# Programación II GMI

# Ejercicio: Club del Lector

## Dpto. LSIIS. Unidad de Programación

La finalidad del presente ejercicio es:

1. Que el alumno practique el uso del TAD ArrayList visto en clase.
2. Que el alumno tome contacto con el sistema web de entrega de prácticas.
3. Que el alumno tome contacto y saque partido a los informes proporcionados por SonarQube. Para acceder a SonarQube se debe tener una IP de la escuela o de la universidad. Si se está en casa o se usa eduroam hay que conectarse la VPN de la escuela (https://www.fi.upm.es/?pagina=373) o a la VPN de la universidad (https://www.upm.es/UPM/ServiciosTecnologicos/vpn). La URL de referencia es http://sonar.fi.upm.es:9000. El usuario y la clave son los mismos que los usados para la LDAP de **la escuela**, que consiste en el número de matrícula con la letra y la clave correspondiente.

**Este ejercicio se hace de forma individual**.

Se pretende realizar una implementación simplificada de un sistema gestor para un club de lectura. El club consta de un conjunto de libros y de socios 'Lectores' que cogen libros, los leen y los retornan. En este ejercicio nos vamos a centrar sólo en las clases Libro y Lector. A continuación, se indica qué atributos y servicios tiene cada uno.

## Diagrama de clases

En la Figura 1 se muestra el diagrama de clases correspondientes a las clases del paquete **ee1**. En este diagrama no se incluye la clase auxiliar Fecha. En el diagrama tampoco se muestran los constructores, ni los gets o sets (que pudieran ser necesarios). El alumno también tendrá que decidir si son necesarios o no más atributos para poder implementar la solución.

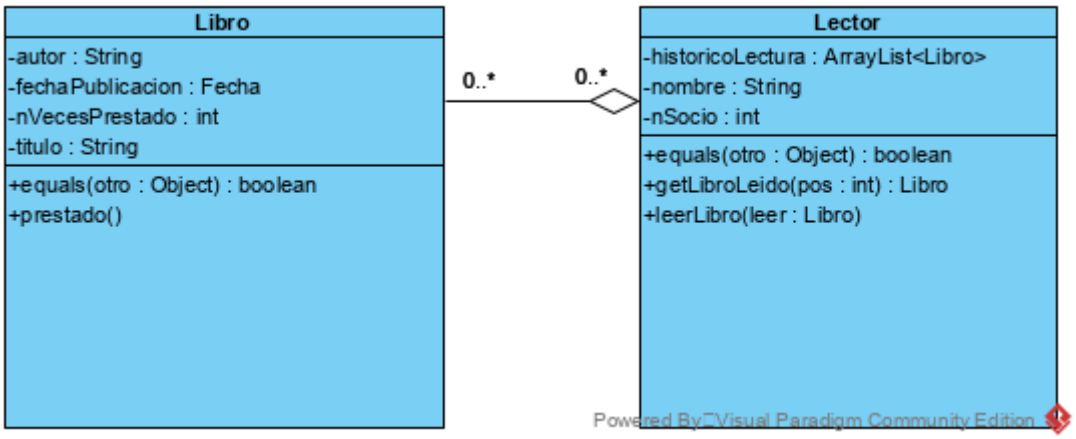



*Figura 1 Diagrama de clases*

## Clase Libro

Esta clase representa un libro y se encuentra en el paquete ee1. La información que representa (estado) un libro es:

- **titulo**: de tipo String que contendrá el título del libro.
- **autor**: de tipo String que contendrá el nombre y apellidos del autor (para este problema se asume que cada libro sólo tiene un autor).
- **fechaPublicacion**: Atributo de tipo fecha, que contendrá la fecha en que fue publicado.
- **nVecesPrestado**: número entero que lleva la cuenta de las veces que ha sido prestado.

Los servicios que ofrecen las instancias de la clase libro son:

- **Constructor**, inicializa los atributos necesarios. Tenga en cuenta que java establece valores por defecto para los atributos del objeto. El constructor no debe inicializar aquellos atributos que estén correctamente inicializados con el valor por defecto.
- **Constructor Copia:** Recibe como parámetro un libro y crea una instancia nueva **sacando una copia de cada uno de sus atributos**. Nota: no se admitirá como implementación válida la de aquel constructor copia que sólo copie las referencias.
- Se ofrecerá gets de todos los atributos.
- Se ofrecerá un servicio llamado '**prestado**' que incrementará en uno el número de veces que el libro ha sido prestado.
- Un método **equals** que indicará si el libro es igual a otro pasado como parámetro. Dos libros son iguales **si y sólo si coinciden el título, el autor y la fecha de publicación**. Para completar este método, el alumno solo tendrá que implementar el método auxiliar privado esIgual().

## Clase Lector

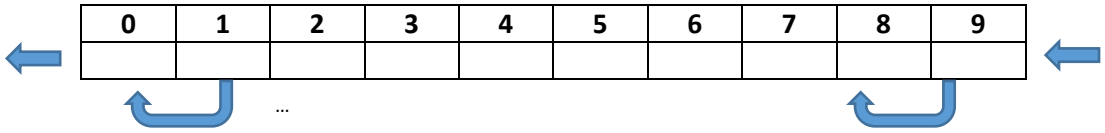
Esta clase representa al lector. Esta clase se ubica en el paquete ee1. Estará formada por los siguientes atributos:

- **nSocio**: entero que representa el número de socio.
- **nombre**: Contiene el nombre del socio.
- **historicoLectura**: lista (ArrayList) que mantiene un histórico de los últimos **10** libros que el lector haya leído. En la posición cero está el libro que hace más tiempo se leyó y en la última posición ocupada el libro que se ha leído más recientemente.

Los servicios que ofrecen las instancias de la clase Lector son:

- **Constructor**, inicializa los atributos necesarios. El constructor **no** recibirá información sobre el histórico de libros leídos ya que este histórico se irá rellenando durante la vida del objeto.
- Gets para número de socio y nombre.
- **getLibroLeido**: Da el libro que se encuentra en el histórico en la posición que se recibe como parámetro. La posición **uno** hace referencia al primer libro que hay en el histórico. Si la posición está fuera del rango, se debe retornar **null**. Si la posición se encuentra entre **1 y el número de libros** en el histórico, entonces se retorna la referencia del libro.
- **leerLibro**: Servicio que recibe una instancia de libro, lo inserta en el histórico y **le indica al libro que ha sido prestado**. Este método funciona de la siguiente forma:
    1. Si el número de libros leídos por el lector **es menor que 10**, este libro se inserta después del último libro que hubiera en el histórico.

2. Si el número de libros leídos **es mayor o igual que 10,** se descarta el libro que hizo más tiempo que se leyó (se desplazan los libros una posición hacia delante dejando un hueco al final) y en el hueco dejado al final se insertará el libro recibido como entrada.




- **equals**: Se considera que dos lectores son iguales cuando coincide el número de socio y los libros leídos que hay en el histórico son los mismos y están en el mismo orden. Para completar este método, el alumno solo tendrá que implementar el método auxiliar privado esIgual().

## Código de apoyo

Al alumno se le proporciona un zip que contiene los materiales necesarios para realizar el ejercicio a excepción de la librería **CorrectorOffline.jar**, que deberá ser descargada por el alumno del sitio Moodle de la asignatura. La estructura del zip es:

- Fecha.jar: Librería que contiene la clase fecha. Se debe añadir al proyecto (Built Path -> Add External Libraries -> …) para poder utilizar dicha clase.
- docFecha: Carpeta o directorio que contiene la documentación de la librería fecha en formato javadoc (páginas html).
- src: Carpeta o directorio que contiene el código de apoyo. Esta carpeta a su vez tiene:
  - TestLectorAlum.java: Pruebas del lector implementadas con un main
  - TestLibroAlum.java: Pruebas del libro implementadas con un main
  - ee1: Carpeta o directorio que define el paquete en el que están la clase Libro y Lector. Se proporcionan las clases Lector.java y Libro.java vacías. El alumno debe completarlas según lo especificado en el enunciado
  - test: Carpeta o directorio que define el paquete en el que están los tests en formato JUnit versión 4: TestLector.java y TestLibro.java

## Consideraciones a la hora de entregar

El alumno debe entregar los archivos **Libro.java y Lector.java, a través del sistema de entrega: https://entrega.fi.upm.es/**. Tanto la clase Libro como la clase Lector deben estar en el paquete **ee1**. El alumno debe añadir al proyecto la librería CorrectorOffline.jar (Built Path -> Add External Libraries -> …) y colocar en el fichero Libro.java (después del último import) los datos personales siguiendo el siguiente formato:

```
import anotacion.*;

@Programacion2 (

        nombreAutor1 = "nombre",            // (del alumno 1)

        apellidoAutor1 = "apellido1 apellido2",  // (del alumno 1)

        emailUPMAutor1 = "usr@alumnos.upm.es"   // (del alumno 1)

)
```

Al alumno se le proporcionan dos programas de prueba uno para Libro (TestLibroAlum.java) y otro para lector (TestLectorAlum.java), estos dos programas se encuentran en el paquete test.

Las pruebas no son exhaustivas. **No se deben modificar las pruebas que hay en estos archivos, pero si se pueden añadir nuevas pruebas después de la última que exista en cada uno de los archivos**. Asimismo, se proporciona un paquete **test** con pruebas JUnit. Si el alumno desea utilizar estas pruebas deberá incorporar las librerías JUnit de la versión 4 al proyecto eclipse (Built Path -> Add Libraries -> …). Para ejecutar las pruebas JUnit del paquete test, se puede seleccionar el paquete test en el *package explorer* de Eclipse y luego ejecutar el comando *Run* (seleccionado JUnit Test). Se recomienda probar primero la clase Libro y cuando esta pase las pruebas proceder con la clase Lector.

El alumno debe verificar que su código compila con los ficheros de prueba suministrados. Si el código no compila, la entrega será desestimada. Por otro lado, para que pueda ser admitida la entrega, es necesario que el código entregado supere los siguientes métodos tests de los JUnitTests:

- TestLector::testGetNombre
- TestLector::testGetnSocio
- TestLector::testLeerLibro
- TestLector::testGetLibroLeido0
- TestLector::testGetLibroLeido4
- TestLector::testAtributos
- TestLibro::testEsIgual
- TestLector::testEsIgual3
- TestLibro::testGets
- TestLibro::testPrestado
- TestLibro::testConstructorCopia

El alumno dispondrá de un **máximo de 10 entregas**. El sistema de entrega se quedará con la última entrega apta, aunque no sea la que consiga la mayor nota.