REQUERIMIENTOS FUNCIONALES:

R1: Añadir un nuevo paciente. el paciente de debe tener un nombre, in ID único (un número de 7 dígitos), el cual no se puede repetir, debe tener raza, edad, la especie a la que pertenece, una descripción del paciente, su estado y el historial médico del paciente.
//entregarle un dueño en caso de que exista, si no lo hace se crea uno nuevo

R2:Crear un nuevo dueño, el dueño debe tener nombre, apellido, teléfono, dirección de residencia.
//Entregarle una mascota existente en el sistema o crear una nueva.

R3: Crear una ficha médica, cuando un paciente está hospitalizado o en tratamiento, se le asigna una ficha médica que contiene la información general y detallada del tratamiento.

R4: Cambiar el estado de la mascota, se le otorga al paciente un nuevo estado.
//Si el paciente cambia a un estado de tratamiento u hospitalización se le asigna una ficha médica en la cual se llevará el seguimiento del paciente debe ser modificado añadiendo información general del tratamiento y los detalles del tratamiento. Cuando vuelva a cambiar de estado el sistema automáticamente debe agregar la información general del tratamiento al historial médico.

R5: Actualizar ficha médica, el usuario puede modificar la ficha médica de un paciente.

R6: Añadir nuevo evento al historial de un paciente. se le agrega un nuevo evento al historial médico del paciente, este se agrega junto con la fecha de realización del procedimiento.

R7: buscar a un paciente mediante su id. se busca un paciente haciendo uso de su número de id, y se imprime su información en pantalla.

R8: Mostrar los pacientes registrados en el sistema. Se despliega en pantalla la información de cada paciente registrado, se pueden mostrar ordenados por:

a. Edad
b. ID
c. Especie

R9: Mostrar en pantalla todos los pacientes que estén hospitalizados. Muestra en pantalla a todos los pacientes que estén hospitalizados actualmente, junto con con la información general de la mascota (edad, nombre, id).

R10: Agregar un nuevo producto. se agrega un nuevo producto al stock, el producto puede ser de diferentes tipos (juguetes, comida y accesorios), cada producto tiene un nombre y un precio,un número de referencia, un contador de cuantos productos de este tipo se han vendido, cuántos quedan en stock, el costo al que lo consigue el vendedor y dependiendo del tipo que sea pasa a tener otra información. Los juguetes tienen un String con el color y

el tamaño que tiene. La comida tiene un String en el cual tiene el nombre de la especie de animal para la que está hecha (caninos, felinos, aves y roedores) y tiene su peso. Los accesorios tienen un color.

R11: Buscar producto por número de referencia, se busca un producto por su número de referencia y se imprime en pantalla su información dependiendo el tipo de producto.

R12: Mostrar productos en el sistema según su tipo, se imprime en pantalla los productos registrados de un tipo.

R13: Mostrar todos los productos en el sistema, se despliega en pantalla la información de todos los productos registrados en el sistema, se pueden ordenar por:

a. Precio (Más caro al más barato o viceversa)
b. Número de referencia(empieza con letra seguido de 3 números)
c. Tipo

R14: Mostrar unidades vendidas vendidos de cada producto, se despliega en pantalla el número de unidades vendidas de cada producto y los beneficios que ha generado (ordenados de mayor a menor).

R15: Eliminar un producto, se elimina un producto del sistema mediante su número de referencia.

R16: llevar la fecha actual del programa. lleva la fecha y hora actuales del programa y las muestra constantemente en pantalla.

R17: Generar un informe con el historial médico de un paciente, se imprime en un archivo el historial médico de un paciente buscándolo mediante su ID.

R18: vender producto. se busca un producto mediante su código y se vente en caso de que se encuentre en stock (el usuario escoge la cantidad de productos que va a vender de este tipo).

R19: aumentar el stock de un producto. Se busca un producto mediante su código y el usuario le puede agregar una cantidad o establecer una nueva cantidad.

R10: Eliminar un paciente, se elimina un paciente del sistema mediante su IID.

REQUERIMIENTOS NO FUNCIONALES:

R1: Persistencia mediante serialización, aquello que se serializará será el apartado de veterinaria del programa. Lo primero que hará el programa al iniciarse será cargar esta información, cada que se añada un paciente, dueño o que se cambie información del programa, este llevará a cabo el proceso de serialización.

R2: Persistencia mediante archivos de texto plano, aquello que se guardará de esta manera será el apartado de productos del programa. Lo primero que hará el programa al iniciarse

será cargar esta información, cada que se añada un producto o que se cambie información del programa, este llevará a cabo el proceso de persistencia.
R3: El programa manejara excepciones del api de java en los siguientes aspectos:

1. Manejo de archivos(persistencia), los métodos en los cuales se realiza escritura o lectura de archivos persistentes, se lanzarán las siguientes excepciones IOException, ClassNotFoundException.
2. Interacción con el usuario, cuando el programa requiere de datos por el usuario, el programa lanzará las siguientes excepciones en caso de alguna entrada que perjudique el funcionamiento del programa, IllegalFormatException, InputMismatchException.

R4: El programa tendrá las siguientes excepciones personalizadas:

1. NoItemsInStockException, esta excepción se lanza cuando el usuario intenta vender un producto que se ha terminado o intenta vender más de las unidades existentes.
2. MissingFieldException, esta excepción se lanza cuando el usuario no llena todos los campos de un formulario de registro.
3. PacientNotFoundException, esta excepción se lanza cuando el usuario busca un paciente que no está registrado en el sistema.
4. ProductNotFoundException, esta excepción se lanza cuando el usuario busca un producto que no está registrado en el sistema.

R5:

- **Bubble sort:** El ordenamiento de burbuja estará presente en el método que muestra todos los productos del sistema. Este ordenamiento se usará cuando los productos se muestran ordenados por precio o por número de referencia.
- **Selection sort:** Este ordenamiento estará presente en dos métodos, el primero es el método que muestra todos los productos del sistema y el segundo es el método que muestra todos los pacientes registrados en el sistema. en el primer método se utilizará cuando los productos se vayan a mostrar ordenados por su tipo y en el segundo método se usará cuando los pacientes se muestran ordenados por su edad.
- **Insertion sort:** Este ordenamiento estará presente en el método que muestra todos los pacientes registrados en el sistema. Este ordenamiento se usará cuando los pacientes se muestran ordenados por su ID o por su especie.

R6: Búsqueda binaria: se hará uso de la búsqueda binaria en los métodos de buscar un producto y buscar un paciente, el criterio con el que se hará se encuentra escrito en los respectivos requerimientos funcionales.
R7: Listas enlazadas: los pacientes y productos serán guardados en estructuras de listas enlazadas.

R8: Herencia: Para clasificar los animales de la clínica veterinaria tendremos una clase padre llamada Animal y esta tendrá 4 hijos denominados como Canine, Feline, Rodent y Bird. Por otro lado, para clasificar nuestros productos tendremos una clase padre llamada Product y unas clases hijas llamadas Toy, Food, Accessory.
Interfaces: El programa tendrá una interfaz llamada CalculateProfit implementada por los productos para calcular las utilidades del producto. Por otro lado, tendrá otra interfaz para

calcular el dinero obtenido por la venta de un producto de un producto está se llamará CalculateEarnings. Por último, se implementará una interfaz llamada SellProduct para vender los productos de la tienda.

R9: Árbol binario: Los dueños de los pacientes serán almacenados en una estructura de árbol binario.

R10: Los métodos para buscar y agregar pacientes serán implementado de manera recursiva en una lista enlazada. Por otro lado, los métodos de buscar y agregar dueño también serán implementados de manera recursiva en una estructura de árbol binario.

R11: El programa tendrá un hilo para llevar la hora del día, un hilo para mantener actualizada la tabla de pacientes y ordenarla según el valor que se le de a la comboBox y otro hilo para mantener actualizada la tabla de productos de la misma manera que la anterior.

Diseño de pruebas unitarias

Escenarios

Nombre	Clase	Escenario
setup1	**Veterinary**	**Se agregan 2 objetos de la clase Accessory a la lista enlazada de productos. Se agregan 2 objetos de la clase Toy a la lista enlazada de productos. Se agregan 2 objetos de la clase Food a la lista enlazada de productos.**
setup2	**Veterinary**	**Se agregan 2 objetos de la clase Cat a la lista enlazada de animales. Se agregan 2 objetos de la clase Dog a la lista enlazada de animales. Se agregan 2 objetos de la clase Bird a la lista enlazada de animales. Se agregan 5 objetos de la clase Rodent a la lista enlazada de animales.**
setup3	**Veterinary**	**Se agregan 7 objetos de la clase Owner al árbol binario de dueños.**
setup4	**Veterinary**	**Se le agrega a cada paciente del setup2 una ficha médica con palabras claves**

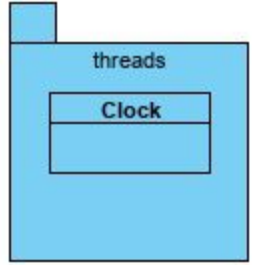
Diseño de Casos de Prueba

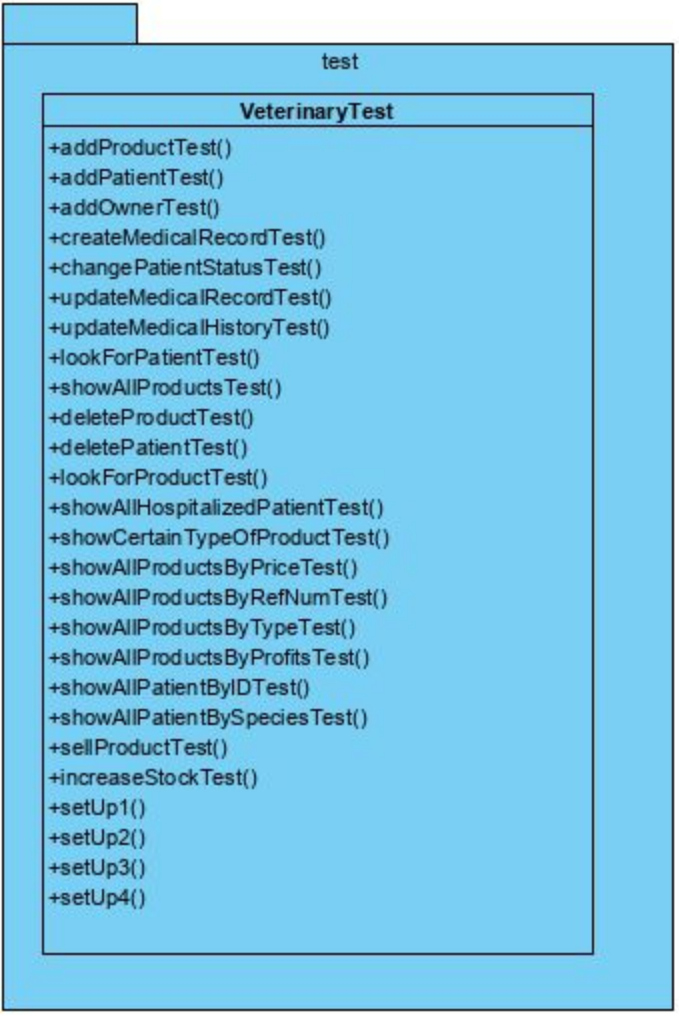
Objetivo de la Prueba:				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Veterinary	**addProduct**	**setup1**	**ninguno**	**Al recorrer la lista enlazada los nombres de los productos deben coincidir con el de los productos que fueron agregados, si no es por que el método de agregar producto no está funcionando correctamente.**
Veterinary	**addPatient**	**setup2**	**ninguno**	**Los nombres de los objetos en la lista enlazada deben coincidir con los de los productos que**

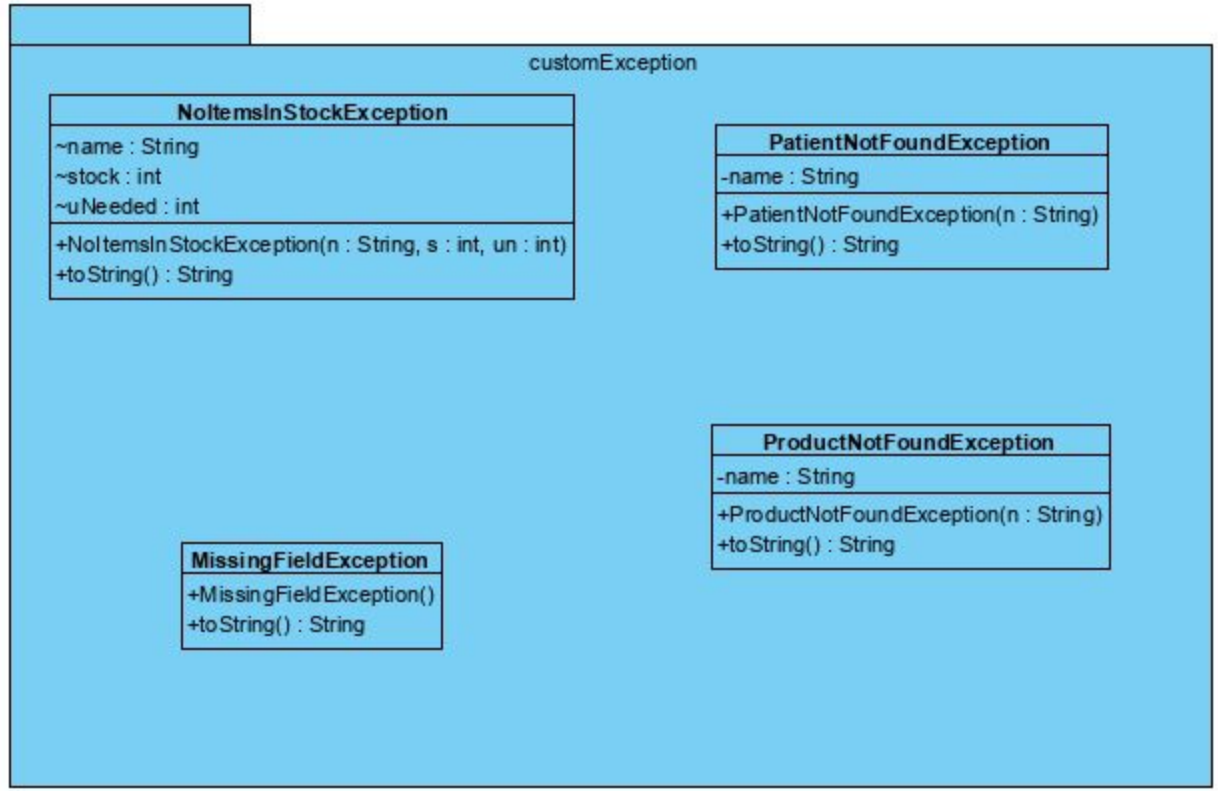
				fueron registrados.
Veterinary	**addOwner**	**setup3**	**ninguno**	**Al imprimir los nombre de los objetos en el arbol binario estos deben coincidir con los nombres de los objetos agregados.**
Veterinary	**createMedicalRecord**	**setup4**	**ninguno**	**Al obtener la información de las fichas médicas de los pacientes estas coinciden con las esperadas.**
Veterinary	**changePatientStatus**	**setup2**	**Constante de la clase Animal con el estado a cambiar**	**Al cambiar el estado de uno de los pacientes en la lista enlazada este coincide con el valor esperado.**
Veterinary	**updateMedicalRecord**	**setup4**	**Palabras claves para identificar si el cambio se lleva a cabo bien**	**Al cambiar la información de una de las fichas médicas de la lista enlazada esta coincide con la esperada.**
Veterinary	**updateMedicalHistory**	**setup2**	**Palabras claves para identificar si el cambio se lleva a cabo bien**	**Al actualizar el historial médico de uno de los pacientes este coincide con la información deseada.**
Veterinary	**lookForPatient**	**setup2**	**Id de uno de los pacientes** **Id que no pertenezca a ninguno de los pacientes**	**Se busca un paciente con su respectivo id y se verifica que coincida. Se busca un id inexistente para comprobar que no devuelve ningún paciente.**
Veterinary	**showAllProducts**	**setup1**	**ninguno**	**Se verifica que el método esté retornando correctamente todos los productos registrados hasta el momento.**
Veterinary	**deleteProduct**	**setup1**	**Numero de referencia de uno de los productos**	**Se elimina el producto y se verifica que ya no existe en el**

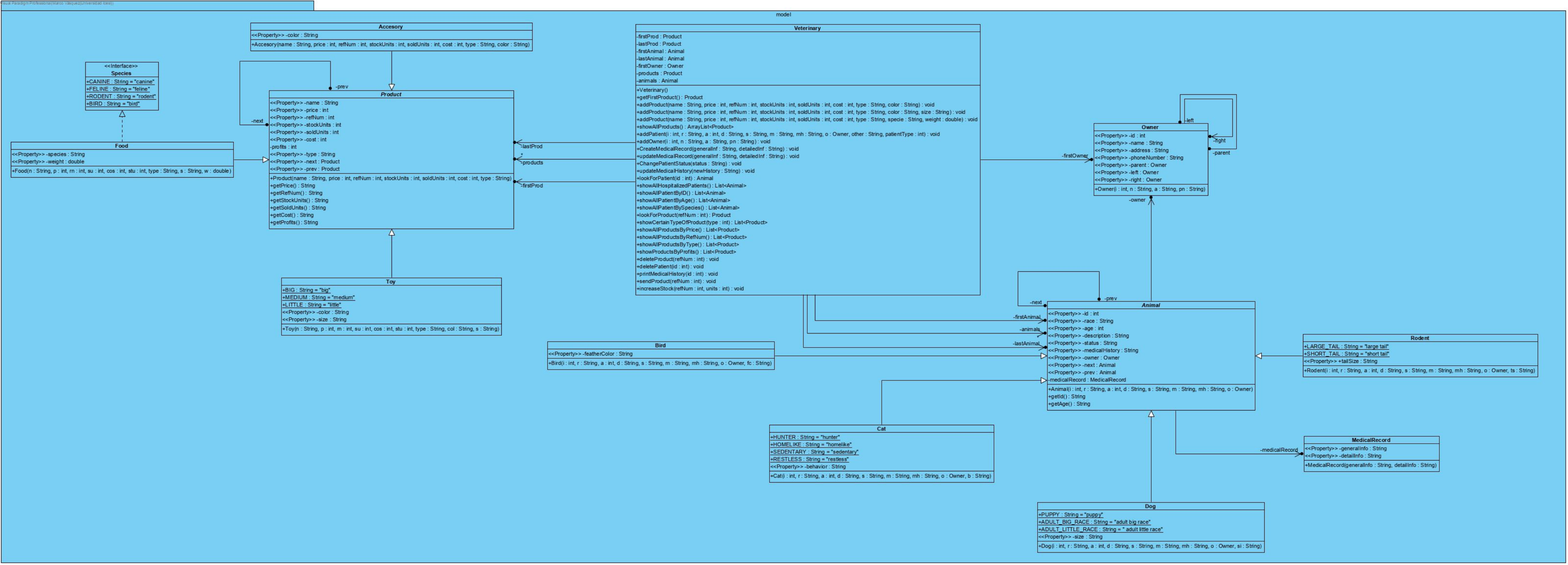
				programa.
Veterinary	**deletePatient**	**setup2**	**ID de un paciente registrado**	**Se elimina el paciente deseado y se verifica que ya no exista en el programa**
Veterinary	**lookForProduct**	**setup1**	**Número de Referencia de uno de los productos Número de Referencia que no pertenezca a ninguno de los productos.**	**Se busca el paciente deseado y se verifica que coincida con el retornado. Se busca un paciente inexistente y se verifica que no devuelve ningún paciente.**
Veterinary	**showAllHospitalized Patients**	**setup2**	**ninguno**	**Se cambia el estado de algunos pacientes a hospitalizado y se verifica que devuelva los deseados.**
Veterinary	**showCertainTypeOf Product**	**setup1**	**String indicado el tipo juguete**	**Se verifica que el método retorne los productos tipo juguete**
Veterinary	**showAllProductsbyP rice**	**setup1**	**ninguno**	**Se ordena en una lista aparte los productos por precio y se compara con la lista retornada por el método**
Veterinary	**showAllProductsby RefNum**	**setup1**	**ninguno**	**Se ordena en una lista aparte los productos por número de referencia y se compara con la lista retornada por el método**
Veterinary	**showAllProductsbyT ype**	**setup1**	**ninguno**	**Se ordena en una lista aparte los productos por su tipo y se compara con la lista retornada por el método**

Veterinary	**showAllProductsbyProficts**	**setup1**	**ninguno**	**Se ordena en una lista aparte los productos por sus ganancias y se compara con la lista retornada por el método**
Veterinary	**showAllPatientbyID**	**setup2**	**ninguno**	**Se ordena en una lista aparte los pacientes por su ID y se compara con la lista retornada por el método.**
Veterinary	**showAllPatientbyAge**	**setup2**	**ninguno**	**Se ordena en una lista aparte los pacientes por su edad y se compara con la lista retornada por el método.**
Veterinary	**showAllPatientbySpecie**	**setup2**	**ninguno**	**Se ordena en una lista aparte los pacientes por su especie y se compara con la lista retornada por el método.**
Veterinary	**sellProduct**	**setup1**	**cantidad a vender Número de referencia del producto**	**Se verifica que el número de unidades vendidas del producto haya aumentado correctamente**
Veterinary	**increaseStock**	**setup1**	**cantidad de unidades Número de referencia del producto**	**Se comprueba que el número de unidades en stock del producto aumentaron correctamente**








Visual Paradigm Professional(Marco Vasquez(Universidad Icesi))
model
Accesory
<<Property>> -color : String
+Accesory(name : String, price : int, refNum : int, stockUnits : int, soldUnits : int, cost : int, type : String, color : String)
<<Interface>>
Species
+CANINE : String = "canine"
+FELINE : String = "feline"
+RODENT : String = "rodent"
+BIRD : String = "bird"
-prev
-next
Product
<<Property>> -name : String
<<Property>> -price : int
<<Property>> -refNum : int
<<Property>> -stockUnits : int
<<Property>> -soldUnits : int
<<Property>> -cost : int
-profits : int
<<Property>> -type : String
<<Property>> -next : Product
<<Property>> -prev : Product
+Product(name : String, price : int, refNum : int, stockUnits : int, soldUnits : int, cost : int, type : String)
+getPrice() : String
+getRefNum() : String
+getStockUnits() : String
+getSoldUnits() : String
+getCost() : String
+getProfits() : String
Food
<<Property>> -species : String
<<Property>> -weight : double
+Food(n : String, p : int, rn : int, su : int, cos : int, stu : int, type : String, s : String, w : double)
-lastProd
*
-products
-firstProd
Toy
+BIG : String = "big"
+MEDIUM : String = "medium"
+LITTLE : String = "little"
<<Property>> -color : String
<<Property>> -size : String
+Toy(n : String, p : int, rn : int, su : int, cos : int, stu : int, type : String, col : String, s : String)
Veterinary
-firstProd : Product
-lastProd : Product
-firstAnimal : Animal
-lastAnimal : Animal
-firstOwner : Owner
-products : Product
-animals : Animal
+Veterinary()
+getFirstProduct() : Product
+addProduct(name : String, price : int, refNum : int, stockUnits : int, soldUnits : int, cost : int, type : String, color : String) : void
+addProduct(name : String, price : int, refNum : int, stockUnits : int, soldUnits : int, cost : int, type : String, color : String, size : String) : void
+addProduct(name : String, price : int, refNum : int, stockUnits : int, soldUnits : int, cost : int, type : String, specie : String, weight : double) : void
+showAllProducts() : ArrayList<Product>
+addPatient(i : int, r : String, a : int, d : String, s : String, m : String, mh : String, o : Owner, other : String, patientType : int) : void
+addOwner(i : int, n : String, a : String, pn : String) : void
+CreateMedicalRecord(generalInf : String, detailedInf : String) : void
+updateMedicalRecord(generalInf : String, detailedInf : String) : void
+ChangePatientStatus(status : String) : void
+updateMedicalHistory(newHistory : String) : void
+lookForPatient(id : int) : Animal
+showAllHospitalizedPatients() : List<Animal>
+showAllPatientByID() : List<Animal>
+showAllPatientByAge() : List<Animal>
+showAllPatientBySpecies() : List<Animal>
+lookForProduct(refNum : int) : Product
+showCertainTypeOfProduct(type : int) : List<Product>
+showAllProductsByPrice() : List<Product>
+showAllProductsByRefNum() : List<Product>
+showAllProductsByType() : List<Product>
+showProductsByProfits() : List<Product>
+deleteProduct(refNum : int) : void
+deletePatient(id : int) : void
+printMedicalHistory(id : int) : void
+sendProduct(refNum : int) : void
+increaseStock(refNum : int, units : int) : void
-firstOwner
-left
-right
-parent
Owner
<<Property>> -id : int
<<Property>> -name : String
<<Property>> -address : String
<<Property>> -phoneNumber : String
<<Property>> -parent : Owner
<<Property>> -left : Owner
<<Property>> -right : Owner
+Owner(i : int, n : String, a : String, pn : String)
-owner
-next
-prev
-firstAnimal
-animals
-lastAnimal
Animal
<<Property>> -id : int
<<Property>> -race : String
<<Property>> -age : int
<<Property>> -description : String
<<Property>> -status : String
<<Property>> -medicalHistory : String
<<Property>> -owner : Owner
<<Property>> -next : Animal
<<Property>> -prev : Animal
-medicalRecord : MedicalRecord
+Animal(i : int, r : String, a : int, d : String, s : String, m : String, mh : String, o : Owner)
+getId() : String
+getAge() : String
Bird
<<Property>> -featherColor : String
+Bird(i : int, r : String, a : int, d : String, s : String, m : String, mh : String, o : Owner, fc : String)
Rodent
+LARGE_TAIL : String = "large tail"
+SHORT_TAIL : String = "short tail"
<<Property>> +tailSize : String
+Rodent(i : int, r : String, a : int, d : String, s : String, m : String, mh : String, o : Owner, ts : String)
Cat
+HUNTER : String = "hunter"
+HOMELIKE : String = "homelike"
+SEDENTARY : String = "sedentary"
+RESTLESS : String = "restless"
<<Property>> -behavior : String
+Cat(i : int, r : String, a : int, d : String, s : String, m : String, mh : String, o : Owner, b : String)
-medicalRecord
MedicalRecord
<<Property>> -generalInfo : String
<<Property>> -detailInfo : String
+MedicalRecord(generalInfo : String, detailInfo : String)
Dog
+PUPPY : String = "puppy"
+ADULT_BIG_RACE : String = "adult big race"
+ADULT_LITTLE_RACE : String = " adult little race"
<<Property>> -size : String
+Dog(i : int, r : String, a : int, d : String, s : String, m : String, mh : String, o : Owner, si : String)