**Definiciones:**

**Ingeniería de software:** es la aplicación de un enfoque sistemático, disciplinado y cuantificable al proceso de desarrollo, operación y mantenimiento de software.

**Modelo:** es una representaición o especificación desde un determinado punto de vista y con un objetivo en concreto.

**Modelo:** es una representación o especificación desde un determinado punto de vista y con un objetivo en concreto.

**Diseño:** conjunto de planes y decisiones para definir un producto con los suficientes detalles como para permitir su realización física de acuerdo a unos requisitos.

**Patrón de diseño:** solución probada que puede aplicarse con éxito a un determinado tipo de problemas que aparecen repetidamente en el desarrollo de software.

**Modelo:** es una representación o especificación desde un determinado punto de vista y con un objetivo en concreto.

**Diferencias entre el patrón estado y el patrón estrategia.**

El patrón estrategia define distintas versiones de un algoritmo para una misma tarea y que pueden ser seleccionados por el usuario en tiempo de ejecución. De esta forma es posible cambiar la implementación de un algoritmo en tiempo de ejecución, dependiendo de ciertos criterios. Por otro lado el patrón estado nos ofrece distintos comportamientos de un objeto dependiendo del estado en el que se encuentre. La diferencia entre ambos patrones es que el patrón estado cambia el comportamiento de un objeto por completo, mientras que el patrón estrategia, trabaja a nivel de método, es decir, cambia el comportamiento de un método en concreto.

Ambos patrones pueden trabajar juntos.

**Refactoring:**

El refactor es el proceso de modificación de la estructura del código fuente de un proyecto sin modificar el comportamiento del mismo.

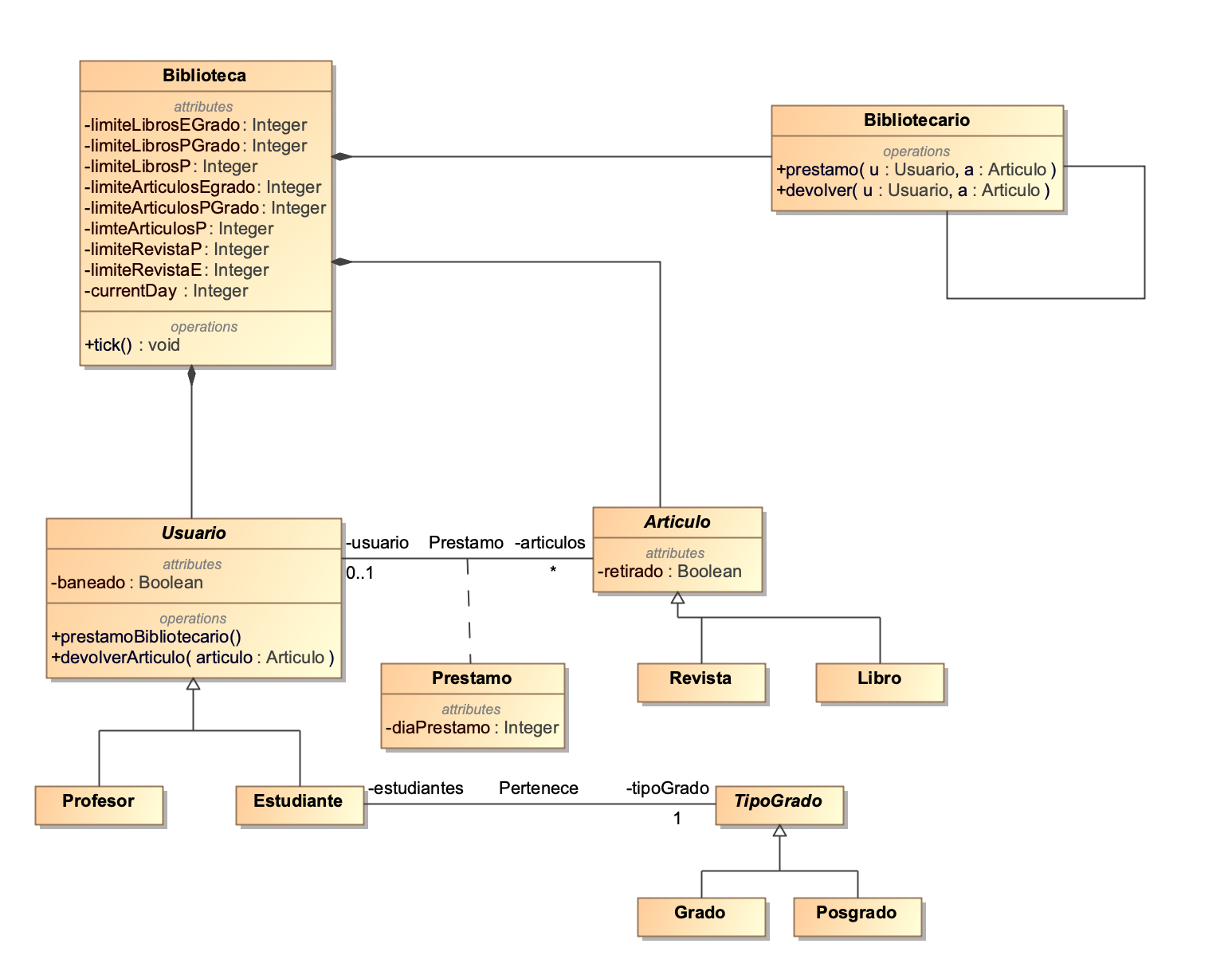
El proceso de refactor comienza cuando encuentras un fragmento de código que “huele mal” o cuando encuentras parte de código que puede mejorarse.

Seleccionar el peor “smell”, después se selecciona el refactor que corrija ese bad smell. Acto seguido se aplica la refactorización, y por último, se comprueba que el comportamiento del programa no haya sido modificado.

**Ejercicio 1:**

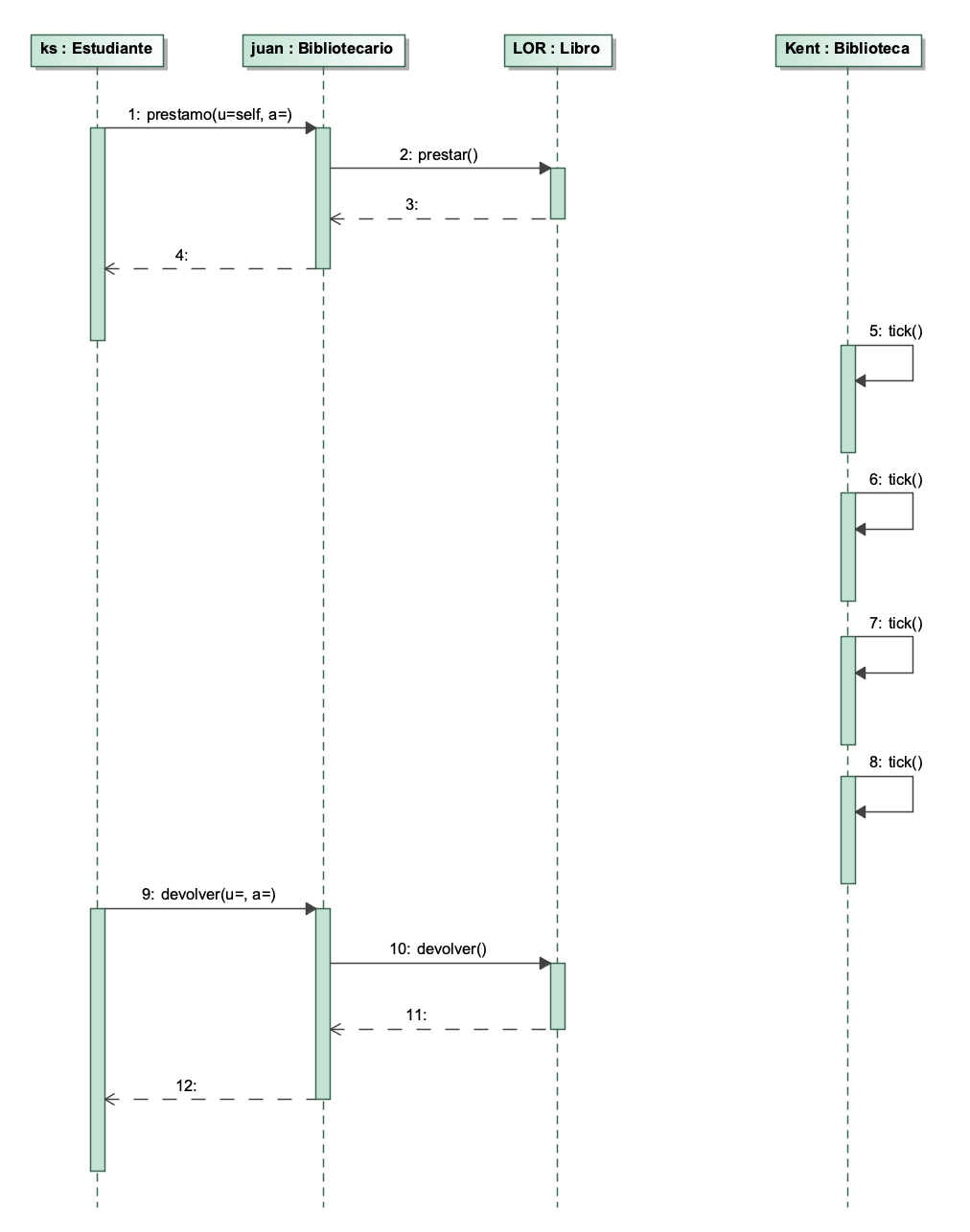
1. Para poder modelar el paso del tiempo en este problema vamos a necesitar crear la clase Biblioteca. La clase biblioteca tendrá las restricciones de integridad mencionadas en el problema, es decir, el máximo de días que tiene un usuario para devolver los libros, dependiendo de qué tipo de usuario es. Además, es una manera de tener centralizado todo. Para evitar repetir relaciones innecesarias vamos a crear una clase abstracta Usuario, del que extenderán 2 clases, Profesor y Estudiante. El estudiante, además, tendrá una relación con el tipo de grado, otra clase abstracta, de la que heredan Grado y Posgrado. Nuestro modelo también permite el préstamo de artículos, otra clase abstracta, que se generaliza en Revistas y Libros. En el enunciado se especifica además que el bibliotecario tendrá que ser quién haga los préstamos a los usuarios, para ello, el bibliotecario tendrá definida la operación prestamo() en el que recibirá como argumentos el usuario que pide el préstamo y el artículo a prestar. Por último, tendremos una operación que nos permite modificar las variables de límites de fecha y de artículos que pueden retirar los usuarios. La operación de devolver del bibliotecario también devuelve al usuario para saber si ha excedido el límite de fecha que tenía para el libro. Nuestro sistema quedará definido de la siguiente manera en MagicDraw.

Diagrama UML:

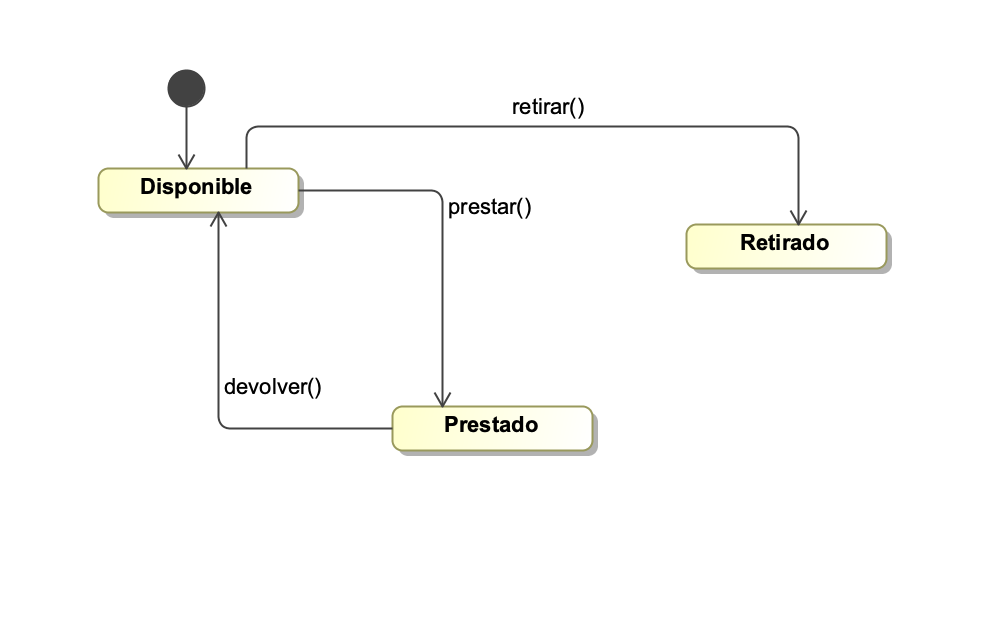


1. En el diagrama de secuencia observamos como pasan 4 días desde que pide el préstamo del libro al bibliotecario hasta que lo devuelve, al devolverlo dentro de la fecha establecida, no se le aplica ninguna penalización.

Diagrama de secuencia:



1. Para el diagrama de estados del Artículo, vamos a necesitar 3 operaciones, de las cuales 2 de ellas ya han sido definidas en el anterior apartado, que son prestar y devolver dentro de la clase Artículo, pero nos falta la operación de retirar un artículo. Por tanto, nuestro libro tendrá 3 estados, Disponible, Prestado, Retirado. La operación de retirar sólo podrá ser llamada por el jefe de la biblioteca. La máquina de estados queda como sigue:



**Ejercicio 2:**

1. Para resolver este problema vamos a utilizar el patrón decorador, que frente a la solución anterior, genera ventaja a la hora de combinar varios tipos de texto. El patrón decorador nos ofrece añadir a nuestro componente original, en este caso el texto, comportamiento adicional sin necesidad de tener que crear subclases de combinaciones de clases, que hacen que nuestra cantidad de objetos crezca de manera exponencial. A pesar de ser pocas clases las que hay que crear en este ejemplo, si tuviéramos 25 clases, para crear todas las combinaciones de ellas, tendríamos que generar una cantidad ingente de clases. Esto provoca que nuestro código sea prácticamente imposible de mantener y de ampliar.
2. WOW, WHAT A SURPRISE