**Informe Nelson Villamizar.**

**Tabla de Contenido.**

[1. Nombre Completo del postulante e información de contacto. 3](#_Toc38140103)

[2. Vía por la que se enteró del desafío: 3](#_Toc38140104)

[3. Generar un diagrama que explique la implementación en los casos 1 al 4 3](#_Toc38140105)

[3.1. Implementación 1. 3](#_Toc38140106)

[3.2. Implementación 2. 5](#_Toc38140107)

[3.3. Implementación 3. 6](#_Toc38140108)

[3.4. Implementación 4. 6](#_Toc38140109)

[4. Enumerar todos los patrones de diseño encontrados en los casos 1 al 4 6](#_Toc38140110)

[4.1. Implementación 1. 6](#_Toc38140111)

[5. Enumerar todos los errores, falencias o posibles mejoras en los casos 1 al 4. 6](#_Toc38140112)

[5.1. Implementación 1. 6](#_Toc38140113)

[5.1.1. Errores o falencias. 6](#_Toc38140114)

[5.1.2. Mejoras. 7](#_Toc38140115)

[5.2. Implementación 2. 10](#_Toc38140116)

[5.2.1. Errores o falencias. 10](#_Toc38140117)

[5.2.2. Mejoras. 11](#_Toc38140118)

[Se propone el siguiente diagrama de clases. 11](#_Toc38140119)

[Se propone organizar el código para que los datos necesarios sean retornados por una clase Servicio. Y sea más legible. 11](#_Toc38140120)

[La clase principal 12](#_Toc38140121)

[El controlador 12](#_Toc38140122)

[El servicio 13](#_Toc38140123)

[5.3. Implementación 3. 14](#_Toc38140124)

[5.3.1. Errores o falencias. 14](#_Toc38140125)

[5.3.2. Mejoras. 14](#_Toc38140126)

[5.4. Implementación 4. 14](#_Toc38140127)

[5.4.1. Errores o falencias. 14](#_Toc38140128)

[5.4.2. Mejoras. 14](#_Toc38140129)

[6. Para la quinta implementación se debe analizar, detallar los niveles y principios de arquitectura que usted cree están definidos en el código. 15](#_Toc38140130)

[7. En el sexto punto se deben identificar los tipos de arquitectura. 15](#_Toc38140131)

[7.1. Diagrama 1 15](#_Toc38140132)

[7.2. Diagrama 2 17](#_Toc38140133)

[7.3. Diagrama 3 18](#_Toc38140134)

[8. Por último indique cualesquiera condiciones adicionales que usted considere necesarios. 18](#_Toc38140135)

1. Nombre Completo del postulante e información de contacto.

**Nombre:** Nelson Orlando Villamizar Mantilla.

**E-mail**: [nelsonvillam@gmail.com](mailto:nelsonvillam@gmail.com)

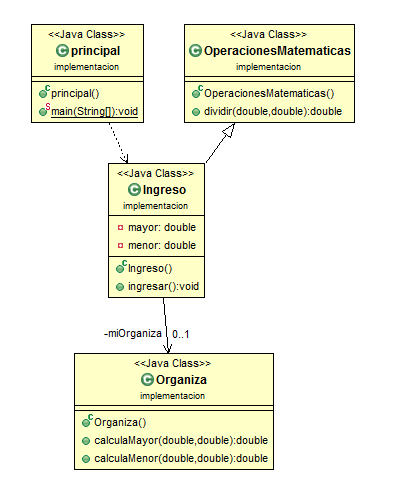
**Cel:** +56959028564

1. Vía por la que se enteró del desafío:

Leonardo Miranda (Tech Consult) me contactó a través de **linkedIn**. También me envió un mensaje a mi correo electrónico

1. Generar un diagrama que explique la implementación en los casos 1 al 4
   1. Implementación 1.

**Diagrama de clases.**



**Se puede concluir del diagrama:**

La clase **principal** depende de la clase **Ingreso**

La clase **Ingreso** depende de la clase **Organiza** y **OperacionesMatematicas**

**Revisando el código fuente.**

En esta implementación veo que la clase **principal** es la que inicia la aplicación.

La clase principal instancia la clase **Ingreso** y ejecuta el método **ingresar()**

La clase Ingreso en su método ingresar tiene la lógica de la vista (captura de datos del usuario) y lógica de negocio. Esta clase hereda de la clase **OperacionesMatematicas.**

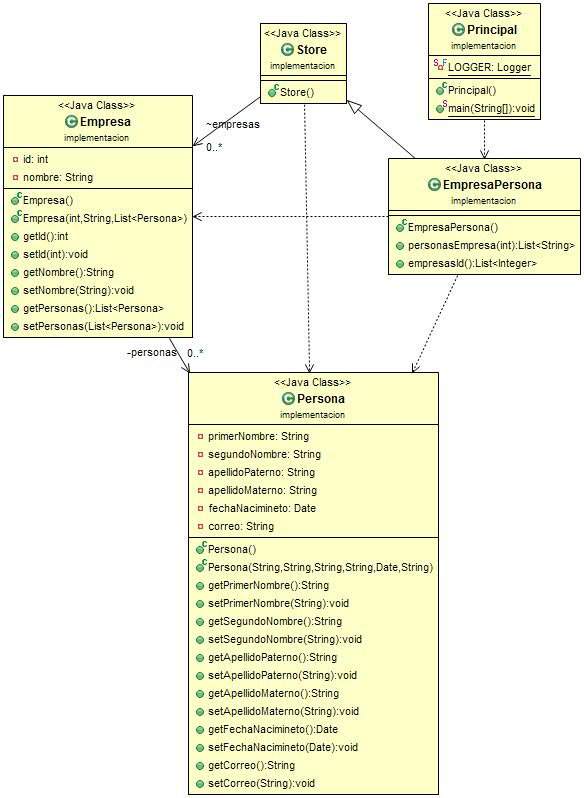
**Revisando la lógica implementada.**

La aplicación captura dos datos del usuario y determina cual es el mayor y menor y retorna la división del mayor entre el menor.

**public** **class** Organiza { es la clase que tiene la lógica que permite calcular el mayor y menor.

* 1. Implementación 2.

**Diagrama de clases.**

****

**Se puede concluir del diagrama:**

Una Store puede tener muchas empresas.

En cada empresa trabajan varias personas.

Una persona puede trabajar en varias empresas.

**Revisando el código fuente.**

Se observa el uso del patrón de diseño Factory

Se observa polimorfismo.

Se usa Enum en su forma más simple.

**Revisando lógica implementada.**

.

Lo que hace es mostrar en la consola los valores del campo id de todas las Empresas en el orden ingresado, Para eso usa Streams map()  de java 8 para obtener los ids y los retorna en una lista para luego imprimirlos en consola.

Y luego imprime los nombres de las personas de la primera empresa que fue ingresada. Para eso también usa Streams map()  de java 8.

* 1. Implementación 3.

**Diagrama de clases.**

* 1. Implementación 4.

**Diagrama de clases.**

1. Enumerar todos los patrones de diseño encontrados en los casos 1 al 4
   1. Implementación 1.

No se ve un patrón de diseño.

1. Enumerar todos los errores, falencias o posibles mejoras en los casos 1 al 4.
   1. Implementación 1.
      1. Errores o falencias.

* Acoplamiento.

Por ejemplo aquí.

**public** Ingreso() {

**this**.miOrganiza = **new** Organiza();

}

Siempre se debe evitar instanciar clases dentro de una clase. Es recomendable pasar las referencias.

* Mezcla de lógica de la vista con el negocio.

JOptionPane no debería estar mezclado con la lógica de negocio ya que eso le resta escalabilidad a la aplicación, si la vista se cambiara tocaría reescribir toda la clase.

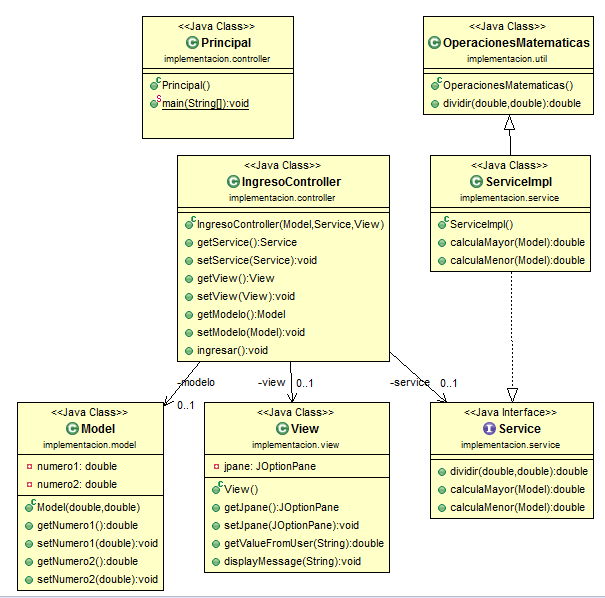
* Nombre de clase en minúscula.

Las clases deben ir en mayúscula. La clase **public** **class** principal {. Se puede ver que está en minúscula.

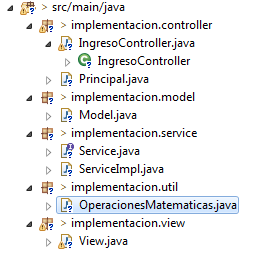
* Crear packages

Crear packages para clasificar las clases de acuerdo a su función. Todo está en el mismo package.

* + 1. Mejoras.
* Se propone el siguiente diagrama de clases.

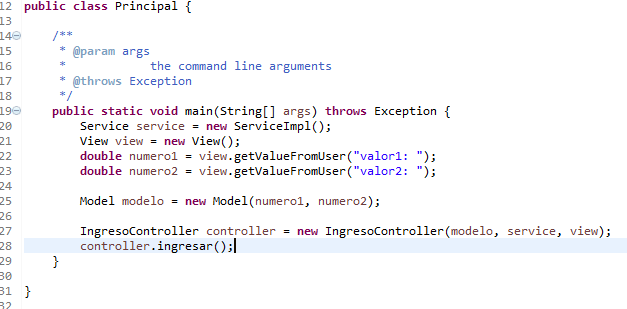
****

Es un cambio para implementar MVC y separar la vista de la lógica de negocio.

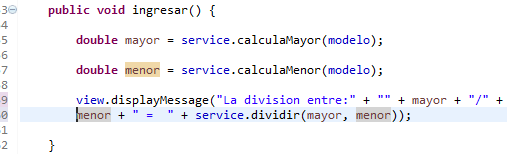


A continuación, en imágenes lo relevante de lo que se propone

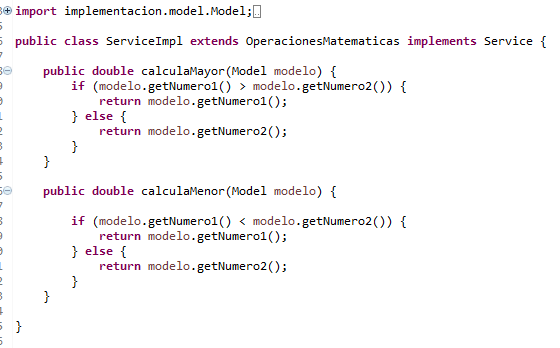
Este sería la clase que iniciaría la aplicación y como se puede ver el controlador estaría desacoplado. Todas las instancias de clase se crean aquí. Por tanto se podría migrar fácil a Spring MVC por ejemplo (usar inyección de dependencia).



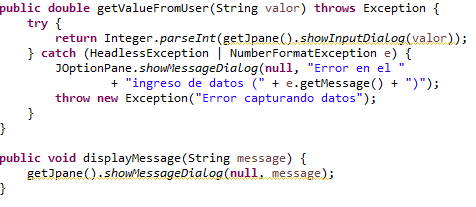
Método relevante del controlador.



La clase del servicio. Que se encarga de la lógica, en este caso determinar el mayo r y menor y la división que se hereda de la clase **OperacionesMatematicas**



Métodos relevantes de la vista:



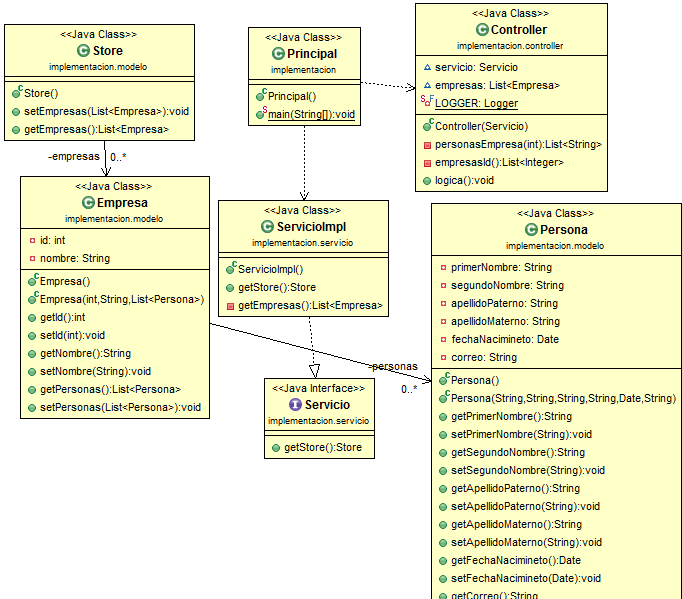
* 1. Implementación 2.
     1. Errores o falencias.
* Acoplamiento
* Código desorganizado.
* Falta clasificar las clases en packages.
  + 1. Mejoras.

Se propone el siguiente diagrama de clases.

Una Store puede tener muchas empresas.

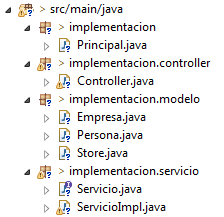
En cada empresa trabajan varias personas.

Una persona puede trabajar en varias empresas.

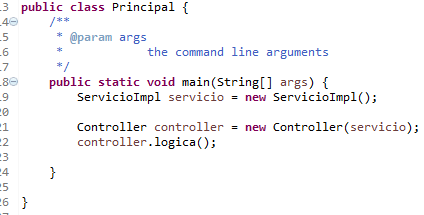


Se propone organizar el código para que los datos necesarios sean retornados por una clase Servicio. Y sea más legible. Se propone un MVC.

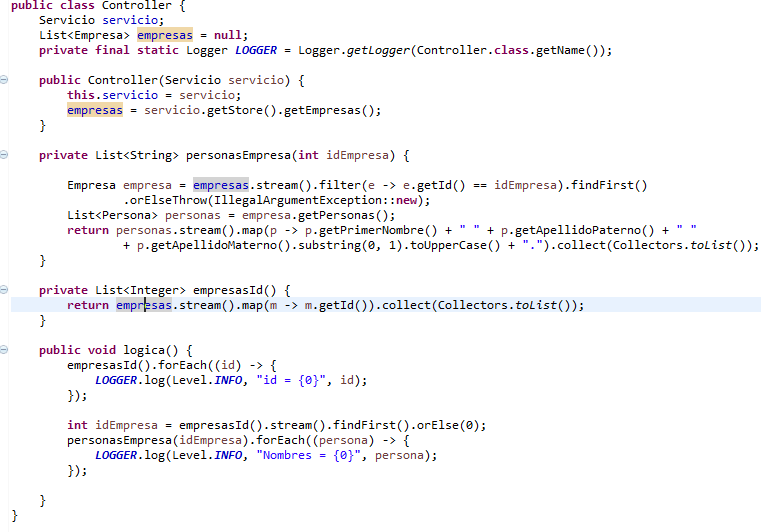
A continuación, en imágenes lo relevante de lo que se propone



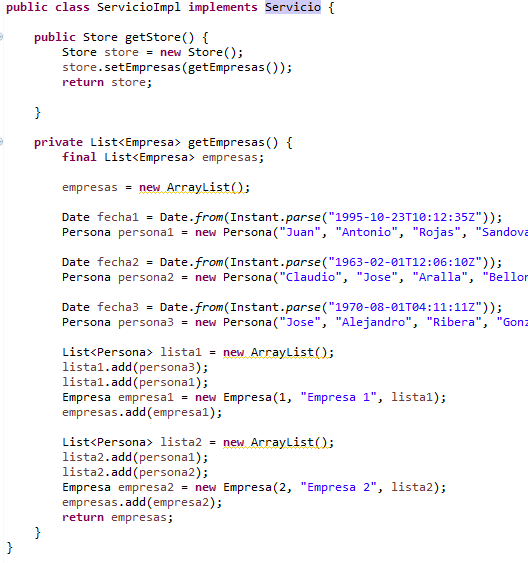
La clase principal



El controlador



El servicio



* 1. Implementación 3.
     1. Errores o falencias.

Para el objetivo del programa veo que hay muchas clases y se puede reescribir.

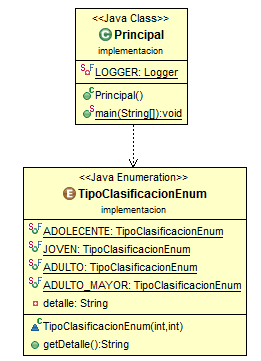
No se usa todo lo que un Enum puede proporcionar.

Exceso de complejidad.

* + 1. Mejoras.

Teniendo en cuenta el objetivo del programa propongo Reescribirlo usando mejor el Enum.

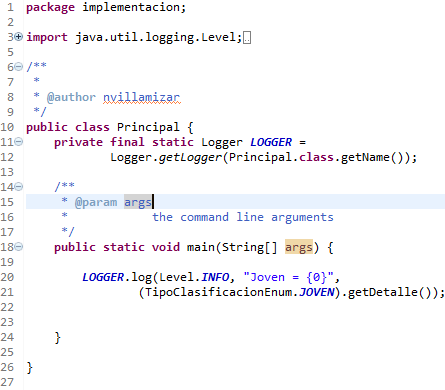
Se propone el siguiente diagrama de clases.



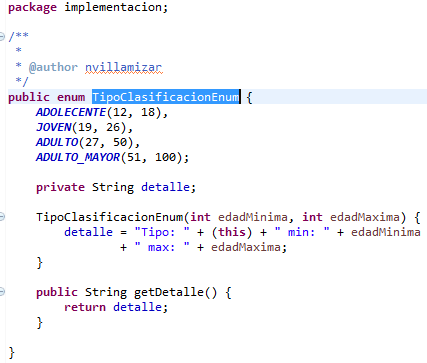
A continuación, en imágenes lo relevante de lo que se propone

Solo dos Archivos

La clase principal



Y el Enum. En donde se ponen los rangos de edad en el constructor del enum.

.

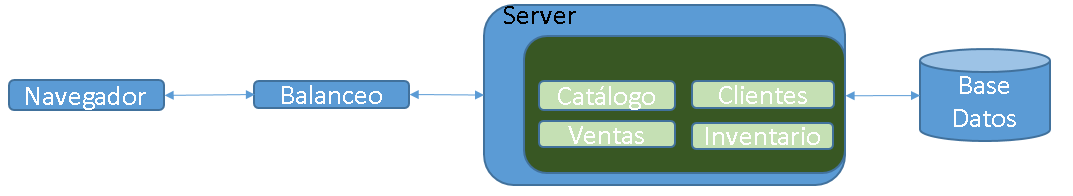
* 1. Implementación 4.
     1. Errores o falencias.
     2. Mejoras.

1. Para la quinta implementación se debe analizar, detallar los niveles y principios de arquitectura que usted cree están definidos en el código.
2. En el sexto punto se deben identificar los tipos de arquitectura.

Distribute the incoming traffic to the network by efficiently distributing the request across multiple servers

Reliability and high availability is maintained by redirecting these requests to the servers which are available

* 1. Diagrama 1



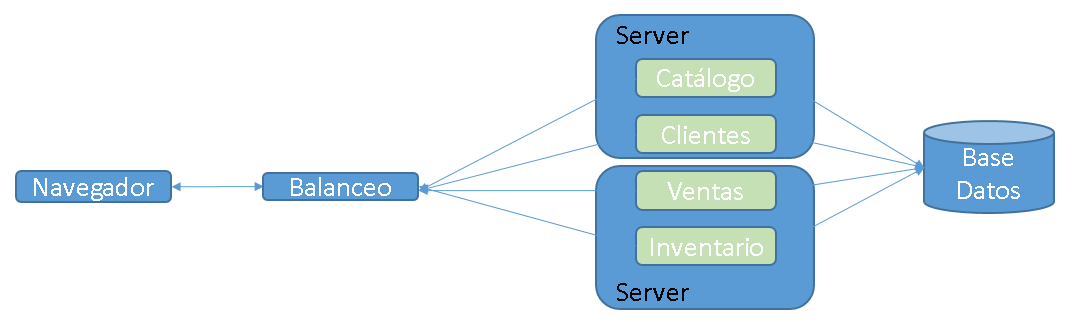
**Es una arquitectura monolítica**. En donde se ven diferentes componentes instalados en el mismo servidor usando la misma base de datos. No queda claro si hay comunicación entre los componentes, si hay, es en memoria a través del llamado a métodos.

Se observa Arquitectura de N niveles. Identifico claramente

* Capa de presentación
* Capa de negocio
* Capa de datos

La arquitectura monolítica tiene varias desventajas:

* Aplicaciones Grandes y complejas.
* Se debe reinstalar toda la aplicación en cada modificación.
* Difícil de escalar
* Son poco confiables, dado que si un componente falla podría fallar toda la aplicación.
  1. Diagrama 2



**Arquitectura de Servicios.**

En donde los servicios son componentes autónomos dentro del servidor. Se puede ver que cada servicio es una aplicación independiente. Todos los servicios se conectan a la misma base de datos

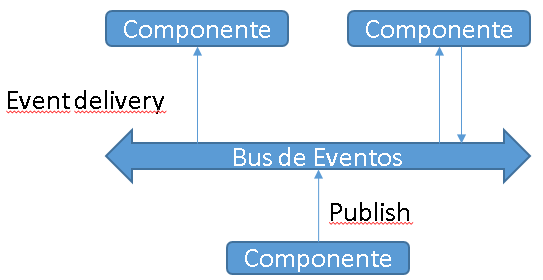
Se ve que hay una especie de Api Gateway entre el FrontEnd y el BackEnd de servicios que se encarga de llamar el servicio necesario para responderle al cliente (Navegador en este caso).

Las ventajas de los servicios:

* Agilidad en el desarrollo.
* Aplicaciones con poco código
* Cada servicio puede usar tecnologías diferente.
* Los servicios se pueden escalar de forma independiente.

.

* 1. Diagrama 3



Es un diagrama relacionado a arquitectura basada en eventos entre componentes (eventos de integración). Asumo que el componente es un microservicio.

Sistema de publicación y suscripción que expone un bus de eventos común donde los componentes se suscriben para recibir notificaciones de eventos. El sistema distribuido consiste de componentes publicadores y suscriptores  comunicándose usando el bus compartido.

Las ventajas son bajo acoplamiento y escalabilidad. Los publicadores nunca sabrán de la existencia de los suscriptores por tanto ambos pueden operar independientemente.

1. Por último indique cualesquiera condiciones adicionales que usted considere necesarios.