Curso 2019-2020. Cuatrimestre Primavera. 12 de junio de 2020

Programación Orientada a Objetos. Examen Final

SOLUCIÓN

Nos piden hacer la gestión de la Biblioteca del Campus Nord a partir del diagrama de clases UML que podéis ver al final.

La biblioteca gestiona libros, usuarios (que pueden ser profesores o alumnos) y préstamos de libros que se hacen a los usuarios. Además, se mantiene una colección de autores que contiene todos los autores de los libros de la biblioteca.

La biblioteca gestiona los usuarios, libros, autores y préstamos de manera que sea fácil identificarlos por su NIF, ISBN, apellidos e identificador de préstamo, respectivamente. Cada libro puede tener más de un autor, y el orden es importante. Asumimos que no hay más de un autor con los mismos apellidos, y que cada autor puede haber escrito más de un libro. La clase autor debe contener los libros escritos por ese autor, identificados a través del ISBN.

Consideramos que en esta biblioteca no habrá más de un ejemplar de un libro. Así, una vez que un libro está en préstamo a un usuario, el libro deja de estar en la biblioteca hasta que el usuario lo devuelva (acabe el préstamo). Para el propósito de este examen, un alumno sólo puede tener tres libros en préstamo, mientras que un profesor puede tener tantos libros como desee mientras no hayan sido escritos por él mismo.

NOTA 1: Al resolver las preguntas del examen podéis suponer implementadas los métodos **getXXX() y setXXX()** que acceden a los atributos de las clases, aunque no aparezcan en el UML. **No es necesario que los implementéis**.

NOTA 2: Si al implementar alguno de los métodos que se solicitan en el examen necesitáis de algún **método diferente a los getXXX() y setXXX(), DEBERÉIS IMPLEMENTARLO**.

NOTA 3: A lo largo del examen, en la implementación de los métodos que se piden, y siempre que sea posible, debéis utilizar los métodos descritos en los apartados anteriores **INDEPENDIENTEMENTE DE QUE HAYÁIS O NO GENERADO SU CÓDIGO**.

NOTA 4: El atributo estático `siguienteId` en la clase `Prestamo` (notad que está subrayado en el diagrama UML) se utiliza para calcular el identificador único de cada objeto `Prestamo` que se cree. El identificador del primer objeto `Prestamo` creado será 1, el del segundo creado 2, y así sucesivamente.

Ejercicio 1 **(1,5 punto). A partir del diagrama (incompleto) de clases UML mostrado en la última hoja de este enunciado**, escribid para todas las clases e interfaces mostradas en el diagrama UML, la parte de código de la definición que comienza con el "`public class…`" correspondiente a la declaración de los todos los atributos de las clases, incluyendo aquellos que aparecen al implementar las relaciones. NO implementéis todavía ningún método.

```
public class Biblioteca {
    private Map<String, Usuario> usuarios;      // key: nif
    private Map<String, Libro> libros;          // key: isbn
    private Map<String,Autor> autores;          // key: apellidos
    private Map<Integer, Prestamo> prestamos;   // key: id
    private Scanner in;
}

public class Libro {
    private String titulo;
    private List<Autor> autores; // Lista porque el orden de los autores es importante
    private String isbn;
```

```
}

public class Prestamo {
    private int id;
    private static int siguienteId = 0;
    private Libro libro;
    private Usuario usuario;
}

public abstract class Persona {
    protected String nombre;
    protected String apellidos;
}

public class Autor extends Persona {
    private Map<String, Libro> librosPublicados; // key: isbn
}

public abstract class Usuario extends Persona {
    protected String nif;
    protected Set<Prestamo> prestamos;    // puede ser List o Map
}

public class Alumno extends Usuario {
    // NADA
}

public class Profesor extends Usuario {
    // NADA
}
```

Ejercicio 2 **(1,5 puntos)**. Implementad el constructor de las clases `Biblioteca`, `Persona`, `Usuario`, `Alumno` y `Autor`. Decidid los argumentos necesarios para cada constructor.

```
public Biblioteca() {
    this.usuarios = new TreeMap<>();          // pot ser HashMap
    this.libros = new TreeMap<>();            // pot ser HashMap
    this.autores = new TreeMap<>();     // pot ser HashMap
    this.prestamos = new TreeMap<>();   // pot ser HashMap
    this.in = new Scanner(System.in);
}

public Persona(String nombre, String apellidos) {
    this.nombre = nombre;
    this.apellidos = apellidos;
}

public Usuario(String nombre, String apellidos, String nif) {
        super(nombre, apellidos);
        this.nif = nif;
        prestamos = new HashSet<>();  // pot ser TreeSet
}

public Alumno(String nombre, String apellidos, String nif) {
       super(nombre, apellidos, nif);
}
public Autor(String nombre, String apellidos) {
       super(nombre, apellidos);
       librosPublicados = new TreeMap<>(); // pot ser HashMap
```

```
}
```

Ejercicio 3 **(1,5 puntos)**. Implementad el método `toString()` de las siguientes clases:

1. La clase `Usuario`. Se debe devolver un String con todos los atributos, excepto los libros que tiene el usuario en préstamo.

```
public String toString() {
    return "" + nombre + " " + apellidos + " (NIF: " + nif + ")";
}
```

2. La clase `Libro`. Se debe devolver un String con todos los atributos del libro. De los autores, sólo deben aparecer el nombre y el apellido.

```
    public String toString() {
        StringBuilder s = new StringBuilder("\"" + titulo + "\" de ");
        for(Autor a: autores) {
            s.append(a).append(", ");
        }
        s.append("ISBN: ");
        s.append(isbn);
        return s.toString();
    }

// Esta solución asume que Autor tiene un método toString() diferente del de Object
public String toString() {
       return "" + nombre + " " + apellidos;
}
```

Ejercicio 4 **(2 puntos)**. Implementad en la clase `Biblioteca` el siguiente método:

`public void devolucionesPendientes(Usuario usuario)`

Este método localiza todos los libros que tiene el usuario en préstamo.

Si el usuario no tiene ningún libro en préstamo, se escribirá por pantalla el mensaje *"El usuario xxx no tiene actualmente ningún libro en préstamo.",* donde xxx es el NIF del usuario correspondiente.

En caso contrario, los libros que tiene el usuario en préstamo se deben guardar en el fichero "Prestamos_xxxx", donde xxxx es el NIF del usuario correspondiente. La primera línea del archivo mostrará los atributos del usuario excepto los libros. A esta línea le seguirán tantas líneas como libros tiene el usuario en préstamo. En cada una de estas líneas se mostrarán todos los atributos de un libro.

Recordad que la apertura de ficheros y su escritura puede producir una excepción del tipo `IOException`.

```
    public void devolucionesPendientes(Usuario usuario) {
        Set<Prestamo> pendientes = usuario.getPrestamos();
        if (pendientes.isEmpty()) {
            System.out.println("El usuario " + usuario.getNif()
                    + " no tiene actualmente ningún libro en préstamo.");
        } else {
            String nombreFichero = "Prestamos_" + usuario.getNif();
            try {
                FileOutputStream fos = new FileOutputStream(nombreFichero, false);
                PrintWriter pw = new PrintWriter(fos);
                pw.println(usuario);
                for (Prestamo prestamo : pendientes) {
                    pw.println(prestamo.getLibro());
                }
```

```
                pw.close();
            } catch (IOException e) {
                System.out.println("Error en la gestión del archivo. Detalles: " +
                e.getMessage());
            }
        }

    }
```

Ejercicio 5 **(1,5 punto)**. Implementad en la clase `Alumno` el siguiente método:

`public void puedePrestarse(Libro libro) throws PrestamoException`

Este método lanza una excepción `PrestamoException` si el alumno ya tiene en préstamo el número máximo de libros permitido. Se asume que el libro pasado por argumento siempre está disponible para ser prestado.

```
    public void puedePrestarse(Libro libro) throws PrestamoException{
        if(this.prestamos.size()==3){
            throw new PrestamoException("El alumno "+ nombre + " " +
                    apellidos + " ha intentado tomar prestado el libro " +
                    libro.getTitulo() + ", pero ya ha llegado al máximo "
                    + "de libros prestados.") ;
         }
    }
```

Implementad, también, en la clase `Profesor` el método:

`public void puedePrestarse(Libro libro) throws PrestamoException`

Este método lanza una excepción `PrestamoException` si el profesor está entre los autores del libro pasado como argumento. Se recuerda que no hay límite superior al número de libros que un profesor puede tomar en préstamo. Se asume que el libro pasado como argumento siempre está disponible para ser prestado.

```
    public void puedePrestarse(Libro libro) throws PrestamoException {
        List<Autor> autores = libro.getAutores();

        for (Autor autor : autores) {
            if (autor.getNombre().equals(this.nombre) &&
autor.getApellidos().equals(this.apellidos)) {
                throw new PrestamoException("El profesor " + this.nombre + " "
                        + this.apellidos + " ha intentado sacar el libro "
                        + libro.getTitulo() + " del que es autor.");
            }
        }
    }
```

Ejercicio 6 **(2 puntos)**. Implementad en la clase `Biblioteca` el método:

`public void realizaPrestamo(Libro libro, Usuario usuario)`

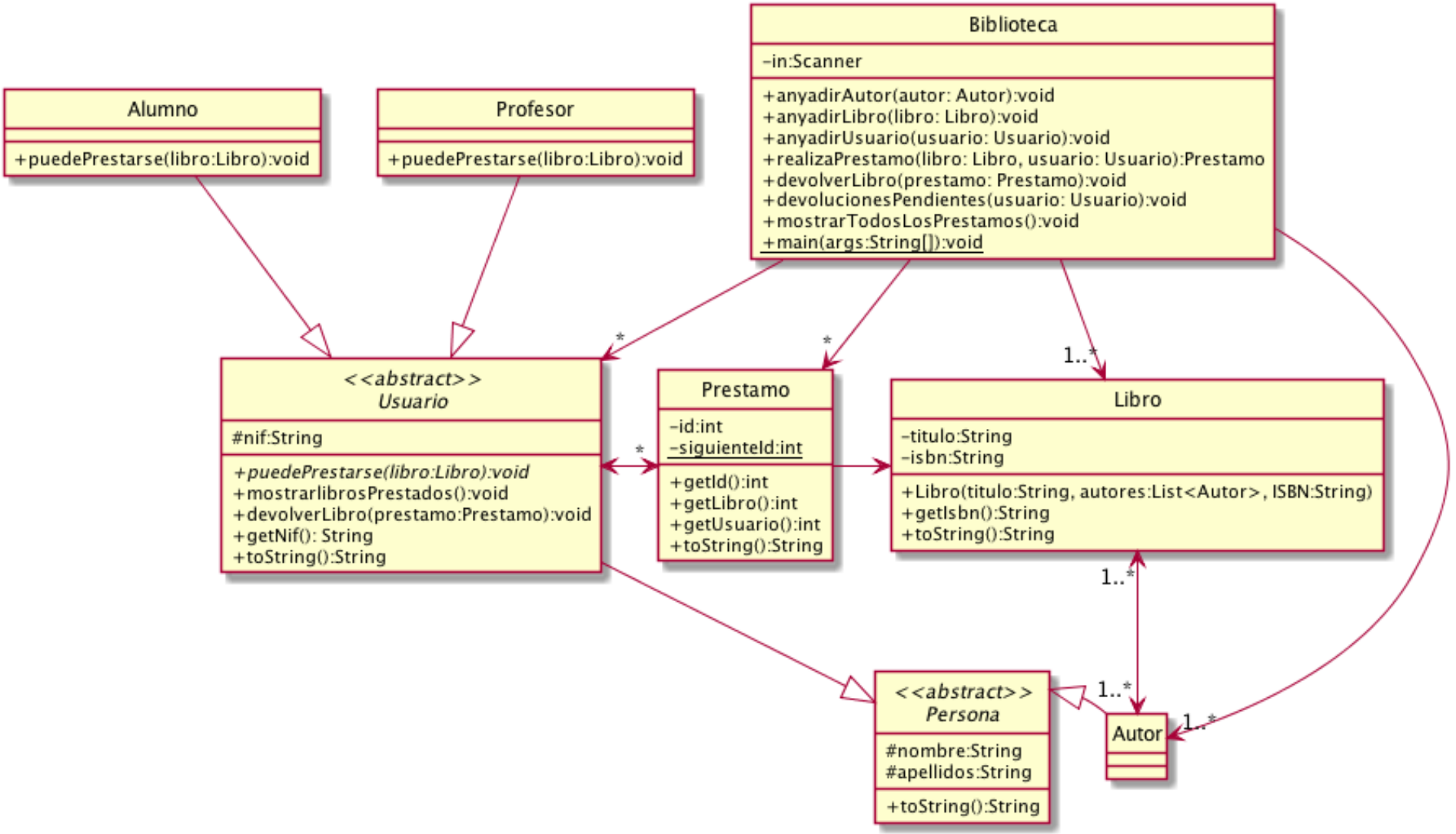
Este método intenta realizar el préstamo de un libro a un usuario. Se debe crear un nuevo objeto `Prestamo` y gestionarlo tal como se indica al inicio del enunciado del examen. **La resolución de este problema exige que implementéis el constructor de la clase `Prestamo`.**

El método muestra un mensaje por pantalla informando de si el préstamo ha podido o no realizarse. No debe suponerse que el libro que se pasa como argumento esté en la biblioteca.

```
// En clase Biblioteca
    public void realizarPrestamo(Libro libro, Usuario usuario) {
```

```
        if (!this.libros.containsKey(libro.getISBN())) {
            System.out.println("El libro cuyo ISBN es "
                    + libro.getISBN() + " no está en la bibloteca");
            return;
        }
        try {
            /* Comprobar si puede prestarse o no. Si no es así,
               lanza una excepción
             */
            usuario.puedePrestarse(libro);
            // Quitar el libro de la biblioteca
            this.libros.remove(libro.getISBN());
            // Crear el prestamo
            Prestamo prestamo = new Prestamo(libro, usuario);
            // Añadir el prestamo al contenedor de prestamos del usuario
            usuario.getPrestamos().add(prestamo);
            // Añadir el prestamo al contenedor de prestamos de la biblioteca
            prestamos.put(prestamo.getId(), prestamo);
            System.out.println("Se ha prestado el libro:\n\t" + libro
                    + "\n\tal usuario: " + usuario);
        } catch (PrestamoException e) {
            System.out.println("\t(No se puede realizar el prestamo de el libro: "
                    + libro + " al usuario: " + usuario + "). Detalles: "
                    + e.getMessage());
        }
    }

// En la clase Prestamo el constructor será como sigue
    public Prestamo(Libro libro, Usuario usuario) {
        this.id = ++Prestamo.siguienteId;
        this.libro = libro;
        this.usuario = usuario;
    }
```


Biblioteca
-in:Scanner
+anyadirAutor(autor: Autor):void
+anyadirLibro(libro: Libro):void
+anyadirUsuario(usuario: Usuario):void
+realizaPrestamo(libro: Libro, usuario: Usuario):Prestamo
+devolverLibro(prestamo: Prestamo):void
+devolucionesPendientes(usuario: Usuario):void
+mostrarTodosLosPrestamos():void
+main(args:String[]):void
Alumno
+puedePrestarse(libro:Libro):void
Profesor
+puedePrestarse(libro:Libro):void
<<abstract>>
Usuario
#nif:String
+puedePrestarse(libro:Libro):void
+mostrarlibrosPrestados():void
+devolverLibro(prestamo:Prestamo):void
+getNif(): String
+toString():String
Prestamo
-id:int
-siguienteId:int
+getId():int
+getLibro():int
+getUsuario():int
+toString():String
Libro
-titulo:String
-isbn:String
+Libro(titulo:String, autores:List<Autor>, ISBN:String)
+getIsbn():String
+toString():String
<<abstract>>
Persona
#nombre:String
#apellidos:String
+toString():String
Autor
*
*
*
*
1..*
1..*
1..*
1..*