Estructuras de Datos

Taller Colas de Prioridad y Comparadores

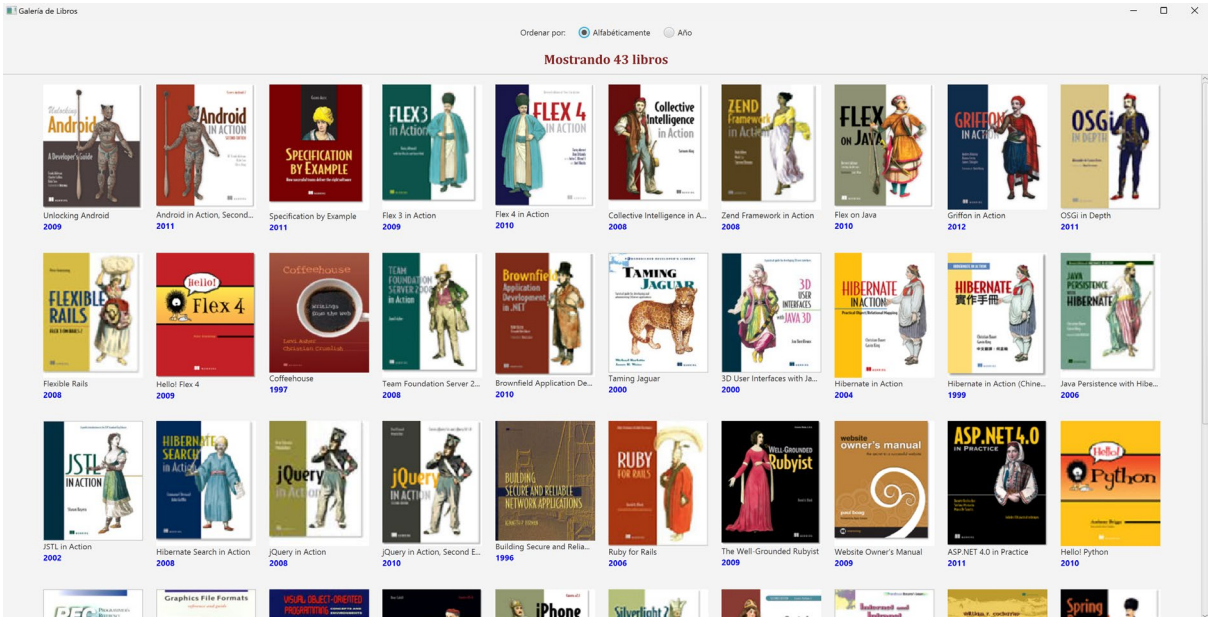
Parte 1

Esta parte se refiere a los archivos **`ImageGallery.java`** y **`Book.java`** del código adjunto

En esta práctica, usted utilizará la clase `PriorityQueue`, que abstrae el concepto de cola de prioridad, para implementar funcionalidad en la interfaz gráfica de una biblioteca digital.

En particular, usted implementará un método que permitirá a los usuarios de la biblioteca ordenar los libros mostrados en la interfaz por año o alfabéticamente (según los títulos).

El archivo `ImageGallery.java`, del paquete `gui` del código adjunto, genera la siguiente interfaz gráfica:



La colección de todos los libros disponibles en la biblioteca se carga mediante el método estático `loadBooks` de la clase Book.

Los *radio buttons* de la interfaz tienen un evento asociado que llama al método `sortBooks` para ordenar los libros mostrados, dependiendo del criterio seleccionado por el usuario (por año o alfabéticamente).

Usted deberá proveer una implementación para el método `sortBooks`. En el código recibido, este método solo muestra un mensaje al usuario indicando lo que eventualmente hará.

Para implementar el método `sortBooks`, usted cuenta con los siguientes atributos de la clase `ImageGallery`:

- `allBooks`: Lista de todos los libros de la galería
- `booksDisplayed`: Esta lista siempre contiene los libros que se están mostrando en la galería en un punto dado. Por tanto, al inicio, es igual a la lista `allBooks`, ya que al inicio se muestran todos los libros. Esta lista es actualizada cada vez que una lista de libros distinta es dibujada en la interfaz (mediante el método `displayBooks` descrito abajo).
- `sorting`: Este atributo lleva control de qué criterio de orden se ha seleccionado en la interfaz. Es una eumeracón de tipo `SortDirection` y puede tomar los valores de `ALPHABETICALLY` o `BY_YEAR`. Este atributo se actualiza cada vez que el usuario da clic sobre uno de los dos radio buttons de la interfaz.

La clase `ImageGallery` cuenta además con el método `displayBooks` que, dada una lista de libros (es decir, un objeto de tipo `List<Book>`), los muestra en la interfaz. Usted debe usar este método, por ejemplo, para mostrar los libros ordenados. Para esto, deberá pasar al método `displayBooks` una lista de libros ordenados. El método se encargaría de crear una vista para cada libro y mostrarlos. Finalmente, este método actualiza la lista `booksDisplayed` para que ésta siempre refleje los libros que se están mostrado en la interfaz.

Para lograr ordenamiento, usted deberá **utilizar colas de prioridad** (clase `PriorityQueue`) de Java. Cuando se ordene por año, si hay varios libros de un mismo año, éstos deben aparecer ordenados alfabéticamente, por título. Si dos libros tienen el mismo título y son de distinto año, éstos deben mostrarse en orden cronológico.

Parte 2

Implemente los ejercicios de esta parte en el archivo **`Parte2.java`** del código adjunto

1. Escriba el método estático **`encontrarMaximo`**, que recibe una lista estática y un comparador como parámetros. El método utiliza el comparador recibido para determinar el máximo elemento en la lista. El mismo método se puede utilizar para encontrar el mínimo si se utiliza un comparador adecuado.

Considere el programa principal mostrado a continuación, donde se utiliza el método **`encontrarMaximo`**. Una vez implementado su método, complete el programa mostrado para que éste imprima lo solicitado:

```
public static void main(String[] args) {

        List<Integer> numeros = new ArrayList<>();
        numeros.add(5);
        numeros.add(8);
        numeros.add(2);
        numeros.add(10);
        numeros.add(3);

        // Crear un comparador que ordene los números en orden descendente
        // cmpDesc = ...

        Integer maximo = encontrarMaximo(numeros, cmpDesc);
        System.out.println("Máximo: " + maximo);


        // Crear un comparador que ordene los números en orden ascendente
        // cmpAsc = ...

        Integer minimo = encontrarMaximo(numeros, cmpAsc);
        System.out.println("Mínimo: " + minimo);
    }
```

Considere que el método **`encontrarMaximo`** que usted debe implementar debe funcionar para listas de cualquier tipo. La lista de enteros utilizada arriba es solo un ejemplo ilustrativo.

Complete la implementación de los siguientes métodos estáticos:

2. **eliminarDuplicados**, que elimina los elementos duplicados de una lista, utilizando un comparador personalizado:

```
/**
 * @param l    La lista de elementos.
 * @param cmp  El comparador personalizado para determinar la igualdad de los
elementos.
 * @param <T>  El tipo de elementos en la lista.
 * @return Una nueva lista sin elementos duplicados.
 */


public static <T> List<T> eliminarDuplicados(List<T> l, Comparator<T> cmp) {


    List<T> listaSinDuplicados = ...


    return listaSinDuplicados;
}
```

3. **buscarIndice**, que busca el índice de un elemento en una lista utilizando un comparador personalizado:

```
/**
 * @param l    La lista de elementos.
 * @param e    El elemento a buscar.
 * @param cmp  El comparador personalizado para determinar la igualdad de los
elementos.
 * @param <T>  El tipo de elementos en la lista.
 * @return El índice del elemento en la lista, o -1 si no se encuentra.
 */


public static <T> int buscarIndice (List<T> l, T e, Comparator<T> cmp) {


    int indice = ...


    return indice;

}
```