

BILBOKO INGENIARITZA ESKOLA

KUDEAKETAREN ETA INFORMAZIO SISTEMEN
INFORMATIKAREN INGENIARITZAKO GRADUA

Aktoreak eta pelikulak kudeatu (3. eginkizuna)

Egileak:

Adei Arias

Jon Barbero

Ander Prieto

Arloa:

Datu-Egiturak eta

Algoritmoak

2. maila

46. taldea

1. lauhilabetea



Aurkibidea

| | |
|---|----------|
| 1. Sarrera eta arazoaren aurkezpena | 1 |
| 2. Diseinua | 2 |
| 3. Datu egituren diseinua | 3 |
| 4. Metodo nagusien diseinu eta inplementazioa | 4 |
| 4.1. Lehen eginkizuneko kodea | 4 |
| 4.1.1. Datuak kargatu fitxategi batetik | 4 |
| 4.1.2. Aktore baten bilaketa | 5 |
| 4.1.3. Aktore berri baten txertaketa | 5 |
| 4.1.4. Aktore baten pelikulak bueltatu | 6 |
| 4.1.5. Pelikula bateko aktoreak bueltatu | 6 |
| 4.1.6. Pelikula baten dirua gehitu | 7 |
| 4.1.7. Aktore baten ezabaketa | 7 |
| 4.1.8. Aktoreen zerrenda fitxategi batean gorde | 8 |
| 4.1.9. Aktoreen zerrenda ordenatua lortu | 9 |
| 4.2. Hirugarren eginkizuneko kodea | 10 |
| 4.2.1. Grafoa sortu | 10 |
| 4.2.2. Konektatuta | 10 |
| 4.2.3. Erlazionatuta | 12 |

| | |
|--|-----------|
| 5. Kodea | 14 |
| 5.1. Lehen eginkizuneko kodea | 14 |
| 5.1.1. Aktore.java | 14 |
| 5.1.2. ArrayPelikulak.java | 14 |
| 5.1.3. ArrayAktoreak.java | 16 |
| 5.1.4. ListaAktoreak.java | 18 |
| 5.1.5. ListaPelikula.java | 22 |
| 5.1.6. Pelikula.java | 24 |
| 5.2. Hirugarren eginkizuneko kodea | 25 |
| 5.2.1. GraphHash.java | 25 |
| 6. JUnitak | 29 |
| 6.1. Lehen eginkizuneko JUnitak | 29 |
| 6.1.1. AktoreTest.java | 29 |
| 6.1.2. ArrayPelikulakTest.java | 31 |
| 6.1.3. ArrayAktoreakTest.java | 33 |
| 6.1.4. ListaAktoreakTest.java | 35 |
| 6.1.5. ListaPelikulaTest.java | 38 |
| 6.1.6. PelikulaTest.java | 40 |
| 6.2. Hirugarren eginkizuneko JUnitak | 43 |
| 6.2.1. GraphHashTest.java | 43 |
| 7. Ondorioak | 46 |
| Erreferentziak | 47 |

1. Sarrera eta arazoaren aurkezpena

Datu-Egiturak eta Algoritmoak ikasgaieko proiektua aktore eta pelikulen kudeaketa egitea da.

Ikasgai honetan, garrantzitsua da programaren kostua. Horretarako, hasiera-hasieratik azpimarratu dugu zer den kostua eta nola zeregin berdin baterako hainbat inplementazio ezberdin dagoen.

Beraz, hau argi ikusteko, hainbat eginkizun bete beharko ditugu lauhilabeteen zehar.

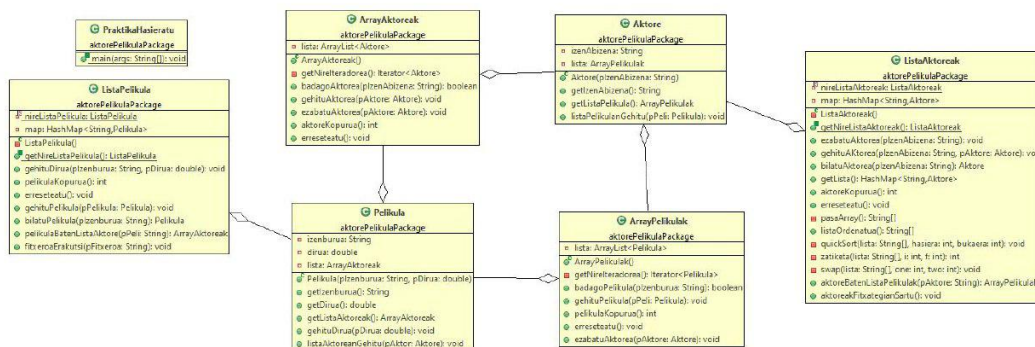
Hirugarren eginkizun^[1] honetan, aktore eta pelikulen arteko loturak aztertuko ditugu. Horretarako, grafoak erabiliko ditugu, guk sortutako *GraphHash* izeneko datu-egitura inplementatuz. Aldi berean, bi aktore erlazionatuta badauden esango digun programa bat garatuko dugu, bai egia ala gezurra den edo aktore batetik besterako lotura nolakoa den esango diguna.

Gure praktikaren *main* metodoa *PraktikaHasieratu* klasean dago. Hala ere, interaktiboa den *main* bat jarri dugu (*Main_Interaktiboa* klasean), non lehen eta hirugarren eginkizunen atazak dauden, egiazko kudeaketa programa bat balitz bezala. Dena dela, denbora proba guztiak *PraktikaHasieratu* klasean agertzen dira.

Berriz ere, Eclipse gure “dantzarako bikote” izango dugu programatzeko momentuan, baita \LaTeX ere lortutako kodea, emaitzak eta hauen erreferentziak idazteko eta islatzeko.

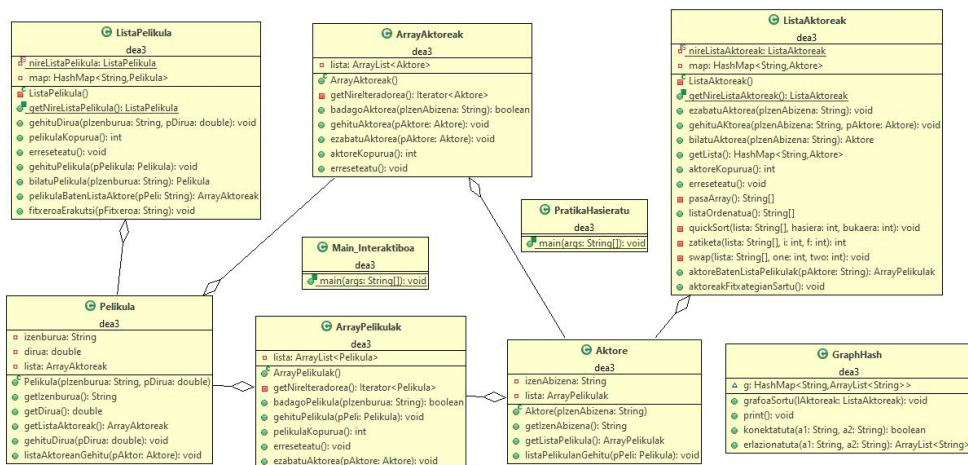
2. Diseinua

Hirugarren eginkizun honetarako, lehenengo eginkizuneko diseinua oinarritzat hartu da (1. irudia).



1. irudia: Lehenengo eginkizuneko diseinua

Orain, aldiz, egin beharreko aldaketak txikiak dira; *GraphHash* izeneko klase berri bat sortuko dugu, datu-egitura mota berri hau inplementatzeko, eta beharrezko klaseekin erlazionatuko dugu (2. irudia).



2. irudia: Diseinu berria

3. Datu egituren diseinua

Eginkizun hau lehenengoaren jarraipena denez, bertan idatzitakoa aipatuko dut:

Hiru datu egitura mota desberdin erabili ditugu eginkizun honetan; *Array*-a, *ArrayList*-a eta *HashMap*-a.

Bi klase ditugu non datu asko gorde behar diren: *ListaAktorea* eta *ListaPelikula*. Kasu hauetan, *HashMap* bat erabili dugu. Datu egitua hau, lista baten objektuak sartzeko, bilatzeko edota ezabatzeko, kostu konstantea du. Aurreko guztia egiteko, datu egitura hau, oso erabilgarria da. Azken batean, exekuzioaren denbora asko jaitsiko da. *ArrayList*-ak, objektu bat lista batean dagoen jakiteko, kostu lineala du; *HashMap* batek, ordea, kostu konstantea. Beraz, xehetasun hauek exekuzioaren denboran asko eragingo dute.

Ondoren, pelikula bakoitzak pelikula horretarako lan egiten duten aktoreen zerrenda bat du. Baita, aktore bakoitzak, lan egiten duen pelikulen zerrenda bat du. Bi zerrenda hauek sortzeko, *ArrayList*-ak erabili ditugu. Azken batean, lista hauek ez dute elementu gehiegirik izango, eta honen ondorioz, ez dugu arazorik izango lista hauek erabiltzeko orduan.

Azkenik, *Array*-ak erabili ditugu. Lehenik eta behin, datuak fitxeretik kargatzean, lerro bakoitza, bi *Array* desberdinetan banatuko ditugu (bat pelikula-rentzat eta bestea aktoreentzat) *split* bat eginez. Datu egitura hau erabiltzea, oso erraza da. Ez dugu iteradore metodorik erabili beharrik, indize batekin lista guztia zeharkatzeko aukera izango dugu.

Bukatzeko, lista ordenatzeko garaian, *QuickSort* metodoa erabili dugu. Metodo honek *String*-eko *Array* bat jasoko du parametro bezala (kasu honetan, *ListaAktorea* klaseko instantzia bat jasoko du, hori bai, lehenago metodo bat sortu beharko dugu, *HashMap* listatik, *Array* motako listara pasatzeko).

Eginkizun honi dagokionez, *HashMap*^[2] egituraz gain *HashSet*^[3] egitura erabili dugu; izan ere, guk sortutako *GraphHash* egituraren oinarriak dira, operazioak egiterako orduan denbora konstantea baitu.

Gainera, pilak eta ilarak ere erabili ditugu, datuen kudeaketa egiteko. Izan ere, pilek LIFO (*Last in, first out*) algoritmoak eta ilarek FIFO (*First in, first out*) algoritmoak erabiltzen dituzte; egoeraren arabera, bata edo bestea erabili behar izango dugu.

4. Metodo nagusien diseinu eta implementazioa

4.1 LEHEN EGINKIZUNEKO KODEA

4.1.1 *Datuak kargatu fitxategi batetik*

public void fitxeroaErakutsi(String pFitxeroa) throws FileNotFoundException, IOException

// Aurre: Fitxeroaren relative edo absolute path-a eman behar diogu.

// Post: Datuak programan kargatuko dira.

- Proba kasuak:

1. Datuak programan kargatzea
2. Fitxategia ez aurkitzea

- Algoritmoa:

```
sarrera = new Scanner(fitxategiaren izena)
while(sarrera.hasNext){
    lerroa=sarrera.nextLine();
    lerroa.split // pelikula | aktore guztiak batera
    pelikula
    pelik = new Pelikula(lerroa[0])
    gehituPelikula(pelik)
    aktoreak = lerroa[1].split // aktoreen Arraya
    while (aktoreak daude arrayan){
        aktor = bilatu aktorea HashMapean
        if (aktor==null){
            sartu aktorea
        }
        peli.gehitu(aktor) //pelikulari aktorea gehitu
        aktor.gehitu(peli) //aktoreari pelikula gehitu
    }
}
```

- Kostua: $O(n \cdot m)$ // n = lerroak iteratu; m = aktoreak iteratu

4.1.2 Aktore baten bilaketa

```
public Aktore bilatuAktorea(String pIzenAbizena)
```

```
// Aurre: Parametro gisa bilatu nahi den aktorearen izena eman behar da.
```

```
// Post: Aktorea aurkitzen badu, aktorea bera itzultzen du; bestela, null itzultzen du.
```

- Proba kasuak:

1. Aktorea badago.

- a) Elementu batez osatutako listan
- b) Elementu anitzez osatutako listan

2. Aktorea ez dago.

- a) Elementuz osatutako listan
- b) Lista hutsean

- Algoritmoa:

```
aktore=null;  
if (HashMap ListaAktoreak aktore badauka){  
    get aktore HashMapetik}  
return aktore;
```

- Kostua: $O(1)$

4.1.3 Aktore berri baten txertaketa

```
public void gehituAktorea(String pIzenAbizena, Aktore pAktore)
```

```
// Aurre: Izena eta aktorea bera pasatu behar zaio. Lehena HashMapean bilatzeko; egon  
ez badago, objektua sartzeko.
```

```
// Post: Aktorea txertatuko da.
```

- Proba kasuak:

1. Aktorea badago jada.
2. Aktorea ez dago oraindik.

- Algoritmoa:

```
if (HashMap ListaAktoreak pIzenAbizena EZ badauka){
    sartu pAktore HashMapean, pIzenAbizena gakoarekin}
```

- Kostua: $O(1)$

4.1.4 Aktore baten pelikulak bueltatu

// Post: Aktoreak fitxero berri batean sartuko dira. public ArrayPelikulak aktoreBaten-ListaPelikulak(String pAktore)

// Aurre: Aktore baten izena jasoko du parametro gisa.

// Post: Izen hori jakinda, HashMapean Aktore objektua bilatuko du eta honen lista bueltatuko du. Aurkitu gabe, null bueltatuko du.

- Proba kasuak:

1. listaAktorean dagoen aktore baten izena pasatzea.
2. listaAktorean ez dagoen aktore baten izena pasatzea.

- Algoritmoa:

```
Aktore aktor = this.bilatuAktorea(pAktore);
if(aktor == null)    return null;
else    return aktor.getListPelikula();
```

- Kostua: $O(1)$

4.1.5 Pelikula bateko aktoreak bueltatu

public ArrayAktoreak pelikulaBatenListaAktore(String pPeli)

// Aurre: Pelikula baten izena jasoko du parametro gisa.

// Post: Izenburu hori jakinda, HashMapean Pelikula objektua bilatuko du eta honen lista bueltatuko du. Aurkitu gabe, null bueltatuko du.

- Proba kasuak:

1. listaPelikulan dagoen pelikula baten izena pasatzea.
2. listaPelikulan ez dagoen pelikula baten izena pasatzea.

■ Algoritmoa:

```
Pelikula peli = this.bilatuPelikula(pPeli);  
if(peli == null) return null;  
else return peli.getListaAktoreak();
```

■ Kostua: $O(1)$

4.1.6 Pelikula baten dirua gehitu

```
public void gehituDirua(String pIzenburua, double pDirua)
```

```
// Aurre: Pelikularen izenburua eta diru kopurua sartu behar da.
```

```
// Post: Pelikula aurkitzean, dirua atxikituko zaio. Ez egotean, ez da ezer gertatuko
```

■ Proba kasuak:

1. Pelikula badago.
2. Pelikula ez dago.

■ Algoritmoa:

```
pelik Pelikula;  
if (HashMap ListaAktoreak pIzenburua badauka){  
    pelik=HashMapetik pelikula lortu //(this.map.get)  
    sartu pDirua pelik objektuan}
```

■ Kostua: $O(1)$

4.1.7 Aktore baten ezabaketa

```
public void ezabatuAktorea(String pIzenAbizena)
```

```
// Aurre: Izena pasatuko zaio parametro gisa.
```

```
// Post: Aktorea ezabatuko da; horretarako, bere pelikula bakoitzetik ere ezabatu beharko da. Aktorea ez balego, ez da ezer gertatuko.
```

- Proba kasuak:

1. Aktorea badago.
2. Aktorea ez dago.

- Algoritmoa:

```
Aktore aktor = bilatu pIzenAbizena HashMapean
ArrayPelikulak lista berria;
if(aktor!=null){
    lista = aktorearen ListaPelikula lortu
    lista.ezabatuAktorea metodoan, banan banan ezabatu aktorea
    ↪ pelikula guztietatik
}
```

- Kostua: $O(1)$

4.1.8 Aktoreen zerrenda fitxategi batean gorde

```
public void aktoreakFitxategianSartu()
```

```
// Aurre: Aktore guztien zerrenda hartuko da.
```

```
// Post: Aktoreak fitxero berri batean sartuko dira.
```

- Proba kasuak:

1. Fitxategia sortzea
2. Arazoa egotea fitxategia sortzean

- Algoritmoa:

```
fitxategia = new FileWriter(fitxategi berriaren izena)
Iterator it = aktoreen zerrenda lortu
while(it.hasNext){
    lerroa=it.next;
    if(it.hasNext){
        idatzi aktorea + "&&&" + lerro saltoa
    }
    else{
        idatzi lerro saltoa //hau soilik behin, hurrengo bueltan ez
        ↪ baita while begiztan sartuko
    }
}
```

```
    }
}
```

- Kostua: $O(n)$ // *While*-ak aktoreen *HashMap* guztia errekorritzen duelako

4.1.9 Aktoreen zerrenda ordenatua lortu

```
public String[] listaOrdenatua()
```

```
// Aurre: Lista ordenatu gabea izango dugu.
```

```
// Post: QuickSort metodoa erabiliz, lista ordenatuta lortuko da.
```

- Proba kasuak:

1. Lista ordenatua tratatu
2. Lista ez ordenatua tratatu
3. Lista hutsa tratatu

- Algoritmoa:

```
String lag = lista[i];
int ezker = i;
int eskuin = f;
while ( ezker < eskuin ){
    while ( lista[ezker].compareTo(lag) <= 0 && ezker < eskuin)
        ezker++;
    while ( lista[eskuin].compareTo(lag) > 0 )
        eskuin--;
    if ( ezker < eskuin )
        swap(lista, ezker, eskuin);
}
lista[i] = lista[eskuin];
lista[eskuin] = lag;
```

- Kostua: $O(n \log n)$

4.2 HIRUGARREN EGINKIZUNEKO KODEA

4.2.1 *Grafoa sortu*

```
public void grafoaSortu(ListaAktoreak lAktoreak)
```

```
// Aurre: --
```

```
// Post: Aktoreen zerrendatik grafoa sortzen du.
```

- Proba kasuak:

- Grafoa ondo kargatzea.
- Grafoa gaizki kargatzea.

- Algoritmoa:

```
aktore = null
pelikula = null
listaAktoreen iteradorea hasieratuko dugu
while(aktoreen lista guztia ez dugun iteratu)
    listaren hurrengo aktorea hartuko dugu iteradorearen bidez(.next())
    g.put(aktorearen izena, aktorearen listaPelikulak String-era pasatuz)
listaPelikulen iteradorea hasieratuko dugu
while(pelikulen lista guztia ez dugun iteratu)
    listaren hurrengo pelikula hartuko dugu iteradorearen bidez(.next())
    g.put(pelikularen izena, pelikularen listaAktoreak String-era pasatuz)
```

- Kostua: $O(n + m)$ // n : aktore kop., m : pelikula kop.

Egia da bai aktore eta bai pelikula listetan ArrayList batzuk iteratzen ditugula, baina hauen tamaina hain txikia denez bi lista nagusiekin konparatuta, algoritmoa kostu lineala izaten jarraitzen du.

4.2.2 *Konektatuta*

```
public boolean konektatuta(String a1, String a2)
```

```
// Aurre: Bi aktoreen izen-abizenak pasatuko dira, ".^bizena, izena"formatuan.
```

```
// Post: Bi aktore horien artean erlazioirik baldin badago esango du programak.
```

■ Proba kasuak:

- Grafoan elementuak ez daude:

| GraphHash | a1 | a2 | Eraitza |
|-----------|----|----|-----------------------------------|
| (a-b-c-d) | b | e | False; "e ez dago grafoan" |
| (a-b-c-d) | e | a | False; "e ez dago grafoan" |
| (a-b-c-d) | e | f | False; "e eta f ez daude grafoan" |

- Grafoan elementuak badaude:

| GraphHash | a1 | a2 | Eraitza |
|---------------------|----|----|---------|
| (a-b-c-d) (e-f-g-h) | a | d | True |
| (a-b-c-d) (e-f-g-h) | a | f | False |

■ Algoritmoa:

```

Queue<String> azTGabe = new LinkedList<String>()
HashSet<String> aztertuak = new HashSet<String>()
konek = false
"a1" sartu azTGabe ilaran
"a1" sartu HashSet-ean
Parametroen bidez pasatutako aktoreak ez baleude grafoan
mezuak erakutsi, agertzen ez den aktorea zein den esanez
while(azTGabe ez den hutsa eta konek false den)
    eg = azTGabe ilaratik lehenengo elementua kendu
    if(eg.equals(a2))
        konek = true
    else
        ArrayList<String> array = eg-ren lista lortu
        array-ren iteradorea lortu
        while(array ez den bukatu)
            izena = array-ren hurrengo elementua
            if(izena HashSet-ean ez badago)
                ilaran sartu izena
                HashSet-ean sartu izena
konek itzuli

```

- Kostua: $O(n)$ // n : grafoko elementu guztiak

Grafoa sortzean gertatu den arazo berdinarekin aurrean gaude: lehenengo while-aren barruan, HashMap-eko elementu bakoitzaren ArrayList-a iteratzen dugu, baina hau hain txikia denez "nrekin konparatuz, kostea linea izaten jarraitzen du.

4.2.3 Erlazionatuta

```
public ArrayList<String>erlazionatuta(String a1, String a2)
```

```
// Aurre: Bi aktoreen izen-abizenak pasatuko dira, "abizena, izena" formatuan.
```

```
// Post: Bi aktore horien artean erlaziorik baldin badago, erlazionatzen duen bidea itzuliko du. Erlaziorik ez balego, aldiz, null.
```

■ Proba kasuak:

- Grafoan elementuak ez daude:

| GraphHash | a1 | a2 | Eraitza |
|-----------|----|----|----------------------------------|
| (a-b-c-d) | b | e | null; "e ez dago grafoan" |
| (a-b-c-d) | e | a | null; "e ez dago grafoan" |
| (a-b-c-d) | e | f | null; "e eta f ez daude grafoan" |

- Grafoan elementuak badaude:

| GraphHash | a1 | a2 | Eraitza |
|---------------------|----|----|---------------------------------------|
| (a-b-c-d-e) (f-g-h) | a | e | <a>,,<c>,<d>,<e> |
| (a-b-c-d-e) (f-g-h) | a | f | null; "Aktoreak ez daude konektatuta" |

■ Algoritmoa:

```
aux=konektatuta(a1,a2);
atera=false;
Queue<String> aztGabe = new LinkedList<String>();
HashSet<String> aztertuak = new HashSet<String>();
Stack<String> nondik = new Stack<String>();
Stack<String> adabegia = new Stack<String>();
ArrayList<String> eraitza=new ArrayList<String>();
"a1" sartu aztGabe ilaran
"a1" sartu HashSet-ean
"a1" sartu adabegia pila
while(aztGabe ez den hutsa eta konek false den)
    eg = aztGabe ilaratik lehenengo elementua kendu
    if(eg.equals(a2))
        konek = true
    else
        ArrayList<String> array = eg-ren lista lortu
        array-ren iteradorea lortu
```

```

while(array ez den bukatu)
    izena = array-ren hurrengo elementua
    if(izena HashSet-ean ez badago)
        ilaran sartu izena
        HashSet-ean sartu izena
        nondik pilan sartu eg
        adabegia pilan sartu izena
aurrekoa = a2
sartu ArrayList emaitzan a2
while(adabegia pila ez den hutsa eta aurrekoa ez den a1)
    while(adabegia-ren lehenengo elementua ez den aurrekoa eta nondik ez den hu
        nondik pilatik elementu bat kendu
        adabegia pilatik elementua bat kendu
    aurrekoa = nondik pilatik elementu bat kenduta
    aurrekoa gorde emaitza arrayListean
    adabegia pilatik elementu bat kendu
else
    mezu bat atera, aktoreak ez daudela konektatuta esanez
emaitza arraylista bueltatu

```

- Kostua: $O(n + m + p)$ // n : grafoko elementu kop., m : adabegi pilaren elementu kop, p : emaitzeko ArrayList-aren elementu kop.

Hemen, bi zati bereizten dira. Lehenengo zatia, n -rena, konektatu metodoaren kostu berdina izango du; hau da, lineala izango da. Eta bigarren zatia, lehenengo zatiarekiko guztiz independentea dena, berriro ere kostu lineala izango du. Azken batean, bigarren zati honetan, ArrayList-a betetzen joango gara, eta honetarako bi while erabiliko ditugu, bat bestearen barruan. Baina kostua ez da koadratikoa izango; izan ere, “adabegia” izeneko pila bat husten joango gara bi while-tan. Horregatik, kostua lineala izaten jarraituko du. Bi zatiak independenteak direnez, kostuak gehituko dira.

5. Kodea

5.1 LEHEN EGINKIZUNEKO KODEA

5.1.1 *Aktore.java*

```
1 package dea3;
2
3 public class Aktore {
4
5     private String izenAbizena;
6     private ArrayPelikulak lista;
7
8     public Aktore(String pIzenAbizena){
9         this.izenAbizena = pIzenAbizena;
10        this.lista = new ArrayPelikulak();
11    }
12
13    public String getIzenAbizena(){
14        return this.izenAbizena;
15    }
16
17    public ArrayPelikulak getListaPelikula(){
18        return this.lista;
19    }
20
21    public void listaPelikulanGehitu(Pelikula pPeli){//Hemen, pelikula bat sartuko
22        ↪ dugu aktorearen listaPelikulan
23        this.lista.gehituPelikula(pPeli);
24    }
```

5.1.2 *ArrayPelikulak.java*

```
1 package dea3;
2
3 import java.util.ArrayList;
4 import java.util.Iterator;
5
6 public class ArrayPelikulak {
7
8     private ArrayList<Pelikula> lista;
9
10    public ArrayPelikulak(){
11        this.lista = new ArrayList<Pelikula>();
12    }
13
14    private Iterator<Pelikula> getNireIteradorea(){
15        return this.lista.iterator();
16    }
17
18    public boolean badagoPelikula(String pIzenburua){//Pelikula bat emanda, listan
19        ↪ dagoen esango digu
20        boolean dago = false;
21        Pelikula pelikula = null;
22        Iterator<Pelikula> itr = this.getNireIteradorea();
23        while(itr.hasNext() && !dago){
24            pelikula = itr.next();
25            if(pelikula.getIzenburua().equals(pIzenburua)){
26                dago = true;
27            }
28        }
29        return dago;
30    }
31
32    public Iterator<Pelikula> getIteradorea(){
33        return this.lista.iterator();
34    }
35
36    public void gehituPelikula(Pelikula pPeli){
37        this.lista.add(pPeli);
38    }
39
40    public int pelikulaKopurua() {
41        return this.lista.size();
42    }
43
44    public void erreseteatu() {
45        this.lista.clear();
46    }
47 }
```

```
45     }
46
47     public void ezabatuAktorea(Aktore pAktore) {
48         Pelikula peli = null;
49         ArrayAktoreak lista=null;
50         Iterator<Pelikula> itr = this.getNireIteradorea();
51         while(itr.hasNext()) {
52             peli = itr.next();
53             lista=peli.getListaAktoreak();
54             lista.ezabatuAktorea(pAktore);
55         }
56     }
57
58     public ArrayList<String> pasaString(){
59         Iterator<Pelikula> itr = this.getNireIteradorea();
60         Pelikula peli = null;
61         ArrayList<String> lista = new ArrayList<String>();
62         while(itr.hasNext()){
63             peli = itr.next();
64             lista.add(peli.getIzenburua());
65         }
66         return lista;
67     }
68 }
```

5.1.3 ArrayAktoreak.java

```
1 package dea3;
2
3 import java.util.*;
4
5 public class ArrayAktoreak {
6
7     private ArrayList<Aktore> lista;
8
9     public ArrayAktoreak(){
10         this.lista = new ArrayList<Aktore>();
11     }
12 }
```

```
12
13 private Iterator<Aktore> getNireIteradorea(){
14     return this.lista.iterator();
15 }
16
17 public boolean badagoAktorea(String pIzenAbizena){//Aktore bat pasata, listan
18     ↪ dagoen esango digu
19     boolean dago = false;
20     Aktore aktor = null;
21     Iterator<Aktore> itr = this.getNireIteradorea();
22     while(itr.hasNext() && !dago){
23         aktor = itr.next();
24         if(aktor.getIzenAbizena().equals(pIzenAbizena)){
25             dago = true;
26         }
27     }
28     return dago;
29 }
30 public Iterator <Aktore> getIteradorea(){
31     return this.lista.iterator();
32 }
33
34 public void gehituAktorea(Aktore pAktore){
35     if(!(this.badagoAktorea(pAktore.getIzenAbizena()))){
36         this.lista.add(pAktore);
37     }
38 }
39
40 public void ezabatuAktorea(Aktore pAktore) {
41     this.lista.remove(pAktore);
42 }
43
44 public int aktoreKopurua() {
45     return this.lista.size();
46 }
47
48 public void erreseteatu() {
49     this.lista.clear();
50 }
51
52 public ArrayList<String> pasaString(){
53     Iterator<Aktore> itr = this.getNireIteradorea();
54     Aktore aktore = null;
55     ArrayList<String> lista = new ArrayList<String>();
56     while(itr.hasNext()){
57         aktore = itr.next();
```

```
57     lista.add(aktore.getIzenAbizena());
58 }
59 return lista;
60 }
61 }
```

5.1.4 *ListaAktoreak.java*

```
1 package dea3;
2
3 import java.util.*;
4 import java.io.FileNotFoundException;
5 import java.io.FileWriter;
6 import java.io.IOException;
7
8 public class ListaAktoreak {
9
10     private static ListaAktoreak nireListaAktoreak = null;
11     private HashMap<String,Aktore> map;
12
13
14     private ListaAktoreak(){
15         this.map = new HashMap<String,Aktore>();
16     }
17
18     public static ListaAktoreak getNireListaAktoreak(){
19         if(nireListaAktoreak == null){
20             nireListaAktoreak = new ListaAktoreak();
21         }
22         return nireListaAktoreak;
23     }
24
25     public void ezabatuAktorea(String pIzenAbizena){
26         Aktore aktor=this.bilatuAktorea(pIzenAbizena);
27         ArrayPelikulak lista=null;
28         if(aktor!=null){
29             lista=aktor.getListaPelikula();
30             lista.ezabatuAktorea(aktor);
```

```
31     this.map.remove(pIzenAbizena);
32 }
33 }
34
35 public void gehituAktorea(String pIzenAbizena, Aktore pAktore){
36     if(!this.map.containsKey(pIzenAbizena)){
37         this.map.put(pIzenAbizena, pAktore);
38     }
39 }
40
41 public Aktore bilatuAktorea(String pIzenAbizena){
42     Aktore aktor = null;
43     if(this.map.containsKey(pIzenAbizena)){
44         aktor = this.map.get(pIzenAbizena);
45     }
46     return aktor;
47 }
48
49 public Iterator<String> getIteradorea(){
50     return this.map.keySet().iterator();
51 }
52
53 public HashMap<String,Aktore> getLista(){
54     return this.map;
55 }
56
57 public int aktoreKopurua() {
58     return this.map.size();
59 }
60
61 public void erreseteatu() {
62     this.map.clear();
63 }
64
65
66 private String[] pasaArray(){
67     String[] lista = new String[this.aktoreKopurua()];
68     int i = 0;
69     Iterator<String> it = map.keySet().iterator();
70     String izena = null;
71     while (it.hasNext()){
72         izena = it.next();
73         lista[i]=izena;
74         i=i+1;
75     }
76     return lista;
```

```
77     }
78
79     public String[] listaOrdenatua(){
80         String[] lista = this.pasaArray();
81         quickSort(lista, 0, lista.length-1);
82         return lista;
83     }
84
85
86     private void quickSort(String[] lista, int hasiera, int bukaera){
87         if ( bukaera - hasiera > 0 ) { // taulan elementu bat baino gehiago
88             int indizeaZatiketa = zatiketa(lista, hasiera, bukaera);
89             quickSort(lista, hasiera, indizeaZatiketa - 1);
90             quickSort(lista, indizeaZatiketa + 1, bukaera);
91         }
92     }
93
94     private int zatiketa(String[] lista, int i, int f){
95
96         String lag = lista[i];
97         int ezker = i;
98         int eskuin = f;
99         while ( ezker < eskuin ){
100             lag.toUpperCase();
101             lista[ezker].toUpperCase();
102             lista[eskuin].toUpperCase();
103             while ( lista[ezker].compareTo(lag) <= 0 && ezker < eskuin)
104                 ezker++;
105             while ( lista[eskuin].compareTo(lag) > 0 )
106                 eskuin--;
107             if ( ezker < eskuin )
108                 swap(lista, ezker, eskuin);
109         }
110         lista[i] = lista[eskuin];
111         lista[eskuin] = lag;
112
113         return eskuin;
114     }
115
116     private void swap(String[] lista, int one, int two) {
117         String temp = lista[one];
118         lista[one] = lista[two];
119         lista[two] = temp;
120     }
121
122
```

```
123 public ArrayPelikulak aktoreBatenListaPelikulak(String pAktore){
124     Aktore aktor = this.bilatuAktorea(pAktore);
125     if(aktor == null){
126         return null;
127     }else{
128         return aktor.getListaPelikula();
129     }
130 }
131
132 public void aktoreakFitxategianSartu(){
133     FileWriter fitxategia1 = null;
134     try {
135
136         fitxategia1 = new FileWriter("./FilmsActors20162017Fitxategia.txt");
137         Iterator<String> it =
138             ↳ ListaAktoreak.getNireListaAktoreak().getList().keySet().iterator();
139         String lerroa = null;
140         // Lerro bakoitza fitxategian idazten dugu
141         while (it.hasNext()) {
142             lerroa = it.next();
143             if(it.hasNext()){
144                 fitxategia1.write(lerroa + " &&& " + "\n");
145             }else{
146                 fitxategia1.write(lerroa + "\n");
147             }
148
149             fitxategia1.close();
150
151         }
152         catch (FileNotFoundException e) {
153             System.out.println("Fitxeroa ez da existitzen. ");
154         } catch (IOException e) {
155             System.out.println("Fitxategiaren idazketak huts egin du. ");
156         }
157     }
158 }
```


5.1.5 *ListaPelikula.java*

```
1 package dea3;
2
3 import java.io.BufferedReader;
4 import java.io.FileInputStream;
5 import java.io.FileNotFoundException;
6 import java.io.FileReader;
7 import java.io.IOException;
8 import java.io.InputStreamReader;
9 import java.util.*;
10
11 public class ListaPelikula {
12
13     private static ListaPelikula nireListaPelikula = null;
14     private HashMap<String,Pelikula> map;
15
16     private ListaPelikula(){
17         this.map = new HashMap<String,Pelikula>();
18     }
19
20     public static ListaPelikula getNireListaPelikula(){
21         if(nireListaPelikula == null){
22             nireListaPelikula = new ListaPelikula();
23         }
24         return nireListaPelikula;
25     }
26
27     public void gehituDirua(String pIzenburua, double pDirua){
28         Pelikula pelik = null;
29         if(this.map.containsKey(pIzenburua)){
30             pelik=this.map.get(pIzenburua);
31             pelik.gehituDirua(pDirua);
32         }
33     }
34
35     public int pelikulaKopurua() {
36         return this.map.size();
37     }
38
39     public void erreseteatu() {
40         this.map.clear();
41     }
42 }
```

```

43 public HashMap<String,Pelikula> getLista(){
44     return this.map;
45 }
46
47 public void gehituPelikula(Pelikula pPelikula){
48     if(this.bilatuPelikula(pPelikula.getIzenburua())==null){
49         this.map.put(pPelikula.getIzenburua(),pPelikula);
50     }
51 }
52
53 public Pelikula bilatuPelikula(String pIzenburua){
54     Pelikula pelikula = null;
55     if (this.map.containsKey(pIzenburua)){
56         pelikula=this.map.get(pIzenburua);
57     }
58     return pelikula;
59 }
60
61 public ArrayAktoreak pelikulaBatenListaAktore(String pPeli){
62     Pelikula peli = this.bilatuPelikula(pPeli);
63     if(peli == null){
64         return null;
65     }else{
66         return peli.getListAktoreak();
67     }
68 }
69
70 public void fitxeroaErakutsi(String pFitxeroa) throws FileNotFoundException,
↪ IOException{
71     try{
72         Scanner entrada = new Scanner(new FileReader(pFitxeroa));
73         String linea;
74         Aktore aktor;
75         Pelikula peli = null;
76         while (entrada.hasNext()) {
77             linea = entrada.nextLine();
78             String[] datuak = linea.split("\\s+--->\\s+");
79             peli = new Pelikula(datuak[0],45.00);
80             ListaPelikula.getNireListaPelikula().gehituPelikula(peli);
81             //System.out.println(datuak[0]);
82             String[] aktoreak = datuak[1].split("\\s+&&\\s+");
83             int i=0;
84             while (i < aktoreak.length){//sartu aktoreak eta pelikulak
85                 aktor =
↪ ListaAktoreak.getNireListaAktoreak().bilatuAktorea(aktoreak[i]);
86                 if(aktor==null){

```

```

87         aktor = new Aktore(aktoreak[i]);
88         ListaAktoreak.getNireListaAktoreak().gehituAktorea(aktoreak[i],
89             ↪ aktor);
89     }
90     //System.out.println(aktoreak[i]);
91     peli.listaAktoreanGehitu(aktor); //Hemen, pelikulari aktore hau
92     ↪ sartuko diogu bere listaAktorean
93     aktor.listaPelikulanGehitu(peli); //Hemen, aktore honi sartuko
94     ↪ diogu pelikula hau bere listaPelikulan
95     i++;
96 }
97 }
98 entrada.close();
99 } catch(IOException e) {e.printStackTrace();}
}
}

```

5.1.6 Pelikula.java

```

1 package dea3;
2
3 public class Pelikula {
4
5     private String izenburua;
6     private double dirua;
7     private ArrayAktoreak lista;
8
9     public Pelikula(String pIzenburua, double pDirua){
10         this.izenburua = pIzenburua;
11         this.dirua = pDirua;
12         this.lista = new ArrayAktoreak();
13     }
14
15     public String getIzenburua(){
16         return this.izenburua;
17     }
18
19     public double getDirua(){

```

```

20     return this.dirua;
21 }
22
23 public ArrayAktoreak getListAktoreak(){
24     return this.lista;
25 }
26
27 public void gehituDirua(double pDirua){
28     this.dirua = this.dirua + pDirua;
29 }
30 public void listaAktoreanGehitu(Aktore pAktor){//Hemen, aktore bat sartuko dugu
31     ↪ pelikularen listaAktorean
32     this.lista.gehituAktorea(pAktor);
33 }

```

5.2 HIRUGARREN EGINKIZUNEN KODEA

5.2.1 *GraphHash.java*

```

1  package dea3;
2
3  import java.util.ArrayList;
4  import java.util.HashMap;
5  import java.util.HashSet;
6  import java.util.Iterator;
7  import java.util.LinkedList;
8  import java.util.Queue;
9  import java.util.Stack;
10
11 public class GraphHash {
12     HashMap<String, ArrayList<String>> g = new HashMap<String, ArrayList<String>>();
13
14
15     public void grafoaSortu(ListaAktoreak lAktoreak){
16         // Post: aktoreen zerrendatik grafoa sortzen du
17         // Adabegiak aktoreen izenak eta pelikulen izenburuak dira
18         // KODEA OSATU

```

```

19     Aktore aktor = null;
20     Pelikula peli = null;
21     Iterator<Aktore> itrAktor = lAktoreak.getList().values().iterator();
22     //Lehenengo aktoreak eta ondoren pelikulak sartuko ditugu hashmap-ean
23     while(itrAktor.hasNext()){
24         aktor = itrAktor.next();
25         String izenaAktor = aktor.getIzenAbizena();
26         g.put(izenaAktor, aktor.getListPelikula().pasaString());
27         //pasaString() metodoa erabili dugu, arrayList-a(bai aktoreak gordetzen
28         ↪ dituen eta bai pelikulak gordetzen dituen)
29         //string-era pasatzeko eta hashmapean sartzeko
30     }
31     Iterator<Pelikula> itrPeli =
32     ↪ ListaPelikula.getNireListaPelikula().getList().values().iterator();
33     while(itrPeli.hasNext()){
34         peli = itrPeli.next();
35         String izenaPeli = peli.getIzenburua();
36         g.put(izenaPeli, peli.getListAktoreak().pasaString());
37     }
38 }
39 public void print(){
40     int i = 1;
41     for (String s: g.keySet()){
42         System.out.print("Element: " + i++ + " " + s + " --> ");
43         for (String k: g.get(s)){
44             System.out.print(k + " ### ");
45         } System.out.println();
46     }
47 }
48 public boolean konektatuta(String a1, String a2){
49     Queue<String> aztGabe = new LinkedList<String>();
50     HashSet<String> aztertuak = new HashSet<String>();
51     aztGabe.add(a1);
52     aztertuak.add(a1);
53     //lehenengo elementua sartuko dugu, geroago agertzen bada, berriro ere ez
54     ↪ sartzeko
55     boolean konek = false;
56     if(ListaAktoreak.getNireListaAktoreak().bilatuAktorea(a1)==null &&
57     ↪ ListaAktoreak.getNireListaAktoreak().bilatuAktorea(a2)==null){
58         System.out.println("Sartutako bi aktoreak ez daude grafoan sartuta");
59     }else if(ListaAktoreak.getNireListaAktoreak().bilatuAktorea(a1)==null){
60         System.out.println("Sartutako lehenengo aktorea ez dago grafoan sartuta");
61     }else if(ListaAktoreak.getNireListaAktoreak().bilatuAktorea(a2)==null){
62         System.out.println("Sartutako bigarren aktorea ez dago grafoan sartuta");

```

```

61     }else{
62         while(!aztGabe.isEmpty() && !konek){
63             String eg= aztGabe.remove();
64             if(eg.equals(a2)){
65                 konek=true;
66             }else{
67                 ArrayList<String> array = g.get(eg);
68                 Iterator<String> itr = array.iterator();
69                 while(itr.hasNext()){//ilaratik hartutako elementuaren arrayList-a
50                     ⇨ iteratuko dugu honen elementuak pilan sartzeko
70                     String izena = itr.next();
71                     if(!aztertuak.contains(izena)){
72                         aztGabe.add(izena);
73                         aztertuak.add(izena);
74                         //bai ilaran eta bai hashSet-ean sartuko ditugu, bi datu
75                         ⇨ egituretan elementuak ez errepikatzeko
76                     }
77                 }
78             }
79         }
80         return konek;
81     }
82
83     public ArrayList<String> erlazionatuta(String a1, String a2){
84         boolean aux=konektatuta(a1,a2);
85         boolean atera=false;
86         Queue<String> aztGabe = new LinkedList<String>();
87         HashSet<String> aztertuak = new HashSet<String>();
88         Stack<String> nondik = new Stack<String>();//pila honen helburua, grafoaren
89         ⇨ elementu bakoitza nondik etorri den jakitea da
90         Stack<String> adabegia = new Stack<String>();//pila honetan, grafoaren elementu
91         ⇨ guztiak sartuko ditugu
92         aztGabe.add(a1);
93         aztertuak.add(a1);
94         adabegia.add(a1);
95         //lehenengo elementua sartuko dugu, geroago aterako balitz, berriro ez sartzeko
96         ArrayList<String> emaitza=new ArrayList<String>();
97         if(aux){//bi aktoreak konektatuta badaude sartuko da
98             while(!aztGabe.isEmpty() && !atera){
99                 String lag= aztGabe.remove();
100                 if(lag.equals(a2)){
101                     atera=true;
102                 }else{
103                     ArrayList<String> array = g.get(lag);
104                     Iterator<String> itr = array.iterator();

```

```

103         while(itr.hasNext()){
104             String izena = itr.next();
105             if(!aztertuak.contains(izena)){
106                 aztGabe.add(izena);
107                 aztertuak.add(izena);
108                 nondik.add(lag); //hemen, lehen esan den bezala, beste datu
109                     ↪ egituretan sartzen ari garen elementuaren
110                 //gurasoak sartuko ditugu
111                 adabegia.add(izena);
112                 //aurreko metodoan egin den bezala, datu egitura guztietan
113                     ↪ sartuko da, berriro ere ateratzen bada
114                 //elementu berdina, ez sartzeko
115             }
116         }
117     }
118 } //ARRAYLIST-A BETETZEN HASI
119 String aurrekoa=a2; //Lehenengo elementua sartuko dugu arrayList-ean
120 emaitza.add(a2);
121 while(!adabegia.isEmpty() && !aurrekoa.equals(a1)){
122     while(!adabegia.peek().equals(aurrekoa) && !nondik.isEmpty()){
123         //while honetan, aurrekoa atributuan daukagun balioa aurkitu beharko
124         ↪ dugu adabegia pilan, eta hau egiten dugun
125         //bitartean, nondik pilan elementuak ateratzen joango gara.
126         //Elementua aurkitzean, nondik pilan dagoen azken String-a,
127         ↪ elementuaren gurasoa izango da, eta hau arrayList.an
128         //sartuko dugu. Nondik, hutsa denean, azken-aurreko elementura iritsi
129         ↪ gara(azken elementua metodoaren parametroko
130         //string bat da)
131         nondik.pop();
132         adabegia.pop();
133     }
134     aurrekoa = nondik.pop();
135     emaitza.add(aurrekoa);
136     adabegia.pop();
137 }
138 }
139 }

```

6. JUnitak

6.1 LEHEN EGINKIZUNEKO JUNITAK

6.1.1 *AktoreTest.java*

```
1 package aktorePelikulaPackage;
2 import static org.junit.Assert.*;
3 import java.io.FileNotFoundException;
4 import java.io.IOException;
5 import org.junit.After;
6 import org.junit.Before;
7 import org.junit.Test;
8 import aktorePelikulaPackage.Aktore;
9
10 public class AktoreTest {
11
12     Aktore aktore1, aktor1;
13     ArrayPelikulak lista, lista1;
14     Pelikula peli1;
15     Pelikula peli2;
16
17     @Before
18     public void setUp() throws Exception {
19         aktore1 = new Aktore("Adeiarias");
20         lista = new ArrayPelikulak();
21         peli1=new Pelikula("El Guason",45.00);
22         peli2=new Pelikula ("El Joker", 60.00);
23     }
24
25     @After
26     public void tearDown() throws Exception {
27         aktore1=null;
28         lista=null;
29         peli1=null;
30         peli2=null;
31     }
32
33     @Test
34     public void testGetIzenAbizena() throws FileNotFoundException, IOException {
35         assertEquals(aktore1.getIzenAbizena(), "Adeiarias");
36         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
```



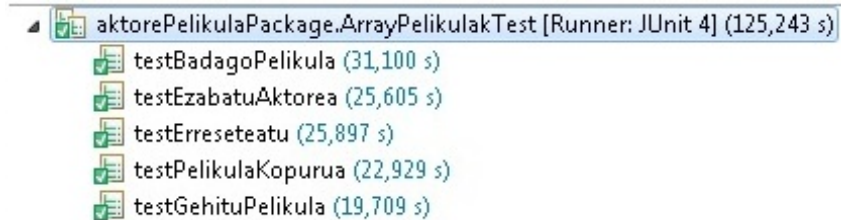
```
37     aktor1 = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Baskin, Cezmi");
38     assertEquals(aktor1.getIzenAbizena(), "Baskin, Cezmi");
39 }
40
41 @Test
42 public void testGetListaPelikula() throws FileNotFoundException, IOException {
43     lista=aktore1.getListaPelikula();
44     assertNotNull(lista);
45     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
46     aktor1 = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Baskin, Cezmi");
47     lista1 = aktor1.getListaPelikula();
48     assertNotNull(lista1);
49 }
50
51 @Test
52 public void testListaPelikulanGehitu() throws FileNotFoundException, IOException {
53     aktore1.listaPelikulanGehitu(peli1);
54     aktore1.listaPelikulanGehitu(peli2);
55     lista=aktore1.getListaPelikula();
56     assertTrue(lista.badagoPelikula(peli1.getIzenburua()));
57     assertTrue(lista.badagoPelikula(peli1.getIzenburua()));
58     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
59     Aktore aktor = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Baskin,
60     ↪ Cezmi");
61     lista = aktor.getListaPelikula();
62     assertEquals(lista.pelikulaKopurua(), 29);
63     lista.gehituPelikula(peli1);
64     assertEquals(lista.pelikulaKopurua(), 30);
65 }
```



6.1.2 *ArrayPelikulakTest.java*

```
1 package aktorePelikulaPackage;
2 import static org.junit.Assert.*;
3 import java.io.FileNotFoundException;
4 import java.io.IOException;
5 import org.junit.After;
6 import org.junit.Before;
7 import org.junit.Test;
8
9 public class ArrayPelikulakTest {
10
11     ArrayPelikulak lista, lista2;
12     Pelikula peli1, peli2, peli3, peli4;
13     Aktore aktor1;
14     ArrayAktoreak lista3;
15
16     @Before
17     public void setUp() throws Exception {
18         lista2 = new ArrayPelikulak();
19         peli1 = new Pelikula("300", 245.00);
20         peli2 = new Pelikula("Annabelle", 47.99);
21         peli3 = new Pelikula("Jurassic Park", 45.00);
22         lista2.gehituPelikula(peli2);
23         lista2.gehituPelikula(peli3);
24     }
25
26     @After
27     public void tearDown() throws Exception {
28
29     }
30
31     @Test
32     public void testBadagoPelikula() throws FileNotFoundException, IOException {
33         assertTrue(lista2.badagoPelikula("Annabelle"));
34         assertFalse(lista2.badagoPelikula("Eager to Die"));
35         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
36         aktor1 = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Devon, Tony");
37         lista = aktor1.getListaPelikula();
38         assertTrue(lista.badagoPelikula("Eager to Die"));
39         assertFalse(lista.badagoPelikula("The Cold Shoulder"));
40     }
41
42     @Test
```

```
43 public void testGehituPelikula() throws FileNotFoundException, IOException {
44     assertEquals(lista2.pelikulaKopurua(), 2);
45     lista2.gehituPelikula(peli3);
46     assertEquals(lista2.pelikulaKopurua(), 3);
47     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
48     aktor1 = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Devon, Tony");
49     lista = aktor1.getListaPelikula();
50     lista.erreseteatu();
51     assertEquals(lista.pelikulaKopurua(), 0);
52     lista.gehituPelikula(peli1);
53     assertEquals(lista.pelikulaKopurua(), 1);
54 }
55
56 @Test
57 public void testPelikulaKopurua() throws FileNotFoundException, IOException {
58     assertEquals(lista2.pelikulaKopurua(), 2);
59     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
60     aktor1 = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Devon, Tony");
61     lista = aktor1.getListaPelikula();
62     assertEquals(lista.pelikulaKopurua(), 4);
63 }
64
65 @Test
66 public void testErreseteatu() throws FileNotFoundException, IOException {
67     assertEquals(lista2.pelikulaKopurua(), 2);
68     lista2.erreseteatu();
69     assertEquals(lista2.pelikulaKopurua(), 0);
70     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
71     aktor1 = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Devon, Tony");
72     lista = aktor1.getListaPelikula();
73     lista.erreseteatu();
74     assertEquals(lista.pelikulaKopurua(), 0);
75 }
76
77 @Test
78 public void testEzabatuAktorea() throws FileNotFoundException, IOException {
79     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
80     aktor1 = ListaAktoreak.getNireListaAktoreak().bilatuAktorea("Devon, Tony");
81     lista = aktor1.getListaPelikula();
82     peli4 = ListaPelikula.getNireListaPelikula().bilatuPelikula("The Wire");
83     lista3 = peli4.getListaAktoreak();
84     assertEquals(lista3.aktoreKopurua(), 702);
85     lista.ezabatuAktorea(aktor1);
86     assertEquals(lista3.aktoreKopurua(), 701);
87 }
88 }
```



6.1.3 ArrayAktoreakTest.java

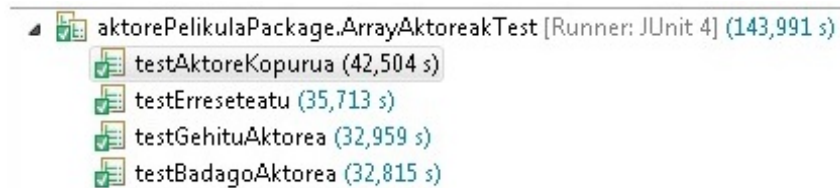
```
1 package aktorePelikulaPackage;
2 import static org.junit.Assert.*;
3 import java.io.FileNotFoundException;
4 import java.io.IOException;
5 import org.junit.After;
6 import org.junit.Before;
7 import org.junit.Test;
8
9 public class ArrayAktoreakTest {
10
11     ArrayAktoreak lista;
12     Aktore aktor1, aktor2, aktor3;
13     Pelikula peli1;
14
15     @Before
16     public void setUp() throws Exception {
17         lista = new ArrayAktoreak();
18         aktor1 = new Aktore("AdeiArias");
19         aktor2 = new Aktore("JonBarbero");
20         aktor3 = new Aktore("AnderPrieto");
21     }
22
23     @After
24     public void tearDown() throws Exception {
25         lista=null;
26         aktor1=null;
27         aktor2=null;
28         aktor3=null;
```

```
29     }
30
31
32
33
34     @Test
35     public void testBadagoAktorea() throws FileNotFoundException, IOException {
36         lista.erreseteatu();
37         lista.gehituAktorea(aktor1);
38         assertEquals(lista.badagoAktorea("AdeiArias"), true);
39         assertEquals(lista.badagoAktorea("JonBarbero"), false);
40         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
41         peli1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("The Ghost of the
42             ↪ Castle");
43         lista = peli1.getListaAktoreak();
44         assertTrue(lista.badagoAktorea("Foti, Leo"));
45         assertFalse(lista.badagoAktorea("Tejada, Beatriz"));
46     }
47
48     @Test
49     public void testGehituAktorea() throws FileNotFoundException, IOException {
50         lista.erreseteatu();
51         assertEquals(lista.aktoreKopurua(), 0);
52         lista.gehituAktorea(aktor2);
53         assertEquals(lista.aktoreKopurua(), 1);
54         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
55         peli1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("The Ghost of the
56             ↪ Castle");
57         lista = peli1.getListaAktoreak();
58         assertEquals(lista.aktoreKopurua(), 17);
59         lista.gehituAktorea(aktor1);
60         assertEquals(lista.aktoreKopurua(), 18);
61     }
62
63     @Test
64     public void testAktoreKopurua() throws FileNotFoundException, IOException {
65         lista.erreseteatu();
66         assertEquals(lista.aktoreKopurua(), 0);
67         lista.gehituAktorea(aktor2);
68         assertEquals(lista.aktoreKopurua(), 1);
69         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
70         peli1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("The Ghost of the
71             ↪ Castle");
72         lista = peli1.getListaAktoreak();
```

```

72     assertEquals(lista.aktoreKopurua(), 17);
73 }
74
75
76
77 @Test
78 public void testErreseteatu() throws FileNotFoundException, IOException {
79     lista.erreseteatu();
80     assertEquals(lista.aktoreKopurua(), 0);
81     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
82     peli1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("The Ghost of the
83         ↪ Castle");
84     lista = peli1.getListaAktoreak();
85     lista.erreseteatu();
86     assertEquals(lista.aktoreKopurua(), 0);
87 }

```



```

▲ aktorePelikulaPackage.ArrayAktoreakTest [Runner: JUnit 4] (143,991 s)
  ✓ testAktoreKopurua (42,504 s)
  ✓ testErreseteatu (35,713 s)
  ✓ testGehituAktorea (32,959 s)
  ✓ testBadagoAktorea (32,815 s)

```

6.1.4 *ListaAktoreakTest.java*

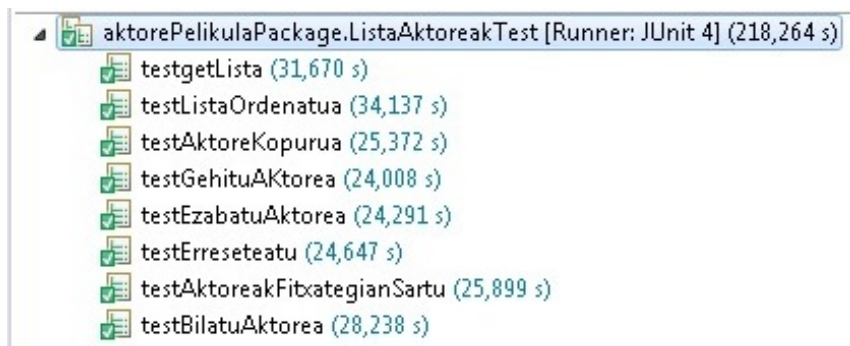
```

1 package aktorePelikulaPackage;
2 import static org.junit.Assert.*;
3 import java.io.FileNotFoundException;
4 import java.io.IOException;
5 import org.junit.After;
6 import org.junit.Before;
7 import org.junit.Test;
8 import aktorePelikulaPackage.Aktore;
9 import aktorePelikulaPackage.ListaAktoreak;
10
11 public class ListaAktoreakTest {
12

```

```
13 ListaAktoreak lista1 = ListaAktoreak.getNireListaAktoreak();
14 ListaPelikula lista2=ListaPelikula.getNireListaPelikula();
15 Aktore aktor1,aktor2;
16
17
18 @Before
19 public void setUp() throws Exception {
20     aktor1 = new Aktore("AdeiArias");
21     aktor2 = new Aktore("AnderPrieto");
22     lista2.fitxeroaErakutsi("./FilmsActors20162017.txt");
23 }
24
25 @After
26 public void tearDown() throws Exception {
27     aktor1=null;
28     aktor2=null;
29     lista1.erreseteatu();
30     lista2=null;
31 }
32
33 @Test
34 public void testEzabatuAktorea() throws FileNotFoundException, IOException {
35     assertEquals(lista1.aktoreKopurua(), 1283445);
36     lista1.ezabatuAktorea("AdeiArias");
37     assertEquals(lista1.aktoreKopurua(), 1283445);
38     lista1.ezabatuAktorea("Devon, Tony");
39     assertEquals(lista1.aktoreKopurua(), 1283444);
40 }
41
42 @Test
43 public void testGehituAktorea() throws FileNotFoundException, IOException {
44     assertEquals(lista1.aktoreKopurua(), 1283445);
45     lista1.gehituAktorea("AnderPrieto", aktor2);
46     assertEquals(lista1.aktoreKopurua(), 1283446);
47 }
48
49 @Test
50 public void testBilatuAktorea() {
51     lista1.gehituAktorea("AdeiArias", aktor1);
52     assertEquals(lista1.bilatuAktorea("AdeiArias"), aktor1);
53     assertEquals(lista1.bilatuAktorea("AnderPrieto"), null);
54     assertNotEquals(lista1.bilatuAktorea("Tarantino, Quentin"), aktor1);
55 }
56
57 @Test
58 public void testgetList() {
```

```
59     lista1.getList();
60     assertNotNull(lista1);
61 }
62
63
64 @Test
65 public void testAktoreKopurua() {
66     assertEquals(lista1.aktoreKopurua(), 1283445);
67 }
68
69 @Test
70 public void testErreseteatu() {
71     assertEquals(lista1.aktoreKopurua(), 1283445);
72     lista1.erreseteatu();
73     assertEquals((lista1.aktoreKopurua()), 0);
74 }
75
76 @Test
77 public void testListaOrdenatua() {
78     lista1.listaOrdenatua();
79     assertNotNull(lista1);
80 }
81
82 @Test
83 public void testAktoreakFitxategianSartu() {
84     lista1.aktoreakFitxategianSartu();
85 }
86 }
```



6.1.5 *ListaPelikulaTest.java*

```
1 package aktorePelikulaPackage;
2 import static org.junit.Assert.*;
3 import java.io.FileNotFoundException;
4 import java.io.IOException;
5 import org.junit.After;
6 import org.junit.Before;
7 import org.junit.Test;
8 import aktorePelikulaPackage.ListaAktoreak;
9 import aktorePelikulaPackage.ListaPelikula;
10 import aktorePelikulaPackage.Pelikula;
11
12 public class ListaPelikulaTest {
13
14     ListaPelikula lista1 = ListaPelikula.getNireListaPelikula();
15     Pelikula peli1, peli2, peli3, peli4;
16
17     @Before
18     public void setUp() throws Exception {
19         peli1 = new Pelikula("Batman", 345.00);
20         peli2 = new Pelikula("Joker", 355.00);
21         peli3 = new Pelikula("WonderWoman", 365.00);
22     }
23
24     @After
25     public void tearDown() throws Exception {
26     }
27
28     @Test
29     public void testGehituDirua() throws FileNotFoundException, IOException {
30         lista1.erreseteatu();
31         lista1.gehituPelikula(peli3);
32         lista1.gehituDirua("WonderWoman", 20.00);
33         assertEquals(peli3.getDirua(), 385.00, 2); //listako elementu bakarrari dirua
34         ↪ gehitu
35         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
36         peli4 = ListaPelikula.getNireListaPelikula().bilatuPelikula("Eager to Die");
37         lista1.gehituDirua("Eager to Die", 20.00);
38         assertEquals(peli4.getDirua(), 65.00, 2); //listako edozein elementuri dirua
39         ↪ gehitu
40     }
41
42     @Test
```

```

41 public void testPelikulaKopurua() throws FileNotFoundException, IOException {
42     lista1.erreseteatu();
43     assertEquals(lista1.pelikulaKopurua(), 0); //lista hutsaren pelikula kopurua
44     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
45     assertEquals(lista1.pelikulaKopurua(), 10412);
46     lista1.gehituPelikula(peli2);
47     assertEquals(lista1.pelikulaKopurua(), 10413); //lista ez hutsaren pelikula
         ↪ kopurua
48 }
49
50 @Test
51 public void testErreseteatu() throws FileNotFoundException, IOException {
52     lista1.erreseteatu();
53     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
54     assertEquals(lista1.pelikulaKopurua(), 10412);
55     lista1.erreseteatu();
56     assertEquals(lista1.pelikulaKopurua(), 0);
57 }
58
59 @Test
60 public void testGehituPelikula() throws FileNotFoundException, IOException {
61     lista1.erreseteatu();
62     assertEquals(ListaPelikula.getNireListaPelikula().pelikulaKopurua(), 0);
63     lista1.gehituPelikula(peli2);
64     assertEquals(ListaPelikula.getNireListaPelikula().pelikulaKopurua(),
65         ↪ 1); //gehitu elementua lista hutsean
66     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
67     assertEquals(ListaPelikula.getNireListaPelikula().pelikulaKopurua(), 10413);
68     lista1.gehituPelikula(peli3);
69     assertEquals(ListaPelikula.getNireListaPelikula().pelikulaKopurua(),
70         ↪ 10414); //gehitu elementua lista ez hutsean
71 }
72
73 @Test
74 public void testBilatuPelikula() throws FileNotFoundException, IOException {
75     lista1.erreseteatu();
76     assertEquals(ListaPelikula.getNireListaPelikula().bilatuPelikula("Eager to
77         ↪ Die"), null); //lista hutsean bilatu
78     lista1.gehituPelikula(peli3);
79     assertEquals(ListaPelikula.getNireListaPelikula().bilatuPelikula("WonderWoman"),
80         ↪ peli3); //lista elementu bakarra
81     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
82     lista1.gehituPelikula(peli1);
83     assertEquals(lista1.bilatuPelikula("Batman"), peli1); //elementua listan dago
84     assertNotEquals(lista1.bilatuPelikula("Joker"), peli1); //elementua ez dago
         ↪ listan

```

```

81     }
82
83     @Test
84     public void testFitxeroaErakutsi() throws FileNotFoundException, IOException {
85         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
86     }
87 }

```



6.1.6 PelikulaTest.java

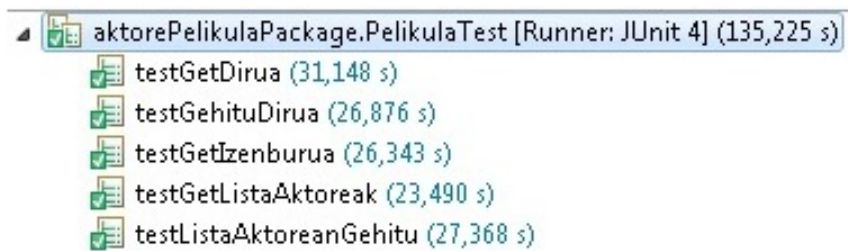
```

1  package aktorePelikulaPackage;
2  import static org.junit.Assert.*;
3  import java.io.FileNotFoundException;
4  import java.io.IOException;
5  import org.junit.After;
6  import org.junit.Before;
7  import org.junit.Test;
8  import aktorePelikulaPackage.ListaAktoreak;
9  import aktorePelikulaPackage.Pelikula;
10
11  public class PelikulaTest {
12
13      Pelikula pelikula1, pelikula2, pelikula3;
14      ArrayAktoreak lista, lista2;
15      Aktore aktor1, aktor2;
16
17      @Before
18      public void setUp() throws Exception {
19          pelikula3 = new Pelikula("La isla", 30.00);

```

```
20     pelikula2 = new Pelikula("Spiderman", 40.00);
21     lista2 = new ArrayAktoreak();
22     aktor1 = new Aktore("Arias, Adei");
23     aktor2 = new Aktore("Prieto, Ander");
24     lista2.gehituAktorea(aktor1);
25     lista2.gehituAktorea(aktor2);
26 }
27
28 @After
29 public void tearDown() throws Exception {
30 }
31
32 @Test
33 public void testGetIzenburua() throws FileNotFoundException, IOException {
34     assertEquals(pelikula2.getIzenburua(), "Spiderman");
35     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
36     pelikula1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("Mind Stroll");
37     assertEquals(pelikula1.getIzenburua(), "Mind Stroll");
38 }
39
40 @Test
41 public void testGetDirua() throws FileNotFoundException, IOException {
42     assertEquals(pelikula3.getDirua(), 30.00, 2);
43     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
44     pelikula1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("Mind Stroll");
45     assertEquals(pelikula1.getDirua(), 45.00, 2);
46 }
47
48 @Test
49 public void testGetListaAktoreak() throws FileNotFoundException, IOException {
50     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
51     pelikula1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("Mind Stroll");
52     lista = pelikula1.getListaAktoreak();
53     assertNotNull(lista);
54 }
55
56 @Test
57 public void testGehituDirua() throws FileNotFoundException, IOException {
58     pelikula3.gehituDirua(30.00);
59     assertEquals(pelikula3.getDirua(), 60.00, 2);
60     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
61     pelikula1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("La sala");
62     pelikula1.gehituDirua(10.00);
63     assertEquals(pelikula1.getDirua(), 55.00, 2);
64 }
65
```

```
66 @Test
67 public void testListaAktoreanGehitu() throws FileNotFoundException, IOException {
68     lista2.erreseteatu();
69     assertEquals(lista2.aktoreKopurua(), 0);
70     lista2.gehituAktorea(aktor1);
71     assertEquals(lista2.aktoreKopurua(), 1);
72     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
73     pelikula1 = ListaPelikula.getNireListaPelikula().bilatuPelikula("Mind Stroll");
74     lista = pelikula1.getListAktoreak();
75     assertEquals(lista.aktoreKopurua(), 6);
76     lista.gehituAktorea(aktor1);
77     assertEquals(lista.aktoreKopurua(), 7);
78 }
79 }
```



6.2 HIRUGARREN EGINKIZUNEKO JUNITAK

6.2.1 *GraphHashTest.java*

```
1 package dea3;
2
3 import static org.junit.Assert.*;
4
5 import java.util.ArrayList;
6
7 import org.junit.After;
8 import org.junit.Before;
9 import org.junit.Test;
10
11 public class GraphHashTest {
12     GraphHash grafoa;
13     ListaAktoreak lista1 = ListaAktoreak.getNireListaAktoreak();
14     ListaPelikula lista2=ListaPelikula.getNireListaPelikula();
15
16     @Before
17     public void setUp() throws Exception {
18         grafoa=new GraphHash();
19         lista2.fitxeroaErakutsi("./FilmsActors20162017.txt");
20     }
21
22     @After
23     public void tearDown() throws Exception { }
24
25     @Test
26     public void testGrafoaSortu() {
27         grafoa.grafoaSortu(ListaAktoreak.getNireListaAktoreak());
28         assertNotNull(grafoa);
29     }
30
31     @Test
32     public void testKonektatuta() {
33         grafoa.grafoaSortu(ListaAktoreak.getNireListaAktoreak());
34         assertTrue(grafoa.konektatuta("Devon, Tony","Nutcher, Greg"));
35         assertFalse(grafoa.konektatuta("Devon, Tony","Malas, Javi"));
36         assertFalse(grafoa.konektatuta("Pearson, Liam","Bardem, Javier"));
37         assertFalse(grafoa.konektatuta("Neeson, Liem","Pit, Brad"));
38         assertFalse(grafoa.konektatuta("Aho, Miina","Pitt, Brad"));
39     }
```

```

40
41 @Test
42 public void testErlazionatuta() {
43     grafoa.grafoaSortu(ListaAktoreak.getNireListaAktoreak());
44     ArrayList<String> lista = new ArrayList<String>(); //hau erabiliko dugu probetan
45     ↪ lista hutsa bat balitz bezala, ez diogulako ezer sartu
46     assertNotNull(grafoa.erlazionatuta("Devon, Tony", "O'Toole, Peter (I)"));
47     assertEquals(grafoa.erlazionatuta("Devon, Tony", "Maalas, Javii"), lista);
48     assertEquals(grafoa.erlazionatuta("Peearson, Liiam", "Bardem, Javier"), lista);
49     assertEquals(grafoa.erlazionatuta("Neeson, Liem", "Pit, Brad"), lista);
50     assertEquals(grafoa.erlazionatuta("Aho, Miina", "Pitt, Brad"), lista);
51 }

```

PRAKTIKAREN EXEKUZIOAREN ADIBIDEAREN HASIERA:

FITXATEGIA KARGATZEN ARI DA
25.505 segundu behar izan ditu

GRAFOA SORTZEN ARI DA
0.878 segundu behar izan ditu

GRAFOA KARGATU DA

KONEKTATU METODOA:

BI AKTOREAK PELIKULA BERDINEAN DAUDENEAN, TRUE EMAN BEHAR DU
true
0.01 segundu behar izan ditu

BI AKTOREAK EZ DAUDE PELIKULA BERDINEAN, BAINA KONEKTATUTA DAUDE
true
1.309 segundu behar izan ditu

BI AKTOREAK EZ DAUDE KONEKTATUTA, FALSE EMANGO DU
false
1.623 segundu behar izan ditu

ORAIN IKUSIKO DUGU AKTOREAK ALEATORIOKI 100 ALDIZ HARTUZ ZENBAT KONEKTATUTA DAUDEN:

Konektatu metodoa 100 aldiz exekutatuta 90 true izan dira

90.829 segundu behar izan ditu

ERLAZIONATU METODOA:

BI AKTOREAK PELIKULA BERDINEAN DAUDENEAN

EMAITZA INPRIMATUKO DUGU:

Devon, Tony

Eager to Die

O'Toole, Peter (I)

0.017 segundu behar izan ditu

BI AKTOREAK EZ DAUDE PELIKULA BERDINEAN, BAINA ERLAZIONATUTA DAUDE

EMAITZA INPRIMATUKO DUGU:

Devon, Tony

>Ed

Martin, Ivan (I)

>You Can't Win

Mark, Henry

>STS FA: The Complete Actor

Ruoss, Anya

2.552 segundu behar izan ditu

BI AKTOREAK EZ DAUDE KONEKTATUTA

Aktoreak ez daude konektatuta

1.231 segundu behar izan ditu

ORAIN IKUSIKO DUGU AKTOREAK ALEATORIOKI 10 ALDIZ HARTUZ, KONEKTATUTA BALEUDE, HAUEN ERLAZIOA AGERTUKO LITZATEKE:

ORAIN EZ DUGU INPRIMATUKO, DENBORA GEHIAGO BEHARKO ZUKEELAKO

20.607 segundu behar izan ditu 10 erlazioak gauzatzeko

7. Ondorioak

Lehenik eta behin, eginkizun honetan *HashMap*, pilak, ilarak, *HashSet*-a, *Array*-ak eta *ArrayList*-ak modu argi batean erabili ditugu.

Gainera, datu egitura bakoitzaren desberdintasunak argi geratu zaizkigu, adibidez, pila eta ilara baten artean.

Hori gutxi balitz, grafoa sortzerakoan, *HashMap* erabiltzea oso ideia bikaina izan da, bilaketak kostu konstantean egiten dituelako.

Are gehiago, konektatuta metodoan, aztertuak *HashSet* bat da, behin eta berriz elementuak aztertuak izan diren ala ez ikusi behar direlako.

Bukatzeko, eginkizun honen helburua grafo bat sortzea eta horrekin bilaketak egitea izan da; horregatik, une oro eraginkortasunari garrantzia eman diogu.

Erreferentziak

- [1] Gojenola, Koldo. Datu-Egiturak eta Algoritmoak: proiektua – Aktoreak eta pelikulak kudeatu – 3. eginkizuna (*GraphHash*). egela.ehu.eus, 2019. URL https://egela.ehu.eus/pluginfile.php/2282884/mod_resource/content/10/Praktika%202019-2020%20ikasturtea-Fase3-euskaraz.pdf.
- [2] Oracle. HashMap (Java Platform SE 7). docs.oracle.com, 2018. URL <https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html>.
- [3] Oracle. HashSet (Java Platform SE 7). docs.oracle.com, 2018. URL <https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html>.