Aplicación para gestionar actores y películas

Fecha: 30-IX-2021

Participantes: Xabier Gabiña
Javier Criado

1 Índice

1 Introducción

Para la asignatura de Estructuras de datos y algoritmos se nos ha planteado realizar una aplicación en la que hemos tenido que gestionar una lista de actores y películas. Hemos diseñado e implementado un diseño con el que hemos podido:

- Cargar los datos desde un fichero
- Buscar un actor/actriz
- Insertar un nuevo actor/actriz
- Devolver (no imprimir) las películas de un actor
- Devolver (no imprimir) los actores de una película
- Incrementar el dinero recaudado por una película en un valor dado
- Borrar un actor/actriz
- Guardar la lista en un fichero
- Obtener una lista de actores ordenada por nombre y apellido

2 Diseño de las clases

Para realizar este proyecto hemos implementado 8 clases, las cuales son Actor, Fichero, HashMap_Actores, HashMap_Peliculas, Main, Menu, Pelicula y StringQuickSort.

Actor

La clase Actor es una TAD, con dos atributos el nombre del actor y una ArrayList de películas donde se guardan las películas en las que aparece el actor. El método getNombre() devuelve el nombre, el método tiene el mismoNombre() devuelve un booleando comparando el nombre dado con el del actor, el método anadirPeliculaALista() añade una película a la lista del actor, el método getListaPelicula() devuelve la lista de películas, el método imprimirLista() imprime las películas en las que aparece el actor.

Fichero

La clase Fichero es una MAE. El método cargarFichero() carga el fichero a la aplicacion, el método crearFichero() crea un fichero.

HashMap_Actores

La clase HahsMap_Actores es una MAE que tiene un atributo HashMap, con una key de tipo String y un value de tipo Actor. El método anadirActor() añade el actor que se ha dado, el método eliminarActor() elimina el actor solicitado, el método buscarActor() devuelve el actor buscado, getListaNombresActoresOrdenada() devuelve un Array de , el método getListaActoresOrdenada() devuelve una lista de actores ordenada, el método reset() resetea el HashMap.

HashMap_Peliculas

La clase HahsMap_Peliculas es una MAE que tiene un atributo HashMap, con una key de tipo String y un value de tipo Pelicula. El método anadirEntrada() añade la película al HashMap, el método buscarPelicula() devuelve la película que se busca, el método escribirContenido() añade la película y la lista de actores al fichero, el método reset() resetea el HashMap.

Main

La clase Main se encarga de llamar al método showMenu() de la clase Menu que inicia la aplicación.

Menu

La clase Menu es una MAE. El método showMenu() muestra un menú con diferentes opciones donde tendrás que elegir la opción correspondiente a la acción que quieras realizar.

Pelicula

La clase Pelicula es una TAD que tiene 3 atributos el título de la película, la cantidad recaudara y una lista de los actores que pertenecen a las películas. El método getNombre() devuelve el nombre de la película, el método incrementarRecaudacion() incrementa la recaudación de la película, el método anadirActorALista() añade el actor a la lista de actores que pertenecen a la película, el método eliminarActorALista() elimina un actor de la lista de actores de la película ,el método actorEstaEnLista() devuelve un booleano indicando si el actor esta en la lista, el método getLista() devuelve la lista de actores, el método imprimirLista() imprime la lista de actores participando en la película.

StringQuickSort

La clase StringQuickSort es una MAE, la cual tiene los métodos necesarios para ordenar los elementos de una lista con un coste O(nlgn)

3 Descripción de las estructuras de datos principales

El problema se ha resuelto creando un menú desde el que se accede a cada función de la aplicación. Para guardar los datos de los ficheros se han creado dos *HashMap* uno para los actores y otro para las películas gracias a su fácil acceso que permite una gran eficiencia.

El *HashMaps* de Películas se divide en dos partes: Las *key*, que es el nombre de la película y el valor que sería un objeto Película. El objeto película a su vez tiene tres atributos, el atributo título te tipo *String* que guarda el mismo valor que la *key*, es decir el título de la película. Un *ArrayList* de *Actores* en los que se guardan los actores que actúan en dicha película. El tercer atributo es un *float* llamado recaudación que como su nombre indica es el dinero recaudado por la película que se inicia a 0 que se puede cambiar desde una opción del menú.

El *HashMap* de *Actores* al igual que el de *Películas* tiene dos partes: La *key* que es el nombre del actor y el valor que es un objeto de clase actor con en este caso dos atributos. El atributo nombre de clase *String* que contiene el nombre y apellido del actor y un *ArrayList* de *Películas* que contiene las películas en las que aparece el actor.

Estas estructuras nos permiten hacer búsquedas y modificaciones en las listas con mucho

menor tiempo que si usásemos otras estructuras como ArrayList o LinkedList.

4 Diseño e implementación de los métodos principales

4.1 Método cargarFichero()

public void cargarFichero() {

/* Precondición:

/* Postcondición: Cargará el fichero

Casos de prueba:

- Que no exista el fichero
- Que exista el fichero

Implementación:

Mientras tenga líneas

Separar en dos arrays(Película y actores)

Por cada elemento del array de actores

Añadir la película a la lista de películas del actor

Añadir el actor a la lista de actores de la película

Añadir el actor al HashMap de actores

Añadir la película al HashMap de Peliculas

Coste: El algoritmo es de coste lineal n. Depende del número de elementos que tengamos que guardar en el fichero. Coste O(n).

4.2 Método showMenu()

public void showMenu() {

/* Precondición:

/* Postcondición: Mostrará el menú

Casos de prueba:

- Introducir un número de 1 al 9.
- Introducir un número mayor que 9 y menor que 1.
- Introducir una letra

Implementación: Muestra el menú con el cual se podrá manejar la aplicación

Coste: El coste del algoritmo variara en función del número de acciones que se realicen hasta finalizar de usar la aplicación

4.3 Método anadirActor ()

public void anadirActor(String key, Actor valor, Pelicula pPeli){

/* Precondición: El valor de key y valor no pueden ser null a no ser que ambos dos

lo sean.

/* Postcondición: Añade el actor

Casos de prueba:

Implementación:

Si la key es null entonces

Pedir que introduzca otra key

Si la key no pertenece al HashMap entonces

Añade el actor

Si la key pertenece al HashMap entonces

Elimina la key en el HashMap y añde una nueva key con la película dada

Coste: El coste del algoritmo es O(1) porque utiliza las funciones HashMap que son de coste constante

4.4 Método eliminarActor()

public void eliminarActor(String key)

/* Precondición:
/* Postcondición: Elimina al actor

Casos de prueba:

- Un actor que exista
- Un actor que no exista

Implementación:

Si la key es null entonces

Pedir que introduzca otra key

Si la key pertenece a un actor entonces

Elimina al actor

Si la key no pertenece a un actor

Imprime un mensaje por pantalla

Coste: El coste es O(1) porque utiliza las funciones HashMap.

4.5 Método buscarActor()

public Actor buscarActor(String key)

/* Precondición:
/* Postcondición: Devuelve el actor

Casos de prueba:

- Un actor que exista
- Un actor que no exista

Implementación:

Si Key==null entonces

Mostrar mensaje (añade una key)

Si la key es valida entonces

Devuelve el actor

Si no entonces

Mostrar mensaje(Key no valida)

Devuelve null

Coste: O(1) las funciones HashMap son de coste constante

4.6 Método anadirEntrada()

/* Precondición:
/* Postcondición: añade la película

Casos de prueba:

- Añadir una key que no existe
- Añadir una key que existe
- Añadir una Pelicula que no existe
- Añadir una Pelicula que existe

Implementación: Si el HashMap no contiene la key, la añade junto a la película

Coste: O(1) las funciones HashMap son de coste constante

4.7 Método buscarPelicula()

public Pelicula buscarPelicula(String key)

/* Precondición:
/* Postcondición: Devuelve la película

Casos de prueba:

- Introducir una key valida
- Introducir una key no valida

Implementación:

Si Key==null entonces

Mostrar mensaje(añade una key)

Si no entonces

Devuelve la película

Coste: O(1) las funciones del HashMap son de coste constante

4.8 Método anadirActorALista()

public void anadirActorALista(Actor unActor)

/* Precondición:
/* Postcondición: Añadirá el actor a la lista

Casos de prueba:

- El actor se encuentra en la lista
- El actor no se encuentra en la lista

Implementación:

```
i := 0
enc := 0
mientras (i <= n) y no enc hacer
        si el elemento i-ésimo es igual a x
        entonces enc := true
        si no i := i + 1
si !enc entonces
        añadir actor a la lista
si enc
        devolver un mensje(actor ya en lista)
```

Coste: O(n) porque tiene que recorrer toda la lista en busca del actor en el peor de los casos

4.9 Método actorBuscarEnLista ()

public Actor actorBuscarEnLista(Actor pActor)

/* Precondición:
/* Postcondición: Devuelve el actor si se encuentra en la lista, en caso contrario devuelve null

Casos de prueba:

- Un actor que este
- Un actor que no este

Implementación:

```
i := 0
enc := 0
mientras (i <= n) y no enc hacer
        si el elemento i-ésimo es igual a x
        entonces enc := true
        si no i := i + 1
si !enc entonces
        devolver null
si enc
        devolver x
```

Coste: O(n) hay que recorrer la lista en el peor de los casos.

4.10 Método eliminarActorDeLista ()

public void eliminarActorDeLista(Actor unActor)

/* Precondición:
/* Postcondición: Elimina el actor de la lista

Casos de prueba:

- Un actor que este
- Un actor que no este

Implementación:

```
i := 0
enc := false
mientras (i <= n) y no enc hacer
        si el elemento i-ésimo es igual a x
        entonces enc := true
        si no i := i + 1
si enc entonces
        escribir mensaje por panatlla(actor eliminado)
        eliminar el actor
si !enc
        escribir mensaje por panatlla(actor no encontrado)
```

Coste: O(n) hay que recorrer la lista en el peor de los casos.

4.11 Método actorEstaEnLista ()

public boolean actorEstaEnLista(Actor pActor)

/* Precondición:
/* Postcondición: True si esta en la lista, False si no esta

Casos de prueba:

- Un actor que este
- Un actor que no este

Implementación:

```
i := 0
enc := 0
mientras (i <= n) y no enc hacer
        si el elemento i-ésimo es igual a x
        entonces enc := true
        si no i := i + 1
si !enc entonces
        devolver null
si enc entonces
        devolver x
```

Coste: O(n) hay que recorrer la lista en el peor de los casos.

5 Código

En este apartado se presentará el código de las clases, junto con los programas de prueba o Junits desarrollados.

Se puede insertar un índice eligiendo “Insertar” → “Índices” → “Índices” y después la pestaña “Índice” (opción “índice de contenido”).

5.1 Método cargarFichero()

```
public void cargarFichero()
   {
      long statTime=System.nanoTime();
      try
      {
         Scanner entrada = new Scanner(new FileReader(Dir));
         String linea;
         while (entrada.hasNext())
         {
            linea=entrada.nextLine();

            String [] sub1 = linea.split(" --->>>"+"\\s+");
            String [] sub2 = sub1[1].split(" #####"+"\\s+");

            String Title=sub1[0];
            Pelicula Peli=new Pelicula(Title);

            for (String s : sub2) {
               Actor unActor = new Actor(s);
               unActor.anadirPeliculaALista(Peli);
               Peli.anadirActorALista(unActor);

               HashMap_Actores.getMiMapa().anadirActor(s,unActor,Peli);
            }

            HashMap_Peliculas.getMiMapa().anadirEntrada(Title,Peli);
         }
         entrada.close();
      }
      catch (IOException e)
      {
         e.printStackTrace();
      }
      long endTime=System.nanoTime();
      System.out.println(((endTime-statTime)/1000000000)+" segundos a tardado en eje-
cutarse");
   }
```

5.2 Método showMenu()

```
public void showMenu()
   {
      Scanner sn=new Scanner(System.in);
      boolean exit=false;
```

```
        int opcion;
        Fichero datos = Fichero.getFichero();
        HashMap_Peliculas HM_Peli = HashMap_Peliculas.getMiMapa();
        HashMap_Actores HM_Actor = HashMap_Actores.getMiMapa();
        Pelicula unaPelicula;
        Actor unActor;

        while(!exit)
        {
            Sys-
tem.out.println("\n#####################################################");
            System.out.println("\t\t Menu Principal \n");
            System.out.println("Seleccione una de las siguientes opciones:\n");
            System.out.println("0. Cargar los datos");
            System.out.println("1. Buscar actor/actriz");
            System.out.println("2. AÃ±adir actor/actriz");
            System.out.println("3. Obtener las peliculas de un actor/actriz");
            System.out.println("4. Obtener los actores/actrices de una pelicula");
            System.out.println("5. Incrementar la recaudacion de una pelicula");
            System.out.println("6. Eliminar un actor/actriz");
            System.out.println("7. Guardar los datos en un fichero");
            System.out.println("8. Obtener lista ordenada de actores");
            System.out.println("9. Finalizar Programa");
            System.out.print("---> ");
            opcion=sn.nextInt();

            switch (opcion)
            {
                case 0:
                    HM_Actor.reset();
                    HM_Peli.reset();
                    datos.cargarFichero();
                    break;
                case 1:
                    unActor=HM_Actor.buscarActor(null);
                    if (unActor!=null) System.out.println(unActor.getNombre()+" encontrado");
                    break;
                case 2:
                    HM_Actor.anadirActor(null,null,null);
                    break;
                case 3:
                    unActor= HM_Actor.buscarActor(null);
                    if (unActor!=null) unActor.imprimirLista();       //COMPROBACION
                    break;
                case 4:
                    unaPelicula=HM_Peli.buscarPelicula(null);
                    if (unaPelicula!=null) unaPelicula.imprimirLista();   //COMPROBACION
                    break;
                case 5:
                    unaPelicula=HM_Peli.buscarPelicula(null);
                    if (unaPelicula!=null) unaPelicula.incrementarRecaudacion(-1);
                    break;
```

```
            case 6:
                HM_Actor.eliminarActor(null);
                break;
            case 7:
                datos.crearFichero();
                System.out.println("El contenido se ha escrito correctamente");
                break;
            case 8:
                for(Actor s:HM_Actor.getListaActoresOrdenada()){Sys-
tem.out.println(s.getNombre());}
                break;
            case 9:
                exit=true;
                break;
            default:
                System.out.println("Solo numeros del 0 al 9\n");
        }
    }
}
```

5.3 Método anadirActor()

```
public void anadirActor(String key, Actor valor, Pelicula pPeli)
  {
    if(key==null)
    {
       System.out.println("Introduce el nombre del Actor a aÃ±adir");
       Scanner sn=new Scanner(System.in);
       key=sn.nextLine();
       if(valor==null)
          valor=new Actor(key);
    }
    if (!mapa.containsKey(key))
    {
       mapa.put(key, valor);
    }
    else
    {
       Actor unActor=mapa.get(key);
       unActor.anadirPeliculaALista(pPeli);
       mapa.remove(key);
       mapa.put(key,unActor);
    }
  }
```

5.4 Método eliminarActor()

```
public void eliminarActor(String key)
  {
    if(key==null)
    {
```

```
            System.out.println("Cual es el nombre del actor a eliminar");
            Scanner sn=new Scanner(System.in);
            key=sn.nextLine();
        }

        Actor unActor=buscarActor(key);
        if (unActor!=null)
        {
            ArrayList<Pelicula> lista=unActor.getListaPelicula();
            for(Pelicula peli : lista)
            {
                HashMap_Peliculas.getMiMapa().buscarPelicula(peli.getNombre()).eliminarAc-
torDeLista(unActor);
            }
            mapa.remove(key);
        }
        else
        {
            System.out.println("No se ha encontrado actor con ese nombre");
        }
```

5.5 } Método buscarActor()

```
public Actor buscarActor(String key)
    {
        if(key==null)
        {
            System.out.println("\nIntroduce el nombre del actor");
            Scanner sn=new Scanner(System.in);
            key = sn.nextLine();
        }
        if (mapa.containsKey(key))
        {
            return mapa.get(key);
        }
        else {
            System.out.println("No se ha encontrado Actor con ese nombre");
            return null;
        }
    }
```

5.6 Método anadirEntrada()

```
public void anadirEntrada(String key, Pelicula valor)
    {
        if(!mapa.containsKey(key))
            mapa.put(key, valor);
    }
```

5.7 Método buscarPelicula()

```
public Pelicula buscarPelicula(String key)
    {
```

```
        if(key==null) {
            System.out.println("\nIntroduce el nombre de la pelicula");
            Scanner sn = new Scanner(System.in);
            key = sn.nextLine();
        }
        return mapa.get(key);
    }
```

5.8 Método anadirActorALista()

```
public void anadirPeliculaALista(Pelicula unaPelicula)
    {
        if(!lista.contains(unaPelicula))
        {
            lista.add(unaPelicula);
        }
    }
```

5.9 Método actorBuscarEnLista ()

```
public Actor actorBuscarEnLista(Actor pActor)
    {
        Iterator<Actor> itr=lista.iterator();
        boolean esta=false;
        Actor unActor=null;
        while(itr.hasNext()&&!esta)
        {
            unActor=itr.next();
            if(unActor.tienenMismoNombre(pActor))
            {
                esta=true;
            }
        }
        if (!esta)
        {
            unActor=null;
        }
        return unActor;
    }
```

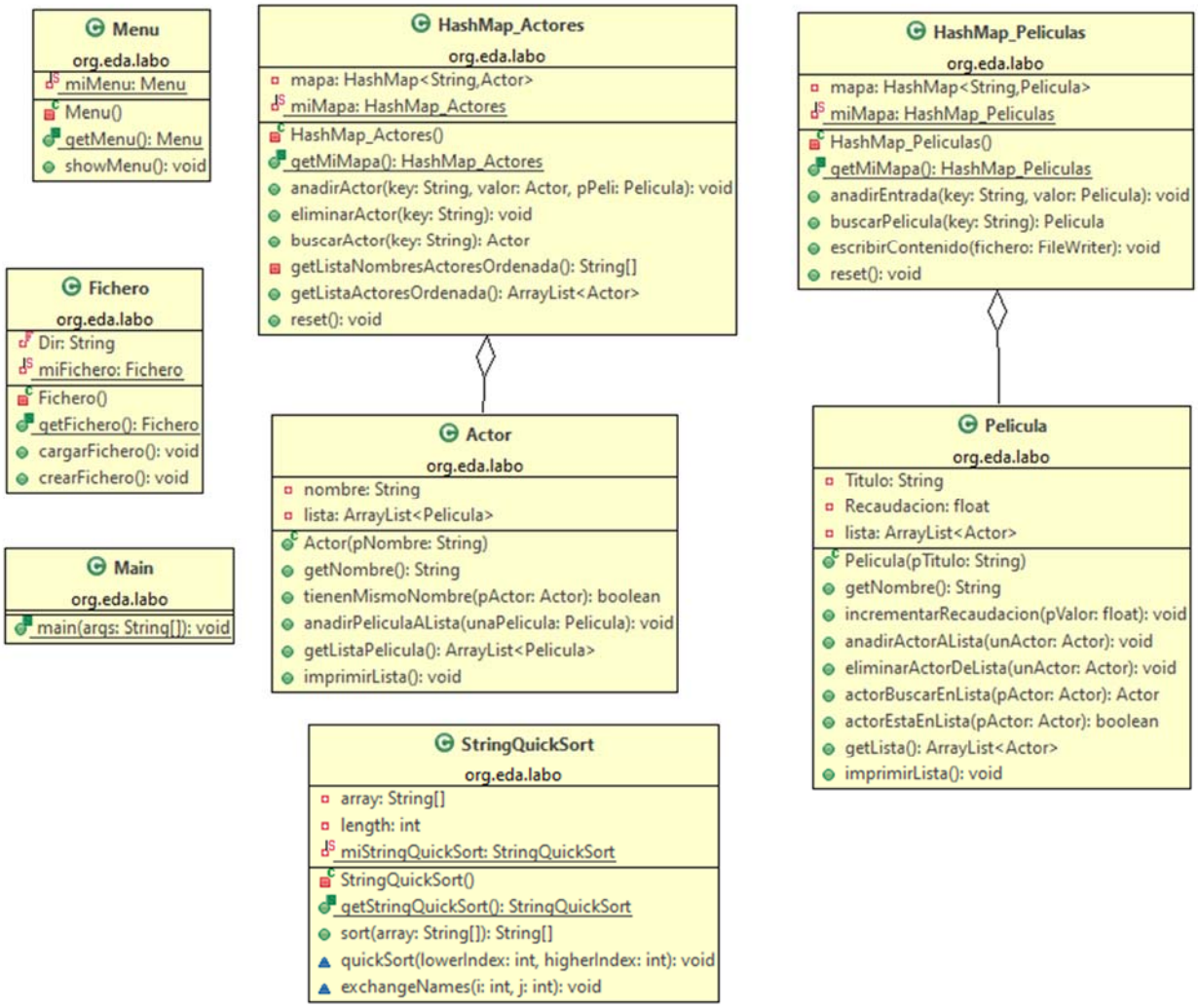
5.10 Método eliminarActorDeLista ()

```
 public void eliminarActorDeLista(Actor unActor)
    {
        if(actorEstaEnLista(unActor))
        {
            System.out.println("Actor "+unActor.getNombre()+" eliminado de "+getNombre());
            lista.remove(actorBuscarEnLista(unActor));
        }
        else
        {
            System.out.println("El actor no estaba en la lista");
        }
```

```
}
```

5.11 Método actorEstaEnLista ()

```
public boolean actorEstaEnLista(Actor pActor)
 {
    Iterator<Actor> itr=lista.iterator();
    boolean esta=false;
    Actor unActor=null;
    while(itr.hasNext()&&!esta)
    {
       unActor=itr.next();
       if(unActor.tienenMismoNombre(pActor))
       {
          esta=true;
       }
    }
    return esta;
 }
```



6 Conclusiones

Debido al gran numero de datos que teníamos que tratar en este proyecto hemos podido comprender la importancia de implementar unos métodos eficientes y las estructuras de datos correctas ya que de otra forma no abría sido posible cumplir con todos los requisitos del proyecto y con el gran número de información con la que trabajar.

Gracias a todo lo aprendido hemos sido capaces de indexar en dos hashmaps un documentos con mas de 3.000.000 de entradas y ser capaces de modificar, ordenar, añadir, eliminar y consultar las entras de forma eficiente.

La principal dificultad encontrada en el proyecto es la búsqueda de esa efectividad de la que tanto estamos hablando y que es la clave para trabajar con tantos datos. Para ello nos hemos visto obligados a investigar en internet sobre las estructuras de datos mas eficientes y los métodos de ordenación más eficientes.