Análisis de problema y propuesta de solución

Para esta ocasión para cada requerimiento funcional se plantea una solución verbal.

Requerimiento 1: Lo que hacemos es pedir los datos del usuario del veterinario en el Main, pasarlos por parámetros a la clase “PetCare”, en donde usamos un método para validar si ya hay un veterinario con esos datos, o si hay espacios disponibles, luego en otro método se usan los parámetros para crear un objeto del tipo “Vet”, y esos datos se pasarían por parámetro al método constructor de la clase “Vet”.

Requerimiento 2: Se le pediría al usuario que ingrese la ID del veterinario que desea eliminar, se pasa por parámetro a la clase “PetCare”, en donde se verificaría que exista el veterinario, luego buscarlo en el arreglo a través de su ID e igualarlo a “null”, se le notificaría que se ha eliminado.

Requerimiento 3: Se le pide al usuario que registre los datos de la mascota y de su dueño, estos datos pasarían por parámetro para que sean verificados de que ya no estén registrados, también de que no sobre pasen el límite de mascotas por día ya sus datos pasarían por parámetro a la clase “Pet” y “Owner” donde se crearía nuevos objetos de tipo “Pet” y “Owner” respectivamente.

Requerimiento 4: Se le pide al usuario que ingrese el nombre de la mascota y el ID de su dueño, donde se verifican que estén registrados en el centro veterinario, una vez verificado no se elimina a la mascota sino que se le actualiza el estado a través de la variable de tipo “Pet” donde se le pasaría por parámetro que la mascota ha salido sin ser atendida.

Requerimiento 5: Para iniciar una consulta lo primero es que se pediría el ID al usuario del veterinario que entraría en consulta, luego se verifica que este veterinario si esté registrado, lo siguiente es que hayan mascotas registradas, para poder saber a qué mascota atender primero se crearía un método que devuelva a la primera mascota que encuentre con la prioridad más alta. se actualizarían los estados del veterinario y de la mascota a consulta se le asignaría el veterinario a la mascota y viceversa . Finalmente saldría un mensaje informando que el veterinario va a atender a la mascota.

Requerimiento 6: Igualmente para finalizar consulta se le pide el ID del veterinario que va a terminar la consulta y el estado clínico para la mascota, se verificaría que el veterinario este registrado y esté en consulta, para este caso como el veterinario tiene una mascota asociada entonces no es necesario verificar que esta exista o esté en consulta, para ambos se les actualizaría el estado, al veterinario estaría nuevamente disponible, y el estado para la mascota dependería del que se le haya ingresado, también pondría “null” la relación que tiene el veterinario con la mascota, pero dentro del requerimiento nos indica que la mascota conserva a quien la atendió, por esa razón las mascota conservaría su relación. Se mostraría el estado de la mascota y también que el veterinario está disponible nuevamente.

Requerimiento 7: Para contar cuantas mascotas faltan por atender, se haría un método en el que por medio de un ciclo y dentro un condicional verifique que si la mascota está registrada y en su estado sea en espera de ser atendido, de ser así se le sumara a una variable hasta salir del ciclo, se le mostraría al usuario el valor de la variable indicando que es la cantidad de mascotas que faltan por atender.

Requerimiento 8: Para generar el cierre diario primero se verificaría que no hayan mascotas por atender, para eso utilizaríamos los métodos implementados en el requerimiento 7, una vez no hayan pacientes por atender se generaría un informe estadístico donde para saber cuál fue el veterinario con mayor número de consultas, se crea un atributo en la clase “Vet” que se relacione con el número de consultas, donde cada vez que el veterinario implemente el requerimiento 5 se le aumente a este atributo, luego en un método en la clase “PetCare” se crearía un método que cree una variable temporal o burbuja, donde primero se implementa un ciclo repetitivo y dentro de este se crea un condicional donde se compara cual veterinario posee el más número de consultas, cuando encuentre a uno con un valor mayor se guarda en la variable temporal hasta hallar un valor mayor o de lo contrario notificar que ese es el veterinario con mayor cantidad de consultas. Por un lado, para mostrar el número de mascotas atendidas por prioridad, se crearía un método en la clase “PetCare”, se tendrían dos ciclos repetitivos el primero para acumular los mensajes de la cantidad de mascotas atendidas y el segundo que estaría dentro del primer ciclo seria para contar la cantidad de mascotas atendidas por prioridad. Por otro lado, Se usaría un ciclo repetitivo para contar el número que están registradas y de esas cuales salieron sin ser atendidas, una vez hecho, se verifica que el número de mascotas ingresadas sea mayor a cero, para que no realizar una división entre cero, dentro del condicional se divide el número de mascotas que salieron entre las que están registradas y se multiplica por cien y se mostraría en pantalla el porcentaje obtenido. Por último, para eliminar todas las mascotas, se usaría un método con un ciclo repetitivo en el cual se verifica que las mascotas si estén ingresadas y se les igualaría a “null” para eliminarlos y salir del programa.

Análisis del problema para TI3

## Requerimientos no funcionales

* Lo primero sería imprimir un mensaje en pantalla que le pregunte al usuario a que servicio desea ingresar, donde el programa debe de leer la opción insertada, en base a eso mostrar el respectivo menú, ya sea para emergencia veterinaria o para guardería, una vez en el menú y ejecuta la opción deseada, se le pregunta al usuario si desea cambiar de servicio, de manera que tenga la posibilidad de pasar de emergencia a guardería y viceversa.

## Requerimientos funcionales

* Para la hospitalización, primero abriría un espacio en el menú de emergencias, luego se preguntaría el nombre de la mascota junto con la ID de su dueño, se pasaría a la clase de “PetCare”, donde en un nuevo meto se comprobaría que dicha mascota esté registrada, que haya sido atendida y muy importante que el veterinario le haya impuesto hospitalización, después se busca en la matriz de hábitat, una que corresponda con la especie de la mascota, donde lo que seguiría es asignarle a la mascota el hábitat en la que va estar y al hábitat asignarle el huésped, también cambiarle el estado a “ocupado-sick”. Finalmente aparece un mensaje que le diga en que hábitat está la mascota.
* Para añadir una mascota en el nuevo menú, lo primero es preguntarles los datos al usuario, que al ser distintos a los que se piden en emergencia, se crea un método añadir mascota que sería sobrecargar el método, sin embargo los datos de la mascota se siguen guardando dentro del arreglo de mascota, por eso lo primero es verificar que la mascota no exista dentro del arreglo y que los hábitats para la respectiva especie se encuentren disponibles, luego se pasa por parámetros para asignar la mascota al hábitat y viceversa, una vez cumplido, mostrar un mensaje en el que notifique al usuario que la mascota está en un hábitat con su respectivo ID.
* Para buscar la mascota dentro de la guardería se le pide al usuario que ingrese el nombre de la mascota y el ID del dueño, esto porque pueden haber nombres de mascotas repetidas pero no con el mismo dueño, después de estas verificaciones se accede a la clase mascota y por medio de la relación que tiene con hábitat que es “residency”, emplear un método en el que se muestren las características del hábitat o residencia donde está la mascota, esto aplica tanto para una mascota sana como enferma.
* Para este no se necesita ninguna verificación, dentro de la clase hábitat se crea un método que permita devolver en letras el estado que se encuentra dicho hábitat, una vez realizado, en petcare como clase controladora se realiza un método en el que se usan 2 ciclos repetitivos y con condicionales para los índices, para recorrer la matriz de tal manera que la posición de cada jaula concuerde con la que piden en el enunciado, finalmente se retorna a clase main donde se imprime y muestra al usuario.
* Primero se le pregunta al usuario si desea ver los ID(s) de los hábitats, debido a que son muchos y así el usuario tiene la opción de elegir, luego se le pide que ingrese el id del hábitat al que dese conocer sus atributos, una vez ingresado se hace 2 ciclos repetitivos en los que se recorre la matriz hasta hallar el hábitat con el que se corresponda el ID ingresado, en este caso se llama un método con polimorfismo que sería “toString” es un método que poseen todas las clases hijas de “Habitat” y cuando se invoca muestra todas sus propiedades. En caso de que no coincida se le informa al usuario que el ID ingresado no corresponde a algún hábitat.
* Para el ultimo método se realiza un método que permita devolver un mensaje, y dentro del método llamar métodos para adquirir la información estadística, es decir, que recolecta la información y la transmite en un mensaje para imprimirla y mostrársela al usuario.
  + El primer método al que se llama se le pasaría por parámetro el tipo de hábitat, debido a que se saca el porcentaje de ocupación por tipo de hábitat, entonces se cuentan para el tipo de hábitat lo que estén ocupados y se dividen entre la cantidad de hábitats que hay de ese mismo tipo de hábitat.
  + Para este es la ocupación general, por lo tanto este método no tiene parámetros, se recorre la matriz con los 2 ciclos repetitivos, contando en un variable si están ocupados, luego el número de hábitats ocupados se divide entre el número de hábitats que hay (30), se multiplica por 100 y se retorna dicho valor
  + Para el ultimo método que es de calcular el porcentaje de ocupados enfermos/sanos, el método tiene un parámetro y es el para ambos tipos “SICK” o “HEALTHY” se recorre la matriz de forma completa contando los hábitats que estén ocupadas con una mascota enferma/sana dependiendo del caso y se divide entre el número total de hábitats, se multiplica por cien y se retorna el valor.
  + Se importa “java.text.DecimalFormat”, para limitar el número de decimales en el resultado.