	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Dra. Rocío Alejandra Aldeco Pérez

Asignatura: Programación orientada a objetos -1323

Grupo: 6

No de Práctica(s): 9

Integrante(s): Mendoza Hernández Carlos Emiliano
Roa Díaz Vanessa

No. de Equipo de cómputo empleado:

No. de Lista o Brigada:

Semestre: 2023-1

Fecha de entrega: 7 de noviembre del 2022

Observaciones:

CALIFICACIÓN: _____

Práctica 9.

UML

OBJETIVO

- Utilizar UML como herramienta para diseñar soluciones de software para un lenguaje de programación orientada a objetos.

ACTIVIDADES

- Elegir el o los diagrama(s) necesarios para mostrar la solución de un problema.
- Crear diagramas UML para mostrar la solución a un problema.

INTRODUCCIÓN

El **Lenguaje de Modelado Unificado** (**UML** – Unified Modeling Language) es un lenguaje gráfico que permite visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software.

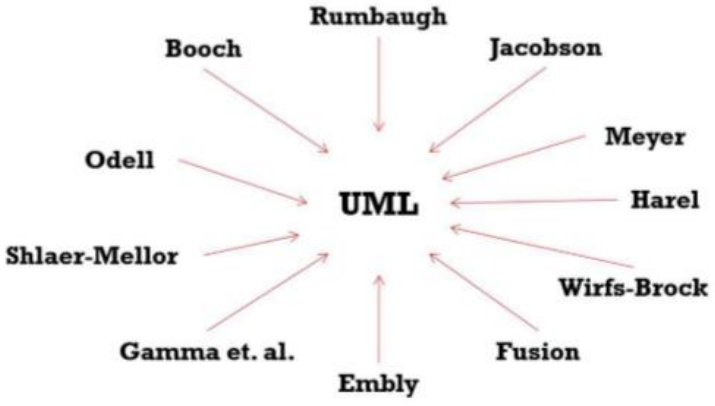


Figura 1. Representación de los lenguajes que componen a UML.

Los **diagramas UML** permiten modelar aspectos conceptuales como procesos de negocios o funcionalidades de un sistema, cumpliendo con los siguientes objetivos:

- *Visualizar*: expresar de forma gráfica la solución y/o flujo del proceso o sistema.
- *Especificar*: mostrar las características del sistema.
- *Construir*: generar soluciones de software.
- *Documentar*: especificar la solución implementada.

UML está compuesto por tres bloques generales de formas:

- *Elementos*: son representaciones de entes reales (usuarios) o abstractos (objetos, acciones, clases, etc.).
- *Relaciones*: es la unión e interacción entre los diferentes elementos.
- *Diagramas*: muestra a todos los elementos con sus relaciones.

Diseño estático o de estructura

Los **diagramas estáticos o estructurales** aportan una visión fija del sistema, los diagramas que permiten modelar estas características para un lenguaje orientado a objetos son:

- Diagrama de casos de uso
- Diagrama de clases
- Diagrama de objetos

Diseño estático o de estructura

Los **diagramas dinámicos o de comportamiento** permiten visualizar la comunicación entre elementos del sistema para un proceso específico, los diagramas que permiten modelar estas características para un lenguaje orientado a objetos son:

- Diagrama de estados
- Diagrama de actividades
- Diagrama de interacción

INSTRUCCIONES

Realiza las siguientes actividades después de leer y revisar en clase la *Práctica de Estudio 9: UML.*

Caso de estudio – Clínica veterinaria

Imagina que un veterinario te solicita el desarrollo de un sistema que le permita administrar su clínica veterinaria. Para esto, un colega tuyo visitó al veterinario y recabó los siguientes requerimientos escritos en forma de prosa.

Un veterinario tiene como pacientes animales y como clientes personas. Un cliente es una persona que tiene un identificador único de cliente, un apellido paterno, un apellido materno y nombre(s), un número de cuenta bancaria, una dirección, un teléfono y un correo electrónico. Los clientes pueden tener varias mascotas, cada mascota tiene un identificador único de mascota, un nombre, una especie, una raza, color de pelo, fecha de nacimiento aproximada, peso medio del animal en las últimas 10 visitas y el peso actual del animal. Asimismo, se guardará un historial médico con cada enfermedad que tuvo y la fecha en la que enfermó. Adicionalmente, cada mascota tiene un calendario de vacunación, en el que se registrará la fecha de cada vacuna, la enfermedad de la que se vacuna.

El veterinario debe realizar las siguientes acciones:

- **a)** CRUD de clientes
- **b)** CRUD de pacientes
- **c)** Visualizar el historial médico de un paciente
- **d)** Registrar visitas en el historial médico de un paciente
- **e)** Registrar el pago de un cliente después de una visita
- **f)** Enviar de manera automática un correo electrónico al cliente una semana antes de la fecha de vacunación de un paciente.

Usando como referencia este caso de estudio, realiza los siguientes diagramas de UML que representen la estructura estática y dinámica de la solución de este caso.

1. Diagrama de casos de uso.
2. Diagrama de clases (uno sólo) que muestre la relación que existe entre todas las clases del programa.
3. Diagrama de estados del método que registrará el peso medio del paciente.
4. Diagrama de secuencia de la acción F.

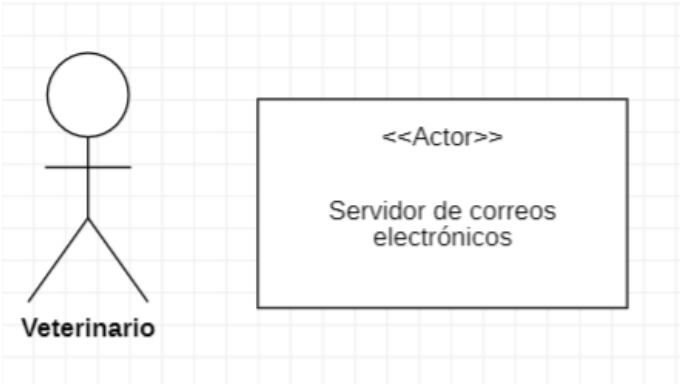
DIAGRAMA DE CASOS DE USO

1. Identificando actores

De acuerdo con la descripción de los requerimientos del programa, se pueden identificar lo siguiente:

Actores:

- Veterinario (responsable de la administración del sistema y los casos de uso principales).
- Servidor de correos electrónicos (sistema externo que se usará para comunicarse con los clientes).



2. Identificando casos de uso

Los requerimientos del cliente son:

- CRUD de clientes

- CRUD de pacientes
- Visualizar historial médico de un paciente
- Registrar visitas en el historial médico de un paciente
- Registrar el pago de un cliente después de una visita
- Enviar de manera automática correo electrónico al cliente una semana antes de la fecha de vacunación de un paciente

Podemos notar que es posible establecer relaciones entre casos de uso.

Primero, podemos clasificar los casos de uso en 4 subconjuntos de casos de uso: dar de alta, consultar, actualizar y eliminar (CRUD). Sabemos que los objetos de interés para este programa son clientes (personas) y pacientes (mascotas), por lo tanto, una generalización de cada uno de estos subconjuntos es apropiada. Además, es posible generalizar estos 4 subconjuntos aún más en un caso de uso *Gestionar información*. Se pretende que el caso de uso *Gestionar información* tenga una relación de herencia con los 4 casos de uso antes mencionados: *Dar de* alta, *Consultar*, *Actualizar* y *Eliminar* (todos ellos siendo casos de uso abstractos). De cada uno de los cuatro casos de uso planteados (CRUD) es posible especializar los casos de uso dependiendo si se trata de clientes o pacientes.

Luego, podemos ver los siguientes tres requerimientos como casos de uso derivados de los 4 que ya se establecieron. En el siguiente punto se analizarán sus relaciones.

Por último, se identifica otro caso de uso para enviar correos automáticamente a los clientes. Por lo tanto, se tiene lo siguiente:

Casos de uso:

- *Gestionar información* (abstracto)
  - *Dar de alta* (abstracto)
    - Dar de alta cliente
    - Dar de alta paciente
  - *Consultar* (abstracto)
    - Consultar datos cliente
    - Consultar datos paciente
  - *Actualizar* (abstracto)
    - Actualizar datos cliente
    - Actualizar datos paciente
  - *Eliminar* (abstracto)
    - Eliminar paciente
    - Eliminar cliente
- Registrar visita
- Registrar pago
- Consultar historial médico
- Notificar vacunación próxima

3. Estableciendo relaciones

Asociaciones entre actores y casos de uso:

- Veterinario
  - *Gestionar información*
  - Registrar visita
  - Registrar pago
  - Consultar historial médico

- Servidor de correos electrónicos
    - Notificar vacunación próxima

Asociaciones entre casos de uso:

- El caso de uso *Gestionar información* hereda a los casos de uso *Dar de alta*, *Actualizar*, *Consultar* y *Eliminar*.
- *Dar de alta* hereda a los casos de uso “Dar de alta paciente” y “Dar de alta cliente”. *Dar de alta* implica siempre al caso de uso “Registrar visita” (puesto que siempre se debe registrar una visita si un cliente es nuevo o lleva a una mascota nueva).
- *Actualizar* hereda a los casos de uso “Actualizar datos paciente” y “Actualizar datos cliente”. “Actualizar datos cliente” puede implicar al caso de uso “Registrar pago” (dado que uno de los datos que se puede actualizar de un cliente es un pago realizado).
- *Consultar* hereda a los casos de uso “Consultar datos paciente” y “Consultar datos cliente”. “Consultar datos paciente” puede implicar al caso de uso “Consultar historial médico” (ya que uno de los datos que se puede consultar de un paciente es su historial médico).
- *Eliminar* hereda a los casos de uso “Eliminar paciente” y “Eliminar cliente”.

Con el anterior análisis se obtuvo el siguiente diagrama

Diagrama de casos de uso:

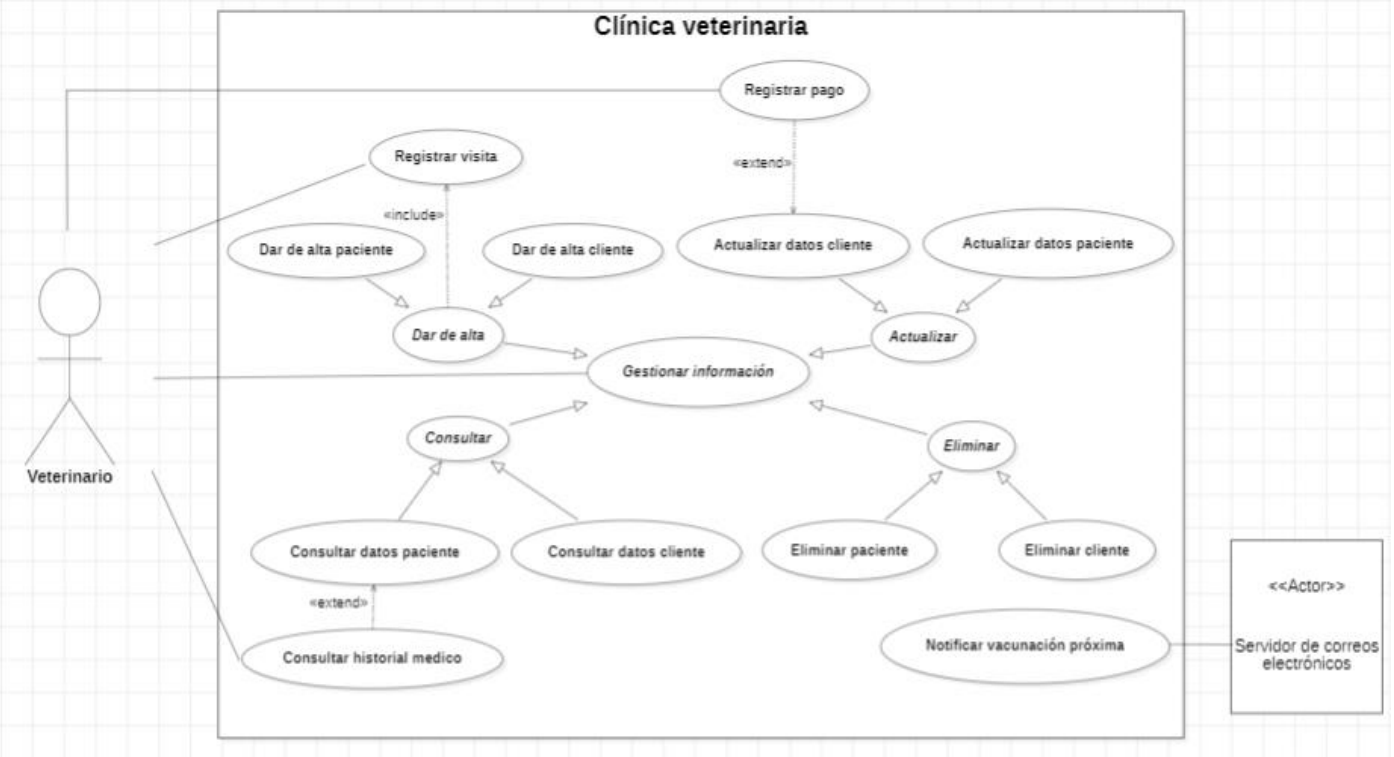


DIAGRAMA DE CLASES

1. Identificando las clases

Clases:

- Cliente
- Mascota
- HistorialVisitas
- HistorialMedico
- CartillaVacunación
- Visita
- Consulta
- Vacuna
- GestorCorreos
- Fecha

2. Identificando los atributos

Atributos:

- Cliente
  - – ID : String {readOnly}
  - – aPaterno : String
  - – aMaterno : String
  - – nombre : String
  - – noDeCuenta : String
  - – direccion : String
  - – telefono : String
  - – email : String
  - – mascotas : Mascota [1…*] {unique}
  - – historialVisitas : HistorialVisitas

- Mascota
  - – ID : String {readOnly}
  - – nombre : String
  - – especie : String
  - – raza : String
  - – colorPelo : String
  - – fechaNac : Fecha
  - – pesoMedio : float
  - – pesoActual : float
  - – historialMed : HistorialMedico
  - – cartillaVac : CartillaVacunacion

- HistorialVisitas
  - – visitas : Visita [*] {unique, ordered}

- HistorialMedico
  - – consultas : Consulta [*] {unique, ordered}

- CartillaVacunacion
  - – vacunas : Vacuna [*] {ordered}

- Visita
  - – fecha : Fecha
  - – cliente : Cliente
  - – paciente : Mascota
  - – statusPagado : boolean
  - – cantidad : float

- Consulta
  - – fecha : Fecha
  - – paciente : Mascota
  - – motivo : String
  - – pesoMedido : float

- Vacuna
  - – fecha : Fecha
  - – proximaVacuna : Fecha
  - – paciente : Mascota
  - – tipo : String

- GestorCorreos
  - – remitente : String

- Fecha
  - – día : int {readOnly}
  - – mes : int {readOnly}
  - – anio : int {readOnly}

3. Identificando los métodos

- Cliente
  - + registrarMascota() : void
  - + ConsultarHistorialVisitas() : HistorialVisitas
  - + actualizarDatos() : void

- Mascota
  - + actualizarDatos() : void
  - + consultarHistorialMed() : HistorialMedico
  - + getPesoMedio() : float

- HistorialVisitas
  - + registrarVisita() :void
  - + registrarPago() : void

- HistorialMedico
  - + registrarConsulta():

- CartillaVacunacion
  - + agregarVacuna : void

- GestorCorreos
  - + enviarCorreo() : void

4. Identificando las relaciones entre clases

Para definir las relaciones entre clases es necesario aclarar que el nivel de abstracción presente es acorde a como se espera que funcione el programa. En este caso la clase *Cliente* tiene una relación de composición con la clase *Mascota*; si no existe un cliente no existe por lo tanto una mascota (en un nivel de abstracción bajo podría parecer contraintuitivo que no exista una mascota sin su dueño, pero para este programa, si no hay un registro de un cliente, no se crea el registro de mascota, y de igual forma si se elimina la instancia de un cliente, se elimina también el registro de sus mascotas). En esta composición, un cliente puede tener 1 o muchas mascotas, pero una mascota está asociada a un solo cliente.

Además, la clase *Cliente* tiene también una relación de composición con la clase *HistorialVisitas* (siguiendo los mismos criterios que la composición anterior). Un cliente tiene un solo historial de visitas y un historial de visitas pertenece a un solo cliente. A su vez, la clase *HistorialVisitas* tiene una composición con *Visita*, donde un historial de visitas tiene de 0 a muchas visitas, pero una visita pertenece a un solo historial.

Por otra parte, la clase *Mascota* tiene composiciones con las clases *HistorialMedico* y *CartillaVacunacion*. Para su multiplicidad: una mascota tiene un solo historial medico y solo una cartilla de vacunación, pero un historial médico y una cartilla de vacunación están asociadas a solo una mascota.

Adicionalmente, las clases *HistorialMedico* y *CartillaVacunacion* tienen composiciones con las clases *Consulta* y *Vacuna* respectivamente. Sucede de manera análoga a la composición entre *HistorialVisitas* y *Visita*.

La clase *GestorCorreos* es una clase de asociación, cuya operación *enviarCorreo* depende de la asociación entre *CartillaVacunacion* y *Vacuna* (al registrar la aplicación de una vacuna, la fecha de la próxima aplicación de una vacuna se agenda para enviar un recordatorio).

Con el anterior análisis se obtuvo el siguiente diagrama

Diagrama de clases:

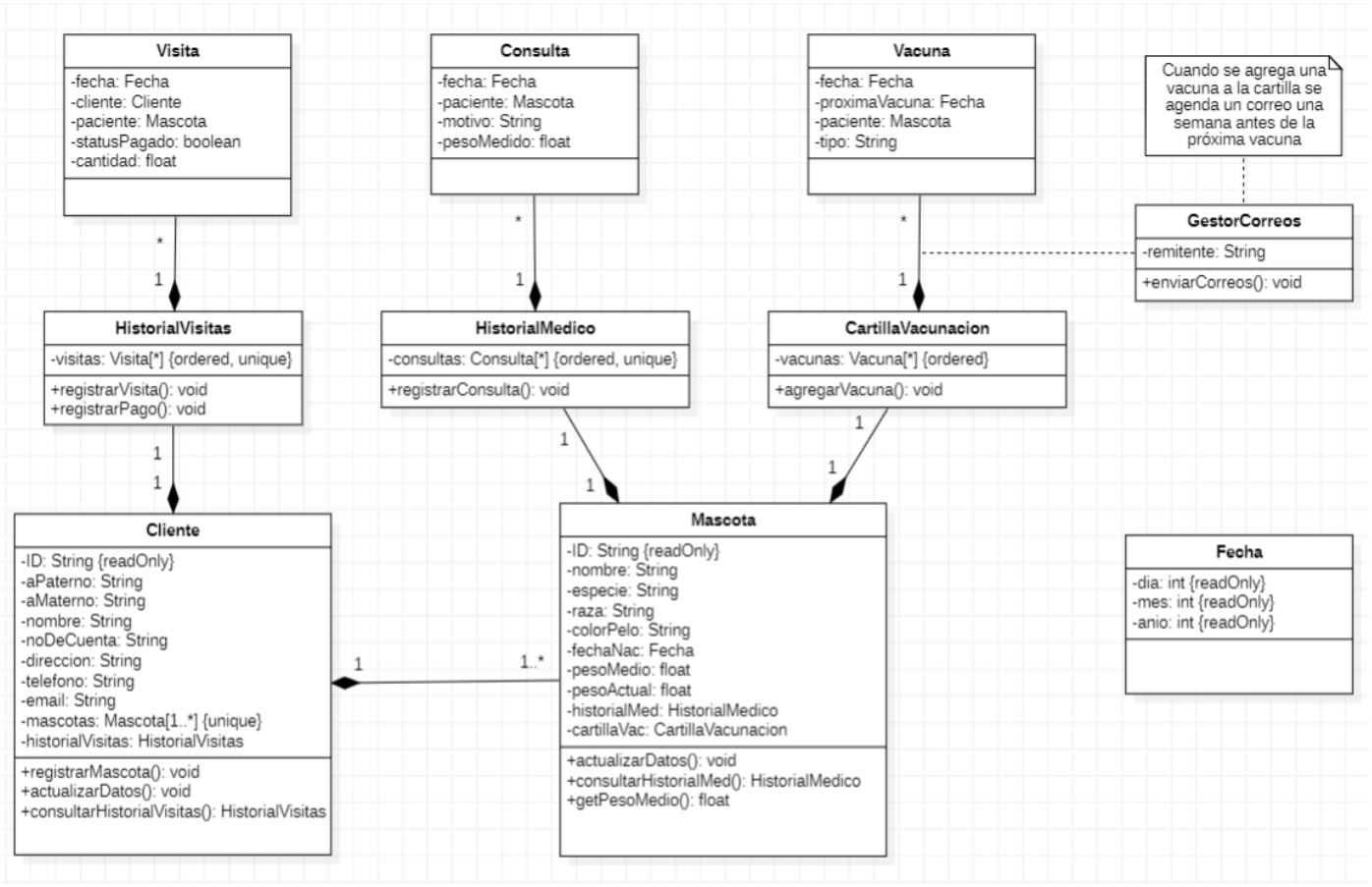
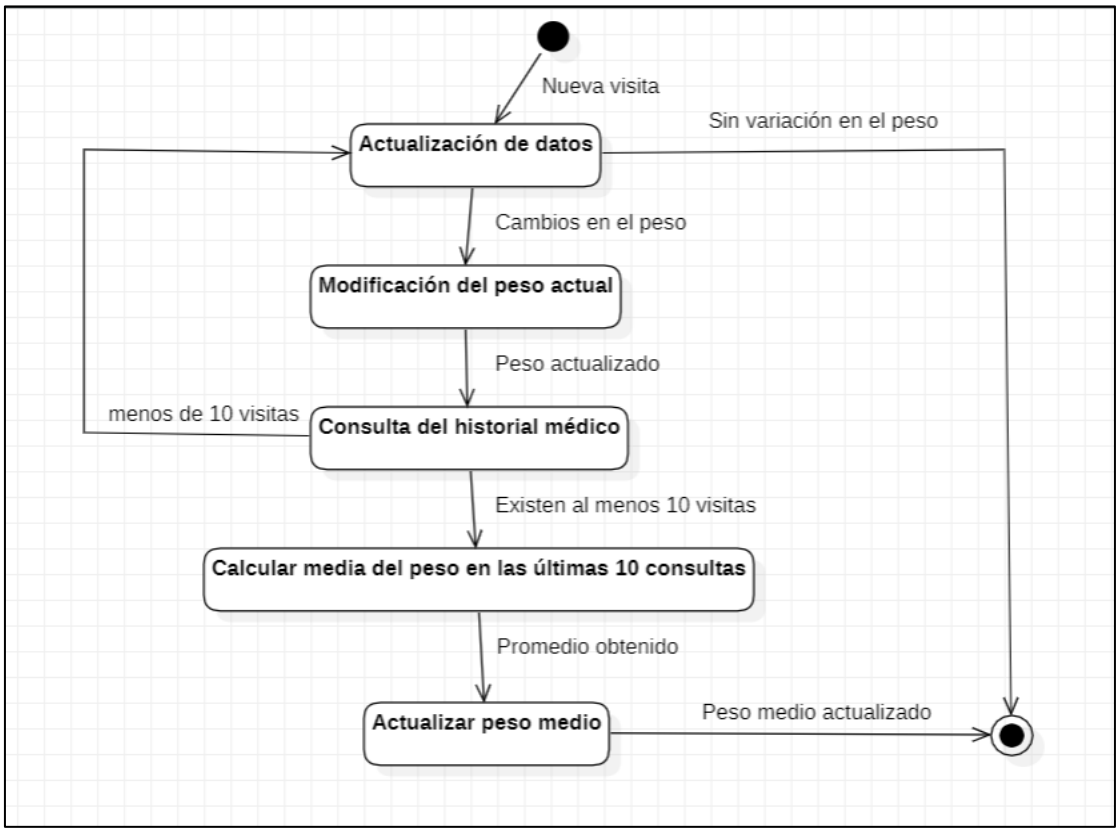


DIAGRAMA DE ESTADOS

Método de registro del peso medio del paciente

Con el fin de crear el diagrama de estados para el método encargado de registrar el peso medio perteneciente a la clase Mascota analizada en el diagrama de clases, se debe tener en cuenta especificaciones indispensables.



1. Inicio: Para cualquier diagrama de estado se debe señalar el nodo del cual se parte el orden de sucesión de los estados. Este nodo es representado con un círculo negro, el cual se encuentra en la parte superior de nuestro diagrama. Para poder continuar con la secuencia en los estados del objeto, que en este caso sería de tipo Mascota, debemos considerar que la primera transición que debe ocurrir de forma definitiva es que exista

una nueva consulta, ya que de esta procede el hecho de efectuar la toma del peso actual del paciente y de esta acción, surge la actualización del peso medio como consecuencia.

2. Estados: En este tipo de diagrama UML, los estados son las partes indispensables para la comprensión de las etapas del objeto en el proceso de ejecución del método, los cuales se encuentran representados por recuadros con esquinas redondeadas. En el caso de este diagrama, contamos con 5 estados sin contar el inicial y final:

    - Actualización de datos: La actualización de datos es posible gracias al método actualizarDatos() de tipo void que pertenece a la clase Mascota, clase cuya creación permitiría la representación de los pacientes del veterinario.
    - Modificación del peso actual: La configuración del atributo pesoActual se efectúa mediante el uso del método actualizarDatos(), el cual se usa por cada consulta nueva por paciente que se registra.
    - Consulta del historial médico: La revisión del historial de consultas de un paciente se puede realizar por medio de otro método de la clase Mascota denominado consultarHistorialMed().
    - Calcular media del peso en las últimas 10 consultas: Al revisar el historial médico del paciente es posible acceder a cada uno de los pesos medidos de las consultas que se le han realizado, esto es accediendo al conjunto de consultas que constituye uno de los atributos de la clase HistorialMedico; a su vez cada una de las consultas tiene por atributo un peso medido que corresponde al peso del paciente. Se calcula el peso medio del animal tomando como referencia las últimas 10 consultas para obtener la media los valores.
    - Actualizar peso medio: Ya obtenida la media de los 10 pesos indicados, debe actualizarse el valor del peso medio, que es atributo de la clase Mascota (representación de los pacientes), para ello se recurriría nuevamente al método actualizarDatos() de la clase.

3. Transiciones: En este caso se cuentan con 8 transiciones. Las transiciones permiten representar las comunicaciones entre los estados del diagrama. Para explicar a profundidad lo anterior podemos distinguir una de las transiciones del proceso en el método analizado: “Cambios en el peso”. Si ocurre que, al medir el peso actual del paciente, existen cambios en este, entonces se procede a realizar la actualización de su peso, en caso contrario, es decir, localizándonos en la transición “sin variación en el peso”, entonces se llega al estado final del objeto, pues al no existir variaciones en el peso actual, no existirán modificaciones realmente significativas en el peso medio.

4. Fin: Para los diagramas de estado es necesario indicar la etapa en la que el ciclo de vida de un objeto finaliza, para lo cual se recurre al uso de un símbolo particular: este se conforma por un círculo blanco en el cual se localiza un círculo negro en el centro. Para el caso del método de estudio, el estado final del objeto ocurre cuando se realiza actualización o registro del peso medio del objeto.

DIAGRAMA DE SECUENCIA

Enviar de manera automática correo electrónico al cliente una semana antes de la fecha de vacunación de un paciente

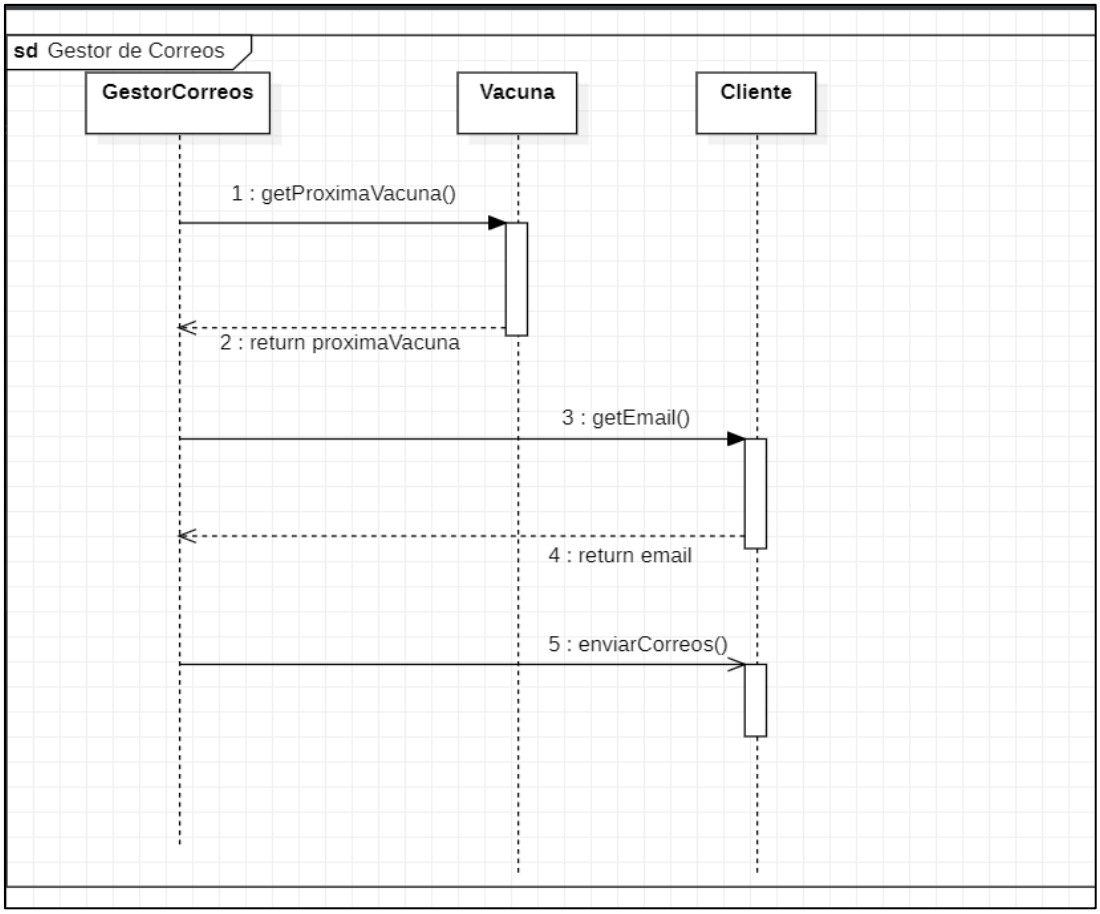
El siguiente diagrama presentado presenta la secuencia en la que se solicitan los mensajes a partir de un objeto GestorCorreos hacia otras dos clases: Vacuna y Cliente. Para poder realizar el envío de manera automática del recordatorio de la próxima vacuna por correo electrónico al cliente una semana antes de la fecha establecida, es necesario recurrir al uso de la clase GestorCorreos, la cual representa el sistema que nos permitirá realizar este envío mediante el uso del método enviarCorreos() ya indicado en el diagrama de clases.

De forma general, se sigue la siguiente secuencia:

1. El Objeto de clase GestorCorreos debe solicitar la fecha de la próxima vacuna, ya que, a partir de esta, es como se decide si enviar el recordatorio al cliente, por lo cual solicita está información mediante el uso de un mensaje sincrónico a un objeto Vacuna, en la cual existe un atributo designado para el almacenamiento de la próxima fecha de vacunación (proximaVacuna). Se dice que es un mensaje de tipo sincrónico ya que el remitente (objeto GestorCorreos) espera una respuesta del mensaje expedido, esta respuesta es dada por el objeto Vacuna, ya que regresa el valor de la fecha solicitado.
2. Una vez teniendo la fecha de vacunación próxima y que se verifique que falta una semana para la próxima aplicación de vacuna a un paciente, el objeto GestorCorreos manda un mensaje sincrónico al objeto Cliente (que tiene por atributo al paciente) solicitando el correo electrónico de este, el cual se encuentra almacenado en el atributo “email” del objeto. Al ser un mensaje sincrónico, se espera una respuesta por parte del objeto receptor, de forma que el objeto Cliente otorga la información solicitada de vuelta.

3. Ya obtenido el correo electrónico del cliente se recurre al envió de otro mensaje al objeto cliente, del cual esta vez no se espera ninguna respuesta, es por ello por lo que se trata de

un mensaje asíncrono, en el que se efectúa la acción del envió del recordatorio al cliente de la próxima visita para la aplicación de la vacuna indicada.



Nótese que la representación de mensajes síncronos se indica mediante el uso de flechas con línea continua y una punta de flecha sólida y de los cuales se indica necesariamente la respuesta esperada utilizando una flecha de línea discontinua y una punta de flecha simple. La simbolización de los mensajes asíncronos se señala usando una flecha de línea continua con una punta de flecha simple.

CONCLUSIONES

A través de la realización de esta práctica tuvimos la oportunidad de estudiar, analizar y desarrollar diagramas en el lenguaje unificado de modelado, mejor conocido como UML con el fin de representar de manera visual con menor complejidad y mayor simplicidad, el desarrollo de software de un sistema. El uso de un lenguaje visual como lo es UML nos permite mostrar con mayor facilidad la comunicación entre los componentes de un sistema computacional a terceras personas, es decir, aquellas que no se encuentran familiarizadas con el uso de los lenguajes con los que nosotros como programadores si lo estamos, es por ello que se dice que el manejo de UML en proyectos de creación de software tiene como una de sus ventajas más destacables el hacer la comprensión de un sistema netamente informático, accesible para cualquier persona. Por lo anterior, es que es indispensable conocer cada uno de los tipos y usos de los diagramas de UML, pues como desarrolladores que posiblemente serviremos a negocios empresariales, es inevitable no recurrir a estos para poder mostrar nuestras ideas, implementaciones o componentes de nuestros sistemas a miembros del proyecto que no necesariamente son expertos en programación. Particularmente, en el paradigma orientado a objetos en el cual nos basamos para desarrollar cada uno de los diagramas presentados en esta archivo, se reconoce una estrecha relación entre UML y la programación orientada a objetos, pues a través de este lenguaje de modelado podemos crear diversos tipos de diagramas dependiendo del tipo de información o relaciones que se desean representar, por ejemplo, si se requiere indicar las clases y la forma en la que se relacionan en un determinado programa, se recurre a la utilización de un diagrama de clases, por otra parte, si se quisiese señalar cada uno de los casos de uso, es decir, cada una de las acciones o actividades que pueden efectuar actores determinados, se puede apelar al empleo de los diagramas de casos de uso.

Una de las propiedades valiosas que pudimos notar en el lenguaje unificado de modelado es que en él se cuenta con dos tipos de diagramas principalmente: diagramas estáticos o de estructura y diagramas dinámicos o de comportamiento: los primeros proporcionan una visión de la parte invariable de un sistema mientras que los segundos ofrecen una visualización de la manera en la que se comunican los elementos del sistema.

En general, la realización de cada uno de los diagramas presentes, creados a partir de los requerimientos indicados, nos posibilitó el desarrollo de nuevas competencias en cuanto al desarrollo de proyectos computacionales se refiere, además de conseguir herramientas útiles para la mejora del análisis y comprensión de cualquier programa que desarrollemos en ocasiones posteriores.

REFERENCIAS

- *MILES, Russ, HAMILTON, Kim*
  Learning UML 2.0
  Boston
  O Reilly Media, 2006

- *GOMAA, Hassan*
  Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures
  Washington
  Cambridge University Press, 2011

- *SCHMULLER, Joseph*
  Aprendiendo UML en 24 Horas
  México
  Prentice-Hall, 2000