

HIRUGARREN PRAKTIKA

GRAFOA

November 23, 2019

Partaideak:

Kerman Sanjuan

Ander San Juan

Josu Ferreras

Aurkibidea

1	Sarrera	4
2	Dokumentazioa	5
2.1	Klaseen diseinua	5
2.2	Datu egitura nagusien azalpena	6
2.2.1	Pelikula lista	6
2.2.2	Aktore lista	6
2.3	Aztertutako aukeren deskribapena eta hartutako soluzioen deskribapen orokorra	6
2.3.1	Hasierako ideia	6
2.3.2	Amaierako ideia	7
2.4	Metodo garrantzitsuenen azalpena	7
2.4.1	Scannera klasean	7
2.4.2	ListaPelikula klasean	8
2.4.3	Pelikulak klasean	9
2.4.4	ListaAktore klasean	9
2.4.5	Aktore klasean	9
2.4.6	GraphHash klasean	10
2.5	Metodo nagusien diseinua eta inplementazioa	10
2.5.1	ListaPelikulak klasean	10
2.5.1.1	gehituPelikula()	10
2.5.1.2	badagoPelikula()	10
2.5.1.3	itzuliPelikula()	11
2.5.2	Pelikulak klasean	12
2.5.2.1	gehituAktore()	12
2.5.3	ListaAktore klasean	12
2.5.3.1	gehituAktorea()	12
2.5.3.2	aktoreaDago()	12

2.5.3.3	aktoreOrdenatuak()	13
2.5.3.4	ezabatuAktorea()	13
2.5.4	Aktore klasean	14
2.5.4.1	badagoPelikula()	14
2.5.4.2	ikusiPelikulak()	15
2.5.5	GraphHash klasean	16
2.5.5.1	grafoaSortu()	16
2.5.5.2	konektatuta()	17
2.6	Programak duen 9 aukeren azalpena	18
2.6.1	1. aukera: Aktore baten pelikula guztiak lortu	18
2.6.2	2. aukera: Aktoreen lista ordenatua lortu	19
2.6.3	3. aukera: Aktore berri bat gehitu	19
2.6.4	4. aukera: Pelikula baten aktoreak lortu	20
2.6.5	5. aukera: Pelikula bati dirua gehitu.	20
2.6.6	6. aukera: Aktore bat ezabatu	21
2.6.7	7. aukera: Fitxategi bat sortu aktoreen zerrenda ordenatuarekin	21
2.6.8	8. aukera: Bi aktoreen arteko erlazioa aipatu	22
2.6.9	9. Aukera: Bi aktoreen arteko erlazioaren proba enpirikoak	23
2.7	Proben emaitza enpirikoak (denborak)	24
2.7.1	Sarrera moduan	24
2.7.2	Fitxategia irakurtzeko eta sortzeko	24
2.7.3	Aktoreen lista alfabetikoki ordenatzeko behar duen denbora	24
2.7.4	Fitxategi bat sortzeko denbora	24
2.7.5	Bilatu pelikula (kasurik txarrean)	25
2.7.6	Gainontzeko metodoak	25
2.8	Proba kasuak	25
2.8.1	Aktore klasea	25
2.8.1.1	gehituPelikula()	25
2.8.1.2	kenduPekikula()	26
2.8.1.3	badagoPelikula()	26
2.8.2	Pelikula klasea	26
2.8.2.1	gehituAktore()	26
2.8.2.2	kenduAktore()	27
2.8.3	listaAktore klasea	27
2.8.3.1	gehituAktorea	27

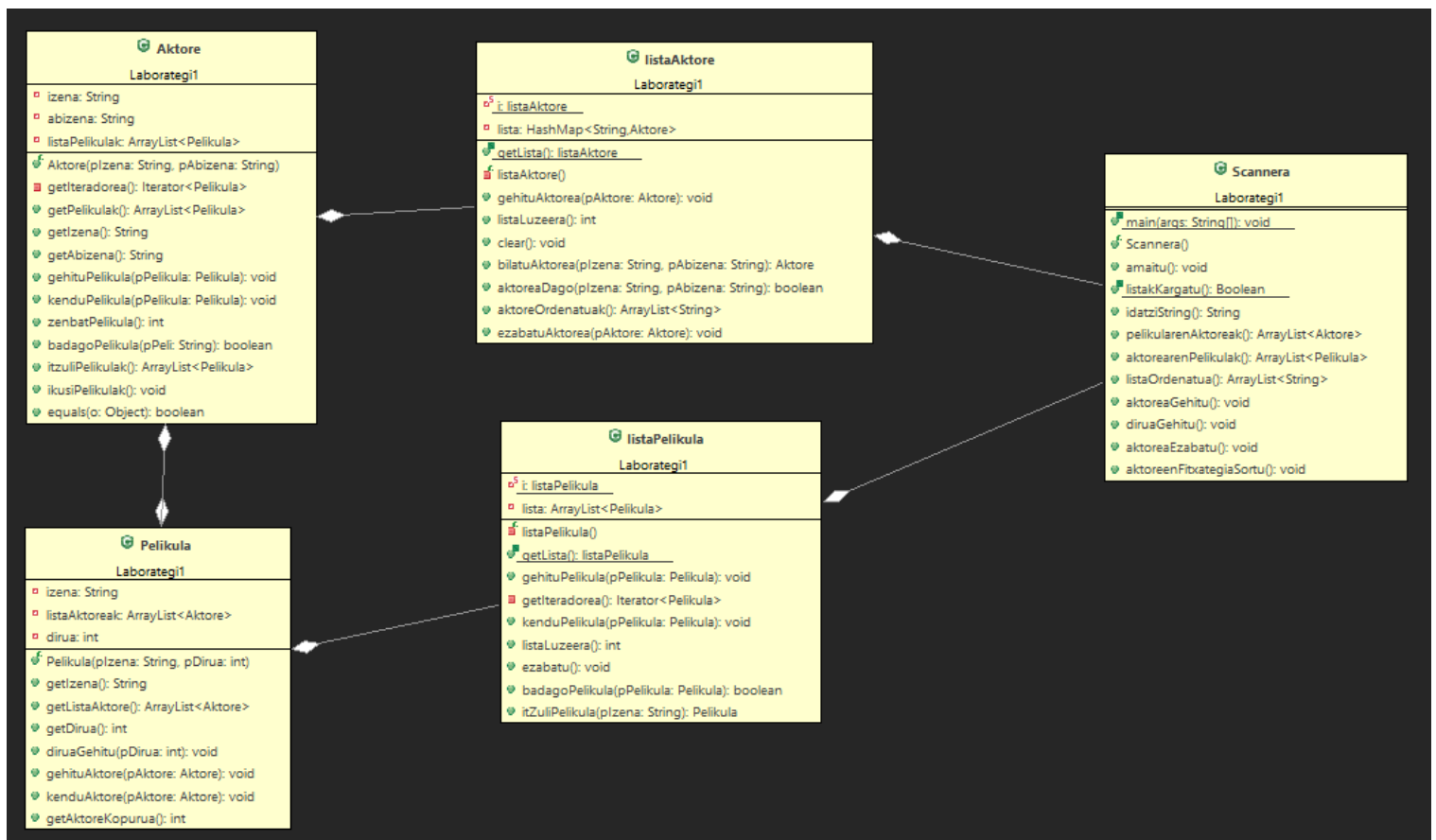
2.8.3.2	kenduAktore()	27
2.8.3.3	bilatuAktorea()	28
2.8.4	listaPelikula klasea	28
2.8.4.1	gehituPelikula	28
2.8.4.2	kenduPelikula()	28
2.8.4.3	bilatuPelikula()	29
2.8.5	GraphHash klasea	30
2.8.5.1	grafoaSortu	30
2.8.5.2	Konektatuta	30
3	Kodea	32
3.1	Scannerra	32
3.2	Aktore	45
3.3	Pelikula	48
3.4	listaPelikula	52
3.5	listaAktore	54
3.6	graphHash	58
3.7	listaAktoreTest	63
3.8	PelikulaTest	67
4	Progama irekitzen	70
5	Amaiera	72
5.1	Konklusioak	72
5.2	Bibliografia	72

Sarrera

Laborategi hau lehenengo laborategian oinarritu egin da. Bi aktoreen arteko erlazioa badagoen ala ez aipatzen duen metodoa gehitu egin da.

Dokumentazioa

2.1 KLASEEN DISEINUA



2.2 DATU EGITURA NAGUSIEN AZALPENA

Aktore lista eta pelikula lista izango dira erabiliko ditugun datu egitura nagusiak. Izan ere, hauek gabe ia ezinezkoa izango litzateke pelikula eta aktore guztiak batera kudeatzea.

2.2.1 Pelikula lista

Pelikula bakoitza aldi bakarrean soilik agertzen denez fitxategian, ez dugu pelikula hori jadanik agertu den ala ez konprobatu behar. Hori dela eta, pelikulez osatutako ArrayList bat erabiltzea aukeratu dugu.

2.2.2 Aktore lista

Pelikulak ez bezala, aktore bakoitza askotan ager daiteke fitxagian. Aktore berri bat gehitu nahi dugun bakoitzean aktore lista osoa konprobatu beharko bagenu, errepikapenak saihesteko, gure algoritmoa kostu konputazional handia edukiko luke. Horregatik HashMap erabiltzea erabaki dugu.

Dena den, bi klase hauek pelikula eta aktore guztiak dituzten listak dira. Hauetaz gain beste bi ArrayList erabiliko ditugu:

- Pelikula bakoitzak bere aktoreen lista edukiko du.

- Aktore bakoitzak bere pelikulen lista edukiko du.

2.3 AZTERTUTAKO AUKEREN DESKRIBAPENA ETA HARTUTAKO SOLUZIOEN DESKRIBAPEN OROKORRA

2.3.1 Hasierako ideia

Gure ideia, laborategiarekin hasi ginenean lanean, bi ArrayList sortzea zen (Biak EMAK). ArrayList batek pelikula guztiak edukiko zituen bere barnean, eta modu berean, pelikula bakoitzak bere izena eta pelikula osatzen zuten aktoreen lista. Beste ArrayList-ak berriz, aktore guztien lista edukiko zuen, eta, aktore bakoitzak, bere izena eta berak parte hartu izandako pelikulen lista. Laborategiarekin aurrera joaten ginen bitartean, konturatu ginen nola, aktoreak errepikatuta egonda, eraginkortasuna gero eta txikiagoa zela. Hau dela eta, ArrayList izatetik HashMap erabiltzera pasatu ginen.

2.3.2 Amaierako ideia

HashMap-aren erabilerarekin gauzak asko erraztu egin ziren. Alde batetik, konpilazio eta exekuzio denborak asko murriztu ziren, eta beste alde batetik, bere barneko metodoekin konprobazioak eta metodoak egitea askoz errezagoa izan zen.

Gainera, Scanerra egiteko momentuan, 8 aukera desberdin inplementatu genituen, erabiltzaileak programaren kontrol osoa edukitzeko.

2.4 METODO GARRANTZITSUENEN AZALPENA

Atal honetan bakarrik gure programaren metodo garrantzitsuenen azalpenak agertzen dira, aipatzen ez diren metodoen eginkizuna oso erraz interpretatzen baita haien izenak irakurtzean (getter eta setter-ak, klaseen eraikitzaile gehienak, getIterator...).

2.4.1 Scannera klasean

main(): Programaren metodo nagusia da, haren exekuzioa gauzatzean nondik hasi behar den adierazten diguna.

Scannera(): Izen berdina duen klasearen eraikitzailea da, haren barnean programa honek eskaintzen dituen 8 aukerak "case" egiturarekin inplementatuta. Aukera bakoitzeko honekin erlazionatuta dagoen metodoari deia egiten dio; Adibidez, erabiltzaileak 3. kasua (Aktore bat gehitu aktoreen zerrendara) exekutatzea aginduz gero, aktoreaGehitu() metodoari deituko zaio.

amaitu(): Eskainitako aukeren bat exekutatu gero erabiltzaileari beste edozein eginkizun egin nahi al duen galdetzen dio. Ezezko kasuan programaren exekuzioa bukatzen da.

listakKargatu(): Eskainitako fitxategiaren informazioa irakurtzeaz eta kudeatzeaz arduratzen da. Informazioaren kudeaketa aurkitutako pelikula guztiak zerrenda batean sartzean eta aurkitutako aktore guztiak beste zerrenda batean sartzean datza.

idatziString() eta idatziInt(): Metodo hauei beste metodo batzuetatik egiten zaie deia, erabiltzaileari zenbaki bat edo karaktere-kate bat idaztea eskatu behar zaion bakoitzean.

pelikularenAktoreak(): Pelikula baten izenburua sartuta, pelikula honetan parte hartu duten aktoreen ("Aktore" motatako objektuak) zerrenda itzultzen du. Sartutako pelikularen izenburua duen pelikularik ez aurkituz gero, errore mezu bat pantailaratuko da.

aktorearenPelikulak(): Aktore baten izena eta abizena sartuta, aktore horrek parte hartutako pelikulen ("Pelikula" motatako objektuak) zerrenda itzultzen du. Sartutako aktorearen izena eta abizena duen aktoreerik ez aurkituz gero, errore mezu bat pantailaratuko da.

listaOrdenatua(): Aktoreen izen eta abizenek osatzen duten String motatako elementuez osatutako alfabetikoki ordenatutako zerrenda bat bueltatzen du, ListaAktore klaseko aktoreOrdenatuak() metodoari deituz.

aktoreaGehitu(): Aktore baten izena eta abizena sartuta, programak izen eta abizen berbera duen aktore bat bilatzen du aktoreen zerrendan. Horrelakorik ez aurkituz gero, aktore berri bat sortzen du datu horiekin, eta aktoreen zerrendara gehitzen du.

diruaGehitu(): Erabiltzaileak emandako titulua duen pelikulari (existitzen bada) erabiltzailea berak idatzitako diru kantitatea gehitzen zaio pelikula aurretik zuen diruari, Pelikula klaseko diruaGehitu() metodoari deituz.

aktoreaEzabatu(): Erabiltzaileak emandako aktorearen izena eta abizenarekin datu berberak dituen aktorea ezabatzen du aktoreen zerrendatik. Horrelako izen eta abizena duen aktoreerik ez bada aurkitzen, errore mezu bat pantailaratzen da.

aktoreenFitxategiaSortu(): Aukeratutako helbidean eta aukeratutako izenarekin aktoreen zerrenda ordenatua (Scannera klaseko aktoreOrdenatuak() metodoa erabiliz) eduki bezala duen "txt" formatuko fitxategi bat sortzen du, helbide berberan izen berdineko fitxategirik ez dagoen ziurtatuz gero eta helbidea balioduna den ziurtatuz gero.

2.4.2 ListaPelikula klasean

gehituPelikula(): Pelikulen zerrendari Pelikula motatako objektu bat gehitzen zaio.

kenduPelikula(): Pelikulen zerrendatik pelikula motatako objektu bat kentzen da.

listaLuzeera(): Pelikulen zerrendak duen luzera bueltatzen du.

ezabatu(): Pelikulen zerrendatik elementu guztiak ezabatzen ditu.

badagoPelikula(): Pelikula motatako objektu bat emanda konprobatzen du elementu hori pelikulen zerrendan dagoen, "true" itzuliz baiezkotasun kasuan.

itzuliPelikula(): String motatako pelikula baten izenburua emanda pelikulen zerrendan izenburu berdina duen pelikula bilatzen du, eta pelikula hori bueltatzen du aurkitzen badu.

2.4.3 Pelikulak klasean

diruaGehitu(): Pelikula baten "dirua" aldagaia inkrementatzen da sartutako kantidadean.

gehituAktore(): Pelikulak duen aktoreen zerrendari sartutako aktorea gehitzen zaio, aktore horrek zerrendan jadanik ez badago.

kenduAktore(): Pelikulak duen aktoreen zerrendatik sartutako aktorea ezabatzen da, aktore hori aurkituz gero.

getAktoreKopurua(): Aktoreen zerrendak duen luzera bueltatzen du.

2.4.4 ListaAktore klasean

gehituAktorea(): Aktoreen zerrendari Aktore motatako objektu bat gehitzen zaio.

listaLuzeera(): Aktoreen zerrendak duen luzera bueltatzen du.

clear(): Aktoreen zerrendatik elementu guztiak ezabatzen ditu.

bilatuAktorea(): Aktore baten izena eta abizena emanda, izen eta abizen berdinak dituen Aktorea bueltatzen du, aktore hori aurkituz gero.

aktoreaDago(): Aktoreen zerrendan sartutako izen eta abizen berdinak dituen aktorea dagoen ala ez bueltatzen du, aldagai boolear moduan.

aktoreOrdenatuak(): Aktoreen zerrendan dauden aktoreetatik haien izen eta abizenak osatzen duten String motatako zerrenda bat sortzen eta alfabetikoki ordenatzen du, ordenean prioritatea emanez zuriunei eta sistemak ezagutzen ez dituen letrei. Sortutako zerrenda bueltatzen du.

ezabatuAktorea(): Aktoreen zerrendan emandako izen eta abizena dituen aktorea aurkituz gero, ezabatzen du.

2.4.5 Aktore klasean

gehituPelikula(): Aktoreak duen pelikulen zerrendari sartutako pelikula gehitzen zaio, pelikula horrek zerrendan jadanik ez badago.

kenduPelikula(): Aktoreak duen pelikulen zerrendatik sartutako pelikula ezabatzen da, pelikula hori aurkituz gero.

zenbatPelikula(): Pelikulen zerrendak duen luzera bueltatzen du.

badagoPelikula(): Pelikula baten izenburua sartuta, pelikulen zerrendan izenburu berdina duen pelikula aurkituz gero egiazko balioa bueltatuko du aldagai boolear moduan.

itzuliPelikulak(): Pelikula motatako objektuez osatutako zerrenda bueltatzen du.

ikusiPelikulak(): Aktorearen pelikulen zerrendan dauden pelikula guztien izenburuak pantailaratzen ditu.

2.4.6 GraphHash klasean

grafoaSortu(): Aktoreen lista emanda, honek "grafo bat sortu egiten du", hau da, HashMap-baten barnean aktoreak eta pelikulak sartu egiten ditu.

konektatuta(): Bi aktore edo pelikulen izena emanda, hauen arteko "konexiorik" bada goen a la ez esaten du.

2.5 METODO NAGUSIEN DISEINUA ETA INPLEMENTAZIOA

2.5.1 ListaPelikulak klasean

2.5.1.1 gehituPelikula()

//Aurre: pPelikula ez-hutsa izatea

//Post: pPelikula gure listan egotea.

```
public void gehituPelikula(Pelikula pPelikula) {  
    lista.add (pPelikula)  
}
```

Programa honen kostua $O(1)$ - ekoa dela esan daiteke.

2.5.1.2 badagoPelikula()

//Aurre: Gure lista gutxienez elementu bat dauka eta pPelikula ez-nulua da.

//Post: True itzuliko du baldin pPelikula listan badago, bestela false.

```
public boolean badagoPelikula(Pelikula pPelikula){
```

```
        itr = this.getIteradorea
        bitartean (itr.HurrengoaDu) loop
            p = itr.next
            baldin (p.izena.Equals(pPelikula.izena)) orduan
                return true
        ambitartean
        return false
    ]
```

Programa honen kostua $O(n)$ -koa da, non n listaren luzeera den.

2.5.1.3 itzuliPelikula()

//Aurre: Gure lista ez dago hutsik.

//Post: Izen bereko pelikula itzuliko du, eta ez badu aurkitzen, null itzuliko du.

```
public Pelikula itzuliPelikula(String pIzena){
    aurkitua = false
    itr = this.getIteradorea
    Pelikula p = itr.next
    bitartean (itr.hurrengoaDu eta ezAurkitua) loop
        baldin (itr.izena == pIzena) orduan aurkitua = true
        bestela itr.next
    ambitartean
    baldin(aurkitua) return true
    bestela return false
}
```

Metodo honen kostua $O(n)$ -koa da, non n listaren luzeera den.

2.5.2 Pelikulak klasean

2.5.2.1 gehituAktore()

```
public void gehituAktore(Aktore pAktore)[  
    baldin(listaHutsaDa) orduan listaAktoreak = new ArrayList<Aktorea>  
    bestela listariGehitu(pAktore)  
]
```

Metodo honen kostua $O(n)$ -koa da ArrayList-ak bere luzeera bikoiztu behar duenengan, baina hori n alditan behin gertatzen denez, kostua n/n edo $O(1)$ da, non n listaren luzeera den.

2.5.3 ListaAktore klasean

2.5.3.1 gehituAktorea()

```
//Post: pAktorea aktoreen listan gehitu da.  
public void gehituAktore(Aktore pAktore)[  
    baldin(listaAktorea == null) orduan  
        sortuAktoreenListaBerria  
    bestela  
        gehituAktorea(pAktore) ]
```

Metodo honen kostua $O(1)$ da.

2.5.3.2 aktoreaDago()

```
//Aurre: Ez hutsa den lista bat.  
//Post: pAktorea aktorea badago, true itzuliko du, bestela false.  
public boolean aktoreaDago(String pIzena)[  
    return this.lista.badaukaKey(pIzena)  
]
```

Metodo honen kostua $O(1)$ da, hashMap bat delako.

2.5.3.3 aktoreOrdenatuak()

//Aurre: Aktore lista ez huts bat.

//Post: Lista ordenatuta alfabetikoki.

```
public ArrayList<String> aktoreOrdenatuak[  
    lista = new ArrayList<lista.keySet()>  
    return mergeSort(lista)  
]
```

Metodo honen kostua $O(n\log(n))$ da, non n listaren luzeera da, Collections.sort mergeSort motako ordenazio algoritmoa erabiltzen du eta.

2.5.3.4 ezabatuAktorea()

```
public void ezabatuAktorea(Aktore pAktore)[  
    this.lista.remove(pAktore.getIzena)  
]
```

Metodo honen kostua $O(1)$ da, HashMap bat erabiltzen delako.

2.5.4 Aktore klasean

2.5.4.1 badagoPelikula()

//Aurre: —

//Post: Aktorea pelikula horretan parte hartu badu true itzuliko du, bestela false.

```
public boolean badagoPelikula(String pPeli){  
    itr = this.getIteradotea  
    pelikula p  
    bitartean(itr.hurrengoaDu)[  
        p = itr  
        baldin (p.izena.equals(pPeli) orduan return true  
        bestela  
        p = itr.next  
        ambaldin  
    ]  
    return false  
}
```

Metodo honen kostua $O(n)$ da, kasurik txarrean lista osoa errekorritu behar duelako.

2.5.4.2 ikusiPelikulak()

```
//Aurre: —  
//Post: Aktorearen pelikulak itzultzen ditu.  
public void ikusiPelikulak()[  
    itr = this.getIteradorea  
    baldin itr.duHurrengoa[  
        Pelikula p  
        bitartean (itr.duHurrengoa) loop  
            p = itr.next  
            printeatu(p.getIzena)  
    ambaldin  
]  
]
```

Metodo honen kostua $O(n)$ da, lista osoa igaro behar duelako.

2.5.5 GraphHash klasean

2.5.5.1 grafoaSortu()

//Aurre: Hutsa ez den aktoreen lista.

//Post: Hash-map bat itzuliko du, non aktoreak eta pelikulak agertuko diren.

```
public void grafoaSortu(listaAktore lAktoreak) [  
    itr = lAktoreak.getIteradorea  
    bitartean(itr.hurrengoaDu) [  
        String aktorea = itr.next  
        Aktore unekoa = lAktoreak.bilatuAktorea(aktorea)  
        ArrayList<String> listaP  
        iteratorPeli = unekoa.getPelikulak.iterator  
        bitartean(iteratorPeli.hurrengoaDu) [  
            String unekoP = iteratorPeli.next.getIzena  
            listaP.add(unekoP)  
            ArrayList<String> pelikularenAkt  
            baldin(this.grafo.containsKey(unekoP)) [  
                pelikularenAkt=this.grafo.get(unekoP)  
            ]bestela [  
                pelikularenAkt=new ArrayList<String> ]  
            pelikularenAkt.add(aktorea)  
            grafo.put(unekoP, pelikularenAkt) ]  
        grafo.put(aktorea, listaP) ]  
    ]
```

Metodo honen kostua $O(m*n)$ da, n "itr" eta m "iteratorPeli" izanda. N-ren elementu bakoitzeko m zerrenda osoa zeharkatzen da.

2.5.5.2 konektatuta()

//Aurre: String motako a1 eta a2 elementuak.

//Post: Eraitza true izango da a1 eta a2 lotzen dituen erlazioen kate bat baldin badago.

```
public boolean konektatuta(String a1, String a2) [  
    HashSet<String> aztertuak  
    Queue<String> aztGabeak  
    aztGabeak.add(a1)  
    baldin(a1.equals(a2)) return true  
    baldin(grafo.hutsaDa) return false  
    bitartean(!aztGabeak.hutsaDa) [  
        baldin(!aztertuak.contains((String)aztGabeak.peek)) [  
            baldin(aztGabeak.peek.equals(a2)) return true  
            bestela [  
                String uneko = aztGabeak.peek  
                aztertuak.add(uneko)  
                aztGabeak.poll  
                aztGabeak.addAll(grafo.get(uneko)) ]  
            ]bestela [  
                aztGabeak.poll ] ]  
    return false  
]
```

Metodo honen kostua $O(n)$ da, kasurik txarrean grafoaren elementu guztiak begiratu beharko liratekeelako.

2.6 PROGRAMAK DUEN 9 AUKEREN AZALPENA

```
C:\Users\elsan>cd \Users\elsan\Desktop
C:\Users\elsan\Desktop>java -Xmx1g -jar Laborategi1.jar
Pelikula eta aktoreak kargatzen daude, prozesu honek ez luke denbora luzerik hartu behar
1283334 aktore ezberdin daude.
238978 pelikula daude.
40 segundu behar izan dira.
Zer egin nahiko zenuke?
1. aukera: Aktore baten pelikula guztiak lortu
2. aukera: Aktoreen lista ordenatua lortu
3. aukera: Aktore berri bat gehitu
4. aukera: Pelikula baten aktoreak lortu
5. aukera: Pelikula bati dirua gehitu.
6. aukera: Aktore bat ezabatu
7. aukera: Fitxategi bat sortu aktoreen zerrenda ordenatuarekin
```

2.6.1 1. aukera: Aktore baten pelikula guztiak lortu

Aukera honek, aktorearenPelikulak() metodoari dei egiten dio.

//Aurre: —

//Post: Aktorearen pelikulen lista bueltatzea.

izena=idatziString()

ArrayList<Pelikula> =listaAktore.getLista.bilatuAktore(izena).getPelikula

Metodo honek, aktore baten pelikulak itzuliko ditu, aktorea ez bada aurkitzen, salbuespen bat bueltatuko da, errore mezu batekin. Metodo honen kostua $O(1)$ izango da, izan ere, HashMap baten bilatzen ari gara aktore bat eta horren kostua konstantea da.

2.6.2 2. aukera: Aktoreen lista ordenatua lortu

Izenak dioen moduan, metodo honek aktoreen zerrenda alfabetikoki ordenatuta itzuliko du.

```
//Aurre: —
```

```
//Post: Aktoreen lista ordenatua itzuli.
```

```
system.out.println("Modu efektiboan lortu da lista ordenatua")
```

```
return listaAktore.getlista().aktoreOrdenatuak()
```

Aurretik azalduta, baina berriro ere, merge-sort erabiliz alfabetikoki ordenatzen du. Metodo honek $O(n\log(n))$ kostua izango du, non n aktore listaren luzeera izango den.

2.6.3 3. aukera: Aktore berri bat gehitu

Modu erraz batean, gu idatzitako izen eta abizen baten bidez, aktorea jartzen du listan. Listan jadanik badago, ez du listan gehituko.

```
//Aurre: —
```

```
//Post: Aktorea jadanik listan ez badago, aktorea listara gehitu. Bestela, ezer ez egin.
```

```
system.out.println("Idatzi izena")
```

```
izena=idatziString()
```

```
baldin(!lista.aktoreadago(izena)) {
```

```
    lista.gehituAktorea(new Aktore(izena))
```

```
}
```

Metodo honen kostua $O(1)$ izango da, hashMap batean gehitzen gaudelako aktorea.

2.6.4 4. aukera: Pelikula baten aktoreak lortu

Lehenengo aukeraren antzekoa, baina honek, pelikula baten aktoreak itzuliko ditu.

//Aurre: —

//Post: Pelikularen aktoreen lista bueltatzea.

```
system.out.println("Idatzi izena")
```

```
peIiIzena=idatziString()
```

```
ArrayList<Aktore> lista=listaPelikula.itzuliPelikula(peIiIzena).getlistaAktore()
```

```
system.out.println(lista.size() + " aktore ditu pelikulak")
```

```
return aktoreak
```

Metodo honen kostua $O(n)$ izango da, non n Pelikula listaren luzeera den.

2.6.5 5. aukera: Pelikula bati dirua gehitu.

Hasieran, metodo honek, erabiltzaileak esandako pelikulari diru kantitate bat gehituko dio. Pelikularen izena ez badago edo diru kantitatea egokia ez bada, errore mezu bat aterako da, eta berriko eskatuko du.

//Aurre: Nahi dugun pelikula listan egotea.

//Post: Lista horri diru hori gehitu izatea.

```
system.out.println("Idatzi izena")
```

```
izena=idatziString()
```

```
baldin(listaPelikula.pelikulaDago(izena)
```

```
    system.out.println("Idatzi diru kopurua")
```

```
    dirua=idatziInt()
```

```
    listaPelikula.getPelikula(izena).diruaGehitu(dirua)
```

```
else
```

```
    system.out.println("Pelikula hori ez da existitzen")
```

```
metodo honi berriro dei egin
```

Metodo honen kostua $O(n+n)$ izango da, beraz $O(n)$, non n Pelikula listaren luzeera den.

2.6.6 6. aukera: Aktore bat ezabatu

Izenak dioen bezala, guk esandako aktorea listatik ezabatuko du. Aktorea ez badago, izena berriro eskatuko du.

```
//Aurre: — (ez da beharrezkoa aktorea existitzea)

//Post: Aktore hori listan ez egotea.

system.out.println("Idatzi izena")

izena = idatziString()

baldin(listaAktore.aktoreaDago(izena)

    listaAktore.ezabatuAktorea(new Aktore(izena)

else

    system.out.println("Aktorea ez da existitzen, berriro sartu datuak")
```

Metodo honen kostua $O(n)$ izango da, aktoreak HashMap baten daudelako eta aktore bat aurkitzeko denbora konstantea delako.

2.6.7 7. aukera: Fitxategi bat sortu aktoreen zerrenda ordenatuarekin

Aukera honetan Scannera klaseko aktoreenFitxategiaSortu() metodoari deitzen zaio.

Metodo honek dei egiten dio ListaAkore klaseko aktoreOrdenatuak() metodoari, aktoreen izen-abizenak alfabetikoki ordenatuta duen zerrenda bat lortuz. Hau egin eta gero "txt" formatuko fitxategi bat sortzen da erabiltzaileak eskainitako helbidean berak idatzitako izenarekin, helbidea

```
C:\\Users\\IZENA\\Desktop
```

formatuan eman behar da.

Behin fitxategia arazorik gabe sortu dela ziurtatuz gero, fitxategian idazten dira lehen lortutako zerrendak dituen datuak, bata albokoengandik bereizita. Azkenik mezu bat pantailaratzen da, fitxategia arazorik gabe sortu den ala ez esanez.

```
//Aurre: Aktore lista ez hutsa izatea.  
  
//Post: Fitxategi bat sortu izatea aktoreen izenekin (alfabetikoki ordenatua).  
  
ArrayList<String> listaOrdenatua = listaAktore.aktoreaOrdenatuak()  
  
system.out.println("Idatzi fitxategiaren izena")  
  
izena = idatziString()  
  
system.out.println("Idatzi non gorde behar den fitxategia")  
  
helbidea = idatziString()  
  
bitartean(aktoreak daude listan)  
  
    fitxategian aktore berria idatzi  
  
system.out.println("Fitxategia sortu da")
```

Metodo honen kostua $O(n+n)$ izango da (n aktore ordenatuak lortzeko eta beste n bat lista oso irakurtzeko), beraz $O(n)$. Klasean azaldu egin zen moduan, metodo honek, lista zaitu egiten du, ardatz "aleatorio" batekin, eta ezkerreko aldea ordenatu egiten du, ondoren, modu errekurtsiboan behin eta berriro deituz, listaren tamaina txikituz. Beste modu batean esanda, lista zatika ordenatu egiten du, gero batu zatiak, eta berriro egin.

2.6.8 8. aukera: Bi aktoreen arteko erlazioa aipatu

Aukera honek, bi aktoren izena emanda, hauen arteko erlazioa existitzen den ala ez esango du.

```
//Aurre: String motako a1 eta a2 elementuak  
  
//Post: Eraitza true izango da a1 eta a2 lotzen dituen erlazioen kate bat baldin badago  
  
GraphHash nodoak = new GraphHash()  
  
nodoak.grafoaSortu(listaAktore.getLista())  
  
System.out.println("Idatzi lehenengo elementuaren izena:")  
  
a1 = this.idatziString()  
  
System.out.println("Idatzi bigarren elementuaren izena:")  
  
a2 = this.idatziString()
```

```
boolean emaitza = nodoak.konektatuta(a1, a2)
```

```
baldin(!emaitza)
```

```
    System.out.println("Ez dago biderik " + a1 + " eta " + a2 + "-ren artean")
```

```
bestela
```

```
    System.out.println("Bide bat dago " + a1 + " eta " + a2 + "-ren artean")
```

Metodo honen kostua $O(n)$ da, kasurik txarrean grafoaren elementu guztiak begiratu beharko liratekeelako.

2.6.9 9.Aukera: Bi aktoreen arteko erlazioaren proba enpirikoak

Aukera honek, 8-garren aukeraren ehun proba egiten ditu, bataz besteko denboraren estimazio bat egiteko.

```
for(ehun alditan)
```

```
    double r1 = BalioAleatoriobat(0 tik graphMap-en luzeera arte)
```

```
    double r2 = BalioAleatoriobat(0 tik graphMap-en luzeera arte)
```

```
    z1 = r1-en balio borobildua (int) Math.round(r1));
```

```
    z2 = r2-en balio borobildua (int) Math.round(r2));
```

```
    String randomValue1 = values[z1].toString();
```

```
    String randomValue2 = values[z2].toString();
```

```
    hasiera = system.milis
```

```
    baldin(konektatuta(randomValue1,randomValue2)) orduan
```

```
        t++
```

```
bestela
```

```
        f++
```

```
ambaldin
```

```
amaiera = system.milis
```

```
denbora = denbora+(amaiera-hasiera)
```



```
amFor sysout(t+"Egiazko kasu")
```

```
sysout(f+"Egiazko kasu")
```

```
sysout(denbora/100+"Milisegundu behar dira mediaz erlazio bakoitzeko");
```

Metodo honen kostua, $O(n)$ da, beran $O(n)$ kostuko metodo bati 100 aldiz deitzen zaiolako.

2.7 PROBEN EMAITZA ENPIRIKOAK (DENBORAK)

2.7.1 Sarrera moduan

Metodo bakoitzak behar duen denbora kalkulatzeko, metodo bakoitzari timer bat inplementatu diogu modu honetan:

```
long startTime = System.nanoTime() //Kodea hemendik aurrera.
```

```
long endTime = System.nanoTime()
```

```
long elapsed =endTime-startTime
```

```
System.out.println(timeElapsed / 1000000000 + " segundu behar izan dira. " );
```

2.7.2 Fitxategia irakurtzeko eta sortzeko

Gure programak, fitxategia irakurtzeko, eta hortik bi listak sortzeko behar duen denbora 31 segundukoa da.

2.7.3 Aktoreen lista alfabetikoki ordenatzeko behar duen denbora

Aktoreen lista hashMap bat izanda, metodo honek bakarrik segundu bat beharko du.

2.7.4 Fitxategi bat sortzeko denbora

Fitxategiaren izena eta bere helbidea pegatu dira, modu honetan, emaitza zehatzagoa izateko. Metodoak, guztira, 8 segundu behar izan ditu lista ordenatu eta fitxategia sortzeko.

2.7.5 Bilatu pelikula (kasurik txarreanean)

Pelikula asko daudenez, kasurik txarreanean, azken pelikula bilatuko dugu. Kasu hone-tan, ere segundo bat behar izan du (borobilduta, kontagailua segundotan baitago).

2.7.6 Gainontzeko metodoak

Hauek dira denbora gehien behar duten metodoak. Bestiak, gehien bat, atributu baten aldaketan edo bilaketa batean datzate. Hau dela eta, denborak oso laburrak izaten dira (segundo bat baino txikiagoak + HashMap-en erabilera), eta ez ditugu kontuan izan.

2.8 PROBA KASUAK

Programaren exekuzioan gure aplikazioak kasu kritikoetan modu eraginkor batean erantzuteko eta arazoak ekiditzeko, metodo aipagarrien proba kasuen analisia egin behar da kodea idazten hasi baino lehen. Horretarako, metodo horietan kasu kritikoak aztertu behar ditugu.

2.8.1 Aktore klasea

2.8.1.1 gehituPelikula()

listaPelikula	Sarrera	listaPelikula (emaitza)
[]	a	[a]
[a]	b	[a,b]
[a]	a	[a,a]
[a,b,c,d]	e	[a,b,c,d,e]
[a,b,c,d]	a	[a,b,c,d,a]
[a,b,c,d]	c	[a,b,c,d,c]
[a,b,c,d]	d	[a,b,c,d,d]

2.8.1.2 kenduPelikula()

listaPelikula	Sarrera	listaPelikula (emaitza)
[]	a	[]
[a]	b	[a]
[a]	a	[]
[a,b,c,d]	e	[a,b,c,d]
[a,b,c,d]	a	[b,c,d]
[a,b,c,d]	c	[a,b,d]
[a,b,c,d]	d	[a,b,c]

2.8.1.3 badagoPelikula()

listaPelikula	Sarrera	Emaitza
[]	"Kima"	False
["Kima"]	"Brahmastram"	False
["Kima"]	"Kima"	True
["Kima", "All City", "Unforgiven", "Brahmastram"]	"Geschichten aus dem Lepratal"	False
["Kima", "All City", "Unforgiven", "Brahmastram"]	"Kima"	True
["Kima", "All City", "Unforgiven", "Brahmastram"]	"Unforgiven"	True
["Kima", "All City", "Unforgiven", "Brahmastram"]	"Brahmastram"	True
["Kima", "All City", "Unforgiven", "Brahmastram"]	"unforgiven"	False
["Kima", "All City", "Unforgiven", "Brahmastram"]	"Unforgiben"	False

2.8.2 Pelikula klasea**2.8.2.1 gehituAktore()**

listaAktore	Sarrera	listaAktore (emaitza)
[]	a	[a]
[a]	b	[a,b]
[a]	a	[a,a]
[a,b,c,d]	e	[a,b,c,d,e]
[a,b,c,d]	a	[a,b,c,d,a]
[a,b,c,d]	c	[a,b,c,d,c]
[a,b,c,d]	d	[a,b,c,d,d]

2.8.2.2 kenduAktore()

listaAktore	Sarrera	listaAktore (emaitza)
[]	a	[]
[a]	b	[a]
[a]	a	[]
[a,b,c,d]	e	[a,b,c,d]
[a,b,c,d]	a	[b,c,d]
[a,b,c,d]	c	[a,b,d]
[a,b,c,d]	d	[a,b,c]

2.8.3 listaAktore klasea**2.8.3.1 gehituAktorea**

listaAktore	Sarrera	listaAktore (emaitza)
[]	a	[a]
[a]	b	[a,b]
[a]	a	[a]
[a,b,c,d]	e	[a,b,c,d,e]
[a,b,c,d]	a	[a,b,c,d]
[a,b,c,d]	c	[a,b,c,d]
[a,b,c,d]	d	[a,b,c,d]

2.8.3.2 kenduAktore()

listaPelikula	Sarrera	listaAktore (emaitza)
[]	a	[]
[a]	b	[a]
[a]	a	[]
[a,b,c,d]	e	[a,b,c,d]
[a,b,c,d]	a	[b,c,d]
[a,b,c,d]	c	[a,b,d]
[a,b,c,d]	d	[a,b,c]

2.8.3.3 bilatuAktorea()

listaAktore	Sarrera	Eraitza
[]	a	False
[a]	b	False
[a]	a	True
[a,b,c]	d	False
[a,b,c,d]	a	True
[a,b,c,d]	c	True
[a,b,c,d]	d	True
[a,b,c,d]	e	False

2.8.4 listaPelikula klasea**2.8.4.1 gehituPelikula**

listaPelikula	Sarrera	listaPelikula (emaitza)
[]	a	[a]
[a]	b	[a,b]
[a]	a	[a]
[a,b,c,d]	e	[a,b,c,d,e]
[a,b,c,d]	a	[a,b,c,d,a]
[a,b,c,d]	c	[a,b,c,d,c]
[a,b,c,d]	d	[a,b,c,d,d]

2.8.4.2 kenduPelikula()

listaPelikula	Sarrera	listaPelikula (emaitza)
[]	a	[]
[a]	b	[a]
[a]	a	[]
[a,b,c,d]	e	[a,b,c,d]
[a,b,c,d]	a	[b,c,d]
[a,b,c,d]	c	[a,b,d]
[a,b,c,d]	d	[a,b,c]

2.8.4.3 bilatuPelikula()

listaPelikula	Sarrera	Eraitza
[]	a	False
[a]	b	False
[a]	a	True
[a,b,c]	d	False
[a,b,c,d]	a	True
[a,b,c,d]	c	True
[a,b,c,d]	d	True
[a,b,c,d]	e	False

2.8.5 GraphHash klasea

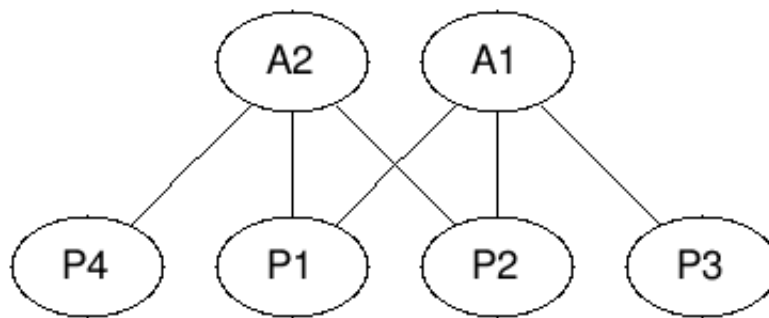
2.8.5.1 grafoaSortu

Lista honakoa da:

Aktore 2 [Pelikula 4, Pelikula 1, Pelikula 2]

Aktore 1 [Pelikula 1, Pelikula 2, Pelikula 3]

Metodo hau erabili hostean honakoa da grafoa:



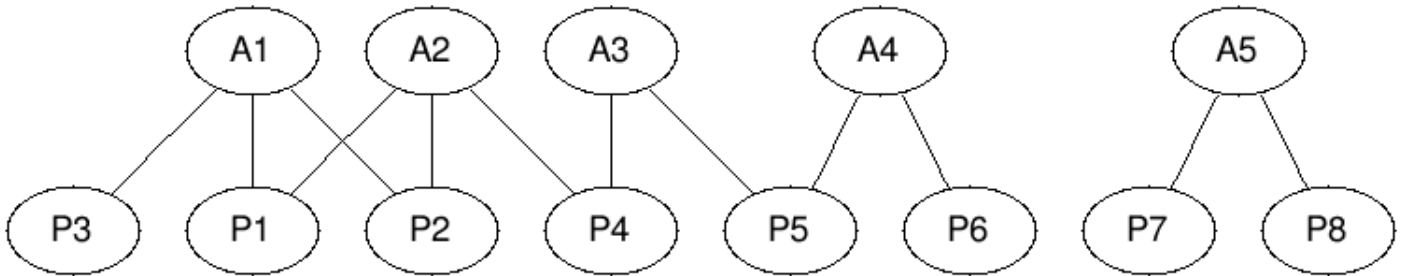
Nodoa	Eraitza (contains metodoa erabiliz)
P1	True
P2	False
P3	True
P4	True
A1	True
A2	True
A3	False

2.8.5.2 Konektatuta

Grafoa hutsik dagoenean:

a1	a2	Eraitza
A1	A4	False
A1	P4	False
P3	P4	False
A1	P6	False

Grafoan elementuak kargatu eta gero:



a1	a2	Emaizta
A1	A4	True
A1	P4	True
P3	P4	True
A1	P6	True
A5	P7	True
A1	P7	False
P7	P1	False
A1	P9	False

Kodea

3.1 SCANNERRA

```
1 package packlaborategi3;
2 import java.io.File;
3 import java.io.FileWriter;
4 import java.io.BufferedWriter;
5 import java.io.IOException;
6 import java.util.concurrent.TimeUnit;
7 import java.util.*;
8
9
10
11 public class Scannera {
12
13     public static void main(String[] args) throws Exception {
14         Boolean kargatuta=false;
15         listaAktore.getList().clear();
16         System.out.println("Pelikula eta aktoreak kargatzen daude, prozesu
17             ↳ honek ez luke denbora luzerik hartu behar");
18         while (!kargatuta) {
19             kargatuta=listakKargatu();
20         }
21         new Scannera();
22     }
23 }
```

```

21 }
22 //----- ELEGIR LAS OPCIONES
23     ↪ -----
24     public Scannera() throws Exception {
25         System.out.println("Zer egin nahiko zenuke?");
26         System.out.println("1. aukera: Aktore baten pelikula guztiak
27             ↪ lortu");
28         System.out.println("2. aukera: Aktoreen lista ordenatua lortu"
29             ↪ );
30         System.out.println("3. aukera: Aktore berri bat gehitu");
31         System.out.println("4. aukera: Pelikula baten aktoreak lortu"
32             ↪ );
33         System.out.println("5. aukera: Pelikula bati dirua gehitu.");
34         System.out.println("6. aukera: Aktore bat ezabatu");
35         System.out.println("7. aukera: Fitxategi bat sortu aktoreen
36             ↪ zerrenda ordenatuarekin");
37         System.out.println("8. aukera: Bi elementuen arteko bidea");
38         System.out.println("9. aukera: Bi elementuren arteko
39             ↪ konexioaren proba enpirikoak");
40         int num = this.idatziInt();
41         switch (num)
42         {
43             case 1: //FUNCIONA. COMPROBADOS CASOS POSIBLES.
44                 //Comprobado que pasa si mete nombre erroneo.
45                 this.aktorearenPelikulak();
46                 amaitu();
47                 break; //para que despues de preguntar amaitu acabe
48             case 2: // FUNCIONA
49                 this.listaOrdenatua();
50                 amaitu();
51                 break; //para que despues de preguntar amaitu acabe
52             case 3: //FUNCIONA. COMPROBADOS CASOS POSIBLES

```

```
50         this.aktoreaGehitu();
51         amaitu();
52         break; //para que despues de preguntar amaitu acabe
53
54     case 4: //Comprobado que ocurre si el nombre es erroneo.
55         this.pelikularenAktoreak();
56         amaitu();
57         break;
58
59     case 5:
60         this.diruaGehitu();
61         amaitu();
62         break;
63
64     case 6:
65         this.aktoreaEzabatu();
66         amaitu();
67         break;
68
69     case 7:
70         this.aktoreenFitxategiaSortu();
71         amaitu();
72         break;
73
74     case 8:
75         this.konektatuta();
76         amaitu();
77         break;
78
79     case 9:
80         this.denbora();
81         amaitu();
82         break;
83
84
```

```
85         default:
86             System.err.println ( "Errorea gertatu da sisteman" );
87             this.amaitu();
88             break;
89     }
90
91 }
92 private void denbora() {
93     GraphHash grafo= new GraphHash();
94     grafo.grafoaSortu(listaAktore.getList());
95     System.out.println("Grafoa sortu da");
96     grafo.probaEnpirikoak();
97
98 }
99 public void amaitu() throws Exception { //Metodo para cerrar lo que
    ↳ hay que hacer.
100
101     System.out.println("Saioa amaitu nahi duzu?");
102     System.out.println("BAI sartu 9");
103     System.out.println("EZ sartu 0");
104     Scanner irten = new Scanner(System.in);
105     try {
106
107         switch (irten.nextInt())
108         {
109             case 9:
110                 System.out.println ( "Saio amaituko da." );
111                 break;
112
113             case 0:
114                 new Scannera();
115                 irten.close();
116                 break;
117
118             default:
```

```

119         System.err.println ( "Ez dago horrelako aukerarik" );
120         irten.close();
121         break;
122     }
123     irten.close();
124     }catch (InputMismatchException e) {
125         System.err.println("Ez duzu zenbakirik sartu, berriro
126             ↳ egin");
127         this.amaitu();
128     }
129 }
130 //----- ESCANER
131 ↳ -----
132
133     public static Boolean listakKargatu() throws Exception{
134
135         long startTime = System.nanoTime();
136         String unekoAktore=null;
137         String[] linea = null;
138         File file = new File("C:\\Users\\elker\\Desktop\\
139             ↳ FilmsActors20162017.txt");
140         Scanner sc = new Scanner(file);
141         TimeUnit.SECONDS.sleep(1);
142         while (sc.hasNextLine()) {
143             linea = sc.nextLine().replace(" &&& ", "<").replace("> ",
144                 ↳ ">").split("[<>]+"); //la primera linea
145             String pelikulaIzena=linea[0].replace("-", "");
146             Pelikula Pelikula1 =new Pelikula(pelikulaIzena,0);
147             for(int i=1;i<linea.length;i++) { //pongo i=1 porque el primer
148                 ↳ dato del array es el nombre de la peli y no un actor
149                 unekoAktore=linea[i].replace(", ", " ");
150                 String izena = unekoAktore;
151                 Aktore Aktore1 = new Aktore(izena);//
152                 ↳ eeeeeeeeeee
153                 if(listaAktore.getList().aktoreaDago(izena)){

```

```

148         Pelikula1.gehituAktore(listaAktore.
           ↳ getList().bilatuAktorea(izena))
           ↳ ;
149         listaAktore.getList().bilatuAktorea(
           ↳ izena).gehituPelikula(Pelikula1)
           ↳ ;
150     }else {
151         Pelikula1.gehituAktore(Aktore1);
152         Aktore1.gehituPelikula(Pelikula1);
153         listaAktore.getList().gehituAktorea(
           ↳ Aktore1);
154     }
155 }
156 listaPelikula.getList().gehituPelikula(Pelikula1);
157 }
158 System.out.println(listaAktore.getList().listaLuzeera() + "
           ↳ aktore ezberdin daude.");
159 System.out.println(listaPelikula.getList().listaLuzeera() + "
           ↳ pelikula daude.");
160 long endTime = System.nanoTime();
161
162 long timeElapsed = endTime - startTime;
163
164 System.out.println(timeElapsed / 1000000000 + " segundu behar
           ↳ izan dira. " );
165 sc.close();
166 return true;
167 }
168
169 //----- METODOS DE
           ↳ LAS ACCIONES-----
170 public String idatziString() {
171     Scanner s = new Scanner(System.in);
172     String gureString =null;
173     gureString=s.nextLine();

```

```

174         return gureString;
175     }
176     public int idatziInt() throws Exception{
177         Scanner s = new Scanner(System.in);
178         int gureInt = 1;
179         try {
180             gureInt=s.nextInt();
181         }catch( InputMismatchException e) {
182             System.err.println("Ez duzu zenbakirik sartu.");
183             return 10;
184         }
185         return gureInt;
186     }
187     public ArrayList<Aktore> pelikularenAktoreak()throws Exception{
188         System.out.println("Idatzi pelikularen izena ");
189         String ad = this.idatziString();
190         // System.out.println(listaPelikula.getList().itZuliPelikula(ad)
191         ↪ instanceof Pelikula);
192         try {
193             ArrayList<Aktore> aktoreak = listaPelikula.getList().
194             ↪ itZuliPelikula(ad).getListAktore();
195             System.out.println(aktoreak.size() + " aktore ditu pelikula
196             ↪ honek, izenen lista modu egokian lortu da.");
197             return aktoreak;
198         } catch (NullPointerException e) {
199             System.err.println("Pelikularen izena ez da existitzen.");
200             return null;
201         }
202     }
203     public ArrayList<Pelikula> aktorearenPelikulak() throws Exception{
204         try {
205             System.out.println("Idatzi aktorearen izena");
206             String izen= this.idatziString();
207             ArrayList<Pelikula> lista = listaAktore.

```

```

206         ↳ getList().bilatuAktorea(izen).
207         ↳ getPelikulak(); //Esto lo hago para que
208         ↳ pueda decir que se ha hecho bien.
209     System.out.println("Listak modu egokian itzuli
210         ↳ dira.");
211     return lista;
212 } catch (NullPointerException e) { //Badakit
213     ↳ ez dela gomendagarria pointer Exception-
214     ↳ a modu honetan tratatzea, baina noizbait
215     ↳ erabiltzeko....
216     System.err.println("Izena ez da existitzen.");
217     return null;
218 }
219 }
220
221 public ArrayList<String> listaOrdenatua(){
222     ArrayList<String> lista =listaAktore.getList().
223     ↳ aktoreOrdenatuak();
224     System.out.println("Modu efektiboan lortu da lista ordenatua."
225     ↳ );
226     return lista;
227 }
228 public void aktoreaGehitu() {
229     System.out.println("Idatzi aktorearen izena");
230     String izen= this.idatziString();
231     if( listaAktore.getList().bilatuAktorea(izen) == null) {
232         listaAktore.getList().gehituAktorea(new Aktore(
233             ↳ izen));
234     if(listaAktore.getList().aktoreaDago(izen)) { //
235         ↳ Esto es si lo ha a adido bien.
236         System.out.println("Aktorea modu egokian
237             ↳ gehitu da.");

```



```

229         }
230     }else {
231         System.err.println("Aktorea jadanik dago listan,
                ↳ orduan ez da gehitu.");
232     }
233
234 }
235 public void diruaGehitu() throws Exception{ //Caso de no mal metido el
                ↳ dinero, o la película no existir, TRATADO:
236     System.out.println("Idatzi pelikularen izena");
237     String iZ=this.idatziString();
238     if(listaPelikula.getList().badagoPelikula(new
                ↳ Pelikula(iZ, 0))) {
239         System.out.println("Idatzi zenbat diru gehitu nahi duzun"
                ↳ );
240         try {
241             int dI=Integer.valueOf(this.idatziString());
242             listaPelikula.getList().itZuliPelikula(iZ).
                ↳ diruaGehitu(dI);
243             System.out.println("Dirua modu egokian gehitu da,
                ↳ eguneko dirua " + listaPelikula.getList().
                ↳ itZuliPelikula(iZ).getDirua()+" eurokoa da." );
244         }catch(NumberFormatException e) {
245             System.err.println("Ez duzu dirua ondo sartu,
                ↳ berro egin");
246             this.diruaGehitu();
247         }
248     }else {
249         System.err.println("Ez dago pelikula, berriro
                ↳ egin");
250         this.diruaGehitu();
251     }
252 }
253 public void aktoreaEzabatu() {
254     System.out.println("Idatzi aktorearen izena:");

```

```
255     String izen = this.idatziString();
256     if(listaAktore.getLista().aktoreaDago(izen)) {
257         listaAktore.getLista().ezabatuAktorea(new Aktore(izen)
258             ↪ );
259         System.out.println("Aktorea modu egokian ezabatu da."
260             ↪ + '\n');
261     }else {
262         System.err.println("Aktorea ez dago, berriro idatzi
263             ↪ datuak");
264     }
265 }
266 public void aktoreenFitxategiaSortu() {
267     ArrayList<String> lista=listaAktore.getLista().
268     ↪ aktoreOrdenatuak();
269     File fitxategia;
270     try {
271         System.out.println();
272         System.out.println("Idatzi sortuko den fitxategiaren
273             ↪ izena:");
274         String fitxIzena=this.idatziString();
275         boolean sortuta=false;
276         System.out.println("Idatzi sortuko den fitxategiaren
277             ↪ helbidea:");
278         String fitxHelbidea=this.idatziString();
279         fitxategia=new File(fitxHelbidea+"\\ "+fitxIzena+".txt"
280             ↪ ); //Intenta crear el archivo
281         while (!sortuta) {
282             sortuta=true;
283             if (!fitxategia.createNewFile()) { //Comprueba
284                 ↪ si hay algun fitxategi con el mismo
285                 ↪ nombre en el mismo helbide
286                 sortuta=false;
287                 System.out.println("Jada existitzen da
```

```

                ↪ fitxategi bat izen horrekin
                ↪ helbide honetan");
281         System.out.println();
282         System.out.println("Idatzi sortuko den
                ↪ fitxategiaren izena:");
283         fitxIzena=this.idatziString();
284         System.out.println("Idatzi sortuko den
                ↪ fitxategiaren helbidea:");
285         fitxHelbidea=this.idatziString();
286         fitxategia=new File(fitxHelbidea+"\\\\"+
                ↪ fitxIzena+".txt");
287     }
288 }
289 if (!fitxategia.isFile()) { //Comprueba si se ha
    ↪ creado el fitxategi
290     throw new Exception();
291 }
292 BufferedWriter bw=new BufferedWriter(new FileWriter(
    ↪ fitxategia)); //El aldagai que escribe en el txt
293 Iterator<String> itr=lista.iterator();
294 String aktorea;
295 while (itr.hasNext()) { //Escribe los actores en el
    ↪ txt
296     aktorea=itr.next();
297     if (itr.hasNext()) {
298         bw.write(aktorea+", ");
299     }
300     else {
301         bw.write(aktorea+".");
302     }
303 }
304 bw.close();
305 System.out.println("Fitxategia sortu da");
306 }
307 catch (Exception e) {

```

```
308         System.err.println("Ezin izan da fitxategia sortu ");
309     }
310 }
311
312
313 public void konektatuta() {
314     boolean dago1= false;
315     String a1=null;
316     String a2=null;
317     GraphHash nodoak = new GraphHash();
318     nodoak.grafoaSortu(listaAktore.getList());
319     while(!dago1) {
320         System.out.println("Idatzi lehenengo elementuaren
321             ↳ izena:");
322         a1 = this.idatziString();
323         if(!listaAktore.getList().aktoreaDago(a1)) {
324             a1=a1+" ";
325         }
326         dago1=nodoak.grafo.containsKey(a1);
327         if(!dago1) {
328             System.out.println("Sartu duzun aktorea ez
329                 ↳ dago aktore listan");
330         }
331     }
332
333     boolean dago2=false;
334     while(!dago2) {
335         System.out.println("Idatzi bigarren elementuaren izena
336             ↳ :");
337         a2 = this.idatziString();
338         if(!listaAktore.getList().aktoreaDago(a2)) {
339             a2=a2+" ";
340         }
341         dago2=nodoak.grafo.containsKey(a2);
342         if(!dago2) {
```

```
340         System.out.println("Sartu duzun aktorea ez  
           ↳ dago aktore listan");  
341     }  
342 }  
343 long startTime = System.nanoTime();  
344 boolean emaitza;  
345 emaitza= nodoak.konektatuta(a1, a2);  
346 long endTime = System.nanoTime();  
347 long timeElapsed = endTime - startTime;  
348 if(!emaitza) {  
349     System.out.println("Ez dago biderik " +a1 + " eta " +  
           ↳ a2+"-ren artean");  
350 }else {  
351     System.out.println("Bide bat dago " + a1+ " eta "+ a2+  
           ↳ "-ren artean");  
352 }  
353 System.out.println(timeElapsed / 1000000000 + " segundu behar  
           ↳ izan dira. " );  
354 }  
355 }
```

3.2 AKTORE

```
1
2 package packlaborategi3;
3
4 import java.util.ArrayList;
5 import java.util.Iterator;
6
7 public class Aktore {
8     private String izena;
9     private ArrayList<Pelikula>listaPelikulak = new ArrayList<Pelikula>();
10
11
12     //Eraikitzailea
13     public Aktore(String pIzena) {
14         this.izena = pIzena;
15         this.listaPelikulak=new ArrayList<Pelikula>();
16     }
17
18     private Iterator<Pelikula> getIteradorea(){
19         return this.listaPelikulak.iterator();
20     }
21     //Getter metodoak.
22     public ArrayList<Pelikula> getPelikulak() {
23         return this.listaPelikulak;
24     }
25     public String getIzena() {
26         return this.izena;
27     }
28     //Metodoak
29     public void gehituPelikula (Pelikula pPelikula) {
30         listaPelikulak.add(pPelikula);
31     }
32     public void kenduPelikula(Pelikula pPelikula) {
```

```

33         listaPelikulak.remove(pPelikula);
34     }
35     public int zenbatPelikula() {
36         return listaPelikulak.size();
37     }
38     public boolean badagoPelikula(String pPeli) {
39         Iterator<Pelikula> itr = this.getIteradorea();
40         Pelikula p;
41         while (itr.hasNext() ){
42             p = itr.next();
43             if (p.getIzena().equals(pPeli)) {
44                 return true;
45             }
46         }
47         return false;
48     }
49     public ArrayList<Pelikula> itzuliPelikulak() {
50         return this.listaPelikulak;
51     }
52
53     public void ikusiPelikulak() {
54         Pelikula p =null;
55         Iterator<Pelikula> itr = this.getIteradorea();
56         if(itr.hasNext()) {
57             while(itr.hasNext()) {
58                 p=itr.next();
59                 System.out.println(p.getIzena());
60             }
61         }
62     }
63
64     @Override
65     public boolean equals(Object o) {
66         Aktore person = (Aktore) o;
67         if (o instanceof Aktore && this.getIzena()== person.getIzena())

```

```
68         ↪ ) {  
69             return true;  
70         }else return false;  
71     }  
72  
73  
74 }
```


3.3 PELIKULA

```
1      package packlaborategi3;
2
3      import java.util.ArrayList;
4
5      public class Pelikula {
6          private String izena;
7          private ArrayList<Aktore> listaAktoreak = new ArrayList<Aktore>
8              >();
9          private int dirua;
10
11      public Pelikula(String pIzena, int pDirua) {
12          this.izena = pIzena;
13          this.listaAktoreak =new ArrayList<Aktore>();
14          this.dirua = pDirua;
15      }
16      public String getIzena() {
17          return this.izena;
18      }
19      public ArrayList<Aktore> getListaAktore(){
20          return this.listaAktoreak;
21      }
22      public int getDirua() {
23          return this.dirua;
24      }
25      public void diruaGehitu(int pDirua) {
26          this.dirua = this.dirua + pDirua;
27      }
28      public void gehituAktore (Aktore pAktore) {
29          if(this.listaAktoreak ==null) {
30              this.listaAktoreak=new ArrayList<Aktore>();
31          }
32          this.listaAktoreak.add(pAktore);
```

```

32         }
33         public void kenduAktore(Aktore pAktore) {
34             listaAktoreak.remove(pAktore);
35         }
36         public int getAktoreKopurua() {
37             return this.listaAktoreak.size();
38         }
39     }
40
41 \section{listaPelikula}
42
43 package packlaborategi3;
44
45 import java.util.ArrayList;
46 import java.util.Iterator;
47
48 public class listaPelikula {
49     private static listaPelikula i = null;
50     private ArrayList<Pelikula> lista;
51
52     //eraikitzailea
53     private listaPelikula() {
54         this.lista = new ArrayList<Pelikula>();
55     }
56
57
58     public static synchronized listaPelikula getList() {
59         if ( listaPelikula.i == null) {
60             listaPelikula.i = new listaPelikula();
61         }
62         return i;
63     }
64     public void gehituPelikula(Pelikula pPelikula) {
65         lista.add(pPelikula);
66     }

```

```

67     private Iterator<Pelikula> getIteradorea() {
68         return this.lista.iterator();
69     }
70     public void kenduPelikula(Pelikula pPelikula) {
71         lista.remove(pPelikula);
72     }
73     public int listaLuzeera() {
74         return lista.size();
75     }
76     public void ezabatu() {
77         lista.clear();
78     }
79     public boolean badagoPelikula (Pelikula pPelikula) {
80         Iterator<Pelikula> itr = this.getIteradorea();
81         Pelikula p1=null;
82         while (itr.hasNext() ) {
83             //         System.out.println(a++);
84             p1 = itr.next();
85             if (p1.getIzena().trim().equals(
86                 ↪ pPelikula.getIzena().trim())) {
87                 return true;
88             }
89             return false;
90         }
91     public Pelikula itZuliPelikula(String pIzena) { //return !
92         ↪ instanceof Pelikula por algun motivo, null?
93         Iterator<Pelikula> itr = this.getIteradorea();
94         Pelikula p1=null;
95         boolean aurkitua = false;
96         while (itr.hasNext() && !aurkitua) {
97             //         System.out.println(a++);
98             p1 = itr.next();
99             if (p1.getIzena().trim().equals(pIzena.trim())
100                 ↪ ) {

```

```
99             aurkitua=true;
100         }
101     }
102     if(!aurkitua) {
103         return null;
104     }else {
105         return p1;
106     }
107 }
108 }
```

3.4 LISTAPELIKULA

```
1      package packlaborategi3;
2
3      import java.util.ArrayList;
4      import java.util.Iterator;
5
6      public class listaPelikula {
7          private static listaPelikula i = null;
8          private ArrayList<Pelikula> lista;
9
10         //eraikitzailea
11         private listaPelikula() {
12             this.lista = new ArrayList<Pelikula>();
13         }
14
15
16         public static synchronized listaPelikula getList() {
17             if ( listaPelikula.i == null) {
18                 listaPelikula.i = new listaPelikula();
19             }
20             return i;
21         }
22         public void gehituPelikula(Pelikula pPelikula) {
23             lista.add(pPelikula);
24         }
25         private Iterator<Pelikula> getIteradorea() {
26             return this.lista.iterator();
27         }
28         public void kenduPelikula(Pelikula pPelikula) {
29             lista.remove(pPelikula);
30         }
31         public int listaLuzeera() {
32             return lista.size();
```

```

33         }
34     public void ezabatu() {
35         lista.clear();
36     }
37     public boolean badagoPelikula (Pelikula pPelikula) {
38         Iterator<Pelikula> itr = this.getIteradorea();
39         Pelikula p1=null;
40         while (itr.hasNext() ) {
41             //         System.out.println(a++);
42             p1 = itr.next();
43             if (p1.getIzena().trim().equals(
44                 ↪ pPelikula.getIzena().trim())) {
45                 return true;
46             }
47             return false;
48         }
49     public Pelikula itZuliPelikula(String pIzena) { //return !
50         ↪ instanceof Pelikula por algun motivo, null?
51         Iterator<Pelikula> itr = this.getIteradorea();
52         Pelikula p1=null;
53         boolean aurkitua = false;
54         while (itr.hasNext() && !aurkitua) {
55             //         System.out.println(a++);
56             p1 = itr.next();
57             if (p1.getIzena().trim().equals(pIzena.trim())
58                 ↪ ) {
59                 aurkitua=true;
60             }
61         }
62         if(!aurkitua) {
63             return null;
64         }else {
65             return p1;
66         }
67     }

```

```
65 }  
66 }
```

3.5 LISTAAKTORE

```
1 package packlaborategi3;  
2 import java.util.ArrayList;  
3 import java.util.Collections;  
4 import java.util.HashMap;  
5 import java.util.Iterator;  
6 import java.util.List;  
7  
8  
9 public class listaAktore {  
10     private static listaAktore i;  
11     private HashMap<String,Aktore> lista = null;  
12  
13     public static synchronized listaAktore getList() {  
14         if (listaAktore.i == null) {  
15             listaAktore.i = new listaAktore();  
16         }  
17         return i;  
18     }  
19     // Eraikitzaile pribatua  
20     private listaAktore() {  
21         lista = new HashMap<String,Aktore>();  
22     }  
23  
24     public void gehituAktorea(Aktore pAktore) {  
25         this.lista.put(pAktore.getIzena(), pAktore);  
26     }  
27  
28     public Iterator<String> getIteradorea() {
```

```

29         Iterator<String> itr = lista.keySet().iterator();
30         return itr;
31     }
32
33     public int listaLuzeera() {
34         return this.lista.size();
35     }
36     public void clear() {
37         this.lista.clear();
38     }
39     public Aktore bilatuAktorea(String pIzena) {
40         return this.lista.get(pIzena);
41     }
42     public boolean aktoreaDago(String pIzena) {
43         return lista.containsKey(pIzena);
44     }
45
46     public void ezabatuAktorea(Aktore pAktore) {
47         this.lista.remove(pAktore.getIzena());
48     }
49
50
51     public ArrayList<String> aktoreOrdenatuak() {
52         ArrayList<String> listaordenatua = new ArrayList<>(lista.
53             ↪ keySet());
54         return mergeSort(listaordenatua);
55     }
56     public ArrayList<String> mergeSort(ArrayList<String> osoa) {
57         ArrayList<String> ezk = new ArrayList<String>();
58         ArrayList<String> esku = new ArrayList<String>();
59         int erdia;
60
61         if (osoa.size() == 1) { //Hau da, listak elementu bakarra badu.
62             return osoa;
63         } else { //Bestela listan bitan zatituko dugu.

```



```

63         erdia = osoa.size()/2;
64         for (int i=0; i<erdia; i++) {
65             ezk.add(osoa.get(i));
66         }
67
68         for (int i=erdia; i<osoa.size(); i++) {
69             esku.add(osoa.get(i));
70         }
71
72         //Alde bakoitzari merge-sort algoritmoa aplikatu modu
73         ↪ errekurtsiboan.
74         ezk = mergeSort(ezk);
75         esku = mergeSort(esku);
76
77         // Batu emaitzak.
78         merge(ezk, esku, osoa);
79     }
80     return osoa;
81 }
82
83 private void merge(ArrayList<String> ezk, ArrayList<String> esku,
84     ↪ ArrayList<String> osoa) {
85     int ezkIndex = 0;
86     int eskuIndex = 0;
87     int osoaIndex = 0;
88
89     //ezk edo esku luzeera duten bitartean, beti zatika joango
90     ↪ gara hartzen eta biak bat egiten.
91
92     while (ezkIndex < ezk.size() && eskuIndex < esku.size()) {
93         if ( (ezk.get(ezkIndex).compareTo(esku.get(eskuIndex))) < 0) {
94             osoa.set(osoaIndex, ezk.get(ezkIndex));
95             ezkIndex++;
96         } else {
97             osoa.set(osoaIndex, esku.get(eskuIndex));
98             eskuIndex++;
99         }
100     }

```

```
95         osoaIndex++;
96     }
97     ArrayList<String> gelditzenDena;
98     int gelditzenDenaIndex;
99     if (ezkIndex >= ezk.size()) {
100         // Ezkerrko aldia guztiz erabili da.
101         gelditzenDena = esku;
102         gelditzenDenaIndex = eskuIndex;
103     } else {
104         //Eskuineko aldea guztiz erabili da.
105         gelditzenDena = ezk;
106         gelditzenDenaIndex = ezkIndex;
107     }
108     // Kopiatu oraindik amaitu ez den aldea osorik.
109     for (int i=gelditzenDenaIndex; i<gelditzenDena.size(); i++) {
110         osoa.set(osoaIndex, gelditzenDena.get(i));
111         osoaIndex++;
112     }
113 }
114
115
116 }
```

3.6 GRAPHHASH

```

1  package packlaborategi3;
2  import java.util.ArrayList;
3  import java.util.HashMap;
4  import java.util.HashSet;
5  import java.util.Iterator;
6  import java.util.LinkedList;
7  import java.util.Queue;
8  import java.util.Random;
9
10 public class GraphHash {
11     HashMap<String, ArrayList<String>> grafo;
12
13
14
15     public GraphHash() {
16         this.grafo= new HashMap<String, ArrayList<String>>();
17     }
18     public void grafoaSortu(listaAktore lAktoreak) {
19         Iterator<String> itr = lAktoreak.getIteradorea();
20         ↪
21         while(itr.hasNext()) {
22             String aktorea = itr.next();
23             Aktore unekoa = lAktoreak.bilatuAktorea(
24                 ↪ aktorea);
25             ArrayList<String> listaP = new ArrayList<
26                 ↪ String>();
27             // listaP.clear(); por si las moscas
28             Iterator<Pelikula> iteratorPeli = unekoa.
29                 ↪ getPelikulak().iterator();
30             while(iteratorPeli.hasNext()) {
31                 String unekoP = iteratorPeli.next().
32                     ↪ getIzena();

```

```

28         listaP.add(unekoP);
29         ArrayList<String> pelikularenAkt;
30         if(this.grafo.containsKey(unekoP)) {
31             pelikularenAkt=this.grafo.get(
32                 ↪ unekoP);
33         }else {
34             pelikularenAkt=new ArrayList<
35                 ↪ String>();
36         }
37         pelikularenAkt.add(aktorea);
38         grafo.put(unekoP, pelikularenAkt);
39     }
40     //una vez hemos gestionado todas su pelikulas,
41     ↪ a adimos el actor + listaP (sus pelis)
42     grafo.put(aktorea, listaP);
43 }
44
45
46
47 public int luzeera() {
48     return this.grafo.size();
49 }
50
51
52
53 public void clear() {
54     this.grafo.clear();
55 }
56 public boolean konektatuta(String a1, String a2) { //Metodo principal,
57     ↪ saber si dos elementos estan conectados
58     HashSet<String> aztertuak = new HashSet<String>();
59     Queue<String> aztGabeak = new LinkedList<>();

```

```

59     aztGabeak.add(a1);
60     if(a1.equals(a2)) {
61         return true; //Caso tonto, pero bueno, por si acaso
62     }
63     if(grafo.isEmpty() || !grafo.containsKey(a1)) { //YO AQUI
64         ↪ PONDRIA || !grafo.containsKey(a1)
65         return false;
66     }
67     while(!aztGabeak.isEmpty()) { //Falta la otra condicion de
68         ↪ salida.
69         if(!aztertuak.contains((String)aztGabeak.peek())) {
70             if(aztGabeak.peek().equals(a2)) {
71                 return true;
72             }else {
73                 String uneko = aztGabeak.peek();
74                 aztertuak.add(uneko);
75                 aztGabeak.poll(); //Me cargo el primer
76                 ↪ elemento.
77                 ArrayList<String> lista = grafo.get(
78                 ↪ uneko);
79                 Iterator<String> itr = lista.iterator
80                 ↪ ();
81                 while(itr.hasNext()) {
82                     String tmp = itr.next();
83                     aztGabeak.add(tmp);
84                 }
85             }else { //Es decir, que el elemento ya ha sido
86                 ↪ investigado
87                 aztGabeak.poll(); //Me cargo el elemento ya
88                 ↪ investigado.
89             }
90         }
91     }
92     return false;
93 }

```

```
87
88
89
90
91     public void print() {
92         /*Iterator<String> itr = grafo.keySet().iterator();
93         while(itr.hasNext()) {
94             String uneko = itr.next();
95             System.out.println("El nodo " + uneko + " esta
96                 ↳ relacionado con:");
97             ArrayList<String> lista = grafo.get(uneko);
98             for (int i=0;i<lista.size();i++) {
99                 System.out.println(lista.get(i));
100             }
101         }*/
102         int i = 1;
103         for (String s: grafo.keySet()){
104             System.out.print("Element: " + i++ + " " + s + " --> ");
105             for (String k: grafo.get(s)){
106                 System.out.print(k + " ### ");
107             }
108             System.out.println();
109         }
110
111     public void probaEnpirikoak() {
112         Object[] values = grafo.keySet().toArray();
113         long denbora = 0;
114         int n = 100;
115         int t = 0;
116         int f = 0;
117         // System.out.println("Estoy fuera del for");
118         for(int i = 1;i<=n;i++) {
119
120         //             System.out.println("Estoy en la iteracion "+ i);
```

```

121         double r1=(Math.random() * ((values.length-1)));
122         double r2=(Math.random() * ((values.length-1)));
123         int z1= (int) Math.round(r1);
124         int z2= (int) Math.round(r2);
125         //      System.out.println(z1 + " eta " + z2 + " balioak sortu
           ↳ ditut");
126         String randomValue1 = values[z1].toString();
127         String randomValue2 = values[z2].toString();
128         //      System.out.println(randomValue1);
129         //      System.out.println(randomValue2);
130         long startTime = System.currentTimeMillis();
131         //      System.out.println(this.konektatuta(randomValue1,
           ↳ randomValue2));
132         if(this.konektatuta(randomValue1, randomValue2)) {
133             t++;
134         }else {
135             f++;
136         }
137         long endTime = System.currentTimeMillis();
138         denbora = denbora + (endTime-startTime);
139         if(i% 10 == 0) {System.out.println(i+" proba egin
           ↳ ditut");
140
141         }
142
143
144     }
145     System.out.println(t+ "True kasu egon dira");
146     System.out.println(f+ "False kasu egon dira");
147     System.out.println((denbora/n)+" Milisegundo behar ditu batatzeko
           ↳ konexioa egiaztatzeke");
148 }
149
150
151 }

```

152 \newpage

3.7 LISTAAKTORETEST

```
1
2 package Laborategi1;
3
4 import static org.junit.jupiter.api.Assertions.*;
5
6 import org.junit.jupiter.api.AfterEach;
7 import org.junit.jupiter.api.BeforeEach;
8 import org.junit.jupiter.api.Test;
9
10 import packlaborategi3.Aktore;
11 import packlaborategi3.listaAktore;
12
13 class listaAktoreTest {
14     Aktore a1,a2,a3,a4;
15     listaAktore listaa,listaa2;
16     @BeforeEach
17     void setUp() throws Exception {
18         listaa = listaaAktore.getListaa();
19         listaa2 = listaaAktore.getListaa();
20         a1 = new Aktore("Sanjuan", "Kerman");
21         a2 = new Aktore("Ruiz", "Alba");
22         a3 = new Aktore("Alonso", "Luna");
23         a4 = new Aktore("Colate", "Pacho");
24     }
25
26     @AfterEach
27     void tearDown() throws Exception {
28         listaaAktore.getListaa().clear();
29         listaaAktore.getListaa().clear();
30     }
31 }
```



```
30         a1 = null;
31         a2 = null;
32         a3 = null;
33     }
34     @Test
35     void testGetLista() {
36         assertNotNull(lista);
37     }
38
39     @Test
40     void testGehituAktorea() {
41         lista.gehituAktorea(a1);
42         assertEquals(1, lista.listaLuzeera());
43         lista.gehituAktorea(a1);
44         assertEquals(1, lista.listaLuzeera()); //ezin dira bi aktore
45         ↪ berdin egon
46         lista.gehituAktorea(a2);
47         lista.gehituAktorea(a3);
48         assertEquals(3, lista.listaLuzeera());
49         lista.gehituAktorea(a4);
50         assertEquals(4, lista.listaLuzeera());
51         lista.gehituAktorea(a1);
52         lista.gehituAktorea(a2);
53         lista.gehituAktorea(a3);
54         lista.gehituAktorea(a4);
55         assertEquals(4, lista.listaLuzeera());
56     }
57
58     @Test
59     void testEzabatuAktorea() {
60         lista.ezabatuAktorea(a1);
61         assertEquals(0, lista.listaLuzeera());
62         lista.gehituAktorea(a1);
63         lista.ezabatuAktorea(a1);
```

```
64         assertEquals(0, lista.listaLuzeera());
65         lista.gehituAktorea(a1);
66         lista.gehituAktorea(a2);
67         lista.ezabatuAktorea(a2);
68         assertEquals(1, lista.listaLuzeera());
69         lista.gehituAktorea(a2);
70         lista.gehituAktorea(a3);
71         lista.ezabatuAktorea(a4);
72         assertEquals(3, lista.listaLuzeera());
73         lista.ezabatuAktorea(a3);
74         assertEquals(2, lista.listaLuzeera());
75     }
76
77     @Test
78     void testListaLuzeera() {
79         lista.gehituAktorea(a1);
80         assertEquals(1, lista.listaLuzeera());
81         lista.gehituAktorea(a2);
82         lista.gehituAktorea(a3);
83         assertEquals(3, lista.listaLuzeera());
84     }
85
86     @Test
87     void testAktoreaDago() {
88         assertFalse(lista.aktoreaDago(a1.getAbizena(), a1.getIzena()));
89         lista.gehituAktorea(a1);
90         assertTrue(lista.aktoreaDago(a1.getAbizena(), a1.getIzena()));
91         lista.gehituAktorea(a2);
92         lista.gehituAktorea(a3);
93         assertTrue(lista.aktoreaDago(a2.getAbizena(), a2.getIzena()));
94         assertTrue(lista.aktoreaDago(a3.getAbizena(), a3.getIzena()));
95         assertFalse(lista.aktoreaDago(a4.getAbizena(), a4.getIzena()));
96     }
97
98     @Test
```

```
99     void testAktoreOrdenatuak() {
100         //Bi lista sortuko ditugu orden diferentean, gero ordenatu eta
           ↳ berdinak izango beharko litzateke.
101         a1 = new Aktore("Kerman", "A");
102         a2 = new Aktore("Ander", "B");
103         a3 = new Aktore("Josu", "C");
104         lista.gehituAktorea(a1);
105         lista.gehituAktorea(a2);
106         lista.gehituAktorea(a3);
107         lista.aktoreOrdenatuak();
108         lista2.gehituAktorea(a3);
109         lista2.gehituAktorea(a1);
110         lista2.gehituAktorea(a2);
111         lista2.aktoreOrdenatuak();
112         assertEquals(lista, lista2);
113     }
114
115 }
```

3.8 PELIKULATEST

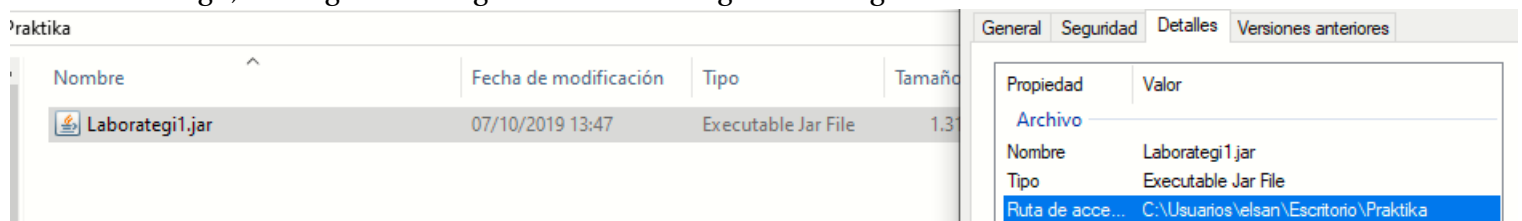
```
1 package Laborategi1;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.AfterEach;
6 import org.junit.jupiter.api.BeforeEach;
7 import org.junit.jupiter.api.Test;
8
9 import packlaborategi3.Aktore;
10 import packlaborategi3.Pelikula;
11
12 class PelikulaTest {
13     Aktore a1,a2,a3;
14     Pelikula p1,p2,p3;
15     @BeforeEach
16     void setUp() throws Exception {
17         a1 = new Aktore("Sanjuan", "Kerman");
18         a2 = new Aktore("Ruiz", "Alba");
19         a3 = new Aktore("Alonso", "Luna");
20         p1 = new Pelikula("as",0);
21         p2 = new Pelikula("qw",0);
22         p3 = new Pelikula("zx",0);
23     }
24
25     @AfterEach
26     void tearDown() throws Exception {
27         a1 = null;
28         a2 = null;
29         a3 = null;
30         p1 = null;
31         p2 = null;
32         p3 = null;
```

```
33     }
34
35     @Test
36     void testPelikula() {
37         assertNotNull(p1);
38     }
39
40     @Test
41     void testGetIzena() {
42         assertEquals("as", p1.getIzena());
43     }
44
45     @Test
46     void testGetListaAktore() {
47         p1.gehituAktore(a1);
48         assertNotNull(p1.getListaAktore());
49     }
50
51     @Test
52     void testDiruaGehitu() {
53         p1.diruaGehitu(15);
54         assertEquals(15, p1.getDirua());
55     }
56
57     @Test
58     void testGehituAktore() {
59         p1.gehituAktore(a1);
60         assertEquals(1, p1.getAktoreKopurua());
61         p1.gehituAktore(a1); //ez dauka zentzu handirik, baina egon
62         //↳ daiteke bi aktore izen abizen berdinarekin
63         assertEquals(2, p1.getAktoreKopurua()); //ez dugunez
64         //↳ implementatu aktore (pelikula bakoitzaren aktoreak)
65         //↳ errepikatuta ez egotea, aktore berdina bi aldiz gehi
66         //↳ dezakegu
67         p1.gehituAktore(a2);
```

```
64         assertEquals(3, p1.getAktoreKopurua());
65         p1.gehituAktore(a3);
66         assertEquals(4, p1.getAktoreKopurua()); //De este modo
           ↳ comprobamos tambien si "zenbat pelikula" funciona.
67     }
68
69     @Test
70     void testKenduAktore() {
71         p1.gehituAktore(a1);
72         p1.kenduAktore(a2);
73         assertEquals(1, p1.getAktoreKopurua()); //aktore bat dago baina
           ↳ ez da kentzen saiatzen duguna
74         p1.gehituAktore(a3);
75         p1.gehituAktore(a2);
76         Aktore a4= new Aktore("Colate","Pacho");
77         p1.kenduAktore(a4);
78         assertEquals(3, p1.getAktoreKopurua()); //hiru aktore daude eta
           ↳ beste laugarren bat kentzen saiatzen gara
79     }
80
81     @Test
82     void testGetAktoreKopurua() {
83         p1.gehituAktore(a1);
84         p1.gehituAktore(a2);
85         assertEquals(p1.getAktoreKopurua(),2);
86         p1.kenduAktore(a1);
87         assertEquals(1, p1.getAktoreKopurua());
88     }
89
90 }
```

Progama irekitzen

Lehenengo eta behin CMD kontsolan gure .jar fitxategia dagoen helbidea idatzi behar dugu, beraz gure fitxategiaren helbidea begiratu dugu:



Eta ondoren CMD-n helbide hori aukeratu dugu:

```
C:\> Símbolo del sistema

Microsoft Windows [Versión 10.0.18362.356]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\elsan>cd \users\elsan\desktop\Praktika

C:\Users\elsan\Desktop\Praktika>
```

Ondoren fitxategia irekitzeko esango diogu, gure kasuan -Xmx1g komandoa erabili dugu programari 1GB-eko RAM memoria emateko (ez da beharrezkoa baina aktore eta pelikula asko gehitzen badira memoria geroz eta gehiago behar izango da):

```
C:\Users\elsan\Desktop\Praktika>java -Xmx1g -jar Laborategi1.jar
Pelikula eta aktoreak kargatzen daude, prozesu honek ez luke denbora luzerik hartu behar
```

Hori eginda programa erabiltzeko prest egongo gara.

Gure programak behar beste memoria ez badu honako errorea lortuko dugu, eta lehen esan dugun komandori esker, memoria gehiago emango beharko diogu.

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
  at java.util.Arrays.copyOfRange(Unknown Source)
  at java.lang.String.<init>(Unknown Source)
  at java.nio.HeapCharBuffer.toString(Unknown Source)
  at java.nio.CharBuffer.toString(Unknown Source)
  at java.util.regex.Matcher.toMatchResult(Unknown Source)
  at java.util.Scanner.match(Unknown Source)
  at java.util.Scanner.hasNextLine(Unknown Source)
  at Laborategi1.Scannera.listakKargatu(Scannera.java:121)
  at Laborategi1.Scannera.main(Scannera.java:20)
```


Amaiera

5.1 KONKLUSIOAK

Laborategi hau egin ostean, hainbat konklusiotara iritzi gara. Alde batetik, lehenengo laborategia izanda, hasieran pixka bat kostatu zitzaigun, baina azkenean erritmoa lortu dugu. Beste alde batetik, datu-egitura berriak erabili ditugu, "bizitza errealeko" arazo bat konpontzeko, eta honek, begiak ireki dizkigu hurrengo laborategietan ideia berriak inplementatzeko eta beti arazoei beste buelta bat emateko. Azkenik, konklusio moduan, laborategi honen zailtasuna listen kudeaketan (elementuak gehitu eta kendu) zatzen, hau lortuta, bestelako arazoen soluzioa eta inplementazioa azkoz errazagoa izan da.

5.2 BIBLIOGRAFIA

- StackOverflow
- W3Schools
- CodeAcademy
- GeekForGeeks
- Oracle
- edu4java youtube kanala - Youtubeko bideoa.