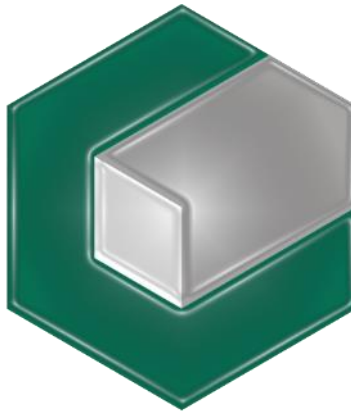


# Universidad Tecnológica de La Habana José Antonio Echeverría



## Trabajo de Curso

“Informatización del Consultorio Médico de la Familia”

### Tema 6

#### **Autores:**

Eduardo Alejandro González Martell

Eva Ercilia Vázquez García

#### **Tutor:**

PhD. Sonia Pérez Lovelle

**Asignatura:** Diseño y Programación Orientada a Objetos

**Año Académico:** 1ro.

**Carrera:** Ing. Informática

La Habana, 2022

## Resumen

Desde la inauguración del Programa del Médico y la Enfermera de la Familia en Cuba, se han fortalecido y desarrollado las bases por las cuales fue creado inicialmente. La instauración y expansión de los consultorios médicos de la familia (CMF) a lo largo de Cuba, ha permitido que la Atención Primaria de Salud llegara a un mayor número de habitantes. A pesar de esto, los sistemas manuales actualmente instaurados en los CMF se convierten en un obstáculo para su evolución. Por ello se propone la informatización de los consultorios médicos de la familia, a través de una propuesta de aplicación de escritorio desarrollada en Java, bajo el paradigma de la Programación Orientada a Objetos (POO). Dicha solución informática permitirá la informatización de los datos referentes a los pacientes; la digitalización del sistema de hojas de cargo; la automatización del proceso de indicación y realización de los análisis; y otras funcionalidades que facilitarán y agilizarán el trabajo de los empleados médicos del CMF.

**Palabras Claves:** CMF, Médico, Enfermera, Atención Primaria de Salud, POO.

# Índice de Contenido

<b>Introducción.....</b>	<b>1</b>
<b>Capítulo 1 .....</b>	<b>3</b>
1.1 Análisis del problema .....	3
1.1.1 Consideraciones Generales.....	3
1.1.2 Posibles Validaciones .....	4
1.1.3 Tipos de Usuario .....	5
1.2 Plan de trabajo .....	6
1.3 Tarjetas CRC .....	8
<b>Capítulo 2 .....</b>	<b>13</b>
2.1 Documentación de las clases identificadas.....	13
2.1.1 Listado de Clases Identificadas.....	13
2.1.2 Diagrama de Clases UML .....	14
2.2 Fundamentación de los tipos de listas implementadas .....	14
2.3 Organización del proyecto.....	14
2.4 Validaciones y tratamiento de errores y excepciones .....	17
2.5 Consideraciones y Documentación de la Interfaz .....	19
2.5.1 Consideraciones Generales de la Interfaz .....	19
2.5.2 Documentación de la Interfaz.....	21
2.5.3 Principios, Reglas de Oro y Patrones de diseño de la Interfaz .....	34
2.6 Documentación de los reportes.....	36
2.6.1 Reporte de pacientes dada enfermedad .....	36
2.6.2 Reporte de análisis faltantes de un paciente dado.....	37
2.6.3 Reporte de pacientes embarazadas .....	38
2.6.4 Reporte de pacientes en riesgo (funcionalidad embebida) .....	38
2.6.5 Reporte de pacientes sin visitas desde una fecha dada .....	39
2.6.6 Reporte de pacientes vacunados dado una vacuna .....	40
2.7 Descripción de los patrones empleados.....	40
2.8 Pruebas Realizadas .....	42
2.8.1 Diseño de Pruebas de Caja Blanca .....	42

2.8.2 Diseño de Pruebas de Caja Negra.....	45
2.8.3 Resultado de las pruebas.....	60
2.9 Aspectos Novedosos.....	61
<b>Conclusiones</b> .....	64
<b>Recomendaciones</b> .....	65
<b>Bibliografía</b> .....	66
<b>Anexos</b> .....	67

## Introducción

**Problemática:** Se desea automatizar la gestión de un consultorio médico de la familia (CMF). De cada CMF se conoce su número que lo identifica dentro del policlínico, el nombre del policlínico, el nombre del director del policlínico, el nombre y apellidos del médico, su número en el registro médico, número de identidad, y fecha de inscripción en el CMF. Además, los datos de la enfermera: nombre y apellidos, número de identidad, si tiene la licenciatura, años de experiencia y fecha en la que comenzó a laborar en el CMF.

Para cada paciente del consultorio se conoce su número de historia clínica, nombre y apellidos, edad, sexo, enfermedades crónicas, vacunación. En el caso de las mujeres, la fecha de la última prueba citológica, y si está embarazada. Además, el médico debe notificar a los pacientes que están en riesgo. Para saber cuáles son los pacientes en riesgo, estos deben tener más de 3 enfermedades crónicas, y en caso de las mujeres, también es de riesgo si lleva más de 3 años sin hacerse la prueba citológica.

Cada vez que un paciente asiste al CMF, debe quedar registrado en la hoja de cargos del día, en la que se consigna, nombre y apellidos, edad, sexo, dirección y el diagnóstico. Adicionalmente, en la historia clínica correspondiente, según el número, se consigna la fecha, de la visita, el diagnóstico, tratamiento, indicaciones de complementarios (análisis), y si se remitió a alguna especialidad.

Diariamente (de lunes a viernes) la enfermera recoge en el policlínico los resultados de los análisis, de los rayos X, ultrasonidos, etc. y además de registrar los resultados en sus respectivas historias clínicas, actualiza el registro general y el histórico del consultorio. Para ello, estos análisis cuentan con los datos necesarios que permiten identificar el paciente y la historia clínica que le corresponde.

**Objetivo:** Dar solución a la problemática planteada con la creación de un *software* que le permita al personal médico del consultorio médico de la familia (médico y enfermera) gestionar toda la actividad diaria en el CMF, de forma tal que dicho proceso sea más rápido y directo comparado con el sistema de trabajo manual vigente actualmente.

Para el desarrollo del presente trabajo se emplearon las facilidades otorgadas por el paradigma de la programación orientada a objetos (POO), empleándola en Java y utilizando el entorno de desarrollo (IDE) Eclipse versión 4.3.0. Para el desarrollo del *software* se empleó como biblioteca principal el jdk 1.8.0\_251, configurando el proyecto con nivel de compilación 1.6; además se utilizó la biblioteca gráfica Java Swing, y se emplearon apis externas: FlatLaf, JCalendar, JUnit 4 y una api desarrollada internamente con componentes visuales. Para el modelado del diagrama de clases UML, se empleó la herramienta en línea **lucidchart.com**.

El presente documento está estructurado en dos capítulos y doce epígrafes principales (tres pertenecientes al primer capítulo y los nueve restantes al

segundo). En el primer capítulo se planteará el proceso de análisis de la problemática y las consideraciones realizadas al respecto tras su investigación. En el segundo capítulo se describirá el proceso de desarrollo del *software*, explicando tanto la lógica utilizada para la solución informática, como la interfaz visual generada para la interacción con el programa creado. Luego se otorgarán las conclusiones alcanzadas tras la finalización del presente proyecto, las recomendaciones y la bibliografía empleada.

# Capítulo 1

El presente capítulo abordará sobre el proceso de análisis de la problemática dada, planteando las consideraciones realizadas al respecto. Además se presentará el plan de trabajo seguido por el equipo de desarrollo para la confección del proyecto. Por otra parte se enumeraran las tarjetas CRC, fruto del análisis de las clases presentes la problemática para llegar a una solución.

## 1.1 Análisis del problema

### 1.1.1 Consideraciones Generales

Para la confección del presente trabajo de curso se realizó una investigación previa acerca del funcionamiento interno de los consultorios médicos de la familia, lo cual fue concretado con la visita a uno de ellos. Tras dicha visita, y el análisis de la problemática dada se determinaron una serie de consideraciones:

- ❖ **Pacientes:** El 4 de enero de 1985 se inauguró oficialmente en Cuba el plan del médico y la enfermera de la familia [1]. Dicho proyecto fue también conocido como “plan médico de las 120 familias”. Debido a esto se llegó a la conclusión de que la cantidad de pacientes de un consultorio debe estar en el rango de los 120 (familias de 1 persona) y 1200 (familias de 10 personas). Se determinó como cota máxima de la edad de los pacientes 130 años, debido a los casos extremos de longevidad que puedan existir.
- ❖ **Pacientes Femeninas:** La prueba citológica es un examen médico para determinar si existe cáncer del cuello del útero. La misma se puede comenzar a realizar a partir de que se comienzan las relaciones sexuales, y no son necesarias después de pasar los 70 años. Debido a esto se estableció que la edad mínima de pruebas es a los 10 años y la máxima los 70 años. Además se definió que la edad mínima recomendada para las pruebas citológicas sean los 25 años y la máxima 65. Por otra parte, se definió que las pacientes pueden resultar embarazadas desde los 10 años (comienzo de la menstruación) hasta los 70 (caso extremo).
- ❖ **Trabajador Médico:** Tanto el médico como la enfermera puede ejercer su trabajo en el consultorio médico de la familia desde su graduación, hasta su jubilación. Por tanto se determinó que la edad mínima del trabajador médico sean los 22 años y la máxima los 80 años (caso extremo).
- ❖ **Médico:** El médico tras graduarse tiene un número de registro médico. Este es único para cada médico y consiste en una letra (por lo general la M) y una serie de dígitos en serie. Se definió como formato M-DDDDDD, donde D es sustituido por un dígito.
- ❖ **Tiempo de obsolescencia:** Se determinó que los datos con vigencia de más de 10 años fueran archivados, es decir, no fueran mostrados, para

así no sobrecargar la muestra de información obsoleta. Por otra parte, se decidió que en la funcionalidad de la enfermera de introducir los resultados de los análisis, solamente se mostraran aquellos análisis indicados con un máximo de obsolescencia de 1 mes, puesto que la mayoría de dichos análisis vencen en ese rango de tiempo.

- ❖ **Formato de número de historia clínica:** Para almacenar e identificar las historias clínicas, se emplean los números de historia clínica. Existen muchos formatos aplicables a los mismos. Se decidió emplear como formato HC-NN-CCCC-FFF, donde N representa el número del consultorio, C el número de la casa según la serialización definida por el consultorio, y F el número de habitante en la casa.
- ❖ **Rango de caracteres:** Existen tipos de datos que deben presentar un mínimo y un máximo de caracteres para que sean válidos. Tal es el caso de los diagnósticos, los tratamientos y los resultados de los análisis. Se decidió que tuvieran un tamaño mínimo de 3 caracteres y uno máximo de 400 caracteres.
- ❖ **Pacientes en Riesgo:** Para la solución de la problemática es necesario implementar un sistema para determinar los pacientes en riesgo. Un paciente se encuentra en riesgo si presenta más de 3 enfermedades crónicas. En el caso de las pacientes femeninas, además se encuentra en riesgo si no se ha realizado una prueba citológica en los últimos 3 años, teniendo la edad recomendada.

Para una mejor adaptabilidad de la aplicación todas estas consideraciones fueron definidas en una sección del programa, de manera tal que en el futuro, puedan ser modificadas fácilmente.

### 1.1.2 Posibles Validaciones

Luego de investigar sobre la problemática dada, se determinaron un cierto número de posibles validaciones a tener en cuenta para el desarrollo del proyecto de *software*:

- ❖ **Carnet de Identidad:** Es necesario comprobar la validez del carnet de identidad de los pacientes. Para ello se debe comprobar la validez de la fecha de nacimiento, referente a los 7 primeros dígitos del carnet, de forma tal que los 2 primeros se refieren al año, el otro par al mes, el siguiente al día, y el séptimo dígito al siglo: 0-5 siglo XX, 6-8 siglo XI, 9 siglo XIX. Los dígitos octavo y noveno son números generados aleatorios que no son necesarios su validación. El décimo dígito indica el sexo: par sexo masculino, impar sexo femenino. El oncenavo dígito es un dígito de control generado por un algoritmo matemático. Dicho algoritmo no fue posible de obtener para su utilización en el sistema para validar esta información, por lo que se decidió solo validar los dígitos anteriores.



Además de la validez del formato, es necesario comprobar la no existencia del carnet en el sistema.

❖ **Datos seleccionables:** Son necesarios un conjunto de datos previamente definidos para la puesta en práctica del problema en un sistema virtual real. Los más destacables son:

- **Vacunaciones:** Los pacientes deben presentar un esquema de vacunaciones. Dicho esquema puede ser previamente definido para evitar la introducción de errores, así como reducir la carga de trabajo del personal médico. Además, permitirá validar el rango de fechas en que se aplicó cada vacuna, puesto que cada una presenta un rango válido de edad [2].
- **Enfermedades Crónicas:** Los pacientes pueden presentar un conjunto de enfermedades crónicas, las cuales pueden ser previamente definidas [3].
- **Especialidades Remitidas:** El médico puede remitir al paciente a una serie de especialidades, las cuales pueden ser previamente definidas [4].
- **Tipos de Análisis:** El médico puede indicarle al paciente una serie de análisis a realizar, los cuales pueden ser previamente definidos. Luego de la visita investigativa a los consultorios médicos de la familia se determinaron los siguientes tipos de análisis principales: hemograma, orina, heces fecales, serología, VIH, rayos-X.
- **Direcciones:** Los pacientes presentan una dirección, por lo que se puede predefinir un sistema de direcciones, puesto que se trata de un área de salud específica.

Las validaciones más concretas se omitieron, puesto que son explicadas en detalle más adelante.

### 1.1.3 Tipos de Usuario

Luego del análisis general de la problemática, se determinaron 2 tipos de usuario del sistema: el médico y la enfermera. Por tanto se hace necesario la implementación de un sistema de autenticación para, además de aumentar la seguridad del sistema, permitirá modificar la estructura interna del programa para mostrar aquellas opciones propias de cada tipo de usuario. Ambos tipos de usuario tendrán acceso a la misma información y podrán modificar los datos que requieran. La diferencia se encuentra en la funcionalidad primaria de cada uno; en el caso del médico, agregar nuevas visitas diarias; y en el caso de la enfermera, agregar los resultados de los análisis indicados por el doctor en las visitas.

## 1.2 Plan de trabajo

En la **Tabla 1** se muestra la propuesta del plan de trabajo a seguir para la correcta solución del problema del trabajo de curso. Este plan puede variar a lo largo del proceso de desarrollo, sufriendo cambios debido a factores externos e internos que lo propicien.

**Tabla 1:** Propuesta del plan de trabajo.

No.	Tarea	Fecha	Responsable	Participantes
Inicio de la <b>Fase 0</b>				
1	Estudiar enunciado	Semana 3	Todos	Todos
2	Estudiar software similares	Semana 3	Todos	Todos
3	Investigar sobre el funcionamiento de los consultorios médicos de la familia	Semana 3	Todos	Todos
Inicio de la <b>Fase 1</b>				
4	Hacer lista de funcionalidades	Semana 4	Todos	Todos
Inicio de la <b>Fase 2</b>				
5	Elaborar tarjetas CRC	Semana 4	Eva Vázquez	Eduardo González
6	Hacer diagrama de clases	Semana 4	Eduardo González	Eva Vázquez
Inicio de la <b>Fase 3</b>				
7	Definir el núcleo del software	Semana 6	Todos	Todos
8	Distribuir funcionalidades entre los miembros del equipo	Semana 6	Todos	Todos
9	Para cada funcionalidad diseñar un boceto de las pantallas asociadas	Semana 6	Todos	Todos
10	Preparar y Entregar Primer Corte	Semana 6	Todos	Todos
Inicio de la <b>Fase 4</b>				
11	Programar prototipo del software	Semana 7	Todos	Todos
12	Implementar clase Persona	Semana 6-10	Eva Vázquez	Eduardo González
13	Implementar clase Medico	Semana 6-10	Eva Vázquez	Eduardo González
14	Implementar clase Enfermera	Semana 6-10	Eva Vázquez	Eduardo González
15	Implementar clase Paciente	Semana 6-10	Eduardo González	Eva Vázquez
16	Implementar clase PacienteFemenina	Semana 6-10	Eduardo González	Eva Vázquez

**Tabla 1:** Propuesta del plan de trabajo.

17	Implementar clase Direccion	Semana 6-10	Eduardo González	Eva Vázquez
18	Implementar clase Vacuna	Semana 6-10	Eduardo González	Eva Vázquez
19	Implementar clase HistoriaClinica	Semana 6-10	Eva Vázquez	Eduardo González
20	Implementar clase Visita	Semana 6-10	Eva Vázquez	Eduardo González
21	Implementar clase Analisis	Semana 6-10	Eduardo González	Eva Vázquez
22	Implementar clase HojaDeCargosDia	Semana 6-10	Eva Vázquez	Eduardo González
23	Implementar clase VisitaHojaDeCargos	Semana 6-10	Eva Vázquez	Eduardo González
24	Implementar clase CMF	Semana 6-10	Eduardo González	Eva Vázquez
25	Implementar Clases auxiliares	Semana 6-10	Todos	
26	Realizar Pruebas	Semana 10	Todos	Todos
27	Integrar	Semana 10	Todos	Todos
28	Preparar y Entregar Segundo Corte	Semana 11	Todos	Todos
Inicio de la <b>Fase 5</b>				
29	Refinar la interfaz GUI	Semana 11-12	Todos	Todos
30	Implementar piscina de datos	Semana 12	Todos	Todos
31	Revisar el código fuente en busca de posibles fallas	Semana 12	Todos	Todos
32	Redactar versión final del documento	Semana 13	Todos	Todos
33	Entregar documento	Semana 14	Todos	Todos
34	Preparar Defensa	Semana 15	Todos	Todos
35	Defensa	Semana 16	Todos	Todos

### 1.3 Tarjetas CRC

Tras la identificación de las principales clases del problema, a continuación se detallan sus respectivas tarjetas CRC:

**Tabla 2:** Tarjeta CRC de la superclase abstracta Persona.

<i>Persona</i>	Colaboradores
<ul style="list-style-type: none"><li>.Dar a conocer el nombre completo de la persona y por separado (nombre, primer apellido y segundo apellido).</li><li>.Dar a conocer el carnet de identidad de la persona.</li><li>.Dar a conocer la fecha de nacimiento de la persona.</li><li>.Dar a conocer la edad de la persona de la persona</li></ul>	

**Tabla 2:** Tarjeta CRC de la subclase de Persona, Medico.

Medico	Colaboradores
<ul style="list-style-type: none"><li>.Dar a conocer la información básica de una persona: el nombre completo y por separado (nombre, primer apellido y segundo apellido), el carnet de identidad, edad, fecha de nacimiento y la edad.</li><li>.Dar a conocer la fecha de inscripción en el consultorio médico de la familia.</li><li>.Dar a conocer el número de registro médico.</li></ul>	

**Tabla 3:** Tarjeta CRC de la clase Direccion.

Direccion	Colaboradores
<ul style="list-style-type: none"><li>.Dar a conocer la calle principal de la vivienda.</li><li>.Dar a conocer el número de la casa.</li><li>.Dar a conocer la información extra referente a la vivienda.</li><li>.Dar a conocer la dirección completa de la vivienda.</li></ul>	

**Tabla 5:** Tarjeta CRC de la subclase de Persona, Paciente.

Paciente	Colaboradores
<p>.Dar a conocer la información básica de una persona: el nombre completo y por separado (nombre, primer apellido y segundo apellido), el carnet de identidad, edad, fecha de nacimiento y la edad.</p> <p>.Dar a conocer la lista de enfermedades crónicas.</p> <p>.Dar a conocer la lista de vacunaciones.</p> <p>.Dar a conocer el sexo del paciente</p> <p>.Dar a conocer la dirección del paciente</p> <p>.Dar a conocer la historia clínica del paciente.</p> <p>.Dar a conocer si el paciente se encuentra en riesgo.</p>	<p>Dirección</p> <p>Vacuna</p> <p>Historia Clínica</p>

**Tabla 6:** Tarjeta CRC de la subclase de Persona, Enfermera.

Enfermera	Colaboradores
<p>.Dar a conocer la información básica de una persona: el nombre completo y por separado (nombre, primer apellido y segundo apellido), el carnet de identidad, edad, fecha de nacimiento y la edad.</p> <p>.Dar a conocer si tiene licenciatura.</p> <p>.Dar a conocer la fecha de inscripción en el consultorio médico de la familia.</p> <p>.Dar a conocer la fecha del comienzo laboral.</p> <p>.Dar a conocer la cantidad de años de experiencia.</p>	

**Tabla 7:** Tarjeta CRC de la clase Vacuna.

Vacuna	Colaboradores
<p>.Dar a conocer la fecha de la vacunación.</p> <p>.Dar a conocer el nombre de la vacuna.</p>	

**Tabla 8:** Tarjeta CRC de la clase HistoriaClinica.

HistoriaClinica	Colaboradores
.Dar a conocer el número de historia clínica. .Dar a conocer el listado de análisis. .Dar a conocer el listado de visitas.	Visita  Análisis

**Tabla 9:** Tarjeta CRC de la clase Visita.

Visita	Colaboradores
.Dar a conocer la fecha de la visita del paciente. .Dar a conocer el diagnóstico. .Dar a conocer el tratamiento. .Dar a conocer los análisis indicados. .Dar a conocer las especialidades remitidas. .Dar a conocer si tuvo análisis indicados. .Dar a conocer si tuvo especialidades remitidas.	

**Tabla 10:** Tarjeta CRC de la clase Analisis.

Analisis	Colaboradores
.Dar a conocer la fecha del análisis. .Dar a conocer el tipo de análisis. .Dar a conocer el resultado de los análisis.	

**Tabla 11:** Tarjeta CRC de la clase HojaDeCargosDia.

HojaDeCargosDia	Colaboradores
.Dar a conocer la fecha de la hoja de cargos del día. .Dar a conocer la lista de las visitas almacenadas en la hoja de cargos del día.	

**Tabla 12:** Tarjeta CRC de la subclase de Paciente, PacienteFemenina.

PacienteFemenina	Colaboradores
<p>.Dar a conocer la información básica de una persona: el nombre completo y por separado (nombre, primer apellido y segundo apellido), el carnet de identidad, edad, fecha de nacimiento y la edad.</p> <p>.Dar a conocer la información básica de un paciente: la lista de enfermedades crónicas, la lista de vacunaciones, el sexo del paciente, la dirección del paciente y la historia clínica del paciente.</p> <p>.Dar a conocer la fecha de la última prueba citológica.</p> <p>.Dar a conocer si está embarazada.</p> <p>.Dar a conocer si está en riesgo.</p>	

**Tabla 13:** Tarjeta CRC de la clase VisitaHojaDeCargos.

VisitaHojaDeCargos	Colaboradores
<p>.Dar a conocer el paciente al que se refiere dicha visita.</p> <p>.Dar a conocer el diagnóstico.</p>	<p>Paciente HojaDeCargosDia</p>

**Tabla 14:** Tarjeta CRC de la clase CMF (Consultorio Médico de la Familia).

CMF	Colaboradores
<p>.Dar a conocer el número del consultorio.</p> <p>.Dar a conocer el nombre del policlínico al que pertenece.</p> <p>.Dar a conocer el nombre del director del policlínico.</p> <p>.Dar a conocer el médico del consultorio.</p> <p>.Dar a conocer la enfermera del consultorio.</p> <p>.Dar a conocer el listado de las hojas de cargos diarias.</p> <p>.Dar a conocer el listado de pacientes.</p> <p>.Dar a conocer el listado de pacientes con una determinada enfermedad.</p> <p>.Buscar y devolver los pacientes que cumplan con ciertas condiciones referidas en DatosBuscadorPaciente.</p> <p>.Dar a conocer el listado de análisis faltantes indicados al paciente, a partir de una fecha.</p> <p>.Dar a conocer el listado de pacientes embarazadas.</p> <p>.Dar a conocer el listado de pacientes que no han realizado visitas al consultorio, a partir de una fecha.</p> <p>.Dar a conocer el listado de pacientes vacunados con una vacuna dada.</p>	<p>Medico</p> <p>Enfermera</p> <p>Paciente</p> <p>HojaDeCargosDia</p> <p>DatosBuscadorPaciente</p> <p>AnalisisIndicados</p>



## Capítulo 2

El presente capítulo abordará sobre el proceso de desarrollo del *software* para dar solución a la problemática dada. Se expondrá la documentación de las clases identificadas tras el análisis de las problemáticas; los tipos de listas implementadas; la organización final del proyecto en capas lógicas; documentación de la interfaz gráfica implementada; explicación de los reportes de la aplicación; descripción de los patrones empleados; las pruebas realizadas; además de algunas consideraciones de interés para comprender el trabajo realizado.

### 2.1 Documentación de las clases identificadas

#### 2.1.1 Listado de Clases Identificadas

Tras el análisis del problema del trabajo de curso, se identificaron las siguientes clases necesarias para el modelado del programa en un diagrama de clases *UML*:

- Persona: Superclase abstracta con los datos generales de una persona.
- Medico: Subclase de Persona que contiene los datos específicos de un médico.
- Enfermera: Subclase de Persona que contiene la información particular de una enfermera.
- Paciente: Subclase de Persona que almacena la información de un paciente.
- Paciente Femenina: Subclase de Paciente que contiene los datos específicos de una paciente de sexo femenino, con las particularidades que ello conlleva.
- Vacuna: Clase que almacena la información de una vacunación.
- Direccion: Clase que contiene todo lo referente a las direcciones particulares.
- HistoriaClinica: Clase que almacena lo referente a las visitas al consultorio Médico de la Familia (CMF) y los análisis del paciente.
- Visita: Clase que contiene los datos de la visita al CMF por parte del paciente.
- Analisis: Clase que almacena la información referente a los resultados del análisis indicado al paciente.
- CMF: Clase controladora que almacena el médico y la enfermera vinculados al mismo, el listado de pacientes del consultorio y el listado de la hoja de cargos por día.
- HojaDeCargosDia: Clase que almacena las informaciones de las visitas de los pacientes al CMF de un día específico.
- VisitaHojaDeCargos: Clase que contiene las anotaciones hechas por el médico tras la visita del paciente al CMF



- Main
- **nucleo:** Este paquete contiene las clases principales para la modelación del proyecto. Se encuentran las siguientes clases:
  - Analisis
  - CMF
  - Direccion
  - Enfermera
  - HistoriaClinica
  - HojaDeCargosDia
  - Medico
  - Paciente
  - PacienteFemenina
  - Persona (abstracta)
  - Vacuna
  - Visita
  - VisitaHojaDeCargos
- **utilidades:** Este paquete contiene las clases auxiliares que realizan operaciones que se necesitan tanto en la lógica como en la interfaz, incluyéndose las validaciones principales del programa y la piscina de datos para la generación de datos automáticos. Se encuentran las siguientes clases:
  - Auxiliares
  - AuxiliaresInterfazGrafica
  - BuscadorPacientes
  - Comparadores
  - DialogosAuxiliares
  - PiscinaDatos
  - Validaciones
  - ValidacionesDefiniciones
- **clases auxiliares:** Este paquete contiene las clases auxiliares. Se encuentran las siguientes clases:
  - AnalisisIndicados
  - DireccionPiscinaDatos
  - TipoCuentaUsuario (enum)
  - Usuario
  - VacunacionPiscinaDatos
- **modelos:** Este paquete contiene las clases que crean componentes genéricos que serán utilizados en las listas. Se encuentran las siguientes clases:
  - AnalisisFaltantesTableModel
  - AnalisisIndicadosComboBoxModel
  - AnalisisIndicadosTableModel
  - AnalisisTableModel
  - CallePrincipalComboBoxModel
  - EmbarazadaTableModel
  - EnfermedadesCronicasComboBoxModel
  - EnfermedadesCronicasTableModel
  - EspecialidadesRemitidasComboBoxModel
  - EspecialidadesRemitidasTableModel

- HojaCargosTableModel
- ModeloPrincipalComboBoxModel(abstracta)
- ModeloPrincipalTableModel (abstracta)
- MultilineaCellRendererEditor
- NoCasaComboBoxModel
- PacienteAnálisisFaltantesTableModel
- PacienteComboBoxModel
- PacienteTableModel
- VacunacionesTableModel
- VacunaComboBoxModel
- VisitaTableModel
- **pruebas:** Este paquete contiene las clases relacionadas con las pruebas realizadas en JUnit. Se encuentran las clases:
  - CMFTestCase
  - PacienteFemeninaTestCase
  - PacienteTestCase
- **definiciones:** Este paquete contiene las clases donde se definen constantes estáticas que facilitan la modificación de valores comunes que se utilizan en distintas funcionalidades que pueden estar en lógica o interfaz, además de los mensajes usados en el tratamiento de errores. Se encuentran las siguientes clases:
  - Definiciones
  - DefinicionesInterfazGrafica
  - Errores
  - ErroresInterfazGrafica

Dentro de la capa de la interfaz se encuentran los paquetes que están relacionados con la visualización de las pantallas y los archivos multimedia que se emplean. A continuación, se presenta el nombre de cada paquete, su descripción y sus clases correspondientes:

- **interfaz\_grafica:** Este paquete contiene las clases que manejan las vistas del programa y solo accede a los datos y funcionalidades lógicas a través de la clase controladora de la lógica (CMF). Se encuentran las siguientes clases:
  - AgregarAnálisis
  - AgregarPaciente
  - AgregarVisita
  - AppPrincipal
  - Autenticacion
  - EditarPaciente
  - PantallaCarga
  - VerHistoriaClinica
  - VerPaciente
- **iconos:** este paquete contiene los iconos utilizados.
- **imagenes:** este paquete contiene las imágenes utilizadas.

## 2.4 Validaciones y tratamiento de errores y excepciones

Se utilizaron diferentes mecanismos de validación para garantizar la integridad y validez de los datos que introduce el usuario. Se logra a través de componentes de la interfaz o en la capa lógica con el empleo del encapsulamiento y métodos de validación de la clase Validaciones. Cuando se introducen datos incorrectos, el sistema permite mostrar mensajes de error para que el usuario pueda rectificarlos. A continuación, se presentan las validaciones realizadas en las clases principales del programa:

- Persona:
  - nombre, primer apellido y segundo apellido: No puede ser todo espacio, solo se pueden introducir letras. Hay un tamaño mínimo y uno máximo que se encuentran en Definiciones.
  - carnet de identidad: Debe ser un número de 11 dígitos, la fecha de nacimiento debe estar correcta (año, mes y día válidos y debe ser anterior a la fecha actual) y la edad de la persona debe estar en el rango de edad que se encuentra en Definiciones. Además se comprueba que no exista en el sistema.
- Médico:
  - número en el registro médico: Debe ser alfanumérico, sin espacio, con un tamaño mínimo y uno máximo que se encuentran en Definiciones.
  - fecha de inscripción: Debe estar en un rango entre la edad mínima en que puede comenzar a laborar un trabajador médico y la fecha actual.
- Enfermera:
  - comienzo laboral: Debe estar en un rango entre la edad mínima en que puede comenzar a laborar un trabajador médico y la fecha actual.
  - fecha de inscripción: Debe estar en un rango entre el comienzo laboral y la fecha actual.
- Paciente:
  - añadir enfermedad crónica: Debe estar en la lista de enfermedades crónicas de la piscina de datos y no puede estar repetida en el listado de enfermedades crónicas del paciente.
  - añadir vacunación: No puede estar repetida en el listado de vacunación del paciente.
  - dirección: La calle principal y el número de la vivienda deben estar en la piscina de datos.
- Vacuna:
  - fecha de la vacunación: Debe ser anterior a la fecha actual.
  - nombre de la vacuna: Debe estar en la lista de nombres de vacunas de la piscina de datos.

- Historia clínica:
  - número de historia clínica: Es generado automáticamente en el constructor de Paciente con el formato HC-(número del consultorio)-(número de la vivienda según listado de consultorio)-(número generado que indica el número del paciente en su dirección).
  - añadir visita: No puede haber análisis indicados en esta visita que estén repetidos en otras visitas del paciente en ese mismo día.
- Visita:
  - fecha: Se crea con la fecha actual.
  - tratamiento: No puede ser todo espacio y hay un tamaño mínimo y uno máximo que se encuentran en Definiciones.
  - tipo de análisis indicados: Debe estar en la lista de análisis de la piscina de datos y no puede haber análisis repetidos.
  - especialidades remitidas: Debe estar en la lista de especialidades de la piscina de datos y no puede haber especialidades remitidas repetidas.
- Análisis:
  - fecha de indicación: Se inicializa con la fecha actual, a la misma par que su visita relacionada.
  - tipo de análisis: Debe estar en la lista de análisis de la piscina de datos.
  - fecha del análisis: Se inicializa con la fecha actual una vez que se añade el resultado del análisis.
  - resultado del análisis: Se guarda cuando no tiene un resultado aún almacenado, y cuando no sea todo espacio y tenga un tamaño mínimo y uno máximo que se encuentran en Definiciones.
- VisitaHojaDeCargos:
  - diagnóstico: No puede ser todo espacio y hay un tamaño mínimo y uno máximo que se encuentran en Definiciones.
- Consultorio Médico de la Familia (CMF):
  - número del CMF: Debe ser un número positivo.
  - nombre del policlínico: No puede ser todo espacio.
  - nombre director del policlínico: No puede ser todo espacio.
  - añadir paciente: No puede estar repetido en el listado de pacientes.

**Nota:** En la mayoría de los casos se validará que los *Strings* no estén vacíos y que los objetos no sean null. Se emplearán usualmente métodos de la clase Validaciones para realizar todas estas comprobaciones y se lanzarán excepciones con mensajes documentados en las clases Errores y ErroresInterfazGrafica en dependencia de la situación. Además de las validaciones aquí mencionadas, se realizan otras con respecto a los datos definidos en la PiscinaDatos.

El sistema de la interfaz gráfica permite que una vez guardados los datos, los mismos estén validados correctamente. Se disminuye el margen de error al utilizar componentes de interfaz gráfica como listas desplegables para seleccionar un elemento, calendario para seleccionar fecha, campos rellenos automáticamente a partir de otra información introducida (el caso de que se puede rellenar sexo y edad si se tiene el carnet de identidad completo). Este mecanismo de tratamiento de errores se pone de manifiesto de la vista AgregarPaciente del paquete interfaz\_grafica, donde se puede comprobar el “motor” de validación del carnet de identidad en tiempo real, el cual, si el mismo es correcto (incluyendo la validación que no existe en el listado), extrae y muestra toda la información posible.

Solamente se utilizó el tratamiento de excepciones con los bloques de código try/catch, en el sistema de seguridad de la autenticación, y en algunas validaciones. No fue necesario el manejo de las excepciones introducidas en las clases principales, puesto que a nivel de interfaz se validan, y una vez llega la información al núcleo, la misma es correcta. Se decidió esta vía, puesto que permitiría mostrar el error del usuario, validando varios campos al mismo tiempo, comportamiento no posible con el tratamiento de las excepciones, puesto que solamente se conocería uno de los errores cometidos.

## 2.5 Consideraciones y Documentación de la Interfaz

### 2.5.1 Consideraciones Generales de la Interfaz

De forma general, se tuvieron las siguientes consideraciones a la hora de diseñar la interfaz gráfica:

- **Colores:** Se empleó un esquema de colores de verdes de varias tonalidades con fondo blanco, como se aprecia en el **Anexo 1**. El motivo de esta selección es que el verde es el color representativo de la medicina, además de ser un color relajante y símbolo de la esperanza.
- **Fuentes:** Para una mayor homogeneidad se empleó como fuente, en todo el programa, **Roboto**, en todas sus variantes (negrita, cursiva y plano). Se empleó la misma puesto que es muy empleada en las aplicaciones hoy en día por su simplicidad, facilidad de lectura y vistosidad. El tamaño de la fuente fue variable, en dependencia de la situación: para los encabezados se empleó un tamaño de entre 24 y 20 puntos; para la representación de la información en tablas se empleó un tamaño de 13 puntos; y para la representación de la información en general, se utilizó un tamaño de entre 15 y 18 puntos. El color de la fuente es negra,

exceptuando en casos que, por ejemplo, es roja porque hay un error en un campo específico.

- **Tablas:** Para el mostrado de la información principal se emplearon las tablas de información. Las mismas fueron programadas para que se pudieran reordenar en dependencia de la columna seleccionada por el usuario, tanto de manera ascendente como descendente. Algunas tablas funcionan como un directorio de pacientes, como es el caso de la de los listados de pacientes, en las cuales, con solo un doble click en el paciente deseado, se podrá acceder a su información e incluso editarla según sea el caso. Para la solución informática de este problema se decidió que las celdas no fueran modificables, puesto que esto podría conllevar a errores en el futuro. Como aspecto novedoso a tener en cuenta, se desarrolló una celda modificada con un panel de desplazamiento vertical embebido en un componente de texto propio de Java Swing (JTextArea) que permite el ajuste automático del texto introducido. Lo anterior permitió el ahorro de recursos a la hora del desarrollo, puesto que no fue necesario la implementación de nuevas interfaces para el mostrado de información compleja.
- **Iconografía:** Se utilizó como biblioteca de iconos, la biblioteca gráfica en línea [icons8.com](https://icons8.com). Se siguió un diseño de iconos simplista e intuitivo para no complejizar la visual del programa.
- **Imágenes:** Se emplearon imágenes en ciertas secciones del programa por diferentes motivos. Por ejemplo en la sección de la pantalla de carga para mostrar el logo del Minsap; y en la sección de autenticación, para el diseño de la interfaz, y como avatar personalizado.
- **Reportes de Salida:** Los reportes de la aplicación se muestran a través de las tablas. Las mismas contienen la información necesaria para la comprensión del significado del reporte.
- **Componentes utilizados. Prevención y Tratamiento de errores:** Para minimizar al máximo la introducción de errores se emplearon los componentes indicados para cada tipo de dato: listas de desplegables, cuadros de edición validados y formateados para cada escenario, y selecciones múltiples. Un ejemplo de validación a nivel de interfaz es en la pantalla de agregar paciente, donde se valida en tiempo real la introducción del carnet de identidad, y si este es válido, permite conocer la información referente al mismo y se habilitan y adaptan los demás componentes que dependen del mismo. En caso de que el usuario no introduzca todos los datos necesarios, o no estén correctamente llenados, se lanza un mensaje de error, y se indican los campos que se requieren modificar para el correcto funcionamiento del sistema de guardado.



De forma general, la aplicación se encuentra documentada con ayuda de cómo usar el sistema en las diferentes secciones. Las pantallas están desarrolladas en forma rectangular horizontal y con las siguientes dimensiones (diseño *desktop*):

- Pantalla de carga: 512x385 (4:3).
- Autenticación: 800x512 (39:25).
- App Principal: 1200x675 (16:9).
- Diálogos generales: 600x589.

### 2.5.2 Documentación de la Interfaz

A continuación se documentan las distintas interfaces gráficas del programa:

#### .Pantalla de Carga

Permite calmar la impaciencia del usuario por la demora del inicio del sistema, y cargar los datos del mismo de manera tal que una vez termine esta pantalla, se hayan cargado completamente los datos principales.

Como aspectos a destacar, contiene un panel de animación desarrollado internamente, y una barra de progreso que en caso de que llegue al final y detecte que no se ha cargado el sistema, la misma vuelve al inicio.



**Figura 2.2:** Pantalla de Carga.

#### .Pantalla de autenticación

Permitirá restringir el ingreso a la aplicación, además de adaptar la misma al tipo de usuario que se autentique.

Contiene campos de edición validados en tamaño, un botón para ocultar y mostrar la contraseña y un botón para ingresar. Además, al igual que sucede en la pantalla principal, se desarrolló un abarra superior personalizada semitransparente, y con las funcionalidades habituales de este tipo de componentes: la movilidad de la ventana, minimizar la ventana y cerrar la aplicación.



**Figura 2.3:** Pantalla de autenticación.

#### .Pantalla principal de la aplicación

Será la pantalla principal en donde interactuará el usuario en todo momento. Mostrará a modo de secciones las diferentes funcionalidades del programa, en dependencia del tipo de usuario registrado. Presentará las siguientes secciones:

- ❖ **Inicio:** Sección principal de esta ventana. Contiene un aspecto novedoso: los tarjeteros personalizados, componente desarrollado internamente que muestra gráficamente (sin la necesidad de complejizar el *software* con gráficos), informaciones específicas. En este caso se incluyen las informaciones relativas a las embarazadas, los pacientes en riesgo y un contador de visitas. Ver **Figura 2.4** y **Figura 2.5**.
- ❖ **Pacientes:** Sección donde se encuentra el listado principal de los pacientes en forma de tablas. Para acceder a la información interior de un paciente simplemente se realiza doble click en el paciente deseado. Como aspecto novedoso se desarrolló un motor de búsqueda de pacientes, el cual filtra el tiempo real el contenido de la tabla, en dependencia de los campos seleccionados. Ver **Figura 2.6**.
- ❖ **Hojas de Cargo:** Sección donde se encuentra el listado de las hojas de cargos en forma de tablas. Su funcionamiento se basa en la selección del día; el sistema buscará las hojas de cargo de dicho día, y las mostrará en la tabla principal de esta sección. Ver **Figura 2.7**.
- ❖ **Visitas:** Sección propia del tipo de usuario médico. En esta el médico podrá adicionar visitas de pacientes al sistema, con todo lo que ello conlleva. Además se pueden visualizar las hojas de cargos de las visitas del día actual. Ver **Figura 2.8**.
- ❖ **Análisis:** Sección propia del tipo de usuario enfermera. En esta la enfermera selecciona del listado de análisis indicados faltantes por resultados aquellos que desee para rellenar la información de los resultados. Ver **Figura 2.9**.

- ❖ **Reportes:** Sección donde se encuentran las opciones de los principales reportes del sistema. Los mismos están estructurados en tarjetas, las cuales presentan información general de cada reporte. Ver **Figura 2.10**.
- ❖ **Embarazadas:** Sección donde se muestra en forma de tabla el reporte de las embarazadas del consultorio. Ver **Figura 2.11**.
- ❖ **Vacunados:** Sección donde se muestra en forma de tabla el reporte de los pacientes vacunados del consultorio. En un primer momento, se muestran los pacientes vacunados con al menos una vacuna. Luego se puede filtrar el listado con aquellos vacunados con la vacuna seleccionada. Ver **Figura 2.12**.
- ❖ **Enfermos:** Sección donde se muestra en forma de tabla el reporte de los pacientes con enfermedades crónicas del consultorio. En un primer momento, se puede filtrar el listado con aquellos que presenten la enfermedad crónica seleccionada. Ver **Figura 2.13**.
- ❖ **En Riesgo:** Sección donde se muestra en forma de tabla el reporte de los pacientes en riesgo del consultorio. Ver **Figura 2.14**.
- ❖ **Análisis Indicados Faltantes:** Sección donde se muestra en forma de tabla el reporte de los análisis indicados faltantes del paciente seleccionado, con un rango de fechas mínimo introducido. Ver **Figura 2.15**.
- ❖ **Sin Visitas:** Sección donde se muestra en forma de tabla el reporte de los pacientes sin visitas en un rango de fechas mínimo introducido. Ver **Figura 2.16**.
- ❖ **Información del Consultorio:** Sección donde se muestra la información general del consultorio médico de la familia. Ver **Figura 2.17**.
- ❖ **Ayuda y Créditos:** Sección donde se muestra la ayuda general de la aplicación, documentación general, y los créditos, incluyendo los autores. Ver **Figura 2.18**.



**Figura 2.4:** Pantalla principal de la aplicación personalizada para el tipo de usuario médico.



**Figura 2.5:** Pantalla principal de la aplicación personalizada para el tipo de usuario enfermera.





**Figura 2.8:** Pantalla principal de la aplicación en la sección de Visitas: funcionalidad propia del tipo de usuario médico.



**Figura 2.9:** Pantalla principal de la aplicación en la sección Análisis: funcionalidad propia del tipo de usuario enfermera.



**Figura 2.10:** Pantalla principal de la aplicación en la sección de Reportes.



**Figura 2.11:** Pantalla principal de la aplicación en la sección de Reportes-> Embarazadas.

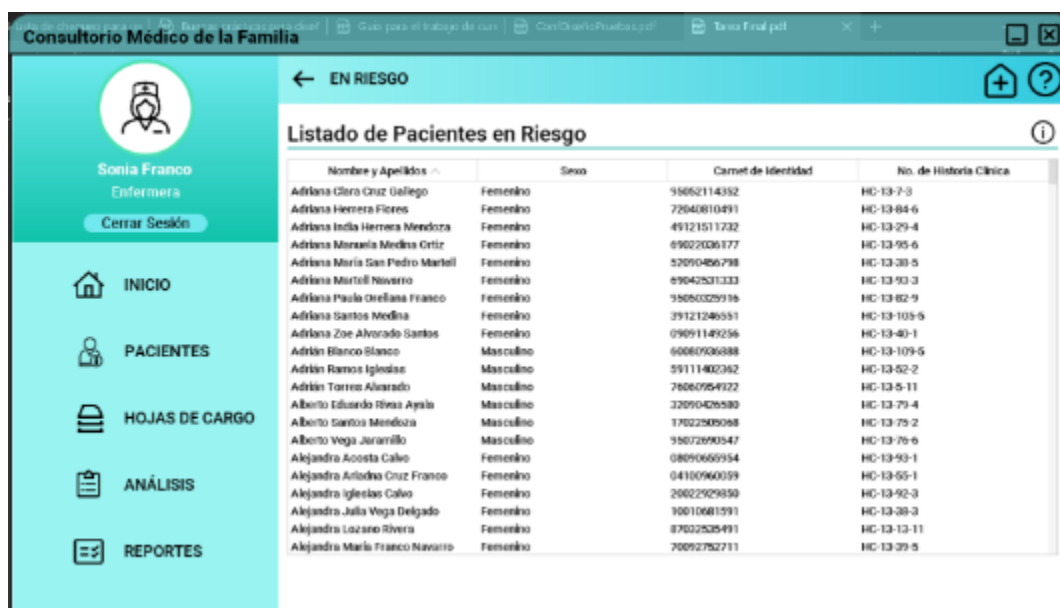


**Figura 2.12:** Pantalla principal de la aplicación en la sección de Reportes-> Vacunados.



**Figura 2.13:** Pantalla principal de la aplicación en la sección de Reportes-> Enfermos.





**Figura 2.14:** Pantalla principal de la aplicación en la sección de Reportes-> En Riesgo.



**Figura 2.15:** Pantalla principal de la aplicación en la sección de Reportes-> Análisis Indicados Faltantes.



**Figura 2.16:** Pantalla principal de la aplicación en la sección de Reportes-> Sin Visitas.



**Figura 2.17:** Pantalla principal de la aplicación mostrando la ventana de información general del consultorio.



**Figura 2.18:** Pantalla principal de la aplicación mostrando la ventana de ayuda y créditos de la aplicación.

#### .Pantalla de Información del Paciente

Permitirá visualizar la información principal de cada paciente. Está programado para que sea adaptable a cualquier paciente, mostrando solo la información necesaria.

Información Personal		
Nombre: Adriana India	Carnet de identidad: 49127511732	
Primer Apellido: Herrera	Dirección: Alma Espinosa 50	
Segundo Apellido: Mendoza	Sexo: Femenino	Edad: 72

Información Médica		
Última Prueba Citológica:	Embarazada:	
Enfermedades Crónicas	Vacunación	
Aritmia	DTPa + Hib + HB	15/12/1949
Diabetes	DTP	15/12/1953
VIH/SIDA	SRP	15/12/1957
Cáncer	AT I	15/12/1964
	TRI	15/12/1966
	A/Griual	15/12/2018

No. Historia Clínica: HC-13-29-4

**Editar**

**Figura 2.19:** Pantalla de Información del Paciente.

#### .Pantalla de Historia Clínica

Permitirá mostrar la información principal de la historia clínica de cada paciente. Mostrará el número de historia clínica, el listado de visitas realizadas, y el listado de los resultados de los análisis indicados.

Como aspecto novedoso, se implementó la celda modificada desarrollada internamente, para economizar en recursos.

**Historia Clínica**

No. Historia Clínica: HC-13-29-4

**Visitas**

Fecha	Diagnóstico	Tratamiento	Análisis Indicados	Especialidades Re...
05/12/2022	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	Heces Fecales Hemogramas Orina	Cardiología Dermatología Gastroenterología

**Resultados de los Análisis**

Fecha	Tipo de Análisis	Resultado
05/12/2022	Orina	Lorem ipsum dolor sit amet, consectetur adipiscing elit, and do eiusmod tempor incididunt ut labore et dolore magna aliqua.

**Figura 2.20:** Pantalla de Historia Clínica.

**.Pantalla de Editar Paciente**

Permitirá la edición de los datos editables de los pacientes del consultorio. Están validados los campos para minimizar la introducción de errores.

**Editar Paciente**

**Información Personal**

Nombre: Adriana India      Carnet de Identidad: 49121511732

Primer Apellido: Herrera      Dirección: Alma Espinosa      50

Segundo Apellido: Mendoza      Sexo: Femenino      Edad: 72

**Información Médica**

Última Prueba Citológica: ☐ Embarazada: ☐

**Enfermedades Crónicas**

- Artritis
- Diabetes
- VIH/SIDA
- Cáncer

**Vacunación**

DTPa + HB + HB	15/12/1949
DTP	15/12/1953
SRP	15/12/1957
AT 1	15/12/1964
TR 1	15/12/1966
A/Galaxi	15/12/2018

No. Historia Clínica: HC-13-29-4

**Aceptar** **Cancelar**

**Figura 2.21:** Pantalla de Editar Paciente.

**.Pantalla de Agregar Paciente**

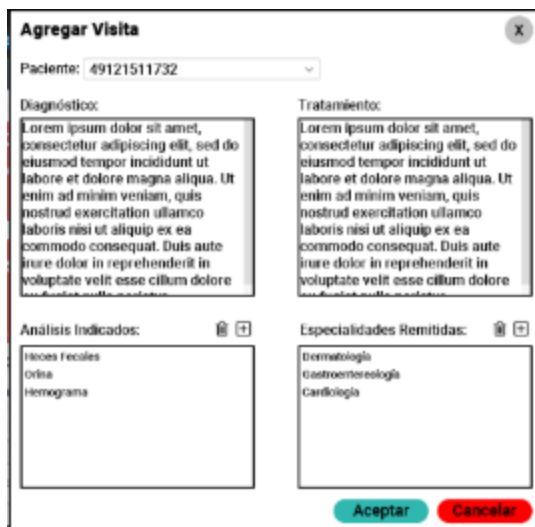
Permitirá agregar nuevos pacientes al consultorio. Los campos están validados de manera tal que se disminuyen al mínimo los posibles errores a introducir. Como aspecto novedoso, se implementó un componente que valida en tiempo real el carnet de identidad, y si el mismo es válido (garantizando la no repetición), se habilitan y rellenan los campos de la persona asociada a dicho carnet.



**Figura 2.22:** Pantalla de Agregar Paciente.

### .Pantalla de Agregar Visita

Esta sección forma parte de las funcionalidades del médico. Permitirá agregar los datos de una nueva visita del paciente seleccionado.



**Figura 2.23:** Pantalla de Agregar Visita.

### .Pantalla de Agregar Resultados de Análisis

Esta sección forma parte de las funcionalidades de la enfermera. Permitirá agregar los resultados del análisis seleccionado anteriormente.

**Agregar Resultados de Análisis**

Paciente: Adriana India Herrera      Tipo de Análisis: Orina

CI: 49121511732      Fecha de Indicación: 09/12/2022

No. de HC: HC-13-29-4

Resultados de Análisis

>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Aceptar Cancelar

**Figura 2.24:** Pantalla de Agregar Resultados de Análisis.

### 2.5.3 Principios, Reglas de Oro y Patrones de diseño de la Interfaz

La interfaz del proyecto se enfoca en el cumplimiento de buenas prácticas y estándares de diseño de interfaces gráficas de usuario en aplicaciones informáticas. Propicia una fácil comprensión del usuario. Las operaciones de interfaz son fáciles de localizar e iniciar, se economizan las acciones que tiene que realizar el usuario. La estética ayuda a la comprensión de cada funcionalidad. Se han organizado las operaciones jerárquicas de una manera que minimice la profundidad a la que un usuario debe navegar y la aplicación está libre de errores ortográficos.

Se ponen de manifiesto las reglas de oro puesto que el usuario tiene en todo momento control sobre la aplicación, se contribuye a reducir su carga de memoria y la interfaz es consistente. El usuario recibe respuesta por cada acción que realiza. Los mecanismos de interacción, íconos y procedimientos son consistentes. El sistema se anticipa a la comisión de errores y ayuda al usuario a corregirlos al introducir datos y se manejan las excepciones para que no deje de funcionar el programa cuando se cometen errores.

Cada componente del programa tiene un nombre con carácter descriptivo que permite comprender la información contenida en él y que es único. Los datos similares dentro de una misma ventana están agrupados visualmente, alineados y existen etiqueta para nombrar grupos de datos similares.

Se decidió seguir el patrón de diseño Dashboard [5] en la pantalla principal del sistema pues es un patrón muy empleado en la actualidad. Se utiliza un diseño basado en el MaterialDesign donde se usan tonalidades de un color y el resto es blanco, negro y tonos de grises. Se decidió utilizar el verde azulado como color principal de la aplicación porque es refrescante y aporta serenidad; además, es el color usual que se referencia a los temas médicos. Se utilizan fondos claros para las áreas principales de las pantallas que contrastan con la letra. También

se utilizan imágenes, íconos y una especie de gráficos (tarjetas) en la pantalla principal para mejor experiencia de usuario y mejorar.

Se usan las cajas de diálogo modales para obligar al usuario a realizar una determinada tarea antes de que pueda continuar el trabajo en cualquier otro cuadro o ventana. Se alerta a los usuarios de las consecuencias de sus acciones que pueden producir errores o que pueden llevar a estados indeseados.

En las figuras **Figura 2.25** y **Figura 2.26** se muestran los principales patrones empleados en la aplicación. Entre los mismos se encuentran:

- ❖ **1. Patrón de organización de página. Secciones tituladas:** Se emplea para agrupar grandes cantidades de información, ordenándolos en secciones de igual temática, con un título sugerente. Se pone de manifiesto en cada sección del programa donde se agrupan las informaciones.
- ❖ **2. Patrón de navegación. Panel Modal:** Se emplea para obligar el cumplimiento de una tarea por parte del usuario, marcando pautas en el flujo de trabajo. Se pone de manifiesto en todas las ventanas emergentes del programa.
- ❖ **3. Patrón de Hacer. Botón “Aceptar”:** Se emplea para demarcar visualmente un botón para aceptar y finalizar correctamente el flujo de trabajo actual. Se pone de manifiesto en todas las secciones del programa que necesiten del mismo.
- ❖ **4. Patrón para obtener información del usuario. Valores predeterminados:** Se emplean para reducir la carga de trabajo del usuario así como prevenir posibles errores. Se pone de manifiesto en cada sección del programa donde se otorguen la selección de valores predeterminados.
- ❖ **5. Patrón de organización de contenido. Buscar:** Se emplea para el filtrado de vastas listas de contenidos, ordenando el resultado en dependencia de la preferencia del usuario.

**Agregar Paciente**

**1** Información Personal

Nombre:  Carnet de Identidad:

Primer Apellido:  Dirección: «Calle Principal» «No. Casa»

Segundo Apellido:  Sexo:  Edad:

**2**

Información Médica

Última Prueba Citológica:  Embarazada: ☐

Enfermedades Crónicas:  Vacunación:

**3** Aceptar Cancelar

Figura 2.25: Patrones ejemplificados en la pantalla de Agregar Paciente.

**Consultorio Médico de la Familia**

**5** PACIENTES

Buscador de Pacientes

Nombre:  Carnet de Identidad:

Primer Apellido:  No. de Historia Clínica:

Segundo Apellido:  Dirección: «Calle Principal» «No. Casa»

Listado de Pacientes

Nombre y Apellido	Sexo	Carnet de Identidad	No. de historia clínica
Adriana Alicia Gallego Gallego	Femenino	07120438931	HC-12-4-1
Adriana Ariadna Ayala Torres	Femenino	08091428716	HC-13-71-2
Adriana Blanco Cabrera	Femenino	20011983091	HC-13-3-10
Adriana Blanco Ferreira	Femenino	11082863732	HC-13-75-1
Adriana Clara Cruz Gallego	Femenino	99052114352	HC-13-7-3
Adriana Clara Molina Flores	Femenino	18011965850	HC-13-116-3
Adriana Ferreira Torres	Femenino	05012325195	HC-13-5-2
Adriana Herrera Flores	Femenino	72040810491	HC-13-84-6
Adriana India Herrera Mendoza	Femenino	49121511732	HC-13-25-4
Adriana Marcela Medina Ortiz	Femenino	69022836177	HC-13-55-6
Adriana María San Pedro Martell	Femenino	50990456798	HC-13-38-5
Adriana Martell Navarro	Femenino	69042531333	HC-13-93-3
Adriana García Cardenas, Emma	Femenino	02060131016	HC-13-81-8

Agregar

Figura 2.26: Patrones ejemplificados en la pantalla de Pacientes.

## 2.6 Documentación de los reportes

### 2.6.1 Reporte de pacientes dada enfermedad

Prototipo del método:

*listadoPacientesEnfermedad(String):ArrayList<Paciente>.*

**Estudiante Responsable:** Eva Vázquez.

Cada paciente tiene una lista de enfermedades crónicas. El reporte consiste en una funcionalidad para buscar en el listado de pacientes aquellos con una enfermedad crónica dada. Esto generará una lista de pacientes con dicha



particularidad. Si no se especifica la enfermedad, se devuelve el listado de pacientes con al menos una enfermedad crónica.

```
public ArrayList<Paciente> listadoPacientesEnfermedad(String enfermedad){
    ArrayList<Paciente> listado = new ArrayList<Paciente>();

    //Buscar los pacientes con enfermedades crónicas
    if(enfermedad==null) {
        for(Paciente paciente : listadoPacientes) {
            if(!paciente.getEnfermedadesCronicas().isEmpty())
                listado.add(paciente);
        }
    }
    else if(!Validaciones.validarEnfermedadCronica(enfermedad))
        throw new IllegalArgumentExceptionErrores.ERROR_ENFERMEDAD_CRONICA_REPORTER);

    //Buscar los pacientes con una enfermedad crónica dada
    else {
        for(Paciente paciente : listadoPacientes) {
            if(Validaciones.validarRepeticionStringLista(paciente.getEnfermedadesCronicas(), enfermedad))
                listado.add(paciente);
        }
    }

    return listado;
}
```

Figura 2.27: Método referente al reporte en CMF.

### 2.6.2 Reporte de análisis faltantes de un paciente dado

Prototipo del método:

*listadoAnálisisFaltantesPaciente(Paciente, Date):AnálisisIndicados.*

**Estudiante Responsable:** Eduardo González.

Cada vez que un paciente asiste a una consulta el médico le consigna una lista de análisis a realizar, en caso que lo requiera. La enfermera luego introduce los resultados de dichos análisis en el sistema. Puede darse el caso que el paciente no haya realizado todos los análisis. Por tanto, se propone una funcionalidad que dado un paciente y una fecha (inclusiva) (para evitar análisis indicados no realizados obsoletos) se obtenga una lista de los análisis indicados no realizados por dicho paciente tomando como inicio una fecha dada, y además devuelva la fecha de la visita en la que se consignó cada uno de estos análisis. Para ello se creará una clase auxiliar AnalisisIndicados, que contenga la información necesaria para dar solución a esta problemática.

```
public AnalisisIndicados listadoAnálisisFaltantesPaciente(Paciente p ,Date fecha) {
    if(fecha==null)
        throw new IllegalArgumentExceptionErrores.ERROR_FECHA_ANALISIS_FALTANTES_REPORTER);
    if(p==null)
        throw new IllegalArgumentExceptionErrores.ERROR_PACIENTE_REPORTER_ANALISIS_FALTANTES);

    return new AnalisisIndicados(p,p.getHistoriaClinica().analisisIndicadosFaltantesEnVDespuesFecha(fecha));
}
```

Figura 2.28: Método referente al reporte en CMF.

```

public ArrayList<Análisis> analisisIndicadosFaltantesEnVDespuesFecha(Date fecha){
    if(fecha == null)
        throw new IllegalArgumentException(Errores.ERROR_FECHA_ANALISIS_INDICADOS_HC);

    Calendar c = Calendar.getInstance();
    c.setTime(fecha);
    ArrayList<Análisis> listado = new ArrayList<Análisis>();

    for(Análisis a : listadoAnálisis) {
        Calendar c1 = Calendar.getInstance();
        c1.setTime(a.getFechaIndicacion());
        if((validaciones.validarCalendarismoDia(c, c1) || a.getFechaIndicacion().after(fecha))
            && !a.tieneResultado())
            listado.add(a);
    }

    return listado;
}

```

**Figura 2.29:** Método referente al reporte en Historia Clínica.

### 2.6.3 Reporte de pacientes embarazadas

**Prototipo del método:** *listadoEmbarazadas():ArrayList<PacienteFemenina>*.

**Estudiante Responsable:** Eva Vázquez.

El médico debe dar un tratamiento especializado a aquellas pacientes que se encuentren en estado de embarazo. Por tanto se propone un reporte para dar a conocer el listado de embarazadas en el área del consultorio médico.

```

public ArrayList<PacienteFemenina> listadoEmbarazadas(){
    ArrayList<PacienteFemenina> listado = new ArrayList<PacienteFemenina>();

    //Buscar las pacientes embarazadas
    for(Paciente paciente : listadoPacientes) {
        if(paciente instanceof PacienteFemenina && ((PacienteFemenina)paciente).getEstaEmbarazada()) {
            listado.add(((PacienteFemenina)paciente));
        }
    }

    Collections.sort(listado, Comparadores.comparadorPersona(Comparadores.PERSONA_NOMBRE_COMPLETO));

    return listado;
}

```

**Figura 2.30** Método referente al reporte en CMF.

### 2.6.4 Reporte de pacientes en riesgo (funcionalidad embebida)

**Prototipo del método:** *pacientesEnRiesgo():ArrayList<Paciente>*.

**Estudiante Responsable:** Eduardo González.

Esta es una funcionalidad embebida en la problemática dada, pero es necesario documentarla como reporte, puesto que es así como actúa. Permitirá conocer el listado de los pacientes en riesgo, ya sea por tener más de 3 enfermedades crónicas, y/o, en el caso de las pacientes femeninas, llevar más de 3 años sin realizarse la prueba citológica.

```

public ArrayList<Paciente> pacientesEnRiesgo(){
    ArrayList<Paciente> listado = new ArrayList<Paciente>();

    for(Paciente paciente : listadoPacientes) {
        if(paciente.enRiesgo())
            listado.add(paciente);
    }

    Collections.sort(listado, Comparadores.comparadorPersona(Comparadores.PERSONA_NOMBRE_COMPLETO));

    return listado;
}

```

**Figura 2.31:** Método referente al reporte en CMF.

```

public boolean enRiesgo() {
    return enfermedadesCronicas.size() > Definiciones.CANT_ENFERMEDADES_CRONICAS_RIESGO;
}

```

**Figura 2.32:** Método referente al reporte en Paciente.

```

@Override
public boolean enRiesgo() {
    boolean enRiesgo = super.enRiesgo();

    if(!enRiesgo && Validaciones.validarEnteroRango(getEdad(), Definiciones.EDAD_MIN_RECOMENDADA_PRUEBA_CITOLOGICA,
        Definiciones.EDAD_MAX_RECOMENDADA_PRUEBA_CITOLOGICA)) {
        int agnosUltimaPruebaCitologica = fechaUltimaPruebaCitologica==null ? getEdad()-Definiciones.EDAD_MIN_RECOMENDADA_PRUEBA_CITOLOGICA :
            Auxiliares.determinarDiferenciaAnyos(fechaUltimaPruebaCitologica, new Date());
        enRiesgo = agnosUltimaPruebaCitologica>Definiciones.CANT_AGNOS_RIESGO_PRUEBA_CITOLOGICA;
    }

    return enRiesgo;
}

```

**Figura 2.33:** Método referente al reporte en PacienteFemenina.

### 2.6.5 Reporte de pacientes sin visitas desde una fecha dada

**Prototipo del método:** *listadoPacientesSinVisita(Date):ArrayList<Paciente>*.

**Estudiante Responsable:** Eva Vázquez.

Se propone una funcionalidad para conocer el listado de pacientes que no han realizado visitas o consultas en el consultorio desde una fecha proporcionada (inclusiva).

```

public ArrayList<Paciente> listadoPacientesSinVisita(Date fecha){
    ArrayList<Paciente> listado = new ArrayList<Paciente>();

    //Buscar los pacientes sin visitas después de una fecha dada (inclusivo)
    if(fecha!=null){
        for(Paciente paciente : listadoPacientes) {
            if(paciente.getHistoriaClinica().visitasEnVDespuesFechaDada(fecha).isEmpty())
                listado.add(paciente);
        }
    }

    return listado;
}

```

**Figura 2.34:** Método referente al reporte en CMF.

```

public ArrayList<Visita> visitasEnVDespuesFechaDada(Date fecha) {
    ArrayList<Visita> visitas = new ArrayList<Visita>();

    if(fecha!=null) {
        Calendar c = Calendar.getInstance();
        c.setTime(fecha);
        for(Visita v : listadoVisitas) {
            Calendar c1 = Calendar.getInstance();
            c1.setTime(v.getFecha());
            if(Validaciones.validarCalendarsMismoDia(c, c1) || v.getFecha().after(fecha))
                visitas.add(v);
        }
    }

    return visitas;
}

```

**Figura 2.35:** Método referente al reporte en HistoriaClinica.

### 2.6.6 Reporte de pacientes vacunados dado una vacuna

Prototipo del método: *listadoPacientesVacuna(Vacuna):ArrayList<Paciente>*.

Estudiante Responsable: Eduardo González.

Cada paciente tiene una lista de vacunas. La propuesta consiste en desarrollar un método que permite buscar en el listado de pacientes aquellos que se hayan puesto una vacuna dada. Esto generará una lista de pacientes con dicha particularidad. En caso de que no se especifique una vacuna, se generará el listado de pacientes con al menos una vacuna.

```
public ArrayList<Paciente> listadoPacientesVacuna(String vacuna){
    ArrayList<Paciente> listado = new ArrayList<Paciente>();
    //Buscar los pacientes vacunados
    if(vacuna==null) {
        for(Paciente paciente : listadoPacientes) {
            if(!paciente.listadoVacunasPuestas().isEmpty())
                listado.add(paciente);
        }
    } else if(!validaciones.validarNombreVacuna(vacuna))
        throw new IllegalArgumentException(Errores.ERROR_VACUNA_REPORT);
    //Buscar los pacientes con una vacuna dada
    else {
        for(Paciente paciente : listadoPacientes) {
            if(validaciones.validarRepeticionStringListo(paciente.listadoVacunasPuestas(), vacuna))
                listado.add(paciente);
        }
    }
    return listado;
}
```

Figura 2.36: Método referente al reporte en CMF.

```
public ArrayList<String> listadoVacunasPuestas(){
    ArrayList<String> listado = new ArrayList<String>();

    for(Vacuna v : vacunaciones) {
        listado.add(v.getNombreVacuna());
    }

    return listado;
}
```

Figura 2.37: Método referente al reporte en Paciente.

## 2.7 Descripción de los patrones empleados

Tras el análisis de la problemática dada, se determinó que solamente era necesario la existencia de un solo consultorio médico de la familia (CMF) en el programa, puesto que el *software* desarrollado solamente tendría como objetivo informatizar un CMF. Para garantizar la unicidad de CMF, se diseñó el problema implementando el patrón creacional **Singleton**, el cual permitió la existencia de un único CMF en toda la ejecución del programa, y su acceso global. Este patrón no solo fue implementado en la lógica de negocios, sino también en la piscina de datos y en la interfaz gráfica, realizando adecuaciones, para adaptar la solución al escenario dado. En el caso de PiscinaDatos, se empleó para que una vez iniciado el programa, se generara un único listado de carnets de identidad, y así garantizar la correcta inicialización de los datos. En el caso de AppPrincipal (interfaz gráfica principal), se empleó para que solamente existiera una instancia

de la misma en todo el programa, variando su visual, en dependencia del tipo de usuario registrado.

```
private static CMF instancia;

private CMF(int numeroConsultorio, String nombrePoliclinico, String nombreDirectorPoliclinico,
            Medico doctor, Enfermera enfermera) {
    setNumeroConsultorio(numeroConsultorio);
    setNombrePoliclinico(nombrePoliclinico);
    setNombreDirectorPoliclinico(nombreDirectorPoliclinico);
    setDoctor(doctor);
    setEnfermera(enfermera);
    listadoPacientes = new ArrayList<Paciente>();
    listadoHojaDeCarga = new ArrayList<HojaDeCargaDia>();
}

public static CMF getInstance(int numeroConsultorio, String nombrePoliclinico, String nombreDirectorPoliclinico,
                              Medico doctor, Enfermera enfermera) {
    if (instancia == null) {
        instancia = new CMF(numeroConsultorio, nombrePoliclinico, nombreDirectorPoliclinico, doctor, enfermera);
    }
    return instancia;
}

public static CMF getInstance() {
    if (instancia == null) {
        throw new NullPointerException(Errores.ERROR_INSTANCIA_CMF_NULLA);
    }
    return instancia;
}
```

**Figura 2.38:** Patrón Singleton en CMF.

```
private static PiscinaDatos instancia;

private PiscinaDatos() {
    listadoCIGenerados = new ArrayList<String>();
    generarListadoCi(generarEntero(Definiciones.CANT_MIN_PACIENTES, Definiciones.CANT_MAX_PACIENTES));
    contadorCI = 0;
    ciMedico = generarCiMedicoAleatorio();
    ciEnfermera = generarCiEnfermeraAleatorio();
}

public static PiscinaDatos getInstance() {
    if (instancia == null) {
        instancia = new PiscinaDatos();
    }
    return instancia;
}
```

**Figura 2.39:** Patrón Singleton en PiscinaDatos.

```
private static AppPrincipal instancia;

public static AppPrincipal getInstance(TipoCuentaUsuario usuarioRegistrado) {
    if (instancia == null || (instancia != null && !usuario.equals(usuarioRegistrado))) {
        instancia = new AppPrincipal(usuarioRegistrado);
    }
    return instancia;
}

private AppPrincipal(TipoCuentaUsuario usuarioRegistrado) {
    // ...
}
```

**Figura 2.40:** Patrón Singleton en AppPrincipal.

Luego del análisis del problema se determinó que no era necesario la implementación de otros patrones de diseño de la programación orientada a objetos además del **singleton**, para dar una solución eficiente a la problemática. Ejemplos de patrones no utilizados son:

- ❖ **Patrón estructural Composite:** Para la solución de la problemática, no hubo la necesidad de representar en relaciones jerárquicas del tipo todo-parte, es decir, no existe una representación arbórea del problema dado.

- ❖ **Patrón creacional Prototype:** Para la solución de la problemática, no es necesario ni factible la creación de fábricas clonadoras de una instancia específica de un objeto, puesto que cada dato es único.

## 2.8 Pruebas Realizadas

El diseño de pruebas funcionales es un elemento crítico para la garantía de calidad del software y representa una revisión final de las especificaciones, del diseño y de la codificación del proyecto. Este proceso de ejecución del programa tiene como objetivo encontrar defectos tan pronto como sea posible. Se han diseñado casos de pruebas dinámicas (utilizando técnicas de caja blanca y de caja negra) para poder ejecutarlas con el *JUnit Testing Framework*, analizar los resultados obtenidos y comprobar si se cumplen los requerimientos especificados y las necesidades o expectativas del usuario.

Para la realización de las presentes pruebas se utilizó de bibliografía principal el libro de Ingeniería de Software Parte 1, de Roger S. Pressman [6].

### 2.8.1 Diseño de Pruebas de Caja Blanca

//Clase CMF

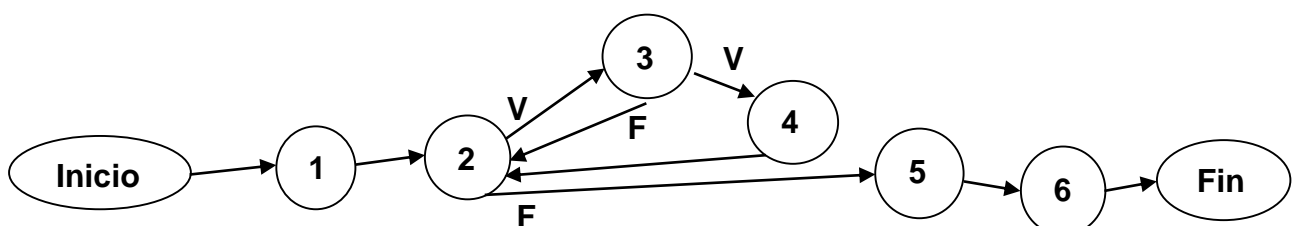
```
public ArrayList<Paciente> pacientesEnRiesgo(){
    ArrayList<Paciente> listado = new ArrayList<Paciente>();

    for(Paciente paciente : listadoPacientes) {
        if(paciente.enRiesgo())
            listado.add(paciente);
    }

    Collections.sort(listado, Comparadores.comparadorPersona(Comparadores.PERSONA_NOMBRE_COMPLETO));

    return listado;
}
```

	Nodos
1	ArrayList<Paciente> listado = new ArrayList<Paciente>();
2	for(Paciente paciente : listadoPacientes)
3	if(paciente.enRiesgo())
4	listado.add(paciente);
5	Collections.sort(listado, Comparadores.comparadorPersona(Comparadores.PERSONA_NOMBRE_COMPLETO));
6	return listado;



Camino 1: 1, 2, 3, 4, 2, 5, 6. (CP 1)

Camino 2: 1, 2, 3, 2, 5, 6. (CP 2)

Camino 3: 1, 2, 5, 6. (CP 3)

Complejidad ciclomática: 3.

**Nota:** Para realizar los CP 1 y 2, el listadoPacientes debe tener al menos un Paciente con la lista de enfermedadesCronicas mayor que 3. Se tomará en cuenta como variable de entrada solamente el atributo enfermedadesCronicas. En el CP 3 el listadoPacientes debe estar vacío.

CASO DE PRUEBA					
Clase		CMF			
Método		ArrayList<Paciente> pacientesEnRiesgo()			
Desarrollador		Eduardo Alejandro González Martell			
Probador		Eva Ercilia Vázquez García			
Fecha					
Combinaciones de valores de entrada			Resultados		
CP	Nombres de Variables de entrada	Valor para cada variable de entrada	Técnica de diseño empleada	Resultados Esperados	Resultados reales
1	enfermedades Cronicas	("Asma", "Diabetes", "Epilepsia", "Esclerosis múltiple")	Prueba de camino básico	Sigue el camino 1. Se añade el paciente de listadoPacientes al listado que se retorna porque está en riesgo.	Retorna el listado con el paciente en riesgo
2	enfermedades Cronicas	("Asma")	Prueba de camino básico	Sigue el camino 2. No se añade el paciente de listadoPacientes al listado que se retorna porque no está en riesgo.	Retorna el listado vacío
3	listadoPacientes	[]	Prueba de camino básico	Sigue el camino 3. Se retorna una lista vacía porque no hay pacientes en listadoPacientes.	Retorna el listado vacío

//Clase CMF

```
public ArrayList<PacienteFemenina> listadoPacientesFemeninas() {
    ArrayList<PacienteFemenina> listado = new ArrayList<PacienteFemenina>();

    for(Paciente paciente : listadoPacientes)
        if(paciente instanceof PacienteFemenina)
            listado.add((PacienteFemenina)paciente);

    return listado;
}
```

Condiciones: for, if.

for: true            listadoPacientes = [Paciente paciente1].    (CP 1)

for: false    listadoPacientes = [].    (CP 3)

if: true       listadoPacientes = [PacienteFemenina paciente1].    (CP 2)

if: false           listadoPacientes = [Paciente paciente1].    (CP 1)

**Nota:** En el CP 1 se debe introducir un paciente masculino al consultorio y en el CP 2 se debe introducir una paciente femenina. Se tomará en cuenta como variable de entrada solamente el atributo ci (carnet de identidad), porque a partir de este es q se conocerá el sexo del paciente. En el CP 3 el listadoPacientes debe estar vacío.

CASO DE PRUEBA					
Clase		CMF			
Método		ArrayList<PacienteFemenina> listadoPacientesFemeninas()			
Desarrollador		Eva Ercilia Vázquez García			
Probador		Eduardo Alejandro González Martell			
Fecha					
Combinaciones de valores de entrada				Resultados	
CP	Nombres de Variables de entrada	Valor para cada variable de entrada	Técnica de diseño empleada	Resultados Esperados	Resultados reales
1	ci	"02101081224"	Prueba de condiciones	Entra en el for pero no entra en el if. Se retorna una lista vacía porque no hay pacientes femeninas.	Retorna una lista vacía
2	ci	"02101081295"	Prueba de condiciones	Entra en el for y entra en el if. Se añade el paciente de listadoPacientes al listado que se retorna porque es instancia de PacienteFemenina.	Retorna la lista con un paciente
3	listadoPacientes	[]	Prueba de condiciones	No entra en el for. Se retorna una lista vacía porque no hay pacientes en listadoPacientes.	Retorna una lista vacía



### 2.8.2 Diseño de Pruebas de Caja Negra

Se han diseñado pruebas asociadas con diferentes funcionalidades del programa. Para ello se ha realizado un análisis de las condiciones que definen a cada escenario y buscar combinaciones de valores de las variables de entrada que permitan probar cada escenario.

//Clase CMF

```
public ArrayList<Paciente> listadoPacientesEnfermedad(String enfermedad){
    ArrayList<Paciente> listado = new ArrayList<Paciente>();

    //Buscar los pacientes con enfermedades crónicas
    if(enfermedad==null) {
        for(Paciente paciente : listadoPacientes) {
            if(!paciente.getEnfermedadesCronicas().isEmpty())
                listado.add(paciente);
        }
    }
    else if(!Validaciones.validarEnfermedadCronica(enfermedad))
        throw new IllegalArgumentExceptionErrores.ERROR_ENFERMEDAD_CRONICA_REPORT);

    //Buscar los pacientes con una enfermedad crónica dada
    else {
        for(Paciente paciente : listadoPacientes) {
            if(Validaciones.validarRepeticionStringLista(paciente.getEnfermedadesCronicas(), enfermedad))
                listado.add(paciente);
        }
    }

    return listado;
}
```

Identificador	Nombre HU	Prioridad	Puntos estimados
HU1	listadoPacientesEnfermedad		
<b>Usuario:</b> Médico o enfermera.			
<b>Descripción:</b> Método para obtener una lista de los pacientes que tienen una enfermedad crónica dada.			
<b>Programador Responsable:</b> Eduardo Alejandro González Martell.			
<b>Escenarios</b>			
Nombre	Condición	Resultado esperado	
El listado de pacientes del consultorio tiene paciente o pacientes con la enfermedad crónica dada.	SI la enfermedad crónica dada está en la lista de enfermedades del consultorio (piscinaEnfermedadesCronicas) AND listadoPacientes tiene algún Paciente que tenga la enfermedad dada en su lista de enfermedades crónicas.	Se retorna una lista con los pacientes que cumplan la condición.	

Se desea obtener listado de pacientes con enfermedades crónicas (otro uso del método).	Si la enfermedad crónica dada es igual a null AND listadoPacientes tiene algún Paciente que tenga su lista de enfermedades crónicas no vacía.	Se retorna una lista con los pacientes que cumplan la condición.
La enfermedad crónica dada no es válida.	Si la enfermedad crónica dada NOT está en la lista de enfermedades del consultorio (piscinaEnfermedadesCronicas).	Se lanza un IllegalArgumentException con el mensaje: "La enfermedad crónica no es válida" y no se retorna nada.
El listado de pacientes del consultorio no tiene un paciente o pacientes con la enfermedad crónica dada.	Si la enfermedad crónica dada está en la lista de enfermedades del consultorio (piscinaEnfermedadesCronicas) AND listadoPacientes NOT tiene ningún Paciente que tenga la enfermedad dada en su lista de enfermedades crónicas.	Se retorna una lista vacía.
<b>Variables</b>		
<b>Nombre</b>	<b>Tipo o Dominio</b>	<b>Descripción</b>
listadoPacientes	Lista de objetos tipo Paciente.	El listado de pacientes del consultorio.
enfermedad	Cadena de caracteres.	La enfermedad crónica dada.
piscinaEnfermedadesCronicas	Lista de cadenas de texto.	Listado de enfermedades crónicas predeterminadas en el consultorio.

<b>CASO DE PRUEBA</b>					
<b>Historia de Usuario</b>		HU1			
<b>Desarrollador</b>		Eduardo Alejandro González Martell.			
<b>Probador</b>		Eva Ercilia Vázquez García.			
<b>Fecha</b>					
<b>Combinaciones de valores de entrada</b>			<b>Escenario</b>	<b>Resultados Esperados</b>	<b>Resultados reales</b>
<b>CP</b>	<b>Nombres de Variables de entrada</b>	<b>Valor para cada variable de entrada</b>			
1	(listadoPacientes, piscinaEnfermedadesCronica,	(["Lia", "Torres", "Torres", "02101081293", PiscinaDatos.generarDireccion()], ["Artritis", "Asma", "Cáncer", "Diabetes",	El listado de pacientes del consultorio tiene paciente o pacientes con la enfermedad crónica dada.	Se retorna una lista con el paciente que se encuentra en listadoPacientes.	Se retorna lista con un paciente

	enfermedad, )	"VIH/SIDA", "Epilepsia", "Esclerosis Múltiple"], "Asma" )			
2	(listadoPacientes,  piscinaEnfermedades Cronica,  enfermedad, )	(["Lia", "Torres", "Torres", "02101081293", PiscinaDatos.generarD ireccion()], ["Artritis", "Asma", "Cáncer", "Diabetes", "VIH/SIDA", "Epilepsia", "Esclerosis Múltiple"], null )	Se desea obtener listado de pacientes con enfermedades crónicas (otro uso del método).	Se retorna una lista con el paciente que se encuentra en listadoPacientes. ntes.	Se retorna una lista con un paciente
3	(listadoPacientes,  piscinaEnfermedades Cronica,  enfermedad, )	(["Lia", "Torres", "Torres", "02101081293", PiscinaDatos.generarD ireccion()], ["Artritis", "Asma", "Cáncer", "Diabetes", "VIH/SIDA", "Epilepsia", "Esclerosis Múltiple"], "Hola" )	La enfermedad crónica dada no es válida.	Se lanza un IllegalArgumentException con el mensaje: "La enfermedad crónica no es válida" y se interrumpe el método.	Se lanza la excepción
4	(listadoPacientes,  piscinaEnfermedades Cronica,  enfermedad, )	(["Lia", "Torres", "Torres", "02101081293", PiscinaDatos.generarD ireccion()], ["Artritis", "Asma", "Cáncer", "Diabetes", "VIH/SIDA", "Epilepsia", "Esclerosis Múltiple"], "Epilepsia" )	El listado de pacientes del consultorio no tiene un paciente o pacientes con la enfermedad crónica dada.	Se retorna una lista vacía.	Se retorna una lista vacía

//Clase abstracta Persona

```
public void setNombre(String nombre) {
    if(Validaciones.validarStringNoVacio(nombre) && Validaciones.validarStringNoTodoEspacio(nombre)
        && Validaciones.validarStringTodoLetra(nombre) && Validaciones.validarTamString(nombre,
            Definiciones.CANT_MIN_CARACTERES_NOMBRE, Definiciones.CANT_MAX_CARACTERES_NOMBRE))
        this.nombre = nombre;
    else
        throw new IllegalArgumentException(Errores.ERROR_NOMBRE_PERSONA);
}
```

Identificador	Nombre HU	Prioridad	Puntos estimados
HU2	setNombre		
<b>Usuario:</b> Médico o enfermera.			
<b>Descripción:</b> Método para insertar el nombre de una persona.			
<b>Programador Responsable:</b> Eva Ercilia Vázquez García.			
<b>Escenarios</b>			
Nombre	Condición	Resultado esperado	
Inserción exitosa del nombre.	SI el nombre no está vacío AND SI no tiene solo espacio AND SI está entre el rango predeterminado de cantidad de caracteres.	Se guarda el nombre dado en el atributo nombre de la persona.	
Nombre demasiado extenso.	SI el nombre sobrepasa el máximo de caracteres predeterminado para la longitud de un nombre.	Se lanza un IllegalArgumentException con el mensaje: “Nombre de persona no válido.” y no se guarda el nombre.	
Nombre vacío.	SI el nombre está vacío.	Se lanza un IllegalArgumentException con el mensaje: “Nombre de persona no válido.” y no se guarda el nombre.	
<b>Variables</b>			
Nombre	Tipo o Dominio	Descripción	
nombre	Cadena no vacía de entre 3 y 30 letras. No se aceptan caracteres o números.	Nombre que el usuario desea ponerle a una persona determinada.	

**Nota:** Los casos de prueba se realizarán llamando al método en una instancia de la clase Paciente, que hereda de la clase abstracta Persona.

CASO DE PRUEBA					
Historia de Usuario		HU2			
Desarrollador		Eva Ercilia Vázquez García.			
Probador		Eduardo Alejandro González Martell.			
Fecha					
Combinaciones de valores de entrada			Escenario	Resultados Esperados	Resultados reales
CP	Nombres de Variables de entrada	Valor para cada variable de entrada			
1	nombre	"Vicente"	Inserción exitosa del nombre.	Se guarda el nombre dado en el atributo nombre de la persona.	Se guarda el nuevo nombre
2	nombre	"qwertyuiopasdfghj lzxvbnmqwerty"	Nombre demasiado extenso.	Se lanza un IllegalArgumentException con el mensaje: "Nombre de persona no válido." y no se guarda el nombre.	Se lanza la excepción
3	nombre	" "	Nombre vacío.	Se lanza un IllegalArgumentException con el mensaje: "Nombre de persona no válido." y no se guarda el nombre.	Se lanza la excepción

//Clase Paciente Femenina

```

public void setEstaEmbarazada(boolean estaEmbarazada) {
    if(estaEmbarazada && !Validaciones.validarEnteroRango(getEdad(),
        Definiciones.EDAD_MIN_PRUEBAS_MUJER, Definiciones.EDAD_MAX_PRUEBAS_MUJER))
        throw new IllegalArgumentExceptionErrores.ERROR_EDAD_EMBARAZO);

    this.embarazada = estaEmbarazada;
}

```

Identificador	Nombre HU	Prioridad	Puntos estimados
HU3	setEstaEmbarazada		
<b>Usuario:</b> Médico o enfermera.			
<b>Descripción:</b> Método para cambiar el estado de embarazo de una paciente femenina.			
<b>Programador Responsable:</b> Eduardo Alejandro González Martell.			
<b>Escenarios</b>			
<b>Nombre</b>	<b>Condición</b>	<b>Resultado esperado</b>	

Paciente femenina que está embarazada.	Si la edad extraída del carnet de identidad de la paciente femenina es mayor o igual que 10 y menor o igual que 85.	La paciente femenina pasa a estar embarazada.
Paciente femenina demasiado joven para salir embarazada.	Si la edad extraída del carnet de identidad de la paciente femenina es menor que 10.	Se lanza un <code>IllegalArgumentException</code> con el mensaje: "La paciente no tiene la edad válida de embarazo." y no se guarda nada.
Paciente femenina con edad muy avanzada para salir embarazada.	Si la edad extraída del carnet de identidad de la paciente femenina es mayor que 85.	Se lanza un <code>IllegalArgumentException</code> con el mensaje: "La paciente no tiene la edad válida de embarazo." y no se guarda nada.
<b>Variables</b>		
<b>Nombre</b>	<b>Tipo o Dominio</b>	<b>Descripción</b>
ci	Cadena numérica no vacía de 11 números.	Carnet de identidad de la paciente femenina.

**Nota:** En el CP 1 se probará el método para una mujer que puede salir embarazada; en el CP 2, para una niña de menos de 10 años; en el CP 3, para una anciana de más de 70 años. Se tomará en cuenta como variable de entrada solamente el atributo ci (carnet de identidad), porque a partir de este es q se conocerá la edad del paciente.

<b>CASO DE PRUEBA</b>					
<b>Historia de Usuario</b>		HU3			
<b>Desarrollador</b>		Eduardo Alejandro González Martell.			
<b>Probador</b>		Eva Ercilia Vázquez García.			
<b>Fecha</b>					
<b>Combinaciones de valores de entrada</b>			<b>Escenario</b>	<b>Resultados Esperados</b>	<b>Resultados reales</b>
<b>CP</b>	<b>Nombres de Variables de entrada</b>	<b>Valor para cada variable de entrada</b>			
1	ci	"02101081297"	Paciente femenina que está embarazada.	El atributo embarazada de la paciente femenina pasa a ser true.	La paciente pasa a ser embarazada
2	ci	"20101081297"	Paciente femenina demasiado joven para salir embarazada.	Se lanza un <code>IllegalArgumentException</code> con el mensaje: "La paciente no tiene la edad válida de embarazo." y se detiene el método.	Se lanza una excepción

3	ci	"30032534375"	Paciente femenina con edad muy avanzada para salir embarazada.	Se lanza un IllegalArgumentException con el mensaje: "La paciente no tiene la edad válida de embarazo." y se detiene el método.	Se lanza una excepción
---	----	---------------	--	---	------------------------

//Clase CMF

```
public void addPaciente(String nombre, String primerApellido, String segundoApellido,
                        String carnetIdentidad, Direccion direccion) {
    if(Validaciones.validarUnicidadCI(carnetIdentidad))
        listadoPacientes.add(new Paciente(nombre, primerApellido, segundoApellido, carnetIdentidad, direccion));
    else
        throw new IllegalArgumentExceptionErrores.ERROR_PACIENTE);
}
```

Identificador	Nombre HU	Prioridad	Puntos estimados
HU4	addPaciente		
<b>Usuario:</b> Médico o enfermera.			
<b>Descripción:</b> Método para insertar el nombre de un paciente.			
<b>Programador Responsable:</b> Eva Ercilia Vázquez García.			
<b>Escenarios</b>			
Nombre	Condición	Resultado esperado	
Inserción exitosa del paciente.	SI el carnet de identidad del paciente es único.	Se añade el paciente al listado de pacientes del consultorio.	
Paciente repetido.	SI el carnet de identidad del paciente NOT es único (ya está registrado ese paciente).	Se lanza un IllegalArgumentException con el mensaje: “Paciente no válido.” y no se añade al paciente.	
Paciente no válido.	SI los datos del paciente son incorrectos.	Se lanza un IllegalArgumentException con el mensaje: “Paciente no válido.” y no se añade al paciente.	
<b>Variables</b>			
Nombre	Tipo o Dominio	Descripción	
paciente	Objeto tipo Paciente.	El paciente a añadir.	

**Nota:** En el CP 1 se probará el método con el listadoPacientes vacío. En el CP 2 y el CP 3, el listadoPacientes tendrá solamente al paciente insertado en el primer caso de prueba.

<b>CASO DE PRUEBA</b>					
<b>Historia de Usuario</b>		HU4			
<b>Desarrollador</b>		Eva Ercilia Vázquez García.			
<b>Probador</b>		Eduardo Alejandro González Martell.			
<b>Fecha</b>					
<b>Combinaciones de valores de entrada</b>			<b>Escenario</b>	<b>Resultados Esperados</b>	<b>Resultados reales</b>
<b>CP</b>	<b>Nombres de Variables de entrada</b>	<b>Valor para cada variable de entrada</b>			
1	paciente	[ _ , _ , _ , "02101081294", _ ]	Inserción exitosa del paciente.	Se guarda el nombre dado en el atributo nombre del paciente.	Se guarda el paciente
2	paciente	[ _ , _ , _ , "02101081294", _ ]	Paciente repetido.	Se lanza un IllegalArgumentException con el mensaje: "Nombre de persona no válido." y no se guarda el nombre.	Se lanza la excepción
3	paciente	[null, null , null, null null]	Paciente no válido.	Se lanza un IllegalArgumentException con el mensaje: "Nombre de persona no válido." y no se guarda el nombre.	Se lanza la excepción



Identificador	Nombre HU	Prioridad	Puntos estimados
HU5	Agregar paciente		
Usuario: Médico o enfermera.			
Descripción: Funcionalidad para agregar un paciente al consultorio.			
Programador Responsable: Eva Ercilia Vázquez García.			
Escenarios			
Nombre	Condición	Resultado esperado	
Agregación exitosa del paciente.	SI todos los datos se introducen correctamente (validarComponentes() == true) AND se ejecuta una Acción sobre el botón “Aceptar” (aceptarBtn) AND se ejecuta una Acción sobre el botón “Sí” (JOptionPane.YES_OPTION).	Se muestra el mensaje de diálogo: “¿Desea agregar al paciente?” y al usuario marcar la opción “Sí”, se agrega el paciente a la lista de pacientes y se muestra el mensaje informativo: “El paciente fue agregado correctamente”.	
Agregación cancelada del paciente.	SI todos los datos se introducen correctamente (validarComponentes() == true) AND se ejecuta una Acción sobre el botón “Aceptar” (aceptarBtn) AND (se ejecuta una Acción sobre el botón “No” (!JOptionPane.YES_OPTION) OR se ejecuta una Acción sobre el botón de salida).	Se muestra el mensaje de diálogo: “¿Desea agregar al paciente?” y al usuario marcar la opción “No”, regresa a la pantalla para continuar con la agregación del paciente.	
Cancelación completa de la agregación del paciente.	SI se ejecuta una Acción sobre el botón “Cancelar” (cancelarBtn) AND se ejecuta una Acción sobre el botón “Sí” (JOptionPane.YES_OPTION).	Se muestra el mensaje de diálogo: “Los cambios no guardados se perderán ¿Desea continuar?” y al usuario marcar la opción “Sí”, se cancela la agregación del paciente y regresa a la pantalla anterior.	
Cancelación incompleta de la agregación del paciente.	SI se ejecuta una Acción sobre el botón “Cancelar” (cancelarBtn) AND (se ejecuta una Acción sobre el botón “No” (!JOptionPane.YES_OPTION) OR se ejecuta una Acción sobre el botón de salida).	Se muestra el mensaje de diálogo: “Los cambios no guardados se perderán ¿Desea continuar?” y al usuario marcar la opción “No”, regresa a la pantalla para continuar con la agregación del paciente.	
Campos vacíos o erróneos.	SI NOT todos los datos se introducen correctamente (validarComponentes() == false) AND se ejecuta una Acción sobre el botón “Aceptar” (aceptarBtn).	Al validar los componentes, se detectan los que están incorrectos, se ponen en rojo para ser identificados por el usuario y se muestra el siguiente mensaje de error: “Compruebe los datos de los campos señalados”.	
Variables			

Nombre	Tipo o Dominio	Descripción
Nombre	Cadena no vacía de entre 3 y 30 letras. No se aceptan caracteres o números.	Nombre del paciente.
Primer Apellido	Cadena no vacía de entre 3 y 25 letras. No se aceptan caracteres o números.	Primer apellido del paciente.
Segundo Apellido	Cadena no vacía de entre 3 y 25 letras. No se aceptan caracteres o números.	Segundo apellido del paciente.
Carnet de identidad	Cadena numérica no vacía de 11 dígitos. Solo se aceptan números.	Carnet de identidad del paciente.
callePrincipalComboBox	Lista desplegable de opciones donde solo se puede seleccionar una.	Lista de calles que pertenecen al consultorio donde se debe escoger en la que vive el paciente.
numeroCasaComboBox.getSelectedIndex()	Lista desplegable de opciones donde solo se puede seleccionar una.	Lista de los números de las viviendas que pertenecen al consultorio donde se debe escoger el que corresponde a la casa del paciente.
(aceptarBtn.getAction != null)	Condición.	Si no es null significa que el botón aceptarBtn ha sido seleccionado.
(cancelarBtn.getAction != null)	Condición.	Si no es null significa que el botón cancelarBtn ha sido seleccionado.
(JOptionPane.YES_OPTION)	Condición.	Para saber si fue seleccionado o no el botón "Sí" del mensaje de diálogo.

**Nota:** Los casos de prueba se realizarán insertando los siguientes datos en la pantalla donde se agrega un paciente al consultorio, a la cual se accede a través de la opción Pacientes que se encuentra a la izquierda de la pantalla principal.

<b>CASO DE PRUEBA</b>					
<b>Historia de Usuario</b>		HU5			
<b>Desarrollador</b>		Eva Ercilia Vázquez García.			
<b>Probador</b>		Eduardo Alejandro González Martell.			
<b>Fecha</b>					
<b>Combinaciones de valores de entrada</b>			<b>Escenario</b>	<b>Resultados Esperados</b>	<b>Resultados reales</b>
<b>CP</b>	<b>Nombres de Variables de entrada</b>	<b>Valor para cada variable de entrada</b>			
1	(Nombre, Primer Apellido, Segundo Apellido, Carnet de identidad, callePrincipalComboBox.getSelectedIndex(), numeroCasaComboBox.getSelectedIndex(), (aceptarBtn.getAction != null), (cancelarBtn.getAction != null), (JOptionPane.YES_OPTION))	(Eva, Vázquez, García, 02101081294, 2, 3, true, false, true)	Agregación exitosa del paciente.	Se crea un paciente con todos los datos insertados y se agrega al consultorio. Se sale del mensaje de diálogo y se regresa a la pantalla de AppPrincipal.	Se crea el paciente, se muestra el mensaje de diálogo y se regresa a la pantalla de AppPrincipal
2	(Nombre, Primer Apellido, Segundo Apellido, Carnet de identidad, callePrincipalComboBox.getSelectedIndex(), numeroCasaComboBox.getSelectedIndex(), (aceptarBtn.getAction != null), (cancelarBtn.getAction != null), (JOptionPane.YES_OPTION))	(Eva, Vázquez, García, 02101081294, 3, 4, true, false, false)	Agregación cancelada del paciente.	Se sale del mensaje de diálogo y se regresa a la pantalla de AgregarPaciente sin perder los datos insertados.	Se regresa a la pantalla AgregarPaciente
3	(... (cancelarBtn.getAction != null), (JOptionPane.YES_OPTION))	(... true, true)	Cancelación completa de la agregación del paciente.	Se cancela la agregación del paciente, se sale del mensaje de diálogo y se regresa a la pantalla de AppPrincipal.	Se cancela la agregación y se regresa a la pantalla de AppPrincipal
4	(...	(...	Cancelación	Se sale del	Se señalan

	(cancelarBtn.getAction != null), (JOptionPane.YES_OPTION) )	true, false )	n incompleta de la agregación del paciente.	mensaje de diálogo y se regresa a la pantalla de AgregarPaciente sin perder los datos insertados.	los errores y se se regresa a la pantalla de AgregarPac iente
5	(Nombre, Primer Apellido, Segundo Apellido, Carnet de identidad, callePrincipalComboBox.getSelectedIndex(), numeroCasaComboBox.getSelectedIndex(), (aceptarBtn.getAction != null), (cancelarBtn.getAction != null), )	(a, , wi, 1234567890, -1, -1, true, false, )	Campos vacíos o erróneos.	Se muestra el mensaje de error: “Compruebe los datos de los campos señalados”. Se regresa a la pantalla de AgregarPaciente y todos los componentes erróneos están en rojo.	Se muestran los campos erróneos y se muestra el mensaje de error, y se regresa a la pantalla de AgregarPac iente

Identificador	Nombre HU	Prioridad	Puntos estimados
HU6	Agregar visita		
<b>Usuario:</b> Médico.			
<b>Descripción:</b> Funcionalidad para agregar una visita de un paciente a la hoja de cargos.			
<b>Programador Responsable:</b> Eduardo Alejandro González Martell.			
<b>Escenarios</b>			
Nombre	Condición	Resultado esperado	
Agregación exitosa de la visita.	SI todos los datos se introducen correctamente (validarComponentes() == true) AND se ejecuta una Acción sobre el botón “Aceptar” (aceptarBtn) AND se ejecuta una Acción sobre el botón “Sí” (JOptionPane.YES_OPTION).	Se muestra el mensaje de diálogo: “¿Desea agregar la visita?” y al usuario marcar la opción “Sí”, se agrega la visita a la hoja de cargos correspondiente y se muestra el mensaje informativo: “La visita fue agregada correctamente”.	
Agregación cancelada de la visita.	SI todos los datos se introducen correctamente (validarComponentes() == true) AND se ejecuta una Acción sobre el botón “Aceptar” (aceptarBtn) AND (se ejecuta una Acción sobre el botón “No” (!JOptionPane.YES_OPTION) OR se ejecuta una Acción sobre el botón de salida).	Se muestra el mensaje de diálogo: “¿Desea agregar la visita?” y al usuario marcar la opción “No”, regresa a la pantalla para continuar con la agregación de la visita.	

Cancelación completa de la agregación de la visita.	SI se ejecuta una Acción sobre el botón “Cancelar” (cancelarBtn) AND se ejecuta una Acción sobre el botón “Sí” (JOptionPane.YES_OPTION).	Se muestra el mensaje de diálogo: “Los cambios no guardados se perderán ¿Desea continuar?” y al usuario marcar la opción “Sí”, se cancela la agregación del paciente y regresa a la pantalla anterior.
Cancelación incompleta de la agregación de la visita.	SI se ejecuta una Acción sobre el botón “Cancelar” (cancelarBtn) AND (se ejecuta una Acción sobre el botón “No” (!JOptionPane.YES_OPTION) OR se ejecuta una Acción sobre el botón de salida).	Se muestra el mensaje de diálogo: “Los cambios no guardados se perderán ¿Desea continuar?” y al usuario marcar la opción “No”, regresa a la pantalla para continuar con la agregación de la visita.
Campos vacíos o erróneos.	SI NOT todos los datos se introducen correctamente (validarComponentes() == false) AND se ejecuta una Acción sobre el botón “Aceptar” (aceptarBtn).	Al validar los componentes, se detectan los que están incorrectos, se ponen en rojo para ser identificados por el usuario y se muestra el siguiente mensaje de error: “Compruebe los datos de los campos señalados”.

#### Variables

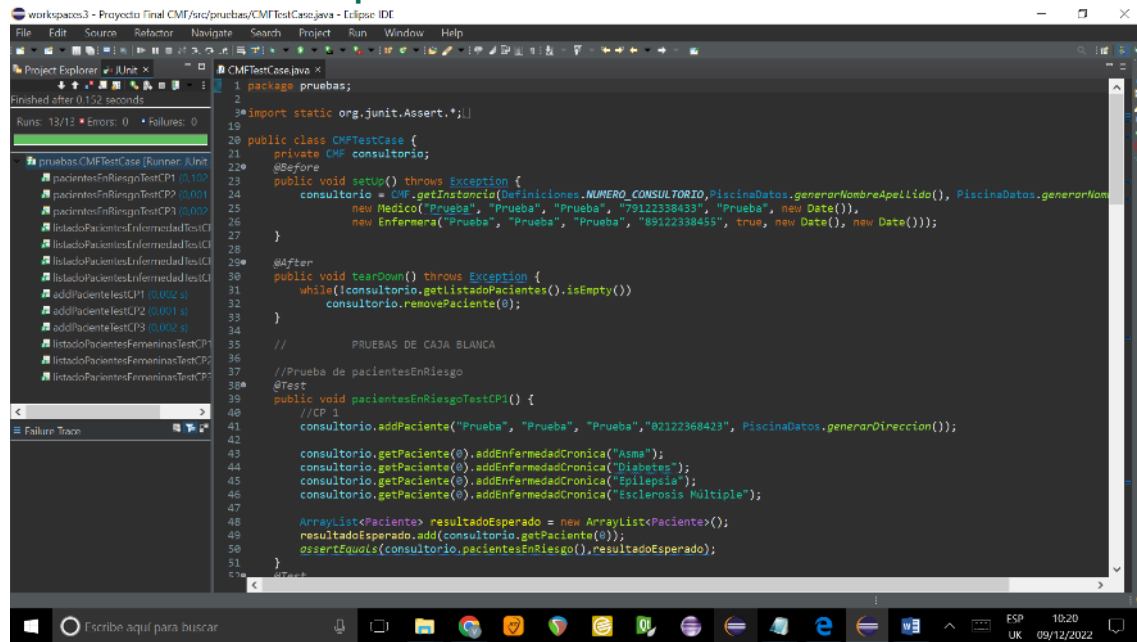
Nombre	Tipo o Dominio	Descripción
Diagnóstico	Cadena no vacía de entre 3 y 400 caracteres.	Nombre del paciente.
Tratamiento	Cadena no vacía de entre 3 y 400 caracteres.	Primer apellido del paciente.
seleccionarPaciente	Lista desplegable de opciones donde solo se puede seleccionar una.	Lista de los carnets de identidad de cada paciente perteneciente al consultorio, donde se debe escoger el que pertenece al paciente que está realizando la visita.
(aceptarBtn.getAction != null)	Condición.	Si no es null significa que el botón aceptarBtn ha sido seleccionado.
(cancelarBtn.getAction != null)	Condición.	Si no es null significa que el botón aceptarBtn ha sido seleccionado.
(JOptionPane.YES_OPTION)	Condición.	Para saber si fue seleccionado o no el botón “Sí” del mensaje de diálogo.

**Nota:** Los casos de prueba se realizarán insertando los siguientes datos en la pantalla donde se agrega una visita a la hoja de cargos del día. Para acceder a esta funcionalidad se debe acceder desde la cuenta de médico y seleccionar el botón “Agregar” en la opción Visitas que se encuentra a la izquierda de la pantalla principal.

<b>CASO DE PRUEBA</b>					
<b>Historia de Usuario</b>		HU6			
<b>Desarrollador</b>		Eduardo Alejandro González Martell.			
<b>Probador</b>		Eva Ercilia Vázquez García.			
<b>Fecha</b>					
<b>Combinaciones de valores de entrada</b>			<b>Escenario</b>	<b>Resultados Esperados</b>	<b>Resultados reales</b>
<b>CP</b>	<b>Nombres de Variables de entrada</b>	<b>Valor para cada variable de entrada</b>			
1	(Diagnostico,  Tratamiento,  seleccionarPaciente.getSelectedIndex(), (aceptarBtn.getAction != null), (cancelarBtn.getAction != null), (JOptionPane.YES_OPTION) )	(“Bronquitis aguda”, “Tomar abundante agua”, 3,  true, false, true )	Agregación exitosa de la visita.	Se crea una visita con todos los datos insertados y se agrega al consultorio. Se sale del mensaje de diálogo y se regresa a la pantalla de AppPrincipal.	Se crea una visita con los datos introducidos y se regresa a la pantalla de AppPrincipal.
2	(Diagnostico,  Tratamiento,  seleccionarPaciente.getSelectedIndex(), (aceptarBtn.getAction != null), (cancelarBtn.getAction != null), (JOptionPane.YES_OPTION) )	(“Bronquitis aguda”, “Tomar abundante agua”, 3,  true, false, false )	Agregación cancelada de la visita.	Se sale del mensaje de diálogo y se regresa a la pantalla de AgregarVisita sin perder los datos insertados.	Se regresa a la pantalla de AgregarVisita
3	(... (cancelarBtn.getAction != null), (JOptionPane.YES_OPTION) )	(... true, true )	Cancelación completa de la agregación de la visita.	Se cancela la agregación de la visita, se sale del mensaje de diálogo y se regresa a la pantalla de AppPrincipal.	Se regresa a la pantalla de AppPrincipal
4	(... (cancelarBtn.getAction != null),	(... true,	Cancelación incompleta	Se sale del mensaje de	Se regresa a la pantalla

	(JOptionPane.YES_OPTION) )	false )	de la agregación de la visita.	diálogo y se regresa a la pantalla de AgregarVisita sin perder los datos insertados.	de AgregarVisi ta
5	(Diagnostico, Tratamiento, seleccionarPaciente.getSelectedIndex(), (aceptarBtn.getAction != null), (cancelarBtn.getAction != null), )	("ab", " ", 3,  true, false, )	Campos vacíos o erróneos.	Se muestra el mensaje de error: "Compruebe los datos de los campos señalados". Se regresa a la pantalla de AgregarVisita y todos los componentes erróneos están en rojo.	Se muestra el mensaje de error con los campos erróneos señalados

## 2.8.3 Resultado de las pruebas



```
package pruebas;

import static org.junit.Assert.*;

public class CMFTestCase {
    private CMF consultorio;

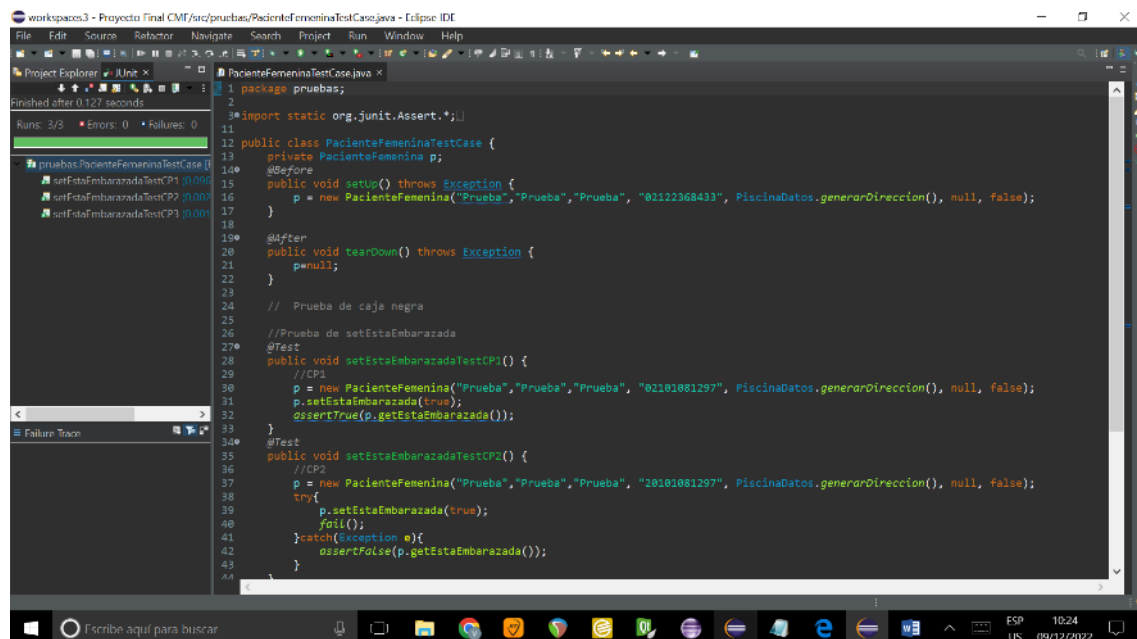
    @Before
    public void setup() throws Exception {
        consultorio = CMF.getInstance(Definiciones.NUMERO_CONSULTORIO, PiscinaDatos.generarNombreApellido(), PiscinaDatos.generarNombreMedico(), "Prueba", "Prueba", "Prueba", "79122338433", "Prueba", new Date(), new Enfermera("Prueba", "Prueba", "Prueba", "89122338455", true, new Date(), new Date()));
    }

    @After
    public void tearDown() throws Exception {
        while(!consultorio.getEstadoPacientes().isEmpty())
            consultorio.removePaciente(0);
    }

    // PRUEBAS DE CAJA BLANCA

    //Prueba de pacientesEnRiesgo
    @Test
    public void pacientesEnRiesgoTestCP1() {
        //CP 1
        consultorio.addPaciente("Prueba", "Prueba", "Prueba", "02122368423", PiscinaDatos.generarDireccion());
        consultorio.getPaciente(0).addEnfermedadCronica("Asma");
        consultorio.getPaciente(0).addEnfermedadCronica("Diabetes");
        consultorio.getPaciente(0).addEnfermedadCronica("Epilepsia");
        consultorio.getPaciente(0).addEnfermedadCronica("Esclerosis Multiple");
        ArrayList<Paciente> resultadoEsperado = new ArrayList<Paciente>();
        resultadoEsperado.add(consultorio.getPaciente(0));
        assertEquals(consultorio.pacientesEnRiesgo(), resultadoEsperado);
    }
}
```

**Prueba 1:** Pruebas desarrolladas en JUnit referentes a CMF.



```
package pruebas;

import static org.junit.Assert.*;

public class PacienteFemeninaTestCase {
    private PacienteFemenina p;

    @Before
    public void setup() throws Exception {
        p = new PacienteFemenina("Prueba", "Prueba", "Prueba", "02122368433", PiscinaDatos.generarDireccion(), null, false);
    }

    @After
    public void tearDown() throws Exception {
        p = null;
    }

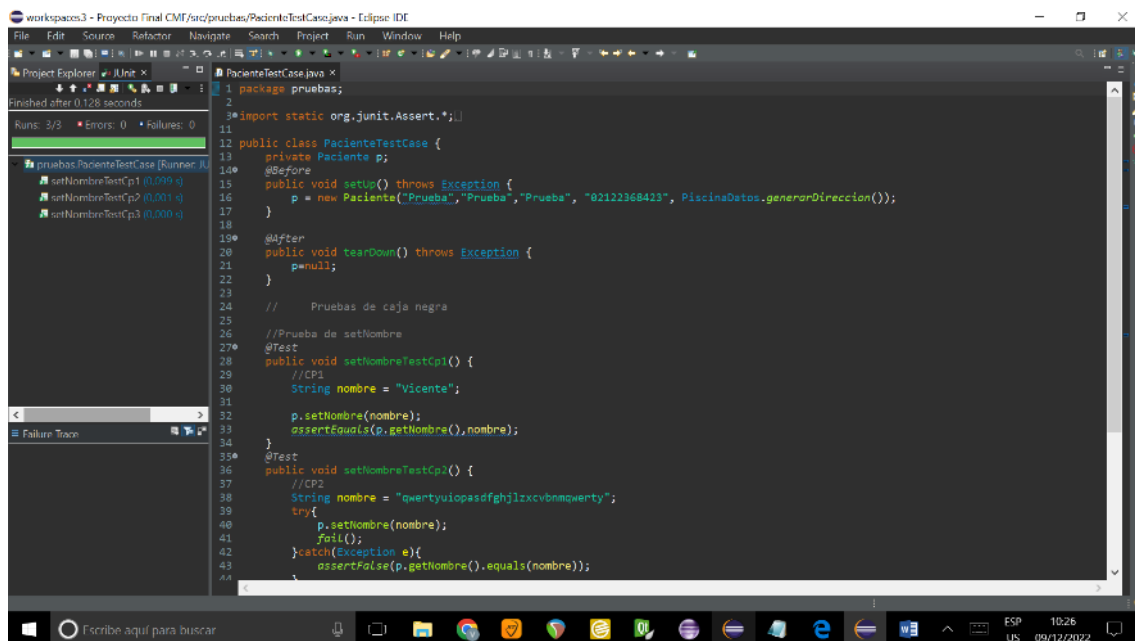
    // Prueba de caja negra

    //Prueba de setEstaEmbarazada
    @Test
    public void setEstaEmbarazadaTestCP1() {
        //CP1
        p = new PacienteFemenina("Prueba", "Prueba", "Prueba", "02101081297", PiscinaDatos.generarDireccion(), null, false);
        p.setEstaEmbarazada(true);
        assertTrue(p.getEstaEmbarazada());
    }

    @Test
    public void setEstaEmbarazadaTestCP2() {
        //CP2
        p = new PacienteFemenina("Prueba", "Prueba", "Prueba", "20101081297", PiscinaDatos.generarDireccion(), null, false);
        try {
            p.setEstaEmbarazada(true);
            fail();
        } catch (Exception e) {
            assertEquals(e.getMessage(), "setEstaEmbarazada()");
        }
    }
}
```

**Prueba 2:** Pruebas desarrolladas en JUnit referentes a PacienteFemenina.





### Prueba 3: Pruebas desarrolladas en JUnit referentes a Paciente.

Las pruebas realizadas a nivel de interfaz fueron satisfactorias.

## 2.9 Aspectos Novedosos

Para la confección del *software* se desarrollaron una serie de componentes tanto visuales como elementos lógicos, con un carácter novedoso. Los más destacables son:

- ❖ **Sección de definiciones:** Para otorgar una mayor configurabilidad al programa, se crearon definiciones referentes a diversos aspectos del mismo. Las mismas permitirán ajustar ciertos comportamientos del sistema para una mayor adaptabilidad en el futuro. De igual manera se creó un sistema para validar dichas definiciones, que permite comprobar antes de cargar el sistema, la validez de las definiciones, y si encuentra algún error, no permite la ejecución del programa.
- ❖ **Piscina de datos:** Se desarrolló un sistema para generar automáticamente y aleatoriamente los datos bases del programa, para así comprobar su correcta ejecución. Este sistema no solo permite inicializar el *software*, sino también define datos seleccionables como un sistema de direcciones (generado aleatoriamente), las vacunaciones, las enfermedades crónicas, los tipos de análisis y las especialidades remitidas.
- ❖ **Modelos de tablas y listas desplegables genéricas:** Se crearon un modelo de tablas genérico (Table Model) y un modelo de listas desplegables genérico (ComboBox Model). El primero otorga que con

solo la definición de las columnas y el comportamiento de agregar una fila, se habiliten ciertas funcionalidades como la actualización de la tabla conociendo solamente el listado de elementos, la organización de la tabla con el listado de elementos y un comparador, y la eliminación total de la información de la tabla. Por otra parte, con el modelo de listas desplegadas, solamente habrá que definir el comportamiento inicial; esto permitirá definir, por ejemplo, el primer valor de dicha lista.

- ❖ **Celda personalizada:** Para economizar en recursos se creó una celda de tablas que permite la muestra de textos de gran longitud, de manera que se pueda ver todo su contenido en una caja de texto con desplazamiento vertical que ajusta el mismo a las dimensiones de la celda.
- ❖ **Tablas reordenables:** Para una mejor lectura y comprensión de los contenidos dispuestos en las tablas, se programaron las mismas para que el usuario pudiera reordenarlas de forma ascendente y descendente de acuerdo al contenido de cada columna.
- ❖ **Trabajo con multi-hilo:** Se emplearon las facilidades que otorga el lenguaje de programación Java para el trabajo con multi-hilo. Esto se empleó para la realización de algunas animaciones como es el caso del panel de la pantalla de carga, y para la inicialización de los datos al mismo tiempo que se muestra la pantalla de carga.
- ❖ **Métodos Auxiliares:** Se desarrollaron un cierto número de métodos auxiliares que resultaron de utilidad para el desarrollo del *software*. Se crearon métodos para la extracción de información del carnet de identidad y el trabajo con fechas y listas.
- ❖ **Validaciones:** Para validar los datos de la aplicación, se creó un sistema de métodos que permitieran realizar dicha tarea. Se desarrollaron validaciones para cadenas de texto (String), números, fechas, trabajo con listas y demás.
- ❖ **Componentes Visuales personalizados:** Para el diseño de la interfaz se desarrollaron una serie de componentes visuales, que permitieron mejorar la estética de la aplicación, así como la disminución de la introducción de posibles errores. Los más destacables son un panel de animación, empleado en la pantalla de carga, un campo de texto ajustable que permite validar lo introducido en dependencia del comportamiento seleccionado, componentes para renderizar imágenes, avatares circulares, paneles gradientes, y un botón ovalado con animación.
- ❖ **“Motor de búsqueda” de pacientes:** Como funcionalidad extra, se desarrolló un motor de búsqueda de pacientes, que dado un listado de pacientes, y los parámetros a buscar, devuelve el listado filtrado con los pacientes que cumplan las precondiciones introducidas. Se decidió que los posibles parámetros a filtrar sean el nombre, primer apellido, segundo

apellido, carnet de identidad, número de historia clínica y la dirección (calle principal y número de casa).

- ❖ **Sistema de Comparadores:** El lenguaje de programación Java permite el ordenamiento de las listas, pudiendo definir su comportamiento, ya sea a través de la definición del “orden natural” de los elementos de la lista (implementando la interfaz Comparable), o la definición personalizada del comportamiento del orden (creando clases que implementen la interfaz Comparator). Se definieron un número de comparadores que permitieron el ordenamiento de las listas a partir de ciertos criterios, como son (en el caso de los pacientes) el nombre, el carnet de identidad, la dirección y el número de historia clínica.

## Conclusiones

Tras la culminación del proceso de desarrollo del proyecto se arribaron a las siguientes consideraciones finales:

- ❖ Se dio cumplimiento al objetivo principal trazado: desarrollar una aplicación de escritorio en el lenguaje de programación Java, cumpliendo con el paradigma de la programación orientada a objetos.
- ❖ Se elaboró un plan de trabajo donde se organizaron las tareas a realizar, en vistas a un desarrollo del proyecto más limpio y eficiente.
- ❖ Se elaboraron las tarjetas CRC, debido a su carácter versátil y sencillo para tener una imagen general del problema.
- ❖ Se elaboró un diagrama de clases bajo una perspectiva más detallada del programa a implementar.
- ❖ Se implementó el patrón diseño de la programación orientada a objetos **singleton**, el cual permitió la creación de una sola instancia de un objeto en todo el programa, habilitando además el acceso global al mismo.
- ❖ La interfaz gráfica fue desarrollada tomando en cuanto los patrones de diseño y los principios y reglas de oro, de forma tal que resultara en una aplicación funcional, simple e intuitiva de usar.
- ❖ Se diseñaron y ejecutaron pruebas de *software* para determinar el correcto funcionamiento de la lógica del programa.
- ❖ Como resultado del proyecto de curso se comenzó a desarrollar una api personalizada, con componentes visuales, validaciones e utilidades. La misma puede ser empleada en el desarrollo de los próximos trabajos de cursos, para agilizar su desarrollo. Dicha api, a pesar de estar aún en desarrollo, puede accederse a ella y a su documentación en <https://github.com/EduardoProfe666/Proyecto-API-Personalizada>

## Recomendaciones

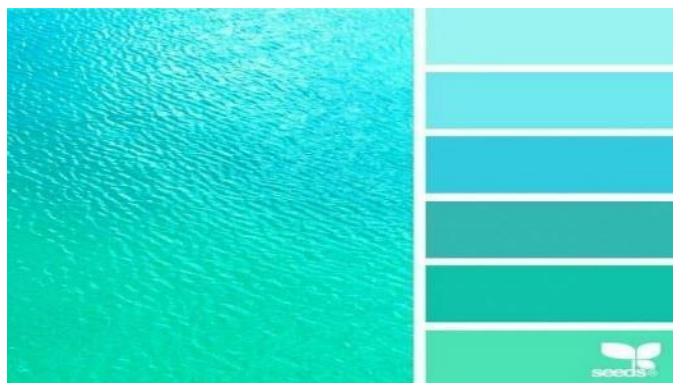
La solución informática propuesta hace caso omiso a ciertas particularidades y especificidades puesto que se desarrolló con el objetivo de dar solución a la problemática específica dada. Por ello es posible mejorar ciertos aspectos del *software*:

1. **Cambiar la perspectiva de la solución:** Debido al enunciado, se decidió que fueran los pacientes los que contuvieran su respectiva historia clínica, lo cual en la realidad no es correcto, pues las historias clínicas son el modelo general de la información del paciente. Por tanto, para una mejor adaptabilidad de la solución a la realidad, se puede cambiar la perspectiva usada.
2. **Validación completa del carnet de identidad:** Para la validación de los carnets de identidad de los pacientes, no se logró validar el último dígito (11no), puesto que es un dígito de control generado automáticamente por un algoritmo matemático cuando los demás dígitos son creados. Este algoritmo no está disponible para el público puesto que es una de las tantas garantías existentes para impedir la falsificación de identidades. Se podría en un futuro vincular el programa con otro, acreditado por el Minint, para la validación de los carnets de identidad, que no solo compruebe la validez del carnet, sino también su existencia.
3. **Implementar un sistema de base de datos:** Uno de los principales inconvenientes del presente *software* es que los datos se reinician con los valores predeterminados al reiniciar el programa. Este problema no es objetivo de estudio de la asignatura del presente trabajo de curso, pero es un inconveniente importante si se va a concretar la solución. Por ello, se hace necesario la implementación de un sistema de base de datos para garantizar, no solo el guardado correcto de los datos, sino también incrementar su seguridad.
4. **Completar los tipos de datos seleccionables:** Existen datos que son seleccionables y están previamente definidos. Entre los mismos se encuentran las **vacunaciones**, las **enfermedades crónicas**, los **tipos de análisis** y las **especialidades remitidas**. Se seleccionaron un conjunto básico de cada uno de estos datos para la prueba del funcionamiento correcto del programa. Se podría implementar un sistema completo de los mismos para aumentar la funcionalidad del *software*.
5. **Agregar funcionalidades:** A la hora de determinar los requisitos del sistema, solamente se tuvieron en cuenta aquellos para dar solución a la problemática dada. En un futuro, se pudieran implementar otras funcionalidades que den solución a escenarios reales, como son un sistema para el guardado de los resultados de los “terrenos” o recorridos realizados por el doctor del consultorio y un sistema para generar documentos habitualmente creados por el personal médico del consultorio, como por ejemplo el chequeo pre-empleo, las recetas médicas, certificados médicos y certificados de medicamentos.

## Bibliografía

- [1] J. A. L. Espinosa, "Enero 4 de 1984. Inicio del plan del médico y la enfermera de la familia," *ACIMED*, vol. 16, no. 6, 2007.
- [2] Minsap. (2021, 17/11/2022). *Plan de Inmunización*. Available: <http://www.bvv.sld.cu>
- [3] M. Plus. (2020, 10/11/2022). *Comunicarse con los demás - al vivir con una enfermedad crónica*. Available: <https://medlineplus.gov/spanish/ency/patientinstructions/000602.htm>
- [4] cubamundomedico. (2019, 10/12/2022). *Estudio de Especialidades Médicas en Cuba*. Available: <https://www.cubamundomedico.com/es/estudio-de-especialidades-medicas-en-cuba>
- [5] J. Tidwell, *Designing Interfaces*, 2 ed. Sebastopol, CA: O'Reilly Media, 2011.
- [6] R. S. Pressman, *Ingeniería de Software Parte 1*, 2 ed. La Habana: Felix Varela, 2005.

## Anexos



**Anexo 1:** Esquema de colores empleado.