Julia García Vega

Programación III – Curso 2021/2022

Práctica Final

Base de datos de películas

El objetivo del proyecto se trata de almacenar y trabajar con una base de datos de películas en la que sea almacena información sobre las películas, los directores y los actores. La estructura del proyecto es la siguiente:

- Paquete practicaFinal:
  - **Clase Practica Final**: Clase en la que se encuentra el main del proyecto. En esta clase se llama inicialmente al método arranque de vista para cargar los datos adecuados, luego se procede a ejecutar el método runMenu de la vista encargado del desarrollo del programa y finalmente tras terminar las operaciones deseadas con el menú se llama al método salida para almacenar los datos tratados con formato binario.
- Paquete vista:
  - **Clase Vista**: Clase que se encarga de presentar los distintos menús y opciones del programa y de mostrar las salidas de cada opción al usuario.
- Paquete controlador:
  - **Clase Controlador**: Clase que se encarga de trasladar información recogida por la vista al modelo o interactuar con este para implementar la lógica de la aplicación.
- Paquete modelo:
  - **Clase Modelo**: Clase que se encarga de almacenar la información recogida y desarrollar la lógica de negocio.
  - **Clase Filmoteca:** Clase cuyos atributos son tres colecciones de datos de tipo Película, Director y Actor y que posee los métodos necesarios para acceder y modificar estos atributos.
  - **Clase Película:** Clase cuyos atributos son los datos de una película y que posee los métodos necesarios para acceder y modificar estos atributos.
  - **Clase Director:** Clase cuyos atributos son los datos de un director y que posee los métodos necesarios para acceder y modificar estos atributos.
  - **Clase Actor:** Clase cuyos atributos son los datos de un actor y que posee los métodos necesarios para acceder y modificar estos atributos.

DISEÑO

Al leer el guion con las instrucciones de la práctica, interpreté que en el modelo debería de a ver las clases Película, Director, Actor, Filmoteca (en la que se almacenan las colecciones) y la clase Modelo en la que se desarrollaran todos los métodos para el desarrollo de las operaciones del programa. Sin embargo, considera que también existe otra interpretación que sería que en el paquete modelo únicamente existieran las clases Película, Director, Actor y Filmoteca y que fuese en esta última clase en la que se encontrar los métodos para el desarrollo de las opciones del programa. Al únicamente por desarrollar un diseño me decanté por el primero. Ambos diseños son muy similares. Para pasar al otro diseño a partir de mi programa, los únicos cambios serían que la clase modelo se eliminaría y todos los métodos que contuviese estarían en la clase Filmoteca y que se accedería a las colecciones y a los atributos de los ficheros directamente.

IMPLEMENTACIÓN

Al ejecutar el programa se cargan los ficheros con formato binario si existen mediante FileInputStream, BufferedInputStream y ObjectInputStream y si no existen se cargan los ficheros.txt. Para implementar esto mediante la bibiblioteca.jar con el método *pathToFileOnDesktop*() de la clase Rutas se obtiene el path al fichero y posteriormente lo convierte en un File para importar una matriz de datos String delimitados por asteriscos mediante el método *importFromDisk*() de la clase *OpMat*. A continuación estos datos leídos se añaden a las respectivas colecciones y se procede a mostrar el menú principal.

Opción Archivos:

- **Exportar directores en formato columnas**. Para ello se calcula los caracteres máximos de cada columnas para establecer el ancho de cada columna y se procede a dar este formato a cada cadena que contiene la información de un director. Este conjunto de cadenas se exporta al fichero obteniendo el File al fichero mediante *fileToFileInFoldesOnDesktop*() de la clase rutas y a continuación escribiendo en el fichero mediante *Files.write().*
- **Exportar películas en un archivo html con formato de tabla**. Para ello se obtiene el File al fichero con el mismo método anterior y con *PrintWriter* se escribe en el fichero las sentencias de html para la presentación y creación de la tabla y se va escribiendo el encabezado de las películas y cada película ya transformadas cada una a una cada con las sentencias de html para su presentación mediante los métodos *tableHeader* y *stateAsStringHTML* que se encuentran en la clase película.

Opción Películas:

- **Altas.** Se piden y se leen los datos de las películas en la vista dando error y volviendo a solicitar el dato si se introduce un dato inválido. A continuación en el modelo, se crea un nuevo ejemplar de tipo Película con los datos recibidos y se añade a la colección de películas. Tras esto, se busca los nombres de los directores y de los actores en sus respectivas colecciones mediante bucles y comprobaciones con *equalsIgnoreCase*() y si coincide en estos nombres con alguno de los ya existentes en sus respectivas colecciones se añade el título de la película añadida a la lista de películas de los actores o directores. Los nombres de aquellos directores y actores que no se encuentren ya registrados, que se detectan al seguir el *flag* denominado *encontrado* con valor falso, se añaden a sus respectivas colecciones con el título de la película dada de alta y el resto

de atributos al no ser conocidos se les da un valor predeterminado para evitar fallos por datos nulos: los tipo string como "-DESCONOCIDO-", las fechas como 9999-09-09 y el año como 9999.

- **Bajas.** Se solicita en la vista el título de la película a eliminar. Ya en el modelo se elimina de la colección de películas aquella cuyo título coincida con el título introducido mediante removeIf con la condición *equalsIgnoreCase().* A continuación se obtiene para cada director su lista de películas y se eliminan de la lista aquellas que también coincidan con el título introducido mediante el mismo método anterior. Lo mismo se realiza para los actores. Aquellos actores y directores que se queden sin películas procedo a eliminarlos al no proporcionar información sobre las películas y poder sobrecargar la base de datos con información que no aporta nada.
- **Modificaciones.** Se solicita en la vista el título de la película, el atributo que se quiere modificar y su información. A continuación en la vista se recorre la colección de películas buscando a aquella que cuyo título coincida con el introducido con *equalsIgnoreCase().* En aquella película que coincida el título se cambia el valor de un *flag* denominado encontrado y con un set de modifica el atributo. Si al terminar de recorrer todas las películas el *flag* sigue con valor falso se devuelve una String con un mensaje de error que indica que la película no se encuentra registrada.
- **Consulta.** El procedimiento es similar al de modificaciones cambiando que únicamente se solicita el título de la película y que al encontrar la película en la colección de películas se añade a la segunda fila de una matriz que tiene en su primera fila el encabezado los datos de la película como datos String. En la vista se imprime esta matriz con el método *printToScreen2()* de la clase OpMat.

Opción director:

- **Alta.** Procedimiento igual que el de la opción película cambiando los atributos solicitados y únicamente dejando la parte de la adicción a la colección.
- **Baja.** Procedimiento de búsqueda de director en la colección igual a los explicados anteriormente. Al encontrar el director en la colección se obtiene su lista de películas y se recorren estas comprobando una a una que no exista en la colección de películas. Aquella que exista se almacena en un nuevo ArrayList. Al terminar de recorrer toda la lista de películas del director si el ArrayList creado sigue vacío se elimina al director. Se devuelve el ArrayList creado y en la vista se comprueba si está vacío o no. Si esta vacío muestra un mensaje de funcionamiento correcto y si no muestra un mensaje de error con las películas que siguen existiendo en la colección de películas.
- **Modificaciones.** Procedimiento igual al de la opción películas.

Opción actor.

- **Alta.** Procedimiento igual a la de director variando los atributos.
- **Baja.** Procedimiento igual al de director
- **Modificaciones.** Procedimiento igual a las anteriores opciones.
- **Películas de un actor.** Se solicita en la vista el nombre del actor y en el modelo se busca el actor en la colección de actores con el mismo procedimiento de búsqueda ya explicado. Al encontrar al actor se guardan su lista de películas en un nuevo ArrayList de Strings. Luego se recorre esta lista haciendo para cada película otro recorrido de toda la colección de películas y en aquellas que coincidan los títulos se almacena en una matriz ya con encabezado los datos de la película a mostrar como Strings. Si la película del actor

no se encuentra en la colección de películas se añade a la matriz por abajo (para mantener el orden por año) el título de la película y el resto de datos como los valores establecidos como desconocidos. En esta matriz ya se encuentran ordenadas las películas por el año al haber hecho previamente a los bucles la ordenación de la colección de películas por el año con la siguiente sentencia: coleccionDePeliculas.sort(Comparator.comparing(Pelicula::getAnio)). Finalmente se devuelve esta matriz y se imprime en la vista mediante el método *printToScreen3();*

Opción listados:

- **Listado películas.** En el modelo se ordena la colección de películas por título mediante la sentencia *coleccionDePeliculas.sort(Comparator.comparing(Pelicula::getTitulo));* y se devuelve la matriz de datos String como se ha hecho anteriormente.
- **Listado directores.** Igual que lo anterior pero ordenando mediante las siguientes sentencias:
  Comparator<Director> nacionalidad =
  Comparator.comparing(Director::getNacionalidad);
  Comparator<Director> fechaNac = Comparator.comparing(Director::getFechaNac);
  coleccionDeDirectores.sort(nacionalidad.thenComparing(fechaNac));
- **Listado actores.** Igual que lo anterior pero ordenando mediante las siguientes sentencias: Comparator<Actor> anio = Comparator.comparing(Actor::getAnio);
  Comparator<Actor> nombre = Comparator.comparing(Actor::getNombre);
  coleccionDeActores.sort(anio.thenComparing(nombre));

Al salir del menú principal, antes de terminar la ejecución del programa, en el main se llama al método salida de la vista desde el cual al final se llega al método de salida del modelo. Desde aquí mediante FileOutputStream, BufferedOutputStream y ObjectOutputStream.