**MEMORIA ESCRITA**

**Descripción General**

Nuestro objetivo es implementar un sistema de gestión para una biblioteca basado en el paradigma de POO y en la forma de almacenar datos de la Biblioteca Efe Gómez de la Universidad Nacional. Cabe aclarar que no se sigue estrictamente toda la metodología tratada allí, dado que para poder implementar lo pedido con el trabajo se adecuaron ciertos puntos que no concuerdan como tal.

En primera instancia se analizan las clases que podrían representar de la manera mas optima el problema de gestión y organización de los datos en una biblioteca, así que basándonos en la biblioteca Efe Gómez se llegó a la conclusión de representar mediante 11 clases ( en cuanto a la maqueta lógica) el problema planteado. Estas clases se dividen en tres paquetes principales que son: paquete de Obras en donde se encuentran los módulos Estantería, Folleto, Libro, Revista y publicación; el paquete de Personas donde se encuentran los módulos Autor, Persona, Estudiante Profesor, Externo y Usuario; además un paquete de Préstamo donde se encuentra un modulo con el mismo nombre. Todo el conjunto de clases que conforman este sistema buscan describir y simular de la mejor manera el sistema bibliotecario a través de operaciones CRUD y demás implementaciones que agilizan la gestión del sistema de datos

El Sistema se desarrollo en Java dentro de 3 paquetes:

* Uno llamado baseDatos, donde se almacenan los datos (objetos) creados y se diseñan todas las clases necesarias para serializar y deserializar objetos
* Otro llamado gestorAplicacion, donde se programó toda la capa lógica; se definen todas las clases a almacenar con su respectivos atributos y métodos
* Un último paquete llamado uiMain, donde se implementa toda la interfaz para que el usuario haga las operaciones que necesite mediante el despliegue de menús en pantalla, elección de opciones y recolección de datos (o números o cadenas de caracteres)

Adicionalmente para garantizar la persistencia de la información al iniciar la interfaz del usuario se cargan todos los objetos anteriormente creados y al finalizar el programa y digitar la opción de guardar cambios, serializamos todos los datos creados y los almacenamos para un uso posterior

**Descripción del diseño estático**

Los diagramas se encuentran en el documento adjunto con el nombre “diagramas.pdf”

**Descripción de la implementación de características de POO**

*Clases Abstractas:*

Hay dos clases abstractas definidas:

Publicación definida en el modulo con el mismo nombre, en el paquete gestorAplicacion.obras la cual hereda a tres clases publicas que representan los tipos de materiales Revista, Libro y Folleto. Como solo se necesitan instanciar estas tres se implementa la clase padre Publicacion como abstracta para que solo herede atributos y establezca métodos abstractos necesarios en cada clase hija. Además cuenta con métodos normales de clase y de instancia no abstractos.

Persona definida en el módulo con el mismo nombre, en el paquete de gestorApliacion.personas donde esta clase es la principal pues es una clase abstracta la cual hereda a todas las demás clases del paquete, ya que establece los atributos comunes tanto para la Clase Autor como para las clases EstudiantePersona y Externo que hacen alusión a los posibles usuarios del sistema de información. Esta define métodos de clase y un metodo abstracto que es implementado por las tres clases hijas (metodo infoPersonal el cual muestra información del objeto con sus atributos detalladamente).

*Métodos Abstractos:*

Los métodos mostrarInfo() y mostrarUbicacion() declarados en la clase abstracta Publicacion paquete Obras, que son heredados e implementados por las clases hijas Folleto, Revista y Libro, muestran información de la publicación con sus atributos y la ubicación (estantería) respectivamente.

El método infoPersonal() declarado en la clase abstracta Persona paquete Personas, con el fin de ser implementado por las tres clases hijas Autor, EstudianteProfesor y Externo, en las cuales se implementa para retornar un string con los datos/atributos principales descriptivos para cada una de las posibles instancias de cada clase. Cabe resaltar que para la clase Autor se utilizan solo tres atributos que son suficientes para identificar al autor de cada publicación, diferente a los usuarios que si amerita que contengan mas información.

El metodo prestar() que se define en la interface Usuario del paquete persona, para ser implementado en las clases que implementan esta interface que son las clases alusivas a los usuarios del sistema es decir EstudiantesPersona y Externo. Este metodo implementa la funcionalidad de prestar un libro, pues al ser heredado por las clases de los usuarios se llama desde las instancias de estos a través de tres parámetros que son la publicación que se quiere prestar, el identificador del préstamo y la fecha cuando se presta dicha publicación

*Interfaces:*

Solo se implementa un interface que es el interface llamado Usuario en el modulo del mismo nombre del paquete gestorAplicacion.personas, introducido como un interface que define los métodos propios que los usuarios del sistema tendrán a la hora de interactuar y participar con las funcionalidades, como por ejemplo el metodo abstracto prestar()

*Herencia*

La implementación de la herencia se implementa en dos de los tres paquetes de la capa lógica, respectivamente en el paquete gestorAplicacion.personas se utiliza en dos casos principales: la herencia que tienen las clases EstudianteProfesores, Autor y Externo con la clase padre Persona, y además al mismo tiempo el interface Usuario hereda métodos abstractos para las clases EstudianteProfesores y Externo. El segundo caso es en el paquete gestorAplicacion.obras donde la super clase abstracta Publicaciones definide la mayoría de atributos, métodos de clase, de instancia y abstractos para las clases hijas Folleto, Revista y Libro

*Ligadura dinámica*

*Constantes*

Se define un atributo privado y constante en la clase Libro del paquete obras denominado Autor, el cual es de tipo Autor y se le asigna el modificador final con el fin de no ser alterado puesto que un libro tiene un autor asignado permanentemente y este hecho no cambia, lo cual se ve reflejado en la implementación. No se inicializa directamente, se hace al asignarle el valor con el constructor de la clase Autor

Se define universidad como atributo privado y constante en la clase EstudianteProfesor del paquete personas, este se inicializa con el valor String “Universidad Nacional de Colombia” dado que dicha clase donde se encuentra el atributo hace referencia a los usuarios que hacen parte de la comunidad universitaria o involucrados directamente con la universidad, por lo que cada que se cree un objeto de esta clase tiene que pertenecer a esta universidad a diferencia de la otra clase Exterior donde pueden ser de otras universidades o no ser de alguna.

El atributo fechaActual de la clase Prestamo ubicada en el modulo del mismo nombre y en el paquete de también el mismo nombre, es definido constante y a su vez de clase pues se refiere al valor LocalDate o fecha del día actual en tiempo real, lo cual es importante a la hora de implementar los métodos y funcionalidades asociados a la clase préstamo que depende en gran medida de cálculos entre fechas teniendo como base el día actual.

*Encapsulamiento*

Public en cuanto al modificador public se utiliza para todas las clases utilizadas en la capa lógica dado que deben ser visibles para todas las demás clases independiente del paquete o ubicación que este, esto mas que todo por la clase préstamo y la clase Bibliotecaria que es donde se aloja el metodo Main principal y que deben acceder a estas clases. A su vez todos los métodos de todas las clases también son públicos por la misma razón. Un caso particular son los Enum que se encuentran en tres clases y que son públicos dado que estos se consideran una clase especial y son también accedidos por clases de otros paquetes.

Protected este modificador es utilizado para todos los atributos de la clase Publicacion con el fin de ser accedidos por sus clases hijas, lo mismo pasa con los atributos de la clase Persona.

Private Las listas definidas en cada clase que almacenan los objetos creados de dichas clases son privadas a excepción de la clase Persona que es protected. Todos los atributos se definen como privadosa excepción de los ya mencionados como protected, con el fin de utilizar los métodos get y set que son públicos de cada clase

Package solo se definen con el modificador de paquete los atributos de la clase estantería puesto que son accedidos en el desarrollo de las clases de su mismo paquete gestorAplicacion mientras que en los demás paquetes se utilizan los métodos get y set

*Sobrecarga de métodos*

La sobrecarga de metodos se presenta en la clase préstamo (modulo y paquete con el mismo nombre) con el metodo denominado determinar de la siguiente forma:

**public** **void** determinarFin(EstudianteProfesor usuario, Libro publicacion)

**public** **void** determinarFin(EstudianteProfesor usuario, Revista publicacion

**public** **void** determinarFin(Externo usuario, Libro publicacion)

**public** **void** determinarFin(Externo usuario, Revista publicacion)

Las primeras dos definiciones contemplan el caso cuando se ingresa una persona de la clase Estudiante profesor para un Libro y una Revista, las otras dos definiciones contemplan los casos en que se ingresa un objeto de la clase Externo para Libro y Revista respectivamente. Se sobrecarga de esta forma con el fin de tener en cuenta los dos casos de usuario, cuando el usuario es externo de la universidad o cuando hace parte de ella puesto que si se declara como tipo Persona no se puede especificar claramente las condiciones especiales que se dan en cada caso

*Sobrecarga de constructores*

La sobrecarga se evidencia en la clase Prestamo de la siguiente forma

**public** Prestamo(EstudianteProfesor usuario, Publicacion publicacion,**int** id,LocalDate inicio)

**public** Prestamo(Externo usuario, Publicacion publicacion,**int** id,LocalDate inicio)

este se implementa con el fin de diferenciar el tipo de usuario que va a crear el prestamo, sea externo o si hace parte de la universidad pues ambos casos tienen particularidades distintas

Se presenta en la clase Externo de la siguiente forma

**public** Externo(String nombre,**int** id,String correo,**int** tel,String direccion, LocalDate nacimiento, String paisOrigen)

**public** Externo(String nombre,**int** id,String correo,**int** tel,String direccion,LocalDate nacimiento, String paisOrigen, String universidad)

esto con el fin de a la hora de crear una persona externa es decir una instancia de la clase Externo se pueda ingresar opcionalmente la universidad a la que pertenece si lo esta, así como en la clase hermana se especifica que todos pertenecen a la Universidad Nacional

En la clase Persona se utiliza con el fin de ser utilizados específicamente el primero para la clase hija Autor que solo son necesarios dichos atributos y el constructor restante para la clases de usuarios que si necesitan la totalidad de atributos de la clase Persona que se heredan

Persona (**int** id,String nombre, LocalDate nacimiento, String paisOrigen)

Persona(String nombre,**int** id,String correo,**int** tel,String direccion,LocalDate nacimiento, String paisOrigen)

En la clase Autor se utiliza para a la hora de crear el objeto se especifique opcionalmente si el autor ya ha fallecido pues por defecto todas las Personas que se crean tienen el atributo vivo igual a true, que da a entender que aun viven

**public** Autor(**int** id, String nombre, LocalDate nacimiento, String paisOrigen)

**public** Autor(**int** id, String nombre, LocalDate nacimiento, String paisOrigen, **boolean** vivo)

*this*

Se utiliza principalmente en todas las clases para indicar el objeto actual de la clase que se esta instanciando y que se quiere añadir a la lista correspondiente donde están todos los objetos creados de su misma clase, por ejemplo:

**public** Autor(**int** id, String nombre, LocalDate nacimiento, String paisOrigen) {

**super**(id, nombre, nacimiento, paisOrigen);

*autores*.add(**this**);

}

También se utilizan en los métodos implementados que brindan información del objeto los cuales están definidos en cada clase, con el fin de especificar los atributos del objeto que se esta instanciando, por ejemplo en la clase EstudianteProfesor

**public** String infoPersonal() {

**return** "DATOS PERSONALES :"+"\n" +"Nombre: " + **this**.nombre +"\n"+ "Rol: "+**this**.rol+ "\n"+ "ID: "+ **this**.id

+ "\n"+"Universidad: " + **this**.universidad +"\n"+ "Correo: "+ **this**.correo +"\n"+ "Telefono: "+**this**.tel

+"\n"+"Direccion: "+ **this**.direccion +"\n"+"Pais de Origen: "+ **this**.paisOrigen +"\n"+ "Fecha de Nacimiento: "+ **this**.nacimiento;

}

En el metodo prestar() definido en las clases alusivas a los usuarios al crear el objeto de Prestamo dentro de este metodo se ingresa como parámetro el objeto que esta instanciando dicho metodo en ese momento de la clase correspondiente hija de Persona

Prestamo prestamo = **new** Prestamo(**this**, publicacion, id, inicio);

En el metodo determinarFin() de la clase Prestamo se utiliza para utilizar el metodo definido por el atributo de tipo LocalDate de la clase que esta instanciando al momento de llamar dicho objeto

**this**.fin= **this**.inicio.plusDays(1).atTime(LocalTime.of(12, 0));

*this()*

Se utiliza para llamar a constructores ya definidos anteriormente en dicha clase, se da el caso en las clases Autor, Persona y Externo, basados en lo ya explicado de la sobrecarga de constructores de cada una de estas clases, por ejemplo en la clase Autor

**public** Autor(**int** id, String nombre, LocalDate nacimiento, String paisOrigen) {

**super**(id, nombre, nacimiento, paisOrigen);

*autores*.add(**this**);

}

**public** Autor(**int** id, String nombre, LocalDate nacimiento, String paisOrigen, **boolean** vivo) { // Constructor si el Autor no esta vivo

**this**(id,nombre, nacimiento, paisOrigen);

**super**.vivo = vivo;

*autores*.add(**this**);

}

*Enumeración*

Se presentan los siguientes casos de enumeraciones en las clases Publicacion, Revista, Libro y EstudianteProfesor respectivamente. En la primera se establecen los dos estados posibles en los que puede estar una publicación, en la segunda los meses necesarios para la asginacion del atributo mes de la clase Revista, en la tercera los tipos de libros que hay en la biblioteca que presentan características especiales cada uno a la hora de prestar y en la cuarta que especifica los roles posibles que puede tomar el usuario de la clase EstudianteProfesor

**public** **static** **enum** Estados {***CIRCULACION***, ***PRESTADO***};

**static** **public** **enum** Meses { ***ENERO***, ***FEBRERO***,***MARZO***,***ABRIL***,***MAYO***,***JUNIO***,***JULIO***,***AGOSTO***,***SEPTIEMBRE***,***OCTUBRE***,***NOVIEMBRE***,***DICIEMBRE***};

**static** **public** **enum** tipoLibro {***COLECCION\_GENERAL***,***RESERVA***,***INVESTIGACION***,***SEMINARIO***,***TESIS***};

**public** **static** **enum** Rol {***ESTUDIANTE***, ***PROFESOR***,***OTRO***};

*Atributos de clase*

En cada clase se especifica una lista que contiene los objetos creados de dicha clase, esta es estática, por ejemplo en Persona

**protected** **static** ArrayList<Persona> *lista* = **new** ArrayList<>();

Como se pudo apreciar en el punto anterior las enumeraciones que se dan en las clases establecidas también proponen como estático a este atributo

**public** **static** **enum** Estados {***CIRCULACION***, ***PRESTADO***};

El caso particular se da en la clase préstamo donde se establece un atributo estático que hace referencia a la fecha actual en tiempo real, con el fin de poder ser accesada por todos los objetos de la clase y es la misma para todos ellos por obvias razones

**private** **final** **static** LocalDate fechaActual= LocalDate.now();

*Métodos de Clase*

En las clases principales es decir Persona, Publicacion y Estanteria se especifican dos métodos estáticos en común para las tres, los cuales son mostrarRegistros(),eliminarEstanteria, eliminarPublicacion() y eliminarPersona(), estos dos tipos de métodos se encargan de las funcionalidades CRUD de mostrar los objetos creados y eliminarlos respectivamente en cada una de sus clases, por ejemplo

**public** **static** **void** eliminarPersona(Persona p) { // elimina un registro de Persona

*lista*.remove(*lista*.indexOf(p));

}

**public** **static** String mostrarRegistros() {

String c="Personas creadas: "+"\n";

**for** (**int** i = 0; i < *lista*.size(); i++) {

**if**(*lista*.get(i) **instanceof** Autor ) {c = c + i+"."+"Autor "+ *lista*.get(i).nombre + " ID "+ *lista*.get(i).id+ "\n"; }

**else** **if**(*lista*.get(i) **instanceof** EstudianteProfesor) {c = c + i+"."+"Usuario de la Universidad "+ *lista*.get(i).nombre + " ID "+ *lista*.get(i).id+ "\n";}

**else** **if**(*lista*.get(i) **instanceof** Externo) {c = c + i+"."+"Usuario externo de la Universidad "+ *lista*.get(i).nombre + " ID "+ *lista*.get(i).id+ "\n";}

}

**return** c;

}

**Descripción de las funcionalidades implementadas**

Las funcionalidades que implementamos son:

1. Añadir Registros
2. Mostrar registros
3. Eliminar registros
4. Realizar Préstamos: Consiste en enlazar un préstamo a un usuario. Intervienen:
   1. EstudianteProfesor y Externo: implementa la interfaz usuario y heredan de la clase Persona
   2. Prestamo: Es lo que se va a registrar;
   3. Publicación: Revista o Libro
   4. Autor: teniendo en cuenta que la publicación debe estar asociada a uno
   5. Múltiples métodos para llegar al método final
5. Realizar una Renovación: Consiste en corroborar si existe el préstamo ingresado y ampliar el plazo de la fecha de vencimiento de este. Intervienen:
   1. EstudianteProfesor y Externo: implementa la interfaz usuario y heredan de la clase Persona. Hay una instancia asociada al préstamo que debe estar registrado
   2. Prestamo: A este se le cambia el atributo fin (fecha de vencimiento)
   3. Publicación: Revista o Libro
   4. Autor: teniendo en cuenta que la publicación debe estar asociada a uno
   5. Múltiples métodos para llegar al método final

Manual de Usuario:

La interfaz por medio de menús es bastante clara e intuitiva. La idea es dejarse llevar por las opciones que se ofrecen en pantalla e ir usando sus funcionalidades.

Las siguientes son las normas que hay que cumplir para garantizar su funcionamiento adecuado:

1. Tenga en cuenta que cuando en los menús de opciones el usuario digite un número incorrecto , se volverá a desplegar el menú de opciones, esto se repetirá hasta que se ingrese la opción válida. También hay unos pocos casos en los que directamente se devolverá a un menú anterior, pero en estos no hay mayor complejidad, simplemente no se completa la operación.
2. Hay que ser precisos con el tipo de dato que se ingrese , puesto que si se ingresa uno incorrecto arrojará error y tocará volver a correr el programa. Afortunadamente solo se ingresan 2 tipos de datos (o numéricos u listas de caracteres) y , para general mayor claridad, antes de solicitar ingresar algún dato la interfaz mostrará que dato se está ingresando y entre paréntesis que tipo de dato se debe ingresar.

Las convenciones para estos tipos de datos son:

* 1. (N) : Simboliza que se debe ingresar un dato únicamente numérico
  2. (S) : Simboliza que se debe ingresar un dato alfanumérico

1. Para que se lea la información correspondiente, el usuario debe presionar la tecla “enter” justo después de terminar de escribir el dato que se quiere almacenar. Sea muy cauteloso a la hora de ingresar una fecha válida en el formato correspondiente, puesto que también se lanzará un error que implicará un reinicio del sistema si no se ingresa adecuadamente
2. Otro aspecto importante es que para poder registrar una publicación (libro, revista o folleto) debe tener un número de la estantería en la que se debe ingresar (la cual debe estar ya registrada) .Si este número no corresponde a un objeto de su correspondiente tipo ya creado, el programa fallará y tendrá que ejecutarse nuevamente
   1. Si desea ingresar la publicación con una estantería desconocida, ingrese el número 0 cuando se le pida el número de estantería
3. El autor se creará al momento de ingresar un libro si no tiene el código de un autor ya creado
4. Para eliminar registros se necesita tener el correspondiente código diferenciador del objeto. Para hallarlo simplemente ir a mostrar registros de los menús de borrado o entrar directamente a la funcionalidad de mostrar registros. Dicho código diferenciador estará precedido de “(CP)”
5. Para realizar un préstamo asegúrese de que la publicación y el usuario están registrados (esto lo puede hacer utilizando la implementación de mostrar registros). Si no se asegura de esto e ingresa un ID de un usuario no almacenado, se genera un error y se deberá reiniciar el sistema. Además, el usuario le debe asignar manualmente el ID al préstamo (este no debe estar registrado en ninguna otra parte antes) y el día en el que se quiere iniciar el préstamo
6. Para realizar una renovación asegúrese de que el usuario y el préstamo usuario están registrados esto lo puede hacer utilizando la implementación de mostrar registros). Si no se asegura de esto e ingresa un ID de un usuario o un código de préstamo no almacenado, se genera un error y se deberá reiniciar el sistema.
7. Para almacenar en archivos y hacer que perduren los objetos creados debe elegir la opción 6 del menú principal (“Guardar datos y salir del sistema”)