

BILBOKO INGENIARITZA ESKOLA

KUDEAKETAREN ETA INFORMAZIO SISTEMEN
INFORMATIKAREN INGENIARITZAKO GRADUA

Aktoreak eta pelikulak kudeatu (1. eginkizuna)

Egileak:

Adei Arias

Jon Barbero

Ander Prieto

Arloa:

Datu-Egiturak eta

Algoritmoak

2. maila

46. taldea

1. lauhilabetea



Aurkibidea

Sarrera eta arazoaren aurkezpena	1
Diseinua	2
Datu egituren diseinua	4
Metodo nagusien diseinu eta inplementazioa	5
Datuak kargatu fitxategi batetik	5
Aktore baten bilaketa	6
Aktore berri baten txertaketa	6
Aktore baten pelikulak bueltatu	7
Pelikula bateko aktoreak bueltatu	7
Pelikula baten dirua gehitu	8
Aktore baten ezabaketa	8
Aktoreen zerrenda fitxategi batean gorde	9
Aktoreen zerrenda ordenatua lortu	10
Kodea	11
Aktore.java	11
ArrayPelikulak.java	11
ArrayAktoreak.java	13
ListaAktoreak.java	14
ListaPelikula.java	18
Pelikula.java	20
JUnitak	22
AktoreTest.java	22
ArrayPelikulakTest.java	23
ArrayAktoreakTest.java	26
ListaAktoreakTest.java	28
ListaPelikulaTest.java	30
PelikulaTest.java	33
Ondorioak	36
Erreferentziak	37

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 lauhilabetean zehar.

Lehen eginkizun^[1] honetan, aktore-kopuru handia eta beraien pelikulak kudeatuko dituen aplikazioa sortu beharko dugu. Honetarako, ezagunak ditugun *ArrayList*-ak eta berria den *HashMap* egitura erabiliko ditugu.

Aplikazio honek hainbat ekintza bideratuko ditu: Adibidez, pelikula baten aktore guztiak lortu (ala alderantziz, aktore baten pelikula guztiak), aktoreak gehitu ala kendu edota pelikula baten aurrekontua gehitu

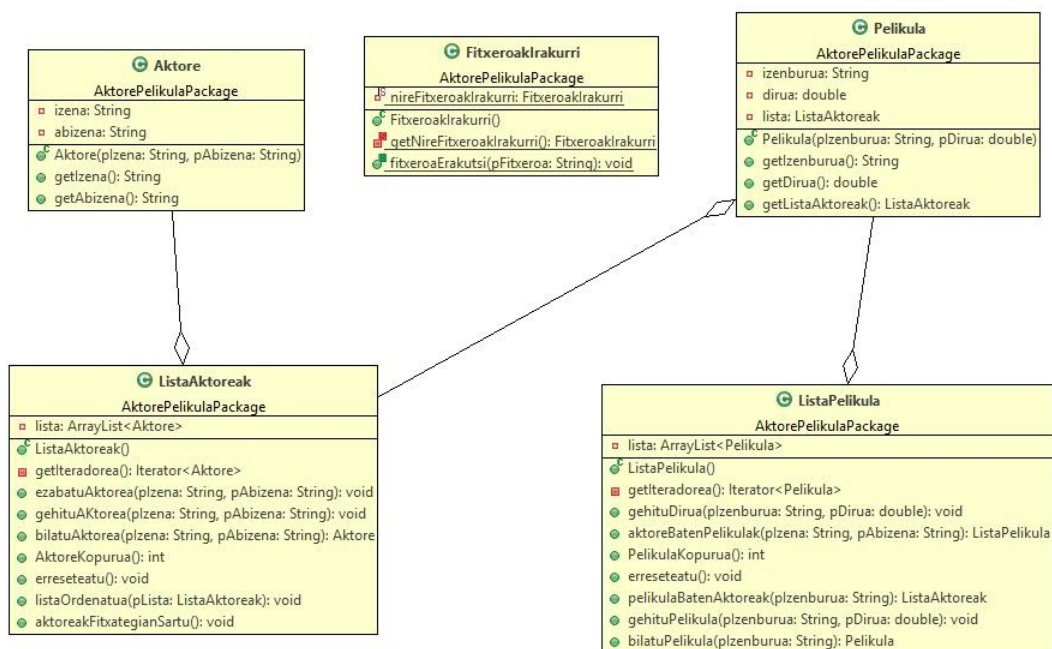
Sei klase izango ditugu: Alde batetik, *Aktore* eta *Pelikula* klaseek elementu bakarra gordeko dute. Beste aldetik, *ListaAktore* eta *ListaPelikulak* klaseak bi *HashMap* izango dira, non datu guztiak bilduko diren. Azkenik, *ArrayAktore* eta *ArrayPelikulak* klaseak erabilgarriak izango dira inplementazioan - aurreko paragrafoan azaldutako hasierako bi ekintzetan, adibidez.

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

Diseinua

Hasieratik, diseinua nahiko ondo bideratuta zegoen.

Lehen klase diagraman (1. irudia), lau klase sartu genituen, bi EMArekin: *Aktore*, *Pelikula* eta haien listak (hauek izango ziren EMak). Gainera, aparteko beste klase bat ere kokatu genuen, *FitxeroakIrakurri* izenekoa; hau aktore eta pelikulak dituen fitxeroa kargatzeko balio du. Hau iazko PMOO proiektutik hartu genuen, honekin lan egin genuelako.

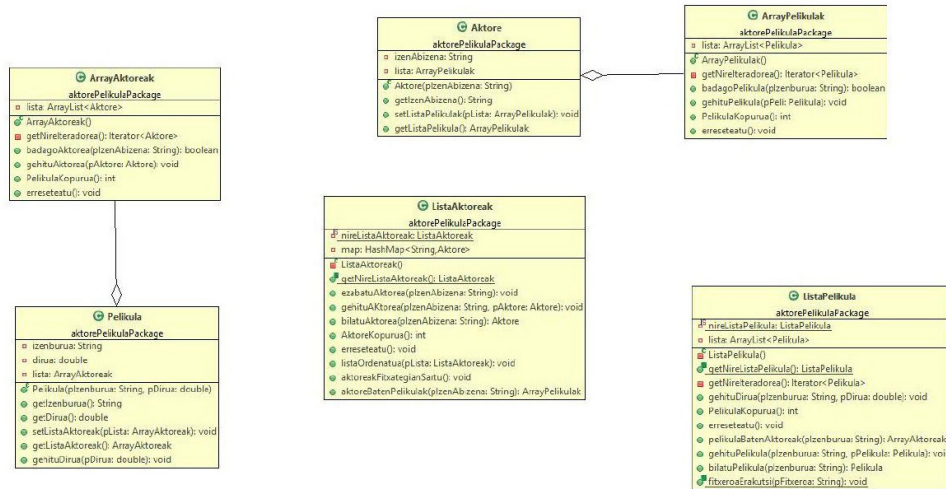


1. irudia: Hasierako diseinua

Diseinua, nahiz eta ideari dagokionez ona izan, efizientzia arazoak zituen; aktore guztiak sartzen denbora oso (oso) luzea ematen zuen.

Horregatik, Aktoreen *ArrayList*-a *HashMap*^[2] bihurtzea aukeratu genuen (2. irudia). Izan ere, askoz ere optimoa da; *ArrayList* batekin denbora gehiago eman behar da elementu bat bilatzen, adibidez.

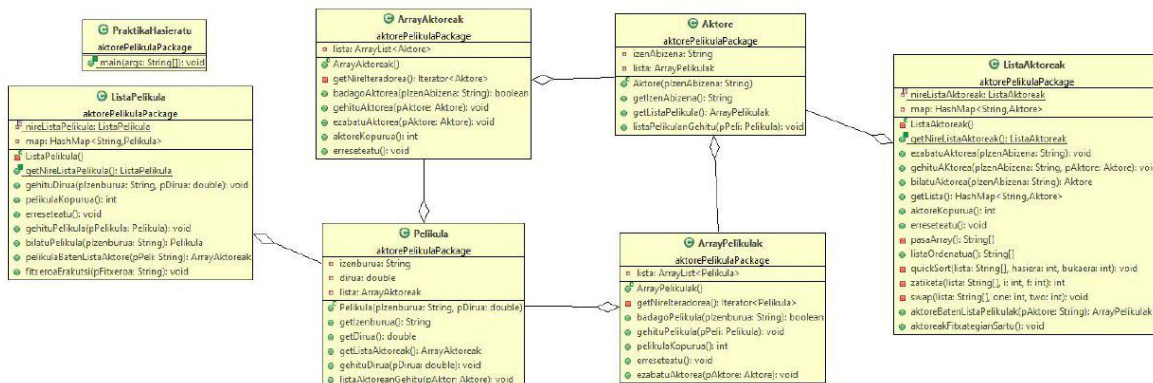
Gainera, *FitxeroakIrakurri* klasea kendu eta bere eginkizuna *fitxeroaErakutsi* metodoan sartu genuen.



2. irudia: Bigarren diseinua

Momentu honetan, proiektua ez zegoen guztiz bukatuta. Hainbat metodo falta ziren, haien artean aktoreak ordenatzearena. *QuickSort*^[3] erabiltzea erabaki genuen, azkarrena zelako; hau aurrerago azalduko da.

Lehen aipatu dugun moduan, sei klase izango ditugu: Hasteko, *Aktore* eta *Pelikula* klaseek elementu bakarra gordeko dute. Gero, *ListaAktore* eta *ListaPelikula* klaseak bi *HashMap* izango dira, non datu guztiak bilduko diren. Bukatzeko, *ArrayAktore* eta *ArrayPelikula* klaseak erabilgarriak izango dira implementazioan.



3. irudia: Amaierako diseinua

Datu egituren diseinua

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 egitura 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 pelikularentzat 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).

Metodo nagusien diseinu eta implementazioa

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

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: $O(1)$ ***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: $O(1)$

Aktore baten pelikulak bueltatu

```
public ArrayPelikulak aktoreBatenListaPelikulak(String pAktore)
```

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

```
// Post: Izen hori jakinda, HashMapean Aktore objektua bilatuko du eta honen lista  
buelatatuko 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: $O(1)$

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: $O(1)$

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: $O(1)$

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: $O(1)$

Aktoreen zerrenda fitxategi batean gorde

public void aktoreakFitxategianSartu()

||Aurre: -

||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

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: $O(n \log n)$

Kodea

Aktore.java

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

ArrayPelikulak.java

```
1 package aktorePelikulaPackage;
2 import java.util.ArrayList;
3 import java.util.Iterator;
4
```

```
5 public class ArrayPelikulak {
6
7     private ArrayList<Pelikula> lista;
8
9     public ArrayPelikulak(){
10         this.lista = new ArrayList<Pelikula>();
11     }
12
13     private Iterator<Pelikula> getNireIteradorea(){
14         return this.lista.iterator();
15     }
16
17     public boolean badagoPelikula(String pIzenburua){//Pelikula bat emanda, listan
18         ↪ dagoen esango digu
19         boolean dago = false;
20         Pelikula pelikula = null;
21         Iterator<Pelikula> itr = this.getNireIteradorea();
22         while(itr.hasNext() && !dago){
23             pelikula = itr.next();
24             if(pelikula.getIzenburua().equals(pIzenburua)){
25                 dago = true;
26             }
27         }
28         return dago;
29     }
30
31     public void gehituPelikula(Pelikula pPeli){
32         this.lista.add(pPeli);
33     }
34
35     public int pelikulaKopurua() {
36         return this.lista.size();
37     }
38
39     public void erreseteatu() {
40         this.lista.clear();
41     }
42
43     public void ezabatuAktorea(Aktore pAktore) {
44         Pelikula peli = null;
45         ArrayAktoreak lista=null;
46         Iterator<Pelikula> itr = this.getNireIteradorea();
47         while(itr.hasNext()) {
48             peli = itr.next();
49             lista=peli.getListaAktoreak();
50             lista.ezabatuAktorea(pAktore);
51         }
52     }
53 }
```

```
50     }
51   }
52 }
```

ArrayAktoreak.java

```
1  package aktorePelikulaPackage;
2  import java.util.*;
3
4  public class ArrayAktoreak {
5
6      private ArrayList<Aktore> lista;
7
8      public ArrayAktoreak(){
9          this.lista = new ArrayList<Aktore>();
10     }
11
12     private Iterator<Aktore> getNireIteradorea(){
13         return this.lista.iterator();
14     }
15
16     public boolean badagoAktorea(String pIzenAbizena){//Aktore bat pasata, listan
17     ↪ dagoen esango digu
18         boolean dago = false;
19         Aktore aktor = null;
20         Iterator<Aktore> itr = this.getNireIteradorea();
21         while(itr.hasNext() && !dago){
22             aktor = itr.next();
23             if(aktor.getIzenAbizena().equals(pIzenAbizena)){
24                 dago = true;}
25         }
26         return dago;
27     }
28
29     public void gehituAktorea(Aktore pAktore){
30         if(!(this.badagoAktorea(pAktore.getIzenAbizena()))){
31             this.lista.add(pAktore);
32         }
33     }
```

```
33
34     public void ezabatuAktorea(Aktore pAktore) {
35         this.lista.remove(pAktore);
36     }
37
38     public int aktoreKopurua() {
39         return this.lista.size();
40     }
41
42     public void erreseteatu() {
43         this.lista.clear();
44     }
45 }
```

ListaAktoreak.java

```
1 package aktorePelikulaPackage;
2 import java.util.*;
3 import java.io.FileNotFoundException;
4 import java.io.FileWriter;
5 import java.io.IOException;
6
7 public class ListaAktoreak {
8
9     private static ListaAktoreak nireListaAktoreak = null;
10    private HashMap<String,Aktore> map;
11
12    private ListaAktoreak(){
13        this.map = new HashMap<String,Aktore>();
14    }
15
16    public static ListaAktoreak getNireListaAktoreak(){
17        if(nireListaAktoreak == null){
18            nireListaAktoreak = new ListaAktoreak();
19        }
20        return nireListaAktoreak;
21    }
22
23 }
```



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

```
70
71
72 public String[] listaOrdenatua(){
73     String[] lista = this.pasaArray();
74     quickSort(lista, 0, lista.length-1);
75     return lista;
76 }
77
78
79 private void quickSort(String[] lista, int hasiera, int bukaera){
80     if ( bukaera - hasiera > 0 ) { // taulan elementu bat baino gehiago
81         int indizeaZatiketa = zatiketa(lista, hasiera, bukaera);
82         quickSort(lista, hasiera, indizeaZatiketa - 1);
83         quickSort(lista, indizeaZatiketa + 1, bukaera);
84     }
85 }
86
87 private int zatiketa(String[] lista, int i, int f){
88
89     String lag = lista[i];
90     int ezker = i;
91     int eskuin = f;
92     while ( ezker < eskuin ){
93         lag.toUpperCase();
94         lista[ezker].toUpperCase();
95         lista[eskuin].toUpperCase();
96         while ( lista[ezker].compareTo(lag) <= 0 && ezker < eskuin)
97             ezker++;
98         while ( lista[eskuin].compareTo(lag) > 0 )
99             eskuin--;
100         if ( ezker < eskuin )
101             swap(lista, ezker, eskuin);
102     }
103     lista[i] = lista[eskuin];
104     lista[eskuin] = lag;
105     return eskuin;
106 }
107
108
109 private void swap(String[] lista, int one, int two) {
110     String temp = lista[one];
111     lista[one] = lista[two];
112     lista[two] = temp;
113 }
114
115
```

```
116 public ArrayPelikulak aktoreBatenListaPelikulak(String pAktore){
117     Aktore aktor = this.bilatuAktorea(pAktore);
118     if(aktor == null){
119         return null;
120     }else{
121         return aktor.getListPelikula();
122     }
123 }
124
125 public void aktoreakFitxategianSartu(){
126     FileWriter fitxategia1 = null;
127     try {
128         fitxategia1 = new FileWriter("./FilmsActors20162017Fitxategia.txt");
129         Iterator<String> it =
130             ↳ ListaAktoreak.getNireListaAktoreak().getList().keySet().iterator();
131         String lerroa = null;
132         while (it.hasNext()) {// Lerro bakoitza fitxategian idazten dugu
133             lerroa = it.next();
134             if(it.hasNext()){
135                 fitxategia1.write(lerroa + " &&& " + "\n");
136             }else{
137                 fitxategia1.write(lerroa + "\n");
138             }
139             fitxategia1.close();
140         }
141         catch (FileNotFoundException e) {
142             System.out.println("Fitxeroa ez da existitzen. ");
143         }
144         catch (IOException e) {
145             System.out.println("Fitxategiaren idazketak huts egin du. ");
146         }
147     }
148 }
```

ListaPelikula.java

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

```

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

```

```
85     }
86   }
87   entrada.close();
88   } catch(IOException e) {e.printStackTrace();}
89 }
90 }
```

Pelikula.java

```
1 package aktorePelikulaPackage;
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 }
```

```
31 public void listaAktoreanGehitu(Aktore pAktor){//Hemen, aktore bat sartuko dugu  
    ↪ pelikularen listaAktorean  
32     this.lista.gehituAktorea(pAktor);  
33 }  
34 }
```

JUnitak

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 }

```



ArrayPelikulakTest.java

```

1 package aktorePelikulaPackage;
2 import static org.junit.Assert.*;

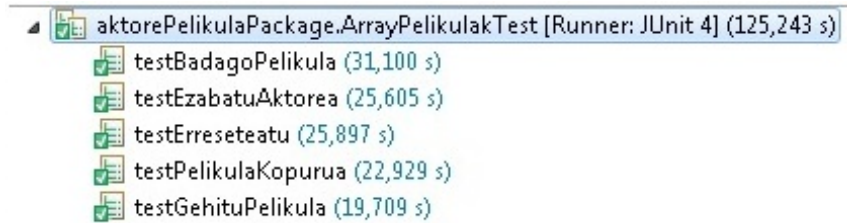
```

```

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 }
```



ArrayAktoreakTest.java

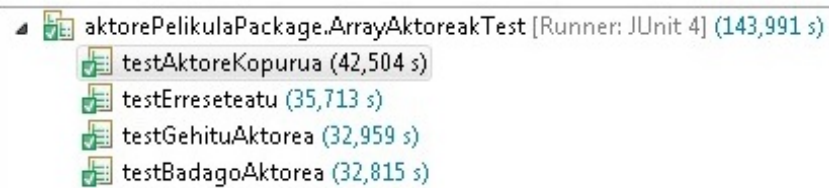
```
1 package aktorePelikulaPackage;
2 import static org.junit.Assert.*;
3 import java.io.FileNotFoundException;
4 import java.io.IOException;
5 import org.junit.After;
6 import org.junit.Before;
7 import org.junit.Test;
8
9 public class ArrayAktoreakTest {
10
11     ArrayAktoreak lista;
12     Aktore 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     @Test
32     public void testBadagoAktorea() throws FileNotFoundException, IOException {
33         lista.erreseteatu();
```

```

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

```

```
76     lista = peli1.getListaAktoreak();
77     lista.erreseteatu();
78     assertEquals(lista.aktoreKopurua(), 0);
79 }
80 }
```



aktorePelikulaPackage.ArrayAktoreakTest [Runner: JUnit 4] (143,991 s)

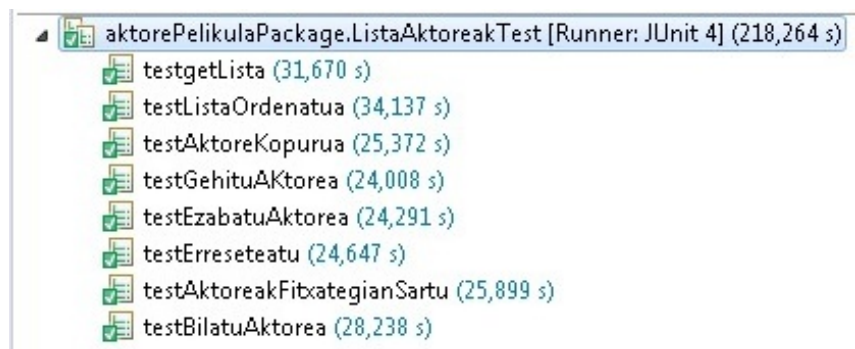
- testAktoreKopurua (42,504 s)
- testErreseteatu (35,713 s)
- testGehituAktorea (32,959 s)
- testBadagoAktorea (32,815 s)

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     @Before
18     public void setUp() throws Exception {
19         aktor1 = new Aktore("AdeiArias");
20         aktor2 = new Aktore("AnderPrieto");
21         lista2.fitxeroaErakutsi("./FilmsActors20162017.txt");
22     }
23
24     @After
```

```
25 public void tearDown() throws Exception {
26     aktor1=null;
27     aktor2=null;
28     lista1.erreseteatu();
29     lista2=null;
30 }
31
32 @Test
33 public void testEzabatuAktorea() throws FileNotFoundException, IOException {
34     assertEquals(lista1.aktoreKopurua(), 1283445);
35     lista1.ezabatuAktorea("AdeiArias");
36     assertEquals(lista1.aktoreKopurua(), 1283445);
37     lista1.ezabatuAktorea("Devon, Tony");
38     assertEquals(lista1.aktoreKopurua(), 1283444);
39 }
40
41 @Test
42 public void testGehituAktorea() throws FileNotFoundException, IOException {
43     assertEquals(lista1.aktoreKopurua(), 1283445);
44     lista1.gehituAktorea("AnderPrieto", aktor2);
45     assertEquals(lista1.aktoreKopurua(), 1283446);
46 }
47
48 @Test
49 public void testBilatuAktorea() {
50     lista1.gehituAktorea("AdeiArias", aktor1);
51     assertEquals(lista1.bilatuAktorea("AdeiArias"), aktor1);
52     assertEquals(lista1.bilatuAktorea("AnderPrieto"), null);
53     assertNotEquals(lista1.bilatuAktorea("Tarantino, Quentin"), aktor1);
54 }
55
56 @Test
57 public void testgetLista(){
58     lista1.getLista();
59     assertNotNull(lista1);
60 }
61
62 @Test
63 public void testAktoreKopurua() {
64     assertEquals(lista1.aktoreKopurua(), 1283445);
65 }
66
67 @Test
68 public void testErreseteatu() {
69     assertEquals(lista1.aktoreKopurua(), 1283445);
70     lista1.erreseteatu();
```

```
71     assertEquals((lista1.aktoreKopurua()), 0);
72 }
73
74 @Test
75 public void testListaOrdenatua() {
76     lista1.listaOrdenatua();
77     assertNotNull(lista1);
78 }
79
80 @Test
81 public void testAktoreakFitxategianSartu() {
82     lista1.aktoreakFitxategianSartu();
83 }
84 }
```



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
```

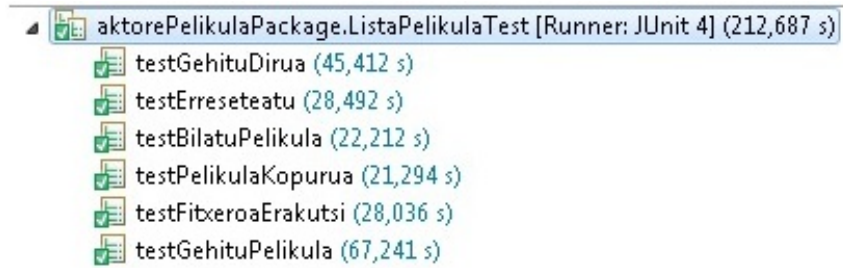


```

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
           ↪ gehitu
34         ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
35         peli4 = ListaPelikula.getNireListaPelikula().bilatuPelikula("Eager to Die");
36         lista1.gehituDirua("Eager to Die", 20.00);
37         assertEquals(peli4.getDirua(), 65.00, 2); //listako edozein elementuri dirua
           ↪ gehitu
38     }
39
40     @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     assertEquals(lista1.bilatuPelikula("Joker"), peli1); //elementua ez dago
85         ↪ listan
86 }
87
88 @Test
89 public void testFitxeroaErakutsi() throws FileNotFoundException, IOException {
90     ListaPelikula.getNireListaPelikula().fitxeroaErakutsi("./FilmsActors20162017.txt");
91 }
92 }
```



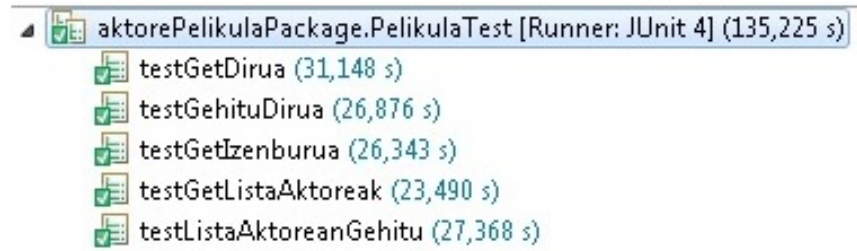
PelikulaTest.java

```
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.getListaAktoreak();
75     assertEquals(lista.aktoreKopurua(), 6);
76     lista.gehituAktorea(aktor1);
77     assertEquals(lista.aktoreKopurua(), 7);
78 }
```

79

}



Ondorioak

Hasteko, praktika honetan, datu kantitate oso handiak kudeatzen ikasi dugu, horretarako datu egitura desberdinak erabili ditugu. Gainera, lerro batean datuak sakabanatzeko *split* metodoa ikasi dugu.

Bestalde, konturatu gara, datu egitura batzuk, beste datu egiturak baino efizienteagoak direla eta honek, gure programaren exekuzio denboran eragin oso handia izan dezake.

Adibidez, *ListaAktorea* eta *ListaPelikula* inplementatzeko garaian, *ListaAktorea HashMap* motakoa zela definitu genuen eta *listaPelikula ArrayList* motakoa. Honek, gure programaren efizientzia asko jaistea egin zuen. Honen ondorioz, bi *HashMap* jartzea erabaki genuen, eta kasu honetan, gure programari oso gutxi kostatzen zitzaion datu guztiak kargatzea.

Amaitzeko, etorkizuneko lan batean, datu asko kudeatzeko beharra baldin badago, badakigu gure programa ahalik eta eraginkor izan dadin, zein datu egitura erabili beharko ditugun.

Erreferentziak

- [1] Gojenola, Koldo. Datu-Egiturak eta Algoritmoak: proiektua – Aktoreak eta pelikulak kudeatu. egela.ehu.eus, 2019. URL https://egela.ehu.eus/pluginfile.php/2282865/mod_resource/content/12/Praktika%202019-2020%20ikasturtea-Fase1-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] Ezezaguna. Quicksort - Wikipedia. en.wikipedia.org, 2019. URL <https://en.wikipedia.org/wiki/Quicksort>.