Estructura de Datos y Algoritmos. Conv Extraordinaria, fase 3. EHU-UPV

Asier Aguilar, Marcos Chouciño, Iker García López

9 de enero de 2023

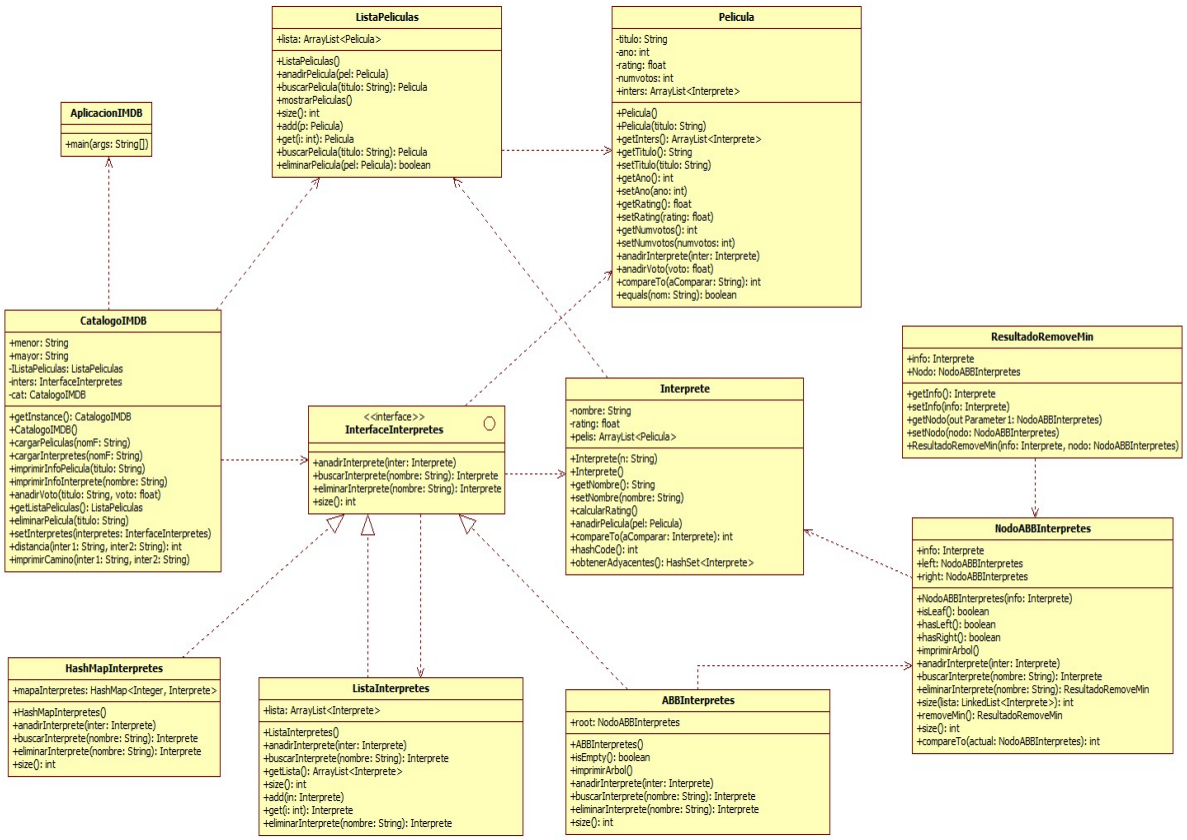
Índice general

Capítulo 1

Diagrama de clases.

1.1. UML

A continuación se muestra un esbozo del diagrama que representa nuestro escenario:



Capítulo 2

Métodos implementados.

Explicaremos cada método utilizado en cada clase de la fase 3 del proyecto y el orden temporal de cada uno.

En la tercera fase del proyecto nos hemos centrado en las estructuras tablas hash y grafos.

2.1. AplicacionWeb

Es la clase main del proyecto. Hemos terminado el menú añadiendo implementaciones de la tercera parte y hemos puesto más opciones para que el usuario pueda elegir a la hora de ejecutar el programa.

- Algoritmo:

```
public class AplicacionIMDB {

public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);
  String auxi;
  String auxi1;
  String auxi2;
  float auxi3;

  // TO DO: ...
  CatalogoIMDB catalogo;
  HashMapInterpretes mapadeInterpretes = new HashMapInterpretes()
 ;

  System.out.println("Bienvenid@ a la aplicaci n IMDB");
  catalogo = CatalogoIMDB.getInstance(mapadeInterpretes);
  // TO DO: Cargar pel culas
  catalogo.cargarPeliculas("src\\Files\\smallerfiles\\films_tiny.
 txt");
  // catalogo.mostrarPeliculas();
  // TO DO Cargar int rpretes
  catalogo.cargarInterpretes("src\\Files\\smallerfiles\\cast_tiny
 .txt");
  // ABBInterpretes mapadeInterpretes = new ABBInterpretes() =
  // catalogo.cargarInterpretes("src\\Files\\smallerfiles\\
 cast_tiny.txt");
  // Men
  try {
    int opcion = -1;
    while (opcion != 0) {
      System.out.println("\n\nEscoja una opci n:");
      System.out.println("1. Mostrar informaci n de pel cula");
      System.out.println("2. Mostrar informaci n de int rprete
 ");
      System.out.println("3. A adir voto a pel cula");
      System.out.println("4. Eliminar pel cula del cat logo");
      System.out.println("5. Calcular distancia entre interpretes
 ");
      System.out.println("6. Mostrar camino entre interpretes");
      System.out.println("0. Salir");

      opcion = Integer.parseInt(sc.nextLine());

      switch (opcion) {
      case 1:
        try {
```

```
          System.out.println("\nIntroduzca el nombre de la
pel cula");
          auxi = sc.nextLine();
          System.out.println("\n");
          catalogo.imprimirInfoPelicula(auxi);
        } catch (java.lang.NullPointerException e) {
          System.out.println("No se ha encontrado esa pel cula,
int ntelo de nuevo\n");
        }
        break;

      case 2:
        try {
          System.out.println("Introduzca el nombre del
int rprete");
          auxi = sc.nextLine();
          System.out.println("\n");
          catalogo.imprimirInfoInterprete(auxi);
        } catch (java.lang.NullPointerException e) {
          System.out.println("No se ha encontrado ese int rprete
, int ntelo de nuevo\\n");
        }
        break;

      case 3:
        try {
          System.out.println("Introduzca el nombre de la pelicula
 que quiere valorar");
          auxi = sc.nextLine();
          System.out.println("Introduzca una puntuaci n entre
0.0 y 10.0");
          auxi3 = Float.valueOf(sc.nextLine());
          if (auxi3 < 0.0 || auxi3 > 10.0) {
            System.out.println("Ese valor no esta admitido, por
favor intrduzca un valor valido\n");
            break;
          }
          catalogo.anadirVoto(auxi, auxi3);
          Pelicula pel = catalogo.getListaPeliculas().
buscarPelicula(auxi);
          System.out.println("El nuevo rating de la pel cula es:
 " + pel.getRating());
          System.out.println("\n");
          break;
        } catch (java.lang.NullPointerException e) {
          System.out.println("No se ha encontrado esa pelicula,
intentelo de nuevo\n");
          break;
        }

      case 4:
        try {
          System.out.println("Introduzca el nombre de la
pel cula que quiere eliminar:");
          auxi = sc.nextLine();
```

```
            catalogo.eliminarPelicula(auxi);
            break;
          } catch (java.lang.NullPointerException e) {
            System.out.println("No se ha encontrado esa pelicula,
   intentelo de nuevo\n");
            break;
          }
        case 5:
          try {
            System.out.println("Introduzca los nombres de los
   interpretes para calcular su distancia:");
            auxi1 = sc.nextLine();
            auxi2 = sc.nextLine();
            System.out.println(catalogo.distancia(auxi1, auxi2));
            break;
          } catch (java.lang.NullPointerException e) {
            System.out.println("No se ha encontrado esa pelicula,
   intentelo de nuevo\n");
            break;
          }
        case 6:
          try {
            System.out.println("Introduzca los nombres de los
   interpretes para mostrar los interpretes mediante los que se
   relacionan:");
            auxi1 = sc.nextLine();
            auxi2 = sc.nextLine();
            catalogo.imprimirCamino(auxi1, auxi2);
            break;
          } catch (java.lang.NullPointerException e) {
            System.out.println("No se ha encontrado esa pelicula,
   intentelo de nuevo\n");
            break;
          }
        }
      }
    } catch (java.lang.NumberFormatException e) {
      System.out.println("T rmino incorrecto, cerrando programa \n
   ...");
    }
    sc.close();
  }
}
```

2.2. HashMapInterpretes

Hemos creado una clase para representar los intérpretes en una tabla hash y hemos implementado los métodos necesarios. En ella tendremos un atributo mapaInterpretes del tipo HashMap¡Integer, Interprete¿.

HashMapInterpretes

- Algoritmo:

```
public HashMapInterpretes() {
  super();
  this.mapaInterpretes = new HashMap<Integer, Interprete>();
}
```

Listing 2.1: Constructora.

Es la constructora de la clase.

- Casos de prueba considerados:
- Orden temporal:
  El orden temporal es 0(1).

anadirInterprete

- Algoritmo:

```
@Override
public void anadirInterprete(Interprete inter) {
  // System.out.print("se ha cargado 1 interprete");
  // AQUI NO VA NOMBRE?
  mapaInterpretes.put(inter.hashCode(), inter);
}
```

Listing 2.2: Añade un intérprete al mapa.

Dado un intérprete por parámetro, lo añade al mapa.

- Casos de prueba considerados:

Entrada	Función
Clint Eastwood	Intérprete añadido.

- Orden temporal:
  El orden temporal es 0(1).

2.3. buscarInterprete

- Algoritmo:

```
@Override
public Interprete buscarInterprete(String nombre) {
  return mapaInterpretes.get(nombre.hashCode());
}
```

Listing 2.3: Busca un intérprete en el mapa.

Dado el nombre de un intérprete por parámetro en forma de String, lo busca en el mapa y la devuelve.

Casos de prueba considerados:

Entrada	Función
Clint Eastwood	Econtrado.

Orden temporal:
El orden temporal es 0(1).

2.4. eliminarInterprete

- Algoritmo:

```
@Override
public Interprete eliminarInterprete(String nombre) {
  return mapaInterpretes.remove(nombre.hashCode());
}
```

Listing 2.4: Elimina un intérprete en el mapa.

Dado el nombre de un intérprete por parámetro en forma de String, lo elimina del mapa.

Casos de prueba considerados:

Entrada	Función
Clint Eastwood	Eliminado.

Orden temporal:
El orden temporal es 0(1).

2.5. size

- Algoritmo:

```
@Override
public int size() {
  return mapaInterpretes.size();
}
```

Listing 2.5: Devuelve el tamaño del mapa.

Devuelve el tamaño del mapa.

Casos de prueba considerados:

Entrada	Función
	Tamaño: 12.

Orden temporal:
El orden temporal es 0(1).

2.6. Interprete

Hemos añadido tres métodos nuevos.

hashCode

- Algoritmo:

```
public int hashCode() {
  return nombre.hashCode();
}
```

Listing 2.6: Devuelve el hashCode del intérprete.

Devuelve el hashCode del intérprete.

- Casos de prueba considerados:

Entrada	Función
	4.

- Orden temporal:
  El orden temporal es 0(1).

obtenerAdyacentes

- Algoritmo:

```
/**
* Devuelve un HashSet con todos los adyacentes del int rprete,
 es decir,
* aquellos int rpretes con los que ha participado en alguna
 pel cula.
* @return: el HashSet con los int rpretes que son adyacentes.
*/
public HashSet<Interprete> obtenerAdyacentes(){
  HashSet<Interprete> auxi = new HashSet<Interprete>();
  for(int i = 0; i<pelis.size();i++) {
```

```
    ListaInterpretes listaInterpretesDeCadaPelicula = pelis.get(i
 ).getInters();
    for(int j = 0; j < listaInterpretesDeCadaPelicula.size(); j
 ++) {
      auxi.add(listaInterpretesDeCadaPelicula.get(j));
    }
  }
  return auxi;
}
```

Listing 2.7: Devuelve todos los adyacentes del intérprete.

Devuelve un HashSet con todos los adyacentes (intérpretes con los que compartió alguna película).

- Casos de prueba considerados:

Entrada	Función
	HashSet.

- Orden temporal:
  El orden temporal es 0(n+m) siendo n el tamaño de la lista de películas y m el tamaño de la lista de intérpretes de cada película.

2.7. CatalogoIMDB

Hemos añadido dos nuevos métodos.

distancia

- Algoritmo:

```
/**
* Devuelve la distancia m nima entre dos int rpretes dados.
* @param inter1: nombre del primer int rprete
* @param inter2: nombre del segundo int rprete
* @return: distancia m nima entre ambos int rpretes. En caso de
   que no
* est n conectados, devuelve -1.
* O(N+A)
*/
public int distancia(String inter1, String inter2){

  Interprete origen = inters.buscarInterprete(inter1);
  HashMap<Interprete, Integer> visitados = new HashMap<Interprete
 , Integer>();
  Queue<Interprete> cola = new LinkedList<Interprete>();
  cola.add(origen);
```

```
    visitados.put(origen, 0);
    boolean enc = false;
    Interprete inter = null;

    while(!cola.isEmpty() && !enc) {
      inter = cola.remove();
      if(inter.getNombre().equals(inter2)) {
        enc = true;
      }
      else {
        for(Interprete aux: inter.obtenerAdyacentes()) {
          if(!visitados.containsKey(aux)) {
            cola.add(aux);
            visitados.put(aux, visitados.get(inter)+1);
          }
        }
      }
    }
    if(enc) {
      return visitados.get(inter);
    }
    else return -1;
    }
```

Listing 2.8: Devuelve la distancia más corta entre dos intérpretes.

Dados por parámetro los nombres de dos intérpretes en forma de String, devuelve la distancia mínima entre ambos intérpretes. En caso de que no estén conectados, devuelve -1.

- Casos de prueba considerados:

Entrada	Función
Clint Eastwood, Antonio Banderas	-1

- Orden temporal:
  El orden temporal es 0(N+A). Puesto que en el peor de los casos se visitan todos los nodos y todas sus aristas.

imprimirCamino

- Algoritmo:

```
/**
* Imprime el camino m s corto entre dos int rpretes. Si no
 existe camino,
* imprime un mensaje indicando este hecho.
* @param inter1: nombre del primer int rprete
* @param inter2: nombre del segundo int rprete
* O(N+A)
*/
```

```
public void imprimirCamino(String inter1, String inter2) {

  Interprete origen = inters.buscarInterprete(inter1);
  LinkedList<String> resultado = new LinkedList<String>();
  HashMap<Interprete, Interprete> visitados = new HashMap<
 Interprete, Interprete>();
  Queue<Interprete> cola = new LinkedList<Interprete>();
  cola.add(origen);
  visitados.put(origen, null);
  boolean encontrado = false;
  while(!cola.isEmpty() && !encontrado) {
    Interprete inter = cola.remove();
    if(inter.getNombre().equals(inter2)) {
      encontrado = true;
    }
    else {
      for(Interprete aux:inter.obtenerAdyacentes()) {
        if(!visitados.containsKey(aux)) {
          cola.add(aux);
          visitados.put(aux, inter);//a aux hemos llegado desde
 calle
        }
      }
    }
  }
  Interprete destino = inters.buscarInterprete(inter2);
  if(encontrado) {
    Interprete actual = destino;
    while(actual!=null) {
      resultado.addFirst(actual.getNombre());
      actual = visitados.get(actual);
    }
    for(String resul: resultado) {
      System.out.println(resul);
    }
  }
  else {
    System.out.println("No hay camino que conecte esos dos
 interpretes");
  }
}
```

Listing 2.9: Imprime el camino más corto entre dos intérpretes.

Dados por parámetro los nombres de dos intérpretes en forma de String, imprime el camino más corto entre ambos intérpretes. En caso de que no estén conectados, imprime un mensaje diciendo que no están conectados.

- Casos de prueba considerados:

Entrada	Función
Clint Eastwood, Antonio Banderas	"No hay camino que conecte esos dos interpretes"

- Orden temporal:
  El orden temporal es 0(N+A). Puesto que en el peor de los casos se visitan todos los nodos y todas sus aristas.

Capítulo 3

Pregunta e

La pregunta e nos plantea lo siguiente: ¿Dónde más habría sido adecuado utilizar una tabla hash? ¿Qué beneficio se habría obtenido con ello?

3.1. Respuesta

Hubiera sido beneficioso haberlo implementado en la clase Pelicula, puesto que el funcionamiento de la misma es similar al de la clase Interprete, con ello hubiéramos mejorado el rendimiento del proyecto.