Programación Orientada a Objetos

Trabajo Práctico Especial

Juan Bautista Artia
artianjuanb@gmail.com

Felipe Serrano
felipefsserrano@gmail.com

Indice

1.1 Introducción y objetivo

Este trabajo práctico especial consiste en desarrollar un sistema de administración de libros y colecciones para una biblioteca. Esta biblioteca puede estar formada por diferentes estantes, estanterías, salas y pabellones. Cada uno de estos conjuntos puede contener otro conjunto dentro de él. A su vez también se nos requirió realizar un sistema de búsquedas a partir de las distintas características de los libros. Y por último, el sistema deberá poder saber la cantidad de elementos que posee, listar las palabras claves que están vinculadas a un libro o un grupo, imprimir los elementos según un formato establecido y realizar una copia de un grupo con aquellos elementos que cumplan un criterio provisto.

2. Desarrollo

A partir de los conceptos y técnicas basadas en la programación orientada a objetos y utilizando java como lenguaje de programación se resolvió el sistema pedido.

2.1 Diagrama de clases

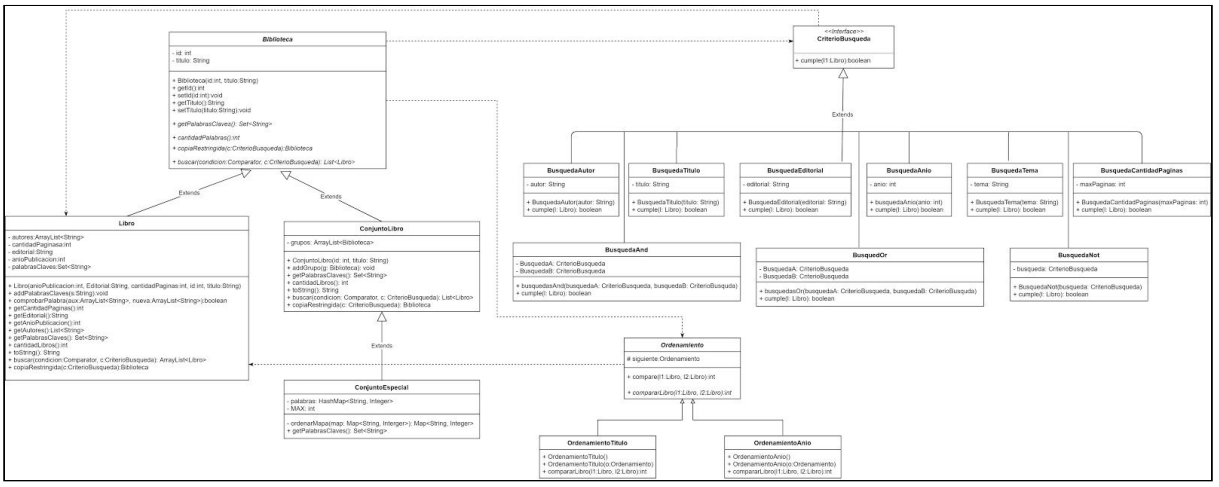

Diagrama de clases del programa completo.(Para mayor detalle ingresar al link clickeando la imagen)

2.2 Consideraciones generales

- Se utilizó la clase *Set* para representar las palabras claves, evitando así los repetidos.
- Estantes, estanterías, salas y pabellones se modelaron con la clase *ConjuntoLibro,* ya que todos comparten las mismas funcionalidades y atributos.
- Palabras claves solo es atributo de *Libro* ya que *ConjuntoLibro* posee sólo las palabras de los libros que contiene.

2.3 Búsquedas

El sistema desarrollado debe permitir las búsquedas de libros según un criterio. Para esto se creó la interfaz *CriterioBusqueda*. Esta interfaz contiene el método *cumple*, el cual a partir de un libro devolvera un *boolean* si el libro cumple con el criterio. Este método deberá ser sobreescrito por cada una de las clases de búsqueda por lo que estas deberán implementar a *CriterioBusqueda*.

Además, estas búsquedas no deben ser *case sensitive*, también se devolverán los libros que cumplan parcialmente con la condición dada y esta lista de libros resultante, se ordenará siguiendo un criterio dado.

2.3.1 Búsqueda autor

Esta búsqueda tiene un atributo *String* con el nombre del autor buscado. Se obtiene la lista de autores y se recorre, verificando si algunos de estos nombre coincida entera o parcialmente con el nombre buscado. Si la búsqueda es exitosa, se devolverá true.

2.3.2 Búsqueda páginas

Esta búsqueda tiene un atributo entero con la cantidad de máxima de páginas que puede tener un libro. Devuelve un *boolean* a partir de la comparación de páginas entre un libro provisto y el máximo de páginas pedido. Si este libro contiene un número menor de páginas al pedido, esta búsqueda devolverá *true*.

2.3.3 Búsqueda tema

Esta búsqueda contiene un atributo String con el cual se desea buscar el tema. Por cada libro se obtiene el set de palabras claves y se lo recorre verificando si alguna de ellas coinciden con el tema ingresado. En caso que esto suceda se devuelve *true*.

2.3.4 Búsquedas lógicas

Se realizaron 3 tipo de búsquedas lógicas, *and*, *or* y *not*. En el caso del *and* posee dos atributos del tipo *CriterioBusqueda* y devuelve *true* en caso de que se cumplan ambos. En caso del *or* también posee los mismos atributos pero devuelve *true* en caso que algunos de los dos lo sea. En cambio *not* solo contiene un atributo *CriterioBusqueda* y como salida retorna el resultado de ese criterio negado.

2.3.5 Búsqueda Año

Esta búsqueda contiene un atributo entero que representa el año con el que se desea buscar. Si el año del libro provisto es mayor al año con el que se quiere buscar, se devolverá *true*.

2.3.6 Búsqueda editorial

Esta búsqueda contiene un atributo *String*, editorial, con la editorial la cual se quiere buscar. Se compara este atributo con la editorial del libro y en caso que ambas coincidan se retorna *true*.

2.4 Operaciones sobre los grupos

El sistema deberá poder realizar ciertas funciones especificas sobre *ConjuntoLibro.*

2.4.1 Cantidad de elementos

Para esta función, se recorre la lista interna de *ConjuntoLibro* para poder ir accediendo a los distintos subgrupos o al libro. Una vez llegado al libro, el método internamente suma este libro a la cantidad final de libros.

2.4.2 Lista de palabras clave

Un conjunto debera ser capaz de listar las palabras claves de todos sus libros. Se recorre la lista de *ConjuntoLibro* y se obtienen las palabras claves de cada elemento contenido. En caso que hayan varios libros con la misma palabra clave, solo aparece una vez en el resultado final. Para esto se utilizó la clase Set que no agrega repetidos.

2.4.3 Copia restringida de grupos

El sistema requiere que se realice una copia de un *ConjuntoLibro* en la que solo se agregaran los libros que cumplan con cierto requisito. Además se debe respetar la estructura original del conjunto. Este requisito es provisto al sistema usando la clase *CriterioBusqueda*. Esta función recorre la lista interna de *ConjuntoLibro* volviendo a entrar a al método, en el caso de que sea un libro este irá al método implementado en dicha clase y en el caso que sea otro *ConjuntoLibro* seguirá entrando recursivamente. Una vez llegado al método de la clase *Libro*, si cumple con el requisito, se creará la instancia de ese mismo libro y se cargará manualmente las palabras claves y los autores involucrados en ese libro. Luego ese libro es agregado a la copia.

2.4.4 Impresión por pantalla

Se sobreescribe el método *toString* en *ConjuntoLibro* y en *Libro*. Este método imprime los datos del conjunto y todos los libros y subconjuntos que hay en su interior. En el caso de los Libros siguen el formato (Año) Título – Autores - Editorial.

2.5 Ordenamiento

Se creó una clase abstracta llamada *Ordenamiento* la cual implementa a la interfaz *Comparator<Libro>,* esta clase posee un atributo llamado *siguiente* que es del tipo Ordenamiento. Este atributo tiene la función de dar más una forma de ordenar la lista además de la forma principal, en caso que la misma no pueda resolver la forma de ordenarlo. Además posee un método abstracto, *compararLibro*, al cual se le pasan dos libros por parámetros.

2.5.1 Ordenamiento por año

Este ordenamiento hereda de la clase abstracta *Ordenamiento,* esta tiene dos constructores para poder diferenciar que tipo de ordenamiento se realizará. Uno siendo vacío y el otro con el atributo *siguiente*. En el método *compararLibro*, el cual sobreescribe al método del padre, devuelve la resta entre los años de publicación de los libros provistos. En caso que coincida el año de dos libros se puede elegir otro criterio con el cual ordenar con el atributo siguiente.

2.5.2 Ordenamiento por título

Este ordenamiento se realiza de igual forma al ordenamiento anterior pero con la diferencia de que compara con el método *compareTo*, el cual compara lexicográficamente, devolviendo el título del libro el cual sea menor alfabéticamente.

2.6 Conjunto especial

Existe un conjunto especial, el cual es una clase que hereda de de *ConjuntoLibro*. Esta solo debe poseer las diez palabras claves más repetidas de todos sus libros. Para realizar esto, se tiene como atributo una variable estática, esta será un entero con valor diez. En esta clase tendremos sobreescrito el método del padre, *getPalabraClaves.* En este caso se creó un mapa de *String*, *Integer*, el cual almacena la palabra clave con su número de repeticiones. Una vez armado, se llama a la función *ordenarMapa*. Esta función pasa los elementos del mapa a una lista, la cual se ordena utilizando un comparador que compara con el valor *Integer* del mapa, debido a que ordena de menor a mayor también utilizamos la función *reverse* para invertir la lista. Una vez ordenada, se crea un mapa con la lista y se lo devuelve. Con este nuevo mapa ordenado, y utilizando un iterador, se recuperan los 10 primeros valores y se los carga al Set que será devuelto.