Laboratorio Unidad 4

En la unidad 4 de nuestro curso hemos trabajado sobre el uso de la definición de contratos para la construcción de métodos y su invocación. Además, usamos las técnicas del experto y descomposición para realizar la asignación de responsabilidades de las clases. Dadas estas herramientas es posible escribir una clase completa del modelo de la solución siguiendo la especificación de un conjunto de contratos. Finalmente, es posible documentar dichos contratos utilizando la sintaxis definida por la herramienta Javadoc y la generación de un API a partir de Javadoc. El presente laboratorio les presenta una actividad en la cual se requiere aplicar todos los conocimientos adquiridos en esta unidad y verificar de esta manera el cumplimiento de los objetivos que han sido planteados para la unidad 4 descritos en el programa del curso. Para llevar a cabo este ejercicio es necesario realizar las actividades listadas a continuación:

Actividades

Lleve a cabo las siguientes actividades de cada una de las etapas de desarrollo de software:

1. Análisis del problema (Definición del problema, identificación de entidades, sus características y relaciones) y especificación de Requerimientos Funcionales
2. Diagrama de Clases Completo (incluye el Modelo y el Main en la interfaz). El modelo debe ser elaborado digitalmente, pero NO generado automáticamente (por ejemplo, no es válido entregar modelos generados por Object Aid o ninguna otra herramienta).
3. Diagrama de Objetos de la situación inicial de su software.
4. Trazabilidad del Análisis al Diseño. Una tabla a dos columnas en la que se relaciona cada requerimiento con el método o métodos que permiten satisfacer dicho requerimiento.
5. Implementación en Java. Incluya en la implementación, los contratos respectivos. Recuerde que todos los artefactos generados de fase de diseño e implementación deben ser en inglés.
6. API generado usando Javadoc (Incluyendo los contratos definidos para los métodos que cumplen con los requerimientos funcionales propuestos en el laboratorio 3 y el presente laboratorio 4).
7. Usar GitHub como repositorio de código fuente utilizando la estructura de carpetas aprendida en clase. Recuerde que se debe evidenciar su avance a través de los días en el laboratorio.

Recuerde que puede encontrar la Rúbrica laboratorio en el siguiente enlace.

Para llevar a cabo la actividad 1 recuerde que:

1. Los sustantivos describen posibles candidatos a entidades.
2. Los verbos asociados a posibles entidades sugiere los requerimientos funcionales.
3. Las relaciones usualmente están dadas cuando encuentre frases como: “se relaciona con” o “tiene”.

Usted debe subrayar con diferentes colores las entidades, sus características, sus relaciones y los requerimientos funcionales identificados.

Nota: Usted debe entregar un archivo en formato pdf con toda la documentación (análisis, diseño y tabla de trazabilidad) y la URL de su repositorio git Hub donde se deben encontrar los archivos de codificación en sus respectivos paquetes.

Tenga en cuenta que su repositorio gitHub debe presentar una estructura base como por ejemplo:

Veterinary/
src/ bin/ docs/

Dentro de los directorios src/ y bin/ estarán presentes estos directorios, representando cada uno de sus paquetes:
model/ ui/

El directorio src (source code) contiene sus clases .java dentro de cada uno de los directorios model y ui. Por otro lado el directorio bin (binary files) contiene los archivos .class en los directorios model y ui.

Su código debería compilar de acuerdo con lo explicado en la diapositiva 13 de esta presentación: http://tinyurl.com/y3bd9bg2

Recuerde el listado de etiquetas disponibles para documentar su código haciendo uso de Javadoc.

Etiquetas Javadoc

Tag	Descripción	Uso	Versión
@author	Nombre del desarrollador.	nombre_autor	1.0
@version	Versión del método o clase.	versión	1.0
@param	Definición de un parámetro de un método, es requerido para todos los parámetros del método.	nombre_parametro descripción	1.0
@return	Informa de lo que devuelve el método, no se puede usar en constructores o métodos "void".	descripción	1.0

@throws	Excepción lanzada por el método, posse un sinonimo de nombre @exception	nombre_clase descripción	1.2
@see	Asocia con otro método o clase.	referencia (#método(); clase#método(); paquete.clase; paquete.clase#método()).	1.0
@since	Especifica la versión del producto	indicativo numerico	1.2
@serial	Describe el significado del campo y sus valores aceptables. Otras formas validas son @serialField y @serialData	campo_descripcion	1.2
@deprecated	Indica que el método o clase es antigua y que no se recomienda su uso porque posiblemente desaparecerá en versiones posteriores.	descripción	1.0

Tomado de: https://es.wikipedia.org/wiki/Javadoc

Enunciado

La primera entrega de avances del desarrollo del software de la veterinaria dejó a las directivas de esta prestigiosa entidad muy impresionadas. Por lo que ahora quieren que se les entregue un programa funcional que resuelva las necesidades anteriormente planteadas (ver Primer Enunciado del Laboratorio de la Veterinaria).

Junto a esas funcionalidades la veterinaria quiere que los usuarios que van a operar el software también puedan realizar las acciones definidas en los siguientes contratos:

```
/**
*Description This method allows to calculate the body mass index for a pet.
*pre: The pet was created before and its attributes height and weight are not null neither height must be zero.
*post: The BMI is calculated.
*@return The pet body mass index. Returns -1 if the height is zero  due to the division on zero does not exist.
*/

/**
*Description This method allows to update the basic data of a veterinary client, these data include, address and phone number.
*pre: The client was created before.
*post: The address and /or phone number of the client is updated.
*@param The new address of the client. This param could be empty.
```

*@param The new phone number of the client. This param could be empty.
*/

/**
*Description This method allows to add new medicines that were prescription during the hospitalization at the patient stories.
*pre: The patient clinic story must be not null.
*post: New medicines were added to the patient clinic story.
*@param The medicine name. This param must be not null.
*@param The medicine dose, this param refers to the amount of medicine supplied to the pet each time according the frequence assigned.
*@param The medicine cost by each dose. This param could be empty.
*@param The frequency of medicine application. This param could be empty.
*@return A message that indiques if medicine was added to the patient clinic story
*/

/**
*Description This method allows to add new notes to the possible diagnostic during the hospitalization at the patient stories.
*pre: The patient clinic story must be not null.
*post: New notes were added to the possible diagnostic in the patient clinic story.
*@param The notes of possible diagnostic. This param must be not null.
*/

/**
*Description This method allows to add a new symptom presented during the hospitalization at the patient stories.
*pre: The patient clinic story must be not null.
*post: A new symptom were added to the patient clinic story.
*@param The new symptom presented. This param must be not null.
*/

Además, la veterinaria espera que se puedan manejar los otros servicios que ofrece. La veterinaria ofrece los servicios de baño de mascotas en la veterinaria, baño de mascotas a domicilio, corte de uñas, profilaxis dental y aplicación de vacunas. De cada servicio se desea conocer el tipo de servicio, el costo, la fecha de realización del servicio, el identificador de la mascota a la que se le realizó el servicio y el identificador del dueño de la mascota a la cual se le realizó el servicio.

Los costos de los servicios de la veterinaria se especifican a continuación.

Tipo de Servicio	Costo del servicio
Baño de mascotas en la veterinaria	$20.000
Baño de mascota a domicilio	$30.000
Corte de Uñas	$8.000

Profilaxis Dental	$12.000
Aplicación de Vacunas	$45.000

Dados los nuevos servicios que presta la veterinaria se requiere implementar las nuevas funcionalidades que se espera el software también permitirá ejecutar. Estas funcionalidades deben ser asignadas a la clase correspondiente, recuerde que puede hacer uso de las técnicas de descomposición y del experto para realizar la asignación. A continuación se listan las nuevas funcionalidades requeridas.

- Calcular los ingresos por servicios
- Calcular los ingresos totales de la veterinaria
- Agregar al sistema nuevos servicios prestados por la veterinaria, es decir los servicios vendidos no nuevos tipos de servicios.
- Promedio de ingresos por servicios.
- Promedio de ingresos de la veterinaria en una semana.
- Reporte de servicios prestados dada una fecha inicial y una fecha final.

Su programa debe cumplir con:

- La creación de algunos valores iniciales para que el programa funcione. Para esto se requiere que:
    - Existan al menos 2 clientes, cada uno con al menos 2 mascotas como valores iniciales existentes en el programa.
    - Exista al menos un animal hospitalizado actualmente y una historia clínica en el historial. Exista al menos 3 servicios prestados.
    - Se calculen de manera correcta todos los RF identificados en la entrega pasada y esta.
- Se despliega un menú que permita al usuario elegir la opción que desea utilizar del programa. Al usuario elegir la opción se realiza la operación solicitada por el usuario y se muestra de nuevo el menú.

Git hub:
https://github.com/CamiloEscobar33/VeterinaryCali

1. Análisis del problema: (Definición del problema, identificación de entidades, sus características y relaciones) y especificación de Requerimientos Funcionales

Requerimientos:

Nombre	**RF1:** Registrar dueños de mascotas.
Resumen	Permite registrar a los dueños de la mascota.
Entrada	Nombre cliente, id, dirección, teléfono de contacto.
Salida	Mostrar en pantalla el cliente fue registrado correctamente.

Nombre	**RF2:** Registrar una mascota a la veterinaria .
Resumen	Registrar una mascota en la veterinaria.
Entrada	Nombre de la mascota, tipo de animal, edad y peso.
Salida	Mostrar en pantalla que la mascota fue añadida correctamente.

Nombre	**RF3:** Crear historia clínica.
Resumen	Crear una historia clínica de una mascota colocando toda su respectiva información, en caso de que exista la historia clínica debe anexarse en la nueva historia clínica.
Entrada	Datos correspondientes a la mascota, su dueño, el estado, fecha de ingreso, los síntomas presentados, el posible diagnóstico y los medicamentos recetados.
Salida	Imprimir en pantalla "Se le ha creado las historia clínica a la mascota" o " Se ha anexado correctamente a la historia clínica a la mascota".

Nombre	**RF4:** Determina la disponibilidad de los mini Cuartos.
Resumen	Permite determinar si hay disponibilidad de un mini cuarto.
Entrada	---
Salida	Mostrar los mini Cuartos que están disponibles.

Nombre	**RF5:** Informe de historias clínicas de mascotas hospitalizadas hasta el momento.
Resumen	Permite mostrar un informe de las historias clínicas de los pacientes que han sido hospitalizados.
Entrada	
Salida	Mostrar el informe.

Nombre	R**F6:** Permite consultar los datos de contacto del dueño de un animalito hospitalizado.
Resumen	Consultar los datos del dueño al cual tiene una mascota hospitalizada.
Entrada	Id del dueño.
Salida	Datos de contacto del dueño de la mascota.

Nombre	**RF7:** Calcular el costo de una hospitalización.

Resumen	Teniendo en cuenta que cada día de hospitalización tiene un costo y con esto poder calcular el costo de la hospitalización.
Entrada	Id del dueño, nombre del animal hospitalizado, fecha de entrada y fecha salida.
Salida	Costo de hospitalización del animal.

Nombre	**RF8:** Calcular el costo de los medicamentos
Resumen	Permite calcular el costo de los medicamentos teniendo en cuenta la cantidad de dosis del medicamento que se le han aplicado al animalito.
Entrada	id del dueño, nombre del animal.
Salida	Mostrar en pantalla el costo del medicamento.

Nombre	**RF9:** Permitir dar alta a un mascota que está hospitalizado.
Resumen	Permite dar de alta a un animal que ha estado hospitalizado y mostrar un informe de datos del animal hospitalizado, y mostrar un mensaje que se le ha guardado su caso en un historial de historias clínicas.
Entrada	Id del dueño, nombre del animal.
Salida	Mostrar en pantalla "(nombre del animal) ha sido de alta exitosamente" y un informe de datos de la hospitalización del paciente.

Nombre	**RF10:** Determina cuánto han sido sus ingresos por concepto de hospitalizaciones
Resumen	Permitir saber cuál es la cantidad monetaria total de las hospitalizaciones.
Entrada	-.-
Salida	Ingreso totales de hospitalizaciones.

Nombre	**RF11:** Permitir saber el número del mini cuarto que ocupa una mascota basado en su nombre.
Resumen	Permite con su nombre determinar en qué Mini cuarto esta la mascota.
Entrada	Nombre animal.
Salida	Numero de mini cuartos que esta la mascota.

Nombre	**RF12:** Permite consultar en el historial de historias clínicas la cantidad de hospitalizaciones que ha tenido un animal.
Resumen	Determinar la cantidad de hospitalizaciones que ha tenido una mascota.
Entrada	Id del dueño y nombre de la mascota.
Salida	Numero de hospitalizaciones de la mascota.

Nombre	**RF13:** Calcular los ingresos por servicios.
Resumen	Permite calcular los ingresos por servicios por las mascotas.
Entrada	---
Salida	Ingresos por servicios.

Nombre	**RF14:** Calcular los ingresos totales de la veterinaria.
Resumen	Permite calcular los ingresos totales de la veterinaria.
Entrada	---
Salida	Ingresos totales.

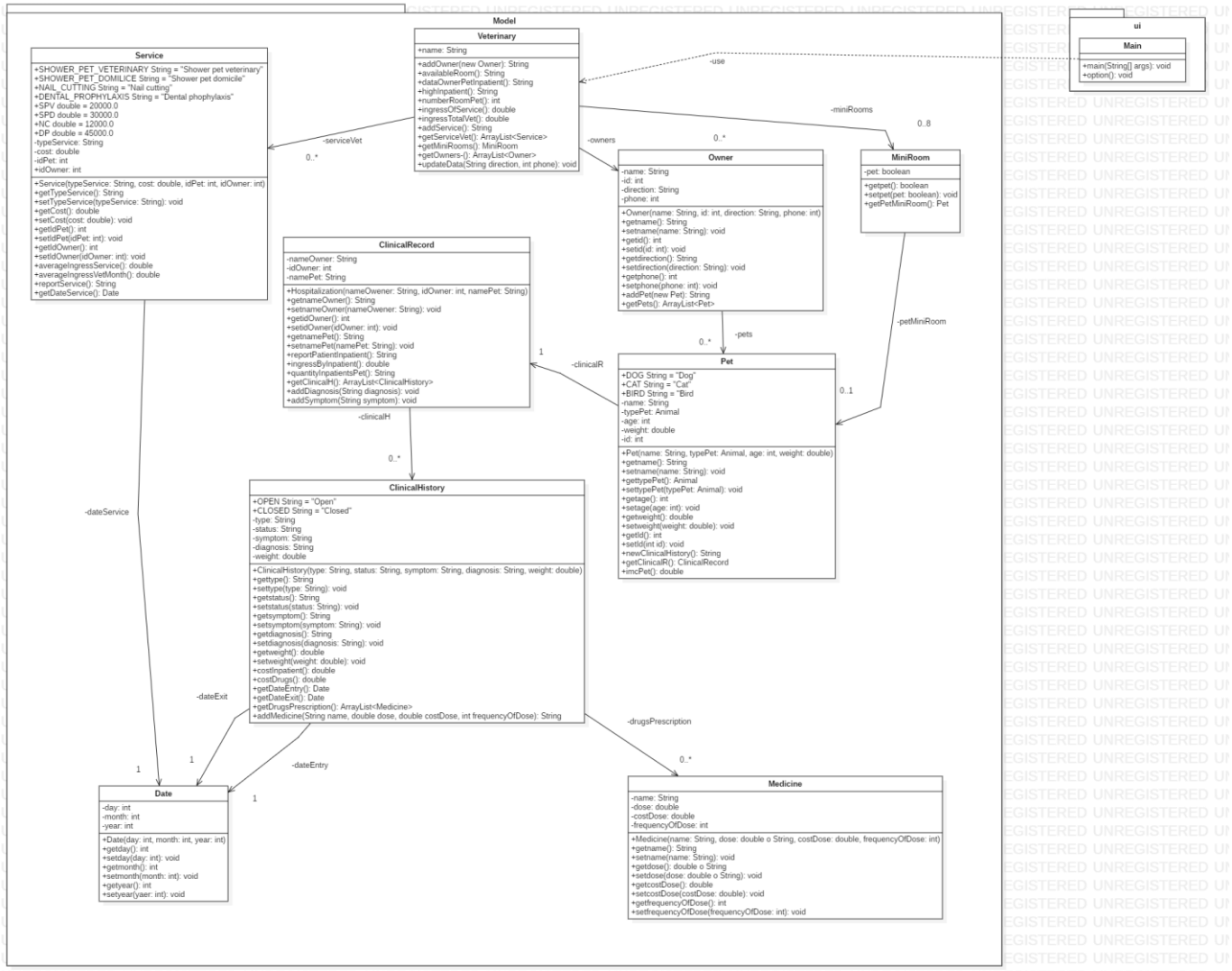
Nombre	**RF15:** Agregar servicio prestado
Resumen	Permite agregar un servicio prestado a la mascota.
Entrada	Id de la mascota, tipo de servicio prestado.
Salida	Mostrar en pantalla " el servicio fue agregado correctamente".

Nombre	**RF16:** Calcular el promedio de ingresos por servicios.
Resumen	Permite calcular el promedio de ingresos por servicios totales.
Entrada	----
Salida	EL promedio de ingresos por servicios.

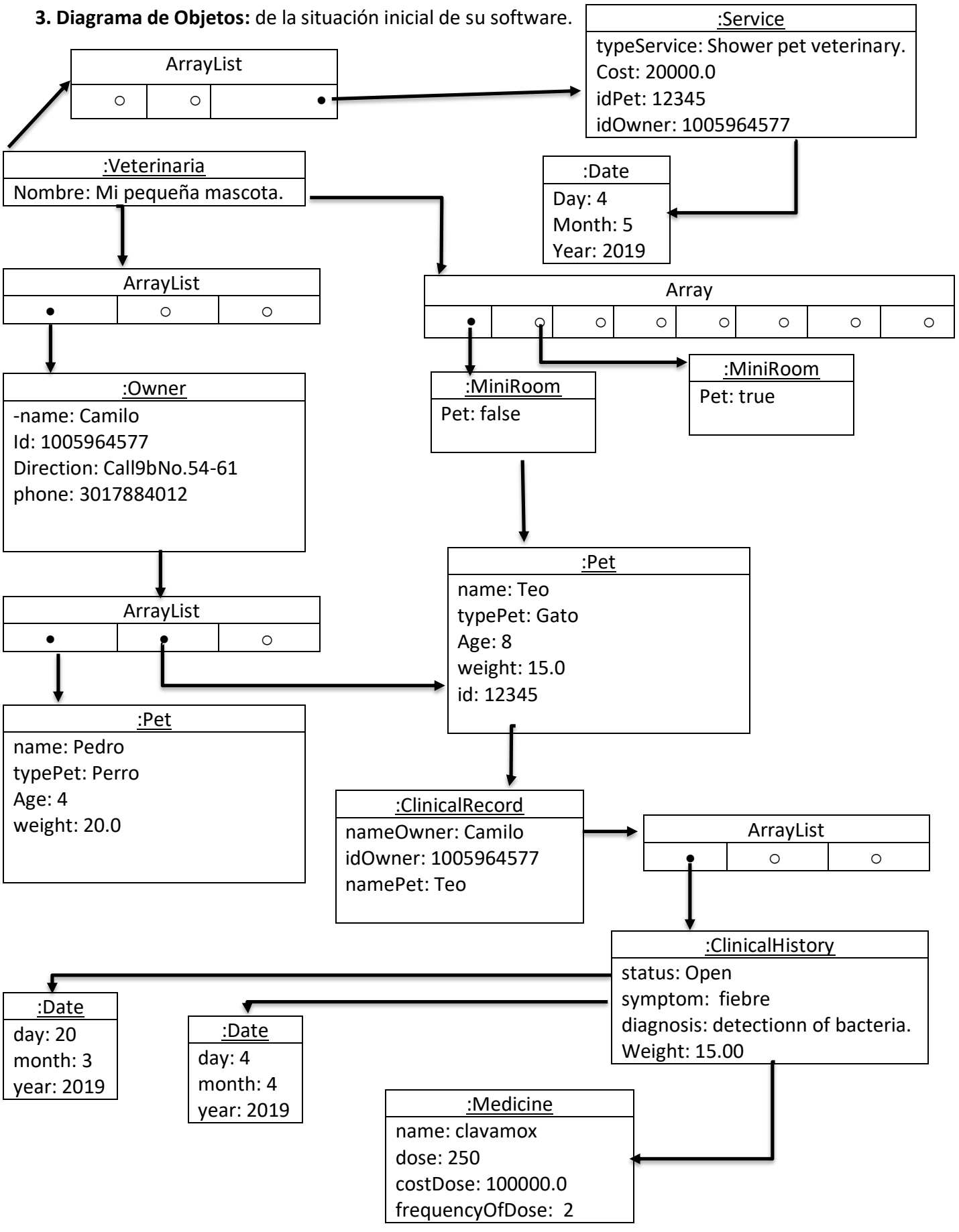
Nombre	**RF17:** Calcular promedio de ingresos de la veterinaria en una semana.
Resumen	El sistema permite calcular el promedio de ingresos de la veterinaria en una semana.
Entrada	-----
Salida	Promedio de ingresos de la veterinaria en una semana.

Nombre	**RF18:** Reporte de servicios prestados
Resumen	El sistema permite mostrar un reporte de servicios prestados en determinadas fechas.
Entrada	Fecha de entrada y fecha de salida.
Salida	Reporte de servicios prestados en ese rango de tiempo condicionado.

2. Diagrama de Clases:



3. Diagrama de Objetos: de la situación inicial de su software.



4. Trazabilidad:

Requerimiento	Metodos
RF1: Permite registrar dueños de mascotas.	Veterinary: addOwner();
RF2: Añadir una mascota a la veterinaria.	Owner: addPet();
RF3: Crear historia clínica de una mascota.	Pet: newClinicalHistory();
RF4: Determina la disponibilidad de los mini Cuartos.	Veterinary: availableRoom();
RF5: Informe de historias clínicas de pacientes hospitalizados.	ClinicalRecord: reportPatientInpatient();
R**F6:** Permite consultar los datos de contacto del dueño de un animalito hospitalizado.	Veterinary: dataOwnerPetInpatient();
RF7: Calcular el costo de una hospitalización.	ClinicalHistory: costInpatient();
RF8: Calcular el costo de los medicamentos.	ClincalHistory: costDrugs();
RF9: Permitir dar alta a un animalito que está hospitalizado.	Veterinary: highInpatient();
RF10: Determina cuánto han sido sus ingresos por concepto de hospitalizaciones.	ClinicalRecord: ingressByInpatient();
RF11: Permitir saber el número del mini cuarto que ocupa una mascota basado en su nombre.	Veterinary: numberRoomPet();
RF12: Permite consultar en el historial de historias clínicas la cantidad de hospitalizaciones que ha tenido un animal.	ClinicalRecord: quantityInpatientsPet();
RF13: Calcular los ingresos por servicios.	Veterinary: ingressOfService();
RF14: Calcular los ingresos totales de la veterinaria.	Veterinary: ingressTotalVet();
RF15: Agregar servicio prestado.	Veterinary: addService();
RF16: Promedio de ingresos por servicios.	Service: averageIngressService();
RF17: Promedio de ingresos de la veterinaria en una semana.	Service: averageIngressVetMonth();
RF18: Reporte de servicios prestados dada una fecha inicial y una fecha final.	Service: reportService();