EDA: Registro de películas

Fecha: 13-X-2016

Participantes:
Mikel Abad
Andima Freire
Julen Mendiguren

ÍNDICE

1 Introducción

Se nos pide crear una aplicación que gestione un número muy grande de actores/actrices y las películas en que participan. Un actor o actriz puede participar en numerosas películas por lo que se debe usar el modelo de dominio siguiente:

Los datos a gestionar inicialmente se deben cargar de un fichero de texto con el siguiente formato:

Kima ---> Corradini, Tom &&& Fazzolari, Daniela (II) &&& Nyokabi, Esther &&& Nyokabi, Wambui &&& Quaglia, Martina

All City ---> Yip, Jay Oliver (I) &&& Bakerdias, Keen &&& Gurney, Francesca &&& Hunter, Lynn (IV) &&& Prowse, Matthew

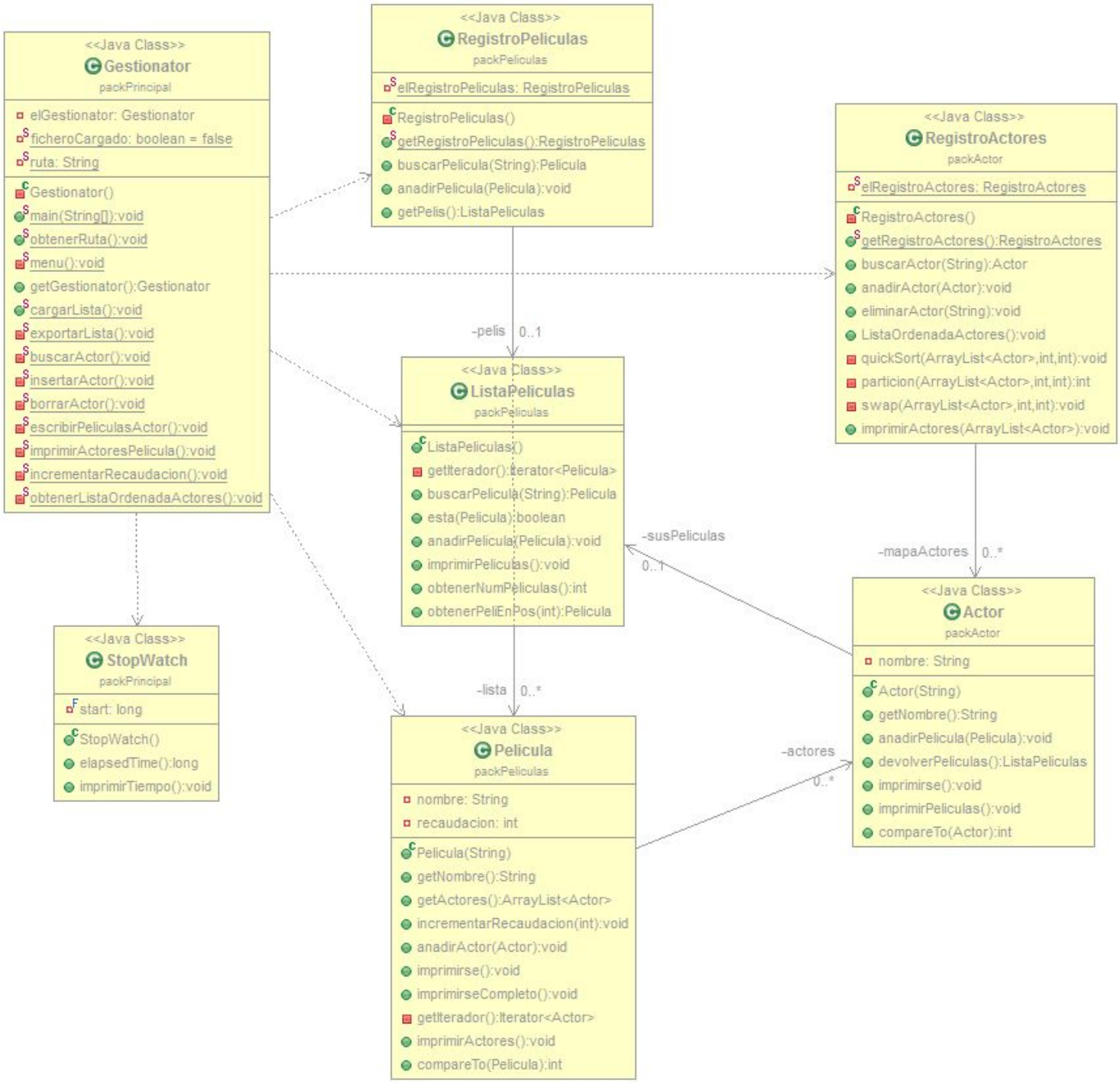
---

Los nombres de las películas se han marcado en negrita para que se distingan mejor (este marcado no se encuentra presente en el fichero de datos proporcionado). Como vemos las películas se separan por "--->" de los actores, y estos últimos por "&&&".

Las funcionalidades de la aplicación que se nos pide son las siguientes:

- Cargar los datos desde un fichero
- Búsqueda de un actor/actriz
- Inserción de un nuevo actor/actriz
- Devolver las películas de un actor dado
- Devolver los actores de una película dada
- Incrementar el dinero recaudado por una película en un valor dado
- Borrado de un actor/actriz
- Guardar la lista en un fichero
- Obtener una lista de actores ordenada (nombre, apellido).

2 Diseño de las clases



3 Descripción de las estructuras de datos principales

En este trabajo hemos utilizado varias estructuras:

Para organizar las películas hemos creado la clase ListaPeliculas, que contiene un ArrayList de películas y sirve para manejarlo. Hemos hecho esto para poder crear varias listas con ese mismo tipo y que sea más eficiente. De este modo, tenemos la lista principal de películas en nuestro RegistroPeliculas y al mismo tiempo cada actor tiene una lista de películas en las que ha participado.

Para los actores hemos optado por hacerlo de una manera diferente. En este caso la estructura principal para organizarlos ha sido un HashMap que contiene el nombre de cada actor junto al objeto, en la clase RegistroActores. De esta manera accedemos con un coste constante al actor deseado. Por otro lado, cada película tiene su propio ArrayList de actores, porque sólo la utilizamos en esa clase y por tanto no hay necesidad de definir una clase específica para ello. Para devolver la lista de actores ordenada volcamos el HashMap en un ArrayList, lo cual resulta muy útil a la hora de ordenar.

Nuestra clase principal, el administrador del programa, tiene un menú que se encarga de realizar las llamadas necesarias a los métodos pertinentes según el deseo del usuario. Esto nos ha evitado la necesidad de realizar diferentes JUnits.

4 Diseño e implementación de los métodos principales

En este apartado se presentarán los métodos principales, indicando por cada uno su especificación (precondiciones y postcondiciones), además de los casos de prueba planteados. Por cada método no trivial, se presentará la descripción del algoritmo implementado. Finalmente, se deberá presentar, de manera razonada, el cálculo del coste de cada algoritmo.

Presentamos un ejemplo de una posible manera de presentar un método de ejemplo.

4.1 Método cargarLista

```
public static void cargarLista(){
/* Postcondición: carga en los registros todo lo que el
fichero contiene separado por strings. Si no encuentra el
fichero lanza excepción. */
```

Casos de prueba:

- Cargar un fichero vacío
- Intentar cargar un fichero que no existe
- Cargar un fichero con muchos elementos

Implementación del algoritmo:

```
intentamos
      leer el fichero ubicado en la ruta que hemos especificado
      mientras haya una línea de texto separamos la línea en un array
      de strings de dos posiciones, por un string especificado en un
      patrón, el nombre de la película y los de los actores de esa
      película
            guardamos la película en el registro
            separamos el array con la posición de los nombres de actores
            en un array de strigs
            recorremos dicho array
                        si no estaba
                         guardamos el actor en el registro
                  añadimos el actor a la película
                  añadimos la película al actor

      fin mientras
```

Coste: Teniendo en cuenta que los actores sean *n* y las películas *m* el coste sería O(n*m)

4.2 Método buscarActor

```
public Actor buscarActor(string nombre) {
/* Postcondición: devuelve el actor si lo encuentra en el
mapa, si no devuelve null */
```

Casos de prueba:

- Búsqueda de un elemento en mapa vacío
- Búsqueda de un elemento en mapa de un solo elemento:
    - El elemento buscado es igual al del mapa
    - El elemento buscado no es el del mapa
- Búsqueda en un mapa no vacío
    - El elemento buscado no está en el mapa
    - El elemento buscado es el primero del mapa
    - El elemento buscado es el último del mapa
    - El elemento buscado está en la mitad del mapa

Implementación del algoritmo:

```
devolver elemento con key igual a nombre
```

Coste: el algoritmo accede directamente al elemento con la key seleccionada y lo devuelve, ya sea null (porque no hay ningún elemento con esa key), o el elemento buscado por lo que el coste es constante O(1).

4.3 Método insertarActor

```
private static void insertarActor() {
/* Se pide el nombre de un actor al usuario, lo crea y llama
a añadirActor que lo añade a la lista.
Postcondición: El actor estará en la lista */
```

Casos de prueba:

- El mapa está vacío.
- El actor ya existe en el mapa.
- El actor no está en el mapa.
- Sólo hay un actor en el mapa:
    - El actor es el que intentamos insertar.
    - No está el actor.
    -

Implementación del algoritmo:

```
Pedimos el nombre del actor por pantalla
leemos el nombre con un escáner que lo guarda en un String
Creamos un objeto actor con ese nombre
comprueba si ese actor está en el mapa de actores
     si no lo está lo añade al mapa
```

Coste: el algoritmo escribe por pantalla, guarda lo recibido en un String, lo busca en el mapa con una key y lo inserta si no está. El coste es constante O(1).

4.4 Método escribirPeliculasActor

```
private static void escribirPeliculasActor() {
/* Se pide el nombre de un actor al usuario, lo busca e
imprime sus películas.
Postcondición: Se verán por pantallas las películas del
actor si estaba en el registro y si no dirá que no estaba */
```

Casos de prueba:

- Casos de prueba de buscarActor
- La lista de películas está vacía:
- La lista de películas no está vaca
    - El actor tiene una película
    - El actor tiene varias películas

Implementación del algoritmo:

```
preguntar nombre actor
crear escáner
escanear nombre
buscar el actor en el registro
si está
    imprimir sus pelis con iterador
si no
    imprimir que no está
```

Coste: El algoritmo accede directamente al elemento en el mapa y lo devuelve con coste constante O(1) y después imprime cada película recorriendo el array. Siendo *n* las películas de dicha lista, este método tiene coste O(n).

4.5 Método imprimirActoresPelicula

```
private static void imprimirActoresPelicula() {
/* Se pide el nombre de una película al usuario, lo busca e
imprime los actores del rodaje.
Postcondición: Se verán por pantallas la película y los
actores de ella si estaba en el registro y si no dirá que no
estaba */
```

Casos de prueba:

- Búsqueda de un elemento en array vacío
- Búsqueda de un elemento en array de un solo elemento:
    - El elemento buscado es igual al del array
    - El elemento buscado no es el del array
- Búsqueda en un mapa no vacío
    - El elemento buscado no está en el array
    - El elemento buscado es el primero del array
    - El elemento buscado es el último del array
    - El elemento buscado está en la mitad del array

Implementación del algoritmo:

```
preguntar nombre película
crear escáner
escanear nombre
buscar la película en el registro
si está
    imprimir sus datos y actores
si no
    imprimir que no está
```

Coste: El algoritmo busca el elemento en el array de películas y lo devuelve, con coste O(n), teniendo en cuenta que comprueba *n* películas hasta encontrarla. Después imprime su información, entre ella los actores del reparto, que recorre uno a uno para imprimirlos, con coste O(m), teniendo en cuenta que hay *m* actores. Por tanto este algoritmo tiene coste O(n*m).

4.6 Método incrementarRecaudacion

```
private static void incrementarRecaudacion() {
/* Se pide el nombre de una película al usuario, lo busca,
pide una cantidad al usuario y la suma a su recaudación.
Postcondición: Se verán por pantallas la película y la nueva
recaudación */
```

Casos de prueba:

- Búsqueda de un elemento en array vacío
- Búsqueda de un elemento en array de un solo elemento:
    - El elemento buscado es igual al del array
    - El elemento buscado no es el del array
- Búsqueda en un mapa no vacío
    - El elemento buscado no está en el array
    - El elemento buscado es el primero del array
    - El elemento buscado es el último del array
    - El elemento buscado está en la mitad del array
- La recaudación es cero
- La recaudación no es cero
-

Implementación del algoritmo:

```
preguntar nombre película
crear escáner
escanear nombre
buscar la película en el registro
si está
    pide recaudación
    escanea el número
    suma la recaudación
si no
    imprimir que no está
```

Coste: El algoritmo busca el elemento en el array de películas y lo devuelve. Después pide la recaudación y la escanea, tras lo cual la suma a su recaudación anterior. Por tanto este algoritmo tiene coste O(n), contando que comprueba *n* películas hasta encontrarla.

4.7 Método borrarActor

```
private static void borrarActor() {
/* El usuario introduce un nombre, y si ese actor está en el
mapa de actores será eliminado.
Postcondición: el actor introducido no estará en la lista.
*/
```

Casos de prueba:

- El mapa de actores no contiene ningún elemento
- Sólo contiene un elemento
    - El actor buscado es igual al del mapa
    - El actor buscado no es el del mapa
- Contiene multitud de elementos
    - El actor buscado no está en el mapa
    - El actor buscado está en el mapa

Implementación del algoritmo:

```
preguntar nombre del actor a borrar
guardar el nombre en un String
comprueba si el actor está en el mapa
Si esta
    borra el actor
    imprime un mensaje diciendo que lo ha borrado
si no está
    imprime un mensaje diciendo que no está
```

Coste: el algoritmo escribe por pantalla, guarda lo recibido en un String, lo busca en el mapa con una key y lo borra si está. El coste es constante O(1).

4.8 Método exportarLista

```
public static void exportarLista(){
/* Postcondición: guarda en un fichero linea a linea la
información de los registros en el mismo formato en el que
los ha cargado. */
```

Casos de prueba:

- Guardar de registros vacíos.
- Intentar guardar en ruta inexistente.
- guardar en un fichero con muchos elementos

Implementación del algoritmo:

```
intentamos
      escribir en el fichero ubicado de la ruta que hemos especificado
      por cada película del registro que haya
            escribimos en el fichero el nombre de la peli y los
            caracteres que la separen de los nombres de actores.
            por cada actor menos el ultimo de la lista de la pelicula
                  escribimos el nombre del actor
                  escribimos los caracteres que la separen los nombres de
                  actores
            escribimos el nombre del ultimo actor
            salta a una nueva línea
```

Coste: Teniendo en cuenta que los actores sean *n* y las películas *m* el coste sería O(n*m)

4.9 Método obtenerListaOrdenadaActores

```
private static void obtenerListaActoresOrdenada() {
/* Se ordena el registro de actores por el método quickSort
y se imprime por pantalla
Postcondición: Se verá por pantallas la lista de los actores
ordenada */
```

Casos de prueba:

- La lista está vacía
- La lista no está vacía:
    - Hay un elemento en la lista
    - Hay más de un elemento en la lista
    - Hay miles de elementos en la lista

Implementación del algoritmo mediante el método quickSort.

Coste: Como ya hemos visto en las clases teóricas este algoritmo tiene coste coste O(log n) para *n* elementos y un coste $O(n^2)$ si ya está ordenada.

5 Código

5.1 Clase Gestionator

```
package packPrincipal;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
import java.util.regex.Pattern;

import packActor.Actor;
import packActor.RegistroActores;
import packPeliculas.Pelicula;
import packPeliculas.RegistroPeliculas;

public class Gestionator {
    private Gestionator elGestionator;
    private static boolean ficheroCargado = false;
    private static String ruta;

    private Gestionator() {}

    public static void main(String[] args) {
        menu();
    }

    public static void obtenerRuta() {
        System.out.println("Inserte la ruta de su archivo (Recuerde que windows pone los slashes al revés)");
       System.out.println("Porejempo: C:/workspaceEclipse/EDA1/documentos/archivos/FilmsActors20162017Small.txt");
        System.out.println("Si introduce mal la ruta el programa no funcionará y tendrá que reiniciar la aplicación");
        Scanner sc = new Scanner(System.in);
        ruta = sc.nextLine();
    }
```

```
    private static void menu() {
        System.out.println("Seleccione operacion:");
        System.out.println();
        System.out.println(" 1. Cargar fichero ");
        System.out.println(" 2. Buscar un actor ");
        System.out.println(" 3. Insertar un actor ");
        System.out.println(" 4. Mostrar peliculas rodadas por un actor ");
        System.out.println(" 5. Mostrar actores de una pelicula ");
        System.out.println(" 6. Incrementar la recaudacion de una pelicula ");
        System.out.println(" 7. Borrar actor ");
        System.out.println(" 8. Exportar registro de peliculas a fichero ");
        System.out.println(" 9. Obtener lista ordenada de actores ");
        System.out.println();

System.out.println("------------------------------------------------------------");
        int seleccion;
        Scanner sc = new Scanner(System.in);
        seleccion=sc.nextInt();

        switch (seleccion) {

       case 1:
       if (!ficheroCargado){
                    obtenerRuta();
                    cargarLista();
                    System.out.println("Fichero cargado");
                    ficheroCargado = true;
       } else System.out.println("El fichero ya había sido cargado");
       break;
       case 2:
       buscarActor();
       break;
       case 3:
       insertarActor();
       break;
       case 4:
       escribirPeliculasActor();
       break;
       case 5:
       imprimirActoresPelicula();
       break;
       case 6:
       incrementarRecaudacion();
       break;
```

```
   case 7:
   borrarActor();
   break;
   case 8:
   obtenerRuta();
   exportarLista();
   break;
   case 9:
   obtenerListaOrdenadaActores();
   break;
   default:
   System.out.println("El número introducido no está en el rango");
   break;
   }
    menu();
}
public Gestionator getGestionator() {
    if (elGestionator == null) elGestionator = new Gestionator();
    return elGestionator;
}

public static void cargarLista(){
    StopWatch sw = new StopWatch();
    Pelicula peli;
    Actor act;
    RegistroActores regAct = RegistroActores.getRegistroActores();
    RegistroPeliculas regPeli = RegistroPeliculas.getRegistroPeliculas();
    String[] sepPeliDeActor, listaActores;
          try{
             BufferedReader buff = new BufferedReader(new FileReader(ruta));
          String linea = buff.readLine();
          Pattern patt1 = Pattern.compile("\\s+--->\\s+");
          Pattern patt2 = Pattern.compile("\\s+&&&\\s+");
```

```
		while (linea!=null) {
			sepPeliDeActor = patt1.split(linea);
			peli = new Pelicula(sepPeliDeActor[0]);
			regPeli.anadirPelicula(peli);
			listaActores = patt2.split(sepPeliDeActor[1]);
			for (int i = 0; i < listaActores.length; i++) {
				act = regAct.buscarActor(listaActores[i]);
				if (act==null) {
					act = new Actor(listaActores[i]);
					regAct.anadirActor(act);
				}
				peli.anadirActor(act);
				act.anadirPelicula(peli);
			}
			linea = buff.readLine();
		}
		buff.close();
		}
		catch (FileNotFoundException e){
			System.out.println("El fichero no ha sido encontrado");
		}
		catch(IOException e) {e.printStackTrace();}
	System.out.println(sw.elapsedTime());

	}

private static void exportarLista() {
	StopWatch sw = new StopWatch();
	RegistroPeliculas regPelis = RegistroPeliculas.getRegistroPeliculas();
	try {
		BufferedWriter bw = new BufferedWriter(new FileWriter(ruta));
		Pelicula peli;
		for (int i = 0; i < regPelis.getPelis().obtenerNumPeliculas(); i++) {
			peli = regPelis.getPelis().obtenerPeliEnPos(i);
			bw.write(peli.getNombre()+" ---> ");
			for (int j = 0; j < peli.getActores().size()-1; j++) {
				bw.write(peli.getActores().get(j).getNombre());
				bw.write(" &&& ");
			}
```

```
            bw.write(peli.getActores().get(peli.getActores().size()-1).getNombre());
                    bw.flush();
                    bw.newLine();
             }
             bw.close();
             }
        catch (IOException e) { e.printStackTrace();}
        System.out.println(sw.elapsedTime());
    }

    private static void buscarActor() {
        System.out.println("Introduce el nombre completo del actor");
        Scanner sc = new Scanner(System.in);
        String apeNom = sc.nextLine();
        StopWatch sw = new StopWatch();
        Actor a = RegistroActores.getRegistroActores().buscarActor(apeNom);
        if (a!=null) System.out.println("El actor ha sido encontrado");
        else System.out.println("El actor no ha sido encontrado");
        System.out.println(sw.elapsedTime());
    }

    private static void insertarActor() {
        System.out.println("Introduce el nombre completo del actor");
        Scanner sc = new Scanner(System.in);
        String nom = sc.nextLine();
        StopWatch sw = new StopWatch();
        Actor a = new Actor(nom);
        RegistroActores.getRegistroActores().anadirActor(a);
        System.out.println(sw.elapsedTime());
    }

    private static void borrarActor() {
        System.out.println("Introduce el nombre completo del actor");
        Scanner sc = new Scanner(System.in);
        String apeNom = sc.nextLine();
        StopWatch sw = new StopWatch();
        RegistroActores.getRegistroActores().eliminarActor(apeNom);
        System.out.println(sw.elapsedTime());
    }
```

```
    private static void escribirPeliculasActor() {
        System.out.println("Introduce el nombre completo del actor");
        Scanner sc = new Scanner(System.in);
        String apeNom = sc.nextLine();
        StopWatch sw = new StopWatch();
        Actor a = RegistroActores.getRegistroActores().buscarActor(apeNom);
        if (a!=null) a.imprimirPeliculas();
        else System.out.println("El actor no ha sido encontrado");
        System.out.println(sw.elapsedTime());
    }

    private static void imprimirActoresPelicula() {
        System.out.println("Introduce el nombre de la película");
        Scanner sc = new Scanner(System.in);
        String nombre = sc.nextLine();
        StopWatch sw = new StopWatch();
        Pelicula peli = RegistroPeliculas.getRegistroPeliculas().buscarPelicula(nombre);
        if (peli!=null) peli.imprimirseCompleto();
        else System.out.println("La pelicula no ha sido encontrada");
        System.out.println(sw.elapsedTime());
    }

    private static void incrementarRecaudacion() {
        System.out.println("Introduce el nombre de la película");
        Scanner sc = new Scanner(System.in);
        String nombre = sc.nextLine();
        StopWatch sw = new StopWatch();
        Pelicula peli = RegistroPeliculas.getRegistroPeliculas().buscarPelicula(nombre);
        System.out.println(sw.elapsedTime());
        if (peli!=null) {
               System.out.println("Introduce la recaudación a incrementar");
               int recau = sc.nextInt();
               peli.incrementarRecaudacion(recau);
        }
        else System.out.println("La película no ha sido encontrada");
    }
    private static void obtenerListaOrdenadaActores() {
        StopWatch sw = new StopWatch();
        RegistroActores.getRegistroActores().ListaOrdenadaActores();
        System.out.println(sw.elapsedTime());
    }

}
```

5.2 Clase Pelicula

```java
package packPeliculas;

import java.util.ArrayList;
import java.util.Iterator;

import packActor.Actor;

public class Pelicula {
    private String nombre;
    private ArrayList<Actor> actores;
    private int recaudacion;

    public Pelicula(String pNombre) {
        this.nombre=pNombre;
        this.recaudacion=0;
        this.actores = new ArrayList<Actor>();
    }

    public String getNombre() {return this.nombre;}
    public ArrayList<Actor> getActores() {return this.actores;}
    public void incrementarRecaudacion(int pRec) {
        this.recaudacion = this.recaudacion + pRec;
        System.out.println("Recaudacion incrementada a "+this.recaudacion+"€");
    }

    public void anadirActor (Actor a) {
        if (!actores.contains(a))
              actores.add(a);
    }
    public void imprimirse() {
        System.out.println("Nombre de la película: " + this.nombre);
    }

    public void imprimirseCompleto() {
        System.out.println("Nombre de la película: " + this.nombre);
        System.out.println();
        System.out.println("Recaudación de la película: " + this.recaudacion+"€");
        System.out.println();
        System.out.println("Actores del reparto: ");
        imprimirActores();
   }
```

```
    private Iterator<Actor> getIterador() {return actores.iterator();}

    public void imprimirActores() {
        Actor a;
        Iterator<Actor> itr = this.getIterador();
        while (itr.hasNext()) {
               a = itr.next();
               a.imprimirse();
        }
    }

    public int compareTo(Pelicula peli) {
        return this.nombre.compareTo(peli.nombre);
    }
}
```

5.3 Clase RegistroPeliculas

```
package packPeliculas;

public class RegistroPeliculas {
    private ListaPeliculas pelis;
    private static RegistroPeliculas elRegistroPeliculas;

    private RegistroPeliculas() {
        this.pelis = new ListaPeliculas();
    }

    public static RegistroPeliculas getRegistroPeliculas() {
        if (elRegistroPeliculas == null) elRegistroPeliculas = new RegistroPeliculas();
        return elRegistroPeliculas;
    }

    public Pelicula buscarPelicula(String pNombre) {
        return pelis.buscarPelicula(pNombre);
    }

    public void anadirPelicula (Pelicula pPeli) {
        pelis.anadirPelicula(pPeli);
    }
    public ListaPeliculas getPelis() {
        return pelis;
    }

}
```

5.4 Clase ListaPeliculas

```
package packPeliculas;

import java.util.ArrayList;
import java.util.Iterator;

public class ListaPeliculas {
    private ArrayList<Pelicula> lista;

    public ListaPeliculas() {
        this.lista = new ArrayList<Pelicula>();
    }
    private Iterator<Pelicula> getIterador() {return lista.iterator();}
    public Pelicula buscarPelicula(String pNombre) {
        Pelicula p = null;
        Boolean esta = false;
        Iterator<Pelicula> itr = this.getIterador();
        while (itr.hasNext() && !esta) {
               p = itr.next();
               if (p.getNombre().equals(pNombre))
                      esta=true;
        }
        if (esta) return p;
        else return null;
    }
    public boolean esta(Pelicula pPeli) {
        return lista.contains(pPeli);
    }
    public void anadirPelicula(Pelicula pPeli) {
        lista.add(pPeli);
    }
    public void imprimirPeliculas() {
        Pelicula p;
        Iterator<Pelicula> itr = this.getIterador();
        while (itr.hasNext()) {
               p = itr.next();
               p.imprimirse();
        }
    }
    public int obtenerNumPeliculas() {return lista.size();}
    public Pelicula obtenerPeliEnPos(int pPos) {return lista.get(pPos);}
    }
```

5.5 Clase Actor

```java
package packActor;

import packPeliculas.ListaPeliculas;
import packPeliculas.Pelicula;

public class Actor implements Comparable<Actor> {
    private String nombre;
    private ListaPeliculas susPeliculas;

    public Actor(String pNom) {
        this.nombre = pNom;
        this.susPeliculas = new ListaPeliculas();
    }

    public String getNombre() {
        return nombre;
    }


    public void anadirPelicula(Pelicula p) {
        if ((p != null) && (!susPeliculas.esta(p))) {
                susPeliculas.anadirPelicula(p);
        }
    }

    public ListaPeliculas devolverPeliculas() {
        return susPeliculas;
    }

    public void imprimirse() {
        System.out.println(this.nombre);
    }

    public void imprimirPeliculas() {
        this.imprimirse();
        if (susPeliculas.obtenerNumPeliculas() > 0) {
                System.out.println("Ha aparecido en las siguientes películas");
                this.susPeliculas.imprimirPeliculas();
        } else
                System.out.println("No ha aparecido en ninguna película");
    }
```

```
    @Override
    public int compareTo(Actor a) {
        return this.nombre.compareTo(a.nombre);
    }
}
```

5.6 Clase RegistroActores

```
package packActor;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Iterator;

public class RegistroActores {
    private Map<String, Actor> mapaActores;
    private static RegistroActores elRegistroActores;

    private RegistroActores() {
        mapaActores = new HashMap<>();
    }
    public static RegistroActores getRegistroActores() {
        if (elRegistroActores == null)
               elRegistroActores = new RegistroActores();
        return elRegistroActores;
    }
    public Actor buscarActor(String pApeNom) {
        return mapaActores.get(pApeNom);
    }

    public void anadirActor(Actor a) {
        if (!mapaActores.containsKey(a.getNombre())){
               mapaActores.put(a.getNombre(), a);
        }
    }

    public void eliminarActor(String pApeNom) {
        if(mapaActores.containsKey(pApeNom)) {
               mapaActores.remove(pApeNom);
               System.out.println("El actor ha sido eliminado con exito");
        }
        else System.out.println("El actor no se encontraba en la lista");
    }
    public void ListaOrdenadaActores() {
        ArrayList<Actor> lista = new ArrayList<Actor>(mapaActores.values());
        quickSort(lista, 0, lista.size() - 1);
        imprimirActores(lista);
    }
```

```
    private void quickSort(ArrayList<Actor> pLista, int pInicio, int pFinal) {
        if (pFinal - pInicio > 0) {
               int indiceParticion = particion(pLista, pInicio, pFinal);
               quickSort(pLista, pInicio, indiceParticion - 1);
               quickSort(pLista, indiceParticion + 1, pFinal);
        }
    }
    private int particion(ArrayList<Actor> pLista, int pInicio, int pFinal) {
        Actor pivote = pLista.get(pInicio);
        int izq = pInicio;
        int der = pFinal;
        while (izq < der) {
               while (pLista.get(izq).compareTo(pivote) <= 0 && izq < der)
                      izq++;
               while (pLista.get(der).compareTo(pivote) > 0)
                      der--;
               if (izq < der)
                      swap(pLista, izq, der);
        }
        pLista.set(pInicio, pLista.get(der));
        pLista.set(der, pivote);
        return der;
    }

    private void swap(ArrayList<Actor> pLista, int pIzq, int pDer) {
        Actor aux = pLista.get(pIzq);
        pLista.set(pIzq, pLista.get(pDer));
        pLista.set(pDer, aux);
    }

    public void imprimirActores(ArrayList<Actor> pLista) {
        Iterator<Actor> itr = pLista.iterator();
        while (itr.hasNext()) {
               itr.next().imprimirse();
        }
    }
}
```

5.7 Clase StopWatch

```
package packPrincipal;

public class StopWatch {

       private final long start;

   /** Create a stopwatch object. */
       public StopWatch() {
       start = System.currentTimeMillis();
       }

   /**
       * Return elapsed time (in seconds) since this object was created.
       */
       public long elapsedTime() {
       long now = System.currentTimeMillis();
       return (now - start);
       }
       public void imprimirTiempo(){
        System.out.println(start);
       }
}
```

6 Conclusiones

Para la resolución de este ejercicio hemos realizado un programa de utilidad completa, es decir, que al iniciar nuestra aplicación se mostrará un menú para que el usuario pueda escoger la opción que desea utilizar. Esto nos ha servido para realizar todas las pruebas necesarias a la hora de comprobar la validez nuestros métodos.

El control sobre una cantidad enorme de datos ha sido bastante fácil gracias a las herramientas utilizadas, como el HashMap o el método de ordenación quickSort de coste logarítmico.

Lo esencial para este trabajo ha sido la lectura del fichero, que ha sido nuestra principal dificultad. Hasta ahora no habíamos necesitado trabajar con ellos más allá de simples líneas, y la utilización del split y los patrones con expresiones regulares ha sido lo más costoso.

Los tiempos obtenidos en la ejecución de los diferentes métodos han sido bastante satisfactorios, tardando escasos diez segundos en cargar todo el fichero grande de actores y películas y un tiempo insignificante en el resto.