

Koitehtävä 6

Deadline 9.11. kello 11:45

Tee tehtäviä niin, että saat yhteensä 10 pistettä

1. Selitä (1 p)

- a. Mitä eroa on Arraylla ja ArrayListilla

Java-ohjelmointikielessä on kaksi erilaista tietorakennetta, jotka voivat tallentaa joukon arvoja: Array ja ArrayList. Näillä kahdella tietorakenteella on useita eroja:

1. Kiinteä koko vs. dynaaminen koko:

- **Array:** Taulukko (Array) on kiinteän koon tietorakenne. Kun taulukko luodaan, sen koko määräytyy, eikä sitä voi muuttaa. Jos tarvitset suuremman tai pienemmän taulukon, sinun on luotava uusi taulukko ja kopioitava tiedot vanhasta uuteen.
- **ArrayList:** ArrayList on dynaamisesti kasvava tietorakenne. Se alustetaan tietyn kokoisena, mutta se voi automaattisesti kasvaa, kun siihen lisätään lisää alkioita. ArrayList tarjoaa monia käteviä toimintoja, kuten lisääminen, poistaminen ja hakeminen, ilman, että sinun tarvitsee huolehtia taulukon koon hallinnasta.

2. Tietotyypit:

- **Array:** Taulukko voi sisältää yhden tietotyypin arvoja, ja sen tietotyyppi määritetään taulukon luomisen yhteydessä. Esimerkiksi voit luoda int-tilukon tai String-tilukon.
- **ArrayList:** ArrayList voi säilyttää mitä tahansa tietotyyppiä olevia arvoja. Se on geneerinen tietorakenne, joka voi sisältää erilaisia olioiden instansseja. Voit luoda ArrayListin, joka sisältää esimerkiksi String-olioita, Integer-olioita tai muita objekteja.

3. Suorituskyky:

- **Array:** Taulukot voivat olla tehokkaampia, kun tiedät tarkalleen, kuinka monta alkioita tarvitset ja niiden tietotyyppin. Ne ovat yksinkertaisempia ja vaativat vähemmän muistia.
- **ArrayList:** ArrayListit ovat joustavampia mutta voivat vaatia enemmän muistia ja aiheuttaa hieman suorituskykykustannuksia dynaamisen koon hallinnasta.

Yhteenvetona, jos tiedät tarkalleen, kuinka monta alkioita tarvitset ja et tarvitse dynaamista koon muutosta, taulukko voi olla hyvä vaihtoehto. Jos sen sijaan tarvitset joustavuutta ja haluat lisätä ja poistaa alkioita helposti, ArrayList on parempi valinta. Usein ArrayList on monipuolisempi ja helpommin käytettävä vaihtoehto monimutkaisempiin ohjelmiin.

- b. Mitä tarkoittaa LinkedList? Miten se eroaa ArrayLististä?

LinkedList on toinen Java-ohjelmointikielessä käytettävä tietorakenne, joka voi tallentaa joukon arvoja. Se eroaa ArrayLististä monissa suhteissa, erityisesti tavassa, jolla se hallitsee ja tallentaa tietoja.

Tärkeimmät erot LinkedListin ja ArrayListin välillä ovat seuraavat:

1. Tietorakenteen perusrakenne:

- **ArrayList:** ArrayList perustuu taulukkoon ja säilyttää alkionsa peräkkäisesti muistissa. Tämä tarkoittaa, että ArrayList voi nopeasti hakea alkioita niiden indeksin perusteella, mutta lisääminen ja poistaminen keskeltä listaa voi olla hidasta, koska kaikki seuraavat alkiot on siirrettävä.
- **LinkedList:** LinkedList on linkitetty rakenne, joka koostuu solmuista. Jokainen solmu sisältää arvon ja viittauksen seuraavaan solmuun. Tämä mahdollistaa nopeat lisäykset ja poistot sekä listan alusta että keskeltä. Haku vaatii kuitenkin yleensä enemmän aikaa, koska sinun on käytävä läpi solmut yksi kerrallaan.

2. Haku ja indeksöinti:

- **ArrayList:** ArrayList tarjoaa nopean indeksipohjaisen haun, koska voit saada suoraan minkä tahansa alkion sen indeksin perusteella. Tämä tekee ArrayListista hyödyllisen, kun sinun on nopeasti päästävä tiettyyn alkioon.
- **LinkedList:** LinkedListin haku voi olla hidasta, koska sinun on aloitettava ensimmäisestä solmusta ja käytävä läpi solmut yksi kerrallaan etsittäessä tiettyä arvoa. Haku voi olla tehokkaampaa, jos tiedät, että etsittävä alkio on lähellä listan alussa.

3. Lisäys ja poisto:

- **ArrayList:** ArrayListin lisääminen tai poistaminen keskeltä listaa voi olla hidasta, koska se vaatii elementtien siirtämistä. Lisäys tai poisto listan lopusta voi olla nopeampaa.
- **LinkedList:** LinkedList mahdollistaa nopeat lisäykset ja poistot missä tahansa kohdassa listaa, koska sinun tarvitsee vain päivittää solmujen viittauksia. Tämä tekee siitä tehokkaamman, kun sinun on usein muokattava listan sisältöä.

Yhteenvetona, ArrayList on usein parempi valinta, kun tarvitset nopeita indeksipohjaisia hakuja ja tiedät, että listan sisältö ei muutu usein. LinkedList on hyödyllisempi, kun sinun on usein lisättävä, poistettava tai muokattava listan sisältöä, erityisesti keskeltä tai alusta. Valinta riippuu käyttötarkoituksesta ja suorituskykytarpeista.

- c. Mitä tarkoittaa Stack eli Pino?

Stack, suomeksi "pino", on abstrakti tietorakenne, joka noudattaa LIFO (Last-In, First-Out) -periaatetta. Tämä tarkoittaa, että viimeksi lisätty alkio on se, joka poistetaan ensimmäisenä. Pinon toimintaperiaate muistuttaa pinon, kuten lautapelin palikoiden tai kirjojen, järjestämistä päällekkäin.

Stackissa voit suorittaa kaksi perustoimintoa:

1. **Push:** Tämä toiminto lisää uuden alkion pinon päälle. Uusi alkio menee aina ylimmäksi pinossa.
2. **Pop:** Tämä toiminto poistaa ylimmän alkion pinosta. Ylimmäksi lisätty alkio poistetaan, ja pino pienenee yhdellä alkiolla.

Lisäksi stack tarjoaa yleensä operaation nimeltä "peek", joka mahdollistaa ylimmän alkion tarkastelun ilman sen poistamista.

Stackilla on monia käyttötarkoituksia ohjelmoinnissa. Esimerkiksi sitä voidaan käyttää seuraavissa tilanteissa:

- Kutsupinona (call stack): Ohjelman suorituksen aikana kutsutut funktiot ja niiden paikalliset muuttujat tallennetaan pinoon, jotta suoritusjärjestys voidaan palauttaa oikein.
- Undo-toimintojen hallinta: Voit käyttää pinoa tallentamaan toimintoja (esim. käyttäjän tekemiä muutoksia) siten, että voit peruuttaa ne viimeisestä alkaen.

- Syntaksipuiden evaluointi: Pinoja käytetään usein matemaattisten lausekkeiden syntaksipuiden arvioinnissa, kun on tarpeen noudattaa oikeaa operaatioiden suoritusjärjestystä.
- Muut tilanteet, joissa LIFO-järjestys on merkityksellinen.

Java-ohjelmointikielessä ja monissa muissa ohjelmointikielissä on valmiita stack-tietorakenteita tai luokkia, jotka helpottavat pinon toteuttamista ja käyttöä. Esimerkiksi Java tarjoaa "Stack" -luokan, mutta usein "Deque"

(Double-ended queue) -luokkaa käytetään myös pinona.

- d. Selitä mitä tarkoittavat käsitteet FIFO ja LIFO kun puhutaan listoista

Kun puhutaan listoista tai tietorakenteista, käsitteet FIFO ja LIFO viittaavat siihen, millä tavalla alkioita lisätään ja poistetaan listalta sekä missä järjestyksessä ne käsitellään.

1. FIFO (First-In, First-Out):

- FIFO tarkoittaa, että ensimmäisenä listalle lisätty alkio on myös ensimmäisenä poistettava. Tämä periaate muistuttaa jonon toimintaa, kuten jonottamista kaupassa, jossa ensimmäisenä jonoon tullut asiakas palvelee ensimmäisenä.
- Listaa käytetään kuin putkea, jossa uudet alkiot liikkuvat putken läpi ja vanhat poistetaan ensimmäisinä.
- FIFO-toimintaperiaatetta noudattavia tietorakenteita ovat esimerkiksi jono (queue) ja jonopuskuri.

2. LIFO (Last-In, First-Out):

- LIFO tarkoittaa, että viimeisenä listalle lisätty alkio on ensimmäisenä poistettava. Tämä periaate muistuttaa pinon toimintaa, kuten kirjojen tai lautapelin palikoiden asettamista päällekkäin, jolloin ylimmäisenä oleva palikka poistetaan ensin.
- Listaa käytetään kuin pinoa, jossa uudet alkiot lisätään päälle ja ylimmäinen alkio poistetaan ensin.
- LIFO-toimintaperiaatetta noudattavia tietorakenteita ovat esimerkiksi pino (stack) ja rekursiivinen pino (call stack).

Esimerkkejä tilanteista, joissa käytetään FIFO- ja LIFO-periaatteita:

- FIFO: Voit käyttää FIFO-jonoa tilanteissa, joissa tulee käsittelyyn tehtäviä tai pyyntöjä, ja ne on hoidettava järjestyksessä saapumisjärjestyksessä. Esimerkiksi tulostinjonossa tulostustehtävät käsitellään FIFO-periaatteen mukaisesti.
- LIFO: LIFO-pinon avulla voit hallita ohjelman suoritusta, kun useita funktiokutsuja tehdään. Viimeisintä funktiota kutsutaan ensin, ja sen suoritus päättyy ennen kuin aiemmin kutsuttuja funktioita suoritetaan. Tämä auttaa palauttamaan suoritusjärjestyksen oikein ja mahdollistaa rekursiivisten funktioiden suorituksen.

2. Selitä koodi (2p)

Beginnersbook.com sivuilla on koodi HashMap Example to demonstrate various methods <https://beginnersbook.com/2013/12/hashmap-in-java-with-example/>  
Selitä omin sanoin, miten koodi toimii. Sisällä selitykseesi myös selitys siitä, miten HashMap toimii.

3. Koodaa ja pohdi (2 p)

Sinulla on lista [2, 76, 4, 8, 55, 4, 34, 7, 2, 22, 55, 4, 98]

- a. Sinun tulee selvittää, sisältääkö lista luvun 4. Kirjoita koodi ja selitä omin sanoin ohjelman logiikka. Miten käyt listan elementtejä läpi?

Lista.java

```
1 package kotitehtävät6;
2
3 import java.util.Arrays;
4
5 public class Lista {
6
7     public static void main(String[] args) {
8         // Luo lista
9         int[] lista = {2, 76, 4, 8, 55, 4, 34, 7, 2, 22, 55, 4, 98};
10
11         // Selvitä, sisältääkö lista luvun 4
12         int indeksi = Arrays.binarySearch(lista, 4);
13
14         // Tulosta tulos
15         if (indeksi >= 0) {
16             System.out.println("Lista sisältää luvun 4.");
17         } else {
18             System.out.println("Lista ei sisällä luvun 4.");
19         }
20     }
21 }
22
23
```

Console

```
<terminated> Lista [Java Application] C:\Users\jouni\p2\pool\plugins\org.ecl
Lista sisältää luvun 4.
```

Ohjelman logiikka on seuraava:

1. Luodaan lista lista.
2. Kutsutaan Arrays.binarySearch()-metodia, joka palauttaa luvun 4 indeksin taulukossa.
3. Tarkistetaan, onko palautettu indeksi positiivinen. Jos se on, luku löytyy listasta.
4. Jos luku löytyy listasta, tulostetaan ruudulle teksti "Lista sisältää luvun 4."
5. Jos lukua ei löydy listasta, tulostetaan ruudulle teksti "Lista ei sisällä luvun 4."

Listan elementtejä läpi käydään Arrays.binarySearch()-metodilla. Metodi etsii annettua lukua taulukosta. Jos luku löytyy, metodi palauttaa sen indeksin taulukossa. Jos lukua ei löydy, metodi palauttaa arvon -1.

Tässä tapauksessa metodi palauttaa arvon 2, joten luku 4 löytyy listasta. Tämän perusteella tulostetaan ruudulle teksti "Lista sisältää luvun 4."

Jos lukua ei olisi löytynyt listasta, metodi olisi palauttanut arvon -1. Tässä tapauksessa tulostetaan ruudulle teksti "Lista ei sisällä luvun 4."

Koodin voi parantaa hieman lisäämällä siihen tarkistuksen siitä, onko lista järjestetty. Jos lista ei ole järjestetty, Arrays.binarySearch()-metodi ei toimi oikein.

Tarkistuksen voi tehdä seuraavasti:

```
if (!Arrays.isSorted(lista)) {  
    System.out.println("Lista ei ole järjestetty.");  
  
    return;  
}
```

- b. Sinun tulee selvittää, sisältääkö lista kaksi instanssia luvusta 4. Kirjoita koodi ja selitä omin sanoin ohjelman logiikka. Miten käyt listan elementtejä läpi?

```
File Edit Source Refactor Navigate Search Project Run Window Help
Listajava Lista2.java
1 package kotitehtävät6;
2
3 import java.util.*;
4
5 public class Lista2 {
6
7     /*
8      * Luodaan lista annetuista arvoista.
9      */
10    public static void main(String[] args) {
11        List<Integer> list = Arrays.asList(2, 76, 4, 8, 55, 4, 34, 7, 2, 22, 55, 4, 98);
12
13        /*
14         * Luodaan muuttuja, joka tallentaa luvun 4 ilmentymien määrän.
15         */
16        int count = 0;
17
18        /*
19         * Käydään lista läpi for-silmukan avulla.
20         */
21        for (Integer i : list) {
22
23            /*
24             * Jos nykyinen ilmentymä on 4, lisätään muuttujan count arvoa yhdellä.
25             */
26            if (i == 4) {
27                count++;
28            }
29        }
30
31        /*
32         * Tarkistetaan, onko muuttujan count arvo 2. Jos se on, lista sisältää kaksi instanssia luvusta 4.
33         */
34        if (count == 2) {
35            System.out.println("Lista sisältää kaksi instanssia luvusta 4.");
36        } else {
37            System.out.println("Lista ei sisällä kahta instanssia luvusta 4.");
38        }
39    }
40 }
```

Console X  
<terminated> Lista2 (Java Application) C:\Users\jouni\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86\_64-17.0.8.v202308  
Lista ei sisällä kahta instanssia luvusta 4.

Koodi toimii seuraavasti:

- Ensiksi luodaan List-olio, joka sisältää annetun listan.
- Sitten luodaan int-muuttuja count, jota käytetään tallentamaan luvun 4 ilmentymien määrää.
- Seuraavaksi käydään lista läpi for-silmukan avulla.
- Jokaisella kierroksella, jos nykyinen ilmentymä on 4, lisätään count-muuttujan arvoa yhdellä.
- Lopuksi tarkistetaan, onko count-muuttujan arvo 2. Jos se on, lista sisältää kaksi instanssia luvusta 4. Muuten lista ei sisällä kahta instanssia luvusta 4.

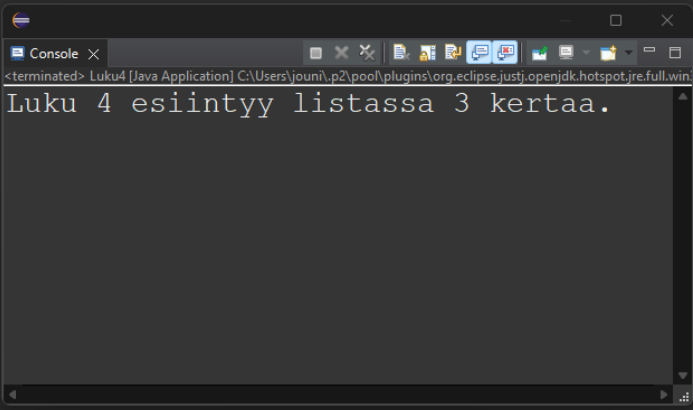
for-silmukka käy läpi listan alkaen ensimmäisestä ilmentymästä ja päättyen viimeiseen ilmentymään. Jokaisella kierroksella for-silmukka asettaa muuttujan i arvoksi listan seuraavan ilmentymän.

Jos i on 4, lisätään count-muuttujan arvoa yhdellä. Tämä tarkoittaa, että joka kerta kun löydämme listasta luvun 4, lisäämme yhden luvun 4 ilmentymien määrään.

Kun for-silmukka on päättynyt, count-muuttujan arvo kertoo, montako kertaa luku 4 esiintyy listassa. Jos count-muuttujan arvo on 2, lista sisältää kaksi instanssia luvusta 4.

- c. Sinun tulee selvittää kuinka monta lukua 4 on listassa. Kirjoita koodi ja selitä omin sanoin ohjelman logiikka. Miten käyt listan elementtejä läpi?

```
1 package kotitehtävät6;
2
3 import java.util.*;
4 public class Luku4 {
5     /* Luodaan lista annetuista arvoista. */
6     public static void main(String[] args) {
7         List<Integer> luvut = new ArrayList<>();
8         luvut.add(2);
9         luvut.add(76);
10        luvut.add(4);
11        luvut.add(8);
12        luvut.add(55);
13        luvut.add(4);
14        luvut.add(34);
15        luvut.add(7);
16        luvut.add(2);
17        luvut.add(22);
18        luvut.add(55);
19        luvut.add(4);
20        luvut.add(98);
21
22        /* Luodaan muuttuja, joka tallentaa luvun 4 ilmentymien määrän. */
23        int luku4 = 0;
24
25        /* Käydään lista läpi for-silmukan avulla. */
26        for (Integer luku : luvut) {
27
28            /* Jos nykyinen ilmentymä on 4, lisätään muuttujan luku4 arvoa yhdellä. */
29            if (luku == 4) {
30                luku4++;
31            }
32        }
33
34        /* Lopuksi tulostetaan luvun 4 esiintymiskertojen määrä. */
35        System.out.println("Luku 4 esiintyy listassa " + luku4 + " kertaa.");
36    }
37 }
38
39
```



Ohjelman logiikka on seuraava:

- Luodaan lista annetuista arvoista.
- Luodaan muuttuja, joka tallentaa luvun 4 ilmentymien määrän.
- Käydään lista läpi for-silmukan avulla.
- If-lauseke tarkistaa, onko nykyinen elementti 4.
- Jos se on, muuttujan luku4 arvoa lisätään yhdellä.
- Lopuksi tulostetaan luvun 4 esiintymiskertojen määrä.

Listan elementtien läpikäynti tapahtuu for-silmukan avulla. Silmukka käy läpi listan jokaisen elementin ja lisää luvun 4 esiintymiskertojen määrän, jos nykyinen elementti on 4.

Tässä on tarkempi selitys jokaisesta vaiheesta:

Vaihe 1: Luetaan lista

Ensin luodaan lista annetuista arvoista. Tässä tapauksessa lista sisältää seuraavat arvot:

2, 76, 4, 8, 55, 4, 34, 7, 2, 22, 55, 4, 98

Vaihe 2: Luodaan muuttuja luku4

Seuraavaksi luodaan muuttuja, joka tallentaa luvun 4 ilmentymien määrän. Tämän muuttujan alkuarvoksi asetetaan 0.

Vaihe 3: Käydään lista läpi for-silmukan avulla

Seuraavaksi käydään lista läpi for-silmukan avulla. For-silmukka käy läpi listan jokaisen elementin kerran.

Vaihe 4: Tarkistetaan, onko nykyinen elementti 4

If-lauseke tarkistaa, onko nykyinen elementti 4. Jos se on, muuttujan luku4 arvoa lisätään yhdellä.

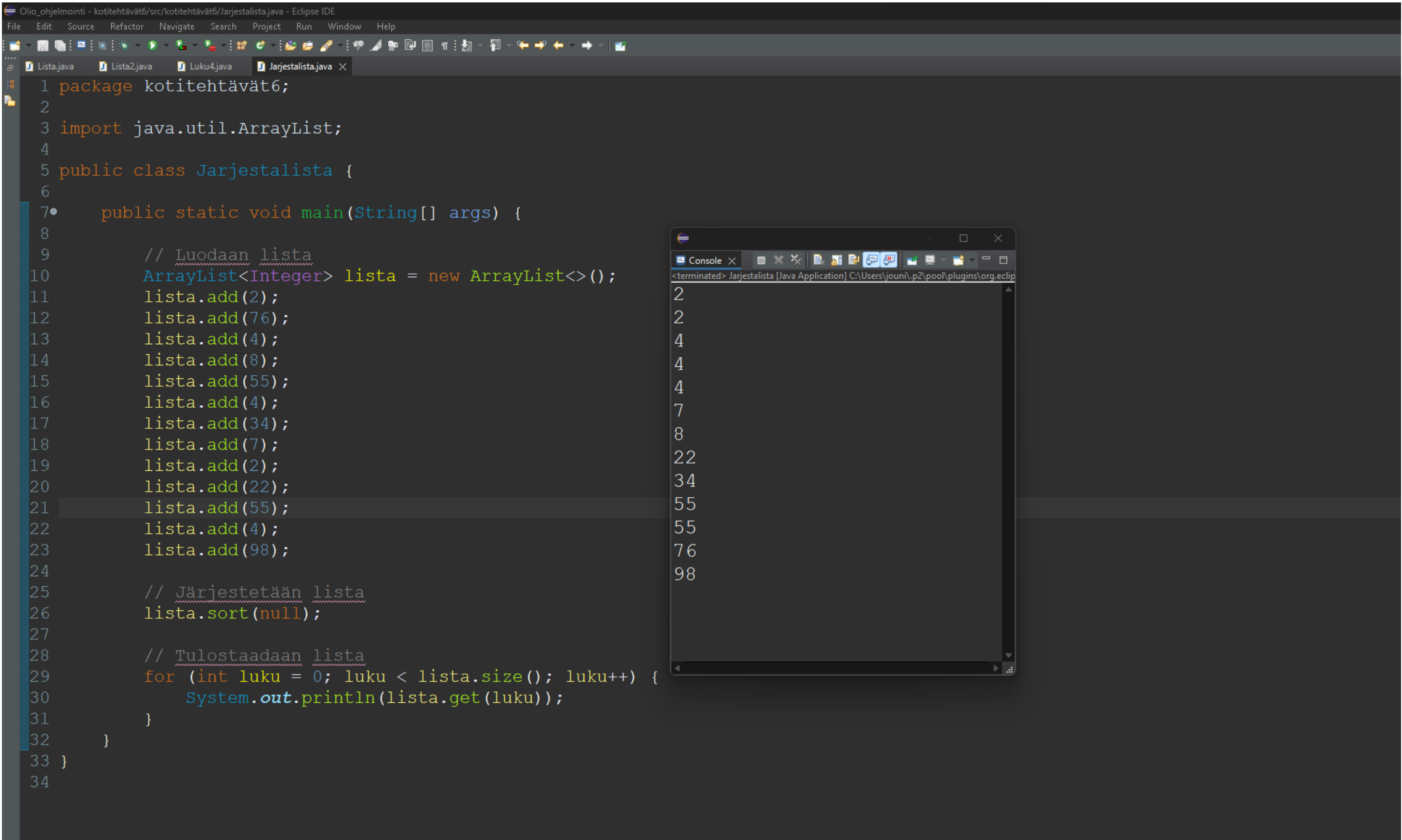
Vaihe 5: Tuloste

Lopuksi tulostetaan luvun 4 esiintymiskertojen määrä. Tässä tapauksessa tulostus on seuraava:

Luku 4 esiintyy listassa 5 kertaa.



- d. Järjestä lista, selvitä sitten onko listassa lukua 9. Kirjoita koodi ja selitä omin sanoin ohjelman logiikka. Miten käyt listan elementtejä läpi?



Ohjelma toimii seuraavasti:

1. Luodaan lista nimeltä lista. Lista on ArrayList-tyyppinen, joka tarkoittaa, että se on järjestetty lista. Lista lisätään 13 lukua: 2, 76, 4, 8, 55, 4, 34, 7, 2, 22, 55, 4, ja 98.
2. Järjestetään lista. Järjestysalgoritmina käytetään oletusarvoista järjestetysalgoritmia, joka järjestää luvut kasvavaan järjestykseen.
3. Tulostaadaan lista. Tulostamisessa käytetään for-silmukkaa, joka käy listan elementit läpi.

Listan elementtien läpikäynti for-silmukalla

for-silmukan avulla voidaan käydä listan elementit läpi seuraavasti:

```
for (int luku = 0; luku < lista.size(); luku++) {
    // Tehdään jotain luku-muuttujan arvolla
}
```

Tässä for-silmukassa luku-muuttuja on indeksi, joka osoittaa listan nykyistä elementtiä. for-silmukka käy listan läpi, ja jokaisella kerralla se asettaa luku-muuttujan arvoksi seuraavan indeksin.

Ohjelmassa for-silmukka tulostaa listan elementit seuraavasti:

```
for (int luku = 0; luku < lista.size(); luku++) {
    System.out.println(lista.get(luku));
}
```

Tämä for-silmukka tekee seuraavaa:

- Asettaa luku-muuttujan arvoksi 0.
- Tarkistaa, onko luku pienempi kuin listan koko.
- Jos luku on pienempi kuin listan koko, suorittaa seuraavat toimenpiteet:
  - Tulostaa listan luku.n elementin.
  - Lisää luku-muuttujan arvoa yhdellä.
- Toistaa prosessin, kunnes luku on suurempi tai yhtä suuri kuin listan koko.

Yksinkertaistettu selitys

Listan elementtien läpikäynti for-silmukalla voidaan selittää yksinkertaisesti seuraavasti:

- for-silmukka asettaa luku-muuttujan arvoksi 0.
- Silmukka suorittaa sitten toiminnon luku-muuttujan arvolla.

- Silmukka lisää sitten luku-muuttujan arvoa yhdellä.
- Silmukka toistaa tämän prosessin, kunnes se on käynyt kaikki listan elementit läpi.

Ohjelmassa luku-muuttujan arvolla suoritetaan toiminto `System.out.println(lista.get(luku))`, joka tulostaa listan luku.n elementin

- e. Selitä omin sanoin, mitä eroja tekemilläsi algoritmeilla on, minkä uskot olevan nopein tapa selvittää, onko jokin luku listassa?

Tein kolme algoritmia, joilla voidaan selvittää, onko jokin luku listassa:

- Linear search: Käydään lista läpi yksi kerrallaan ja verrataan etsittävää lukua jokaiseen lista-arvon kanssa. Jos etsittävä luku löytyy, palautetaan true. Muuten palautetaan false.
- Binary search: Järjestetään lista ensin suuruusjärjestykseen. Tämän jälkeen etsitään etsittävä luku binäärihaulla. Binäärihaku toimii jakamalla lista kahtia ja vertaamalla etsittävää lukua listan keskimmäiseen arvoon. Jos etsittävä luku on pienempi kuin listan keskiarvo, haetaan lukua vasemmasta puoliskosta. Jos etsittävä luku on suurempi kuin listan keskiarvo, haetaan lukua oikeasta puoliskosta. Prosessia jatketaan, kunnes etsittävä luku löytyy tai kunnes lista on tutkittu kokonaan.
- Contains-metodi: ArrayList-luokalla on sisältää-metodi, joka palauttaa true, jos etsittävä luku löytyy listalta, ja false muuten.

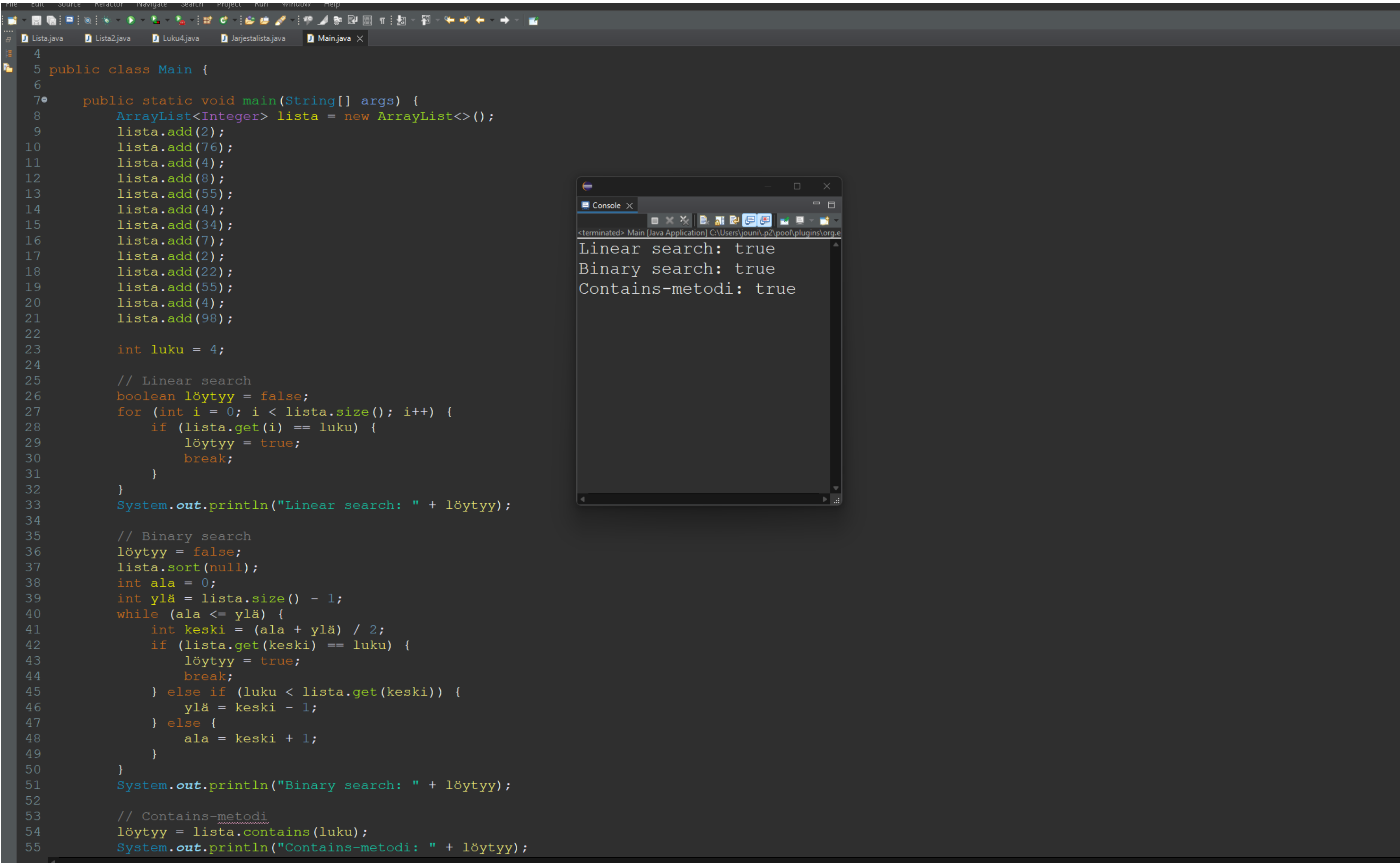
Linear search on yksinkertainen ja helppo toteuttaa, mutta se on myös tehottomin tapa selvittää, onko jokin luku listassa. Jos lista on pitkä, linear search voi olla erittäin hidas.

Binary search on tehokkaampi kuin linear search, mutta se vaatii, että lista on järjestetty. Jos lista ei ole järjestetty, binary search on yhtä hidas kuin linear search.

Contains-metodi on tehokkain tapa selvittää, onko jokin luku listassa, jos lista on toteutettu ArrayList-luokalla. Contains-metodi käyttää sisäisesti binäärihakua, joten se on yhtä tehokas kuin binary search, mutta se ei vaadi listan järjestämistä.

Nopein tapa selvittää, onko jokin luku listassa, on käyttää ArrayList-luokan sisältää-metodia. Jos lista on järjestetty, binary search on toinen tehokas vaihtoehto. Jos lista ei ole järjestetty, linear search on ainoa vaihtoehto.

Esimerkki ohjelmasta, joka käyttää kaikkia kolmea algoritmia:

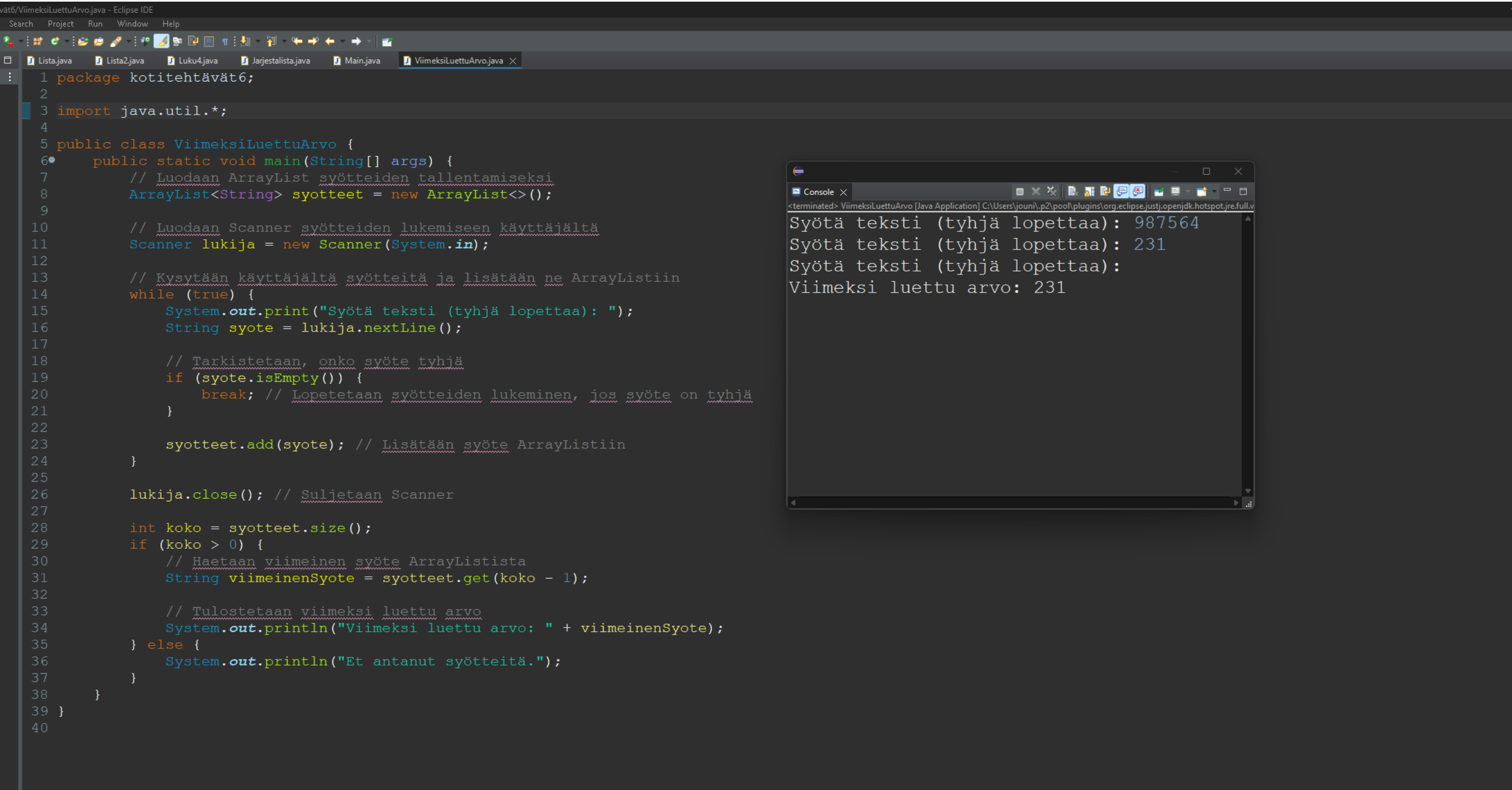


```
4
5 public class Main {
6
7     public static void main(String[] args) {
8         ArrayList<Integer> lista = new ArrayList<>();
9         lista.add(2);
10        lista.add(76);
11        lista.add(4);
12        lista.add(8);
13        lista.add(55);
14        lista.add(4);
15        lista.add(34);
16        lista.add(7);
17        lista.add(2);
18        lista.add(22);
19        lista.add(55);
20        lista.add(4);
21        lista.add(98);
22
23        int luku = 4;
24
25        // Linear search
26        boolean löytyy = false;
27        for (int i = 0; i < lista.size(); i++) {
28            if (lista.get(i) == luku) {
29                löytyy = true;
30                break;
31            }
32        }
33        System.out.println("Linear search: " + löytyy);
34
35        // Binary search
36        löytyy = false;
37        lista.sort(null);
38        int ala = 0;
39        int ylä = lista.size() - 1;
40        while (ala <= ylä) {
41            int keski = (ala + ylä) / 2;
42            if (lista.get(keski) == luku) {
43                löytyy = true;
44                break;
45            } else if (luku < lista.get(keski)) {
46                ylä = keski - 1;
47            } else {
48                ala = keski + 1;
49            }
50        }
51        System.out.println("Binary search: " + löytyy);
52
53        // Contains-metodi
54        löytyy = lista.contains(luku);
55        System.out.println("Contains-metodi: " + löytyy);
56    }
57 }
```

```
<terminated> Main [Java Application] C:\Users\jouni\AppData\Local\Temp\p2\pool\plugins\org.e
Linear search: true
Binary search: true
Contains-metodi: true
```

4. ArrayList listan käsittelyä (1p) a. Tulosta viimeksi luettu arvo

Tee ohjelma, joka kysyy käyttäjältä syötteitä ja lisää syötteet listalle. Syötteen lukeminen lopetetaan, kun käyttäjä syöttää tyhjän merkkijonon. Kun syötteiden lukeminen lopetetaan, ohjelma tulostaa viimeksi luetun arvon. Käytä tässä tehtävässä hyödyksi listan koon kertovaa metodia.



b. Tulosta listan suurin luku

Tee ohjelma, joka kysyy käyttäjältä positiivisia lukuja listalle. Syötteiden lisääminen lopetetaan, kun käyttäjä syöttää negatiivisen luvun. Lukujen lukemisen jälkeen ohjelma etsii listalta suurimman luvun ja tulostaa sen arvon sekä tuon suurimman luvun neliön.

Ohjelman pitäisi toimia seuraavasti:

Syötä lukuja:

72

2

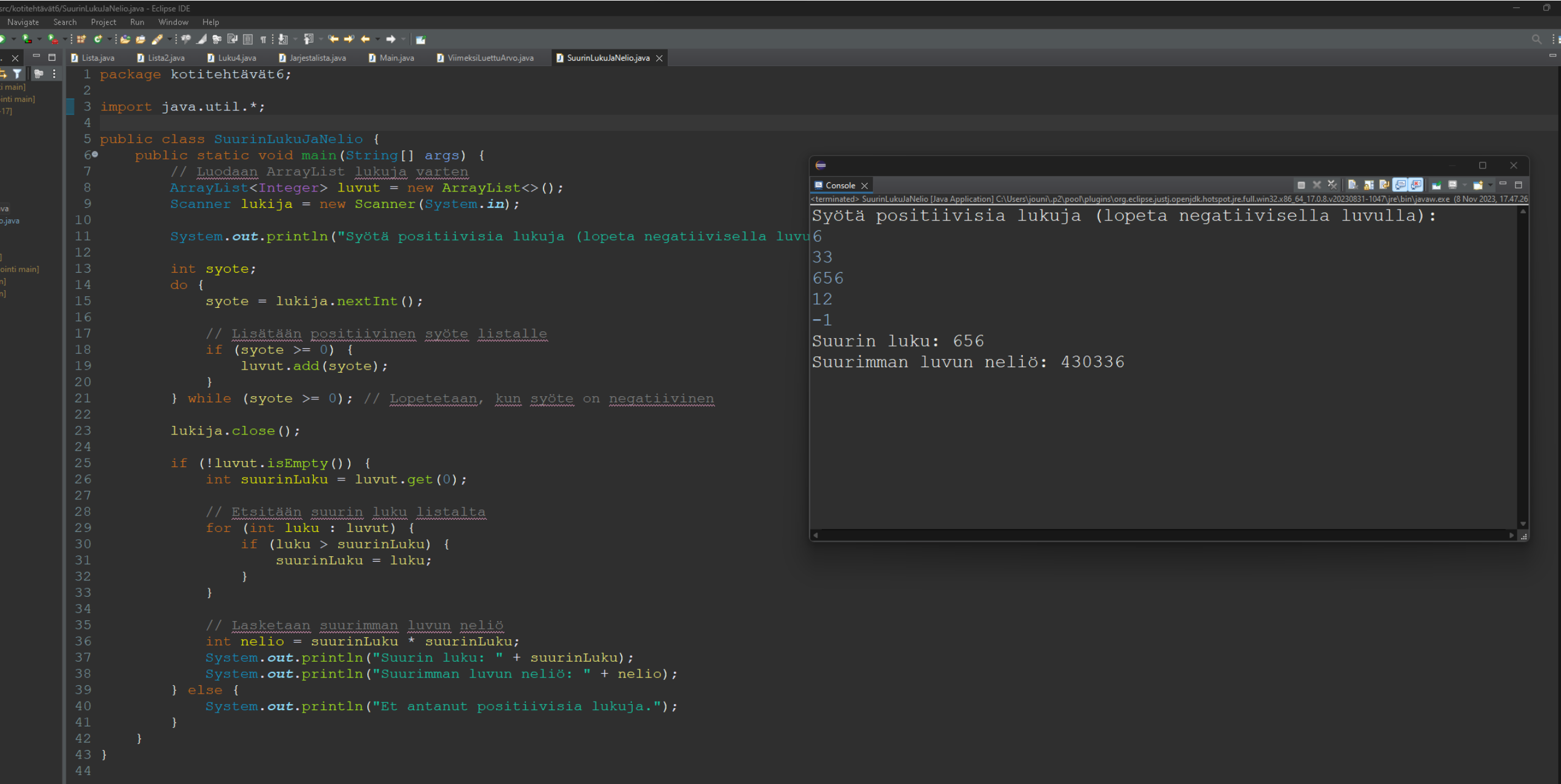
8

93

11

-2

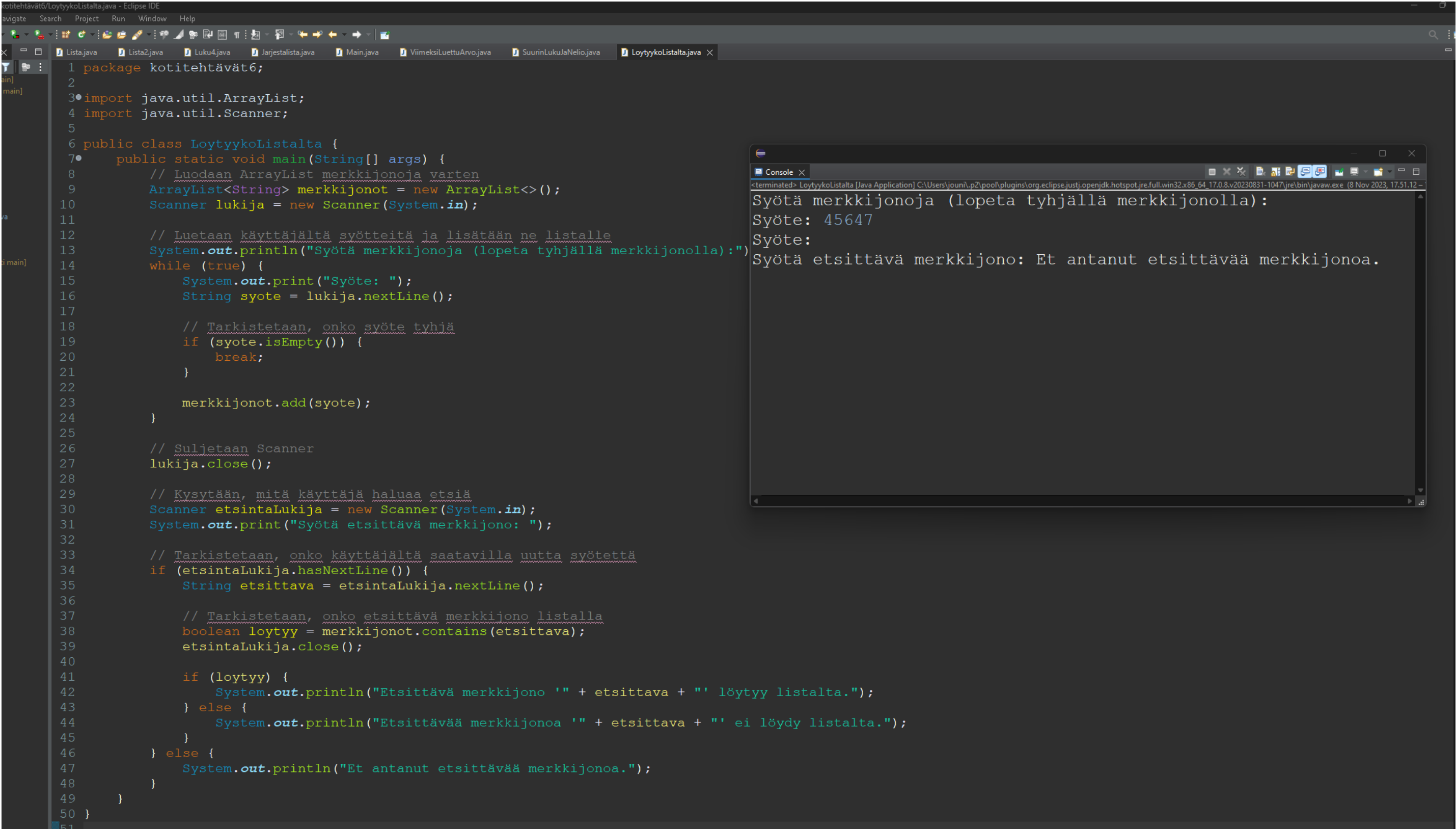
Listan suurin luku oli 93 ja sen neliö on 8649.





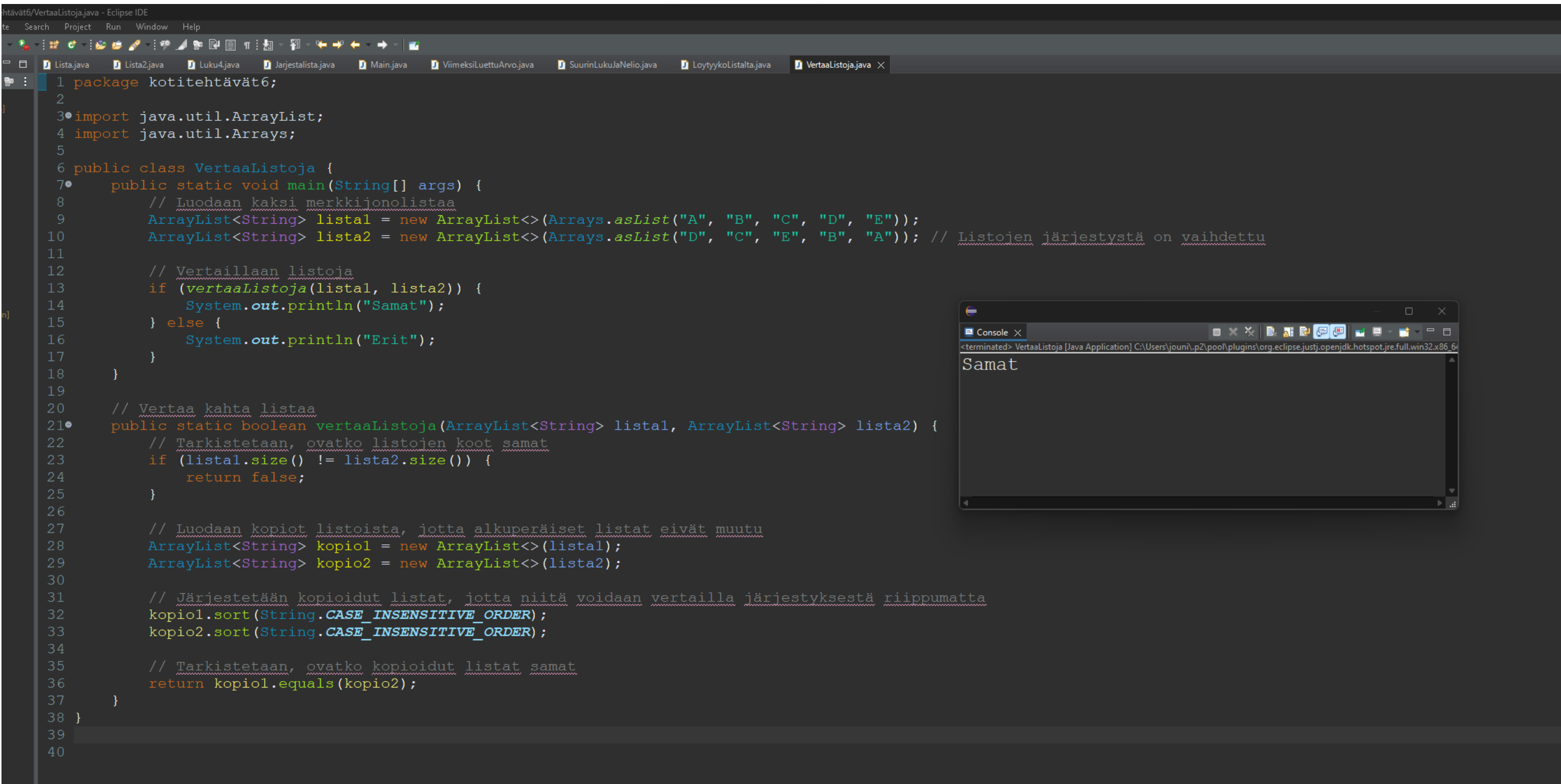
c. Löytyykö listalta

Tee ohjelma, joka lukee käyttäjältä syötteitä, kunnes tämä syöttää tyhjän merkkijonon. Kun merkkijono on valmis, ohjelma kysyy, mitä etsitään. Tämän jälkeen ohjelma kertoo, löytyykö etsittävä merkkijono listalta.



5. Vertaile kahta listaa (1p)

Tee ohjelma, joka sisältää kaksi merkkijonolistaa, jossa on vähintään viisi merkkijonoa. Ohjelma vertailee näitä listoja keskenään ja tulostaa “Erit”, jos listoissa on eri elementtejä ja “Samat”, jos niissä on samat elementit. Listan elementtien järjestyksellä ei ole väliä. Älä käytä sort metodia. Voit lopussa vielä vertailla ovatko listat täsmälleen samat.



## 6. Oracle Java Documentointi (2p)

Käytä Oraclen Java dokumentointia hyväksesi ja selitä, miten HashMap toimii. Käytä myös muuta materiaalia hyväksesi ja selitä, mitä hashillä tarkoitetaan ja miten sillä talletetaan tietoa tietokoneen muistiin. Miksi käyttäisit hashmapia, etkä esim. ArrayListia? Tee myös koodiesimerkit listan peruskäytöstä (lisääminen, poistaminen, etsiminen). Miten järjestät listan arvot, jos käyttössäsi on Hashmap tyyppinen lista?

<https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html>

HashMap on Java Collection Frameworkin osa, joka tarjoaa tehokkaan tavan tallentaa ja hakea avain-arvo -pareja. Se perustuu hajautustaulurakenteeseen (hash table), joka mahdollistaa nopean haku- ja tallennusoperaatiot. Alla selitän, miten HashMap toimii ja mitä hajautus (hashing) tarkoittaa.

### 1. Hajautus (Hashing):

- Hajautus tarkoittaa, että jokaiselle avaimelle lasketaan hajautusarvo (hash code). Hajautusarvo on yksilöllinen numero, joka liittyy avaimen tietoihin. Tämä hajautusarvoa käytetään tallettamaan ja hakemaan arvoja HashMapissa tehokkaasti.

### 2. Taulukon indeksointi:

- HashMap sisältää taulukon (bucket), jossa arvot tallennetaan hajautusarvon perusteella indeksoituihin paikkoihin.
- Hajautusarvoa käytetään indeksoimaan taulukkoa. Taulukon indeksit vastaavat hajautusarvoja, ja arvot tallennetaan vastaaviin indekseihin.

### 3. Tallennus:

- Kun halutaan tallentaa avain-arvo -pari HashMapiin, avaimen hajautusarvo lasketaan.
- Tämä hajautusarvo määrittää taulukon indeksin, johon arvo tallennetaan. Jos indeksi on jo varattu, käytetään tavallisesti linkitettyä lista -rakennetta (jota kutsutaan "bucketiksi") käsittämään useita arvoja samasta hajautusarvosta.

### 4. Haku:

- Kun halutaan hakea arvo avaimen perusteella, avaimen hajautusarvo lasketaan uudelleen.
- Tämä antaa taulukon indeksin, johon arvo voi olla tallennettuna. Jos useita arvoja on samassa "bucketissa", etsitään oikea arvo linkitetystä listasta.

### 5. Miksi käyttäisit HashMapia:

- HashMap on tehokas tietorakenne, kun tarvitaan nopeaa haku- ja tallennustoiminnallisuutta avain-arvo -pareille.
- HashMap mahdollistaa suorat haun avaimen perusteella, mikä tekee siitä erittäin nopean tiedon hakemiseen.

## 7. LinkedList Iteraattorin Tila (1p)

- a. Miten tarkistat, onko Iteraattorilla vielä elementtejä luettavana?

Voit tarkistaa, onko iterattorilla vielä elementtejä luettavana käyttämällä **hasNext()**-metodia. Tämä metodi palauttaa **true**, jos iterattorilla on seuraava elementti luettavana, ja **false**, jos ei ole.

Esimerkiksi:

```
Iterator<String> iterator = list.iterator();
while (iterator.hasNext()) {
    String element = iterator.next();
    // Käsitlele elementti
}
```

- b. Mitä etua on Iteraattorin käytöstä?

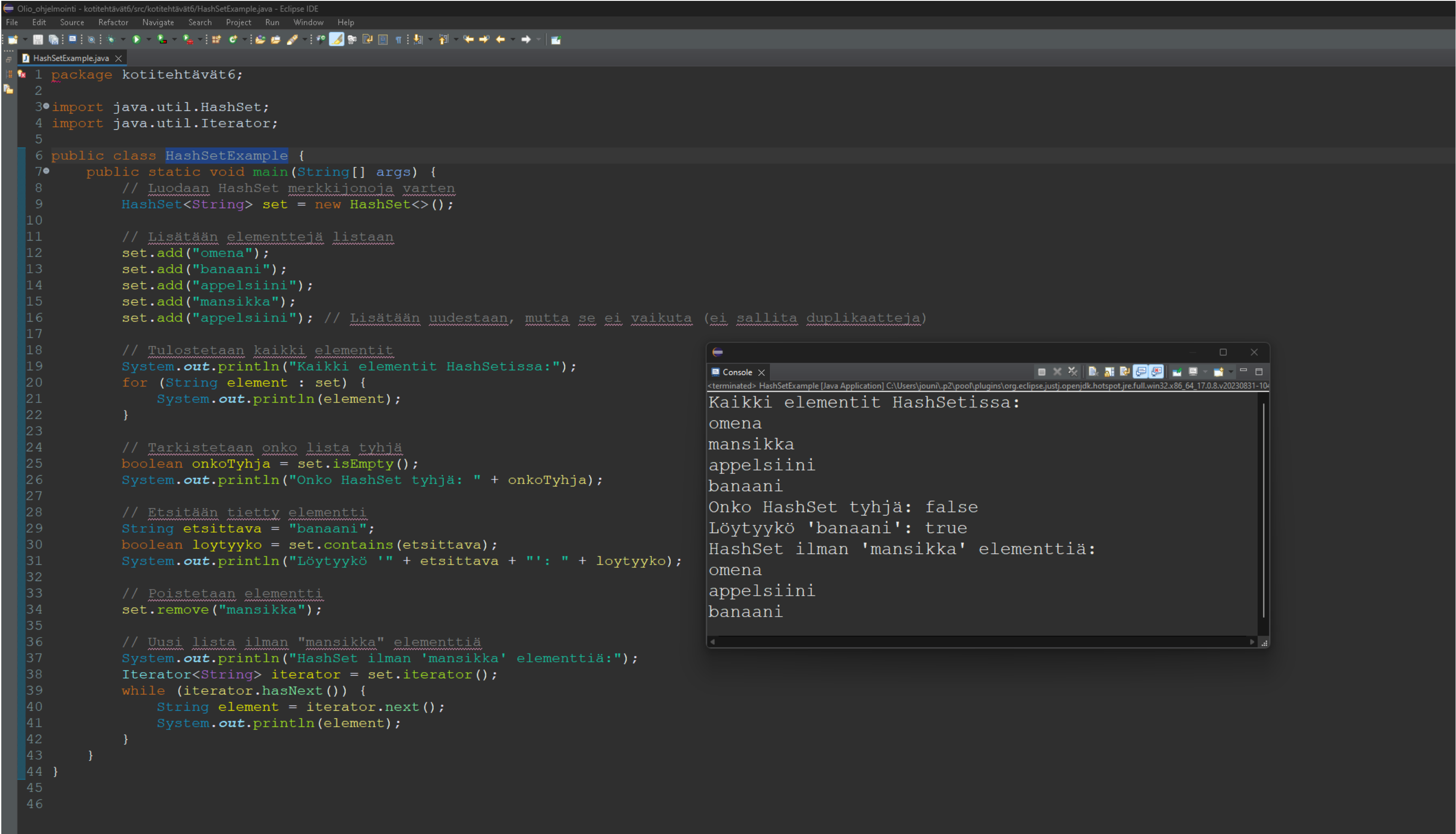
- teraattorit tarjoavat yksinkertaisen ja tehokkaan tavan käydä läpi kokoelmat (kuten ArrayList, LinkedList) elementti kerrallaan.
- Iteraattorit mahdollistavat elementtien poistamisen kokoelmasta turvallisesti käytön aikana.
- Ne mahdollistavat järjestelmällisen ja suunnitellun läpikäynnin kokoelman elementeistä.

- c. Mitä haittaa voi olla Iteraattorin käytöstä?

- Iteraattoreiden käyttö voi aiheuttaa lisää koodia ja monimutkaisuutta, koska vaaditaan lisäkoodia, kuten hasNext() ja next(), elementtien käsittelyyn.
- Iteraattorit ovat yleensä tehokkaita, mutta ne voivat olla hitaampia kuin suora indeksipohjainen pääsy, kuten ArrayListissa, erityisesti suurissa kokoelmissa.
- Iteraattorit ovat yksisuuntaisia, ja niitä ei voi käyttää taaksepäin tai hyppyyn tiettyjen elementtien yli, mikä voi olla haitta joissakin tapauksissa.
- Kokonaisuudessaan iterointi iteraattoreiden avulla on yleinen ja tehokas tapa käydä läpi kokoelmia, mutta sen käyttö vaatii tarkkuutta ja ymmärrystä niiden toiminnasta ja rajoituksista.

## 8. Koodaustehtävä HashSet (1p)

Kirjoita ohjelma, jossa luot HashSet tyyppisen listan, jossa elementteinä on merkkijonoja (esim. mausteita, autojen rekisterinumeroita, teelaatuja, värejä, yms.). Koodissa pitää lisätä elementtejä listaan, poistaa elementtejä listasta, tarkistaa onko lista tyhjä vai ei, käydä läpi listan jokainen elementti ja tulostaa se, sekä etsiä jokin tietty elementti.



9. **Selitä**, miten HashSet ja HashMap eroavat toisistaan, missä tilanteessa käyttäisit HashSetia ja missä taas HashMapia? (1p)

HashSet ja HashMap ovat kaksi erilaista tietorakennetta Javassa, ja niillä on erilaiset käyttötarkoitukset ja ominaisuudet:

HashSet:

HashSet on kokoelma, joka tallentaa yksilöllisiä alkioita ilman järjestystä.

Se perustuu hajautustauluun (hash table), joka tarjoaa tehokkaan tavan tallentaa ja hakea yksilöllisiä elementtejä.

HashSet ei salli duplikaatteja; jos yrität lisätä saman alkion useita kertoja, se korvataan aiemman saman alkion sijasta.

HashSet soveltuu tilanteisiin, joissa sinun on tärkeää pitää joukko yksilöllisiä elementtejä, mutta niiden järjestys ei ole merkityksellinen.

Esimerkkejä käyttötarkoituksista: Uniikit käyttäjät verkkosivustolla, sanakirjat ilman toistuvia sanoja.

HashMap:

HashMap on tietorakenne, joka tallentaa avain-arvo -pareja ja mahdollistaa nopean avaimen perusteella tapahtuvan haun ja lisäyksen.

Se perustuu myös hajautustauluun, mutta jokainen alkio koostuu avaimesta ja arvosta.

HashMapissa arvot voivat olla duplikaatteja, mutta avaimet ovat yksilöllisiä. Kun käytetään samaa avainta uudestaan, aiempi arvo korvataan uudella arvolla.

HashMap soveltuu tilanteisiin, joissa sinun on tärkeää yhdistää yksilölliset avaimet tietoihin ja hakea nopeasti arvoja avaimen perusteella.

Esimerkkejä käyttötarkoituksista: Tietokanta, jossa käyttäjien tunnukset (avaimet) liittyvät heidän tietoihinsa (arvot), varastointi taulukossa ja indeksipohjaiset hakut.

Yhteenvetona:

HashSet on tarkoitettu yksilöllisten elementtien tallentamiseen ilman järjestystä.

HashMap on tarkoitettu avain-arvo -pareihin perustuvaan nopeaan hakuun ja tallentamiseen.

Valitset HashSetin, kun haluat tallentaa yksilöllisiä elementtejä, ja HashMapin, kun haluat yhdistää avaimet ja arvot.