

Practica 2

Franquicia Funeraria

Programación orientada a objetos

Grupo: 01

Equipo: 5

Integrantes:

Violeta Gómez Arteaga
Sebastian Guerra Guerra
Andrés Pérez Cano

PROFESOR

Jaime Alberto Guzmán Luna

**Universidad Nacional de Colombia - Sede
Medellín 2024**

Índice

1. Portada	2
2. Descripción general de la solución.	3
3. Descripción del diseño estático del sistema en la especificación UML.	4
4. Descripción de la implementación de características de programación orientada a objetos en el proyecto.	5
5. Descripción de cada una de las 5 funcionalidades implementadas.	6
6. Manejo de excepciones	7
7. FieldFrame	8
8. Manual de usuario.	9

■

1. Descripción general de la solución.

El proyecto consiste en la simulación de una franquicia de funerarias, diseñado para organizar y coordinar diferentes servicios y procesos relacionados con la cremación, entierro, exhumación y administración financiera de las funerarias.

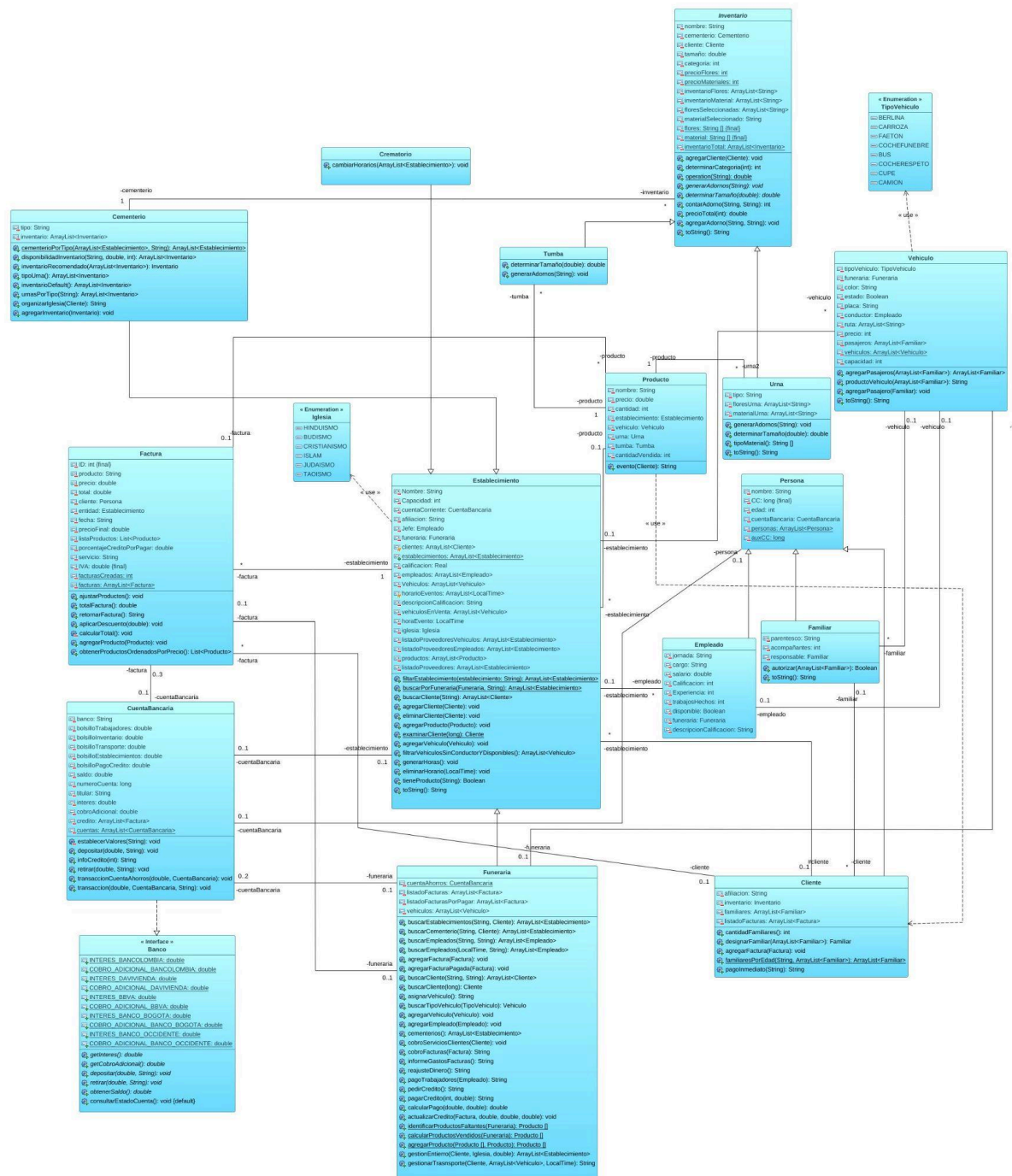
Gestionar Funerarias: Permite a los usuarios seleccionar y administrar funerarias, incluyendo la posibilidad de realizar intercambios de productos y gestionar inventarios y recursos, como la contratación de empleados y adquisición de vehículos.

Organizar Eventos Funerarios: Facilita la planificación y coordinación de eventos de cremación y entierro. Esto incluye la selección de clientes, crematorios, cementerios, urnas y la organización de ceremonias de acuerdo con las preferencias del cliente y requisitos específicos.

Realizar Exhumaciones: Ofrece la funcionalidad para gestionar el proceso de exhumación, permitiendo la selección de clientes y la organización del traslado a diferentes cementerios según el tipo de objeto asociado (urna o tumba).

Administrar Aspectos Financieros: Incluye la gestión de cobros, pagos de facturas y reajustes financieros entre las funerarias, utilizando cuentas bancarias y créditos según sea necesario. La plataforma asegura la correcta administración de los fondos y facilita el análisis financiero

https://app.genmymodel.com/api/projects/_dtC-4F3PEe-f95DqthbXog/diagrams/_dtC-413PEe-f95DqthbXog/svg



Nombre de la Clase: Cementerio

La clase Cementerio es parte del paquete gestorAplicacion.establecimientos y representa un establecimiento funerario que puede gestionar cementerios que tienen asociaciones con objetos de tipo Urna o Tumba:

Atributos:

- **tipo:** Un atributo String que indica si el cementerio maneja "cenizas" (usando urnas) o "cuerpos" (usando tumbas).
- **inventario:** Una lista de objetos de la clase Inventario que almacena urnas o tumbas disponibles en el cementerio, dependiendo del tipo.
- **cementerios:** Una lista estática que almacena todos los objetos Cementerio creados, lo que permite un acceso global a ellos.

Constructor:

El constructor de la clase inicializa un objeto Cementerio con sus atributos básicos, incluyendo su tipo (cenizas o cuerpos), capacidad, cuenta bancaria, afiliación, empleado responsable, y la funeraria a la que está asociado. Además, agrega el nuevo cementerio a la lista estática cementerios.

Métodos:

- **cementerioPorTipo:** Filtra y devuelve una lista de objetos Cementerio que coinciden con un tipo específico (cenizas o cuerpos).
- **disponibilidadInventario:** Filtra y devuelve una lista de objetos Inventario (urnas o tumbas) que cumplen con ciertos criterios de tamaño y edad del cliente.
- **inventarioRecomendado:** Selecciona y devuelve el objeto Inventario con el menor tamaño de entre una lista filtrada.
- **tipoUrna:** Filtra las urnas disponibles en el inventario del cementerio según la iglesia a la que está afiliado.
- **inventarioDefault:** Filtra y devuelve los objetos Inventario con nombre "default" que ya están asociados a un cliente.
- **urnasPorTipo:** Filtra y devuelve las urnas del inventario que coinciden con un tipo específico.
- **organizarIglesia:** Organiza la disposición de los familiares del cliente en la iglesia del cementerio, asignando flores y generando una factura por los productos utilizados.
- **agregarInventario:** Permite agregar objetos al inventario del cementerio, sean urnas o tumbas, dependiendo del tipo de cementerio.

Información adicional:

- La clase implementa la interfaz Serializable, lo que indica que los objetos de esta clase pueden ser serializados, es decir, convertidos en un formato que se puede almacenar o transmitir y luego reconstruir.
- La clase Cementerio extiende la clase Establecimiento, lo que significa que hereda todos los atributos y métodos de Establecimiento,
- La clase Crematorio es parte del paquete gestorAplicación.establecimientos y representa un establecimiento dedicado a la gestión de la cremación, extendiendo la funcionalidad de la clase Establecimiento.

Nombre de la Clase: Crematorio

Constructor:

El constructor de la clase Crematorio inicializa un objeto Crematorio con atributos básicos como el nombre,

capacidad, cuenta bancaria, afiliación, empleado responsable, y la funeraria a la que está asociado.

Métodos:

- **cambiarHorarios:** Este método toma como parámetro una lista de objetos Establecimiento, que se espera que sean cementerios. Modifica el horario de eventos de cada cementerio en la lista ingresada. Para cada cementerio, genera y agrega varias horas aleatorias al arreglo de horarios de eventos del cementerio, basándose en la duración del evento en la iglesia y utilizando números aleatorios para determinar las horas y minutos.

Nombre de la Clase: Establecimiento

Atributos Principales:

- **nombre:** Nombre del establecimiento.
- **capacidad:** Capacidad máxima del establecimiento.
- **cuentaCorriente:** Cuenta bancaria asociada al establecimiento.
- **afiliacion:** Información de afiliación del establecimiento.
- **Jefe:** Empleado que funge como jefe del establecimiento.
- **funeraria:** Funeraria a la que está asociado el establecimiento.
- **clientes:** Lista de clientes asociados al establecimiento.
- **empleados:** Lista de empleados que trabajan en el establecimiento.
- **Vehiculos:** Lista de vehículos pertenecientes al establecimiento.
- **productos:** Lista de productos disponibles en el establecimiento.
- **horarioEventos:** Lista de horarios para eventos en el establecimiento.
- **vehiculosEnVenta:** Vehículos que están disponibles para la venta.
- **listadoProveedores, listadoProveedoresVehiculos, listadoProveedoresEmpleados:** Listas que contienen otros establecimientos que funcionan como proveedores de productos, vehículos y empleados, respectivamente.
- **calificacion:** Calificación general del establecimiento.
- **descripcionCalificacion:** Descripción de la calificación del establecimiento.

Constructores:

- **Establecimiento(String nombre, int capacidad, CuentaBancaria cuentaCorriente, String afiliacion, Empleado jefe, Funeraria funeraria):** Constructor completo que inicializa todos los atributos principales.
- **Establecimiento(String nombre, int capacidad, CuentaBancaria cuentaCorriente, Empleado jefe, double calificacion):** Constructor que omite la funeraria y la afiliación, pero permite definir una calificación inicial.
- **Establecimiento(String nombre, CuentaBancaria cuentaCorriente):** Constructor simplificado, utilizado para inicializar un establecimiento con solo el nombre y la cuenta corriente.

Métodos Principales:

- **filtrarEstablecimiento(String establecimiento):** Filtra los establecimientos por su tipo (cementerio, crematorio, funeraria).
- **buscarPorFuneraria(Funeraria funeraria, String tipoEstablecimiento):** Filtra los establecimientos de acuerdo a la funeraria asociada y al tipo de establecimiento.
- **buscarCliente(String adultoNiño):** Filtra clientes según su edad (adulto o niño) basándose en su número de identificación.
- **examinarCliente(long CC):** Busca un cliente en todos los establecimientos de tipo funeraria y cementerio por su número de identificación.
- **agregarCliente(Cliente cliente) y eliminarCliente(Cliente cliente):** Métodos para agregar o eliminar clientes del establecimiento.

- `agregarVehiculo(Vehiculo vehiculo)` y `filtrarVehiculosSinConductorYDisponibles()`: Gestionan la lista de vehículos del establecimiento y filtran aquellos que están disponibles y sin conductor.
- `generarHoras()`: Genera horarios de trabajo aleatorios para empleados con roles específicos (sepulturero o cremador).
- `tieneProducto(String nombreProducto)`: Verifica si el establecimiento tiene un producto específico.
- `agregarProveedorEmpleado(Establecimiento proveedor)`, `agregarProveedorVehiculos(Establecimiento proveedor)`, `agregarProveedores(Establecimiento proveedor)`: Agregan establecimientos como proveedores de empleados, vehículos o productos, respectivamente.

Nombre de la Clase: Funeraria

La clase *Funeraria* es parte de un sistema de gestión para establecimientos, específicamente funerarias, y extiende de la clase *Establecimiento*.

Atributos

1. **cuentaAhorros**: Es un atributo estático que representa la cuenta de ahorros asociada a la funeraria. Aunque es común tener una cuenta de ahorros por funeraria, el hecho de que sea estático implica que todas las instancias de *Funeraria* compartirán la misma cuenta de ahorros.
2. **empleados**: Es una lista que almacena los empleados que trabajan en la funeraria.
3. **listadoFacturas**: Es una lista que almacena las facturas que ya han sido pagadas.
4. **listadoFacturasPorPagar**: Es una lista que almacena las facturas que aún están pendientes de pago.
5. **vehiculos**: Es una lista que almacena los vehículos asociados a la funeraria.

Constructor

El constructor de la clase *Funeraria* recibe tres parámetros:

- **nombre**: El nombre de la funeraria.
- **cuentaCorriente**: La cuenta corriente asociada a la funeraria.
- **cuentaAhorros**: La cuenta de ahorros asociada a la funeraria.

Este constructor inicializa la cuenta de ahorros y pasa el nombre y la cuenta corriente al constructor de la clase padre *Establecimiento*.

Métodos mas relevantes

1. **buscarEstablecimientos**: Este método recibe un tipo de establecimiento (como "crematorio" o "cementerio") y un cliente. Busca y devuelve una lista de establecimientos asociados a la funeraria que cumplen con dos condiciones:
 - La afiliación del cliente debe coincidir con la del establecimiento.
 - La capacidad del establecimiento debe ser mayor o igual al número de familiares del cliente.
2. **buscarCementerios**: Este método busca cementerios que sean adecuados según el tipo (como "cuerpos" o "cenizas") y el cliente. Devuelve una lista de cementerios filtrados por estas condiciones.
3. **buscarEmpleados**: Hay dos versiones de este método:
 - La primera versión recibe la jornada (como "mañana", "tarde", o "noche") y el cargo (como "conductor", "sepulturero", etc.), y devuelve una lista de empleados que coinciden con esos criterios.
 - La segunda versión está sobrecargada y recibe una hora (*LocalTime*) y un cargo. Determina la jornada basada en la hora y luego filtra los empleados por cargo y jornada.
4. **agregarFactura** y **agregarFacturapagada**: Estos métodos permiten agregar facturas a las listas **listadoFacturasPorPagar** y **listadoFacturas**, respectivamente.
5. **buscarCliente**: Este método busca clientes en los cementerios asociados a la funeraria, filtrados por

el tipo de cementerio y si el cliente es un adulto o un niño.

Nombre de la Clase: Iglesia

La clase Iglesia es una enumeración (enum) en Java que define diferentes tipos de templos religiosos asociados a varias religiones. Cada constante de esta enumeración representa una iglesia específica con características particulares, como el nombre, la cantidad de sillas, la posibilidad de cremación, la duración de eventos, los tipos de urnas disponibles, y los líderes religiosos asociados.

Constantes de la Enumeración

Las constantes de la enumeración son:

- HINDUISMO
- BUDISMO
- CRISTIANISMO:
- ISLAM
- JUDAISMO
- TAOISMO

Cada uno de estos templos tiene atributos específicos que definen sus características.

Atributos

- nombre: Nombre del templo o iglesia.
- sillas: Número de sillas disponibles en el templo.
- cremacion: Un valor booleano que indica si el templo permite la cremación.
- duracionEvento: Duración en horas de un evento religioso típico en el templo.
- tipoUrnas: Un arreglo de cadenas que especifica los tipos de urnas disponibles en el templo para cenizas, si se permite la cremación.
- religioso: Título del líder religioso que oficia en el templo.
- religiosoAltoRango: Título del líder religioso de mayor rango asociado al templo.

Constructor

El constructor de Iglesia es privado y se utiliza para inicializar las constantes de la enumeración. Este recibe los siguientes parámetros:

1. nombre: Nombre del templo.
2. sillas: Número de sillas disponibles en el templo.
3. cremacion: Indica si el templo permite cremaciones.
4. duracionEvento: Duración típica de un evento religioso.
5. tipoUrnas: Arreglo con los tipos de urnas disponibles para las cenizas, si se permiten cremaciones.
6. religioso: Título del líder religioso principal.
7. religiosoAltoRango: Título del líder religioso de mayor rango.

Métodos

- getDuracionCremacion(): Devuelve la duración de un evento en el templo, en horas.
- duracionEvento(LocalTime horaInicio): Calcula y devuelve la hora de finalización de un evento en función de la hora de inicio y la duración del evento.

Nombre de la clase: Banco

La clase Banco define un contrato para la implementación de funcionalidades básicas que los diferentes tipos de bancos deben cumplir dentro del sistema. Esta clase establece constantes que representarán los

atributos comunes entre distintas entidades bancarias, como intereses, cobros adicionales Constantes

Las constantes definidas en la clase representan valores fijos de intereses y cobros adicionales asociados a diferentes bancos. Estas son:

- **INTERES_BANCOLOMBIA:** Tasa de interés asociada a BANCOLOMBIA (0.0005).
- **COBRO_ADICIONAL_BANCOLOMBIA:** Cobro adicional asociado a BANCOLOMBIA (1800).
- **INTERES_DAVIVIENDA:** Tasa de interés asociada a DAVIVIENDA (0.08).
- **COBRO_ADICIONAL_DAVIVIENDA:** Cobro adicional asociado a DAVIVIENDA (2000).
- **INTERES_BBVA:** Tasa de interés asociada a BBVA (0.005).
- **COBRO_ADICIONAL_BBVA:** Cobro adicional asociado a BBVA (1500).
- **INTERES_BANCO_BOGOTA:** Tasa de interés asociada a BANCO DE BOGOTÁ (0.0001).
- **COBRO_ADICIONAL_BANCO_BOGOTA:** Cobro adicional asociado a BANCO DE BOGOTÁ (1600).
- **INTERES_BANCO_OCCIDENTE:** Tasa de interés asociada a BANCO DE OCCIDENTE (0.02).
- **COBRO_ADICIONAL_BANCO_OCCIDENTE:** Cobro adicional asociado a BANCO DE OCCIDENTE (2200).

Estas constantes son valores predefinidos y final (implícitamente estáticos y públicos), lo que significa que no pueden ser modificados una vez definidos.

Nombre de la Clase: CuentaBancaria

La clase CuentaBancaria hereda de la clase Banco y representa una cuenta bancaria. Atributos Principales

- **banco:** El nombre del banco asociado a la cuenta.
- **bolsilloTrabajadores, bolsilloInventario, bolsilloTransporte, bolsilloEstablecimientos, bolsilloPagoCredito:** Son "bolsillos" o subcuentas dentro de la cuenta principal que se utilizan para separar y gestionar diferentes tipos de fondos.
- **saldo:** El saldo total de la cuenta, que es la suma de todos los bolsillos.
- **numeroCuenta:** Número de cuenta bancaria.
- **titular:** El titular o propietario de la cuenta.
- **interes:** El porcentaje de interés asociado con el banco.
- **cobroAdicional:** Un cargo adicional aplicado en ciertas transacciones.
- **credito:** Una lista de objetos Factura que representa las facturas asociadas con la cuenta.
- **cuentas:** Una lista estática que mantiene un registro de todas las cuentas creadas.

Constructores

1. **CuentaBancaria(long numeroCuenta, String titular, double saldoInicial, String banco):**
 - Inicializa una cuenta bancaria con un número de cuenta, titular, saldo inicial y banco.
 - Llama a establecerValores(banco) para configurar el interés y el cobro adicional según el banco.
2. **CuentaBancaria(long numeroCuenta, String titular, double bolsilloTrabajadores, double bolsilloInventario, double bolsilloTransporte, double bolsilloEstablecimientos, double bolsilloPagoCredito, String banco):**
 - Inicializa la cuenta con valores específicos para cada bolsillo y luego calcula el saldo total sumando estos valores.

Métodos Principales

- **depositar(double cantidad, String tipo):** Deposita una cantidad en la cuenta, ya sea en el saldo general o en uno de los bolsillos específicos.
- **retirar(double cantidad, String tipo):** Retira una cantidad de la cuenta, ya sea del saldo general o de uno de los bolsillos.
- **transaccionCuentaAhorros(double valor, CuentaBancaria cuentaAhorros):** Realiza una

transferencia entre la cuenta corriente y una cuenta de ahorros. Si el banco de destino es diferente, se aplica un cobro adicional y el saldo de la cuenta de destino se ajusta según el interés.

- **transaccion(double valor, CuentaBancaria cuentaCorriente, String tipo):** Similar a la anterior, pero permite la transferencia entre cualquier tipo de cuenta.
- **establecerValores(String banco):** Configura el interés y el cobro adicional en función del banco asociado.
- **infoCredito(int credito):** Proporciona información sobre una factura específica de la lista de créditos.

Nombre de la Clase: Factura

La clase Factura representa una factura en el sistema de gestión de inventario y ventas. Atributos

- **ID:** Identificador único para cada factura, asignado automáticamente al crear una nueva factura.
- **producto:** Nombre del producto asociado a la factura.
- **precio:** Precio del producto antes de aplicar IVA u otros ajustes.
- **total:** Total de la factura, calculado como precio más IVA.
- **cliente:** Objeto Persona que representa al cliente que realiza la compra.
- **Entidad:** Objeto Establecimiento que está asociado con la factura.
- **Fecha:** Fecha en la que se emite la factura.
- **precioFinal:** Precio final después de aplicar descuentos o ajustes adicionales.
- **listaProductos:** Lista de objetos Producto asociados con la factura.
- **porcentajeCreditoPorPagar:** Porcentaje del crédito que queda por pagar.
- **Servicio:** Tipo de servicio relacionado con la factura.
- **IVA:** Tasa de IVA fija aplicada a la factura (19%).
- **facturasCreadas:** Contador estático que lleva un registro de cuántas facturas se han creado.
- **facturas:** Lista estática de todas las facturas creadas.

Constructores

1. **Factura(String producto, double precio, String Fecha, Persona cliente, String Servicio):** Crea una factura con un producto, precio, fecha, cliente y servicio. Calcula el total y asigna un ID único a la factura.
2. **Factura(String nombre):** Crea una factura basada en un nombre de producto, con un total inicial de 0. Asigna un ID único y el servicio predeterminado "Inventario".
3. **Factura(String producto, double precio, String Fecha, Establecimiento Entidad, String Servicio):** Crea una factura con un producto, precio, fecha, establecimiento y servicio. Calcula el total y asigna un ID único.
4. **Factura(ArrayList<Producto> productos):** Crea una factura a partir de una lista de productos. Ajusta los productos para evitar duplicados y calcula el total.
5. **Factura(ArrayList<Producto> productos, String servicio):** Similar al constructor anterior, pero también asigna un servicio específico.

Métodos

- **ajustarProductos():** Ajusta la lista de productos para consolidar los productos con el mismo nombre y sumar sus cantidades. Usa un HashMap para contar las ocurrencias de cada nombre.
- **totalFactura():** Calcula el total de la factura sumando el precio de cada producto multiplicado por su cantidad.
- **retornarFactura():** Retorna una cadena con el resumen de los productos de la factura, incluyendo nombre, precio unitario, cantidad y el total.
- **aplicarDescuento(double porcentaje):** Aplica un descuento al precio de la factura. El porcentaje debe estar entre 0 y 100.
- **calcularTotal():** Calcula el total de la factura incluyendo el IVA (19%).
- **agregarProducto(Producto producto):** Agrega un producto a la lista de productos de la factura y

actualiza el precio total.

- **obtenerProductosOrdenadosPorPrecio():** Retorna una lista de productos ordenada por precio en orden descendente.

Nombre de la Clase: Inventario

La clase Inventario es una clase abstracta que representa el inventario en el contexto de un cementerio o funeraria. Gestiona la disponibilidad y los detalles de adornos, como flores y materiales.

Atributos

- **nombre:** Nombre del inventario.
- **cementerio:** Objeto Cementerio asociado con este inventario.
- **cliente:** Objeto Cliente asociado con el inventario.
- **tamaño:** Tamaño del inventario, representado como un número.
- **categoría:** Categoría del inventario, determinada por la edad del cliente.
- **precioFlores:** Precio base para las flores.
- **precioMateriales:** Precio base para los materiales.
- **inventarioFlores:** Lista de flores disponibles en el inventario.
- **inventarioMaterial:** Lista de materiales disponibles en el inventario.
- **floresSeleccionadas:** Flores seleccionadas por el cliente.
- **materialSeleccionado:** Material seleccionado por el cliente.
- **flores:** Arreglo estático de nombres de flores disponibles.
- **material:** Arreglo estático de nombres de materiales disponibles.
- **inventarioTotal:** Lista estática de todos los inventarios creados.

Constructor

- **Inventario(String nombre, Cementerio cementerio, double tamaño, int categoria):** Inicializa el inventario con un nombre, un cementerio, un tamaño y una categoría. También agrega el inventario a la lista estática **inventarioTotal**.

Métodos

- **agregarCliente(Cliente cliente):** Asocia un cliente al inventario y actualiza las listas de clientes en el cementerio y la funeraria.
- **determinarCategoria(int edad):** Determina la categoría del cliente basada en su edad. Retorna 0 si es menor de 18, 1 si está entre 18 y 60, y 2 si es mayor de 60.
- **precios(String adorno):** Retorna el precio de un adorno (flores o material). El precio base se ajusta según el índice del adorno en los arreglos estáticos flores y material.
- **generarAdornos(String tipoAdorno):** Método abstracto para generar adornos. Debe ser implementado en una subclase.
- **determinarTamaño(double tamaño):** Método abstracto para determinar el tamaño del inventario. Debe ser implementado en una subclase.
- **contarAdorno(String adorno, String floresMaterial):** Cuenta cuántas veces aparece un adorno específico en el inventario de flores o materiales.
- **precioTotal(int cantidadF):** Calcula el precio total de los adornos basado en la cantidad y los precios de flores y materiales.
- **agregarAdorno(String adorno, String floresMaterial):** Agrega un adorno a la lista de flores seleccionadas o a la lista de materiales seleccionados, según el tipo especificado.
- **toString():** Retorna el nombre del inventario como una representación en cadena.

Nombre de la Clase: Producto

A través de los diversos constructores, la clase puede ser utilizada para crear instancias de productos con diferentes características y asociaciones.

Atributos

- **nombre:** Nombre del producto.
- **precio:** Precio del producto.
- **cantidad:** Cantidad disponible del producto.
- **cantidadVendida:** Cantidad del producto que ha sido vendida.
- **establecimiento:** Objeto Establecimiento con el que el producto está asociado, puede ser un Crematorio o un Cementerio.
- **vehiculo:** Objeto Vehiculo relacionado con el producto.
- **urna:** Objeto Urna relacionado con el producto.
- **tumba:** Objeto Tumba relacionado con el producto.
- **productos:** Lista estática que mantiene un registro de todos los productos creados.

Constructores

- **Producto(String nombre, double precio, int cantidad, int cantidadVendida):** Inicializa un producto con un nombre, precio, cantidad disponible y cantidad vendida. Agrega el producto a la lista productos.
- **Producto(String nombre, double precio, int cantidad):** Inicializa un producto con un nombre, precio y cantidad disponible. Agrega el producto a la lista productos.
- **Producto(String nombre, double precio, int cantidad, Establecimiento establecimiento):** Inicializa un producto con un nombre, precio, cantidad disponible y un establecimiento asociado. Agrega el producto a la lista productos.
- **Producto(Establecimiento establecimiento):** Inicializa un producto basado en un establecimiento, configurando el nombre del producto como el nombre del establecimiento y la cantidad como 1. Agrega el producto a la lista productos.
- **Producto(Urna urna, int cantidadFlores):** Inicializa un producto con una urna y una cantidad de flores, calculando el precio total a partir de la urna. Agrega el producto a la lista productos.
- **Producto(Tumba tumba, int cantidadFlores):** Inicializa un producto con una tumba y una cantidad de flores, calculando el precio total a partir de la tumba. Agrega el producto a la lista productos.
- **Producto(Vehiculo vehiculo):** Inicializa un producto con un vehículo, configurando el nombre y precio del producto según el vehículo. Agrega el producto a la lista productos.
- **Producto(Vehiculo vehiculo, Establecimiento establecimiento):** Inicializa un producto con un vehículo y un establecimiento asociado, configurando el nombre y precio del producto según el vehículo. Agrega el producto a la lista productos.
- **Producto(String nombre, int cantidadVendida):** Inicializa un producto con un nombre y cantidad vendida. Agrega el producto a la lista productos.

Métodos

- **evento(Cliente cliente):** Genera una descripción detallada del evento (cremación o entierro) para el cliente. Incluye la hora del evento, lugar, nombre de la iglesia, y los familiares del cliente.

Nombre de la Clase: TipoVehiculo

La clase TipoVehiculo es una enumeración (enum) en python que define distintos tipos de vehículos disponibles en el sistema. Las enumeraciones en python son una forma de definir un conjunto fijo de constantes, en este caso, tipos de vehículos con atributos específicos.

Enumeraciones Definidas

1. **BERLINA:** Capacidad para 4 personas, permite agregar tanto al cliente como a familiares. Precio: 70,000.
2. **CARROZA:** Capacidad para 6 personas, permite solo al cliente. Precio: 150,000.
3. **FAETON:** Capacidad para 4 personas, permite agregar tanto al cliente como a familiares. Precio:

120,000.

4. **COCHEFUNEBRE:** Capacidad para 1 persona, solo el cliente. Precio: 80,000.
5. **BUS:** Capacidad para 6 personas, permite agregar tanto al cliente como a familiares. Precio: 50,000.
6. **COCHERESPETO:** Capacidad para 8 personas, solo familiares. Precio: 75,000.
7. **CUPE:** Capacidad para 4 personas, solo el cliente. Precio: 65,000.
8. **CAMION:** Capacidad para 5 personas, solo el cliente. Precio: 69,000.

Atributos

- **capacidad:** Número máximo de personas que el vehículo puede transportar.
- **cliente:** Booleano que indica si el vehículo permite agregar al cliente (true o false).
- **familiar:** Booleano que indica si el vehículo permite agregar familiares (true o false).
- **precio:** Precio del vehículo.

Nombre de la Clase: Tumba

La clase **Tumba** simula una tumba asociada a un cementerio en un sistema de gestión de inventario para cementerios. Esta clase permite calcular el tamaño de la tumba, generar adornos y gestionarla dentro del cementerio.

Atributos

- **Nombre:** Nombre de la tumba.
- **Cementerio:** El cementerio al que está asociada la tumba.
- **Tamaño:** El tamaño calculado de la tumba.
- **Categoría:** Una categoría que puede influir en los adornos asociados a la tumba.

Constructor

El constructor inicializa la tumba con un nombre, cementerio, tamaño y categoría. Si el cementerio al que está asociada la tumba es del tipo "cuerpos", la tumba se añade al inventario del cementerio.

Métodos

- **Determinar Tamaño:** Calcula el volumen de la tumba utilizando dimensiones estándar y la longitud proporcionada. El volumen se basa en medidas fijas para ancho y profundidad, y se ajusta en función del largo especificado.
- **Generar Adornos:** Permite añadir adornos a la tumba. Dependiendo del tipo de adorno (flores o materiales) y la categoría de la tumba, se generan adornos aleatoriamente y se añaden al inventario correspondiente. El método utiliza aleatoriedad para determinar cuántos adornos se agregan y de qué tipo.

Nombre de la Clase: Urna

La clase **Urna** se utiliza para manejar urnas en un cementerio. Estas urnas pueden tener diferentes tipos y características, y la clase proporciona métodos para calcular su tamaño, gestionar adornos, y otros aspectos relacionados con las urnas.

Atributos

- **Tipo:** El tipo de urna, que puede ser "fija" u "ordinaria".
- **FloresUrna:** Una lista de flores asociadas a la urna.
- **MaterialUrna:** Una lista de materiales asociados a la urna.

Constructor

El constructor inicializa la urna con un nombre, cementerio, peso, categoría y tipo. Si el cementerio es del tipo "cenizas", la urna se añade al inventario del cementerio.

Métodos

- **Generar Adornos:** Dependiendo del tipo de adorno (flores o material), este método agrega adornos a la urna. Utiliza aleatoriedad para seleccionar y añadir adornos a las listas correspondientes si están vacías.
- **Determinar Tamaño:** Calcula el volumen necesario para la urna basado en el peso del cliente. Se estima que se necesita un cierto volumen de espacio por cada unidad de peso, y este volumen se calcula en centímetros cúbicos.
- **Tipo Material:** Devuelve un array de materiales disponibles para la urna basado en el tipo de urna. Para urnas de tipo "fija", se devuelven materiales específicos como vidrio, bambú y piedra; para urnas ordinarias, se devuelven materiales como madera, metal y cerámica.
- **Determinar Categoría:** Determina la categoría de la urna basándose en el número de adornos. La categoría se asigna en función de la cantidad de adornos presentes.
- **toString:** Proporciona una representación en cadena de texto del objeto **Urna**, incluyendo su nombre y tipo.

Nombre de la Clase: Vehículo

La clase **Vehículo** modela vehículos utilizados por una funeraria para el transporte de familiares. Ofrece métodos para asignar pasajeros, gestionar la capacidad del vehículo, y proporcionar detalles sobre el vehículo.

Atributos

- **TipoVehículo:** Un enumerado que define el tipo de vehículo (berlina, carroza, bus).
- **Funeraria:** La funeraria a la que pertenece el vehículo.
- **Color:** El color del vehículo.
- **Estado:** Indica si el vehículo está disponible (**true**) o no (**false**).
- **Placa:** La matrícula del vehículo.
- **Conductor:** El empleado que conduce el vehículo.
- **Ruta:** La ruta asignada al vehículo (en formato de lista de cadenas).
- **Precio:** El costo asociado al vehículo.
- **Capacidad:** La capacidad máxima del vehículo (número de pasajeros que puede transportar).
- **Pasajeros:** Lista de familiares que están en el vehículo.

Constructores

- **Vehículo(TipoVehículo, Funeraria, String, String, int, int):** Inicializa un vehículo con tipo, funeraria, color, placa, precio y capacidad.
- **Vehículo(TipoVehículo, Funeraria, String, String, int):** Inicializa un vehículo con tipo, funeraria, color, placa y capacidad, sin especificar precio.

Métodos

- **Agregar Pasajeros:** Este método asigna familiares al vehículo basado en la capacidad del mismo. Prioriza agregar familiares menores con sus responsables y luego agrega adultos si hay capacidad restante.
- **Producto Vehículo:** Devuelve una descripción del vehículo y sus pasajeros en

formato de cadena de texto.

- **Agregar Pasajero:** Añade un pasajero al vehículo y cambia el estado del vehículo a no disponible (**estado=false**).
- **Determinar Estado:** Métodos **getEstado()** y **setEstado(Boolean estado)** para obtener y establecer el estado del vehículo.
- **Obtener/Establecer Atributos:** Métodos **get** y **set** para los atributos **TipoVehiculo**, **Placa**, **Conductor**, **Ruta**, **Pasajeros**, **Funeraria**, **Color**, **Capacidad**, y **Precio**.

Nombre de la Clase: Cliente

La clase **Cliente** modela a los clientes de una funeraria y proporciona funcionalidades para gestionar la información sobre su afiliación, familiares, y facturas. Además, permite realizar pagos inmediatos por adornos.

Atributos

- **Afiliación:** Tipo de afiliación del cliente (por ejemplo, oro, plata, bronce).
- **Inventario:** Un objeto que representa el inventario asociado con el cliente.
- **Familiares:** Una lista de familiares del cliente.
- **Listado de Facturas:** Una lista de facturas asociadas con el cliente.

Constructores

- **Cliente(String nombre, long CC, int edad, CuentaBancaria cuentaBancaria, String afiliacion, ArrayList<Familiar> familiares):** Constructor para clientes mayores de edad. Inicializa el nombre, cédula, edad, cuenta bancaria, afiliación, y familiares.
- **Cliente(String nombre, int edad, String plan, ArrayList<Familiar> familiares):** Constructor para clientes menores de edad. Inicializa el nombre, edad, plan, y familiares. La cédula y la cuenta bancaria se establecen en valores predeterminados.

Métodos

- **Cantidad de Familiares:** Calcula el número total de familiares y acompañantes del cliente. Suma la cantidad de acompañantes de cada familiar a la lista de familiares.
- **Designar Familiar:** Retorna el familiar con el parentesco más cercano (como cónyuge, hijo, padre, hermano) desde una lista de familiares.
- **Agregar Factura:** Añade una factura al listado de facturas del cliente.
- **Familiares por Edad:** Filtra y retorna una lista de familiares según si son adultos o niños.
- **Pago Inmediato:** Realiza el pago inmediato de adornos generados. Genera una factura basada en el tipo de adorno (flores o material) y verifica si el familiar designado tiene suficiente saldo para cubrir el total de la factura. Si es así, se realiza la transacción. Retorna una descripción del pago y la factura generada.

Nombre de la Clase: Empleado

La clase **Empleado** extiende de **Persona** y modela a los empleados de una funeraria, permitiendo gestionar información sobre su jornada laboral, cargo, salario, y otros atributos relevantes.

Atributos

- **Jornada:** Indica el turno del empleado (mañana, tarde, noche).
- **Cargo:** El puesto que ocupa el empleado (por ejemplo, sepulturero, cremador, padre, obispo, conductor).
- **Salario:** La remuneración del empleado.
- **Calificación:** Una puntuación que representa el desempeño del empleado. Inicialmente está en 5.
- **Experiencia:** Años de experiencia del empleado.
- **Trabajos Hechos:** Número de trabajos realizados por el empleado.
- **Disponible:** Estado de disponibilidad del empleado (true si está disponible, false si no lo está).
- **Funeraria:** La funeraria en la que trabaja el empleado.
- **Descripción de Calificación:** Descripción de la calificación del empleado, utilizada en encuestas.

Constructores

- **Empleado(String nombre, CuentaBancaria cuentaBancaria, String jornada, String cargo, double salario, Funeraria funeraria):** Constructor que inicializa el nombre, cuenta bancaria, jornada, cargo, salario y opcionalmente agrega al empleado a una funeraria.
- **Empleado(String nombre, CuentaBancaria cuentaBancaria, String jornada, String cargo, double salario):** Constructor que inicializa el nombre, cuenta bancaria, jornada, cargo y salario sin agregar al empleado a una funeraria.
- **Empleado(String nombre, int edad, CuentaBancaria cuentaBancaria, String jornada, String cargo, double salario, int Experiencia, int trabajosHechos):** Constructor que además de los parámetros anteriores, inicializa la edad, experiencia y trabajos hechos del empleado.

Nombre de la Clase: Familiar

La clase **Familiar** extiende de **Persona** y se utiliza para representar a los miembros de la familia de un cliente, diferenciando entre mayores y menores de edad, así como gestionando aspectos relacionados con el parentesco y la autorización para ciertos trámites.

Atributos

- **Parentesco:** El tipo de parentesco del familiar con el cliente (por ejemplo, cónyuge, hijo, padre, etc.).
- **Acompañantes:** Número de personas adicionales que acompañan al familiar.
- **Responsable:** En el caso de los menores de edad, este atributo representa al familiar que se encarga de ellos.

Constructores

- **Familiar(String nombre, long CC, int edad, CuentaBancaria cuentaBancaria, String parentesco, int acompañantes):** Constructor para mayores de edad. Inicializa el nombre, número de cédula, edad, cuenta bancaria, parentesco y número de acompañantes.
- **Familiar(String nombre, int edad, String parentesco, Familiar responsable):** Constructor para menores de edad. Inicializa el nombre, edad, parentesco y el familiar responsable. El número de cédula y la cuenta bancaria son nulos o cero en este caso, ya que los menores no tienen estos atributos.

Métodos

- **autorizar(ArrayList<Familiar> familiares):** Método que simula la autorización basada en el parentesco del familiar. Si el parentesco es "conyuge" y un número aleatorio es menor que 6, retorna **true**; en caso contrario, retorna **false**. Este método podría usarse para validar permisos o autorizaciones específicas.
- **toString():** Método que devuelve una representación en cadena del objeto **Familiar**. Devuelve el parentesco en mayúsculas seguido del nombre del familiar.

Nombre de la Clase: Persona

La clase **Persona** encapsula los datos y comportamientos comunes a todas las personas en el sistema, y sirve como una clase base para las clases **Empleado**, **Familiar** y **Cliente**.

Atributos

- **nombre:** El nombre de la persona.
- **CC:** Número de cédula o identificación de la persona. Este atributo es **final**, lo que significa que su valor no puede cambiar una vez asignado.
- **edad:** Edad de la persona.
- **cuentaBancaria:** Objeto de la clase **CuentaBancaria** asociado a la persona, que maneja las transacciones financieras.
- **personas:** Una lista estática (**ArrayList**) que mantiene un registro de todas las instancias de **Persona** creadas.
- **auxCC:** Un valor estático utilizado para asignar números de cédula únicos automáticamente (aunque en el código proporcionado no se usa directamente).

Constructores

- **Persona(String nombre, long CC, int edad, CuentaBancaria cuentaBancaria):** Constructor principal que inicializa los atributos de la persona. También agrega la instancia a la lista estática **personas** y actualiza el contador **auxCC**.

Métodos

- **getPersonas():** Retorna la lista estática de todas las personas registradas en el sistema.
- **toString():** Devuelve el nombre de la persona como una representación en cadena del objeto.
- **getNombre() y setNombre(String nombre):** Permiten acceder y modificar el nombre de la persona.
- **getCC():** Retorna el número de cédula, que es inmutable.
- **getEdad() y setEdad(int edad):** Permiten acceder y modificar la edad de la persona.
- **getCuentaBancaria() y setCuentaBancaria(CuentaBancaria cuentaBancaria):** Permiten acceder y modificar la cuenta bancaria asociada a la persona.

3. Descripción de la implementación de características de programación orientada a objetos en el proyecto.

*Clase abstracta y métodos abstractos

```
gestorAplicacion > inventario >  inventario.py
1  from abc import ABC, abstractmethod
2
3  #from gestorAplicacion.personas.cliente import Cliente
4
5  class Inventario(ABC):
6      precioFlores = 35000
7      precioMateriales = 45000
8
9      flores = ["Rosas", "Lirios", "Claveles", "Orquídeas", "Peonías"]
10     material = ["Madera", "Metal", "Cerámica", "Vidrio", "Bambu", "Piedra"]
```

gestorAplicacion.inventario.inventario; clase abstracta Inventario línea 5

La clase abstracta gestiona datos y operaciones que son relevantes para todos los inventarios, como el manejo de flores y materiales, y el almacenamiento de inventarios totales (inventarioTotal). Esto asegura que todas las subclases tengan acceso a esta información compartida sin tener que duplicar el código.

```
56
57     @abstractmethod
58     def generarAdornos(self, tipoAdorno: str):
59         pass
60
61     @abstractmethod
62     def determinarTamaño(self, tamaño: float) -> float:
63         pass
64
```

gestorAplicacion.inventario.inventario; clase abstracta Inventario líneas 57-63

generarAdornos(self, tipoAdorno:str) y determinarTamaño(self, tamaño:float). Estos métodos no tienen implementación en la clase abstracta, sino que deben ser implementados por las subclases (en este caso, Urna). Esto permite que cada tipo específico de inventario tenga su propia implementación para estos métodos, adaptada a sus necesidades.

*Herencia:

```

gestorAplicacion > personas > familiar.py
1
2 from gestorAplicacion.personas.persona import Persona
3
4
5 class Familiar(Persona):
6     def __init__(self, nombre, cc=0, edad=0, cuentaBancaria=None,
7         parentesco=None, acompañantes=0, responsable=None):
8         super().__init__(nombre, cc if cc is not None else 0, edad, cuentaB

```

gestorAplicacion.personas.familiar; clase Familiar línea 5

```

gestorAplicacion > establecimientos > crematorio.py
1 import random
2 from datetime import time
3 import time
4
5 from gestorAplicacion.establecimientos.establecimiento import Establecimien
6 #from gestorAplicacion.establecimientos.cementerio import Cementerio
7
8 class Crematorio(Establecimiento):
9     def __init__(self, nombre, capacidad, cuentaCorriente, afiliacion, empl
10         super().__init__(nombre, capacidad, cuentaCorriente, afiliacion, em

```

gestorAplicacion.establecimientos.crematorio; clase Familiar línea 8

*Atributos de clase y métodos de clase

```

gestorAplicacion > establecimientos > funeraria.py
1 from typing import List, Optional
2 from datetime import time
3 from gestorAplicacion.establecimientos.establecimiento import Establecimie
4 #from gestorAplicacion.personas.empleado import Empleado
5 #from gestorAplicacion.personas.cliente import Cliente
6 #from gestorAplicacion.inventario.vehiculo import Vehiculo
7 #from gestorAplicacion.financiero.cuentaBancaria import CuentaBancaria
8 from gestorAplicacion.establecimientos.cementerio import Cementerio
9 from gestorAplicacion.financiero.factura import Factura
10 from gestorAplicacion.inventario.tipoVehiculo import TipoVehiculo
11 from gestorAplicacion.inventario.tumba import Tumba
12 from gestorAplicacion.personas.familiar import Familiar
13 class Funeraria(Establecimiento):
14     _cuentaAhorros = None
15

```

gestorAplicacion.establecimientos.funeraria; clase Funeraria línea 14

Implementamos un atributo de clase de tipo CuentaBancaria en la clase Funeraria, de modo que todas las instancias de Funeraria compartan la misma cuenta de ahorros. De esta manera, los pagos de los clientes se depositarán en esta cuenta compartida y luego se repartirán entre las cuentas corrientes de cada Funeraria según los gastos que tengan en distintos ámbitos

*Constante:

```
gestorAplicacion > financiero > factura.py

2  from multimethod import multimethod
3  from datetime import datetime
4  from typing import List
5  from collections import defaultdict
6
7  class Factura():
8      # Atributos de clase
9      _IVA = 0.19
```

gestorAplicacion.financiero.factura; clase Factura línea 9

Definimos un atributo final, IVA, , que se emplea en el método de cálculo de totales para calcular el impuesto y determinar el valor total final de las facturas

*Encapsulamiento:

```
gestorAplicacion > establecimientos > establecimiento.py

14
15     def __init__(self, nombre=None, capacidad=0, cuentaCo
16         self._nombre = nombre
17         self._capacidad = capacidad
18         self._cuentaCorriente = cuentaCorriente
19         self._afiliacion = afiliacion
20         self._jefe = empleado
21         self._funeraria = funeraria
22         self._calificacion = calificacion
23         self.clientes = []
24         self._empleados = []
25         self._vehiculos = []
```

gestorAplicacion.establecimientos.establecimiento; clase Establecimiento líneas 16-25

*Sobrecarga de constructores:

```

gestorAplicacion > financiero > factura.py
9      _IVA = 0.19
10     _facturasCreadas = 0
11     facturas = []
12
13     #@multimethod
14     def __init__(self, producto=None, precio=0, fecha=None, cliente=None, entidad=None,
15                 # Atributos de instancia
16                 self._ID = Factura._facturasCreadas + 1
17                 Factura._facturasCreadas = self._ID
18                 self._producto = producto
19                 self._precio = precio
20                 self._total = 0
21                 self._cliente = cliente
22                 self._entidad = entidad
23                 self._fecha = fecha
24                 self._precioFinal = 0
25                 self._listaProductos = lista_productos if lista_productos else []
26                 self._porcentajeCreditoPorPagar = 1.0
27                 self._servicio = servicio
28                 self._nombre = f"Factura con ID: {self.getID()}"

```

gestorAplicacion.financiero.factura; clase Factura lineas 14-28

```

gestorAplicacion > personas > cliente.py
7
8  class Cliente(Persona):
9
10     #@multimethod
11     def __init__(self, nombre, cc, edad, cuentaBancaria=None, afiliacion=None, familia
12                 super().__init__(nombre, cc, edad, cuentaBancaria)
13                 self._afiliacion = afiliacion
14                 self._inventario = None
15                 self._familiares = familiares
16                 self._listadoFacturas = []
17

```

gestorAplicacion.personas.cliente; clase Cliente lineas 11-16

*Referencia self:

```

gestorAplicacion > establecimientos > establecimiento.py
29     self._horaEvento = None
30     self._iglesia = None
31     self._listadoProveedoresVehiculos = []
32     self._listadoProveedoresEmpleados = []
33     self._productos = []
34     self._listadoProveedores = []
35
36     Establecimiento._establecimientos.append(self)
37

```

gestorAplicacion.establecimientos.establecimiento; clase Establecimiento linea 36

*Enum:

```

gestorAplicacion > establecimientos > iglesia.py
1  from enum import Enum
2  from typing import List, Optional
3  from datetime import datetime, time, timedelta
4
5  class Iglesia(Enum):
6
7      HINDUISMO = {
8          "nombre": "Kamakhya Temple",
9          "sillas": 7,
10         "cremacion": True,
11         "duracionEvento": 4,
12         "tipoUrna": ["ordinaria"],
13         "religioso": "Pundit",
14         "religiosoAltoRango": "Guru"
15     }
16
17     BUDISMO = {
18         "nombre": "Templo de Nanjing",
19         "sillas": 8,
20         "cremacion": True,
21         "duracionEvento": 3,
22         "tipoUrna": ["fija", "ordinaria"],
23         "religioso": "Monje budista",
24         "religiosoAltoRango": "Abad"

```

gestorAplicacion.establecimientos.iglesia; enum Iglesia lineas 5-24

```

gestorAplicacion > financiero > banco.py
1  from enum import Enum
2
3  class Banco(Enum):
4
5      BANCOLOMBIA = ("BANCOLOMBIA", 0.0005, 1800)
6      DAVIVIENDA = ("DAVIVIENDA", 0.08, 2000)
7      BBVA = ("BBVA", 0.005, 1500)
8      BANCO_BOGOTA = ("BANCO_BOGOTA", 0.0001, 1600)
9      BANCO_OCCIDENTE = ("BANCO_OCCIDENTE", 0.02, 2200)
10
11  def __init__(self, NOMBRE, INTERES, COBRO_ADICIONAL):
12      self.NOMBRE = NOMBRE
13      self.INTERES = INTERES
14      self.COBRO_ADICIONAL = COBRO_ADICIONAL
15

```

gestorAplicacion.financiero.banco; enum Banco lineas 3-14

4. Descripción de cada una de las 5 funcionalidades implementadas.

■ **Descripción de la funcionalidad:** Proporcione una descripción detallada y completa del comportamiento de cada funcionalidad. Asegúrese de explicar claramente cada una de las interacciones que componen la funcionalidad, describiendo cómo los diferentes componentes del sistema colaboran para lograr el objetivo de la funcionalidad.

■ **Diagrama de interacción:** Incluya una imagen del diagrama de interacción correspondiente a cada funcionalidad. Asegúrese de que el diagrama sea claro y permita observar el comportamiento general de la funcionalidad, destacando las interacciones clave y los flujos de datos entre los componentes del sistema.

■ **Captura de pantalla:** Agregue capturas de pantalla donde se evidencia el correcto funcionamiento de cada funcionalidad.

FUNCIONALIDAD CREMATORIO

Diagrama:

https://drive.google.com/file/d/14s0JRMF-XtJhNXbUUhk0XHYSHYHEslz1p/view?usp=drive_link

•Función principal:

La función funcionalidadCrematorio se encarga de inicializar el proceso de cremación mostrando un título y opciones de selección como funerarias y tipos de clientes (adulto o menor de edad). El usuario selecciona la funeraria y el tipo de cliente desde listas desplegables.

•Selección del cliente:

Después de elegir la funeraria y el tipo de cliente, se procede a la función seleccionCliente, donde se muestran los clientes disponibles en función de la selección anterior.

Dependiendo de si el cliente es adulto o menor, se filtran los resultados y se muestra una lista específica de clientes correspondientes.

Una vez seleccionado el cliente, se confirma con el usuario si desea continuar con el proceso. Si el usuario acepta, se avanza al siguiente paso, que es la organización del crematorio.

•Organización del crematorio:

En esta parte del proceso, se muestran los crematorios disponibles para el cliente según su afiliación. Además, se le da al usuario la opción de seleccionar la iglesia en la que se desea realizar la ceremonia de cremación.

Las iglesias que aparecen en la lista permiten la cremación como un acto final, y se presentan con su nombre y religión.

El usuario debe ingresar el ID de la iglesia seleccionada.

•Selección de horarios en el crematorio:

Una vez elegida la iglesia y el crematorio, el sistema abre una nueva ventana para que el usuario seleccione una hora disponible en el crematorio.

El crematorio genera una lista de horarios en formato de 12 horas (AM/PM) para que el usuario elija.

Una vez seleccionado el horario, se finaliza el proceso organizando la cremación en el crematorio y vinculando la iglesia seleccionada.

•**Validación y control de errores:**

Si el usuario comete un error, como ingresar un ID incorrecto, el sistema muestra mensajes de error para guiarlo a corregir el dato ingresado.

Se incluye también una validación de que todas las opciones han sido seleccionadas correctamente antes de avanzar al siguiente paso.

•**Inicio y Presentación:** La función comienza mostrando un título y un texto para que el usuario seleccione datos relevantes. Utiliza frame para la interfaz gráfica.

•**Selección de Empleados y Cementerios:**

Los empleados disponibles se determinan basados en la hora del evento y el tipo de trabajo (en este caso, "cremador").

Se buscan cementerios que acepten cenizas y que estén afiliados al cliente. Los horarios del cementerio se adaptan según la hora del evento de cremación.

•**Interacción del Usuario:** El usuario selecciona entre los empleados y cementerios disponibles a través de una interfaz interactiva, utilizando un objeto frame1 para visualizar las opciones. Si decide continuar, se genera una nueva ventana donde puede elegir la hora del evento en el cementerio.

•**Confirmación de Cementerio y Horarios:** El sistema muestra los horarios disponibles del cementerio, preguntando al usuario si desea continuar. Si lo hace, se permite elegir un horario específico.

•**Selección de Urnas:** Después de confirmar el cementerio y el horario, el usuario debe elegir una urna para el cliente, basándose en la categoría y peso del cliente. Si no hay urnas disponibles, se añade una urna "default".

•**Verificación de la Urna:** El usuario selecciona la urna mediante un ID, validando su elección. Si el ID es correcto, se confirma la selección y se procede a la ceremonia.

•**Finalización del Proceso:** Se organiza la invitación para la ceremonia, añadiendo al cliente al sistema del crematorio y mostrando la invitación en pantalla.

FUNCIONALIDAD ENTIERRO

Diagrama:

https://drive.google.com/file/d/1MUwIUfPS7STHuQwVA34bnGF7yaZJheRk/view?usp=drive_link

1. Función titulo(frame, titulo)

Descripción: Limpia el frame y coloca un título en el centro.

Parámetros:

frame: El contenedor donde se colocará el título.

titulo: El texto que se mostrará como título.

Funcionamiento:

Se destruyen todos los elementos previos en el frame.

Se crea una etiqueta (Label) con el título y se coloca centrada en el frame utilizando el método pack.

2. Función funcionalidadEntierro(frame)

Descripción: Define el flujo inicial del servicio de entierro.

Flujo:

Se llama a la función titulo() para poner un encabezado: "Servicio de Entierro".

Se obtienen las funerarias disponibles (Establecimiento.filtrarEstablecimiento("funeraria")), lista de tipos de clientes (Mayor de edad, Menor de edad), y las iglesias.

Se llama a la función frame1() que crea un formulario con opciones como funerarias, tipo de cliente y religión.

La función interna organizacion() recoge los datos del formulario y llama a seleccionCliente() con los datos seleccionados (funeraria, tipo de cliente, iglesia).

3. Función seleccionCliente(frame, funeraria, indiceCliente, iglesia)

Descripción: Administra los datos específicos del cliente para el entierro.

Flujo:

Se coloca un nuevo título: "Datos Cliente".

Dependiendo de si el cliente es adulto o menor, el sistema busca los clientes en la funeraria.

Si el cliente es menor de edad, se busca a través del familiar responsable y se muestra la información en tablas.

Se pide la estatura del cliente a través de FieldFrame().

Se valida la estatura y, si es correcta, se bloquean las opciones seleccionadas y se buscan cementerios con tumbas disponibles.

La función interna confirmacion() confirma si hay inventario de tumbas disponible y procede si el usuario acepta continuar.

4. Función organizacionCementerio(frame, cliente, cementerio, estatura)

Descripción: Organiza la sección del cementerio y asigna una tumba al cliente.

Flujo:

Se muestra un mensaje de invitación para el evento del entierro.

Se obtiene la disponibilidad del inventario de tumbas según la estatura del cliente y se muestra en una tabla.

Se selecciona una tumba y se asigna al cliente.

Finalmente, se llama a la función factura() para mostrar los detalles finales.

5. Función factura(frame, cliente)

Descripción: Genera una factura final del servicio de entierro.

Flujo:

Se muestra un resumen de los datos del entierro.

Se coloca un título de factura y se detallan los pagos relacionados (ej., pago de flores).

Se incluye un botón para regresar a la pantalla principal.

FUNCIONALIDAD EXHUMACION.

Diagrama:

https://drive.google.com/file/d/14s0JRMF-XtJhNXbUHK0XHvSHYHEslz1p/view?usp=drive_link

Título y Descripción:

Primero, se establece un título visible en la interfaz que indica que se está manejando el servicio de exhumación.

Luego, se muestra una breve descripción para los usuarios, explicando qué es la exhumación y qué pasos seguirán en el proceso.

Selección Inicial:

Se solicita al usuario seleccionar una funeraria, el tipo de cliente (mayor o menor de edad), y el lugar donde se buscará al cliente (cementerio, urna, o tumba predeterminada). Esto se realiza a través de un formulario interactivo que recopila los datos necesarios.

Continuación del Proceso:

Cuando el usuario hace clic en el botón "Continuar", el sistema toma las selecciones ingresadas y las procesa. Las opciones seleccionadas quedan bloqueadas para evitar modificaciones posteriores.

El flujo continúa hacia la selección del cliente correspondiente, de acuerdo con la funeraria y el lugar de búsqueda escogidos.

Interactividad:

Una vez que se completan los datos iniciales, se actualiza el contenido de la interfaz para permitir que el usuario proceda con los siguientes pasos de la exhumación, como identificar al cliente en el cementerio o lugar especificado.

Esta función, seleccionCliente, gestiona la selección de un cliente para el servicio de

exhumación en una interfaz gráfica. Aquí está la explicación de su funcionamiento:

Separador y Variables Iniciales:

Se crea un separador visual para mejorar la disposición de los elementos en la interfaz.

Se definen varias variables para manejar los datos de clientes, tipo de búsqueda, y mensajes específicos.

Condiciones de Búsqueda:

Dependiendo del tipo de búsqueda (Cementerios de cuerpos, Urna default, o Tumba default), se realiza una búsqueda de clientes en diferentes contextos.

Cementerios de cuerpos: Si el tipo de cliente es "Mayor de edad" o "Menor de edad", se buscan los clientes en el cementerio correspondiente.

Urna default / Tumba default: Se buscan cementerios específicos que contengan urnas o tumbas disponibles, filtrando los cementerios según su tipo (cenizas o cuerpos).

Interfaz de Selección:

Se presentan al usuario dos formularios:

Clientes en Cementerio: Muestra los clientes que están en el cementerio seleccionado.

Cementerios Disponibles: Muestra los cementerios disponibles para urnas o tumbas, junto con opciones para el traslado.

Proceso de Selección y Confirmación:

Para Cementerios de Cuerpos:

Se permite seleccionar un cliente desde un cementerio. Después de seleccionar, se muestra un mensaje de confirmación para asegurar que el usuario desea continuar con ese cliente.

Para Urna o Tumba:

Se muestra una lista de cementerios disponibles y sus inventarios (urnas o tumbas).

Se permite seleccionar una urna o tumba específica. Luego, se muestra un mensaje de confirmación sobre la selección del cliente asociado a esa urna o tumba.

Acciones Posteriores:

Si el usuario confirma la selección, el sistema procede al siguiente paso del proceso de exhumación.

Si el usuario no confirma, se deshacen las selecciones realizadas y se permiten nuevas selecciones.

Botón de Continuar:

Se coloca un botón "Continuar" que ejecuta la función correspondiente dependiendo de la selección del usuario y los datos disponibles, gestionando así la transición a los siguientes pasos del proceso.

La función siguiente se encarga de organizar el traslado de un cliente a un nuevo cementerio, dependiendo del tipo de traslado (cenizas o cuerpos). A continuación, se detalla su funcionamiento:

Título y Preparativos:

Se muestra un título en la interfaz gráfica para indicar que se está organizando el traslado.

Se inicializan listas para almacenar información sobre las iglesias disponibles, así como para almacenar el nombre y la religión de cada iglesia.

Configuración según Tipo de Traslado:

Traslado de Cenizas:

Se solicita el peso del cliente y se define el tipo de objeto relacionado con el traslado como "urna".

Se filtran las iglesias que permiten la cremación.

Traslado de Cuerpos:

Se solicita la estatura del cliente y se define el tipo de objeto como "tumba".

Se obtienen todas las iglesias, ya que no hay restricciones basadas en la cremación.

Interfaz de Usuario:

Se presentan campos para ingresar el peso o la estatura del cliente.

Se muestran las iglesias disponibles para seleccionar la que realizará la ceremonia, con su nombre y religión.

Organización del Traslado:

Validación de Datos:

Se verifica que el peso o estatura del cliente esté dentro de los rangos válidos.

Se verifica que el ID de la iglesia seleccionado sea válido.

Proceso de Organización:

Si los datos son válidos, se procede a organizar el traslado. Se obtiene la información del cementerio actual del cliente y se actualiza con la iglesia seleccionada.

Se busca en los cementerios disponibles de la funeraria del cliente para encontrar uno que pueda acomodar al cliente según el tipo de objeto (urna o tumba).

Gestión de Inventario:

Se actualiza el inventario de los cementerios según sea necesario:

Si no hay inventario disponible, se añade un inventario por defecto.

Se muestra al usuario una lista de cementerios disponibles con la cantidad de inventario disponible para la selección final.

Botón de Continuar:

Se coloca un botón "Continuar" que, al ser presionado, ejecuta el proceso de organización del traslado y muestra la información correspondiente al usuario.

organizarDatos

Selección del Cementerio:

Validación del ID:

Se obtiene el ID del cementerio seleccionado por el usuario y se verifica que sea válido.

Si el ID es válido, se selecciona el cementerio correspondiente.

Eliminación del Cliente:

Se elimina el cliente del cementerio actual para evitar duplicados.

Verificación del Inventario:

Se verifica el inventario disponible en el nuevo cementerio para el tipo de objeto (urna o tumba).

Si hay solo una opción disponible, se muestra un mensaje y se organiza la iglesia.

Si hay más de una opción disponible, se le pregunta al usuario si desea trasladar al cliente a la opción más adecuada. Si acepta, se muestra una ventana con información detallada sobre el inventario recomendado. Si no acepta, se permite al usuario seleccionar entre las opciones disponibles.

Ventana de Selección de Inventario:

Opciones Recomendadas:

Si el usuario acepta la opción recomendada, se muestra una ventana con la información del inventario recomendado y se le da la opción de continuar.

Si el usuario no acepta, se muestran las opciones de inventario disponibles para que el usuario seleccione la deseada.

Botón de Continuar:

Se crea un botón "Continuar" que, al ser presionado, ejecuta la función organizarDatos para continuar con el proceso de selección del cementerio y organización del traslado.

organizarIglesia

Asignación del Cliente al Inventario:

Se agrega el cliente al inventario escogido del cementerio.

Si se está utilizando una ventana adicional, se cierra una vez que se haya agregado el cliente al inventario.

Organización en la Iglesia:

Se muestra un título en la interfaz gráfica para indicar que se está organizando la ceremonia en la iglesia.

Se organiza la ceremonia en la iglesia correspondiente al nuevo cementerio.

Se crea un marco para mostrar la invitación con la información de la iglesia.

Regresar al Menú Principal:

Se crea un botón "Regresar" que permite volver al menú principal de la aplicación.

FUNCIONALIDAD FINANZAS

Título y Descripción:

Diagrama: <https://drive.google.com/file/d/1toeVVJYvdZleUsXnzKpl0Cyo9KvuFxiN/view?usp=sharing>

Primero, se establece un título visible en la interfaz que indica que se está manejando el servicio de finanzas.

Luego, se muestra una breve descripción para los usuarios, explicando qué es finanzas y qué pasos seguirán en el proceso.

Selección Inicial:

Se solicita al usuario seleccionar una funeraria, y el tipo de servicio que desea realizar. Esto se realiza a través de un formulario interactivo que recopila los datos necesarios.

Continuación del Proceso:

Cuando el usuario hace clic en el botón "Continuar", el sistema genera un nuevo frame, según el tipo de servicio escogido

Para cobro clientes:

El flujo continúa hacia la selección del cementerio correspondiente, de acuerdo con la funeraria.

Se procesa el cementerio y se buscan los clientes pertenecientes a ella y que contengan alguna factura.

Se solicita al usuario seleccionar un cliente por medio de una combo box

Cuando el usuario hace clic en el botón "Continuar", el sistema busca las facturas y las muestra en la interfaz y se pide dar click en botón continuar

Se genera un informe de lo ocurrido en el proceso

Para pago facturas:

El flujo continúa hacia la selección de una factura correspondiente, de acuerdo con la funeraria.

Se procesa la factura y su pago, y se genera un informe de lo ocurrido en el proceso.

Para Pago de empleados:

El flujo continúa hacia la selección de un empleado correspondiente, de acuerdo con la funeraria.

Se procesa el empleado y su pago, y se genera un informe de lo ocurrido en el proceso.

Para crédito:

El flujo continúa hacia la selección de un servicio de crédito, de acuerdo con la funeraria.

Se solicita al usuario seleccionar un servicio por medio de una combo box

Cuando el usuario hace clic en el botón "Continuar", se procesa el servicio

Para pedir crédito:

Se procesa el pedido de un crédito teniendo en cuenta varios factores, como lo son, si ya hay uno activo o si no se ha pagado por lo menos el 50%

Se genera un informe y se muestra por pantalla el resultado del proceso

Para pagar crédito:

Se procesan los créditos activos y se muestran por medio de una combo box,
se pide seleccionar uno y se pide dar click a continuar

Se procesa el crédito seleccionado y se pide ingresar un porcentaje del crédito a pagar,
se procesa el crédito y su porcentaje y se inicia el pago, se genera un informe

Para ver crédito:

Se procesan los créditos activos y se muestran por medio de una combo box,
se pide seleccionar uno y se pide dar click a continuar

Se procesa el crédito seleccionado y se genera un informe detallado del crédito

Para reajuste de dinero:

El flujo continúa hacia la selección de un servicio de reajuste de dinero, de acuerdo con la funeraria.

Se solicita al usuario seleccionar un servicio por medio de una combo box

Cuando el usuario hace clic en el botón "Continuar", se procesa el servicio

Para ver informe gastos:

Se procesa la solicitud y se analizan los gastos y todas las facturas pagadas de la funeraria

se genera un informe de los gastos

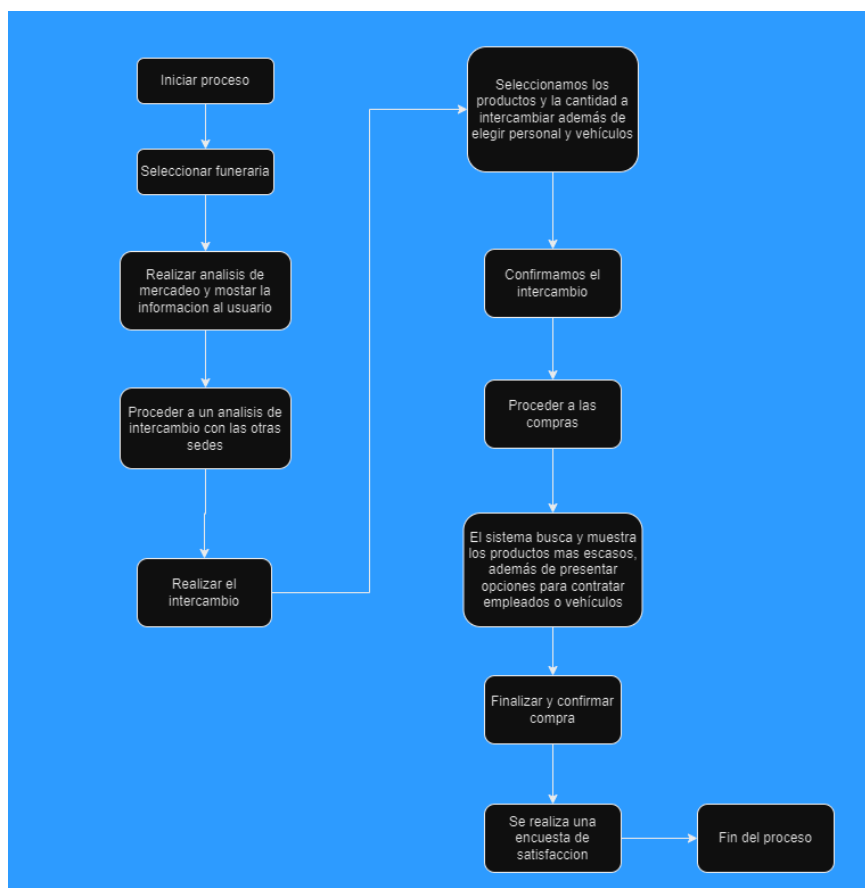
Para reajuste de dinero:

Se procesa la solicitud y se analizan los gastos y todas las facturas pagadas de las funerarias

se genera un informe de los gastos y un análisis de cuales tuvieron más gastos,

se hace un reajuste en las funerarias

FUNCIONALIDAD GESTION INVENTARIO



Variable Global.

Se define una variable global `selected_funeraria` que inicialmente no tiene un valor asignado. Se usa para almacenar la funeraria seleccionada por el usuario.

Función principal: funcionalidadGestionInventario(frame)

Esta función es la puerta de entrada al flujo de gestión de inventario. Llama a la función `seleccionar_funeraria(frame)` para que el usuario pueda elegir una funeraria.

Función seleccionar_funeraria(frame)

- **Interfaz para seleccionar funeraria:** Se crea un frame donde el usuario puede seleccionar una funeraria de una lista generada dinámicamente por `Establecimiento.filtrarEstablecimiento("funeraria")`.
- **Menú desplegable (OptionMenu):** Permite al usuario elegir una funeraria de una lista.
- **Continuar:** Al presionar el botón "Continuar", la funeraria seleccionada se busca en la lista mediante `obtener_funeraria_por_nombre`, y luego se pasa a la función `analisisMercadeo` si se selecciona correctamente.
- **Función analisisMercadeo(funeraria, master_frame)**
- **Productos vendidos:** Llama al método `calcularProductosVendidos` de la funeraria seleccionada para obtener la lista de productos vendidos.
- **Validación:** Filtra los productos para asegurar que solo aquellos con una cantidad vendida válida sean analizados.
- **Visualización:** Muestra en pantalla los productos vendidos, ordenados por cantidad, con dos columnas: "Producto" y "Cantidad Vendida".
- **Botón para continuar:** Una vez que se analiza el inventario, el usuario puede continuar al análisis de intercambio presionando el botón "Continuar", lo que llama a la función `continuar_analisis`.
- **Función analisis_intercambio(master_frame, funerarias, funeraria_origen)**
- **Interfaz para intercambio:** Muestra los productos disponibles en otras funerarias para un posible intercambio. Se omiten los productos de la funeraria de origen.
- **Diccionario product_widgets:** Se almacena cada producto mostrado en un diccionario para futuras interacciones o selección.
- **Botón para intercambio:** Al presionar el botón "Continuar", se selecciona la funeraria de destino (la primera distinta de la funeraria de origen) y se llama a `realizar_intercambio`.
 - **Creación del Frame de Intercambio:** Se crea un nuevo contenedor visual llamado `frame_intercambio`, donde se organizarán todos los elementos necesarios para llevar a cabo el intercambio. Este frame ocupa todo el espacio disponible en la ventana.
 - **Selección de Productos:** Se extraen los productos de las dos funerarias

involucradas. Se presenta una lista de productos disponibles en la funeraria de origen, mostrando el nombre del producto y su stock. El usuario selecciona un producto de esta lista y luego ingresa la cantidad a transferir.

- **Confirmación del Producto Seleccionado:** El botón "Confirmar Producto" permite validar la selección del producto y la cantidad ingresada. Si el stock es suficiente, se actualiza la cantidad en la funeraria de origen y se transfiere el producto a la funeraria de destino, incrementando su stock o agregando el producto si no existía previamente. Se muestra un mensaje de éxito o error según el resultado.
- **Selección de Empleados:** El usuario puede seleccionar empleados de la funeraria de origen para participar en el intercambio. Se presenta una lista con los nombres de los empleados disponibles. Tras confirmar la selección, se muestra un mensaje de éxito.
- **Selección de Vehículos:** Similar a la selección de empleados, el usuario selecciona un vehículo de la funeraria de origen. La lista muestra las placas y el tipo de vehículo. Tras la selección, se muestra un mensaje de éxito o error según corresponda.
- **Finalización del Intercambio:** Una vez seleccionados los productos, empleados y vehículos, el usuario puede finalizar el intercambio mediante un botón. Esto limpia el frame de intercambio y muestra un mensaje indicando que el proceso ha finalizado exitosamente. Además, tras finalizar, se llama a una función para comprar productos.
- **Cancelación del Intercambio:** El usuario puede cancelar el intercambio en cualquier momento, lo que cierra el `frame_intercambio` y descarta cualquier selección realizada hasta ese momento.
- **Creación de la interfaz:** Se genera un nuevo panel dentro de la ventana principal, donde se desplegará toda la información y opciones necesarias para el proceso de compra.
- **Identificación de productos faltantes:** La función detecta y muestra aquellos productos que tienen menos de 10 existencias en la funeraria. Si no se encuentra ningún producto con bajo stock, se muestra un mensaje y la función termina.
- **Selección de productos:** Se presenta una lista desplegable con los productos que tienen menos de 10 existencias, permitiendo al usuario elegir cuál desea comprar.
- **Selección de proveedores:** Se muestra otra lista desplegable con los proveedores disponibles asociados a la funeraria, para que el usuario seleccione quién suministrará el producto.
- **Cantidad de compra:** Se solicita al usuario que ingrese la cantidad de productos que desea comprar. Este valor es validado para asegurarse de que sea un número entero válido.
- **Confirmación de compra:** Cuando se selecciona un producto, proveedor y cantidad, y la cantidad es válida, se actualiza el stock del producto seleccionado en la funeraria, sumando la cantidad comprada al inventario. Si algo no está bien seleccionado o la cantidad no es válida, se muestra un mensaje de error.

- **Finalización de la compra:** Al finalizar, se muestra un mensaje de éxito, indicando que el proceso ha sido completado. Además, el panel se limpia y se procede a evaluar el proceso con una función de calificación.
- **Opciones adicionales:** Se ofrecen botones para comprar vehículos o contratar empleados, redirigiendo a las respectivas funciones dentro de la misma interfaz.
- **Creación de la interfaz:** Se genera un nuevo panel dentro de la ventana principal, donde se llevarán a cabo todas las acciones relacionadas con la contratación de empleados.
- **Selección de proveedores de empleados:** Se obtiene una lista de proveedores que ofrecen empleados para contratar en la funeraria. Si no hay proveedores disponibles, se muestra un mensaje y la función se termina.
- **Visualización de empleados:** Después de seleccionar un proveedor de la lista desplegable, se actualiza la interfaz para mostrar los empleados que ese proveedor tiene disponibles para contratar. Si el proveedor no tiene empleados disponibles, se muestra un mensaje al usuario.
- **Confirmación de contratación:** Una vez seleccionado un empleado, el usuario puede confirmar la contratación. El sistema agrega el empleado seleccionado a la lista de empleados de la funeraria y muestra un mensaje de éxito. Si no se ha seleccionado un empleado válido, se muestra un mensaje de error.
- **Finalización o cancelación:** El usuario tiene la opción de finalizar el proceso de contratación, lo que destruye el panel de contratación y muestra un mensaje de finalización. También se ofrece la opción de cancelar el proceso en cualquier momento, lo que simplemente cierra el panel sin realizar ninguna acción.
- **Interfaz para la compra:** Se crea un panel específico dentro de la ventana principal para manejar todo lo relacionado con la compra de vehículos.
- **Selección de proveedores de vehículos:** Se obtiene una lista de proveedores que ofrecen vehículos para la funeraria. Si no hay proveedores disponibles, se muestra un mensaje indicando que no se pueden realizar compras.
- **Visualización de vehículos:** Una vez seleccionado un proveedor, la interfaz se actualiza para mostrar los vehículos disponibles que ese proveedor ofrece. Si no hay vehículos disponibles, se informa al usuario mediante un mensaje.
- **Confirmación de la compra:** El usuario puede seleccionar un vehículo y confirmar la compra. El vehículo elegido se agrega a la lista de vehículos de la funeraria, y se muestra un mensaje de éxito al usuario. Si el usuario intenta realizar la compra sin seleccionar un vehículo válido, se genera un mensaje de error.
- **Finalización o cancelación:** El usuario tiene la opción de finalizar el proceso, lo que destruye el panel y muestra un mensaje indicando que la compra ha sido completada. También puede cancelar el proceso en cualquier momento, lo que cierra el panel sin realizar ninguna acción.
- **Interfaz de calificación:** Se crea un panel dentro de la ventana principal

específicamente para la calificación. Aquí, el usuario puede interactuar con los diferentes elementos que se presentan.

- **Campo para la nota de calificación:** Se le pide al usuario que ingrese una calificación numérica en una escala del 1 al 10, lo que permite medir la calidad del proceso.
- **Campo para la descripción:** Además de la calificación numérica, se le solicita al usuario que proporcione una descripción detallada del proceso. Este campo es de texto y permite que el usuario comparta comentarios más amplios sobre su experiencia.
- **Validación de la información:** Al finalizar, el sistema verifica que tanto la calificación numérica como la descripción hayan sido ingresadas correctamente. Si alguno de estos datos falta, se muestra un mensaje de error, pidiéndole al usuario que complete los campos necesarios.
- **Finalización del proceso:** Si todo está correcto, la información proporcionada (calificación y descripción) se guarda en la funeraria, y se muestra un mensaje de agradecimiento al usuario. Posteriormente, se cierra el panel de calificación, indicando que el proceso ha finalizado.

VENTANA INICIO

```
iUMain > ventanaInicio.py > ...
9 def irVentanaPrincipal():
11     ventanaPrincipal.ventanaPrincipal()
12
13 def salir():
14     ventana.quit()
15
16 def descripcion(frame):
17     for widget in frame.winfo_children():
18         widget.destroy()
19
20     tk.Label(frame, text="Funeraria Rosario te brinda un servicio de calidad en tus ceremonias fúnebres").pack(pady=4)
21
22
23 def ventanaInicio():
24
25     #Objeto tipo ventana
26     global ventana, indiceValor, button100, imagenes, hojasVida, label00, label01, label10, label11, indiceImagenes, imagenesProyecto, indiceImagenP4, labelP4
27     ventana = tk.Tk()
28     ventana.geometry("600x400")
29
30
31     #Generacion de los contenedores principales
32     #Frame principal izquierda (P1)
33     frameIzquierda=tk.Frame(ventana, bg="#9fd5d1",bd=5,relief="ridge")
34     frameIzquierda.pack(side="left",expand = True, fill ="both",padx=10,pady=10)
35     #Frame principal derecha (P2)
36     frameDerecha=tk.Frame(ventana,bg="#9fd5d1",bd=5,relief="ridge")
37     frameDerecha.pack(side="right",expand=True,fill="both",padx=5,pady=5)
38
39
40     #Frame secundario arriba izquierda (P3)
41     frameArribaIzquierda=tk.Frame(frameIzquierda,bg="white",bd=2, relief="groove")
42     frameArribaIzquierda.place(relx=0.05, rely=0.05, relwidth=0.9, relheight=0.4)
43
44     # Lista de mensajes de bienvenida
45     mensajes_bienvenida = [
46         "¡Bienvenido a la aplicación!",
47         "¡Hola! Esperamos que disfrutes de tu experiencia.",
48         "¡Saludos! Prepárate para explorar nuestra app.",
49         "¡Bienvenido de nuevo! Nos alegra verte.",
50     ]
51
```

```

51
52     # Selecciona un mensaje aleatorio
53     mensaje_aleatorio = random.choice(mensajes_bienvenida)
54
55     # Crear Label para mostrar el mensaje de bienvenida
56     label_bienvenida = tk.Label(
57         frameArribaIzquierda,
58         text=mensaje_aleatorio,
59         bg="#f0f0f0", # Fondo claro para contraste
60         fg="#333",    # Color de texto oscuro
61         font=("Helvetica", 14, "bold"), # Fuente más elegante
62         wraplength=250, # Limitar el largo del texto
63         justify="center",
64         padx=20, # Padding interno para más espacio
65         pady=20
66     )
67
68     label_bienvenida.pack(expand=True, fill = "both", padx=10, pady=10)
69
70     #Objeto de menú
71     menuPrincipal= tk.Menu(ventana)
72
73     #Asociar el objeto
74     ventana.config(menu = menuPrincipal)
75
76     frmDescripcion =tk.Frame(frameArribaIzquierda)
77     frmDescripcion.pack(pady=2)
78
79     #Crear menu opciones
80     opciones = tk.Menu(menuPrincipal, tearoff=0)
81     menuPrincipal.add_cascade(label="Opciones",menu=opciones)
82     opciones.add_command(label="Descripcion",command=lambda:descripcion(frmDescripcion)) #,command=mostrarDescripcion)
83     opciones.add_separator()
84     opciones.add_command(label="Salir",command=salir)
85

```

```

#Frame secundario abajo izquierda (P4)
imagenesProyecto = ["iuMain/imagenes/imagen1F.png","iuMain/imagenes/imagen2F.png","iuMain/imagenes/imagen3F.png","iuMain/imagenes/imagen4F.png","iuMain/imagenes/imagen5F.png"]
indiceImagenP4 = 0
imagen1 = Image.open(imagenesProyecto[0])
imagen1 = ImageTk.PhotoImage(imagen1)
frameAbajoIzquierda=tk.Frame(frameIzquierda,bg="white",bd=2, relief="groove")
frameAbajoIzquierda.place(relx=0.05,relx=0.47,relwidth=0.9,relheight=0.5)
frameArribaAbajoIzquierda=tk.Frame(frameAbajoIzquierda,bg="white")
frameArribaAbajoIzquierda.place(relx=0.04,relx=0.04,relwidth=0.9,relheight=0.8)
labelP4 = tk.Label(frameArribaAbajoIzquierda,image=imagen1, bg="white")
labelP4.pack(expand=True,padx=5,pady=5,fill="both")
labelP4.bind("<Leave>", cambiarImagenP4)
btnPrincipal=tk.Button(frameAbajoIzquierda, text="Iniciar Aplicación", command=irVentanaPrincipal)
btnPrincipal.pack(expand=True,anchor="s",padx=5,pady=5)

#Frame secundario arriba derecha (P5)
frameArribaDerecha=tk.Frame(frameDerecha,bg="white",bd=2, relief="groove")
frameArribaDerecha.place(relx=0.05,relx=0.05,relwidth=0.9,relheight=0.4)
button100 = tk.Button(frameArribaDerecha, bg= "white", text="Hojas de vida de los desarrolladores", font=("Arial",8), wraplength=200,justify="center",anchor="center",command=cambiarHojasDeVida)
button100.pack(expand= True, fill= "both", padx=5,pady=5)

#Frame secundario abajo derecha (P6)
frameAbajoDerecha=tk.Frame(frameDerecha,bg="white",bd=2, relief="groove")
frameAbajoDerecha.place(relx=0.05,relx=0.47,relwidth=0.9,relheight=0.5)

#Configuración del frame en cuadrícula
# Configurar el frame inferior derecho para que tenga 2 filas y 2 columnas de igual tamaño
frameAbajoDerecha.grid_rowconfigure(0, weight=1)
frameAbajoDerecha.grid_rowconfigure(1, weight=1)
frameAbajoDerecha.grid_columnconfigure(0, weight=1)
frameAbajoDerecha.grid_columnconfigure(1, weight=1)

# Partir el frame en dos filas y dos columnas
# Crear los sub-frames dentro del frame inferior derecho (cuadrícula 2x2)
frame00 = tk.Frame(frameAbajoDerecha, bg="purple", bd=2, relief="solid")
frame00.grid(row=0, column=0, padx=2, pady=2, sticky="nsew")
label00 = tk.Label(frame00)
label00.pack(fill="both", expand=True)

frame01 = tk.Frame(frameAbajoDerecha, bg="blue", bd=2, relief="solid")
frame01.grid(row=0, column=1, padx=2, pady=2, sticky="nsew")
label01 = tk.Label(frame01)
label01.pack(fill="both", expand=True)

```

```

frame01 = tk.Frame(frameAbajoDerecha, bg="blue", bd=2, relief="solid")
frame01.grid(row=0, column=1, padx=2, pady=2, sticky="nsew")
label01 = tk.Label(frame01)
label01.pack(fill="both", expand=True)

frame10 = tk.Frame(frameAbajoDerecha, bg="blue", bd=2, relief="solid")
frame10.grid(row=1, column=0, padx=2, pady=2, sticky="nsew")
label10 = tk.Label(frame10)
label10.pack(fill="both", expand=True)

frame11 = tk.Frame(frameAbajoDerecha, bg="purple", bd=2, relief="solid")
frame11.grid(row=1, column=1, padx=2, pady=2, sticky="nsew")
label11 = tk.Label(frame11)
label11.pack(fill="both", expand=True)

hojaVida1 = "Soy Violeta, una estudiante de Ingeniería de Sistemas de 19 años, apasionada por la tecnología y el desarrollo de software. Me interesa aprender y crecer en el campo c
hojaVida2 = "Sebastian soy un estudiante de ingeniería de sistemas apasionado por la tecnología, los gatos, el jazz y las historias de fantasía. Disfruta programar mientras escuch
hojaVida3 = "Soy Andrés Pérez, tengo 18 años y soy estudiante de Ingeniería de Sistemas en la Universidad Nacional de Colombia. Me gustan los videojuegos, los animales y mejorar mi
hojasVida = [hojaVida1, hojaVida2, hojaVida3]
imagenes = ["iuMain/imagenes/imagen1.png", "iuMain/imagenes/imagen2.png", "iuMain/imagenes/imagen3.png", "iuMain/imagenes/imagen4.png", "iuMain/imagenes/imagen5.png", "iuMain/imagenes/i
indiceValor = 0
indiceImagenes = 0
ventana.mainloop()

def cambiarHojaVidaImagenes():
    global indiceValor, button100, imagenes, hojasVida, label00, label01, label10, label11, indiceImagenes
    button100.config(text=hojasVida[indiceValor])

    ancho = label00.winfo_width()
    alto = label00.winfo_height()

    # Redimensionar las imágenes según el tamaño de los labels
    imagen00 = Image.open(imagenes[indiceImagenes]).resize((ancho, alto))
    imagen00 = ImageTk.PhotoImage(imagen00)
    label00.config(image=imagen00)
    label00.image = imagen00

```

Activar Windows

```

def cambiarHojaVidaImagenes():
    global indiceValor, button100, imagenes, hojasVida, label00, label01, label10, label11, indiceImagenes
    button100.config(text=hojasVida[indiceValor])

    ancho = label00.winfo_width()
    alto = label00.winfo_height()

    # Redimensionar las imágenes según el tamaño de los labels
    imagen00 = Image.open(imagenes[indiceImagenes]).resize((ancho, alto))
    imagen00 = ImageTk.PhotoImage(imagen00)
    label00.config(image=imagen00)
    label00.image = imagen00

    imagen01 = Image.open(imagenes[(indiceImagenes+1) % len(imagenes)]).resize((ancho, alto))
    imagen01 = ImageTk.PhotoImage(imagen01)
    label01.config(image=imagen01)
    label01.image = imagen01

    imagen10 = Image.open(imagenes[(indiceImagenes+2) % len(imagenes)]).resize((ancho, alto))
    imagen10 = ImageTk.PhotoImage(imagen10)
    label10.config(image=imagen10)
    label10.image = imagen10

    imagen11 = Image.open(imagenes[(indiceImagenes+3) % len(imagenes)]).resize((ancho, alto))
    imagen11 = ImageTk.PhotoImage(imagen11)
    label11.config(image=imagen11)
    label11.image = imagen11

    # Actualizar los índices
    indiceValor = (indiceValor+1) % len(hojasVida)
    indiceImagenes = (indiceImagenes+4) % len(imagenes)

def cambiarImagenP4(event):
    global indiceImagenP4, imagenesProyecto, labelP4
    ancho1 = labelP4.winfo_width()
    alto1 = labelP4.winfo_height()
    indiceImagenP4 = (indiceImagenP4+1) % len(imagenesProyecto)
    imagenP4 = Image.open(imagenesProyecto[indiceImagenP4]).resize((ancho1, alto1))
    imagenP4 = ImageTk.PhotoImage(imagenP4)
    labelP4.config(image=imagenP4)
    labelP4.image = imagenP4

```

Funciones de la Interfaz Gráfica

1. irVentanaPrincipal()

Esta función cierra la ventana actual (ventana) y abre la ventana principal de la aplicación (ventanaPrincipal). Esto es útil para cambiar entre diferentes ventanas de la aplicación.

2. salir()

Esta función termina la ejecución de la aplicación cerrando la ventana principal (ventana). Utiliza ventana.quit() para cerrar la aplicación.

3. descripcion(frame)

Esta función limpia todos los widgets dentro del frame dado y muestra un mensaje de descripción en una etiqueta (Label). La descripción proporcionada es un texto fijo que describe el servicio de la empresa "Funeraria Rosario".

4. ventanaInicio()

Esta función configura la ventana principal de la aplicación. Aquí están los detalles principales:

Configuración Inicial:

Crea una ventana principal (ventana) con un tamaño de 600x400 píxeles.

Define dos Frames principales: frameIzquierda y frameDerecha, ubicados a la izquierda y derecha de la ventana, respectivamente.

Frame Secundario Arriba Izquierda (P3):

Contiene un mensaje de bienvenida aleatorio que se selecciona de una lista predefinida.

El mensaje se muestra en un Label con un estilo de fuente más elegante y un fondo claro para contraste.

Menú Principal:

Se crea un menú principal en la ventana (menuPrincipal) con opciones para mostrar una descripción y salir de la aplicación.

Al seleccionar "Descripción", se llama a la función descripcion() para mostrar la información en un Frame secundario.

Frame Secundario Abajo Izquierda (P4):

Muestra una imagen y un botón. La imagen cambia cuando el mouse sale del área de la etiqueta (labelP4), gracias al evento asociado.

El botón "Iniciar Aplicación" llama a irVentanaPrincipal() para abrir la ventana principal.

Frame Secundario Arriba Derecha (P5):

Contiene un botón que muestra las hojas de vida de los desarrolladores. Al hacer clic en el botón, se actualizan las imágenes y la descripción.

Frame Secundario Abajo Derecha (P6):

Configurado en una cuadrícula 2x2, con cuatro sub-frames (frame00, frame01, frame10, frame11) que contienen etiquetas (label00, label01, label10, label11).

Configuración Adicional:

Se definen listas de imágenes y hojas de vida para actualizar los Labels en el Frame inferior derecho.

Se inicializan los índices para navegar a través de las imágenes y hojas de vida.

5. cambiarHojaVidaeImágenes()

Esta función actualiza el texto del botón y las imágenes en los Labels del Frame inferior derecho. Cada vez que se llama, se actualiza el texto del botón con una hoja de vida diferente y se actualizan las imágenes en los Labels con nuevas imágenes redimensionadas.

Actualización de Imágenes:

Las imágenes se redimensionan según el tamaño de los Labels y se actualizan en el Frame.

Actualización de Índices:

Los índices de las imágenes y hojas de vida se actualizan para mostrar las siguientes en la lista.

6. cambiarImagenP4(event)

Esta función cambia la imagen en labelP4 cada vez que el mouse sale del área de la etiqueta (labelP4). La imagen se actualiza con una nueva imagen de la lista imagenesProyecto, y se redimensiona según el tamaño del labelP4.

Actualización de Imagen:

La imagen se redimensiona para ajustarse al tamaño del labelP4.

Se actualiza el índice de la imagen para la próxima actualización.

VENTANA PRINCIPAL


```

9
10 #Regresar a la ventana de inicio
11 def irVentanaInicio():
12     ventana.destroy()
13     ventanaInicio.ventanaInicio()
14
15
16 def framePrincipal(frame):
17     for widget in frame.winfo_children():
18         widget.destroy()
19
20     etiquetaInicial = tk.Label(frame,
21                                text="Funeraria Rosario, su muerte mi salario",
22                                bg="#92abc3", # Color de fondo de la etiqueta
23                                fg="#333333", # Color del texto
24                                font=("Helvetica", 16, "bold"), # Tipo y tamaño de fuente
25                                padx=10, # Espacio interno horizontal
26                                pady=10, # Espacio interno vertical
27                                relief="raised", # Estilo del borde
28                                borderwidth=2) # Ancho del borde
29     etiquetaInicial.pack(pady=10)
30
31     tk.Label(frame, text="Lo invitamos a consultar nuestros servicios en el menú de Procesos y consultas",
32              bg="#8ad0b2", # Color de fondo de la etiqueta
33              fg="#333333", # Color del texto
34              font=("Helvetica", 16, "bold"), # Tipo y tamaño de fuente
35              padx=10, # Espacio interno horizontal
36              pady=10, # Espacio interno vertical
37              relief="raised", # Estilo del borde
38              borderwidth=2).pack(pady=6)
39
40
41
42
43
44
45 def ayudaFunc():
46     tk.messagebox.showinfo("Ayuda", "Nombres desarrolladores: \n-Violeta Gomez\n-Andres Perez\n-Sebastian Guerra")
47

```

```

45 def ayudaFunc():
46     tk.messagebox.showinfo("Ayuda", "Nombres desarrolladores: \n-Violeta Gomez\n-Andres Perez\n-Sebastian Guerra")
47
48
49 #Aplicacion
50 def aplicacion():
51     ventanaDescripcion = tk.Toplevel()
52     ventanaDescripcion.title("Funeraria Rosario")
53     ventanaDescripcion.geometry("500x500")
54
55     texto= """
56         Funeraria Rosario: se dedica a ofrecer un servicio completo y compasivo en momentos de necesidad.
57         Con años de experiencia en el sector, nuestra misión es proporcionar apoyo integral a las familias
58         durante los momentos más difíciles, asegurando que cada detalle sea manejado con la máxima
59         sensibilidad y respeto.
60
61         Nuestros Servicios Incluyen:
62         1. Servicio de Cremación
63         2. Servicio de Exhumacion
64         3. Servicio de Entierro
65
66         En Funeraria Rosario, entendemos que cada vida es única y que cada despedida debe reflejar ese valor.
67         Nos comprometemos a ofrecer un servicio personalizado que respete las tradiciones, creencias y
68         deseos de cada familia.
69
70         Servicios particulares de la funeraria
71         1. Finanzas
72         2. Gestion de Inventario
73     """
74
75     label = tk.Label(ventanaDescripcion, text=texto, padx=10, pady=10, anchor="w", width=70, height=20, wraplength=480)
76     label.pack(pady=15)
77
78     btnContinuar = tk.Button(ventanaDescripcion, text="Continuar",
79                             command=lambda: ventanaDescripcion.destroy())
80     btnContinuar.pack(pady=20)
81
82     ventanaDescripcion.mainloop()
83
84 def ventanaPrincipal():
85     global ventana
86     ventana = tk.Tk()
87     ventana.geometry("600x400")
88     ventana.title("Funeraria Rosario, su muerte mi salario")
89     #Frame 2 - Zona 1 - Menus
90     zona1=tk.Frame(ventana)
91     zona1.pack(side="top", fill="x", anchor="nw", padx=2, pady=2)

```

```

89 #Frame 2 - Zona 1 - Menus
90 zona1=tk.Frame(ventana)
91 zona1.pack(side="top",fill="x",anchor="nw",padx=2,pady=2)
92
93 # Implementacion de las funcionalidades (Zona 2)
94 zona2 = tk.Frame(ventana)
95 zona2.pack(fill=tk.BOTH, expand=True)
96 zona2.configure(bg="white")
97
98 #MenuPrincipal
99 menuPrincipal = tk.Menu(ventana)
100 ventana.config(menu=menuPrincipal)
101
102 #Menú archivo
103 archivo=tk.Menu(menuPrincipal,tearoff=0)
104
105 menuPrincipal.add_cascade(label="Archivo", menu=archivo)
106 #Armar menú archivo
107 archivo.add_command(label="Aplicación", command=aplicacion)
108 archivo.add_separator()
109 archivo.add_command(label="Salir", command=irVentanaInicio)
110
111 #Menú procesos
112 procesos=tk.Menu(menuPrincipal,tearoff=0)
113
114 menuPrincipal.add_cascade(label="Procesos y consultas",menu=procesos)
115 procesos.add_command(label="Cremación",command=lambda:cremacion.funcionalidadCrematorio(zona2))
116 procesos.add_separator()
117 procesos.add_command(label="Exhumacion",command=lambda:exhumacion.funcionalidadExhumacion(zona2))
118 procesos.add_separator()
119 procesos.add_command(label="Entierro",command=lambda:entierro.funcionalidadEntierro(zona2))
120 procesos.add_separator()
121 procesos.add_command(label="Finanzas",command=lambda:finanzas.funcionalidadFinanzas(zona2))
122 procesos.add_separator()
123 procesos.add_command(label="Gestion de inventario",command=lambda:inventario.funcionalidadGestionInventario(zona2))
124
125 #Menú ayuda
126 ayuda=tk.Menu(menuPrincipal,tearoff=0)
127
128 menuPrincipal.add_cascade(label="Ayuda",menu=ayuda)
129 ayuda.add_command(label="Informacion",command=ayudaFuncion)
130
131 framePrincipal(zona2)
132

```

irVentanaInicio:

Esta función cierra la ventana actual (ventana) y luego abre la ventana de inicio (ventanaInicio).

framePrincipal:

Esta función limpia el contenido de un frame dado (elimina todos los widgets hijos) y luego agrega dos etiquetas (tk.Label) al frame con textos informativos sobre la funeraria. La primera etiqueta muestra un eslogan de la funeraria, mientras que la segunda invita a consultar los servicios en el menú de procesos y consultas. Ambas etiquetas tienen un estilo visual específico, incluyendo colores de fondo, colores de texto, fuentes, y bordes elevados.

ayudaFuncion:

Muestra una ventana emergente con información sobre los desarrolladores de la aplicación, incluyendo sus nombres.

aplicacion:

Crea una nueva ventana (ventanaDescripcion) que presenta una descripción de los servicios ofrecidos por la funeraria. La ventana incluye un texto detallado sobre la misión y los servicios de la funeraria, y un botón para cerrar la ventana.

ventanaPrincipal:

Configura la ventana principal de la aplicación (ventana) y su interfaz. Dentro de esta ventana, se crean varias áreas:

Zona 1: Un frame en la parte superior de la ventana que probablemente contiene menús.

Zona 2: Un frame que ocupa el resto de la ventana, usado para mostrar contenido principal.

Se crea un menú principal con varias opciones:

Archivo: Incluye opciones para abrir la descripción de la aplicación y salir.

Procesos y Consultas: Contiene opciones para realizar varias acciones relacionadas con servicios como cremación, exhumación, entierro, finanzas y gestión de inventario. Cada opción está asociada a una función que actualiza el contenido de la Zona 2.

Ayuda: Ofrece una opción para mostrar información sobre los desarrolladores.

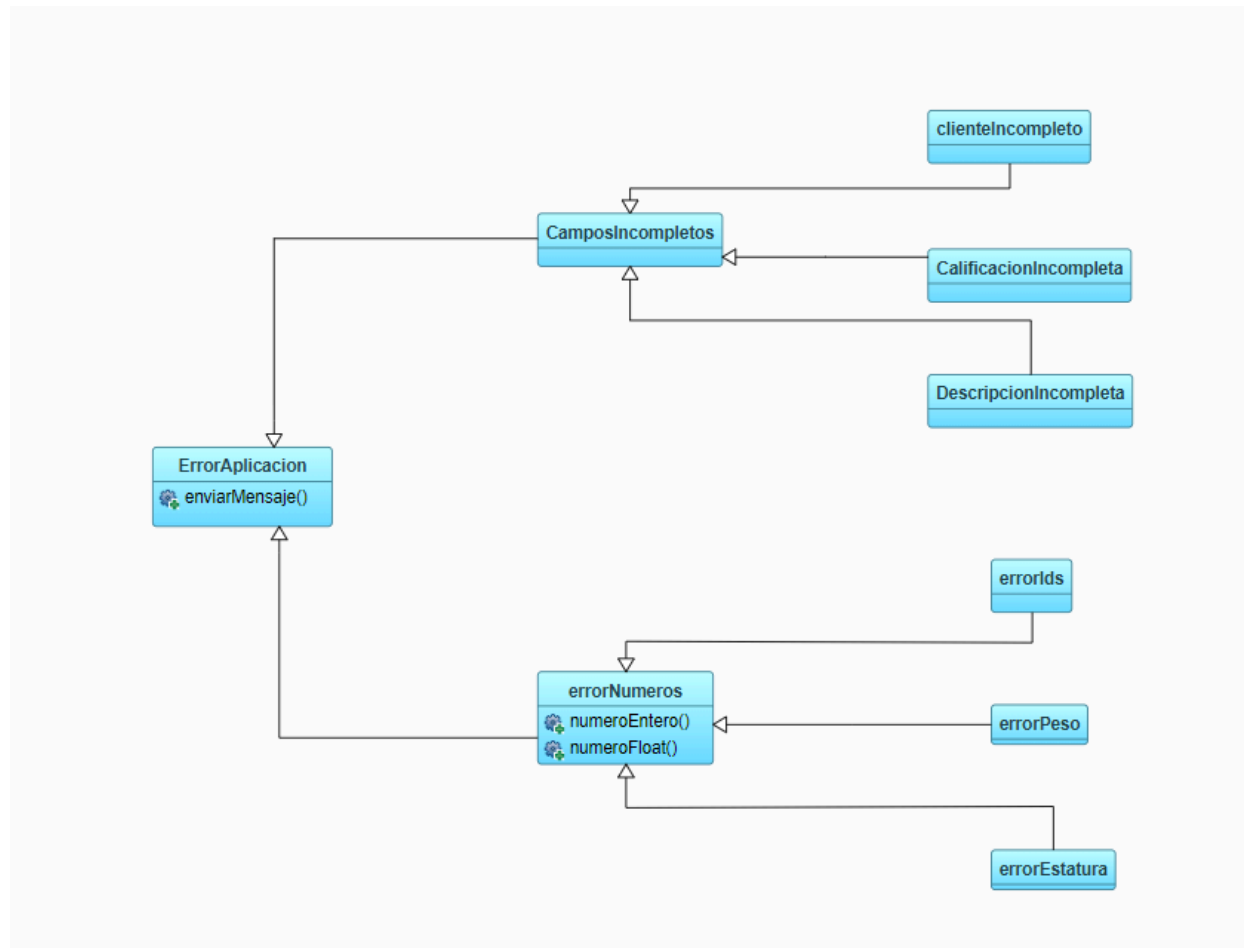
Al final, se llama a la función `framePrincipal` para inicializar la Zona 2 con contenido inicial.

5. Manejo de excepciones

Descripción: Proporcione una descripción detallada de cada una de las excepciones implementadas y las implicaciones que tienen en el funcionamiento de su aplicación.

Captura de pantalla:

- **Código:** Agregue capturas de pantalla que muestren la implementación en el código de cada una de las excepciones.
- **Ejecución de excepción:** Incluya capturas de pantalla que evidencien el disparo o la ejecución de cada excepción en la aplicación.



https://app.genmymodel.com/api/projects/_6clpwHTqEe-m5L2WiEAA5A/diagrams/_6clpw3TqEe-m5L2WiEAA5A/svg

iuMain > manejoErrores > errorAplicacion.py > ...

```
1  from tkinter import messagebox
2
3
4  class ErrorAplicacion(Exception):
5      def __init__(self, mensaje):
6          self._mensaje = f"Manejo de errores de la Aplicación\n{mensaje}"
7          super().__init__(self._mensaje)
8          self.enviarMensaje()
9
10     def enviarMensaje(self):
11         messagebox.showerror("Error", self._mensaje)
12
13 class CamposIncompletos(ErrorAplicacion):
14     def __init__(self, mensaje):
15         self._mensaje=mensaje
16         super().__init__(f"Campo incompleto {self._mensaje}")
17
18 class clienteIncompleto(CamposIncompletos):
19     def __init__(self):
20         super().__init__("No es posible continuar sin un Cliente")
21
22 class CalificacionIncompleta(CamposIncompletos):
23     def __init__(self):
24         super().__init__("No se puede continuar sin una calificación")
25
26
27 class DescripcionIncompleta(CamposIncompletos):
28     def __init__(self):
29         super().__init__("No se puede continuar sin una descripción")
30
31
32
33
34
35 class errorNumeros(ErrorAplicacion):
36     def __init__(self, mensaje):
37
38         self._mensaje = mensaje
39         self._verificar = True
40         super().__init__(f"Error números {mensaje}")
41
42     def numeroEntero(self,valor):
43         try:
44             int(valor)
45         except ValueError:
46             # Actualiza el mensaje de error y relanza la excepción
47             raise ErrorAplicacion("Debes ingresar un dígito")
48
49
```

```

49
50     def numeroFloat(self,valor):
51         try:
52             float(valor)
53         except ValueError:
54             # Actualiza el mensaje de error y relanza la excepción
55             raise ErrorAplicacion("Debes ingresar un dígito")
56
57     class errorIds(errorNumeros):
58         def __init__(self, valor, mensaje=None, valMin=0, valMax=0):
59             self.numeroEntero(valor)
60             # Inicializa la clase base
61             # Verifica si el valor es un número
62
63             # Verifica si el valor es mayor o igual al valor mínimo
64             if int(valor)<valMin or int(valor)>valMax:
65                 raise super().__init__("Id incorrecta")
66             else:
67                 super().__init__(mensaje)
68
69
70     class errorPeso(errorNumeros):
71         def __init__(self, valor, pesoMax, pesoMin=0):
72             self.numeroFloat(valor)
73
74             # Verifica si el valor es mayor o igual al valor mínimo
75             if float(valor)<pesoMin or float(valor)>pesoMax:
76                 raise super().__init__("El valor del peso del cliente no es correcto")
77
78
79
80     class errorEstatura(errorNumeros):
81         def __init__(self, valor, estaturaMax, estaturaMin=0):
82             self.numeroFloat(valor)
83
84             # Verifica si el valor es mayor o igual al valor mínimo
85             if float(valor)<estaturaMin or float(valor)>estaturaMax:
86                 raise super().__init__("El valor de la estatura del cliente no es correcta")
87
88
89

```

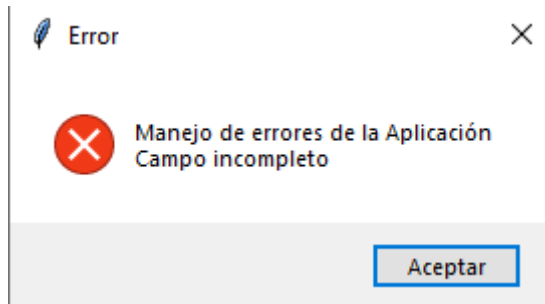
1. Clase ErrorAplicacion (clase base de excepciones):

- **Hereda de:** Exception, por lo tanto, es una excepción personalizada.
- **Constructor (__init__):** Recibe un mensaje como parámetro, que se asigna a self._mensaje. Este mensaje contiene información sobre el error que ocurrió.
- **Mostrar el mensaje:** Se llama al método enviarMensaje() que muestra un cuadro de diálogo emergente de error (messagebox.showerror) con el mensaje correspondiente.

Esta clase sirve como base para otras excepciones específicas en la aplicación.

2. Clase CamposIncompletos (hereda de ErrorAplicacion):

- **Hereda de:** ErrorAplicacion.
- **Propósito:** Representa una excepción específica que se lanza cuando falta información en un campo obligatorio.
- **Mensaje personalizado:** El mensaje de error indicará que un "Campo incompleto" es la razón del fallo.



3. Clases hijas de CamposIncompletos:

Estas clases representan casos específicos de campos faltantes.

- **clienteIncompleto:** Se lanza cuando falta un cliente. Muestra el mensaje "No es posible continuar sin un Cliente".

```

128     def establecerCliente():
129         if buscar=="Cementerios de cuerpos":
130             if frameCliente.continuar():
131                 frameCliente.bloquearOpciones()
132                 cliente=clientes[(frameCliente.getValores())[0]]
133                 texto=f"Has seleccionado al cliente {cliente} desde cementerio: {(cliente.getInventario()).getCementerio()}\n ¿Deseas continuar?"
134                 result = tk.messagebox.askyesno("Confirmar Datos",texto)
135
136                 traslado=None
137
138                 if (opciones[frameCliente.getValores()[1]]=="cementerio cuerpos":
139                     traslado="cuerpos"
140                 else:
141                     traslado="cenizas"
142
143                 if result:
144                     siguiente(frame,cliente,traslado)
145                 else:
146                     clienteIncompleto()
147                     frameCliente.desbloquearOpciones()
148

```

- **CalificacionIncompleta:** Se lanza cuando falta una calificación. Muestra el mensaje "No se puede continuar sin una calificación".

```

# Función para validar y finalizar la calificación
def finalizar_calificacion():
    calificacion = calificacion_var.get()
    descripcion = descripcion_text.get("1.0", tk.END).strip()

    try:
        # Validar que haya una calificación
        if not calificacion:
            raise CalificacionIncompleta()

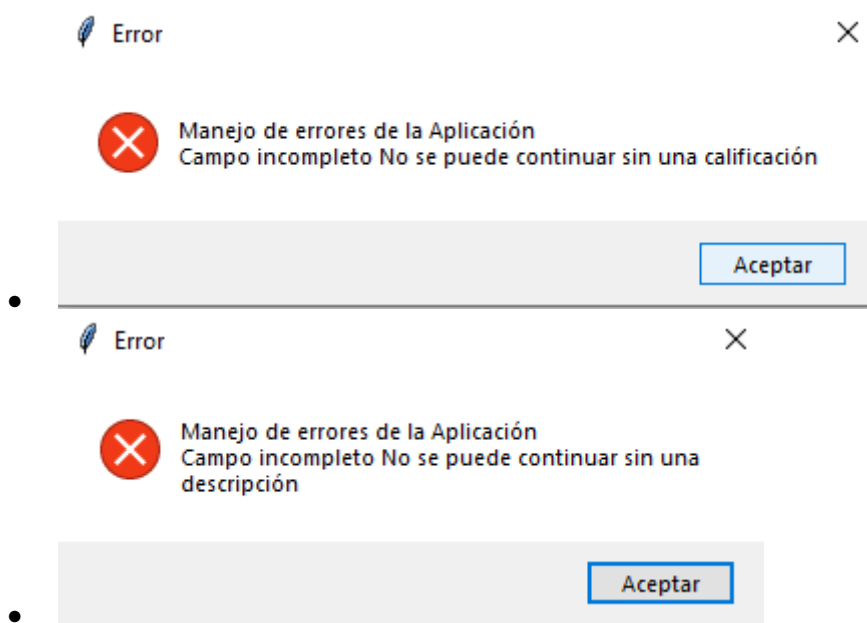
        # Validar que haya una descripción
        if not descripcion:
            raise DescripcionIncompleta()

        funeraria.setCalificacion(calificacion)
        funeraria.setDescripcion(descripcion)

        tk.messagebox.showinfo("Finalizado", "Gracias por calificar el proceso. Proceso finalizado.")
        frame_calificacion.destroy() # Destruir el frame al finalizar
    except CamposIncompletos as e:
        tk.messagebox.showerror("Error", str(e))

```

- **DescripcionIncompleta:** Se lanza cuando falta una descripción. Muestra el mensaje "No se puede continuar sin una descripción".



4. Clase errorNumeros (validación de números):

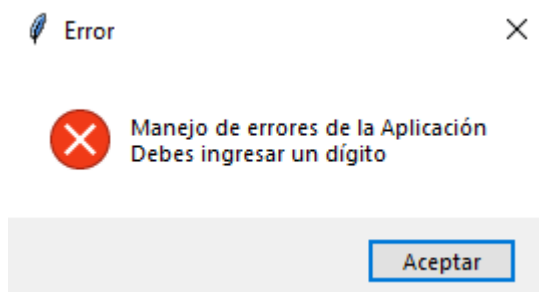
- **Hereda de:** ErrorAplicacion.
- **Propósito:** Manejar errores relacionados con la validación de números, ya sea enteros o flotantes.
- **Métodos de validación:**
 - **numeroEntero:** Intenta convertir un valor a un entero. Si falla, lanza una excepción con el mensaje "Debes ingresar un dígito".

```
# Confirmar compra
def confirmar_compra():
    producto_seleccionado = lista_productos_var.get()
    proveedor_seleccionado = lista_proveedores_var.get()
    cantidad_comprada = cantidad_var.get()

    try:
        # Validar que la cantidad sea un número entero
        errorNumeros("Validación cantidad").numeroEntero(cantidad_comprada)

        cantidad_comprada = int(cantidad_comprada) # Convertir a entero si es válido

        if producto_seleccionado and proveedor_seleccionado and cantidad_comprada > 0:
            # Actualizar el stock del producto
            nombre_producto = producto_seleccionado.split(" ")[0]
            for p in productos_faltantes:
                if p.getNombre() == nombre_producto:
                    p.setCantidad(p.getCantidad() + cantidad_comprada)
                    tk.messagebox.showinfo("Éxito", f"Compra de {cantidad_comprada} unidades de {nombre_producto} realizada con éxito.")
                    break
```



- **numeroFloat:** Similar a numeroEntero, pero para números de punto flotante (decimales).

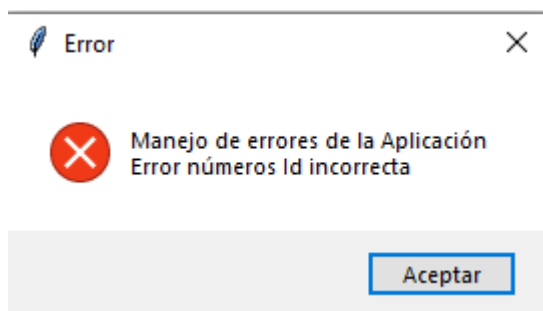
5. Clases hijas de errorNumeros (validación de IDs, peso y estatura):

Estas clases son para manejar errores específicos en la validación de ciertos atributos como IDs, peso y estatura.

6. errorIds:

- **Propósito:** Verifica si un valor dado es un ID válido (un número entero dentro de un rango específico).
- **Constructor:** Llama a numeroEntero para verificar si el valor es un número entero. Luego, comprueba si está dentro de los límites (valMin, valMax). Si no está dentro del rango, lanza un error con el mensaje "Id incorrecta".

```
def porInventario(entradaInventario, inventarioDefault, traslado):
    if entradaInventario.continuar():
        num=0
        try:
            urnaTumba=inventarioDefault[int(entradaInventario.getValores()[0])]
            if int(entradaInventario.getValores()[0])<0:
                errorIds(inventarioDefault[int(entradaInventario.getValores()[0])], "El ID ingresado es incorrecto", 0, len(urnaTumba)-1)
            cliente=urnaTumba.getCliente()
            num=1
        except:
            errorIds(entradaInventario.getValores()[0], "El ID ingresado no es correcto")
            entradaInventario.borrar()
        if num==1:
            texto=f"Has seleccionado al cliente {cliente}\n ¿Deseas continuar?"
            result = tk.messagebox.askyesno("Confirmar Datos", texto)
            if result:
                siguiente(frame, cliente, traslado)
            else:
                clienteIncompleto()
                frameCementerios.desbloquearOpciones()
```



7. errorPeso:

- **Propósito:** Valida el peso del cliente, comprobando si está dentro de un rango permitido.
- **Constructor:** Llama a numeroFloat para validar que el valor sea un número decimal, luego verifica si está dentro del rango (pesoMin y pesoMax). Si no lo está, lanza un error con el mensaje "El valor del peso del cliente no es correcto"

```
def validarUrnas(valoresUrna):
    num=0
    if valoresUrna.continuar():
        categoria =(valoresUrna.getValores()[0])

        peso = (valoresUrna.getValores()[1])

        try:
            int(categoria)
            if int(categoria)<0 or int(categoria)>2:
                errorIds(int(categoria),"La categoria ingresada no es correcta",0,2)
        except:
            errorIds(categoria,"La categoria ingresada no es correcta",0,2)
        try:
            float(peso)
            if float(peso)>=120 or float(peso)<0:
                errorPeso(peso,120)
            num=1
        except:
            errorPeso(peso,120)
    if num==1:
        valoresUrna.bloquear()
        btnContinuar.destroy()
        tablaUrnas(frame,cementerio,crematorio,cliente,valores,categoria,peso)
```

Error



Manejo de errores de la Aplicación

Error números El valor del peso del cliente no es correcto

Aceptar


8. errorEstatura:

- **Propósito:** Similar a errorPeso, pero para la estatura del cliente. Verifica si la estatura está en el rango permitido y, de no estarlo, lanza un error con el mensaje "El valor de la estatura del cliente no es correcta".

```
def organizacion():
    if datoCliente.continuar() and datoEstatura.continuar():
        cliente=clientes[datoCliente.getValores()[0]]
        num=0
        try:
            float(datoEstatura.getValores()[0])
            estatura=float(datoEstatura.getValores()[0])
            if estatura>2 or estatura <0:
                errorEstatura(estatura,2)
            num=1
        except:
            errorEstatura((datoEstatura.getValores()[0]),2)
            datoEstatura.borrar()
    if num==1:
        datoCliente.bloquearOpciones()
        datoEstatura.bloquear()
        btnContinuar.destroy()

        cementerios = funeraria.gestionEntierro(cliente, iglesia,estatura)
        datoCementerio=frame1(frame,["Cementerios con tumbas disponibles:"],[cementerios])

        btnContinuar1= tk.Button(frame,text="Continuar", command=lambda:confirmacion(datoCementerio,cementerios,estatura,cliente))
        btnContinuar1.pack(side="top",pady=10)
```

 Error



Manejo de errores de la Aplicación
Error números El valor de la estatura del cliente no es correcta

Aceptar

9. FieldFrame

En el ítem 2.3.1 se solicita la implementación de la clase FieldFrame, en esta parte de la memoria escrita usted debe:

Descripción: Explique detalladamente la implementación de esta clase y describa cada uno de sus componentes. Además, especifique el alcance y la funcionalidad que esta clase tiene en la interfaz gráfica de la aplicación.

Captura de pantalla: Incluya capturas de pantalla de diversas partes del código donde se utilicen instancias y objetos de la clase FieldFrame. Estas capturas deben mostrar claramente cómo y dónde se implementan y utilizan estas instancias en el código.

```
iuMain > frame.py > FieldFrame > __init__
1  from tkinter import ttk, Label, Frame, Button, Entry
2  import tkinter as tk
3  from iuMain.manejoErrores.errorAplicacion import CamposIncompletos
4
5
6  class FieldFrame(Frame):
7      def __init__(self, master, nombresEtiquetas, etiquetas, valores=None, editables=None):
8          super().__init__(master)
9          self.frame = Frame(master)
10         self.pack()
11         self.nombresEtiquetas=nombresEtiquetas
12         self.etiquetas=etiquetas
13         self.valores=valores
14         self.editables=editables
15         self.btnBorrar=None
16
17         # Asegurar que default_values, editables, y criterios_names sean listas
18         if self.valores is None:
19             self.valores = [''] * len(self.etiquetas)
20         if self.editables is None:
21             self.editables = [True] * len(self.etiquetas)
22
23         self.entries = []
24         self.secundario=Frame(self)
25         self.secundario.pack()
26
27         self.widget()
28
29
30     def widget(self):
31         self.entries=[]
32         # Crear etiquetas y cajas de texto
33         for etiqueta, valor, editable in zip(self.etiquetas, self.valores, self.editables):
34             entrada = Frame(self.secundario)
35             label = Label(entrada, text=etiqueta)
36             entry = Entry(entrada)
37
38             # Configurar el valor por defecto
39             entry.insert(0, valor)
40
41             # Configurar si es editable o no
42             entry.config(state=tk.NORMAL if editable else tk.DISABLED)
43
44             # Organizar los widgets
45             label.pack(side=tk.LEFT)
46             entry.pack(side=tk.LEFT)
47
48             # Añadir al contenedor de entradas
49             self.entries.append(entr
```

```

50
51         # Empaquetar el frame de cada entrada
52         entrada.pack(padx=5, pady=5, fill=tk.X)
53
54         self.btnBorrar = tk.Button(self.secundario, text="Borrar", command=self.borrar)
55         self.btnBorrar.pack(side=tk.LEFT, padx=5)
56
57         #button_frame.pack(pady=(5, 10))
58
59     def bloquear(self):
60         # Cambiar el estado de todas las entradas a 'disabled'
61         for entrada in self.entries:
62             entrada.config(state=tk.DISABLED)
63         if self.btnBorrar:
64             self.btnBorrar.destroy()
65
66     def getValores(self):
67
68         valores = [entrada.get() for entrada in self.entries]
69         print(f"Valores obtenidos: {valores}")
70         return valores
71
72     def borrar(self):
73         for entrada, valor in zip(self.entries, self.valores):
74             entrada.config(state=tk.NORMAL)
75             entrada.delete(0, tk.END)
76             entrada.insert(0, valor)
77             entrada.config(state=tk.NORMAL if entrada.cget('state') == tk.NORMAL else tk.DISABLED)
78     def continuar(self, mensaje=""):
79         # Verifica que todos los campos no estén vacíos
80         for entrada in self.entries:
81             if entrada.get().strip() == '':
82                 CamposIncompletos(mensaje)
83                 return False
84         return True
85
86
87 class frame1(Frame):
88     def __init__(self, master, etiquetas, opciones):
89         super().__init__(master)
90         self.pack() # Usa pack para el Frame principal
91
92         self.master=master
93         self.etiquetas=etiquetas
94         self.opciones=opciones
95         self.opcionesAlmacenadas = []
96         self.etiquetasAlmacenadas = []

```

```

98         self.secundario = Frame(self)
99         self.secundario.grid(row=0, column=0)
100
101         self.widget()
102         #btnContinuar=Button()
103     def continuar(self,mensaje="faltante"):
104         seleccionados = all(combobox.get() for combobox in self.opcionesAlmacenadas)
105         if seleccionados:
106             return True
107         else:
108             CamposIncompletos(mensaje)
109         return False
110
111     def widget(self):
112         for i,etiquetaFor in enumerate(self.etiquetas):
113             etiqueta = Label(self.secundario, text=etiquetaFor)
114             etiqueta.grid(row=i,column=0,padx=5,pady=5,sticky="e")
115             self.etiquetasAlmacenadas.append(etiqueta)
116             opciones = ttk.Combobox(self.secundario, values=self.opciones[i], state="readonly",width=30)
117             opciones.grid(row=i,column=1,padx=5,pady=5,sticky="w")
118             self.opcionesAlmacenadas.append(opciones)
119
120             #btnContinuar=Button(self.secundario,text="Continuar", command=self.continuar)
121             #btnContinuar.grid(row=i+1,column=2,columnspan=2)
122
123     def getValores(self):
124         listaIndices =[]
125         if self.continuar():
126             for i,opcionSeleccionada in enumerate(self.opcionesAlmacenadas):
127                 eleccion = opcionSeleccionada.current()
128                 listaIndices.append(eleccion)
129         return listaIndices
130
131     def bloquearOpciones(self):
132         for opcion in self.opcionesAlmacenadas:
133             opcion.config(state='disabled')
134
135     def desbloquearOpciones(self):
136         for opcion in self.opcionesAlmacenadas:
137             opcion.config(state="normal")
138
139 class tablas(Frame):
140     def __init__(self, master, etiquetas, valores):
141         super().__init__(master)
142         self.pack()
143         self.etiquetas=etiquetas

```

```

143         self.etiquetas=etiquetas
144         self.valores=valores
145
146         self.secundario = Frame(self)
147         self.secundario.grid(row=0, column=0)
148         self.widget()
149
150     def widget(self):
151         # Crear encabezados de columna
152         for i, c in enumerate(self.etiquetas):
153             etiqueta = Label(self.secundario, text=c,borderwidth=2, relief="solid", width=12, height=1, bg="lightgray", font=('Helvetica', 11, 'bold'))
154             etiqueta.grid(row=i,column=0,padx=2,pady=3)
155
156         # Crear filas de valores
157         for a, fila in enumerate(self.valores):
158             for i, valor in enumerate(fila):
159                 valor_label = Label(self.secundario, text=valor, borderwidth=1, relief="solid", width=25, height=1)
160                 valor_label.grid(row=a, column=i+1, padx=2, pady=1)
161
162
163

```

1. Clase FieldFrame:

- **Propósito:** Crear un formulario dinámico de etiquetas y entradas (Entry) donde cada entrada puede tener un valor predeterminado y se puede configurar si es editable o no.
- **Constructor (__init__):** Inicializa un contenedor de etiquetas y entradas en función de los parámetros que recibe:
 - nombresEtiquetas: Nombres que identificarán a cada campo.

- etiquetas: Las etiquetas visibles que acompañan a cada campo de texto.
- valores: Los valores predeterminados que se rellenan en los campos (opcional).
- editables: Si cada campo es editable o no.
- **Método widget:**
 - Crea etiquetas y entradas en un ciclo, y las añade a la interfaz gráfica. Si editables es False, desactiva la entrada.
 - Crea un botón "Borrar" para restaurar los valores predeterminados.
- **Método bloquear:** Desactiva todas las entradas, lo que las hace no editables.
- **Método getValores:** Devuelve los valores actuales de todas las entradas.
- **Método borrar:** Restaura los valores predeterminados en cada entrada.
- **Método continuar:** Verifica si todos los campos tienen un valor antes de continuar. Si falta alguno, lanza la excepción CamposIncompletos.

2. Clase frame1:

- **Propósito:** Crear un conjunto de etiquetas y menús desplegables (Combobox) que se pueden seleccionar en una interfaz gráfica.
- **Constructor (__init__):** Inicializa etiquetas y menús desplegables basados en los parámetros etiquetas y opciones:
 - etiquetas: Las etiquetas visibles que describen las opciones.
 - opciones: Listas de opciones disponibles para cada menú desplegable.
- **Método widget:** Crea las etiquetas y los menús desplegables.
- **Método continuar:** Verifica si todas las opciones en los menús desplegables han sido seleccionadas. Si falta una opción, lanza la excepción CamposIncompletos.
- **Método getValores:** Devuelve los índices de las opciones seleccionadas en los menús desplegables.
- **Métodos bloquearOpciones y desbloquearOpciones:** Bloquean o desbloquean los menús desplegables para que sean o no editables.

3. Clase tablas:

- 10. Propósito:** Mostrar datos en forma de tabla, donde las filas y las columnas contienen etiquetas y valores.
- 11. Constructor (__init__):** Inicializa una tabla con etiquetas (encabezados de columna) y valores (datos de las filas).
- 12. Método widget:** Crea las etiquetas de las columnas y las filas, organizando los datos en una estructura de tabla.


```

def seleccionCliente(frame,valores):

    if valores.continuar():
        funerarias= Establecimiento.filtrarEstablecimiento("funeraria")
        #Funeraria seleccionada
        funeraria= funerarias[(valores.getValores())[0]]
        indiceCliente =valores.getValores()[1]
        etiquetaCliente=[]

        if indiceCliente == 0:
            etiquetaCliente=["Clientes mayor de edad"]
            listaClientes=funeraria.buscarCliente("adulto")
            opcionesCliente=[listaClientes]

        elif indiceCliente==1:
            etiquetaCliente=["Clientes menor de edad"]
            listaClientes=funeraria.buscarCliente("niño")
            opcionesCliente=[listaClientes]

        global current_frame,separador
        if current_frame:
            current_frame.destroy()
        if separador:
            separador.destroy()

        frameSeparador=tk.Frame(frame)
        frameSeparador.pack(pady=25)
        valorCliente = frame1(frame,etiquetaCliente,opcionesCliente)

```

```

321     def datosCrematorio():
322
323         if entradaUrna.continuar():
324             num=0
325             try:

```

```

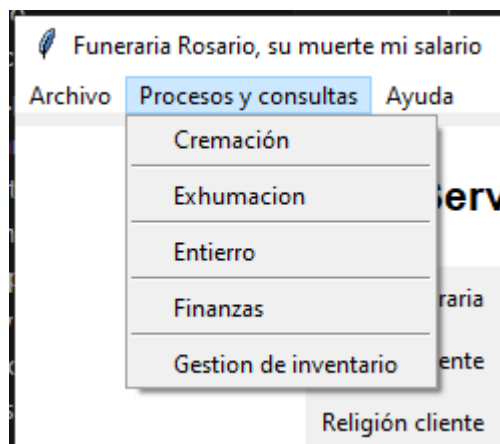
def urnas(frame,cementerio,crematorio,cliente,valores):
    #titulo(frame,"")
    valores.bloquearOpciones()
    frameSeparador=tk.Frame(frame)
    frameSeparador.pack(pady=20)
    valoresUrna=FieldFrame(frame,[],["Categoria urna (0-2)","Peso cliente (0-120)kg"],[0,0])

```

13. Manual de usuario.

FUNCIONALIDAD ENTIERRO.

Primer paso seleccionar la Funcionalidad de Entierro



Se mostrara por pantalla la siguiente información

Servicio de Entierro

Funeraria	<input type="text"/>
Cliente	<input type="text"/>
Religión cliente	<input type="text"/>
<input type="button" value="Continuar"/>	

En este paso, debemos seleccionarla funeraria.

Funeraria	<input type="text"/>
Cliente	<input type="text"/>
Religión cliente	<input type="text"/>

Luego el tipo de cliente

Servicio de Entierro

Funeraria	<input type="text"/>
Cliente	<input type="text"/>
Religión cliente	<input type="text"/>
<input type="button" value="Continuar"/>	

Luego debemos seleccionar la religion del cliente.

Servicio de Entierro

Funeraria

Caminos de Luz

Cliente

Religión cliente

CRISTIANISMO

HINDUISMO

BUDISMO

CRISTIANISMO

ISLAM

JUDAISMO

TAOISMO

Una vez hayamos seleccionado todas las opciones daremos click en el boton continuar.

Servicio de Entierro

Funeraria

Caminos de Luz

Cliente

Mayor de edad

Religión cliente

CRISTIANISMO

Continuar

Nos solicitan elegir un cliente y agregar la estatura del cliente.

Datos Cliente

Cliente

Ingrese la estatura del cliente

Borrar

Continuar

Procedemos a llenar los datos

Datos Cliente

Cliente

Ingrese la estatura del cliente

Despues de seleccionar, se guardan los datos y nos seleccionan un cementerio con tumbas disponibles

Datos Cliente

Cliente

Ingrese la estatura del cliente

Cementerios con tumbas disponibles:

Luego se creara la invitacion para sus familiares.

Organizacion cementerio

A continuación encontrará la invitación y resumen de su evento de Entierro

CONYUGUE Libia
PADRE Armando
HIJO Lyriel
HERMANO Andres

Hora de la Ceremonia: 10:26

Debemos dar en el boton de continuar que aparece en la pestaña.

y se nos mostraran las siguientes opciones.

Organización Tumba

Tumba	mbita Aquí Reposa un Corazón Nd	mbita Siempre en Nuestros Corazo	Tumbita Lugar de Paz
ID	0	1	2

Seleccione el ID de la tumba:

En donde debemos elegir la tumba.

Luego daremos clic en en continuar y nos mostrar la factura.

Detalle final

Resumen de los datos de entierro

Se generó la siguiente factura:

FACTURA

Descripción: Pago Adornos Entierro

Familiar: PADRE Armando

Concepto: Rosas - Precio unitario: 30000 - Cantidad: 1

Concepto: Lirios - Precio unitario: 35000 - Cantidad: 2

Concepto: Claveles - Precio unitario: 40000 - Cantidad: 2

Concepto: Orquídeas - Precio unitario: 45000 - Cantidad: 1

Concepto: Peonías - Precio unitario: 50000 - Cantidad: 2

Total: 325000

GESTION INVENTARIO.

Procesos y consultas	Ayuda
Cremación	
Exhumacion	
Entierro	
Finanzas	
Gestion de inventario	

Como primer paso, debemos seleccionar la funcionalidad.

Una vez completado el primer paso, se nos pide seleccionar la Funeraria.

Seleccione una funeraria:

Eterna Paz

Eterna Paz

Caminos de Luz

Recuerdos Eternos

Al momento de seleccionar, daremos clic en continuar.

Se nos mostrar una lista de los productos mas vendidos en dicha funeraria, debemos dar click en continuar.

Producto	Cantidad Vendida
Portarretratos digitales	45
Velas blancas	30
Vestidos de dama	20
Álbumes de fotos	18
Joyas conmemorativas	15
Medalla conmemorativa	12
Trajes de caballero	10
Velas rojas	5

El sistema realizara un analisis, y nos mostrar una posible intencion de intercambio con otra seda.

Análisis de Intercambio

Productos Disponibles para Intercambio

Productos en Caminos de Luz

Trajes de caballero - Cantidad: 6
Vestidos de dama - Cantidad: 18
Medalla conmemorativa - Cantidad: 10
Joyas conmemorativas - Cantidad: 25
Álbumes de fotos - Cantidad: 9
Portarretratos digitales - Cantidad: 15
Velas rojas - Cantidad: 7
Velas blancas - Cantidad: 0

Productos en Recuerdos Eternos

Trajes de caballero - Cantidad: 6
Vestidos de dama - Cantidad: 8
Medalla conmemorativa - Cantidad: 10
Joyas conmemorativas - Cantidad: 0
Álbumes de fotos - Cantidad: 30
Portarretratos digitales - Cantidad: 35
Velas rojas - Cantidad: 7
Velas blancas - Cantidad: 0

Continuar

procederemos dando en continuar.

Se nos despliegan las siguientes opciones

Realizar Intercambio

Vestidos de dama (Stock: 6) ▾

Cantidad a transferir

Confirmar Producto

Seleccionar Empleado

Seleccionar Vehículo

Finalizar Intercambio

Cancelar

Como prima parte, debemos elegir el producto a intercambiar y la cantidad a transferir, luego nos da la opción de delegar empleados y vehículos para la labor

Para confirmar el producto le damos click en el botón con el mismo nombre.

podemos seleccionar empleado

Seleccionar Empleado

▾

Confirmar Selección

Cancelar

o seleccionar Vehículo

Seleccionar Vehículo

Confirmar Selección

Cancelar

En cada opción, se nos mostrarán las opciones permitidas.

Finalmente, le damos en finalizar intercambio, y se nos desplegarán las opciones de compra. en este apartado, podremos comprar los productos con menos de 10 existencias, además de contratar o comprar vehículos a nuestro gusto.

Comprar Productos

Productos con menos de 10 existencias:

Cantidad a comprar

Confirmar Compra

Finalizar Compra

Cancelar

Comprar Vehículos

Contratar Empleados

Comprar Productos

Productos con menos de 10 existencias:

Vestidos de dama (Stoc

ARA productos

Cantidad a comprar

10

Confirmar Compra

Finalizar Compra

Cancelar

Comprar Vehículos

Contratar Empleados

Comprar Vehículos

Selecciona un proveedor de vehículos:

KTM

Vehículos disponibles para comprar:

COCHERESPETO

Confirmar Compra

Finalizar

Cancelar

Contratar Empleados

Selecciona un proveedor de empleados:

Establecimiento 6 ▼

Empleados disponibles para contratar:

Andrea Gómez ▼

Confirmar Contratación

Finalizar

Cancelar

al final del proceso, se nos pide calificar

Calificación del Proceso

Calificación (1-10):

Descripción del proceso:

Finalizar

al rellenar los datos, damos en finalizar y concluimos con el fin del sistema.

FUNCIONALIDAD EXHUMACIÓN

Paso1 : Debemos elegir la Funcionalidad en el menu de procesos.

Procesos y consultas Ayuda

Cremación
Exhumacion
Entierro
Finanzas
Gestion de inventario

Paso 2: Inmediatamente el sistema nos solicitara los datos iniciales para llevar a cabo la funcion.

Servicio de Exhumación

La exhumación es el proceso de retirar un cuerpo de su lugar de sepultura

Ingrese los siguientes datos para la búsqueda del Cliente

Seleccione la funeraria:

Buscar Cliente en:

Cliente:

Continuar

Cada barra, nos mostrara las diferentes opciones, una vez elegidas le daremos en continuar para el siguiente paso.

Servicio de Exhumación

La exhumación es el proceso de retirar un cuerpo de su lugar de sepultura

Ingrese los siguientes datos para la búsqueda del Cliente

Seleccione la funeraria:

Buscar Cliente en:

Cliente:

Continuar

Los datos se guardan y el sistema nos muestra y pide seleccionar un cementerio de ceniza de donde se buscara al cliente para realizar el proceso y una opción de traslado, el usuario deberá elegir según sus preferencias. y dar click en continuar.

Servicio de Exhumación

La exhumación es el proceso de retirar un cuerpo de su lugar de sepultura

Ingrese los siguientes datos para la búsqueda del Cliente

Seleccione la funeraria:	Eterna Paz	▼
Buscar Cliente en:	Urna default	▼
Cliente:	Menor de edad	▼

Cementerios cenizas	Jardín de la Eternidad	▼
Opciones para traslado	cementerio cenizas	▼

Continuar

Despues se nos mostrada información de las urnas disponibles para la exhumacion, en este apartado el usuario debiera elegir la urna segun el id mostrado en la aplicacion.

Urnitas	default
Cliente	Miguel Rodríguez
Tipo	fija
ID	0

Indique el ID de la Urnitas

0

Borrar

Continuar

Una vez seleccionada el id, daremos click en continuar

El sistema nos solicita datos extra del cliente, como el peso, y la religion.

Organización del traslado

Ingrese el peso del cliente (0-120) kg :

Borrar

Seleccione la religión con la que se va a realizar la ceremonia del cliente

Religión	HINDUISMO	BUDISMO	CRISTIANISMO	TAOISMO
Nombre Iglesia	Kamakhya Temple	Templo de Nanjing	Sagrada Familia	Valle del Jade
ID	1	2	3	4

Indique el ID de la iglesia

Borrar

Continuar

Procedemos a completar los campos correspondientes. y daremos click en continuar.

Organización del traslado

Ingrese el peso del cliente (0-120) kg : 60				
<input type="button" value="Borrar"/>				
Seleccione la religión con la que se va a realizar la ceremonia del cliente				
Religión	HINDUISMO	BUDISMO	CRISTIANISMO	TAOISMO
Nombre Iglesia	Kamakhya Temple	Templo de Nanjing	Sagrada Familia	Valle del Jade
ID	1	2	3	4

Indique el ID de la iglesia 4	
<input type="button" value="Borrar"/>	
<input type="button" value="Continuar"/>	

Luego el sistema nos mostrara, los cementerios disponibles con suficiente espacio.

Organización traslado

Cementerio	Campos de tranquilidad	Valle del Silencio
Inventario disponible	2	3
ID	0	1

Indique el ID del Cementerio 0	
<input type="button" value="Borrar"/>	
<input type="button" value="Continuar"/>	

Luego el sistema encontrará la urna mas congruente con las elecciones tomadas hasta el momento.

Cementerio Campos de tranquilidad	
urna	Sombra Amada
Tipo	ordinaria
<input type="button" value="Continuar"/>	

Al dar en continuar se mostrara en pantalla la organización de los familiares dentro de la iglesia.

Organización de los familiares dentro de la iglesia

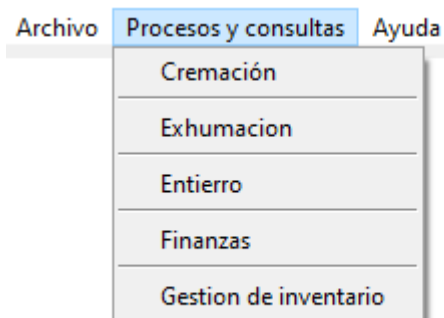
Silla [1] - Familiar CONYUGE Alberto Flores para decorar silla - Rosas
Silla [2] - Familiar PADRE Mario Flores para decorar silla - Rosas
Silla [3] - Familiar PADRE Samantha Flores para decorar silla - Lirios
Silla [4] - Familiar HERMANO Carlos Flores para decorar silla - Lirios

Regresar

y con esto concluimos el proceso.

FUNCIONALIDAD CREMACION

Paso 1: Seleccionamos la función de cremación.



Al iniciar el proceso el sistema solicita llenar dos campos. La funeraria y el tipo de cliente.

Servicio de Cremación

Funeraria:	<input type="text" value="Caminos de Luz"/>
Cliente:	<input type="text" value="Menor de edad"/>
<input type="button" value="Continuar"/>	

Al llenar los campos, daremos en continuar.

En este momento el sistema nos pide elegir al cliente al que vamos a cabo el proceso.

Servicio de Cremación

Funeraria: Caminos de Luz

Cliente: Menor de edad

Continuar

Cientes menor de edad Rafael Morales

Continuar

En el siguiente paso, se debe seleccionar los crematorios y la iglesia donde se va a realizar la ceremonia.

Organizacion Crematorio

Se debe seleccionar el crematorio y la iglesia por la que se quiere realizar la ceremonia

Los crematorios disponibles para la afiliación oro son:

Crematorios Crematorio del Silencio

Iglesias disponibles:

Religión	HINDUISMO	BUDISMO	CRISTIANISMO	TAOISMO
Nombre Iglesia	Kamakhya Temple	Templo de Nanjing	Sagrada Familia	Valle del Jade
ID	1	2	3	4

Indique el ID de la iglesia 4

Borrar

Continuar

Luego el sistema nos pedira seleccionar la hora

Crematorio Crematorio del Silencio

Horarios disponibles: 08:06 Am

Continuar

Luego el sistema nos pedira elegir un empleado para la organizacion del crematorio.

Organizacion Cementerio

Seleccione los siguientes datos:

Empleados:	<input type="text" value="Natalia Ortega"/>
Cementerios cenizas	<input type="text" value="Cementerio del Silencio"/>

y el horario

Cementerio Cementerio del Silencio

Horarios disponibles:

Al completar el paso anterior, el sistema solicitar una categoria de urna y que especifiquemos el peso del cliente.

Categoria urna (0-2)	<input type="text" value="0"/>
Peso cliente (0-120)kg	<input type="text" value="12"/>

Luego el sistema nos pedirá seleccionar una urna, para esto nos presenta la información de diferentes urnas, debemos elegir segun el id

Organizacion Cementerio

Seleccione los siguientes datos:

Empleados:	Natalia Ortega	▼
Cementerios cenizas	Cementerio del Silencio	▼

Categoría urna (0-2) 0

Peso cliente (0-120)kg 12

Urnas disponibles para su religión: fija, ordinaria

Urnita	Urnita Memoria Serene	Urnita Descanso Sagrado	Urnita del Futuro
Cementerio	Cementerio del Silencio	Cementerio del Silencio	Cementerio del Silencio
Tipo	fija	ordinaria	fija
ID	0	1	2

Indique el ID de la Urna

Borrar

Continuar

Al elegir la urna el sistema mostrará la invitación a la ceremonia.

Invitación a la ceremonia

Asunto: Invitación a la Ceremonia de Cremación de Rafael Morales

Invita

PADRE Camila

PADRE Luis

HERMANO Tomas

HERMANO Andres

Hora de la Ceremonia: 08:06

Lugar de Cremación: Crematorio del Silencio

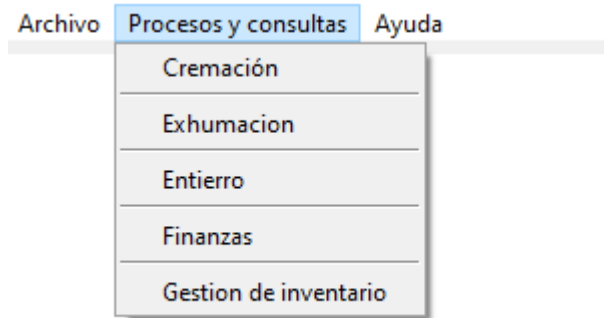
Centro religioso: Templo de Nanjing

Regresar

y con esto concluye el proceso.

FUNCIONALIDAD FINANZAS

Paso 1: Seleccionamos la función de finanzas.



Al iniciar el proceso el sistema solicita llenar dos campos. La funeraria y el tipo de servicio que se quiere realizar.

Servicio de finanzas

Funeraria:

Servicios:

Continuar

Al llenar los campos, daremos en continuar.

Para cobro clientes:

Cobro clientes

Los cementerios disponibles para la funeraria Eterna Paz son:

Cementerios

Continuar

El sistema solicita llenar un campo. El cementerio con el que se desea trabajar.

Al llenar el campo, daremos en continuar.

Cobro clientes

Los cementerios disponibles para la funeraria Eterna Paz son:

Cementerios

Los clientes disponibles para el cementerio Jardín de la Eternidad son:

Cientes

Continuar

En este momento el sistema nos pide elegir el cliente con el que vamos a realizar el proceso.

Seleccionamos uno y damos click en el botón continuar



Cobro de facturas del cliente: Juan Pérez, realizado correctamente

Aceptar

Se genera un informe del proceso

Para pagar facturas:

Pago Facturas

Las facturas disponibles para la funeraria Eterna Paz son:

Facturas

Continuar

En este momento el sistema nos pide elegir la factura con la que vamos a realizar el proceso.

Seleccionamos uno y damos click en el botón continuar



Factura con ID: 109 pagada con éxito


Aceptar

Se genera un informe del proceso

Para pago empleados:

Pago empleados

Los empleados a los que se les puede liquidar su pago en la funeraria Eterna Paz son:

Empleados 

En este momento el sistema nos pide elegir un empleado con el que vamos a realizar el proceso.

Seleccionamos uno y damos click en el botón continuar




El trabajador ha hecho: 2 trabajos,
Y tiene una calificación de: 5
por lo que obtuvo una paga de: 10500.0

Se genera un informe del proceso

Para credito:

Credito

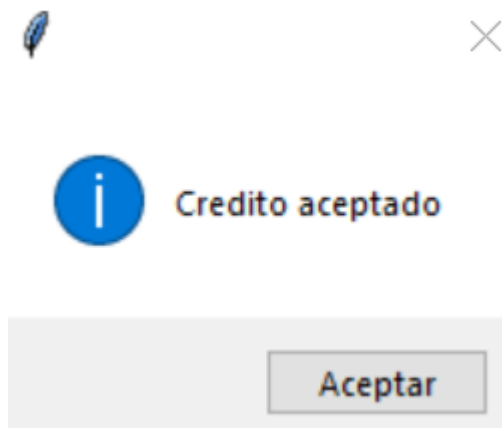
Los servicios de credito disponibles en la funeraria Eterna Paz son:

Servicios credito: 

En este momento el sistema nos pide elegir un servicio de crédito.

Seleccionamos uno y damos click en el botón continuar

Para pedir credito:



Se genera un informe del proceso

Para Pago credito:

Los creditos activos en la Eterna Paz son:

Credito activos:

Continuar

En este momento el sistema nos pide elegir un crédito activo con el que vamos a realizar el proceso.

Seleccionamos uno y damos click en el botón continuar

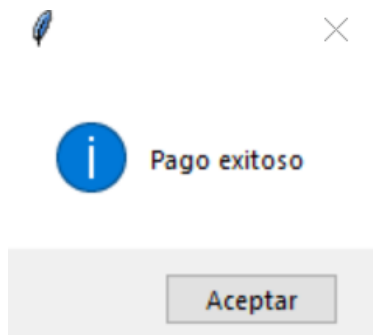
Que porcentaje del credito desea pagar

Porcentaje Credito:

Continuar

En este momento el sistema nos pide elegir el porcentaje del crédito que queremos pagar.

Seleccionamos uno y damos click en el botón continuar



Se genera un informe del proceso

Para ver crédito

Servicios credito: Ver credito

Los creditos activos en la Eterna Paz son:

Credito activos: Factura con ID: 178

Continuar

En este momento el sistema nos pide ingresar el crédito que queremos ver.

Seleccionamos uno y damos click en el botón continuar

Informe Credito

ID: 178
Precio: 542700.0
Porcentaje por pagar: 0.2

Aceptar

Se genera un informe del crédito y se muestra en pantalla.

Para reajuste de dineros:

Reajuste de dinero

Los servicios de reajuste de dinero disponibles en la Eterna Paz son:

Servicios reajuste dinero: Ver informe gastos

Continuar

En este momento el sistema nos pide seleccionar el servicio de reajuste de dinero que deseamos realizar.

Seleccionamos uno y damos click en el botón continuar

Para ver informe gastos:



Informe de gastos:
Facturas inventario: 2
Gastos inventario: 165000
Facturas transporte: 1
Gastos transporte: 70000
Facturas establecimientos: 1
Gastos establecimientos: 500
Facturas trabajadores: 0
Gastos trabajadores: 0
Facturas pago credito: 0
Gastos credito: 0

Aceptar

Se genera un informe de gastos de la funeraria y se muestra en pantalla.

Para reajuste:



La funeraria: Recuerdos Eternos requiere mayor cantidad de dinero para actualizar el inventario, por lo que se le ha transferido 1000000
La funeraria: Caminos de Luz requiere mayor cantidad de dinero para la compra y la gestion de vehiculos, por lo que se le ha transferido 1000000
La funeraria: Recuerdos Eternos requiere mayor cantidad de dinero para el pago a los establecimientos, por lo que se le ha transferido 1000000
No hubo Funerarias que necesitaran un reajuste de dinero para trabajadores
No hubo Funerarias que necesitaran un reajuste de dinero para credito

Aceptar

Se genera un informe de reajuste de dinero de las funerarias y se muestra en pantalla.