Sesión 1

Modelado UML de clases

El administrador de un zoológico necesita una pequeña aplicación para administrar su parque:

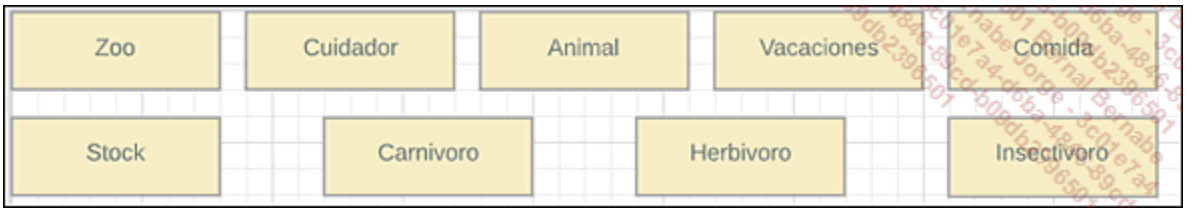
"El problema es que tengo dificultades para seguir la asignación de los **cuidadores** a los **animales**. Especialmente cuando alguien se va de **vacaciones**, tengo que prever su sustitución.

Y luego también está la **comida**. Manejo de **stocks** de alimentos y distribución a los **carnívoros**, **herbívoros** e **insectívoros**. Hoy, está completamente desordenado, de repente pierdo mucho tiempo alimentando a los animales. ¡Realmente necesito un sistema que me haga la vida más fácil!"

Esta es una descripción puramente de negocio (y ciertamente lejos de la realidad, pero esto es solo un ejemplo). Lo primero que hay que hacer es extraer los conceptos de negocio para poder manejarlos en la aplicación.

Estos conceptos se han resaltado en **negrita** y serán las clases principales del software.

Identificación de Clases:



1. Contenidos y continentes

Antes de completar estas clases con sus atributos y métodos, nos vamos a organizar un poco. Instintivamente, aparecen categorías: humanos, animales y logística.

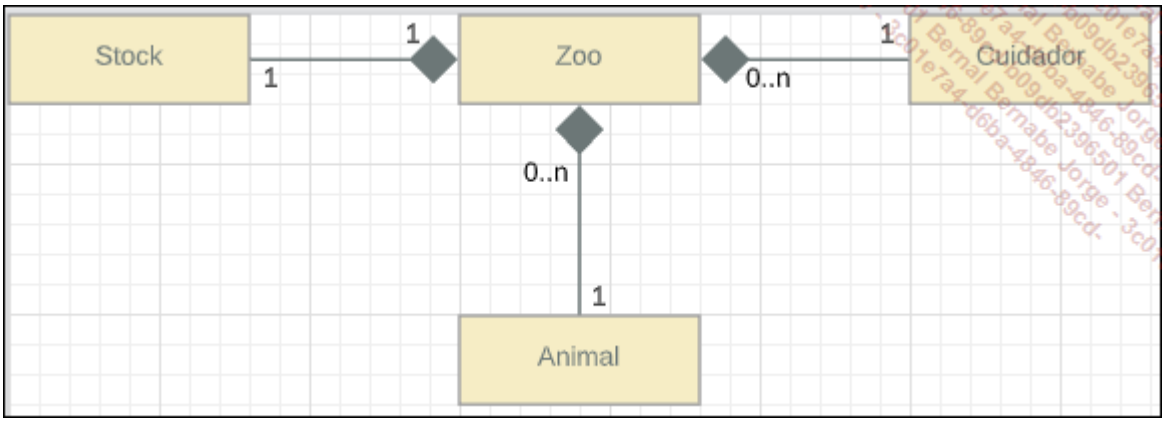
- conceptos "humanos": **Zoológico**, **Vacaciones** y **Cuidador**;
- conceptos "animales": **Animal**, **Carnívoro**, **Herbívoro** e **Insectívoro**;
- los conceptos de "logística": **Stock** y **Comida**.
- ...

Incluso sin conocer perfectamente el negocio, parece que tiene sentido considerar que las vacaciones están asociadas con un cuidador, que un cuidador está asociado con el zoológico, así como con los animales. El stock del zoológico tiene comida, que a su vez está relacionada con la dieta de los animales. Vayamos paso a paso:

Un zoológico acoge a los animales, tiene existencias de comida y emplea cuidadores. ¿Qué tipo de enlace usar aquí: una composición o una agregación? Para orientar esta elección, la técnica siempre es la misma: basta preguntarse si el contenido tiene una razón de ser o existir, si se destruye el contenedor. En este caso, si no hay zoológicos, no hay animales que manejar, ni cuidadores ni stock. Por tanto, es un enlace fuerte, es decir, una composición.

En el caso de los seres vivos, la cardinalidad es la misma: 0..n del zoológico hacia los cuidadores y los animales (un zoológico puede no tener empleados y también puede tener varios; lo mismo para los animales), y 1 en el otro sentido (un cuidador y un animal solo pueden pertenecer a un zoológico, ya que la aplicación solo administra uno). En cuanto al stock, el zoológico solo tiene uno, y el stock solo pertenece a un zoológico (uno podría imaginar un sistema de mutualización de comida si la aplicación administrara varios zoológicos, pero no es el caso aquí).

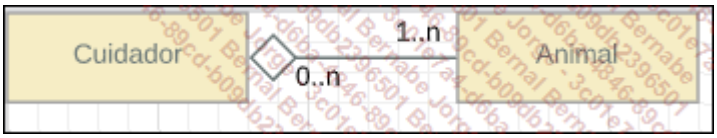
En realidad, los animales y los cuidadores ciertamente seguirán existiendo, pero esta no es una razón válida para elegir una agregación. Así como una habitación no tiene sentido sin una casa, un cuidador no tiene sentido si el zoológico no existe. Ningún zoológico vive sin cuidadores ni animales. Posiblemente podríamos considerar una agregación en el caso de que la aplicación que gestione varios zoológicos, donde un cuidador podría ser trasladado a otro parque y donde un animal sería trasladado unos kilómetros más lejos.



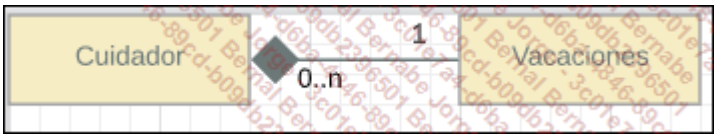
Se asigna un cuidador a uno o más animales. De hecho, a la pregunta "¿el animal desaparece con el cuidador?", la respuesta aquí es no: si el cuidador deja su trabajo, el animal se queda en el zoológico. Por tanto, es una agregación, una relación débil.

Además, la descripción realizada por el cliente sobre la cardinalidad de esta relación, no es completa. ¿Se asigna un cuidador a un solo animal o a varios? Por tanto, es bueno pedirle una aclaración. Dependiendo de la respuesta, la cardinalidad de esta relación puede variar.

Supongamos que un cuidador puede "poseer" varios animales. Por lo tanto, la cardinalidad de la clase Cuidador hacia la clase Animal es 0..n (podemos imaginar que un cuidador recién contratado aún no tiene un animal bajo su cuidado). ¿Y al revés? ¿Puede un animal ser cuidado por varias personas diferentes? Uno podría imaginarlo, porque si un cuidador se va de vacaciones, parece inaceptable dejar a un animal sin cuidados. Por tanto, supongamos que la cardinalidad es 1..n (no 0..n porque podemos suponer que no dejamos un animal sin cuidados). Todos estos supuestos se deben enviar al patrocinador del software para transformarlos en funcionalidades validadas.



Un cuidador se toma vacaciones, pero ¿se debe liberar el área de memoria que ocupa cuando se elimina al cuidador? Por un lado, si el cuidador dimite, sus vacaciones ya no tienen ningún significado real ya que el empleado ya no trabaja en la empresa. Sin embargo, por motivos contables, puede resultar útil realizar un seguimiento de estos días festivos. Las dos propuestas son iguales y solo el cliente del software podrá responder. Para este ejemplo, supongamos que las vacaciones no tienen por qué ser permanentes y, por lo tanto, pueden desaparecer con quien las disfrutó.



En este tipo de situación en la que surge un dilema, es mejor hacer una pregunta abierta al cliente sin comunicar inmediatamente las posibilidades. Si la persona para quien está escrito el software tiene una respuesta clara y segura, entonces el dilema está resuelto. Pero a menudo no es el caso, y además sucede que esta persona no ha visto las cosas desde este ángulo y necesita pensar en ello, o incluso ignora por completo la dirección a tomar. En este caso, hacer sugerencias puede ser útil para, quizás, sacar una idea o un recuerdo que pueda resolver el problema.

Si, a pesar de ello, no hay respuesta y el proyecto debe avanzar, entonces es mejor optar por la solución que cubra el mayor número de casos, según la filosofía "quien puede lo más, puede lo menos".

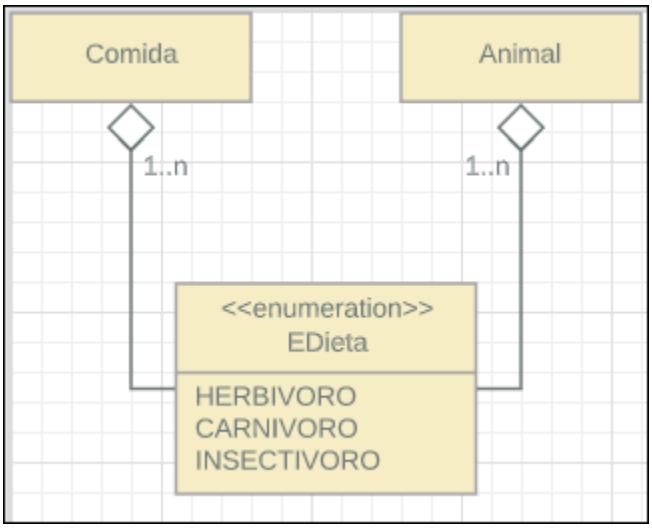
Pero ojo, esto no es motivo para complicar demasiado el proyecto y planificar absolutamente todas las posibilidades. Es necesario encontrar un equilibrio entre tomarse el tiempo para hacer que el software sea fácilmente escalable y complementar las funcionalidades para entregar el proyecto a tiempo.

El cliente describió tres dietas diferentes: carnívoro, herbívoro, insectívoro. Por cierto, aquí hay una palabra clave interesante: dieta. Este concepto permite generalizar los tres regímenes mencionados por el cliente, lo que posteriormente podría permitir agregar

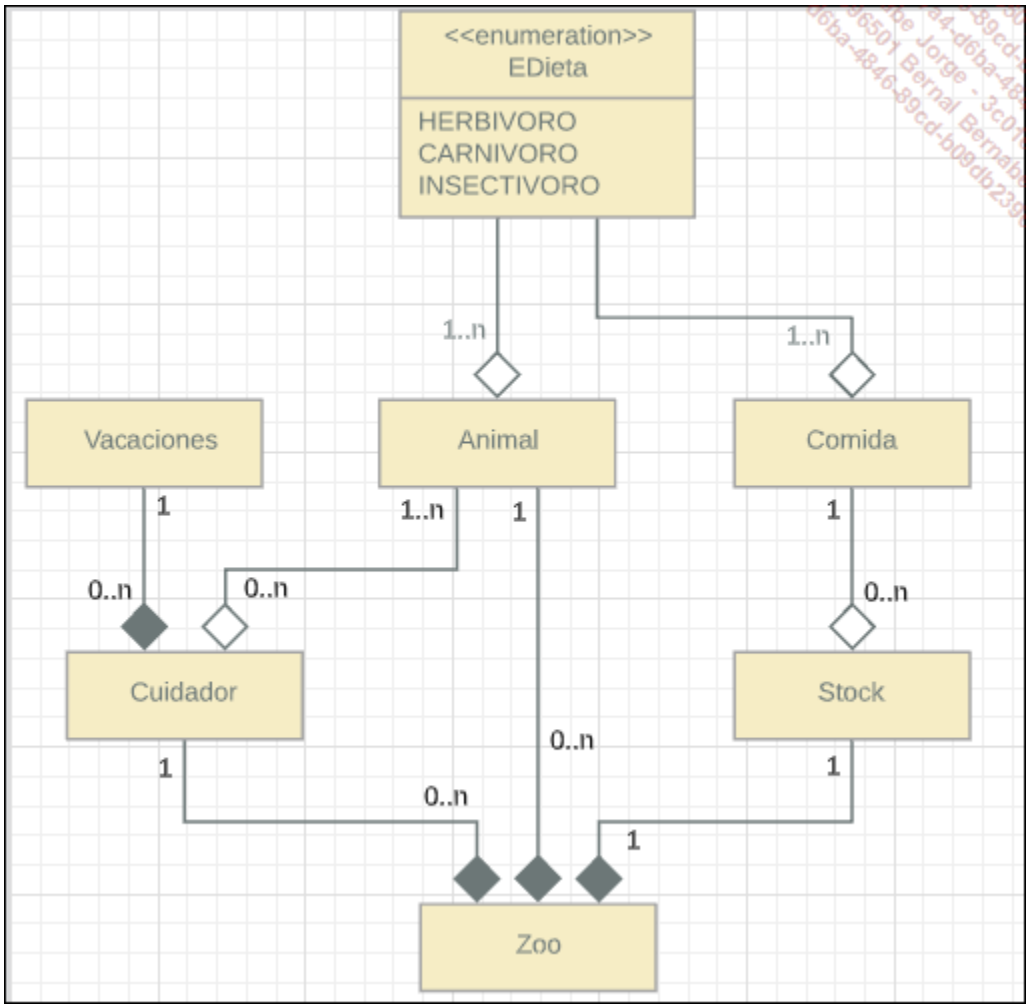
otros sin modificar en profundidad la arquitectura del software. Insectívoro, carnívoro y herbívoro son dietas: por tanto, una relación de herencia las vincula. Por definición, una dieta está estrechamente relacionada con la comida. Lo que come un animal depende de su dieta. Así, podemos considerar la dieta como el intermediario entre el animal y su alimento.

Dado que la simplicidad es un gran consejero, en lugar de crear clases y vincularlas por herencia, pensemos en otra solución: la enumeración. Una enumeración Dieta que contiene los valores Herbívoro, Carnívoro e Insectívoro. Un animal tendría un atributo de este tipo, y cada alimento también, para poder adecuarlos: tal animal cuya dieta es x puede comer tal comida adaptada a la misma dieta x. Pero, ¿no limita esto a un animal a tener una sola dieta? Hay omnívoros que comen plantas y carne. ¿Deberíamos entonces agregar "omnívoro" como valor a la enumeración? El inconveniente de esta solución es que pierde modularidad en la gestión de las dietas. De hecho, si "omnívoro" corresponde a la unión de herbívoros y carnívoros, ¿qué pasa con los herbívoros-insectívoros? ¿Qué pasa con los carnívoros-insectívoros? Agregar un valor de enumeración por combinación sería difícil de mantener dadas las diferentes combinaciones posibles. Aquí solo hay tres, pero si alguna vez aparece una cuarta…

Es preferible establecer una arquitectura lógicamente sólida que buscar soluciones aceptables solo porque, en un caso concreto, "sucede". Entonces, en lugar de un solo valor enumerado, a los animales y las comidas se les asignará una lista de esos valores.



Se acaba de dar un primer paso en el modelado del proyecto: saber qué contiene qué. Este es un paso importante porque da una descripción general de la importancia de las clases dentro del software. Cuanto más central sea una clase en las relaciones, más crítica será, por lo que será necesario prestarle una atención particular durante su implementación.



Aquí, podemos ver que la clase Zoo es muy central, porque es la que contendrá las instancias de Cuidador, Animal y Stock. Por tanto, es a ella a quien será necesario solicitar información relativa a estas tres clases (¿quién es el cuidador asociado con tal o cual animal? ¿Estará de vacaciones en tal o cual fecha? ¿Cuál es la previsión de stock en 15 días? etc.).

Solo con las agregaciones y composiciones, podemos ver que el diseño del software ya está dando un buen giro. Los conceptos de negocio descritos por el cliente están relacionados entre sí de forma lógica e intuitiva, cada uno con responsabilidad funcional.

2. Miembros

Los actores están en su lugar. Ahora es necesario asignarles sus miembros. Empecemos por los atributos.

La descripción del producto realizada por el cliente no es muy detallada en cuanto a los datos. Por lo tanto, será necesario usar el sentido común para proporcionar el nivel mínimo vital, sin perder tiempo en modelar información que no servirá para nada. De todos

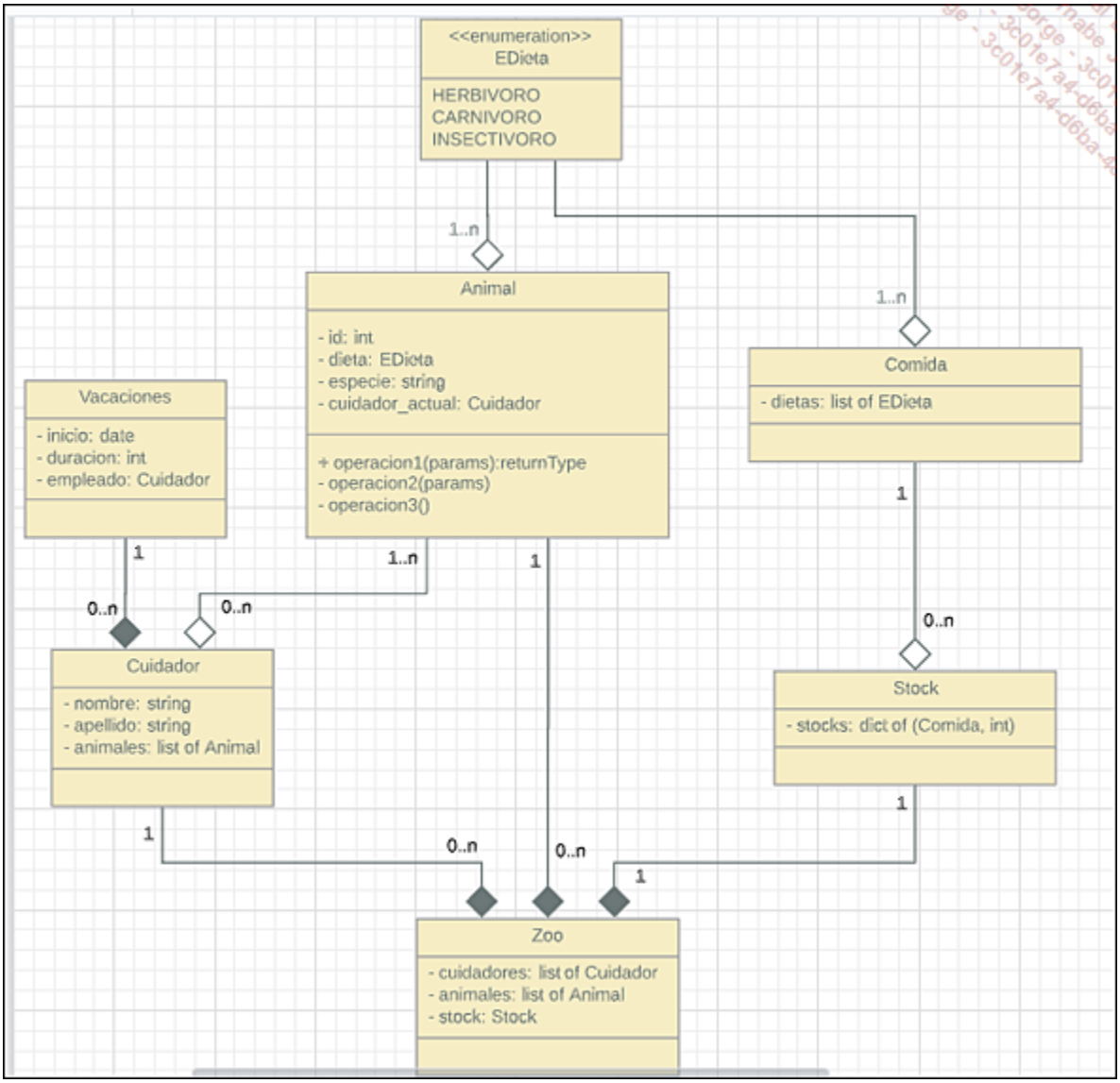
modos, antes de iniciar cualquier desarrollo todas estas decisiones deben ser validadas por el cliente, de manera constante.

- El zoológico en sí mismo no contiene necesariamente información relevante. Podríamos agregar a la clase que lo representa atributos que indiquen su nombre, dirección, número de teléfono, dirección de correo electrónico, etc. Dicho esto, el administrador de este zoológico utilizará la aplicación. Por un lado, él ya conoce esta información y, por otro lado, esta información es completamente inútil para gestionar cuidadores, animales y comida.
- Un cuidador debe poder ser identificado en un horario, por ejemplo, y en consecuencia, se le debe asignar su nombre y apellidos. Uno se podría imaginar agregando un número de Seguro Social en caso de homónimos (elección a discreción del cliente). La lista de animales que cuida también se debe establecer como un atributo. Evidentemente, esta lista será modificable en caso de vacaciones, reasignación, cambio de plantilla, etc.
- Unas vacaciones se definen por una fecha de inicio y una duración (también podríamos imaginar una fecha de fin). Para poder encontrar quién las disfrutó, debe estar presente un atributo que haga referencia al empleado en cuestión.
- Un animal tiene una dieta, que se define mediante una lista de valores de enumeración. Debe poder identificar de manera única a un animal, lo que puede ser útil si está enfermo o simplemente para asignarle un cuidador. Por tanto, se debe proporcionar un tributo a tal efecto. Dado que en un momento u otro es muy probable que desee mostrar información del animal (en una página web, aplicación móvil o software de "escritorio"), se puede agregar el nombre de la especie del animal, para ayudar con la identificación ("Cuidador Ángel asociado con el animal # 1244" es poco claro, mientras que "Cuidador Ángel asociado con el animal # 1244 (guepardo)", es un poco más explícito y permite visualizar inconsistencias más rápidamente, especialmente si Ángel teme a los felinos).
- La reserva de alimentos contiene, por necesidad, instancias de alimentos. Por su naturaleza de stock, estas instancias se deben asociar con un contador de cantidad (kilogramos, toneladas, litros, etc.). Uno podría imaginar agregar atributos estadísticos (velocidad de consumo de tal alimento, frecuencia de compra de ese otro), pero por falta de especificaciones, es contraproducente querer imaginar todos los usos posibles. Será necesario entablar una conversación con el cliente para aclarar esto.
- La comida en sí misma no debe contener mucha información, porque su único propósito es ser distribuida a los animales adecuados. Por tanto, se le deben asignar las dietas a las que corresponde, así como un nombre para que, como en el caso de los animales, se puedan leer fácilmente las existencias en una interfaz gráfica.

Esta lista sigue el proceso de pensamiento que un desarrollador podría tener en tal situación. No existe un método riguroso y matemático para establecer la lista de los atributos de un conjunto de clases. O la descripción del negocio que hemos realizado es muy precisa y, por tanto, el trabajo ya está "prestablecido", o está fraccionada. En este caso, solo el sentido común y la intuición pueden hacer avanzar esta primera versión del modelado, mientras se espera un intercambio con el cliente para sacar una segunda.

En este caso, aunque no se sepa completamente cómo funciona un zoológico, una central nuclear o un motor de helicóptero, todavía es posible imaginarlo "ingenuamente" y establecer una lista de atributos en consecuencia.

El diagrama de clases de la aplicación, una vez que se definen los atributos, se parece a lo siguiente:



Los nombres de tipos "básicos" como integer (entero), string (cadena), etc. no se corresponden necesariamente con su nombre real en el lenguaje de desarrollo (en Python, una cadena de caracteres es de tipo str, un entero es de tipo int, etc.). Esto se debe a que es necesario recordar que UML no se ocupa de la implementación, sino de la arquitectura del software y, por lo tanto, el lenguaje de desarrollo no tiene lugar aquí.

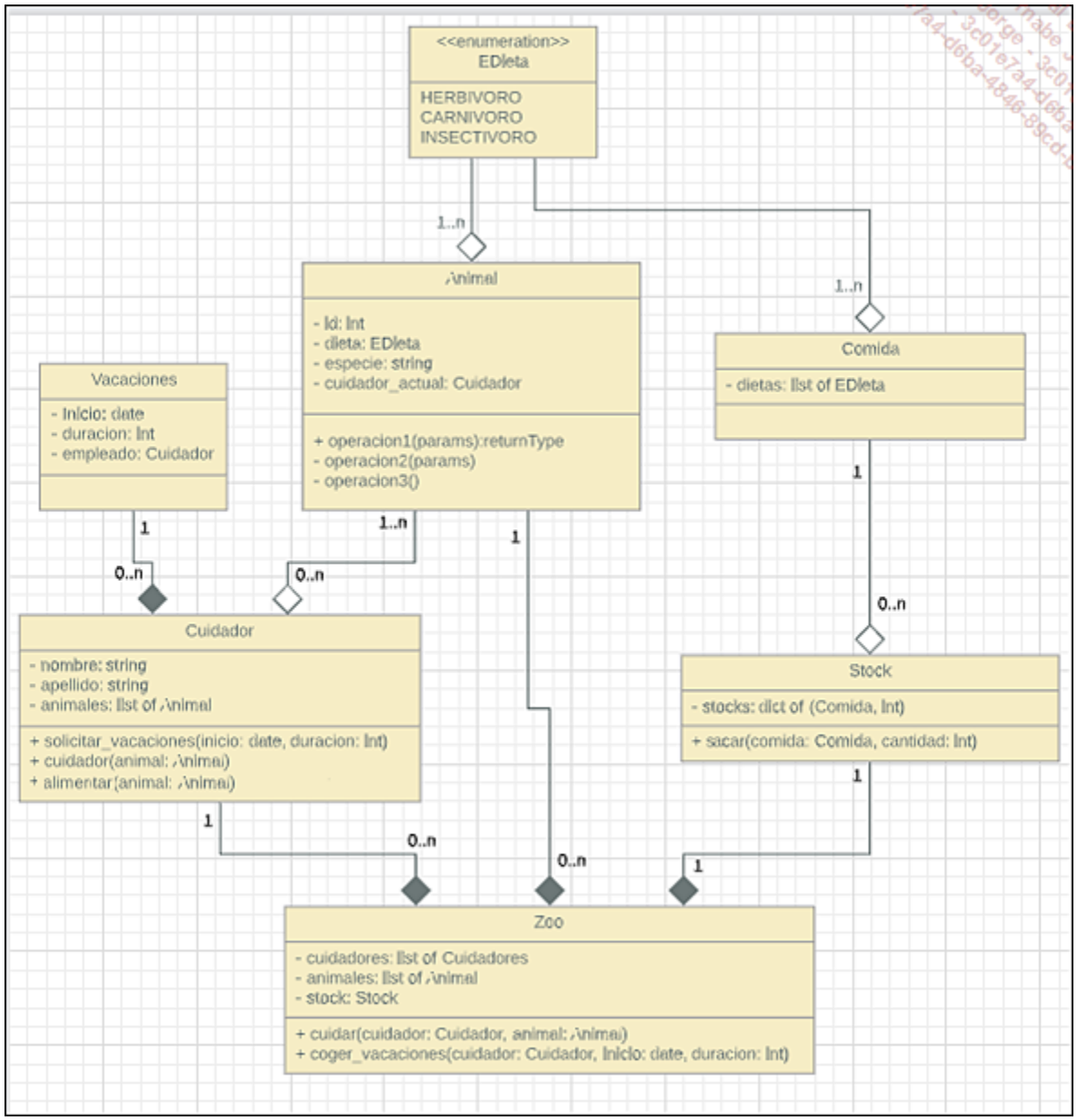
Por esta razón describimos los tipos de atributo utilizando términos más "humanos" que de "máquina".

Todos los atributos tienen visibilidad privada. Hacer que la visibilidad de los atributos sea privada es algo bueno en POO. Los descriptores de acceso de estos atributos no se han representado para evitar sobrecargar el esquema.

Ahora es el momento de pasar al comportamiento de estas clases, es decir, sus métodos.

- El zoológico, como clase principal y central, tendrá un comportamiento que impactará a las clases que contiene, es decir, los cuidadores, los animales y el stock, así como las relaciones que existen entre ellos. En el contexto del funcionamiento de un zoológico, esta clase actuará como un "jefe" (podríamos haber definido una clase dedicada a la gestión administrativa, pero en la descripción que hace el cliente, esta nueva clase no aporta ningún valor añadido). De esta manera, la asignación de cuidadores a los animales la hará el zoológico, lo que tiene sentido porque esta es la clase que tiene el conocimiento completo de los animales. Por la misma razón, también le corresponde al zoológico decidir si un empleado puede irse de vacaciones o no.
- El cuidador, por definición, cuida a los animales y también los alimenta. Dado que le corresponde a él tomar la iniciativa de solicitar las vacaciones, se debe definir un método para tal fin.
- Un periodo de vacaciones en sí mismo no tiene comportamiento propio. Es solo una clase "contenedor" que almacena datos relacionados con las vacaciones tomadas por los empleados.
- Los animales tienen un papel pasivo en el zoológico: se les alimenta y ya está.
- El stock se encarga de gestionar los alimentos y las cantidades solicitadas. Depende de él asegurarse de que las reservas no estén agotadas y, por qué no, dar una alerta cuando lo estén.

Aquí está la descripción general del diagrama de clases completo de la aplicación de administración del zoológico:



Este esquema es el resultado de una reflexión "ingenua" basada en la descripción proporcionada por el cliente. En cualquier caso, ni este primer borrador, ni los siguientes, deben grabarse en piedra: los diagramas de clases van a evolucionar con los detalles proporcionados por el cliente, los requisitos técnicos, las refactorizaciones, etc. Cuando el cliente acepta el diagrama, o más normalmente, su explicación, puede servir como base para comenzar el desarrollo.

Referencias

- **Aprender la Programación Orientada a Objetoscon el lenguaje Python**, V. Boucheny, ENI, 2021