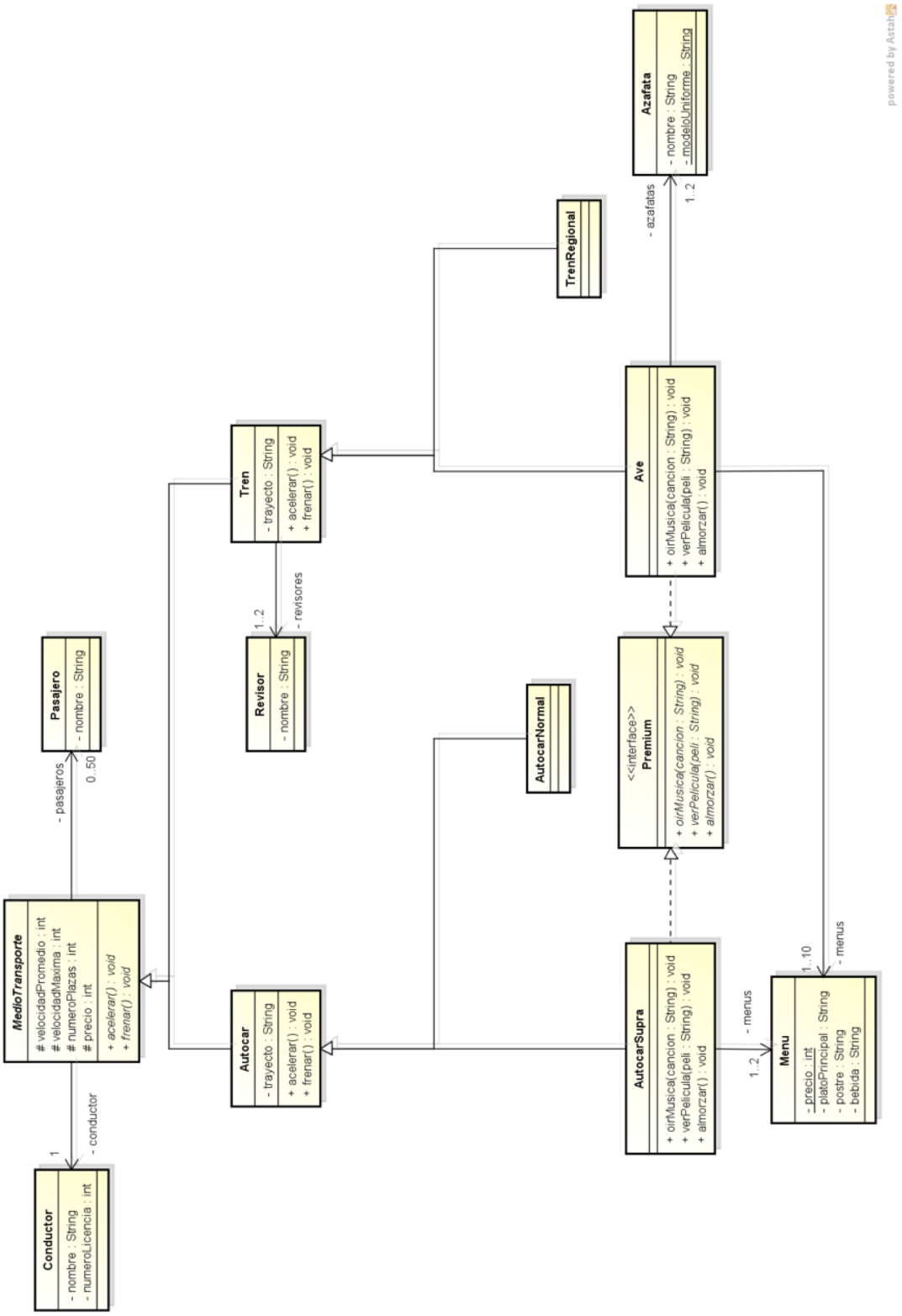

Figura 1: Diagrama de Clases del Diseño para el Test II.

Instrucciones generales

- La duración del examen es de 3 horas y 30 minutos.
- No podrá usarse ningún material o herramienta de apoyo (libros, apuntes, calculadora, etc.).
- Los móviles permanecerán apagados durante todo el examen.
- El DNI o carné UVA del alumno estará visible sobre la mesa durante toda la duración del examen.
- Sobre la mesa, además del DNI y del carné UVA, solo estarán las hojas de examen y el bolígrafo.
- Las respuestas deben escribirse con letra clara y con bolígrafo azul o negro.
- Se recomienda leer todas las preguntas antes de empezar el examen, así como comenzar contestando aquellas que se consideren más sencillas. También se sugiere tener en cuenta la puntuación que otorga cada pregunta a la hora de decidir en qué orden realizarlas.
- En el enunciado de cada pregunta se indica la puntuación máxima obtenible. En las preguntas en las que hay cuestiones de verdadero o falso, la puntuación total de la pregunta se divide a partes iguales entre todas las preguntas propuestas, debiendo tenerse además en cuenta que **una pregunta contestada de forma errónea resta lo mismo que suma una pregunta contestada de forma correcta** y que las preguntas sin contestar no recibirán ninguna puntuación, ni positiva ni negativa.
- La puntuación máxima obtenible en todo el examen es de 10 puntos. Esta puntuación se escalará para suponer un 30 % de la nota de la asignatura, es decir, que 10 puntos en este examen equivalen a 3 puntos en la nota final de la asignatura. Por otra parte, es necesario obtener 5 puntos en este examen para poder aprobar la asignatura.
- Se pueden utilizar tantas hojas de examen como resulte necesario, tanto de borrador, como para proporcionar las respuestas definitivas.
- Se entregarán todas las hojas recibidas con el enunciado del examen, además de todas las hojas de examen que se utilicen.
- No olvides poner el nombre, apellidos y DNI en la portada de este examen, en la tabla destinada a tal efecto, así como en todas las hojas de examen que utilices, incluidas las usadas como borrador, las cuales se identificarán tachando su contenido.

Test II (1,25 puntos)

Determina si las siguientes afirmaciones propuestas sobre el Diagrama de Clases mostrado en la Figura 1 son verdaderas o falsas. Nótese que por su grado de detalle, el Diagrama de Clases mostrado pretender ser un Diagrama de Clases del Diseño realizado como parte de la Actividad de Diseño.

1. La clase `MedioTransporte` se declararía en Java de la siguiente manera:

```
public class MedioTransporte extends Tren, Autocar {
//Atributos correspondientes
//Métodos correspondientes
}
```

2. La clase `Pasajero` se declararía en Java de la siguiente manera:

```
public class Pasajero implements MedioTransporte {
//Atributos correspondientes
}
```

3. La clase `TrenRegional` no tendría ningún atributo.
4. Como la clase `MedioTransporte` es abstracta, la clase `Tren` también debería haberse definido como una clase abstracta por la propiedad de herencia.
5. La clase `MedioTransporte` tiene un atributo que se llama `pasajeros` y que sería razonable que se implementara mediante alguna clase de colección de Java.
6. La clase `MedioTransporte` tiene 6 atributos.
7. La clase `Autocar` hereda la implementación del método `frenar()` de la clase `MedioTransporte`.
8. La operación `acelerar()` de la clase `MedioTransporte` es un ejemplo de operación polimórfica.
9. La operación `almorzar()` definida en la interfaz `Premium` es abstracta pero no es una operación polimórfica.
10. En nuestro sistema no podrá haber medios de transporte (es decir, instancias de la clase `MedioTransporte`) que no sean trenes o autocares (es decir, instancias de las respectivas clases).
11. El atributo `numeroPlazas` de la clase `MedioTransporte` es un atributo de clase.
12. Podría utilizarse el constructor de la clase `MedioTransporte` para crear tantas instancias de dicha clase en nuestro sistema como resultara necesario.
13. Se puede afirmar que existe visibilidad de atributo desde la clase `MedioTransporte` a la clase `Conductor`.
14. En la clase `MedioTransporte` el método `acelerar()` se debería especificar en Java de la siguiente manera:

    ```
    public abstract void acelerar();
    ```

15. En la clase `Tren` el método `frenar()` se debería especificar de la siguiente manera:

    ```
    public abstract void frenar();
    ```

16. La clase `Autocar` tiene un atributo que se llama `conductor` de tipo `Conductor`.
17. La clase `AutocarSupra` se declararía en Java de la siguiente manera:

    ```
    public class AutocarSupra extends Autocar implements Premium {
    //Métodos correspondientes
    }
    ```

18. Todos los tipos de `Autocar` tienen una o dos `Azafatas`.
19. Cada objeto de clase `Azafata` tiene un `modeloUniforme` diferente.
20. Todos los `AutocaresSupra` tienen el mismo `postre` en el `menú`.

Proyecto de ingeniería de software

Las siguientes preguntas parten del planteamiento de un pequeño proyecto software, de dificultad semejante al realizado en la asignatura. Lee con detenimiento la descripción del proyecto, y posteriormente contesta a las preguntas planteadas, produciendo artefactos semejantes a los que has entregado a lo largo de las prácticas de la asignatura.

Descripción del proyecto

Una biblioteca rural ha decidido crear un sistema informático para gestionar los préstamos de libros. Así, la biblioteca quiere catalogar todos sus libros, inscribir a sus socios y llevar un registro de los préstamos y devoluciones. Para realizar este proyecto, la biblioteca ha contratado a una empresa de software que ha decidido contratarte para llevar a cabo el software de gestión. A continuación se describen algunos requisitos sobre dicho software:

- La biblioteca mantiene un catálogo de libros.
- Cada libro tiene un ISBN único, un título y un autor. Un libro puede estar disponible o prestado. No hay ningún libro repetido y no se contempla que pueda haberlos en el futuro.
- El sistema reconocerá tres tipos de usuarios: administrador, bibliotecario y socio. Todos estarán identificados mediante un login y una clave. Además, de cada usuario se guardará su nombre, apellidos y DNI.
- Habrá un único administrador que se encargará de dar de alta a los bibliotecarios.
- Habrá al menos dos bibliotecarios en la biblioteca. Los bibliotecarios se encargarán de registrar a los socios de la biblioteca (a su petición), registrar los libros en el catálogo y de gestionar los préstamos de libros (como se indica más adelante).
- Los socios pueden hacer búsquedas en el catálogo para encontrar libros y consultar su disponibilidad.
- Un socio puede ir a la biblioteca y solicitar el préstamo de un libro disponible a un bibliotecario. El bibliotecario entregará el libro al socio y registrará el préstamo en la biblioteca con la fecha de inicio (la del día actual). Los socios pueden tener prestados un máximo de 3 libros.
- Para devolver un libro, el socio va a la biblioteca y lo entrega al bibliotecario. El bibliotecario se encarga de colocar el libro en su sitio y de eliminar el registro del préstamo.

Actividades de análisis

Pregunta 1 (1 punto) Prepara un modelo de dominio de este sistema a partir de las especificaciones proporcionadas. Ten en cuenta que en un modelo de dominio no suele haber operaciones y el número de atributos es más bien reducido. Tienes que indicar las multiplicidades en las asociaciones de acuerdo con las especificaciones y etiquetar las asociaciones. En tu modelo no inventes ni añadas nada que no esté indicado.

Pregunta 2 (1 punto) Elabora los casos de uso detallados para `RegistrarLibro` **(0,4 puntos)** y `PrestarLibro` **(0,6 puntos)**. En el primer caso de uso el bibliotecario registra un libro en el catálogo de la biblioteca. En el segundo caso de uso el bibliotecario comprueba si el socio está registrado y si ha superado el límite de préstamos, luego busca en el sistema el libro solicitado por el socio (le habrá facilitado el título), consulta la disponibilidad del mismo y registra el préstamo en el sistema.

En cada caso de uso indica el **nombre** (`RegistrarLibro` o `PrestarLibro`), las **precondiciones**, los **pasos de la secuencia normal**, las **postcondiciones** y los **pasos de las secuencias alternativas** más relevantes. Las precondiciones y postcondiciones deben referirse a los conceptos y asociaciones presentes en el modelo de dominio, al modo de los contratos de las operaciones del sistema (por ejemplo, "existe alguna instancia de Y", "se ha creado una instancia de X y se ha asociado con la instancia de Y", "se ha modificado el atributo Z de Y", etc.).

Actividades de diseño mediante ingeniería inversa

El ingeniero de software consigue un código en Java que implementa parte de la funcionalidad que quiere para su aplicación, pero no es exactamente igual al que has modelado en la pregunta 2. Este código se presenta en el Apéndice. Para ver si la biblioteca puede aprovecharlo hará falta hacer ingeniería inversa para reconstruir el diseño. Al hacerlo, no inventes ningún atributo, método, asociación o tipo que no esté descrito.

Pregunta 3 (1 punto) Prepara el diagrama de clases de diseño correspondiente al código entregado. Contempla las siguientes restricciones:

- Trata los atributos de clases e interfaces de la API de Java como si fuesen de tipos primitivos.
- No hace falta que muestres los tipos de los atributos.
- Trata convenientemente las colecciones en el diagrama.
- Muestra las asociaciones de atributo, incluye flechas de navegabilidad y los roles.
- Muestra los nombres de las operaciones de las clases, pero no es necesario que pongas la signatura completa.
- Muestra la visibilidad tanto para atributos como para operaciones.
- Muestra las dependencias entre clases en el diagrama.

Pregunta 4 (1 punto) Elabora el diagrama de secuencia que realiza con éxito el caso de uso `PrestarLibro` basado en el código del Apéndice. Este diagrama de secuencia debe cubrir todas las interacciones necesarias (identificar el bibliotecario, seleccionar un socio, buscar un libro y prestar el libro). Ten en cuenta las siguientes recomendaciones:

- Comienza el diagrama a partir del código de la clase `ControladorSesionBibliotecario`.
- Nombra los objetos implicados según el código.

- Indica el valor de retorno de los mensajes (puedes omitir el tipo).
- Pinta las colecciones de objetos con dos cajas superpuestas.
- Ajusta bien el diagrama al espacio, es recomendable dibujarlo en formato apaisado.

Actividades de diseño

Tras analizar el código del Apéndice, el ingeniero de software cree que puede reutilizarlo, pero debería modificarlo para permitir a cada socio tener un máximo de tres libros prestados. Por otra parte, la biblioteca le informa de un cambio en los requisitos, siendo necesario ahora que pueda haber varias copias de un mismo libro.

Pregunta 5 (2 puntos) Tienes que realizar un rediseño que incorpore las dos mejoras indicadas: (1) permitir que haya varias copias de un mismo libro y (2) permitir a cada socio tener un máximo de tres libros prestados. Para ello se pide:

a) Un diagrama de secuencia que ilustre el registro de un libro y la creación de varias copias del mismo. Aquí basta con que te centres en el registro del libro y la creación de las copias, no hace falta indicar otros aspectos como la identificación del bibliotecario. **(0,5 puntos)**

b) Un diagrama de secuencia para el préstamo de libros en el que se ilustre la restricción del número de libros prestados. De nuevo puedes centrarte sólo en los aspectos clave de este rediseño (puedes apoyarte en el diagrama de la Pregunta 4). **(0,5 puntos)**

c) Un diagrama de clases de diseño en el que se enfaticen los cambios con respecto al diagrama creado en la Pregunta 3. Por ejemplo, es importante mostrar las clases nuevas creadas (si hiciera falta crearlas) e indicar las relaciones con el resto de clases. También podrían crearse nuevas operaciones o realizar otras modificaciones. Este diagrama también debe ser consistente con los diagramas preparados en los apartados a) y b) de esta pregunta. **(0,5 puntos)**

d) Una explicación razonada y convincente de cómo tu solución soporta las mejoras pedidas. Aquí se espera que te apoyes en los diagramas anteriores para ilustrar tu solución y que justifiques tus decisiones de diseño con patrones software. **(0,5 puntos)**

Actividades de implementación

Pregunta 6 (1,5 puntos) Después de completar el diseño en la pregunta anterior, se prosigue con la implementación de los casos de uso de registro de libros y creación de copias, así como del préstamo de libros (de acuerdo con el nuevo diseño de la Pregunta 5). Se completarán los escenarios de éxito, así como los escenarios de fallo en que el socio no puede reservar un nuevo libro por tener ya 3 libros prestados y en el que no haya copias disponibles (utiliza en ambos casos una excepción de tipo `LibroException` – definida en el Apéndice).

A partir del código proporcionado, prepara el código Java de las clases nuevas (si hay alguna), los métodos nuevos, y los métodos que hayas tenido que modificar para

realizar los casos de uso indicados. Indica también si has tenido que incluir algún atributo nuevo en alguna clase, especificando el tipo en tal caso. Intenta que el código esté bien organizado y sea legible, aunque no se tendrán en cuenta errores de sintaxis como la falta de algún punto y coma, o algún paréntesis o llave de más o menos.

Apéndice: código Java del sistema de biblioteca

```
public class Bibliotecario {

  private String password;
  private String login;

  public Bibliotecario(String log, String pass) {
    login=log;
    password=pass;
  }

  public  String getLogin() {
    return login;
  }

  public  String getPwd() {
    return password;
  }
}
```

```
import java.util.HashMap;
import java.util.Map;

public class GestorUsuarios {

  private Map<String,Socio> mapaSocios;
  private Bibliotecario biblio;

  public GestorUsuarios() {
    mapaSocios = new HashMap<String,Socio>();
    biblio = new Bibliotecario("biblio", "1234");
  }

  public void crearSocio(String id) {
    Socio soc = new Socio(id);
    mapaSocios.put(id, soc);
  }

  public Socio seleccionarSocio(String id) {
    return mapaSocios.get(id);
  }

  public Bibliotecario idBiblio(String l, String p) {
    if (biblio != null && p.equals(biblio.getPwd()))
      return biblio;
    else
      return null;
  }
}
```

```
public class Socio {
  private String idSocio;

  public Socio(String id) {
    idSocio=id;
  }

  public  String getId() {
    return idSocio;
  }
}
```

```
public class Libro {

  private String isbn;
  private String titulo;
  private String autor;
  private Socio prestatario;

  public Libro(String i, String t, String a) {
    isbn = i;
    titulo = t;
    autor = a;
    prestatario = null;
  }

  public String getIsbn() {
    return isbn;
  }

  public  boolean estaDisponible() {
    return prestatario == null;
  }

  public boolean contieneCadena(String c) {
    return titulo.contains(c) || autor.contains(c);
  }

  public void prestar(Socio soc) {
    prestatario=soc;
  }

  public void devolver() {
    prestatario = null;
  }
}
```

```
import java.util.ArrayList;
import java.util.List;

public class ControladorSesionBibliotecario {

  private GestorUsuarios gus;
  private GestorLibros gli;
  private Bibliotecario bib;
  private Socio soc;

  public ControladorSesionBibliotecario() {
    gus = new GestorUsuarios();
    gli = new GestorLibros();
  }

  public void identificarBiblio(String l, String p){
    bib = gus.idBiblio(l, p);
  }

  public void seleccionarSocio(String id) {
    if (bib != null)
      soc = gus.seleccionarSocio(id);
  }

  public List<String> buscarLibro(String cad) {
    if (bib != null) {
      List<String> llibs = new ArrayList<String>();
      for (Libro lib : gli.buscarLibro(cad))
        llibs.add(lib.getIsbn());
      return llibs;
    }
    else
      return null;
  }

  public void prestarLibroSocio(String isbn)
      throws LibroException {
    if (bib != null && soc != null)
      gli.prestarLibro(isbn, soc);
  }

  public void devolverLibroSocio(String isbn)
      throws LibroException {
    if (bib != null && soc != null)
      gli.devolverLibro(isbn);
  }
}
```

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class GestorLibros {

  private Map<String,Libro> catalogo;

  public GestorLibros() {
    catalogo = new HashMap<String,Libro>();
  }

  public void crearLibro(String i, String t, String a)
      throws LibroException {
    if (catalogo.get(i) != null)
        throw new LibroException(isbn+" ya existe");
    Libro lib = new Libro(i,t,a);
    catalogo.put(isbn, lib);
  }

  public List<Libro> buscarLibro(String cadena) {
    List<Libro> listal = new ArrayList<Libro>();
    for (Libro lib : catalogo.values()) {
      if (lib.contieneCadena(cadena))
        listal.add(lib);
    }
    return listal;
  }

  public void prestarLibro(String isbn, Socio soc)
      throws LibroException {
    Libro lib = catalogo.get(isbn);
    if (lib == null)
     throw new LibroException(isbn+" no existe");
    if (lib.estaDisponible())
      lib.prestar(soc);
    else
     throw new LibroException(isbn+" no disponible");
  }

  public void devolverLibro(String isbn)
      throws LibroException {
    Libro lib = catalogo.get(isbn);
    if (lib == null)
     throw new LibroException(isbn+" no existe");
    else
     lib.devolver();
  }
}
```

```
public class LibroException extends Exception {
  public LibroException(String causa) {
    super(causa);
  }
}
```