Laboratorio 4
EDA

Fecha: 09-I-2015
Galder Revilla
Javier Jiménez Cidre
Begoña Martinez de Marañon

Tabla de contenido

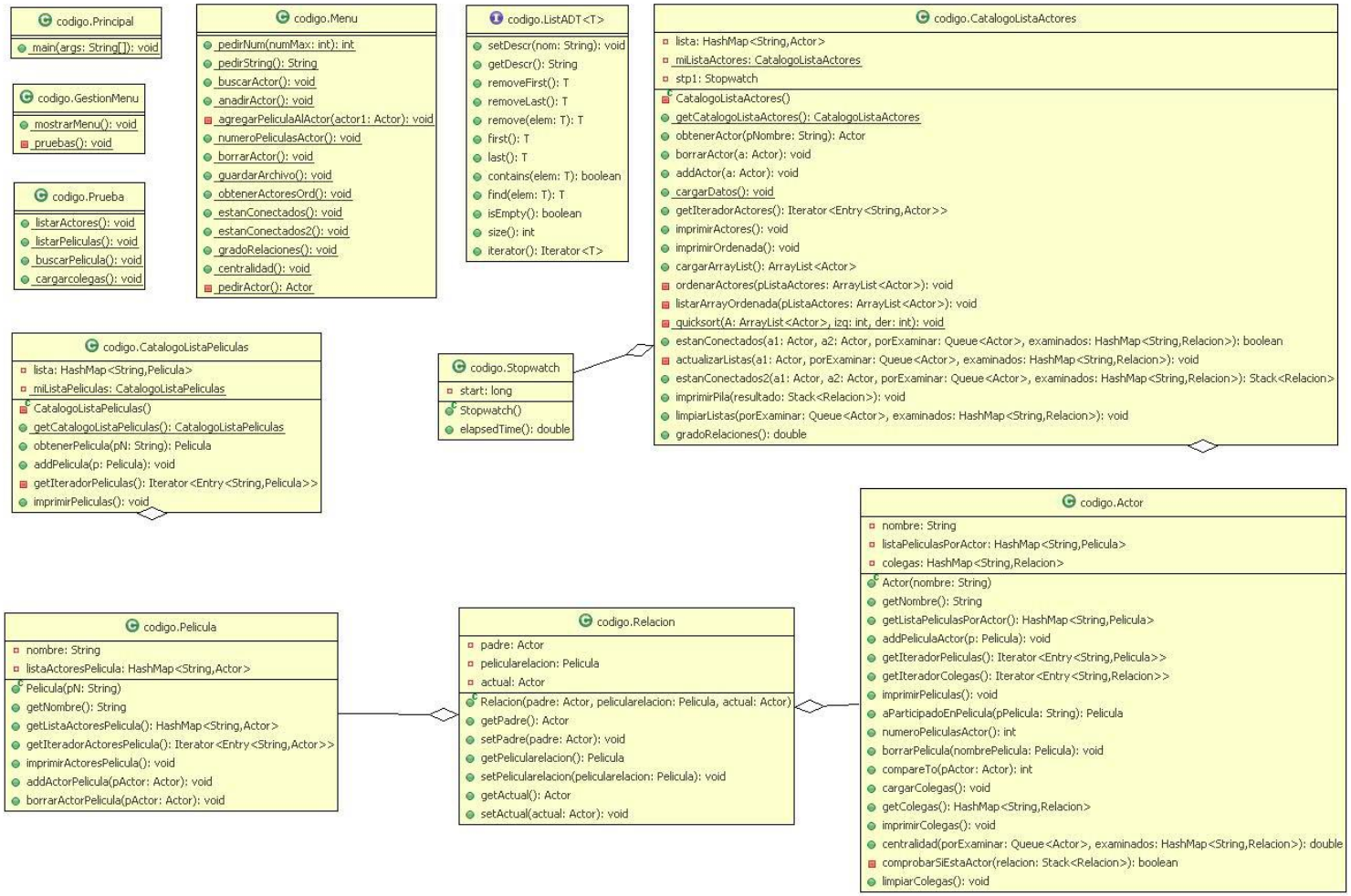
1. Introducción

En este laboratorio se pide implementar dos nuevas funcionalidades respecto a los anteriores.

La primera funcionalidad consiste en calcular un valor que dará el número medio correspondiente a la distancia del camino que une a cualquier par de actores, según una serie de relaciones.

La segunda funcionalidad consiste en calcular un resultado que es una media de la centralidad de un actor que no es más que una estimación con un número suficiente de pares.

2. Diseño de las clases



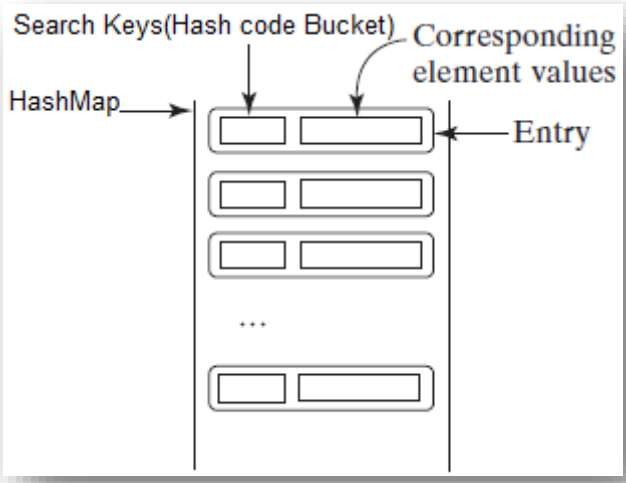
3. Descripción de las estructuras de datos principales

Las estructuras principales que se ha utilizado en esta práctica son “HashMap”, “cola” y “pila”

3.1. HashMap

Permiten guardar pares de clave y valor. Se utiliza la clave para buscar, acceder o recuperar valores, entre otras operaciones. No soportan claves duplicadas, aunque sí valores.

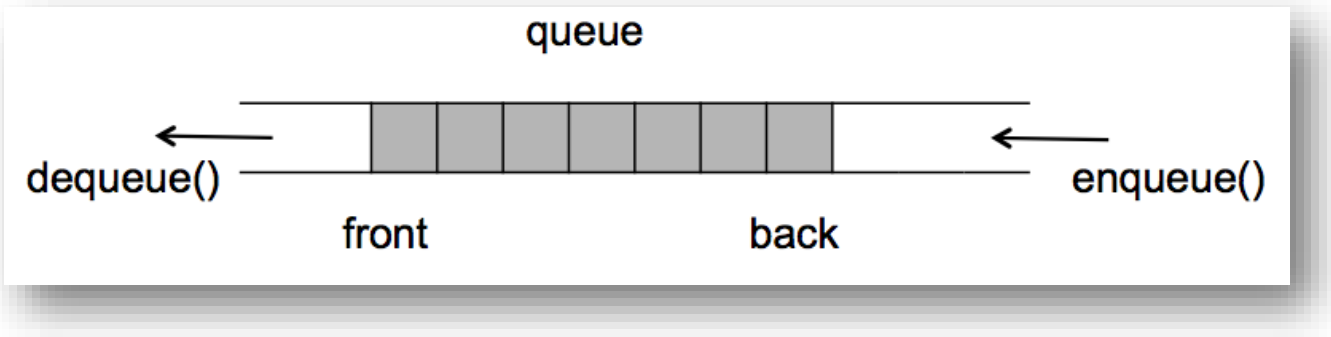
Hemos implementado esta estructura para guardar los actores examinados con sus relaciones y para guardar los compañeros de reparto de un actor (colegas). Así nos permitirá acceder inmediatamente a un actor en concreto y su relación con la película.



3.2. Cola

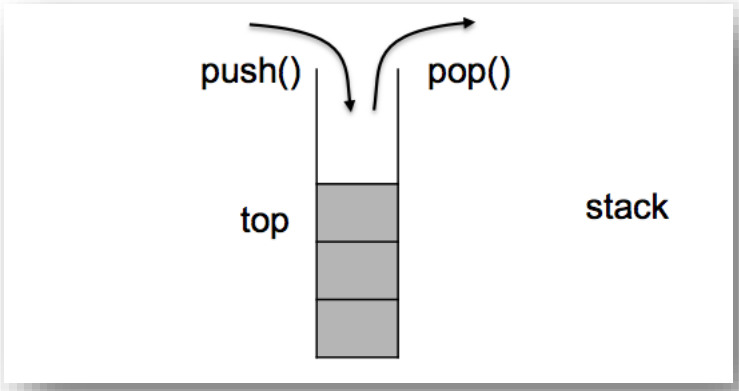
Es una estructura de datos, caracterizada por ser una secuencia de elementos en la que la operación de inserción push se realiza por un extremo y la operación de extracción pop por el otro. También se le llama estructura FIFO debido a que el primer elemento en entrar será también el primero en salir.

Se ha implementado esta estructura para guardar los actores que se van a examinar e ir eliminándolos automáticamente uno a uno según se vayan examinando para guardarlos en la estructura “examinados”.



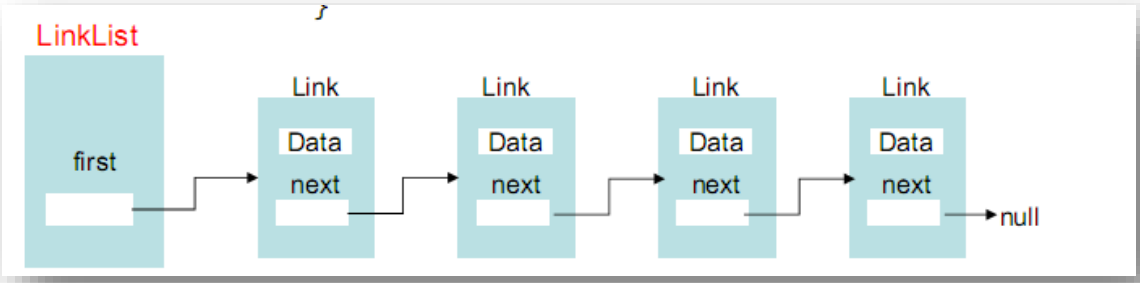
3.3. Pila

Es una lista ordenada o estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO debido a que el último elemento en entrar será el primero en salir. Este tipo de estructura permite almacenar y recuperar datos. En todo momento sólo se tiene acceso a la parte superior de la pila, es decir, al último objeto. La operación "pop" permite la obtención de este elemento, que es retirado de la pila permitiendo el acceso al siguiente. Se ha implementado esta estructura en el método "centralidad" para guardar en un resultado de tipo "Relacion" todos los actores conectados entre dos actores dados.



3.4. LinkedList

Una estructura enlazada es una estructura de datos que utiliza variables de referencia a objetos con el fin de crear enlaces entre objetos Generalmente, se diseña una clase de nodos. En el caso del laboratorio la hemos usado para implementar una cola.



4. Diseño e implementación de los métodos principales

ACTOR

4.1. Método limpiarColegas

Descripción

Limpia el hashMap de colegas de cada actor para liberar carga a la hora de usar el método "estanConectados".

Implementación del algoritmo:

```
public void limpiarColegas(){
		this.colegas.clear();
	}
```

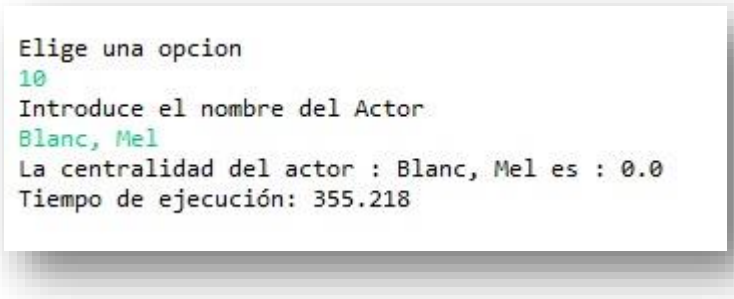
Coste: O(1) , coste lineal

4.2. Método centralidad

Descripción

Calcula una media de la centralidad de un actor dado. No se calculan todos los pares porque sería inviable, por tanto se usan 50 pares de actores escogidos al azar.

Casos de prueba con archivo de 40.000



Implementación del algoritmo:

```
public double centralidad(Queue<Actor> porExaminar,
			HashMap<String, Relacion> examinados) {
		double centralidad = 0.0;

		ArrayList<Actor> listaArray = CatalogoListaActores
				.getCatalogoListaActores().cargarArrayList();

		for (int i = 0; i < 50; i++) {
			Random rand = new Random();
			int randomNum1 = rand.nextInt((listaArray.size() - 1) + 1) + 0;
			int randomNum2 = rand.nextInt((listaArray.size() - 1) + 1) + 0;

			Actor a1 = listaArray.get(randomNum1);
			Actor a2 = listaArray.get(randomNum2);
```

```
                try {
                        Stack<Relacion> resultado = CatalogoListaActores
                                        .getCatalogoListaActores().estanConectados2(a1,
a2,
                                                        porExaminar, examinados);
                        CatalogoListaActores.getCatalogoListaActores().limpiarLis-
tas(porExaminar, examinados);
                        if (resultado != null) {
                                if (comprobarSiEstaActor(resultado)) {
                                        centralidad++;
                                }
                        }
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        return centralidad;
    }
```

Coste:O(n), coste constante.

Donde "n" es el número de elementos que contiene la estructura "HashMap""examinados" en el método “estanConectados2”. En el peor de los casos se recorrerá toda la lista.

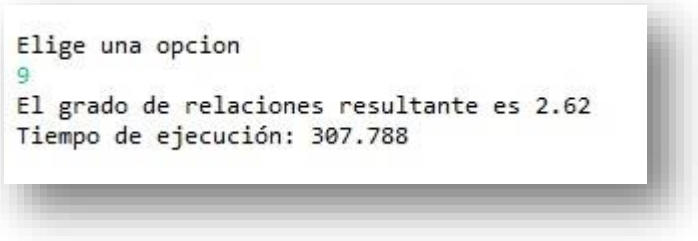
CATALOGOLISTAACTORES

4.3. Método gradoRelaciones

Descripción

Calcula el número medio correspondiente a la distancia del camino que une cualquier par de actores.

Casos de prueba



Implementación del algoritmo:

```
public double gradoRelaciones() {
		Queue<Actor> porExaminar = (Queue<Actor>) new LinkedList<Actor>();
		HashMap<String, Relacion> examinados = new HashMap<String, Relacion>();
		Actor a1 = null;
		Actor a2 = null;
		int numeroPruebas = 50;
		double media = 0.0;
		Stack<Relacion> resultado = new Stack<Relacion>();

		ArrayList<Actor> listaArray = CatalogoListaActores
				.getCatalogoListaActores().cargarArrayList();

		for (int i = 0; i < numeroPruebas - 1; i++) {
			Random rand = new Random();
			int randomNum1 = rand.nextInt((listaArray.size() - 1) + 1) + 0;
			int randomNum2 = rand.nextInt((listaArray.size() - 1) + 1) + 0;
			a1 = listaArray.get(randomNum1);
			a2 = listaArray.get(randomNum2);
			try {
				resultado = estanConectados2(a1, a2, porExaminar, examina-
dos);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
			limpiarListas(porExaminar, examinados);

			if (resultado != null) {
				media += resultado.size();
			}
		}
		return media / numeroPruebas;
	}}
```

Coste:O(n), coste constante.

Donde "n" es el número de elementos que contiene la estructura "HashMap""examinados" en el método “estanConectados2”. En el peor de los casos se recorrerá toda la lista.

4.4. Método comprobarSiEstaActor

Descripción

Devuelve un boolean si el actor existe en la “pila” resultante de hacer “estanConectados2”.

Implementación del algoritmo:

```
private boolean comprobarSiEstaActor(Stack<Relacion> relacion) {
		Relacion r = null;
		while (!relacion.isEmpty()) {
			r = relacion.pop();
```

```
                    if (r.getActual().getNombre().equalsIgnoreCase(this.nombre)) {
                          return true;
                    }
              }
              return false;
        }
```

Coste: O(n), coste constante.

Donde “n” es el número máximo de relaciones. En el peor de los casos recorrerá todas las relaciones.

5. Código

5.1. Clase Menu

```
public class Menu {

	// Comprueba que el dato introducido es un numero del 0 al número máximo con 3
	// intentos máximos
	public static int pedirNum(int numMax) {
		int eleccion = -1;
		int contadorErrores = 0;
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		while (eleccion < 0 || eleccion > numMax) {
			try {
				System.out.println("Elige una opcion");
				String linea = br.readLine();
				eleccion = Integer.parseInt(linea);
			} catch (IOException e) {
				e.printStackTrace();
			} catch (NumberFormatException ex) {
				System.out.println("Debes introducir un numero del 0 al "
						+ numMax);
			}
			contadorErrores++;
			if (contadorErrores == 3) {
				return -1;
			}
		}
		return eleccion;
	}

	// Comprueba que introduce un String
	public static String pedirString() {
		String nombre = null;
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		try {
			nombre = br.readLine();
		} catch (IOException e) {
			e.printStackTrace();
		}
		return nombre;
	}

	// # 1
	public static void buscarActor() {
		String nombreActor;
		Actor actor1 = null;
		System.out.println("Introduce el nombre del Actor ");
		nombreActor = pedirString();
		actor1 = CatalogoListaActores.getCatalogoListaActores().obtenerActor(
				nombreActor);
		if (actor1 == null) {
			System.out.println("No existe el actor \n");
		} else {
			System.out.println("El actor " + nombreActor
					+ " tiene las siguientes peliculas:");
			actor1.imprimirPeliculas();
		}
	}
```

```
	// #2
	public static void anadirActor() {
		String nombreActor;
		Actor actor1 = null;
		System.out.println("Introduce el Actor a anadir: ");
		nombreActor = pedirString();
		// Si el actor a añadir ya existe no hacemos nada
		if (CatalogoListaActores.getCatalogoListaActores().obtenerActor(
				nombreActor) != null) {
			System.out.println("Ya existe el actor");
		} else {
			actor1 = new Actor(nombreActor);
			boolean resp = false;
			String respuesta = null;
			CatalogoListaActores.getCatalogoListaActores().addActor(actor1);
			System.out.println("Actor agregado");

			// Pedir si quiere añadir peliculas
			while (!resp) {
				System.out.println("Quieres agregarle alguna pelicula? S/N");
				respuesta = pedirString();
				if (respuesta.equalsIgnoreCase("S")) {
					agregarPeliculaAlActor(actor1);
				} else if (respuesta.equalsIgnoreCase("N")) {
					resp = true;
				}
			}
		}
	}

	// # 2.1 Cuando añadimos peliculas al actor creado
	private static void agregarPeliculaAlActor(Actor actor1) {
		String nombrePelicula = "";
		Pelicula pelicula1 = null;
		System.out.println("Introduce el titulo de la pelicula ");
		nombrePelicula = pedirString();
		// Comprobamos si existe la pelicula en el catalogo sino habrá
		// que crearla
		pelicula1 = CatalogoListaPeliculas.getCatalogoListaPeliculas()
				.obtenerPelicula(nombrePelicula);
		if (pelicula1 == null) {
			// si no existe es una nueva pelicula
			pelicula1 = new Pelicula(nombrePelicula);
			CatalogoListaPeliculas.getCatalogoListaPeliculas().addPelicula(
					pelicula1);
		}
		actor1.addPeliculaActor(pelicula1);
		pelicula1.addActorPelicula(actor1);
	}

	// # 3
	public static void numeroPeliculasActor() {
		String nombreActor = "";
		int numPeliculas = 0;
		Actor actor1 = null;
		System.out.println("Introduce nombre del actor: ");
		nombreActor = pedirString();
		actor1 = CatalogoListaActores.getCatalogoListaActores().obtenerActor(
				nombreActor);
		if (actor1 != null) {
			numPeliculas = actor1.numeroPeliculasActor();
```

```
                System.out.println("El actor ha participado en " + numPeliculas);
            } else {
                System.out.println("El actor introducido no existe");
            }
        }

        // #4
        public static void borrarActor() {
            String nombreActor = "";
            Actor actor1 = null;
            System.out.println("Introduce nombre del actor a borrar: ");
            nombreActor = pedirString();
            actor1 = CatalogoListaActores.getCatalogoListaActores().obtenerActor(
                    nombreActor);
            if (actor1 != null) {
                CatalogoListaActores.getCatalogoListaActores().borrarActor(actor1);
                System.out.println("Actor borrado");
            } else {
                System.out.println("No existe el actor");
            }
        }



        // # 6
        public static void obtenerActoresOrd() {
            CatalogoListaActores.getCatalogoListaActores().imprimirOrdenada();
        }

        // # 7
        public static void estanConectados() {
            // introducimos dos actores por consola y nos aseguramos de que los dos
            // existan y no sean iguales
            Actor a1 = pedirActor();
            Actor a2 = pedirActor();
            boolean esta = false;
            if (a1 != null && a2 != null
                    && !a1.getNombre().equalsIgnoreCase(a2.getNombre())) {
                Queue<Actor> porExaminar = (Queue<Actor>) new LinkedList<Actor>();
                HashMap<String, Relacion> examinados = new HashMap<String, Rela-
cion>();
                try {
                    esta = CatalogoListaActores.getCatalogoListaActores()
                            .estanConectados(a1, a2, porExaminar, examina-
dos);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println(esta);
                CatalogoListaActores.getCatalogoListaActores().limpiarListas(
                        porExaminar, examinados);
            }

        }

        // # 8
        public static void estanConectados2() {
            Actor a1 = pedirActor();
            Actor a2 = pedirActor();
            Stack<Relacion> resultado = null;
            if (a1 != null && a2 != null
```

```
                        && !a1.getNombre().equalsIgnoreCase(a2.getNombre())) {
                    Queue<Actor> porExaminar = (Queue<Actor>) new LinkedList<Actor>();
                    HashMap<String, Relacion> examinados = new HashMap<String, Rela-
cion>();
                    try {
                        resultado = CatalogoListaActores.getCatalogoListaActores()
                                .estanConectados2(a1, a2, porExaminar, examina-
dos);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    if (resultado != null) {
                        CatalogoListaActores.getCatalogoListaActores().imprimirPila(
                                resultado);
                    } else {
                        System.out.println(a1.getNombre() + " no esta relacionado con
"
                                + a2.getNombre());
                    }
                    CatalogoListaActores.getCatalogoListaActores().limpiarListas(
                            porExaminar, examinados);
                }
                ;
        }

        // # 9
        public static void gradoRelaciones() {
            Stopwatch timer = new Stopwatch();
            double resultado = CatalogoListaActores.getCatalogoListaActores()
                    .gradoRelaciones();
            System.out.println("El grado de relaciones resultante es " + resultado);
            System.out.println("Tiempo de ejecución: " + timer.elapsedTime());
        }

        // # 10
        public static void centralidad() {
            double resultado;
            Queue<Actor> porExaminar = (Queue<Actor>) new LinkedList<Actor>();
            HashMap<String, Relacion> examinados = new HashMap<String, Relacion>();

            Actor a1 = pedirActor();
            Stopwatch timer = new Stopwatch();
            if (a1 != null) {
                resultado = a1.centralidad(porExaminar, examinados);
                System.out.println("La centralidad del actor : " + a1.getNombre()
                        + " es : " + resultado);
            }else{
                System.out.println("No existe el actor introducido.");
            }
            System.out.println("Tiempo de ejecución: " + timer.elapsedTime());

        }

        private static Actor pedirActor() {
            String nombreActor = "";
            System.out.println("Introduce el nombre del Actor ");
            nombreActor = pedirString();
            if (CatalogoListaActores.getCatalogoListaActores().obtenerActor(
                    nombreActor) == null) {
                return null;
            } else {
```

```
                    return CatalogoListaActores.getCatalogoListaActores().obtenerActor(
                                nombreActor);
            }
      }

}
```

5.2. Clase CatalogoListaActores

```
public class CatalogoListaActores {

      private HashMap<String, Actor> lista = new HashMap<String, Actor>();
      private static CatalogoListaActores miListaActores;
      private Stopwatch stp1;

      private CatalogoListaActores() {

      }

      public static CatalogoListaActores getCatalogoListaActores() {
            if (miListaActores != null) {
                  return miListaActores;
            } else {
                  return miListaActores = new CatalogoListaActores();
            }
      }

      public Actor obtenerActor(String pNombre) {
            Actor a = null;
            if (this.lista.containsKey(pNombre)) {
                  a = this.lista.get(pNombre);
            }
            return a;
      }

      public void borrarActor(Actor a) {
            Iterator<Entry<String, Pelicula>> itrPeliculasActor = a
                        .getIteradorPeliculas();
            while (itrPeliculasActor.hasNext()) {
                  itrPeliculasActor.next().getValue().borrarActorPelicula(a);
            }
            lista.remove(a.getNombre());
      }

      public void addActor(Actor a) {
            lista.put(a.getNombre(), a);
      }

      public static void cargarDatos() {
            Stopwatch timer = new Stopwatch();
            String[] datosLinea;
            String nombreActor;
            String linea = null;
            Actor = null;
            Pelicula;
            try {
                  String fichero = "docs/listaactores40000.txt";
                  // String fichero = "docs/lista1M.txt";
                  Scanner entrada = new Scanner(new FileReader(fichero));
                  while (entrada.hasNext()) {
```

```
                        linea = entrada.nextLine();
                        // si hay líneas en blanco pasa de ellas
                        if (linea.trim().length() == 0)
                              continue;
                        // Cada palabra de "linea" se mete en una posición del
                        // Array de strings "datosLinea"
                        datosLinea = linea.split("\\s\\\\\\s");
                        // Añadimos el primer elemento que es la pelicula
                        pelicula = new Pelicula(datosLinea[0]);

                        for (int i = 1; i < datosLinea.length; i++) {
                              // Ahora añadimos actores
                              nombreActor = datosLinea[i];
                              actor = CatalogoListaActores.getCatalogoListaActores()
                                          .obtenerActor(nombreActor);
                              if (actor == null) {
                                    // si no existe lo creamos y añadimos a la mae
                                    actor = new Actor(nombreActor);
                                    CatalogoListaActores.getCatalogoListaActores()
                                                .addActor(actor);
                              }
                              actor.addPeliculaActor(pelicula);
                              // añadimos cada actor a la pelicula
                              pelicula.addActorPelicula(actor);
                        }

                        // Anadimos la peli al catalogo
                        CatalogoListaPeliculas.getCatalogoListaPeliculas().addPeli-
cula(
                                    pelicula);

                  }
                  entrada.close();
            } catch (IOException e) {
                  e.printStackTrace();
            }
            System.out.println(timer.elapsedTime());
      }

      public Iterator<Entry<String, Actor>> getIteradorActores() {
            return lista.entrySet().iterator();
      }

      public void imprimirActores() {
            Iterator<Entry<String, Actor>> itr = this.getIteradorActores();
            Actor a = new Actor("");
            while (itr.hasNext()) {
                  a = itr.next().getValue();
                  System.out.println("Actor: " + a.getNombre());
            }
      }

      public void imprimirOrdenada() {
            stp1 = new Stopwatch();
            this.ordenarActores(this.cargarArrayList());
      }

      // convierto el hasMap en un ArrayList<Actor>
      public ArrayList<Actor> cargarArrayList() {
            ArrayList<Actor> listaActores = new ArrayList<Actor>();
            Iterator<Entry<String, Actor>> itr = this.getIteradorActores();
```

```
		Actor nuevo = null;
		while (itr.hasNext()) {
			nuevo = itr.next().getValue();
			listaActores.add(nuevo);
		}
		return listaActores;
	}

	private void ordenarActores(ArrayList<Actor> pListaActores) {
		// utilizo el método quickSort reescrito para ordenar la lista
		// y la imprimo por pantalla;
		CatalogoListaActores.quicksort(pListaActores, 0,
				pListaActores.size() - 1);
		System.out.println(stp1.elapsedTime());
		System.out.println("¿Mostrar la lista ordenada? ¿S/N?");
		String resp = Menu.pedirString();
		if (resp.equalsIgnoreCase("s"))
			listarArrayOrdenada(pListaActores);
	}

	private void listarArrayOrdenada(ArrayList<Actor> pListaActores) {
		Iterator<Actor> itr = pListaActores.iterator();
		while (itr.hasNext()) {
			System.out.println(itr.next().getNombre());
		}
	}

	private static void quicksort(ArrayList<Actor> A, int izq, int der) {
		Actor pivote = A.get(izq); // tomamos primer elemento como pivote
		int i = izq; // i realiza la búsqueda de izquierda a derecha
		int j = der; // j realiza la búsqueda de derecha a izquierda
		Actor aux;
		while (i < j) { // mientras no se crucen las búsquedas
			while (A.get(i).compareTo(pivote) <= 0 && i < j)
				i++; // busca elemento mayor que pivote
			while (A.get(j).compareTo(pivote) > 0)
				j--; // busca elemento menor que pivote
			if (i < j) { // si no se han cruzado
				aux = A.get(i); // los intercambia
				A.set(i, A.get(j));
				A.set(j, aux);
			}
		}
		A.set(izq, A.get(j)); // se coloca el pivote en su lugar de forma que
		                      // tendremos
		A.set(j, pivote); // los menores a su izquierda y los mayores a su
		                  // derecha
		if (izq < j - 1)
			quicksort(A, izq, j - 1); // ordenamos subarray izquierdo
		if (j + 1 < der)
			quicksort(A, j + 1, der); // ordenamos subarray derecho
	}

	public boolean estanConectados(Actor a1, Actor a2,
			Queue<Actor> porExaminar, HashMap<String, Relacion> examinados)
			throws InterruptedException {
		// pre: los dos actores exiten en el catalogo de los actores y el actor
		// 1 diferente al actor 2

		a1.cargarColegas();// cargamos los colegas del actor
		Relacion rPrimera = new Relacion(null, null, a1);
```

```
			porExaminar.add(rPrimera.getActual());
			examinados.put(rPrimera.getActual().getNombre(), rPrimera);

			while (!porExaminar.isEmpty()) {
				Actor a = porExaminar.poll();
				if (a2.getNombre().equalsIgnoreCase(a.getNombre())) {
					return true;
				} else {
					this.actualizarListas(a, porExaminar, examinados);
				}
			}
			return false;
		}

		private void actualizarListas(Actor a1, Queue<Actor> porExaminar,
				HashMap<String, Relacion> examinados) {
			a1.cargarColegas();
			Iterator<Entry<String, Relacion>> itr2 = a1.getIteradorColegas();
			while (itr2.hasNext()) {
				Relacion aux = itr2.next().getValue();
				if (!examinados.containsKey(aux.getActual().getNombre())) {
					porExaminar.add(aux.getActual());
					examinados.put(aux.getActual().getNombre(), aux);
				}
			}
			a1.limpiarColegas();
		}

		public Stack<Relacion> estanConectados2(Actor a1, Actor a2,
				Queue<Actor> porExaminar, HashMap<String, Relacion> examinados)
				throws InterruptedException {
			if (estanConectados(a1, a2, porExaminar, examinados)) {
				Stack<Relacion> resultado = new Stack<Relacion>();
				Relacion relacionActual = examinados.get(a2.getNombre());
				while (!relacionActual.getActual().getNombre()
							.equalsIgnoreCase(a1.getNombre())) {
					resultado.add(relacionActual);
					relacionActual = examinados.get(relacionActual.getPadre()
								.getNombre());
				}
				limpiarListas(porExaminar, examinados);
				return resultado;
			}
			limpiarListas(porExaminar, examinados);
			return null;
		}

		public void imprimirPila(Stack<Relacion> resultado) {
			while (!resultado.isEmpty()) {
				Relacion rActual = resultado.pop();
				System.out.println("El actor " + rActual.getActual().getNombre()
							+ " esta relacionado con " + rActual.getPadre().getNom-
bre()
							+ " con la pelicula "
							+ rActual.getPelicularelacion().getNombre());
			}

		}

		public void limpiarListas(Queue<Actor> porExaminar,
				HashMap<String, Relacion> examinados) {
```

```
			porExaminar.clear();
			examinados.clear();
		}

		public double gradoRelaciones() {
			Queue<Actor> porExaminar = (Queue<Actor>) new LinkedList<Actor>();
			HashMap<String, Relacion> examinados = new HashMap<String, Relacion>();
			Actor a1 = null;
			Actor a2 = null;
			int numeroPruebas = 50;
			double media = 0.0;
			Stack<Relacion> resultado = new Stack<Relacion>();

			ArrayList<Actor> listaArray = CatalogoListaActores
					.getCatalogoListaActores().cargarArrayList();

			for (int i = 0; i < numeroPruebas - 1; i++) {
				Random rand = new Random();
				int randomNum1 = rand.nextInt((listaArray.size() - 1) + 1) + 0;
				int randomNum2 = rand.nextInt((listaArray.size() - 1) + 1) + 0;
				a1 = listaArray.get(randomNum1);
				a2 = listaArray.get(randomNum2);
				try {
					resultado = estanConectados2(a1, a2, porExaminar, examina-
dos);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
				limpiarListas(porExaminar, examinados);

				if (resultado != null) {
					media += resultado.size();
				}
			}
			return media / numeroPruebas;
		}
}
```

5.3. Clase Actor

```
public class Actor implements Comparable<Actor> {

	private final String nombre;
	private HashMap<String, Pelicula> listaPeliculasPorActor = new HashMap<String,
Pelicula>();
	private HashMap<String, Relacion> colegas = new HashMap<String, Relacion>();

	public Actor(String nombre) {
		this.nombre = nombre;
	}

	public String getNombre() {
		return this.nombre;
	}

	public HashMap<String, Pelicula> getListaPeliculasPorActor() {
		return this.listaPeliculasPorActor;
	}

	public void addPeliculaActor(Pelicula p) {
```

```
			this.listaPeliculasPorActor.put(p.getNombre(), p);
		}

		public Iterator<Entry<String, Pelicula>> getIteradorPeliculas() {
			return this.listaPeliculasPorActor.entrySet().iterator();
		}

		public Iterator<Entry<String, Relacion>> getIteradorColegas() {
			return this.colegas.entrySet().iterator();
		}

		public void imprimirPeliculas() {
			Iterator<Entry<String, Pelicula>> itr = this.getIteradorPeliculas();
			Pelicula p;
			while (itr.hasNext()) {
				p = itr.next().getValue();
				System.out.println("Pelicula: " + p.getNombre());
			}
		}

		public Pelicula aParticipadoEnPelicula(String pPelicula) {
			return this.listaPeliculasPorActor.get(pPelicula);
		}

		public int numeroPeliculasActor() {
			return this.listaPeliculasPorActor.size();
		}

		public void borrarPelicula(Pelicula nombrePelicula) {
			this.listaPeliculasPorActor.remove(nombrePelicula);
		}



		public int compareTo(Actor pActor) {
			return this.getNombre().compareTo(pActor.getNombre());
		}

		public void cargarColegas() {
			this.colegas.clear();
			// recorremos listapeliculas y cargamos cada relacion
			Iterator<Entry<String, Pelicula>> itrpeliculas = this
					.getIteradorPeliculas();
			while (itrpeliculas.hasNext()) {
				// Obtenemos cada pelicula
				Pelicula p = itrpeliculas.next().getValue();
				// Por cada pelicula obtenemos los actores y los cargamos en la
				// hashmap
				Iterator<Entry<String, Actor>> itractores = p
						.getIteradorActoresPelicula();
				while (itractores.hasNext()) {
					Actor actual = itractores.next().getValue();
					if (actual.aParticipadoEnPelicula(p.getNombre()) != null
							&& !colegas.containsKey(actual.getNombre())
							&& !actual.getNombre().equals(this.nombre)) {
						// Si el actor ha participado en la pelicula, no existe
en
						// la lista de colegas y no es el mismo se añade
						Relacion r = new Relacion(this, p, actual);
						colegas.put(actual.getNombre(), r);
					}
```

```
                }
            }
        }

        public HashMap<String, Relacion> getColegas() {
            return colegas;
        }

        public void imprimirColegas() {
            Iterator<Entry<String, Relacion>> itr = this.getIteradorColegas();
            while (itr.hasNext()) {
                System.out.println(itr.next().getValue().getActual().getNombre());
            }
            System.out.println("El numero de Colegas del Actor " + this.nombre
                    + " es: " + this.colegas.size());
        }

        public double centralidad(Queue<Actor> porExaminar,
                HashMap<String, Relacion> examinados) {
            double centralidad = 0.0;

            ArrayList<Actor> listaArray = CatalogoListaActores
                    .getCatalogoListaActores().cargarArrayList();

            for (int i = 0; i < 50; i++) {
                Random rand = new Random();
                int randomNum1 = rand.nextInt((listaArray.size() - 1) + 1) + 0;
                int randomNum2 = rand.nextInt((listaArray.size() - 1) + 1) + 0;

                Actor a1 = listaArray.get(randomNum1);
                Actor a2 = listaArray.get(randomNum2);

                try {
                    Stack<Relacion> resultado = CatalogoListaActores
                            .getCatalogoListaActores().estanConectados2(a1,
a2,
                                                    porExaminar, examinados);
                    CatalogoListaActores.getCatalogoListaActores().limpiarLis-
tas(porExaminar, examinados);
                    if (resultado != null) {
                        if (comprobarSiEstaActor(resultado)) {
                            centralidad++;
                        }
                    }
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }

            return centralidad;
        }

        private boolean comprobarSiEstaActor(Stack<Relacion> relacion) {
            Relacion r = null;
            while (!relacion.isEmpty()) {
                r = relacion.pop();
                if (r.getActual().getNombre().equalsIgnoreCase(this.nombre)) {
                    return true;
                }
            }
            return false;
```

```
	}

	public void limpiarColegas() {
		this.colegas.clear();
	}
}
```

5.4. Clase Relacion

```
public class Relacion {
	private Actor padre;
	private Pelicula peliculasrelacion;
	private Actor actual;

	public Relacion(Actor padre, Pelicula peliculasrelacion, Actor actual) {
		this.padre = padre;
		this.peliculasrelacion = peliculasrelacion;
		this.actual = actual;
	}

	public Actor getPadre() {
		return padre;
	}

	public void setPadre(Actor padre) {
		this.padre = padre;
	}

	public Pelicula getPeliculasrelacion() {
		return peliculasrelacion;
	}

	public void setPeliculasrelacion(Pelicula peliculasrelacion) {
		this.peliculasrelacion = peliculasrelacion;
	}

	public Actor getActual() {
		return actual;
	}

	public void setActual(Actor actual) {
		this.actual = actual;
	}

}
```

6. Conclusiones

- Librerías

En un principio usabamos la librería “sun” para crear la “pila” que provoca un alto consumo de recursos, por eso decidimos usar la librería de java con implementación LinkedList.

- Cambio al modelo A+B

Se ha decidido cambiar al modelo A+B, tanto el actor como la película tienen información de las películas en donde han participado y los actores que han participado en ella respectivamente, para realizar este laboratorio y de esta forma hacerlo más eficiente.

- Problemas con la memoria

Al realizar las pruebas del método “gradoRelaciones” no se conseguía mirar todos los actores y daba un fallo de memoria sin terminar el proceso. Una de las soluciones para liberar memoria ha sido vaciar la lista de colegas de cada actor una vez examinados puesto que no nos servía para nada.

- Pruebas

  Para realizar las pruebas hemos obtenido una muestra de 40 mil del archivo de 1 millón ya que realizarlas todas sería inviable.