Tecnicatura Universitaria en Programación

Laboratorio de computación II
Programación II

Trabajo Práctico Integrador

Integrantes:

- Andre, Tomas Oscar.
- Canelo, Emmanuel Marcelo.
- Cecchel, José Luis.
- Suarez, Gabriel.

Año 2023

Índice

VISTAS

VistaCliente

La clase VistaCliente representa la interfaz de usuario para las acciones relacionadas con el cliente.

Métodos:

- **ingreso_menu(self)**: Método para solicitar al usuario que seleccione una opción del menú.
- **error_ingreso(self)**: Método para mostrar un mensaje de error cuando el usuario ingresa una opción incorrecta.
- **vista_atras(self)**: Método para solicitar al usuario que presione una tecla para volver al menú principal.
- **carga_nombre(self)**: Método para solicitar al usuario que ingrese su nombre.
- **carga_apellido(self)**: Método para solicitar al usuario que ingrese su apellido.
- **carga_dni(self)**: Método para solicitar al usuario que ingrese su DNI.
- **carga_telefono(self)**: Método para solicitar al usuario que ingrese su número de teléfono.
- **carga_direccion(self)**: Método para solicitar al usuario que ingrese su dirección.
- **error_documentacion(self)**: Método para mostrar un mensaje de error cuando no se completan todos los campos relacionados a los datos personales del cliente.
- **error_sin_evento(self)**: Método para mostrar un mensaje cuando no se encuentran eventos registrados con el DNI proporcionado.
- **dni_usado(self)**: Método para mostrar un mensaje cuando ya existe un evento registrado con el mismo usuario.
- **evento_reservado(self, fecha):** Método para mostrar la fecha del evento reservado.

VistaEvento

La clase VistaEvento representa la interfaz de usuario para las acciones relacionadas con los eventos.

Métodos:

- **ingreso_fecha(self)**: Método para solicitar al usuario que ingrese el día en el que le gustaría realizar el evento en el mes de julio.
- **error_ingreso(self)**: Método para mostrar un mensaje de error y solicitar al usuario que vuelva a intentarlo.
- **evento_ocupado(self)**: Método para mostrar un mensaje indicando que la fecha ya se encuentra reservada.
- **fecha_libre_proxima(self, dia)**: Método para preguntar al usuario si desea reservar la próxima fecha disponible.
- **ultima_fecha_libre(self, dia)**: Método para preguntar al usuario si desea reservar la última fecha disponible.
- **evento_guardado(self)**: Método para mostrar un mensaje indicando que se guardó con éxito la fecha del evento.
- **senia_evento(self, monto)**: Método para mostrar el monto de la seña requerida y solicitar al usuario que ingrese el monto correspondiente.
- **error_senia(self)**: Método para mostrar un mensaje de error en el ingreso del monto de la seña.
- **error_documentacion(self)**: Método para mostrar un mensaje de error indicando que se debe seleccionar una fecha para realizar la reserva.
- **reserva_exito(self)**: Método para mostrar un mensaje indicando que se realizó con éxito la reserva del evento.
- **fecha_evento(self, fecha)**: Método para mostrar la fecha del evento que se cancelará.
- **cancelacion_evento(self)**: Método para preguntar al usuario si desea cancelar el evento.
- **confirmacion_cancelacion(self)**: Método para mostrar un mensaje de confirmación de cancelación del evento.
- **error_buscar_dni(self):** Método para mostrar un mensaje de error indicando que no se encontró el DNI ingresado.
- **fecha_reservada(self, fecha)**: Método para mostrar un mensaje indicando que el usuario ya reservó la fecha y preguntar si desea modificarla.

- **devolucion_reserva(self, monto, total)**: Método para mostrar el monto que se devolverá al usuario correspondiente al 20% de la seña abonada y solicitar al usuario que presione una tecla para continuar.
- **cancelacion_fuera_fecha(self, fecha)**: Método para mostrar un mensaje indicando que la fecha del evento es próxima y no se realizará la devolución del 30% de la seña.

VistaPrincipal

La clase VistaPrincipal representa la interfaz de usuario principal del programa.

Métodos:

- **menu_presentacion(self)**: Método para mostrar un mensaje de bienvenida y presentación del programa.
- **menu_opciones(self)**: Método para mostrar el menú de opciones disponibles.
- **saludo_final(self)**: Método para mostrar un mensaje de despedida y agradecimiento al usuario.
- **ingreso_menu(self)**: Método para solicitar al usuario que seleccione una opción del menú.
- **error_ingreso(self)**: Método para mostrar un mensaje de error en el ingreso y solicitar al usuario que vuelva a intentarlo.
- **vista_atras(self)**: Método para solicitar al usuario que presione una tecla para volver al menú principal.

VistaServicio

La clase VistaServicio representa la interfaz de usuario para la selección y visualización de servicios.

Métodos:

- **titulo_menu(self)**: Método para mostrar un mensaje de título antes de mostrar el listado de servicios.
- **ingreso_menu(self)**: Método para solicitar al usuario que ingrese la opción deseada del menú.
- **error_ingreso(self)**: Método para mostrar un mensaje de error en el ingreso y solicitar al usuario que vuelva a intentarlo.
- **consulta_seguimiento(self)**: Método para solicitar al usuario si desea continuar seleccionando servicios.
- **error_valor_duplicado(self)**: Método para mostrar un mensaje de error cuando se ingresa un servicio que ya está seleccionado.
- **vista_atras(self)**: Método para solicitar al usuario que presione una tecla para volver al menú principal.
- **titulo_servicios_elegidos(self)**: Método para mostrar un título antes de la lista de servicios seleccionados.
- **servicios_elegidos(self, nombre)**: Método para mostrar el nombre de un servicio seleccionado.
- **seleccion_vacia(self)**: Método para mostrar un mensaje cuando no hay servicios seleccionados.
- **seleccion_eliminar_servicios(self)**: Método para solicitar al usuario que seleccione un servicio para eliminar.
- **mostrar_costo(self, gastoServicios, gastoAdm, total)**: Método para mostrar el detalle del costo total de los servicios seleccionados.
- **error_documentacion(self)**: Método para solicitar al usuario que realice al menos una selección de servicio para realizar la reserva.
- **servicios_para_eliminar(self, index, nombre)**: Método para mostrar la lista de servicios disponibles para eliminar, enumerados con un índice.

MODELOS

Cliente

La clase Cliente representa a un cliente y sus atributos.

Atributos:

- **nombre**: El nombre del cliente.
- **apellido**: El apellido del cliente.
- **dni**: El número de identificación del cliente.
- **direccion**: La dirección del cliente.
- **telefono**: El número de teléfono del cliente.

Métodos:

- **__init__(self, nombre, apellido, dni, direccion, telefono)**: Constructor de la clase Cliente. Recibe como parámetros el nombre, apellido, DNI, dirección y teléfono del cliente, y los asigna a las variables correspondientes.
- **get_nombre(self)**: Método para obtener el nombre del cliente.
- **get_apellido(self)**: Método para obtener el apellido del cliente.
- **get_dni(self)**: Método para obtener el DNI del cliente.
- **get_telefono(self)**: Método para obtener el teléfono del cliente.
- **get_direccion(self)**: Método para obtener la dirección del cliente.
- **set_nombre(self, newNombre)**: Método para establecer un nuevo valor para el nombre del cliente.
- **set_apellido(self, newApellido)**: Método para establecer un nuevo valor para el apellido del cliente.
- **set_dni(self, newDni)**: Método para establecer un nuevo valor para el DNI del cliente.
- **set_telefono(self, newTelefono)**: Método para establecer un nuevo valor para el teléfono del cliente.
- **set_direccion(self, newDireccion)**: Método para establecer un nuevo valor para la dirección del cliente.
- **__str__(self)**: Método especial para representar el objeto Cliente como una cadena de texto legible. Devuelve una cadena que contiene los valores de los atributos del cliente.

Evento

La clase Evento representa un evento y sus atributos.

Atributos:

- **fecha**: La fecha del evento.
- **cliente**: El cliente asociado al evento.
- **servicio**: El servicio seleccionado para el evento.
- **senia**: El monto de la seña pagada para el evento.
- **costo**: El costo total del evento.
- **estado**: El estado del evento (por ejemplo, "confirmado", "disponible").

Métodos:

- **__init__(self, fecha, cliente, servicio, senia, costo, estado)**: Constructor de la clase Evento. Recibe como parámetros la fecha, cliente, servicio, seña, costo y estado del evento, y los asigna a las variables correspondientes.
- **get_cliente(self)**: Método para obtener el cliente asociado al evento.
- **get_fecha(self)**: Método para obtener la fecha del evento.
- **get_costo(self)**: Método para obtener el costo total del evento.
- **get_servicio(self)**: Método para obtener el servicio seleccionado para el evento.
- **get_estado(self)**: Método para obtener el estado del evento.
- get_senia(self): Método para obtener el monto de la seña pagada para el evento.
- **set_cliente(self, newCliente)**: Método para establecer un nuevo valor para el cliente asociado al evento.
- **set_fecha(self, newFecha)**: Método para establecer un nuevo valor para la fecha del evento.
- **set_senia(self, newSenia)**: Método para establecer un nuevo valor para el monto de la seña pagada para el evento.
- **set_costo(self, newCosto)**: Método para establecer un nuevo valor para el costo total del evento.
- **set_servicio(self, newServicio)**: Método para establecer un nuevo valor para el servicio seleccionado para el evento.
- **set_estado(self, newEstado)**: Método para establecer un nuevo valor para el estado del evento.

- **calcular_senia(self)**: Método para calcular el monto de la seña del evento. Devuelve el resultado del cálculo, que es el costo total del evento multiplicado por 0.3.
- **__str__(self):** Método especial para representar el objeto Evento como una cadena de texto legible. Devuelve una cadena que contiene los valores de los atributos del evento.

Servicio

La clase Servicio representa un servicio y sus atributos.

Atributos:

- nombre: El nombre del servicio.
- costo: El costo del servicio.

Métodos:

- **__init__(self, nombre, costo)**: Constructor de la clase Servicio. Recibe como parámetros el nombre y costo del servicio, y los asigna a las variables correspondientes.
- **get_nombre(self)**: Método para obtener el nombre del servicio.
- **get_costo(self)**: Método para obtener el costo del servicio.
- **set_nombre(self, newName)**: Método para establecer un nuevo valor para el nombre del servicio.
- **set_costo(self, newCosto)**: Método para establecer un nuevo valor para el costo del servicio.
- **__str__(self, index)**: Método especial para representar el objeto Servicio como una cadena de texto legible. Recibe el índice del servicio y devuelve una cadena que contiene el nombre y costo del servicio.

CONTROLADOR

ControllerPrincipal

Estructura del Código

El código se divide en las siguientes secciones:

1. **Importación de módulos:** En esta sección se importan los módulos necesarios para el funcionamiento del programa. Entre ellos se encuentran os, sys, datetime, así como vistas y modelos personalizados.

2. **Definición de Clases:**
   - **ControllerPrincipal**: Esta clase representa el controlador principal del programa. Contiene métodos para inicializar y gestionar los servicios, eventos, clientes y vistas. También proporciona funciones para mostrar información, calcular costos, consultar disponibilidad y cargar datos de clientes.
   - **Servicio**: Esta clase representa un servicio y sus atributos, como el nombre y el costo.
   - **Evento**: Esta clase representa un evento y sus atributos, como la fecha, el cliente, el servicio, la señal, el costo y el estado.
   - **Cliente**: Esta clase representa un cliente y sus atributos, como el nombre, el apellido, el DNI, la dirección y el teléfono.
   - **VistaPrincipal**: Esta clase representa la vista principal del programa. Es responsable de mostrar el menú principal y las opciones al usuario, así como recibir la entrada del usuario y comunicarse con el controlador principal para ejecutar las acciones correspondientes.
   - **VistaEventos**: Esta clase representa la vista de eventos del programa. Muestra la lista de eventos disponibles y permite al usuario seleccionar una fecha para consultar la disponibilidad de eventos en esa fecha.
   - **VistaServicios**: Esta clase representa la vista de servicios del programa. Muestra la lista de servicios disponibles y permite al usuario seleccionar servicios y ver los servicios seleccionados actualmente.

3. **Métodos del Controlador Principal:**
    - **reset_variables(self)**: Este método restablece todas las variables a sus valores iniciales. Es llamado al iniciar el programa para asegurar un estado limpio.
    - **menu_principal(self)**: Este método muestra el menú principal del programa y gestiona las acciones del usuario. Permite al usuario seleccionar diferentes opciones, como elegir servicios, ver servicios seleccionados, calcular el costo total, consultar disponibilidad de eventos y salir del programa.
    - **iniciar_servicio(self)**: Este método lee los datos de los servicios desde un archivo y los guarda en una lista. Estos servicios representan las opciones disponibles para que los clientes elijan durante la gestión de eventos.
    - **iniciar_evento(self)**: Este método lee los datos de los eventos desde un archivo y los guarda en una lista. Los eventos contienen información sobre la fecha, el cliente, el servicio, la señal, el costo y el estado de cada evento programado.
    - **total_lineas_archivo(self)**: Este método devuelve el número total de líneas en un archivo de servicios. Se utiliza para calcular el número de servicios disponibles y mostrarlos al usuario.
    - **servicios_elegidos(self, estado)**: Este método muestra los servicios seleccionados o disponibles para elegir, según el estado proporcionado. El estado puede ser "1" o "2", lo que permite mostrar servicios que el cliente ha seleccionado previamente o servicios seleccionados listos para ser eliminados.
    - **servicios_para_elegir(self)**: : Este método muestra los servicios disponibles para que el usuario los elija. Utiliza el método servicios_elegidos(**self**) con el estado "1" para mostrar solo los servicios seleccionados.
    - **limpiar_consola(self)**: Este método limpia la consola o la pantalla para mejorar la presentación y evitar la acumulación de información en cada iteración del programa.
    - **mostrar_servicios(self)**: Este método muestra la lista de servicios disponibles.
    - **modificar_servicios_seleccionados(self)**: Este método permite al usuario eliminar servicios seleccionados. Muestra los servicios que el cliente ha seleccionado previamente utilizando el método servicios_elegidos() con el estado "2". Luego, el usuario puede elegir un servicio para eliminarlo de la lista de selecciones.

- **calcular_costo(self)**:Este método calcula el costo total de los servicios seleccionados. Suma los costos de los servicios para obtener el costo total
- **mostrar_costo(self, costo, total)**: Este método muestra el costo detallado de los servicios seleccionados. Recibe el costo total calculado en el método calcular_costo() y muestra el costo individual de cada servicio, así como el costo total.
- **consultar_disponibilidad(self)**: Este método consulta la disponibilidad de eventos en una fecha determinada.
- **carga_nombre(self)**: Este método permite al usuario ingresar el nombre de un cliente. Se utiliza durante el proceso de registro de un nuevo cliente.
- **carga_apellido(self)**: Este método permite al usuario ingresar el apellido de un cliente. Se utiliza durante el proceso de registro de un nuevo cliente.
- **carga_dni(self)**: Este método permite al usuario ingresar el DNI de un cliente. Se utiliza durante el proceso de registro de un nuevo cliente. También verifica si el DNI ya está en uso por otro cliente mediante el método verifica_dni().
- **verifica_dni(self, dni)**: Este método verifica si un número de DNI ya está en uso por otro cliente. Recibe un número de DNI y verifica si existe otro cliente con el mismo DNI en la lista de clientes.
- **carga_telefono(self)**: Este método solicita al usuario que ingrese un número de teléfono. Verifica si el número ingresado está vacío o contiene solo espacios en blanco. Si es válido, establece el número de teléfono del cliente y finaliza el bucle. Si se ingresa '1', regresa al menú principal.
- **carga_direccion(self):** Este método solicita al usuario que ingrese una dirección. Verifica si la dirección ingresada está vacía o contiene solo espacios en blanco. Si es válida, establece la dirección del cliente y finaliza el bucle. Si se ingresa '1', regresa al menú principal.
- **carga_senia(self)**: Este método calcula el costo del evento y solicita al usuario que ingrese una reserva (seña). Verifica si el valor ingresado es un número. Si el número no coincide con el cálculo de la seña o se ingresa '1', muestra un mensaje de error o regresa al menú principal. Si los datos ingresados son válidos, verifica la documentación ingresada y, si es correcta, carga el evento y el cliente en la base de datos, muestra un mensaje de reserva exitosa, restablece las variables y finaliza el bucle.

- **verificar_datos_ingresados(self)**: Este método verifica si se han ingresado todos los datos requeridos para cargar un evento y servicio en la base de datos. Si falta algún dato, muestra un mensaje de error correspondiente y devuelve el estado True. De lo contrario, devuelve el estado False.
- **cargar_evento(self)**: Este método carga un evento en la base de datos. Abre el archivo "eventos.txt" y reemplaza la línea correspondiente al evento con los datos actualizados. Si ocurre algún error durante la escritura en el archivo, muestra un mensaje de error.
- **cargar_cliente(self)**: Este método carga un cliente en la base de datos. Abre el archivo "clientes.txt" y verifica si el cliente ya está registrado mediante su número de documento (DNI). Si no está registrado, agrega una nueva línea con los datos del cliente al archivo. Si ocurre algún error durante la escritura en el archivo, muestra un mensaje de error.
- **cancelar_evento(self)**: Este método permite al usuario cancelar un evento. Solicita al usuario que ingrese un número de documento (DNI) y verifica si existe un evento asociado a ese cliente. Si se encuentra un evento, verifica si la cancelación está dentro del plazo permitido. Luego, muestra un mensaje de confirmación y calcula el monto de devolución correspondiente antes de eliminar el evento de la base de datos. Si el DNI no coincide con ningún evento o se cancela la operación, regresa al menú principal.
- **calcular_devolucion(self, dni)**: Este método calcula el monto de devolución para un cliente en función del monto de la seña. Recorre la lista de eventos y verifica si el DNI del cliente coincide. Si se encuentra una coincidencia, devuelve el monto de devolución.
- **verificar_cancelacion(self, fecha)**: Este método verifica si la cancelación de un evento está dentro del plazo permitido. Compara la fecha del evento con la fecha actual y devuelve True si la diferencia es mayor o igual a 15 días. De lo contrario, devuelve False.
- **eliminar_evento(self, fecha)**: Este método elimina del archivo “eventos.txt” el evento que coincida con el parámetro fecha que recibe el método y se le asigna la misma fecha con el resto de valores en “0”.
- **verificar_reserva(self):** Este método verifica el dni ingresado con los almacenados en el archivo “eventos.txt” y si encuentra coincidencia, muestra los servicios que tiene cargado dicho evento, caso contrario, muestra un error de que no se encontró el dni.

DIAGRAMA DE CLASES U.M.L