Análisis y diseño de software
Informe - Práctica 3
Fecha: 22/3/2017
Profesor: Miguel Ángel Mora
Grupo 2201

Pareja Nº 4
Luis Cárabe
(luis.carabe@estudiante.uam.es)
David García
(david.garcia03@estudiante.uam.es)

Introducción

En este informe explicamos las decisiones de diseño más relevantes y necesarias para la implementación de los distintos ejercicios de los que se compone la *Practica 3* de la asignatura de *Análisis y diseño de software*.

Apartado 1

Gestión de obras

En este ejercicio hemos creado todas las clases necesarias y sus métodos más básicos. La clase abstracta *Obra* contiene los datos comunes a libros y películas. Al especificar el enunciado que los atributos *autor*, *titulo* y *año* no pueden ser modificados los hemos declarado como **final** y no hemos implementado sus respectivos "setters".

Por otro lado, hemos creado una enumeración en la clase *Pelicula* con los diferentes generos a los que pueden pertenecer las películas. Como el plazo de cada obra es común a libros y películas, pero en cada uno de ellos su valor inicial es distinto hemos decidido que sea un campo de la clase abstracta *Obra*. En el constructor de la misma pasamos el valor del plazo inicial, de esta manera al construir *Libro* y *Pelicula* llamamos al constructor de la clase abstracta con el valor específico de cada una.

En cuanto al hecho de que los plazos varíen segun el número de veces que han sido prestados ha sido implementado en una función **prestar()**. Esta función está implementada en *Libro*, donde aumenta un contador que registra el número de veces que ha sido prestada una obra y disminuye el plazo si alcanza un múltiplo de 10. En *Pelicula* el método está declarado pero no ha sido implementado (en el enunciado dice que puede ser implementado en futuras versiones).

Gestión de usuarios

Hemos creado la clase abstracta *Usuario* de la que heredan *Publico* y *Empleado*. Estas dos clases se diferencian por la forma de bonificar y sancionar por la devolución de obras. Por esa razón hemos creado los métodos que sancionan o que eliminan una sanción de manera abstracta y los hemos implementado en las clases herederas. En el caso de la clase *Publico* no se ha implementado el método que elimina la sanción ya que no es una acción posible para este usuario.

Un método importante para estas clases es **anyadirPrestamo()**. Este método realiza varias acciones: comprueba que un ejemplar de una obra pueda ser prestado (que no se haya superado el máximo de préstamos y que no haya dos ejemplares de la misma obra), añade el prestamo al array del usuario y llama a la función **prestar()** de *Obra* explicado anteriormente. En el caso de los usuarios públicos, además maneja las bonificaciones.

En cuanto al manejo de sanciones, tenemos dos métodos. El método **sancionarPorRetraso()** aumenta la cantidad de dinero a deber en el caso de los empleados y elimina las bonificaciones a los usuarios públicos.

Apartado 2

En este apartado únicamente hemos tenido que implementar el método **equals()** de la clase *Obra*. Sobreescribiendo el ya existente simplemente devolvemos true en el caso de que el nombre del autor y el título coincidan. Para distinguir entre un usuario u otro no ha sido necesaria ninguna implementación adicional, ya que el método predefinido tiene la funcionalidad deseada.

Apartado 3

Una de las principales decisiones de diseño tomadas en este apartado ha sido la utilizada para asignar un identificador numérico a cada ejemplar de una obra. Nuestra implementación ha consistido en crear una variable estática propia de la clase *Ejemplar* llamada *lastID*. El valor de esta variable comienza siendo uno, pero tras cada llamada al constructor de la clase este valor es asignado al id del ejemplar y es incrementado. Es decir, el id de cada ejemplar va a ser único en la biblioteca, independientemente de la obra a la que haga referencia.

Por otro lado los métodos **prestar()** y **devolver()** de la clase *Ejemplar* han sido implementados de manera sencilla gracias a los métodos de otras clases. El primero realiza una llamada a la función **anyadirPrestamo()** de *Usuario*, esta realiza las comprobaciones necesarias para validar el préstamo y lo añade en el array de préstamos del usuario. Ésta, a su vez, llama a la función **prestar()** de *Obra* explicada anteriormente. El método **devolver()** por su parte llama a la función **eliminarPrestamo()** que elimina el prestamo del usuario y le aplica la sanción (si es necesaria). También hace una llamada al método **devolucion()** de *Prestamo* que junto a **addNuevoPrestamo()** maneja el Array List donde almacenamos los prestamos no devueltos todavía.

Apartado 4

Como el cálculo de sanciones ya había sido implementado, nos hemos centrado en crear métodos que nos permita saber cúando se da el vencimiento de los préstamos. Hemos cambiado las variables que en *Prestamo* se guaradaban como un tipo int por listas donde guardamos los préstamos no devueltos.

Para ello hemos creado los métodos **diasAtrasado()**, **conVencimientoHoy** y **pasadosDeVencimeinto()**. El primero nos devuelve el número de días que un préstamo está atrasado (útil para simplificar código en los métodos de las sanciones). Los otros dos iteran sobre las listas y, ayudados de la primera función muestran los préstamos que cumplen las respectivas condiciones.

Apartado 6 (opcional)

Para la realización de este apartado hemos dado privacidad *private* al constructor de la biblioteca, de esta manera hemos creado una única instancia que será la que maneje toda la información de la aplicación.

Para eliminar ejemplares simplemente hemos creado el método **eliminarEjemplar()** que comprueba si el ejemplar está prestado (ya que la clase *Ejemplar* tiene una referencia a préstamo, que en caso de ser null quiere decir que no está prestado), y después lo eliminamos del array de préstamos disponibles del que dispone la biblioteca. Nótese que por eliminar el préstamo de esta lista no se elimina su instancia, la cual estará disponible en el registro de préstamos historicos que puede ser consultado en cualquier momento, aunque el ejemplar esté fuera de circulación.

Para eliminar obras hemos pensado que la solución más consistente consiste en que, si alguno de los ejemplares está prestado, ni la obra ni ninguno de sus ejemplares puedan ser eliminados. Por ello en el método **eliminarObra()** comprobamos uno a uno el conjunto de ejemplares de una obra (para lo que hemos tenido que añadir un getter en *Obra*). Después eliminamos uno a uno cada ejemplar del array de disponibles y eliminamos la obra del catálogo.

Diagrama de clases:

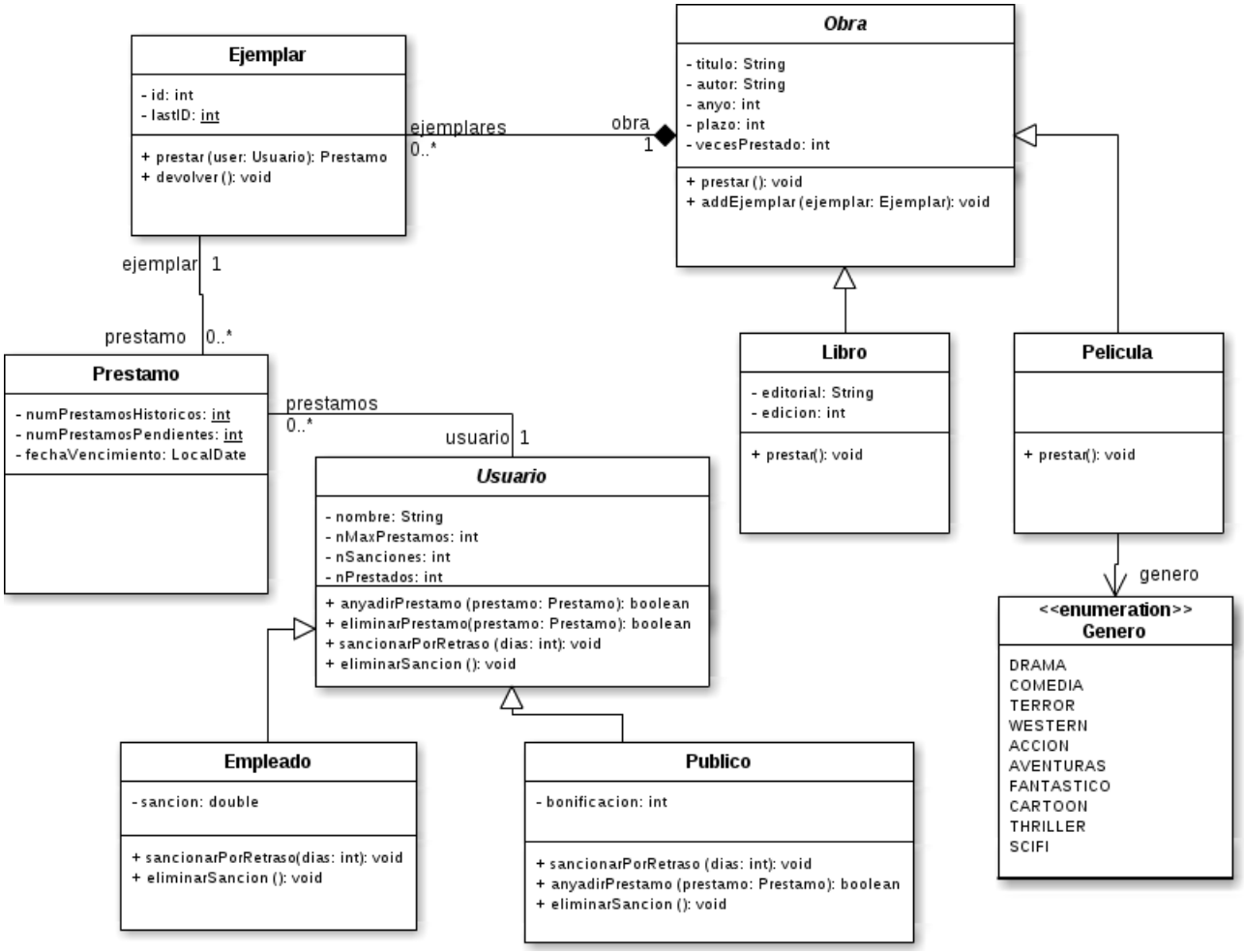

Figura 1: Diagrama de clases Apartados 1 - 4