obtendremos información exacta sobre los proyectos.



Usaremos este paquete para organizar y estructurar el código, este código nos indicará que el archivo pertenece al paquete **org.example.Controlador y** nos ayudará a reflejar los directorios en el sistema almacenado.



Se declarará una clase con acceso público la cual lo llamaremos proyecto.

Texto

Descripción generada automáticamente con confianza media

Se declarará los siguientes atributos de modo privado que ayudará a representar información del proyecto, estos atributos sólo serán accesibles dentro de la propia clase

**private String código**: almacenará un código que servirá como identificador para cada proyecto, será de tipo **String** ya que será de uso textual

**private String nombre:** Almacenará el nombre del proyecto, será de tipo **String** ya que es de carácter textual.

**private int anioInicio:** Almacenará el año en el que proyecto se inició, será de tipo **int** ya que almacenará un valor numérico enteros

**private Encargado encargado:** es una referencia a un objeto de la clase encargado

**private Cliente cliente:**  será usado como referencia a un objeto de clase cliente

**private String tipoProyecto:**  almacenará el tipo de proyecto como **prelosa, previga, parapeto, friso**. Será de tipo **String** ya que almacenará valores de carácter textual.

**private String nombreempresa:**  almacenará el nombre de la empresa la cual será asociada

**private double metradoTotal:**  almacenará el metraje total del proyecto, usaremos el valor de tipo **double** ya que se almacenará valores numéricos decimales

**Texto

Descripción generada automáticamente**

Se definirá un constructor que inicializará los atributos de la clase Proyecto., estos aseguran que tenga un estado inicial específico para que pueda ser usado.

Texto

Descripción generada automáticamente

Los siguientes códigos están destinados para iniciar los tributos de la clase proyecto, usaremos la clave this ya que nos ayudará hacer referencia al objeto de la propia clase.

Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente

Los siguientes códigos se usará para acceder y modificar los atributos de la clase, usaremos los métodos Getters y Setters

**Getters (Métodos de Obtención):** lo usaremos para obtener el valor actual de los atributos.

**Setters (Métodos de Establecimiento):** este método se usa para establecer el valor de los atributos

Texto

Descripción generada automáticamente

El código obtenerFactorPr nos ayuda a calcular y devolver un valor numérico basado en el **tipo de proyecto**, se usará la estructura **switch** para poder seleccionar un valor especifico dependiendo del valor **tipoProyecto**

Dentro de la Estructura switch encontraremos casos de selección del tipo de proyecto prelosa, previga, parapeto, friso en cada una de ellas nos indicara un valor de devolución.

Texto

Descripción generada automáticamente

Este método nos ayudará a calcular el costo del proyecto, se multiplicará el metrado total por el factor obtenido

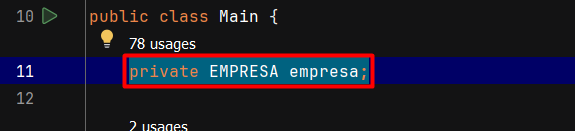
Usaremos el Tipo double ya que usaremos el valor numérico decimal.

# III.I.II Modelo

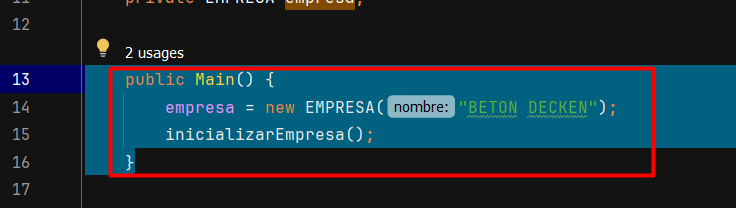
# III.I.II.I Main

En la clase “Main” que está ubicado en el paquete “Modelo” se tiene diferentes métodos, atributos, constructores, objetos, etc. Los cuales describiremos y explicaremos su función en el sistema.

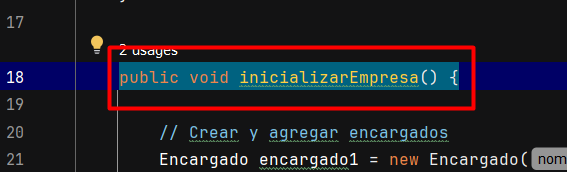
La primera linea de codigo que encontramos es la siguiente “private EMPRESA empresa”, el cual encontramos que se tiene un “private” que es un modificador de acceso dentro de la misma clase de una variable. Lo siguiente que encontramos es una variable el cual es “empresa” y esta cumplirá la función de almacenar y/o usar objetos que se puedan encontrar dentro de la clase “EMPRESA”.



Lo siguiente que encontramos es un constructor para la clase “Main”, el cual cumplirá la función de permitir ser accesible a otras clases. Dentro del constructor encontramos la siguiente linea de codigo “empresa = new EMPRESA("BETON DECKEN")” el cual podemos identificar que se está llamando la variable identificada anteriormente “empresa”, para luego crear una instancia de la clase “EMPRESA” y luego se pedirá identificar el nombre de la empresa “BETON DECKEN”. La siguiente linea de codigo que encontramos dentro del constructor es un método el cual es “inicializarEmpresa();” el cual cumplirá la función de llamar e inicializar algunas operaciones adicionales ubicados en la clase “EMPRESA”.

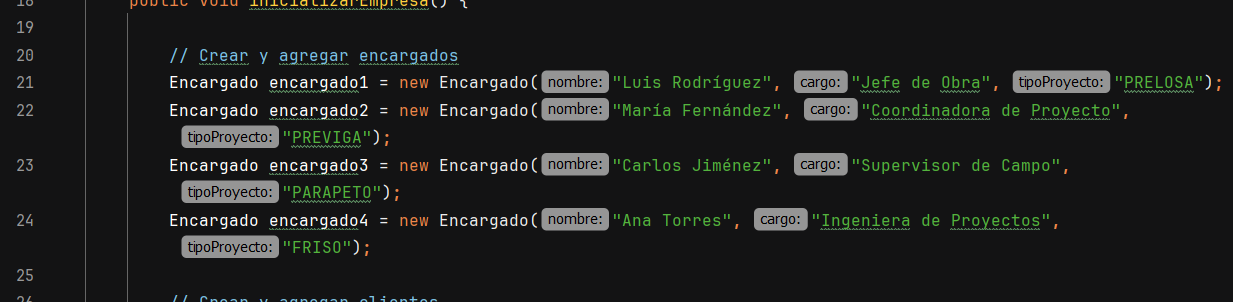


La siguiente linea de programación que encontramos es la siguiente “public void inicializarEmpresa()” el cual identificamos que tiene modificadores de acceso “public” y esto permitirá el acceso de otras clases a los datos que se ingresara dentro. Tambien identificados que el tipo de retorno es “void” y se está llamando al método “inicializarEmpresa” y este cuenta con paréntesis vacíos “()”, el cual significa que los datos ingresados a continuación serán dentro del codigo y no se solicitara al usuario el ingreso de ningún dato.



Dentro del método anteriormente mencionado encontraremos los siguientes datos ingresados:

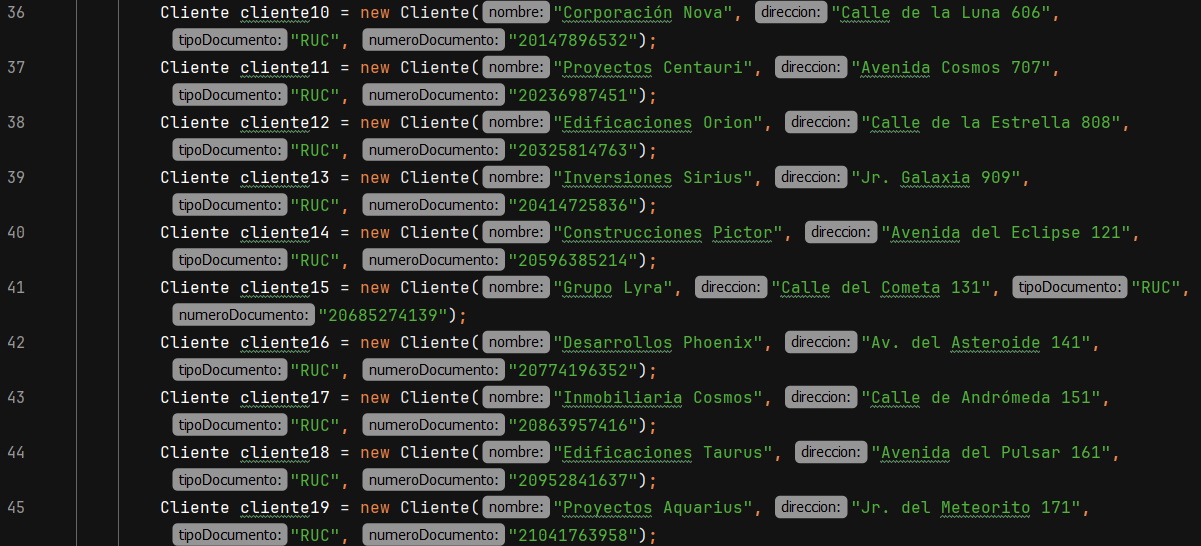
Las siguientes líneas de codigo se determinará los datos de los encargados, por ejemplo, en la primera línea de codigo identificamos “Encargado”, el cual cumple la función de llamar los campos a llenar de la clase “Encargado”, y los datos que se ingresara se guardan en el objeto “encargado1”, los cuales son el nombre del encargado que será Luis Rodríguez, con el cargo de Jefe de Obra y el tipo de proyecto que se especializa que es Pre-Losa, el objeto “encargado2” tiene los datos del encargado Maria Fernández, con el cargo de Coordinadora de Proyecto y con el tipo de proyecto que se especializa que es Pre-Viga, el objeto “encargado3” tendrá los datos de encargado Carlos Jiménez, con el cargo de Supervisor de Campo y el tipo de proyecto que se especializa que es Parapeto, y el objeto “encargado4” tendrá los datos del encargado Ana Torres, con el cargo de Ingeniera de Proyectos y el tipo de proyecto que se especializa que Friso.

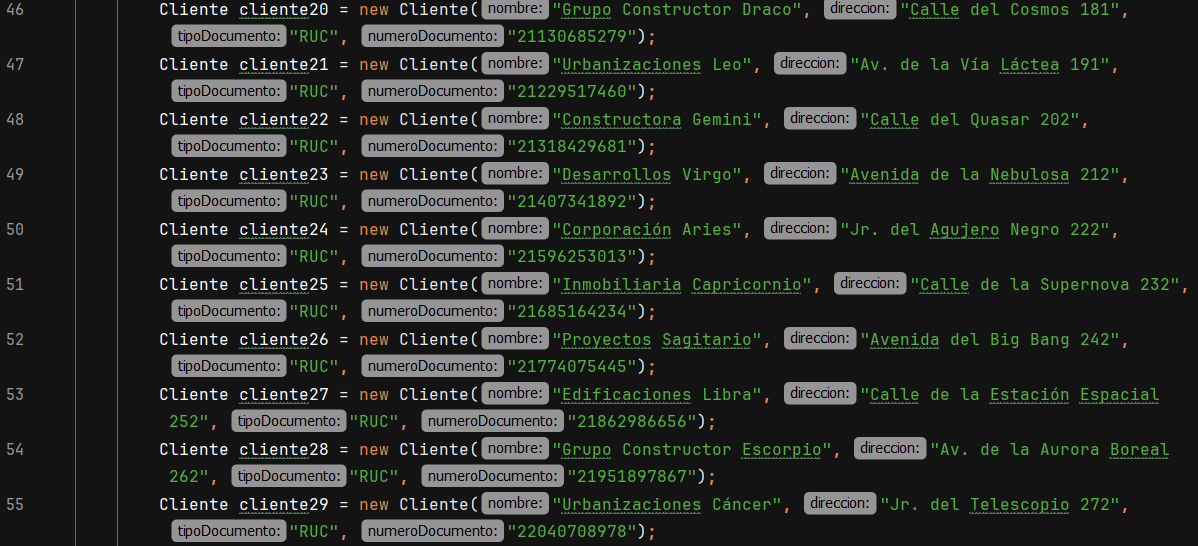


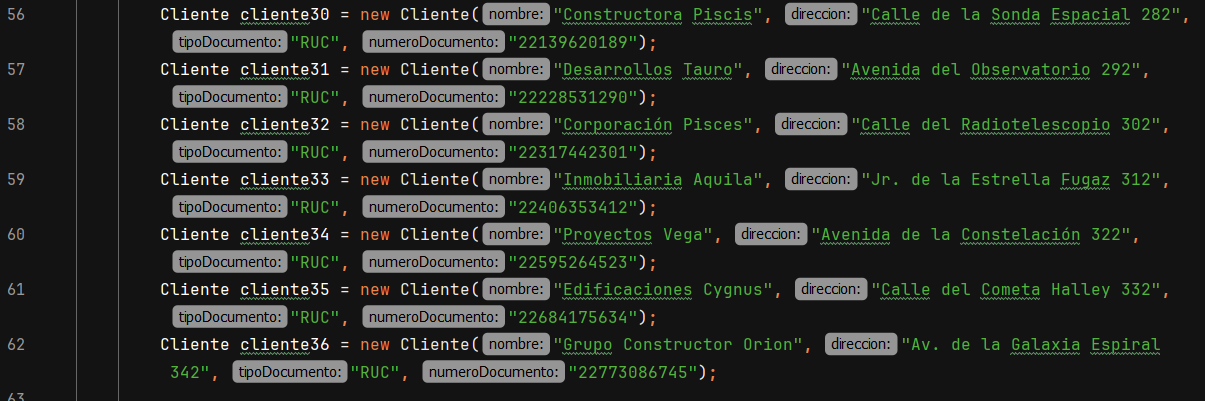
Las siguientes líneas de codigo se determinará los datos de los clientes, por ejemplo, en la primera línea de codigo identificamos “Cliente”, el cual cumple la función de llamar los campos a llenar de la clase “Cliente”, y los datos que se ingresara se guardan en el objeto “cliente1”, los cuales son el nombre del cliente que será Grupo Constructor Zeta, con la dirección Avenida Principal 123, el tipo de Documento que es RUC y el número del documento que es 20547896541, el objeto “cliente2” tiene los datos del cliente Inmobiliaria El Sol, con dirección Calle Los Laureles 456, el tipo de documento que es RUC y el número de documento que es 20587412365, el objeto “cliente3” tiene los datos del cliente Corporación Alta Vista,

con dirección Jr. Los Cedros 789, el tipo de documento que es RUC y el número de documento que es 20659874512, el objeto “cliente4” tiene los datos del cliente Desarrollos Urbanos Gamma, con dirección Av. Las Flores 321, el tipo de documento que es RUC y el número de documento que es 20541236987, y los mismos campos a completar se creara un objeto para cada cliente que guardaran sus datos, finalizando con el objeto “cliente36” que tiene los datos del cliente Grupo Constructor Orion, con dirección Av. de la Galaxia Espiral 342, el tipo de documento que es RUC y el número de documento que es 22773086745.

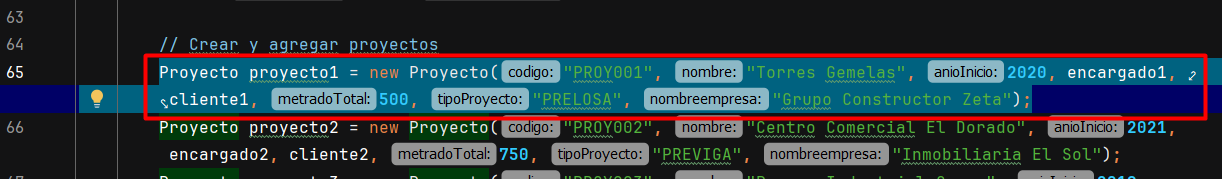


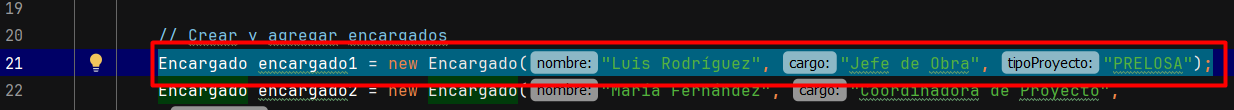






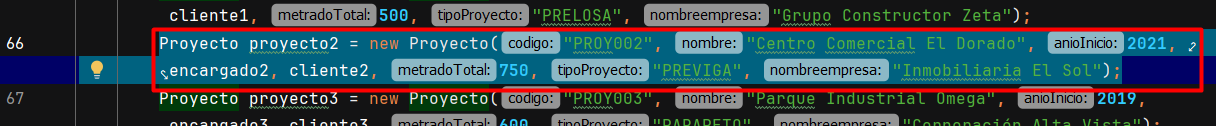
Las siguientes líneas de codigo se determinará los datos de los proyectos que se realizaron con un encargado y un cliente, por ejemplo, en la primera línea de codigo identificamos “Proyecto”, el cual cumple la función de llamar los campos a llenar de la clase “Proyecto”, y los datos que se ingresara se guardan en el objeto “proyecto1”, los cuales son el codigo del proyecto que es PROY001, el nombre del proyecto que es Torres Gemelas, el año en que se inició el proyecto que fue el 2020, el encargado responsable que es encargado1, el cliente al que se realizó el proyecto que fue al cliente1, el metrado total del proyecto que fue 500, el tipo de proyecto que fue Pre-Losa y el nombre de la empresa que se realizado el proyecto que fue Grupo Constructor Zeta.

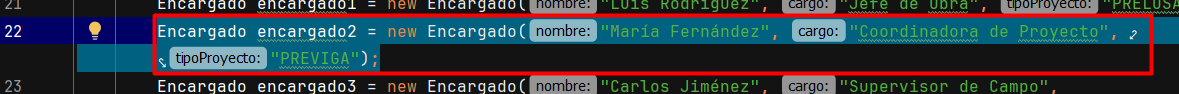


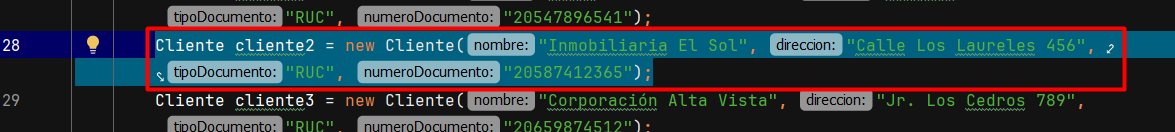




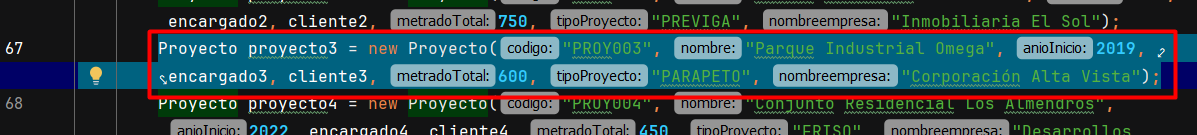
El objeto “proyecto2”, tiene los datos como el codigo del proyecto que es PROY002, el nombre del proyecto que es Centro Comercial El Dorado, el año de inició del proyecto que fue el 2021, el encargado es encargado2, el cliente del proyecto es cliente2, el metrado total es 750, el tipo de proyecto es Pre-Viga y el nombre de la empresa es Inmobiliaria El Sol.

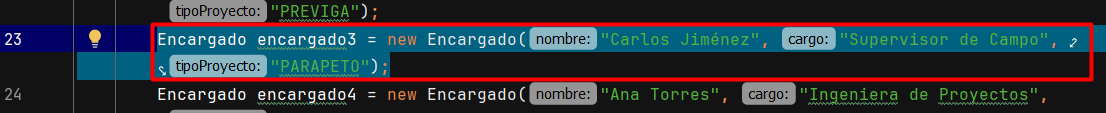


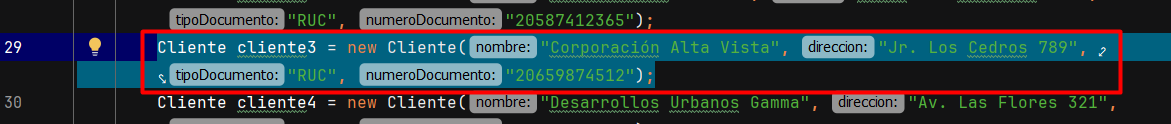




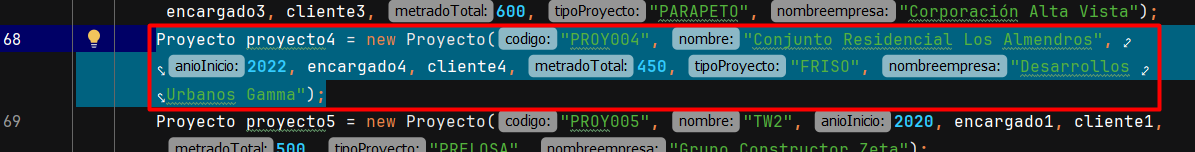
El objeto “proyecto3”, tiene los datos como el codigo del proyecto que es PROY003, el nombre del proyecto que es Parque Industrial Omega, el año de inició del proyecto que fue el 2019, el encargado es encargado3, el cliente del proyecto es cliente3, el metrado total es 600, el tipo de proyecto es Parapeto y el nombre de la empresa es Corporación Alta Vista.

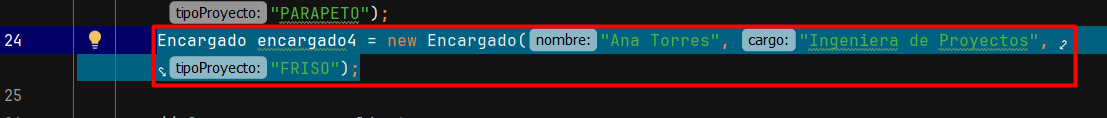


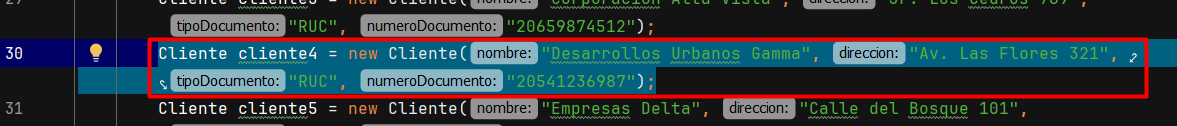




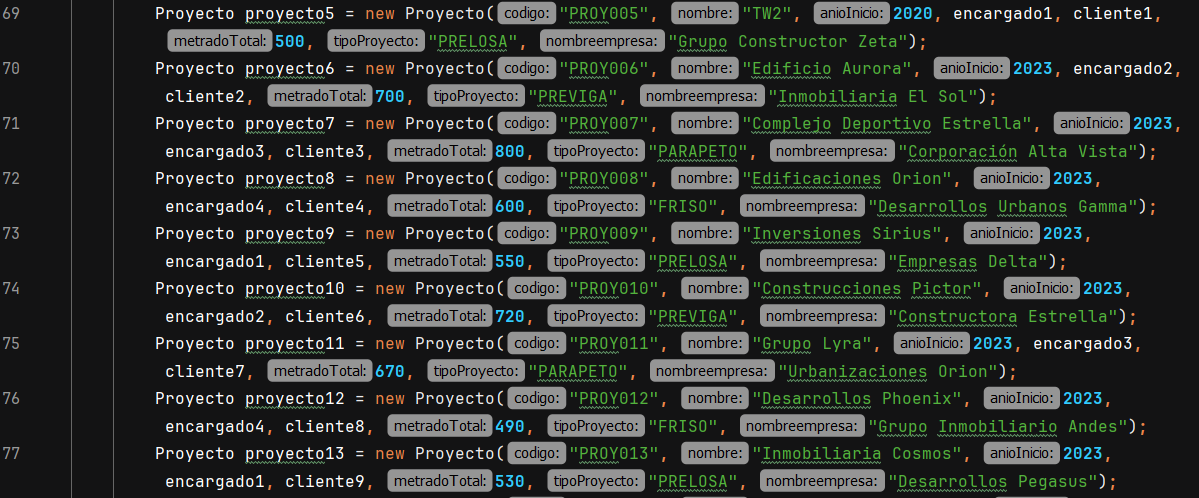
El objeto “proyecto4”, tiene los datos como el codigo del proyecto que es PROY004, el nombre del proyecto que es Conjunto Residencial Los Almendros, el año de inició del proyecto que fue el 2022, el encargado es encargado3, el cliente del proyecto es cliente3, el metrado total es 450, el tipo de proyecto es Friso y el nombre de la empresa es Desarrollos Urbanos Gamma.



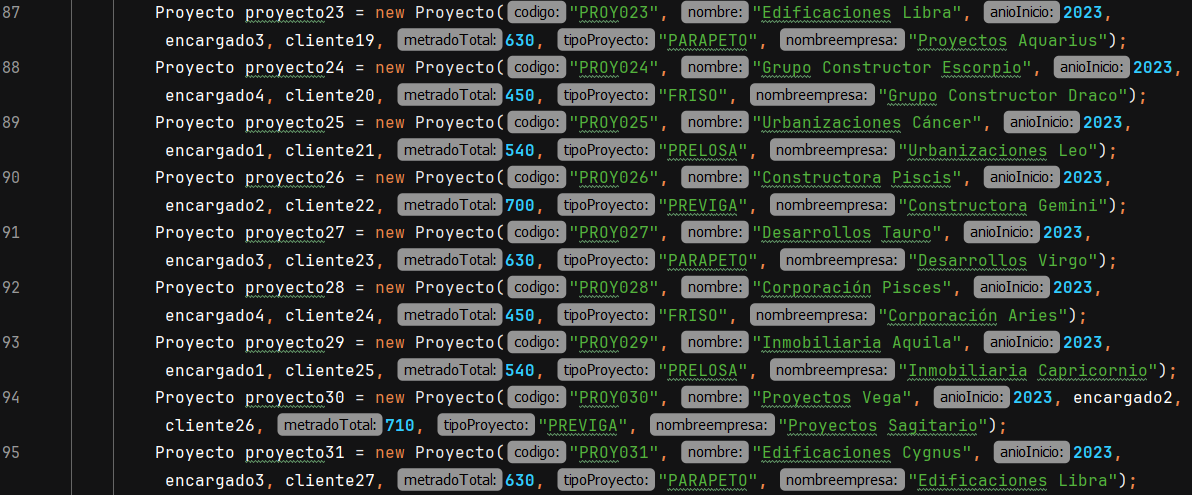




Y los mismos campos a completar se creará un objeto para cada proyecto que guardaran sus datos, finalizando con el objeto “proyecto36”, tiene los datos como el codigo del proyecto que es PROY036, el nombre del proyecto que es Plaza Comercial Galaxia, el año de inició del proyecto que fue el 2023, el encargado es encargado4, el cliente del proyecto es cliente32, el metrado total es 450, el tipo de proyecto es Friso y el nombre de la empresa es Corporación Pisces.

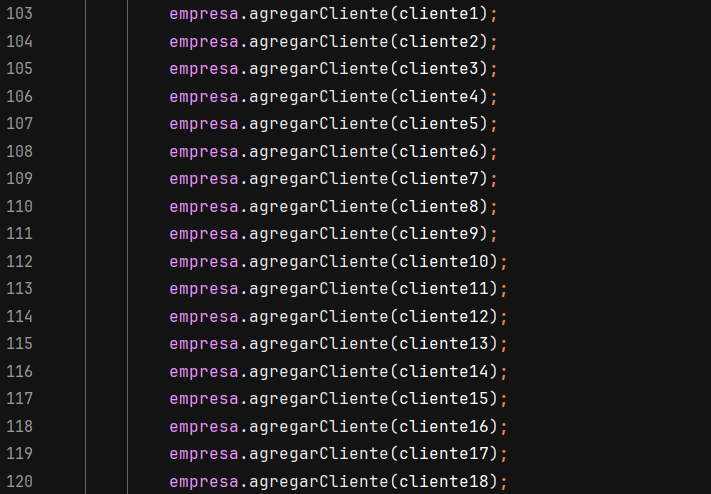


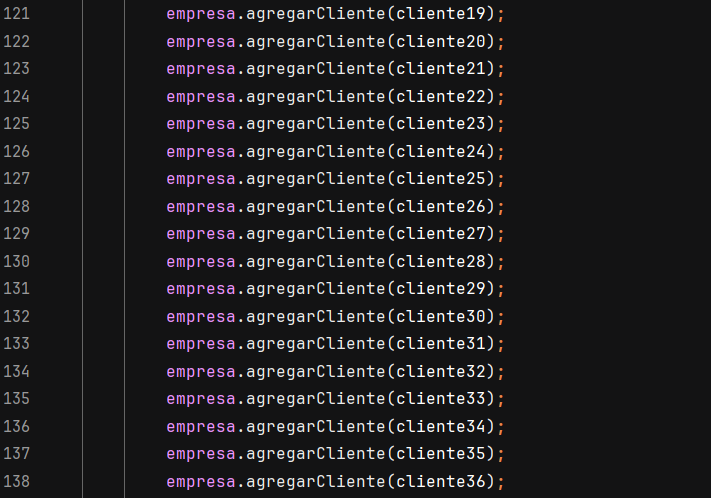




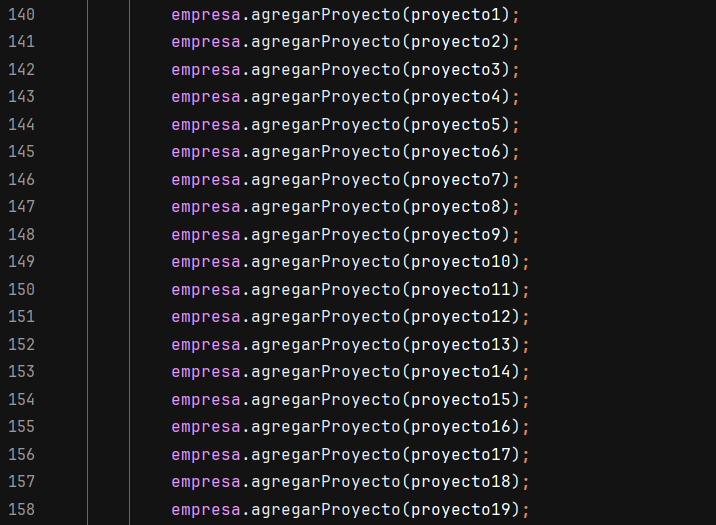


Las siguientes líneas de codigo se muestra que se estara llamando un método con el nombre “agregarCliente” a la variable anteriormente mencionada con el nombre “empresa” para agregar cada cliente determinado a una lista de los clientes. Como, por ejemplo, el cliente1 es agregado al método de agregarCliente, donde lo estará listando en una lista de clientes que se ejecutará en la clase “EMPRESA”. Y lo mismo sucederá con el cliente2, cliente3, cliente4, cliente5, hasta llegar a agregar al cliente36.



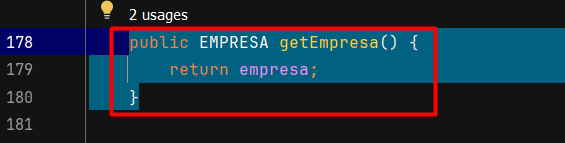


Las siguientes líneas de codigo se muestra que se estara llamando un método con el nombre “agregarProyecto” a la variable anteriormente mencionada con el nombre “empresa” para agregar cada proyecto determinado a una lista de los proyectos. Como, por ejemplo, el proyecto1 es agregado al método de agregarProyecto, donde lo estará listando en una lista de proyectos que se ejecutará en la clase “EMPRESA”. Y lo mismo sucederá con el proyecto2, proyecto3, proyecto4, proyecto5, hasta llegar a agregar al proyecto36.

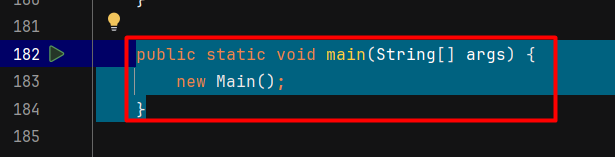




La siguiente linea de programación que encontramos es la siguiente “public EMPRESA getEmpresa()” el cual identificamos que tiene modificadores de acceso “public” y esto permitirá el acceso de otras clases a los datos que se ingresaran dentro. Tambien encontramos que llama de la clase “EMPRESA” un getter “getEmpresa", con el resultado de retornar todos los datos determinados a la clase “EMPRESA”, y esto sucederá con el siguiente codigo “return empresa” donde retornará a la clase “EMPRESA” todos los datos guardados en “empresa”.

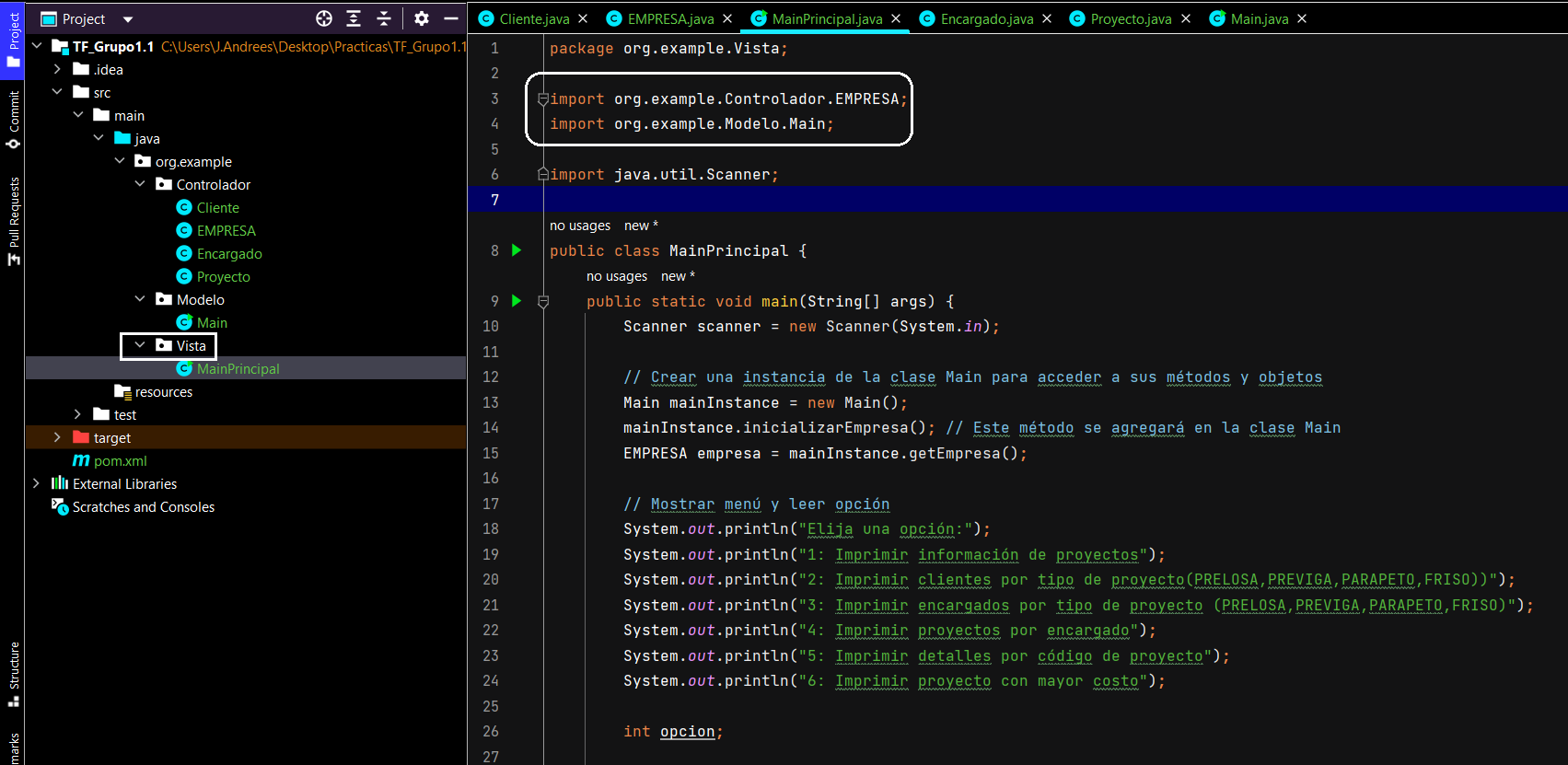


La siguiente linea de programación que encontramos es la siguiente “public static void main(String[] args)” que determinara la ejecucion del programa. Aunque por no contener una variable en la siguiente linea de codigo “new Main()” quedara en modo anónimo, ejecutando todo los datos pero sin arrogar ningún valor.



## **Vista**

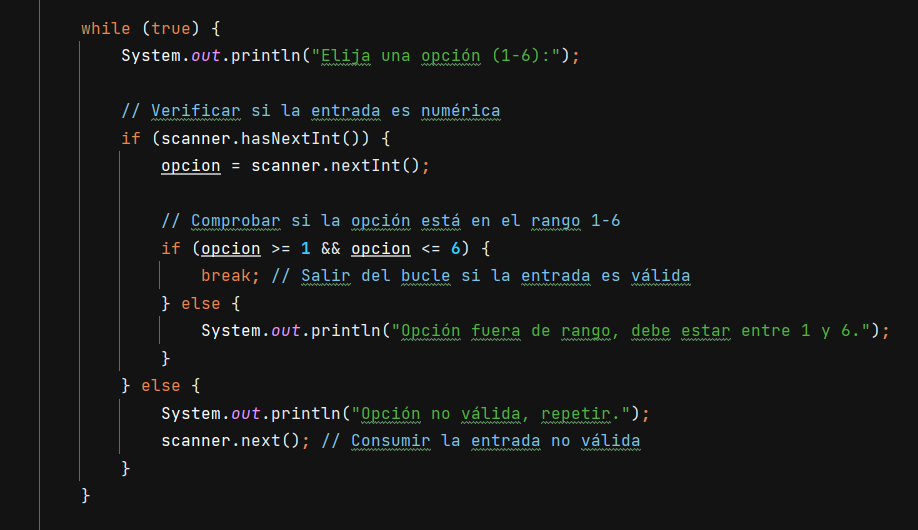
Escenario de visualizacion donde estarán siendo empleados todos los diseños de pantalla, y permitiendo del debido despliegue de nuestro trabajo.



Las estructuras que emplearemos de opción de múltiple, requerirán de un atributo registrado como opción de tipo int, nos permitirá desplegar los casos que tenemos registrado a través de switch, desentendiendo de la necesidad del usuario

# III.II. Vistas de Pantalla

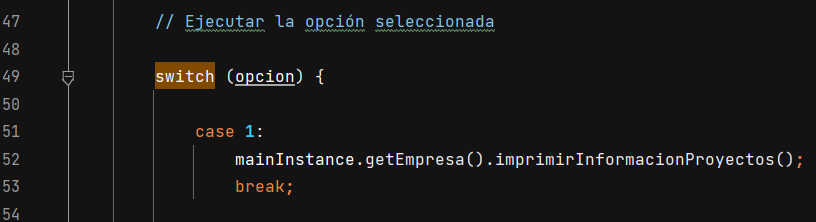
Desarrollado en un trabajo de múltiples vistas, establecidos con motores de búsqueda directamente asociados:



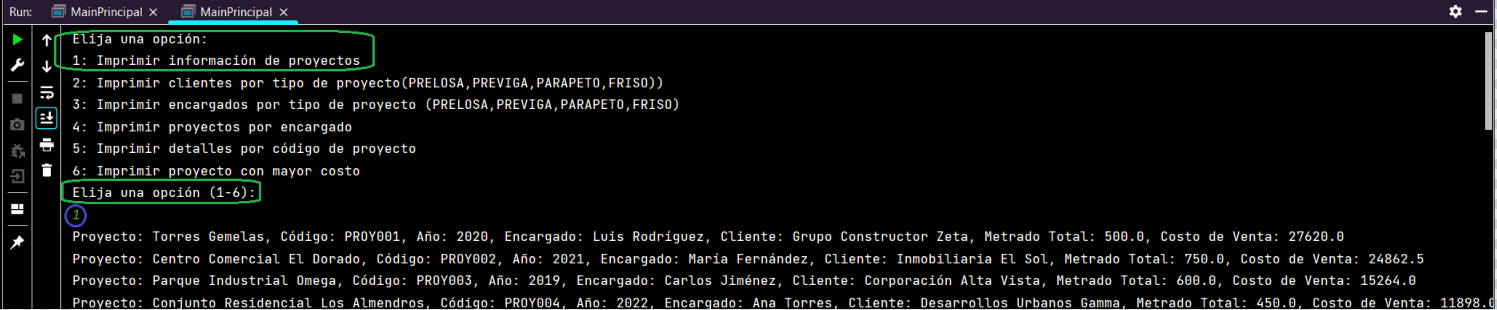
Aplicando un while de identificación de las entradas y comprobación de la ,múltiples opciones entre los rangos especificas entre 1- 6. Despliegue de ejecuciones para cada caso documentado en registro dentro del sistema de gestión de la información

1. **Mostrar 1: Imprimir información de proyectos**

Refleja todos los proyectos, permitiendo reflejar un alto despliegue de los proyecto que se trabajan y orientar al usuario en la necesidad de conocer todo contenido relevante.

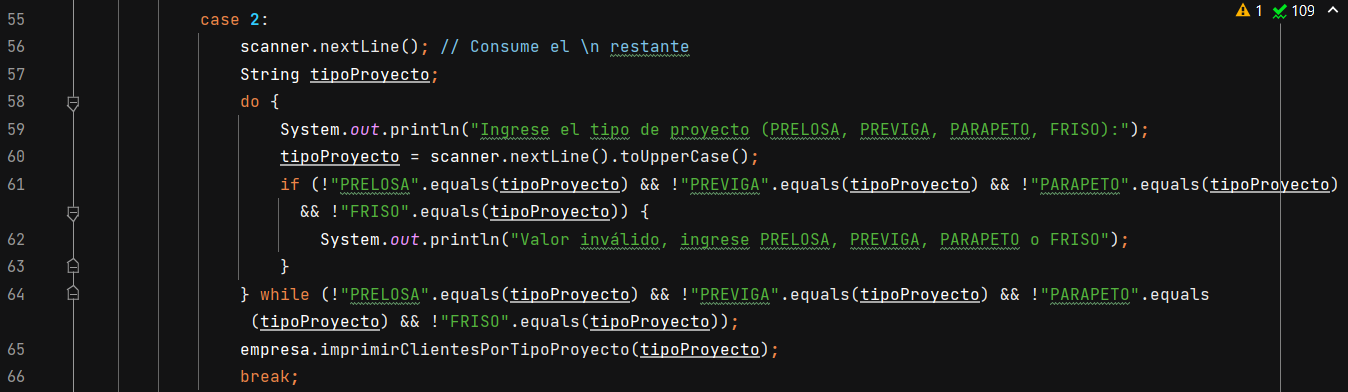


En este caso, la expresión que determina el flujo del programa es la variable opcion. Si opcion tiene un valor de 1, se ejecutará el bloque de código que se encuentra debajo del case 1.

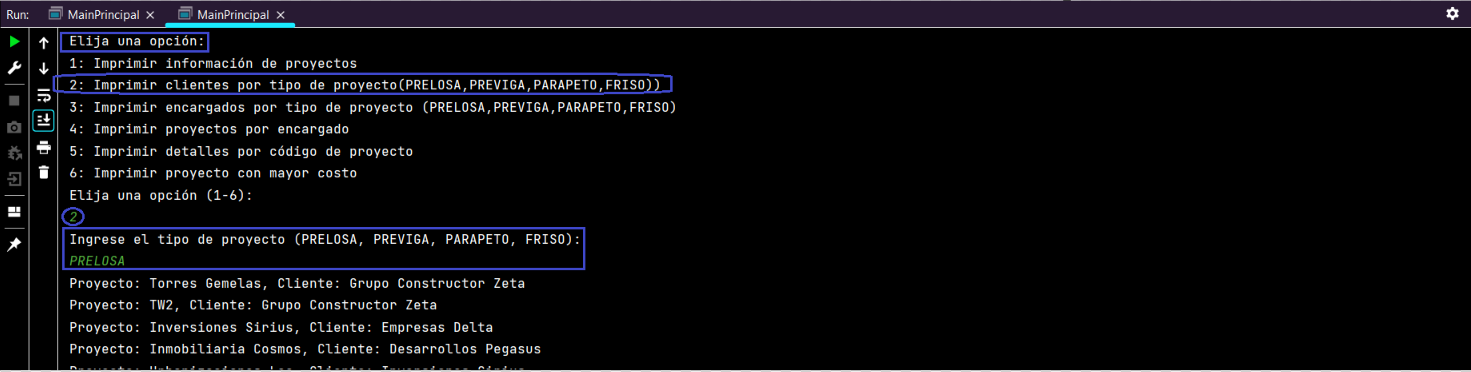
**switch (opcion):** Esta línea inicia la estructura switch utilizando la variable opcion como la expresión de control. El programa verificará el valor de opcion y ejecutará el código correspondiente al case que coincida con ese valor. **mainInstance.getEmpresa().imprimirInformacionProyectos() :** Este es el código que se ejecutará si el valor de opcion es 1

1. **Buscar 2: Imprimir clientes por tipo de proyecto (PRELOSA,PREVIGA,PARAPETO,FRISO)**

Búsqueda de los proyecto que se han realizado dependiendo de un tipo de producto referente, permitiendo realizar un estudio de mercado dependiendo de las necesidades entre proyecto.

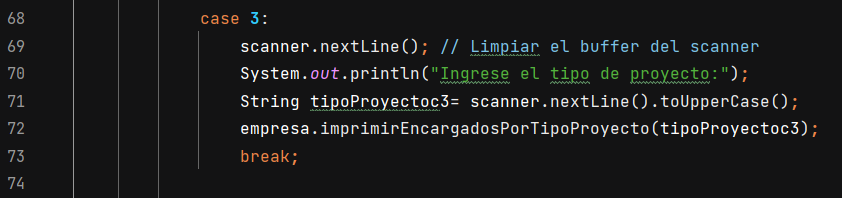


* **String tipoProyecto:** Declara una variable tipoProyecto de tipo String.
* **do { ... } while (...):** Inicia un bucle **do-while**. Este bucle se ejecutará al menos una vez y luego seguirá ejecutándose mientras la condición especificada entre los paréntesis sea verdadera.
* **tipoProyecto = scanner.nextLine().toUpperCase():** Lee la entrada del usuario y la almacena en la variable **tipoProyecto**. Luego, se convierte a mayúsculas usando **toUpperCase()**
* **empresa.imprimirClientesPorTipoProyecto(tipoProyecto):** Una vez que se ingresa un tipo de proyecto válido, llama a un método **imprimirClientesPorTipoProyecto()** del objeto **Empresa.**

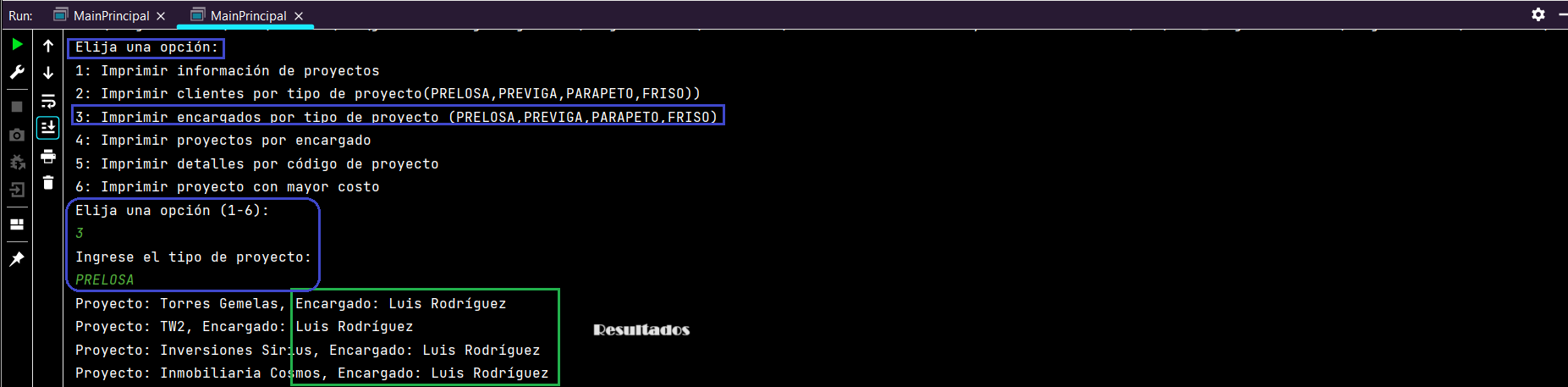


1. **Visualizar 3: Imprimir encargados por tipo de proyecto (PRELOSA,PREVIGA,PARAPETO,FRISO)**

Teniendo en cuenta que debemos visualizar en posibles escenarios de responsable contacto entre las empresas solicitantes de nuestros productos y servicios orientaremos este escenario.

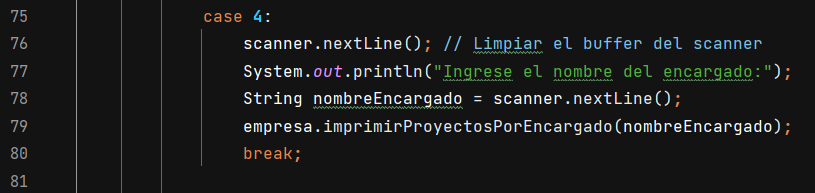


* **String tipoProyectoc3 = scanner.nextLine().toUpperCase():** Lee la entrada del usuario y la almacena en la variable **tipoProyectoc3**
* **empresa.imprimirEncargadosPorTipoProyecto(tipoProyectoc3):** Llama al método **imprimir EncargadosPorTipoProyecto()** de la instancia Empresa.

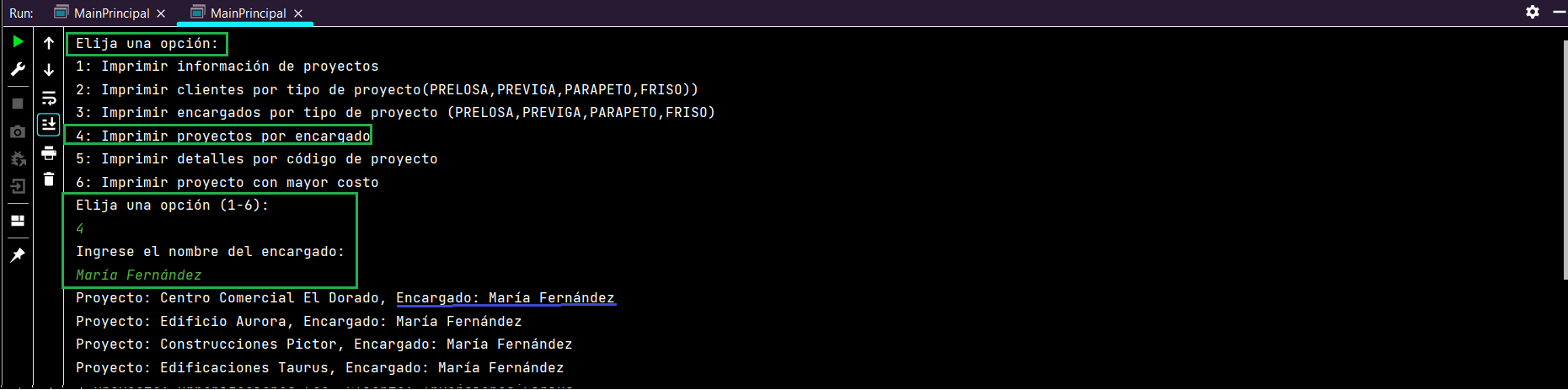


1. **Buscar 4: Imprimir proyectos por encargado**

Búsqueda orientada a devolver solo los proyectos asociados al encargo a cargo de la gestión.

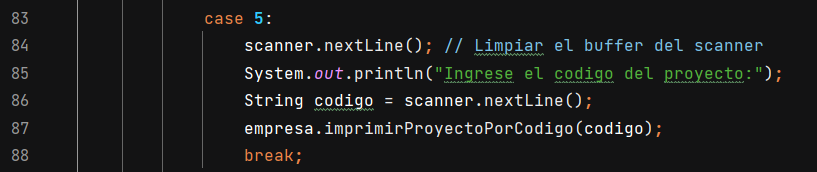


* **System.out.println(**"Ingrese el nombre del encargado:"**):** Muestra un mensaje solicitando al usuario que ingrese el nombre del encargado.
* **String nombreEncargado = scanner.nextLine():** Lee la entrada del usuario y la almacena en la variable **nombreEncargado**
* **empresa.imprimirProyectosPorEncargado(nombreEncargado):** Llama al método **imprimirProyectosPorEncargado()** de la instancia empresa

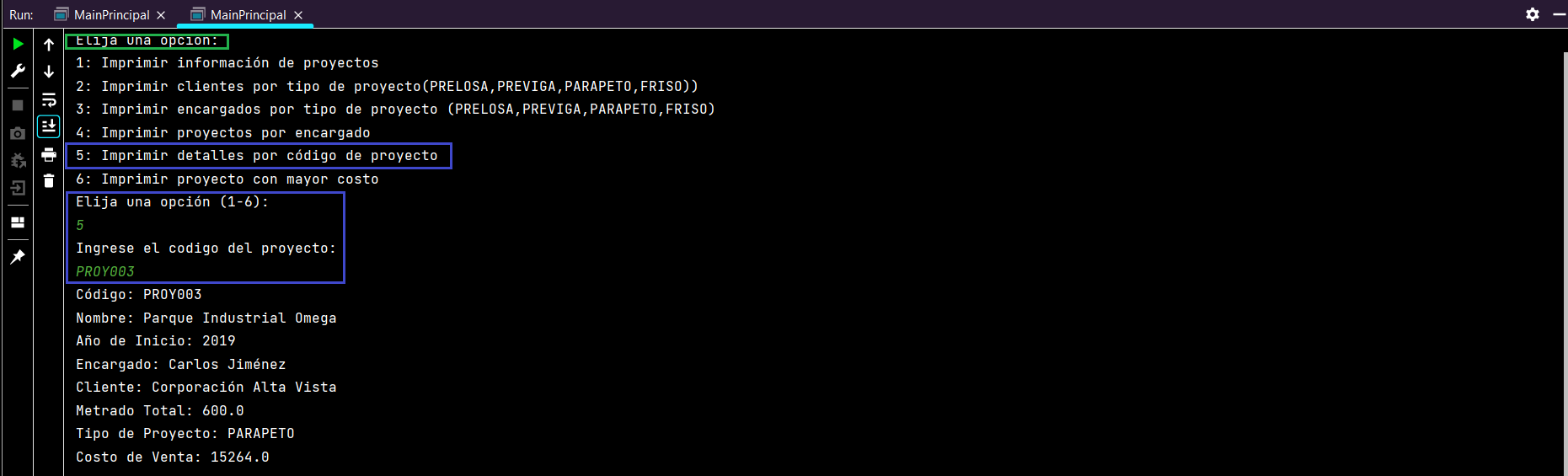


1. **Buscar 5: Imprimir detalles por código de proyecto**

Vista que desplegada será exclusivamente a analizar códigos de entre proyecto las ventas realizadas de nuestros productos. Dependiendo de un manejo admisible entre todos los componentes, packages y desarrollo de nuestros modelos lógicos y operativos

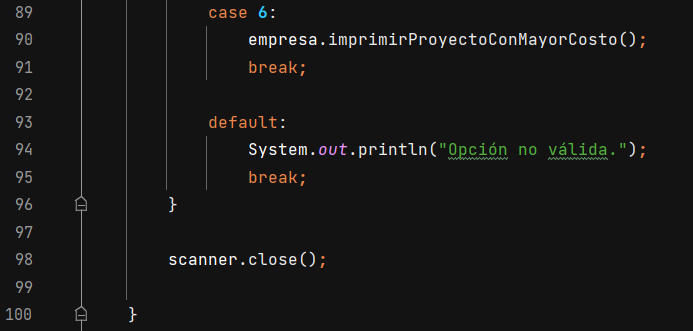


* **scanner.nextLine():** Esta línea se utiliza para limpiar el buffer del scanner y evitar problemas de lectura cuando se solicite una nueva entrada de usuario.
* **System.out.println**("Ingrese el código del proyecto:"): Muestra un mensaje solicitando al usuario que ingrese el código del proyecto.
* **String codigo = scanner.nextLine():** Lee la entrada del usuario y la almacena en la variable **codigo.**
* **empresa.imprimirProyectoPorCodigo(codigo):** Llama al método **imprimirProyectoPorCodigo()** de la instancia **Empresa.**
* **break:** Finaliza la ejecución del case actual dentro del **switch**, saliendo del bloque de control.

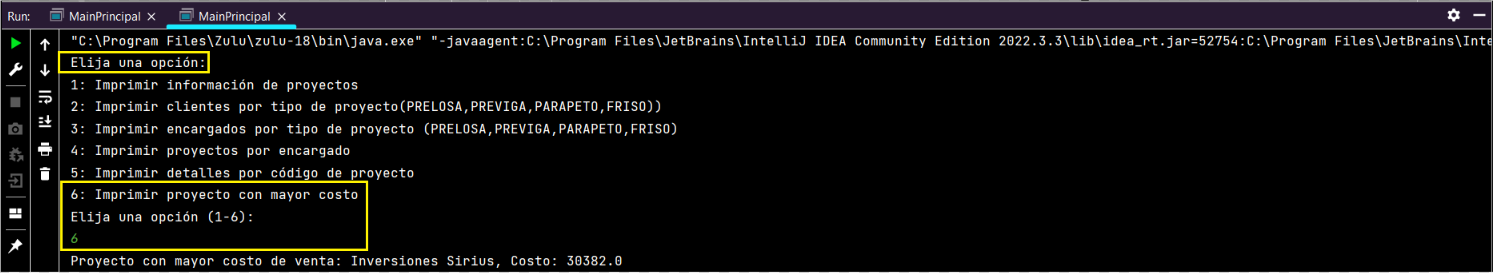


1. **Buscar 6: Imprimir proyecto con mayor costo**

Vista que devolverá el mayor costo admisible entre proyectos, las ventas realizadas de nuestros productos. Teniendo el registro con el valor máximo de su venta realizada.



* **default:** Este es el caso por defecto que se ejecutará si **opcion** no coincide con ninguno de los casos anteriores.
* **break:** Después de la ejecución de cada case, se utiliza break para salir del switch y evitar que se ejecuten los bloques de código de otros case o del default una vez que se haya realizado la acción correspondiente al valor de **opcion.**



En resumen, este código presenta un **switch** que ejecuta una acción específica basada en el valor de **opción**. Luego, el código cierra el objeto Scanner.

# III.I.III Vista

Escenario de visualizacion donde estarán siendo empleados todos los diseños de pantalla, y permitiendo del debido despliegue de nuestro trabajo.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Las estructuras que emplearemos de opción de múltiple, requerirán de un atributo registrado como opción de tipo int, nos permitirá desplegar los casos que tenemos registrado a través de switch, desentendiendo de la necesidad del usuario

Desarrollado en un trabajo de múltiples vistas, establecidos con motores de búsqueda directamente asociados:

Texto

Descripción generada automáticamente

Aplicando un while de identificación de las entradas y comprobación de la ,múltiples opciones entre los rangos especificas entre 1- 6. Despliegue de ejecuciones para cada caso documentado en registro dentro del sistema de gestión de la información

**Mostrar 1: Imprimir información de proyectos**

Refleja todos los proyectos, permitiendo reflejar un alto despliegue de los proyecto que se trabajan y orientar al usuario en la necesidad de conocer todo contenido relevante.

Texto

Descripción generada automáticamente

En este caso, la expresión que determina el flujo del programa es la variable opcion. Si opcion tiene un valor de 1, se ejecutará el bloque de código que se encuentra debajo del case 1.

Texto

Descripción generada automáticamente con confianza media**switch (opcion):** Esta línea inicia la estructura switch utilizando la variable opcion como la expresión de control. El programa verificará el valor de opcion y ejecutará el código correspondiente al case que coincida con ese valor. **mainInstance.getEmpresa().imprimirInformacionProyectos() :** Este es el código que se ejecutará si el valor de opcion es 1

**Buscar 2: Imprimir clientes por tipo de proyecto**

Búsqueda de los proyecto que se han realizado dependiendo de un tipo de producto referente, permitiendo realizar un estudio de mercado dependiendo de las necesidades entre proyecto.

Texto

Descripción generada automáticamente con confianza media

* **String tipoProyecto:** Declara una variable tipoProyecto de tipo String.
* **do { ... } while (...):** Inicia un bucle **do-while**. Este bucle se ejecutará al menos una vez y luego seguirá ejecutándose mientras la condición especificada entre los paréntesis sea verdadera.
* **tipoProyecto = scanner.nextLine().toUpperCase():** Lee la entrada del usuario y la almacena en la variable **tipoProyecto**. Luego, se convierte a mayúsculas usando **toUpperCase()**
* **empresa.imprimirClientesPorTipoProyecto(tipoProyecto):** Una vez que se ingresa un tipo de proyecto válido, llama a un método **imprimirClientesPorTipoProyecto()** del objeto **Empresa.**

Captura de pantalla de computadora

Descripción generada automáticamente

**Visualizar 3: Imprimir encargados por tipo de proyecto**

Teniendo en cuenta que debemos visualizar en posibles escenarios de responsable contacto entre las empresas solicitantes de nuestros productos y servicios orientaremos este escenario.

Texto

Descripción generada automáticamente

* **String tipoProyectoc3 = scanner.nextLine().toUpperCase():** Lee la entrada del usuario y la almacena en la variable **tipoProyectoc3**
* **empresa.imprimirEncargadosPorTipoProyecto(tipoProyectoc3):** Llama al método **imprimir EncargadosPorTipoProyecto()** de la instancia Empresa.

Captura de pantalla de computadora

Descripción generada automáticamente

**Buscar 4: Imprimir proyectos por encargado**

Búsqueda orientada a devolver solo los proyectos asociados al encargo a cargo de la gestión.

Texto

Descripción generada automáticamente

* **System.out.println(**"Ingrese el nombre del encargado:"**):** Muestra un mensaje solicitando al usuario que ingrese el nombre del encargado.
* **String nombreEncargado = scanner.nextLine():** Lee la entrada del usuario y la almacena en la variable **nombreEncargado**
* **empresa.imprimirProyectosPorEncargado(nombreEncargado):** Llama al método **imprimirProyectosPorEncargado()** de la instancia empresa

Texto

Descripción generada automáticamente

**Buscar 5: Imprimir detalles por código de proyecto**

Vista que desplegada será exclusivamente a analizar códigos de entre proyecto las ventas realizadas de nuestros productos. Dependiendo de un manejo admisible entre todos los componentes, packages y desarrollo de nuestros modelos lógicos y operativos

Texto

Descripción generada automáticamente

* **scanner.nextLine():** Esta línea se utiliza para limpiar el buffer del scanner y evitar problemas de lectura cuando se solicite una nueva entrada de usuario.
* **System.out.println**("Ingrese el código del proyecto:"): Muestra un mensaje solicitando al usuario que ingrese el código del proyecto.
* **String codigo = scanner.nextLine():** Lee la entrada del usuario y la almacena en la variable **codigo.**
* **empresa.imprimirProyectoPorCodigo(codigo):** Llama al método **imprimirProyectoPorCodigo()** de la instancia **Empresa.**
* **break:** Finaliza la ejecución del case actual dentro del **switch**, saliendo del bloque de control.

Texto

Descripción generada automáticamente

**Buscar 6: Imprimir proyecto con mayor costo**

Vista que devolverá el mayor costo admisible entre proyectos, las ventas realizadas de nuestros productos. Teniendo el registro con el valor máximo de su venta realizada.

Texto

Descripción generada automáticamente

**default:** Este es el caso por defecto que se ejecutará si **opcion** no coincide con ninguno de los casos anteriores.

**break:** Después de la ejecución de cada case, se utiliza break para salir del switch y evitar que se ejecuten los bloques de código de otros case o del default una vez que se haya realizado la acción correspondiente al valor de **opcion.**

Captura de pantalla de computadora

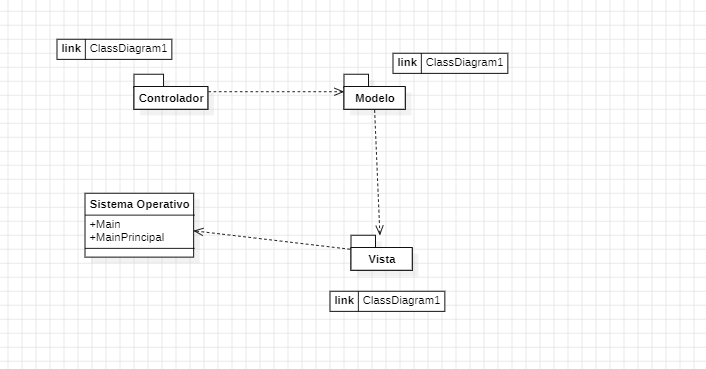
Descripción generada automáticamente

En resumen, este código presenta un **switch** que ejecuta una acción específica basada en el valor de **opción**. Luego, el código cierra el objeto Scanner.

# III.II. Diagrama de Clase Modelo

En el diagrama de Clase Modelo se realizó un UML de paquetes que utilizaremos para el trabajo, como tambien, determinar las clases que tendría nuestro programa y las variables que tendría cada clase, al igual que los métodos y listas.

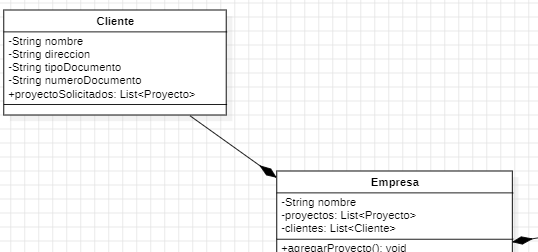
**Paquetes:** Para el desarrollo del programa utilizaremos 3 paquetes, las cuales serán Controlador, Modelo y Vista. Dentro de cada paquete habra clases las cuales ayudara al funcionamiento del programa. Los 3 paquetes se podrán compartir datos, variables, listas y métodos, para reducir la programación en la clase Main y MainPrincipal, siendo el paquete Controlador comparta variables y métodos al paquete Modelo, y el paquete Modelo comparta variables y métodos al paquete Vista.



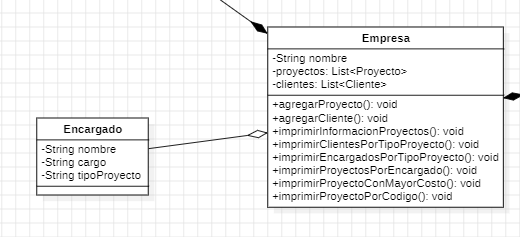
**Paquete Controlador:** Dentro del paquete encontramos las siguientes clases con sus respectivas variables, listas y métodos.



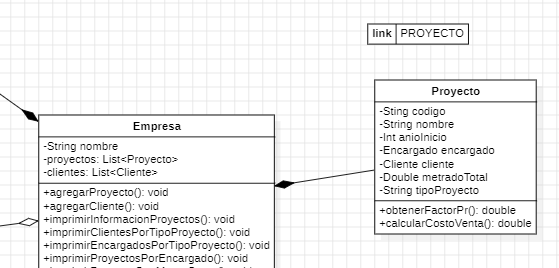
**Clase Cliente:** En la clase cliente podemos encontrar las variables “nombre”, “direccion”, “tipoDocumento” y “numeroDocumento” las cuales serán de tipo String por ser variables de texto y serán variables privadas. Además, se le agregara un método “proyectoSolicitado” el cual consistirá de una lista que almacene los datos de los proyectos y será un método público. La clase Cliente tendrá una conexión de tipo composición con la clase Empresa.



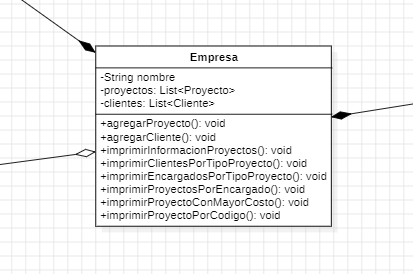
**Clase Encargado:** En la clase encargado podemos encontrar las variables “nombre”, “cargo” y “tipoProyecto” las cuales serán de tipo String por ser variables de texto y privada de la clase. La clase Encargado tendrá una conexión de tipo agregación con la clase Empresa.



**Clase Proyecto:** En la clase proyecto podemos encontrar las variables “codigo”, “nombre” y “tipoDocumento” las cuales serán de tipo String por ser variables de texto, la variable “anioInicio” será de tipo int por ser variable de número real, y la variable “metradoTotal” será de tipo double por ser variable de numero con decimal. Tambien tendra variables “cliente” y “encargado” que cumplirán la función de almacenar y/o usar variables de la clase Cliente y Encargado. Además, tiene métodos de “obtenerFactorPr” y “calcularCostoVenta” de será de tipo double por ser variable de numero con decimal y si retornara un valor.

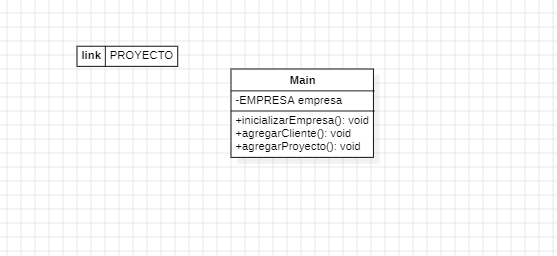


**Clase Empresa:** En la clase empresa podemos encontrar las variables “nombre” de tipo String por ser variables de texto, tambien habra listas para proyecto y cliente, y serán de tipo privado. Y como métodos se tienen el “agregarProyecto”, “agregarCliente”, “imprimirInformacionProyectos”, “imprimirClientePorTipoProyecto”, “imprimirEncargadoPorTipoProyecto”, “imprimirProyectoPorEncargado”, “imprimirProyectoConMayorCosto” y “imprimirProyectoPorCodigo”, las cuales son de tipo void, es decir, no se devuelve ningún valor.



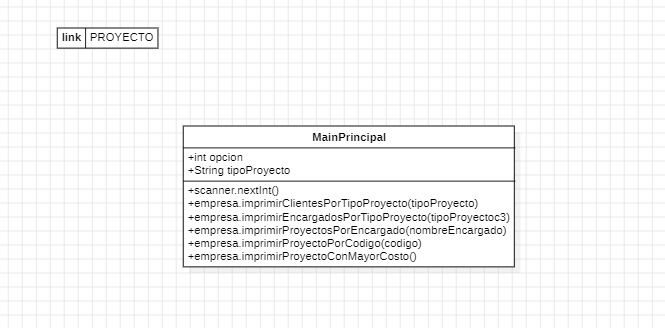
**Paquete Modelo**: Dentro del paquete encontramos las siguientes clases con sus respectivas variables y métodos.

**Clase Main: En la clase Main podemos encontrar la variable “empresa”** que cumplirán la función de almacenar y/o usar variables de la clase EMPRESA. Además, se tiene como métodos “inicializarEmpresa”, “agregarCliente” y “agregarProyecto” las cuales son de tipo void, es decir, no se devuelve ningún valor.



**Paquete Vista:** Dentro del paquete encontramos las siguientes clases con sus respectivas variables y métodos.

**Clase MainPrincipal:** En la clase MainPrincipal podemos encontrar las variables “opcion” de tipo int por ser variable de número real y variables “tipoProyecto” de tipo String por ser variables de texto. Y como métodos se tienen el scanner.nextInt, empresa.imprimirClientesPorTipoProyecto, empresa.imprimirEncargadosPorTipoProyecto,empresa.imprimirProyectosPorEncargado, empresa.imprimirProyectoPorCodigo, empresa.imprimirProyectoConMayorCosto().



III.III. **Diseño de Clase Patrones de Diseño a Usar**

En el diseño de Clase Patrones de Diseño a Usar se realizó un UML de paquetes que utilizaremos para el trabajo, como tambien, determinar las clases que tendría nuestro programa y las variables que tendría cada clase, al igual que los métodos y listas. Además, crear clases que cumplirán el rol de respaldo o agregación de más clases con la misma función.

Diagrama

Descripción generada automáticamente

# Estructura de la clase Modelo:

Se usará el diagrama de la clase Modelo anterior explicado que estaban ubicados en distintos paquetes, y se unirá en una misma clase. Luego realizaremos las relaciones que tendrían entre las clases.

Diagrama

Descripción generada automáticamente

**Encargado y Empresa:** La relación que encontramos en estas dos clases es de tipo agregación y es de 1 a 0 a muchos, partiendo desde la clase encargado.

Diagrama

Descripción generada automáticamente

**Cliente y Empresa:** La relación que encontramos en estas dos clases es de tipo agregación y es de 1 a 0 a muchos, partiendo desde la clase cliente.

Diagrama

Descripción generada automáticamente

**Proyecto y Empresa:** La relación que encontramos en estas dos clases es tipo composición y es de 1 a 1, partiendo desde la clase Proyecto.

Gráfico

Descripción generada automáticamente

**Empresa y Sistema Operativo:** La relación que encontramos en estas dos clases es de tipo asociación directa, partiendo desde la clase empresa.

Gráfico

Descripción generada automáticamente

**Empresa y Modelo:** La relación que encontramos en estas dos clases es de tipo dependencia, partiendo desde la clase empresa.

Gráfico

Descripción generada automáticamente con confianza media

# Modelo y Sistema Operativo:

La relación que encontramos en estas dos clases es de tipo asociación directa, partiendo desde la clase modelo.

Diagrama

Descripción generada automáticamente con confianza media

# Patrones de diseño implementados:

Al diagrama se le agrega un patrón de tipo Abstract Factory llamada “ProyectoFactory”, que nos permitirá la creación de objetos con el método de tipo gestor con el nombre “crearElementoProyecto”. La relación que tendrá con las clases encargado, Cliente y Proyecto será de tipo asociación directa.

Diagrama

Descripción generada automáticamente

Tambien agregaremos otro patrón llamado “InterfazAgregación”, que nos permitirá la unión o creación de más clases que cumplan la función agregar cliente o agregar proyecto. Además, cumplirá la función de respaldar si el mismo método falla en otras clases. La relación que tendrá con las clases empresa y modelo es dependencia.

Diagrama

Descripción generada automáticamente

**III.IV. Link de video mostrando el funcionamiento del Programa**

Se adjunta el link del video: https://youtu.be/5qYbeAwraGk?si=di4FLKZ894p2DGuN

# CONCLUSIONES:

* El proyecto muestra una organización modular, con una clara separación de responsabilidades entre los paquetes y clases. Esto facilita la mantenibilidad y la comprensión del código.
* Existe un uso efectivo de la orientación a objetos, evidenciado por la creación de clases específicas para distintos componentes del sistema (como Cliente y Encargado). Esto sugiere un entendimiento de los principios de encapsulamiento y abstracción.
* La implementación de un menú interactivo en la clase MainPrincipal indica una consideración hacia la interfaz de usuario, aunque es basada en consola. Esto demuestra un enfoque en la experiencia del usuario final.
* La repetición en la adición de clientes y proyectos en la clase Main sugiere una oportunidad para optimizar el código, posiblemente a través de un método que automatice la adición de múltiples entidades, reduciendo la redundancia y mejorando la eficiencia.
* El proyecto refleja una planificación cuidadosa y una implementación detallada, especialmente en la clase Main donde se inicializa y configura la empresa con encargados, clientes y proyectos, lo que indica una comprensión profunda del dominio del problema.
* La utilización de constantes (como los tipos de proyecto en el menú de opciones) y la estructura clara de los menús demuestran buenas prácticas de programación, lo que facilita la lectura y mantenimiento del código.
* La inclusión de un bucle para validar las entradas del usuario en MainPrincipal muestra una preocupación por la robustez en la interacción del usuario, asegurando que el programa maneje correctamente las entradas inválidas.

# Recomendaciones:

* Implementar pruebas unitarias para validar la funcionalidad de las clases y métodos, asegurando que el código funcione como se espera y facilitando la detección de errores.
* Considerar la integración de un sistema de gestión de base de datos para manejar la información de clientes, proyectos y empleados, en lugar de almacenar los datos directamente en la memoria. Esto mejoraría la escalabilidad y persistencia de los datos.
* Explorar el uso de interfaces gráficas de usuario (GUI) para reemplazar el menú basado en consola, lo que podría mejorar significativamente la experiencia del usuario y hacer el software más accesible para un público más amplio.
* Agregar documentación interna en el código, como comentarios detallados y documentación de la API, para facilitar la comprensión y el mantenimiento del código por otros desarrolladores.
* Optimizar el código en la clase `Main` para evitar la repetición al agregar clientes y proyectos, tal vez utilizando bucles o métodos auxiliares, para hacer el código más eficiente y fácil de actualizar.
* Incorporar un manejo de excepciones más robusto en toda la aplicación, para asegurar que el programa pueda manejar situaciones inesperadas sin fallar y proporcionar retroalimentación útil al usuario en caso de errores.

# Objetivo del estudiante (Student Outcome):

**ABET – Student Outcome 2:** La capacidad de aplicar el diseño de ingeniería para producir soluciones que satisfagan necesidades específicas con consideración de salud pública, seguridad y bienestar, así como factores globales, culturales, sociales, ambientales y económicos.

**Marquina Chamorro, Benjamín Flaviano:**

En mi opinión, logramos el objetivo del curso de manera excepcional gracias a una colaboración sólida y continua entre los miembros del equipo. Desde el inicio, establecimos roles claros y comunicación abierta, lo que permitió una distribución eficiente de tareas. A través de sesiones regulares de lluvia de ideas y revisiones conjuntas, pudimos aprovechar la diversidad de habilidades en el grupo, generando soluciones innovadoras y completas. La flexibilidad y apoyo mutuo fueron clave para superar obstáculos y garantizar un proyecto final integral.

**Gutiérrez Puma, José Andrés:**

La sinergia en nuestro equipo fue esencial para alcanzar el objetivo del curso. Adoptamos un enfoque colaborativo desde el principio, fomentando un ambiente donde cada miembro se sintiera valorado y escuchado. Utilizamos plataformas en línea para compartir recursos de manera eficiente y garantizar que todos estuvieran en la misma página. La responsabilidad compartida y la comunicación efectiva nos permitieron abordar desafíos de manera proactiva, garantizando un producto final que refleja el esfuerzo colectivo.

**Aquije Dulanto, Alberto Daniel:**

En el contexto del trabajo en equipo, logramos sobresalir en el curso al cultivar un ambiente de confianza y respeto mutuo. Establecimos canales de comunicación abiertos y alentamos la retroalimentación constante. La asignación de tareas se basó en las fortalezas individuales, lo que resultó en una distribución equitativa y eficiente del trabajo. La colaboración continua no solo mejoró la calidad de nuestro proyecto, sino que también fortaleció nuestra comprensión colectiva de los fundamentos del curso.

**Jara Pinto, Alessandro Paris:**

La clave para nuestro éxito en el curso fue nuestra habilidad para trabajar en equipo de manera ágil y adaptable. Implementamos metodologías ágiles para gestionar eficientemente las tareas y adaptarnos a cambios inesperados. Aprovechamos las habilidades complementarias de cada miembro, permitiendo una resolución rápida y eficaz de problemas. Mantuvimos una comunicación transparente mediante reuniones regulares y herramientas colaborativas, lo que resultó en un proyecto final bien estructurado y ejecutado.

**Echegaray Vilcatoma, Estefany Naylu:**

Considero que alcanzamos el objetivo del curso mediante una colaboración estructurada y enfocada en resultados. Desde la fase inicial de planificación, establecimos metas claras y plazos realistas. Implementamos un sistema de seguimiento eficiente para garantizar que cada miembro cumpliera con sus responsabilidades.

# Bibliografía:

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional. pp. 25-35. Este libro es fundamental en el campo de la informática para entender los patrones de diseño en la programación orientada a objetos.

Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall. pp. 45-60. Robert Martin ofrece principios y técnicas para escribir código limpio y bien estructurado.

Fowler, M. (2018). Refactoring: Improving the Design of Existing Code (2nd ed.). Addison-Wesley Professional. pp. 75-90. Este texto es clave para comprender cómo mejorar y optimizar código ya existente.

McConnell, S. (2004). Code Complete: A Practical Handbook of Software Construction (2nd ed.). Microsoft Press. pp. 100-120. En este libro, McConnell ofrece una guía extensa sobre técnicas de desarrollo de software, cubriendo desde la concepción de un proyecto hasta su implementación final.