

Instituto Tecnológico CTC Colonia

**Analista Programador
Programación 2**

Obligatorio Grupal

Carlos Rodriguez

Baltazar Boné
Fausto Clara

2025

Índice

Índice	2
Abstract del Proyecto	3
Declaración de Autoría	4
Cronograma de Trabajo	5
Diccionario de Clases	6
Diagrama UML de Clases	13
Resultado de Pruebas	14
Juego de Datos Iniciales	15

Abstract del Proyecto

Descripción del Proyecto – Clínica Vida Sana

La aplicación Clínica Vida Sana es un sistema de gestión desarrollado en C#, para administrar las consultas médicas de la clínica. Permite registrar pacientes, médicos y turnos, así como gestionar pagos y emitir comprobantes.

El sistema está orientado al personal administrativo, quien puede agendar, modificar o cancelar consultas, verificar disponibilidad de médicos y generar reportes.

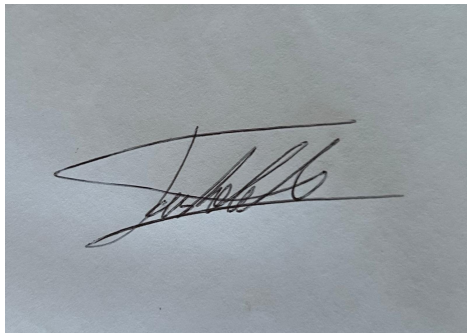
Entre sus principales funcionalidades se incluyen la gestión de usuarios, turnos y pagos, junto con la emisión de estadísticas sobre pacientes, médicos y especialidades.

El diseño del sistema se basa en una estructura modular y en principios de programación orientada a objetos, garantizando claridad, orden y facilidad para futuras ampliaciones.

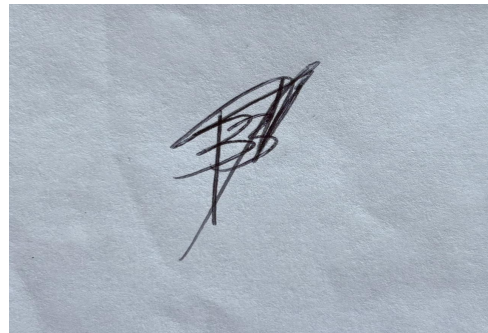
Declaración de Autoría

Nosotros, Fausto Clara y Baltazar Boné, declaramos que el trabajo que se presenta es de nuestra propia mano. Puedo asegurar que:

- La obra fue producida mientras cursaba la materia de Programación 2 con el docente Carlos Rodriguez;
- Hemos tenido en cuenta las clases dictadas por el Docente Carlos Rodriguez, quién además proporcionó el material;
- Cuando he consultado el trabajo publicado por otros, lo he atribuido con claridad;
- Cuando he citado obras de otros, he indicado las fuentes. Con excepción de estas citas, la obra es enteramente mía;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, he explicado claramente que fue construido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.
- Ninguna parte de este trabajo ha sido realizada exclusivamente con Inteligencia Artificial.

A photograph of a handwritten signature in black ink on a light-colored surface. The signature is stylized and appears to read 'Fausto Clara'.

Firma Fausto Clara

A photograph of a handwritten signature in black ink on a light-colored surface. The signature is stylized and appears to read 'Baltazar Boné'.

Firma Baltazar Boné

Cronograma de Trabajo

Fase	Tarea / Actividad	Responsable	Duracion estimada	Fecha de inicio	Fecha de fin
1	Preparación y planificación: definir estructura del proyecto, clases y relaciones	Baltazar y Fausto	1 día	20/10/2025	20/10/2025
2	Implementación de clases básicas: Paciente, Médico, Administrativo, Consulta, Pago	Baltazar y Fausto	2 días	21/10/2025	22/10/2025
3	Implementación de la clase Clinica con métodos CRUD y validaciones	Baltazar y Fausto	3 días	23/10/2025	26/10/2025
4	Carga de datos iniciales: CargaDeDatosIniciales.cs	Baltazar y Fausto	1 día	26/10/2025	26/10/2025
5	Implementación del menú y Program.cs: entradas, validaciones, navegación	Baltazar y Fausto	3 días	27/10/2025	29/10/2025

Diccionario de Clases

Clase: Persona (abstracta)

Descripción: Clase base para todas las personas del sistema (pacientes, médicos, administrativos). Contiene los datos comunes y garantiza la validación básica de información.

Atributos:

ID: int -> Identificador único de la persona (autoasignado).

Nombre: string -> Nombre de la persona.

Apellido: string -> Apellido de la persona.

NombreUsuario: string -> Email o nombre de usuario para login.

Contraseña: string -> Contraseña de acceso.

Métodos:

ToString(): string -> Devuelve un resumen de la persona (ID, nombre y apellido).

Relaciones: Clase base para Paciente, Medico y Administrativo.

Clase: Paciente

Descripción: Representa a un paciente registrado en la clínica. Hereda de Persona.

Atributos:

NumeroDocumento : string -> Documento de identidad del paciente.

Telefono : string -> Número de teléfono (solo dígitos).

FechaNacimiento : DateTime -> Fecha de nacimiento del paciente.

Email : string -> Correo electrónico del paciente.

ObraSocial : string -> Obra social (opcional).

Métodos:

ToString() : string -> Devuelve la información completa del paciente, incluyendo datos de contacto y obra social.

Relaciones: Un Paciente puede tener múltiples Consulta (turnos médicos).

Clase: Medico

Descripción: Representa a un médico de la clínica. Hereda de Persona.

Atributos:

Especialidad : string -> Ejemplo: Pediatría, Dermatología, Clínica Médica.

Matricula : string -> Matrícula profesional del médico.

DiasAtencion : List<string> -> Días de la semana en que atiende.

HorariosDisponibles : List<TimeSpan> -> Horarios disponibles en bloques de 30 minutos.

Métodos:

ToString() : string -> Devuelve la información completa del médico, incluyendo especialidad, matrícula, días y horarios disponibles.

Relaciones: Un Medico puede tener múltiples Consulta asignadas.

Clase: Administrativo

Descripción: Representa al personal administrativo o recepcionista de la clínica. Hereda de Persona.

Atributos:

NumeroDocumento : string -> Documento de identidad del administrativo.

Métodos:

ToString() : string -> Devuelve la información básica del administrativo, incluyendo su número de documento.

Relaciones: Administra Paciente, Medico, Consulta y Pago dentro del sistema.

Clase: Pago

Descripción: Representa un pago realizado por un paciente para una consulta médica.

Atributos:

ID : int -> Identificador único del pago (autoasignado).

IdConsulta : int -> Identificador de la consulta asociada al pago.

FechaPago : DateTime -> Fecha en que se realizó el pago.

Monto : decimal -> Monto abonado.

Metodo : MetodoPago -> Forma de pago (Efectivo, Débito o Crédito).

Métodos:

ToString() : string -> Devuelve un resumen del pago incluyendo ID, consulta asociada, fecha, monto y método.

Relaciones:

Cada Pago está asociado a una única Consulta.

Enumeraciones:

MetodoPago -> Enum que define los posibles métodos de pago: Efectivo, Debito, Credito.

Clase: Consulta

Descripción: Representa un turno médico agendado entre un paciente y un médico en la clínica.

Atributos:

ID : int → Identificador único de la consulta (autoasignado).

IdPaciente : int → Identificador del paciente asociado.

IdMedico : int → Identificador del médico asignado.

FechaHora : DateTime → Fecha y hora del turno (bloques de 30 minutos).

Estado : EstadoConsulta → Estado del turno (Agendada, Realizada, Cancelada).

Métodos:

Cancelar() : void → Cancela la consulta si no ha sido realizada.

MarcarComoRealizada() : void → Marca la consulta como realizada si no está cancelada.

Reprogramar(DateTime nuevaFechaHora) : void → Cambia la fecha y hora respetando las reglas de anticipación y duración de 30 minutos.

ToString() : string → Devuelve un resumen de la consulta, incluyendo paciente, médico, fecha/hora y estado.

Relaciones:

Cada Consulta está asociada a un Paciente y a un Medico.

Puede estar asociada a un Pago.

Enumeraciones:

EstadoConsulta → Enum con los posibles estados de la consulta:

Agendada

Realizada

Cancelada

Clase: Clinica

Descripción: Clase principal que gestiona todas las entidades del sistema (pacientes, médicos, administrativos, consultas y pagos). Actúa como “controlador” de la clínica, centralizando la lógica de negocio.

Atributos:

pacientes : List<Paciente> → Lista de todos los pacientes registrados.

medicos : List<Medico> → Lista de todos los médicos de la clínica.

administrativos : List<Administrativo> → Lista de todos los usuarios administrativos.

consultas : List<Consulta> → Lista de todas las consultas médicas agendadas.

pagos : List<Pago> → Lista de todos los pagos realizados.

Métodos:

Gestión de pacientes:

AgregarPaciente(Paciente nuevoPaciente) : void → Agrega un paciente, verificando que no exista otro con el mismo documento.

ObtenerPacientes() : List<Paciente> → Devuelve los pacientes ordenados alfabéticamente por apellido.

Gestión de médicos:

AgregarMedico(Medico nuevoMedico) : void → Agrega un médico verificando matrícula única.

ObtenerMedicosPorEspecialidad(string especialidad) : List<Medico> → Devuelve médicos de una especialidad.

ObtenerMedicosPorDia(string dia) : List<Medico> → Devuelve médicos disponibles un día específico.

ObtenerTodosLosMedicos() : List<Medico> → Devuelve todos los médicos.

Gestión de administrativos:

AgregarAdministrativo(Administrativo nuevoAdministrativo) : void → Agrega un administrativo verificando documento único.

Login(string nombreUsuario, string contrasenia) : bool → Verifica credenciales de un administrativo.

VerificarContrasenia(string usuario, string contrasenia) : bool → Verifica la contraseña de un administrativo.

CambiarContrasenia(string usuario, string nuevaContrasenia) : void → Permite cambiar la contraseña de un administrativo.

Gestión de consultas:

AgendarConsulta(int idPaciente, int idMedico, DateTime fechaHora) : void → Agenda una consulta respetando reglas de negocio (máximo 6 turnos, 30 min, 24h de anticipación, no duplicar consulta paciente-médico el mismo día).

CancelarConsulta(int idConsulta) : void → Cancela una consulta existente.

ReprogramarConsulta(int idConsulta, DateTime nuevaFechaHora) : void → Reprograma una consulta verificando reglas de negocio.

ObtenerConsultasPorPaciente(int idPaciente) : List<Consulta> → Devuelve el historial de consultas de un paciente.

ObtenerConsultasPorMedico(int idMedico) : List<Consulta> → Devuelve todas las consultas de un médico.

Gestión de pagos:

RegistrarPago(int idConsulta, decimal monto, MetodoPago metodo, bool emitirComprobante = true) : void → Registra un pago para una consulta.

EmitirComprobante(int idPago) : void → Muestra en consola el comprobante de pago.

ObtenerPagosPorPaciente(int idPaciente) : List<Pago> → Devuelve todos los pagos realizados por un paciente.

Estadísticas y reportes:

ObtenerConsultasMasFrecuentesPorEspecialidad() : List<(string Especialidad, int Cantidad)> → Devuelve ranking de especialidades más consultadas.

ObtenerMedicosMasConsultados() : List<(Medico Medico, int Cantidad)> → Ranking de médicos con más consultas realizadas.

Relaciones:

Contiene y administra listas de Paciente, Medico, Administrativo, Consulta y Pago.

Interactúa con Consulta y Pago aplicando las reglas de negocio.

Clase: CargaDeDatosIniciales

Descripción: Clase estática encargada de inicializar la aplicación con datos de prueba. Permite cargar pacientes, médicos, administrativos, consultas y pagos al iniciar el sistema, facilitando las pruebas y demostraciones.

Atributos:

No posee atributos propios; utiliza la instancia de Clinica que se pasa como parámetro para agregar los datos.

Métodos:

Cargar(Clinica clinica) : void → Carga los datos iniciales:

5 médicos con distintas especialidades y horarios.

10 pacientes con datos personales y obra social.

2 administrativos.

5 consultas agendadas respetando reglas de negocio (30 min, 24h anticipación, máximo 6 turnos por día).

2 pagos asociados a consultas.

Relaciones:

Interactúa con Clinica para agregar objetos.

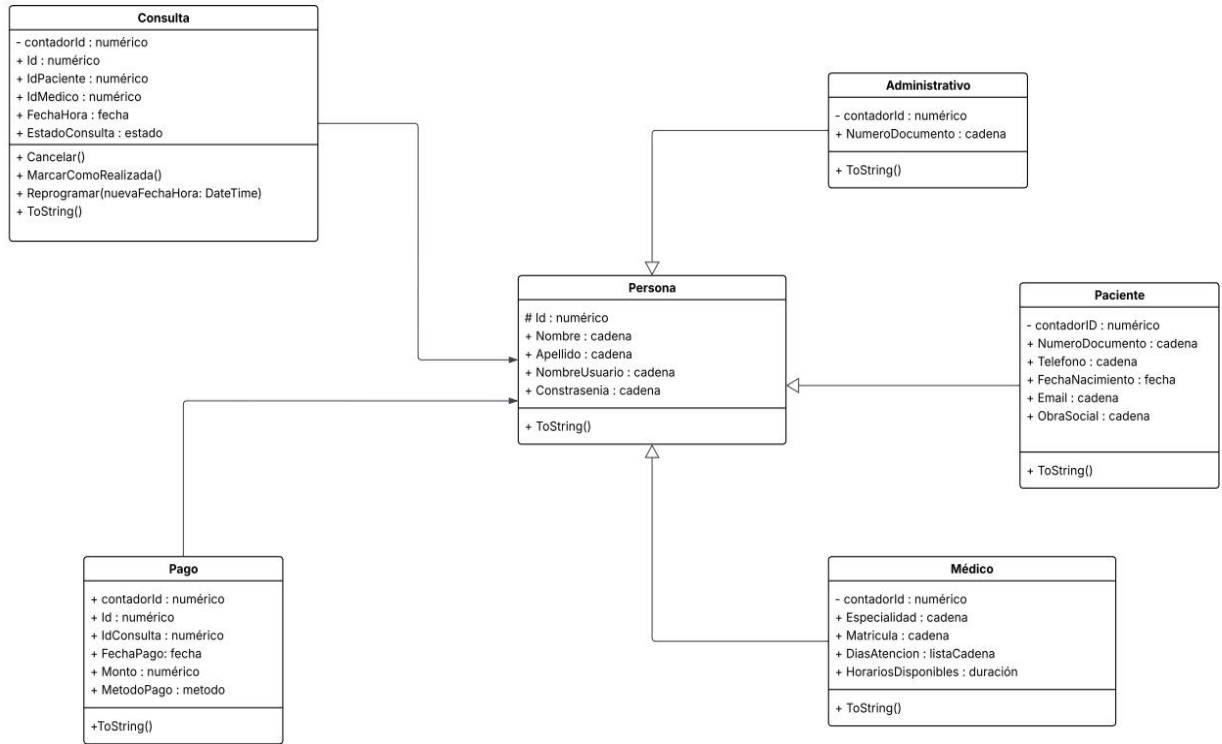
Utiliza las clases Paciente, Medico, Administrativo, Consulta y Pago.

Observaciones:

Es una clase auxiliar, no forma parte de la lógica de negocio central.

Facilita la carga automática de datos para pruebas o demostraciones.

Diagrama UML de Clases



Resultado de Pruebas

PRUEBA	ENTRADA	RESULTADO ESPERADO	RESULTADO OBTENIDO	ESTADO	OBSERVACIONES
Agendar consulta	Paciente ID 1, Médico ID 2, Fecha 30/10/2025 10:00	Consulta agendada correctamente	Consulta agendada correctamente	Aprobada	-----
Cancelar consulta	Consulta ID 1	Estado de consulta cambia a Cancelada	Estado de consulta cambia a Cancelada	Aprobada	-----
Registrar pago	Consulta ID 3, Monto 1500, Efectivo	Pago registrado y comprobante emitido	Pago registrado y comprobante emitido	Aprobada	-----
Reprogramar consulta	Consulta ID 2, Nueva fecha 31/10/2025 09:30	Fecha actualizada, estado Agendada	Error: Médico ya tiene turno en ese horario	Fallida	Conflicto de horario detectado

Juego de Datos Iniciales

Para facilitar las pruebas del sistema y demostrar su funcionamiento, se implementó una carga automática de datos al iniciar la aplicación, mediante la clase CargaDeDatosIniciales.

Esta clase crea registros predeterminados de médicos, pacientes, administrativos, consultas y pagos.

Los datos permiten verificar el correcto funcionamiento de las funcionalidades principales (agenda de consultas, registro de pagos, gestión de usuarios, etc.) sin necesidad de ingreso manual.

Cantidad cargada	Cantidad cargada	Descripción
Médicos	5	Cada médico tiene nombre, especialidad, días y horarios de atención.
Pacientes	10	Pacientes con diferentes coberturas médicas y datos personales.
Administrativos	2	Usuarios del sistema con permisos para gestionar consultas y pagos.
Consultas	5	Consultas agendadas con fechas futuras (válidas para pruebas).
Pagos	2	Pagos asociados a consultas, con distintos métodos (efectivo y débito).