20. Tietorakenteita

Sisältö

- Abstrakti tietotyyppi ja tietorakenne.
  - Pino ja jono.
- Javan Collections-kirjasto.
- Taulukkolista (*ArrayList*<*E*>) ja sanakirja (*HashMap*<*E*>).
- Linkitetty lista.
  - *LinkedList*<*E*>-luokka.
- Collections-kirjastoon perustuva oma tietorakenneluokka.
  - OmaLista<*E*> ja oma listaoperaatio.

Abstrakti tietotyyppi ja tietorakenne

- Abstrakti tietotyyppi (abstract data type, ADT) on tyypin kuvaus, jossa ei oteta kantaa tyypin toteutukseen tai edes toteutuskieleen.
- ADT määrittelee vain tyypin operaatiot, jotka muodostavat tyypin liittymän ympäristöön.
  - Idea on hyvin samankaltainen kuin abstraktien luokkien ja rajapintojen.
- Tietorakenne (data structure) on abstraktin tietotyypin toteutus.
- Taulukko on tietorakenne, joka löytyy useimmista ohjelmointikielistä.

Abstrakti tietotyyppi ja tietorakenne

- Tietorakenteen toteutus perustuu usein toiseen tietorakenteeseen.
  - Esimerkiksi lista voidaan toteuttaa taulukon avulla.
- Olio-ohjelmointi tarjoaa mainion tavan toteuttaa abstrakteja tietotyyppejä.
  - Operaatioista tehdään metodeja ja tiedoista **private**-tyyppisiä attribuutteja.
- Abstraktin tietotyypin ja tietorakenteen sisältämiä tietoja sanotaan tietoalkioksi (data element) ja lyhyemmin alkioiksi (element).

Abstrakti tietotyyppi ja tietorakenne

- Tietorakenteen soveltuvuutta tiettyyn tehtävään voidaan pohtia arvioimalla rakenteen operaatioiden tehokkuutta ajan suhteen.
- Operaatioiden aikakompleksisuus annetaan suuruus-luokkina (“ordo”, iso *O*) suhteessa “syötteen” kokoon *n*, joka on tässä tietorakenteen alkioiden lukumäärä.
  - $O(1)$: suoritusaika on vakio tehtävän koosta riippumatta.
  - $O(\log n)$: suoritusaika on logaritminen.
  - $O(n)$: suoritusaika on lineaarinen.
  - $O(n^2)$: suoritusaika on neliöllinen, vielä laskettavissa pienillä *n*:n arvoilla.

Pino

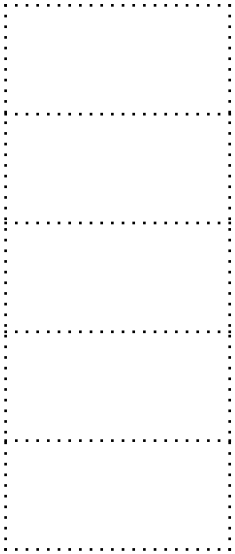
- Pino (stack) on abstrakti tietotyyppi, jossa
  - lisättävä alkio menee aina pinon päälle ja
  - poistettava alkio otetaan aina pinon päältä.
- Last-In-First-Out (LIFO) periaate.
- Lisäykset ja poistot alkio kerrallaan.
- Analogiana voi ajatella korttipakkaa pasianssissa:
  - Pakkaan voidaan lisätä kortti vain päälle, ei pakan väliin tai pohjalle.
  - Pakasta voidaan ottaa vain päällimmäinen kortti, ei pakan välistä tai pohjalta.

Pinon operaatioita

- **lisää**(Alkio a): lisää alkion *a* pinon päälle, mikäli pinoon mahtuu vielä uusi alkio.
- **poista**(): palauttaa ja poistaa pinon päällimmäisen alkion, mikäli pinossa on alkioita.
- **koko**(): alkioiden lukumäärä (≥ 0) pinossa.
- **onkoTyhjä**(): palauttaa totuusarvon **true**, jos pino on tyhjä ja totuusarvon **false**, jos pinossa ainakin yksi alkio.
- **ylin**(): palauttaa päällimmäisen alkion sitä poistamatta, mikäli pinossa on alkioita.
- Tietorakenne: esimerkiksi taulukko tai linkitetty lista.

Esimerkki

Luodaan tyhjä,
maksimissaan
5 alkiota
sisältävä pino.



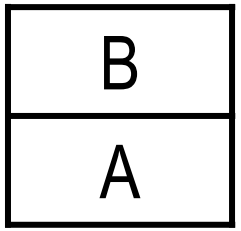
lisää(A)

koko() = 1

onkoTyhjä() = **false**

lisää(B)

koko() = 2

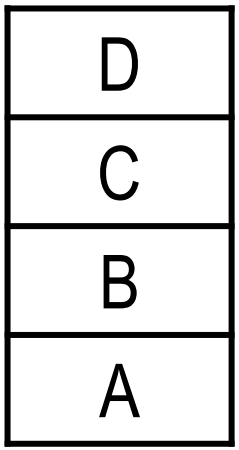


lisää(C)

koko() = 3

lisää(D)

koko() = 4

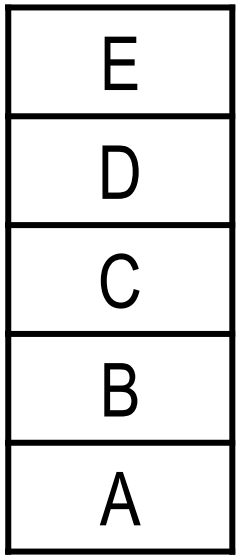


Esimerkki

lisää(E)

Pino täynnä:

koko() = 5

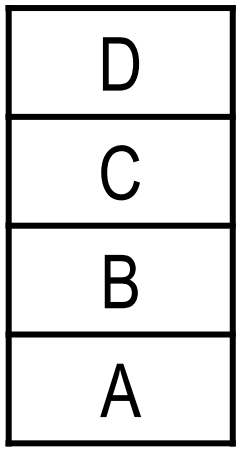


poista() = E

koko() = 4

ylin() = D

koko() = 4



poista() = D

koko() = 3

poista() = C

koko() = 2

poista() = B

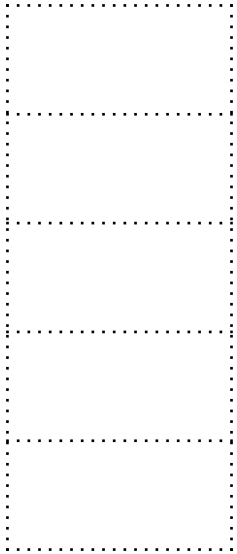
koko() = 1

poista() = A

Pino tyhjä:

koko() = 0

onkoTyhjä() = **true**



Jono

- Jono (queue) on abstrakti tietotyyppi, jossa
  - lisättävä alkio menee aina jonon loppuun ja
  - poistettava alkio otetaan aina jonon alusta.
- First-In-First-Out (FIFO) periaate.
- Lisäykset ja poistot alkio kerrallaan.
- Esimerkiksi ruokalan jono, jossa ensiksi tullutta asiakasta palvellaan ensiksi ja uudet asiakkaat sijoittuvat kiltisti jonon loppuun.

Jonon operaatioita

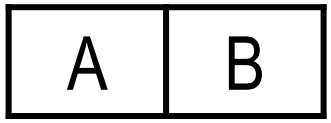
- **lisää**(Alkio a): lisää alkion *a* jonon loppuun, mikäli jonossa on vielä tilaa.
- **poista**(): palauttaa ja poistaa ensimmäisen alkion jonosta, mikäli jono ei ole tyhjä.
- **koko**(): alkioiden lukumäärä (≥ 0) jonossa.
- **onkoTyhjä**(): palauttaa totuusarvon **true**, jos jono on tyhjä ja totuusarvon **false**, jos jonossa ainakin yksi alkio.
- **keula**(): palauttaa jonon ensimmäisen alkion sitä poistamatta, mikäli jono ei ole tyhjä.
- Tietorakenne: esimerkiksi taulukko tai linkitetty lista.

Esimerkki

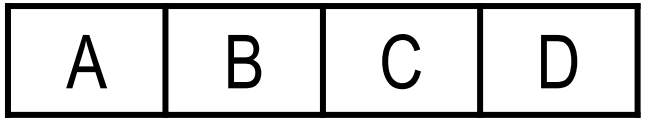
- Luodaan tyhjä, maksimissaan 5 alkiota sisältävä jono.

- **lisää**(A), **koko**() = 1, **onkoTyhjä**() = **false**

- **lisää**(B), **koko**() = 2

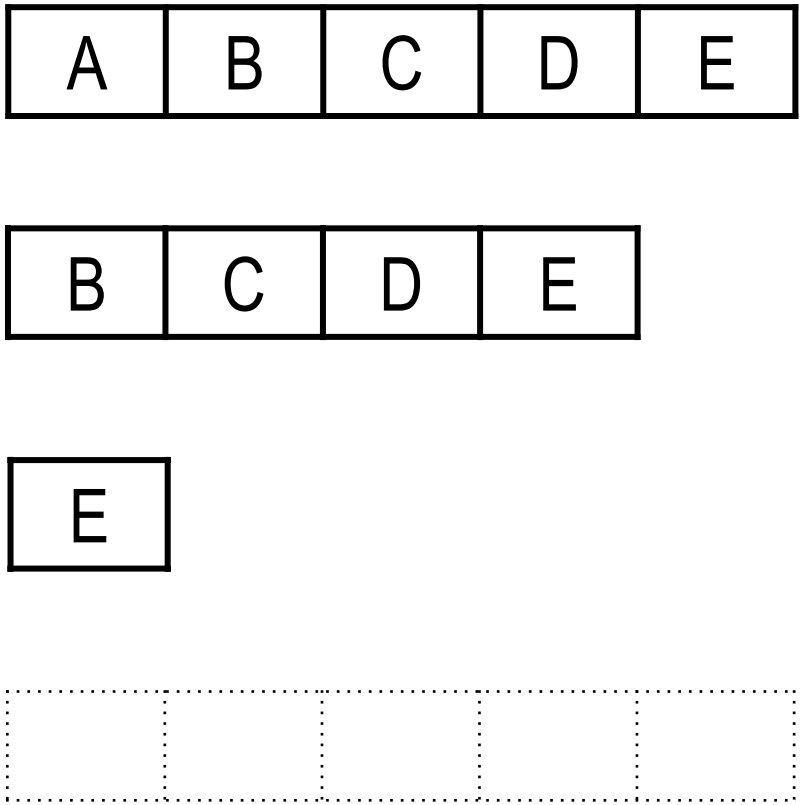


- **lisää**(C), **koko**() = 3, **lisää**(D), **koko**() = 4



Esimerkki

- **lisää**(E),
  Jono täynnä: **koko**() = 5
- **poista**() = A, **koko**() = 4,
  keula() = B, **koko**() = 4
- **poista**() = B, **koko**() = 3,
  poista() = C, **koko**() = 2,
  poista() = D, **koko**() = 1
- **poista**() = E,
  Jono tyhjä: **koko**() = 0,
  onkoTyhjä() = **true**



Collections-kirjasto

- Joukko kokoelmia eli abstrakteja tietotyyppejä mallintavia rajapintoja ja tietorakenteita toteuttavia luokkia.
- Kirjasto on osa Javan *java.util*-pakkausta.
  - Kirjaston luokkia käytettäessä tarvitaan **import**-lause.
- Suurin osa kirjaston luokista toteuttaa *Collection*-rajapinnan, joka määrittelee metodeja kokoelmille, jotka koostuvat yksittäisistä tietoalkioista.
- Kirjaston avain–arvo-parien kokoelmia mallintavat *Map*-rajapinta, rajapinnan jälkeläiset ja rajapinnat toteuttavat luokat.

Collections-kirjasto

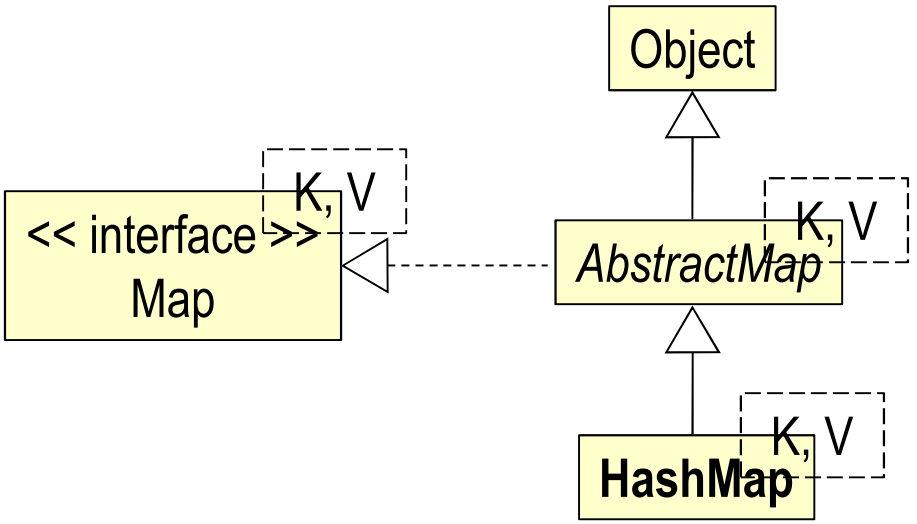
- Kirjaston luokat ja rajapinnat ovat geneerisiä.
- *Collection*-rajapinnan toteuttavat luokat voidaan käydä läpi for-each-rakenteella.
- *Collections*-luokkaan on koottu kokoelmien käsittelyyn soveltuvia luokkametodeja.
- Alkeistyyppiset arvot kääritään automaattisesti kokoelmaan lisättäessä. Kääre avataan automaattisesti alkeistyypiksi, jos mahdollista.
- Taulukot eivät ole suoraan osa Collections-kirjastoa, mutta kirjastossa on *Arrays*-luokka taulukoiden käsittelyyn.

Collections-kirjasto

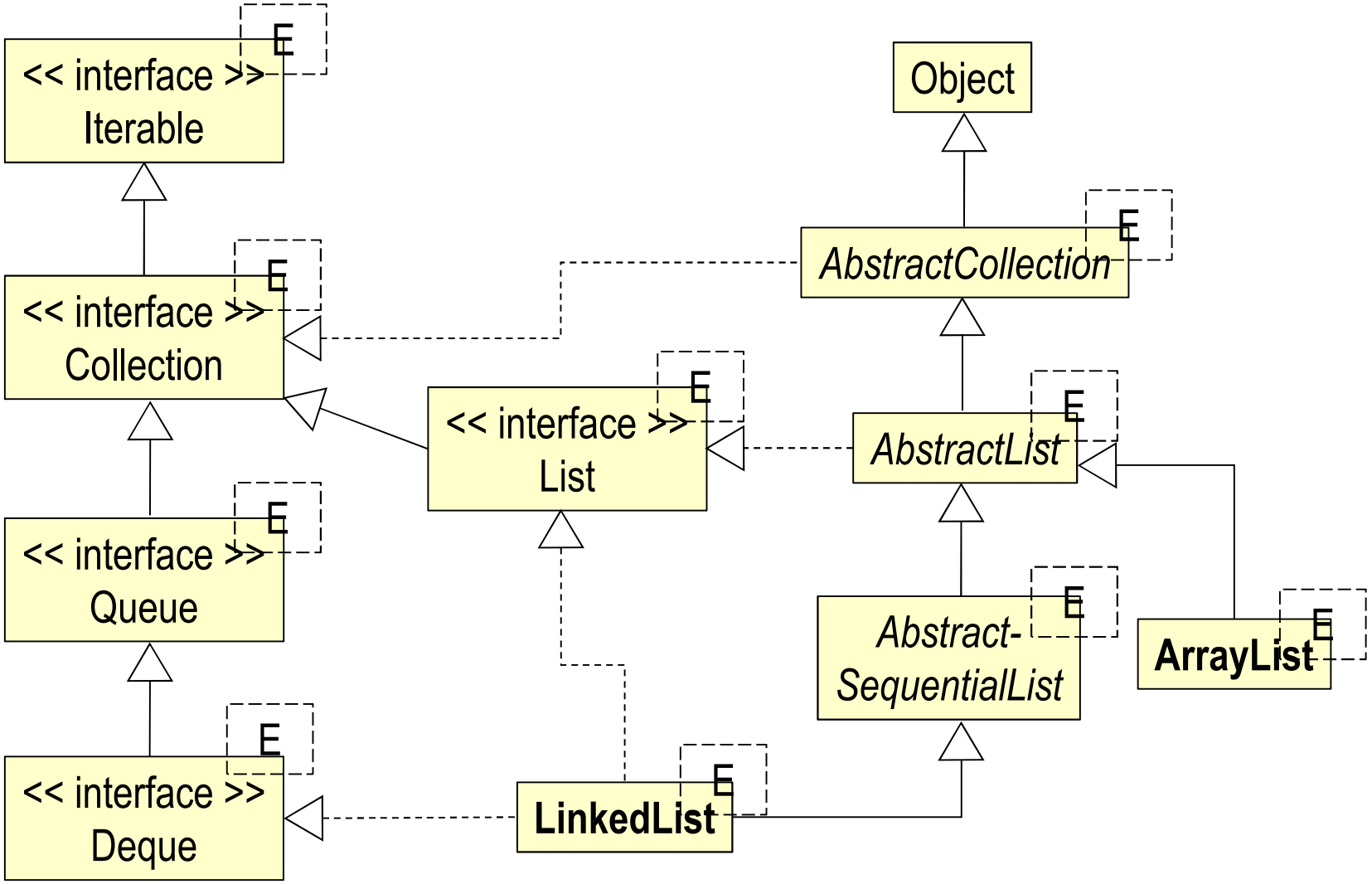
- Joitakin *Collection*-rajapinnan metodeja.
  - *Add*(*E*): alkion lisäys kokoelmaan.
  - *Contains*(*Object*): alkion haku kokoelmasta *equals*-metodia hyödyntäen.
  - *IsEmpty*(): tutkii onko kokoelma tyhjä.
  - *Remove*(*Object*): valinnaisesti toteutettava operaatio, joka poistaa kokoelman alkion *equals*-metodia hyödyntäen.
  - *Size*(): kokoelman alkioiden lukumäärä.
  - *Stream*(): luo ja palauttaa tietovirran, johon voidaan soveltaa aggregaatio-operaatioita.
  - *ToArray*(): luo ja palauttaa taulukon, jossa on viitteet kokoelman alkioihin.

Collections-kirjasto

- Tarkastellaan kolmea keskeistä kokoelma-luokkaa.
  - *ArrayList*, muuttuvamittainen taulukko.
  - *HashMap*, avain–arvo-parien kokoelma.
  - *LinkedList*, linkitetty lista.



Collections-kirjasto



Taulukkolista (ArrayList<E>)

- Taulukkolista on viitetyyppisten alkioiden taulukko, jonka koko voi kasvaa tai pienetä.
  - Korvaa *Vector*-luokan peräkkäisissä sovelluksissa.
- Lisätilaa varataan automaattisesti aina tarvittaessa.
- Taulukkolistan kapasiteetti (capacity) tarkoittaa taulukon alkioiden lukumäärää, kun taas koko (size) on taulukoon säilöttyjen tietoalkioiden (viitteiden) lukumäärä.
- Geneerinen luokka, jossa kiinnitetään alkioiden tyyppi.
- // Luodaan taulukkolista, johon voidaan tallentaa vain
  // Integer-tyyppisiä viitteitä.
  ArrayList<Integer> luvut = **new** ArrayList<Integer>();

Taulukkolista (ArrayList<E>)

- Taulukkolistan alkioihin voidaan viitata taulukon tapaan nollasta alkavalla indeksiarvolla.
- Taulukkolista on toteutettu taulukon avulla.
  - Alkion arvon lukemisen (*get*) ja alkion arvon muuttamisen (*set*), kuten myös alkion lisääminen taulukon loppuun (*add*(*E*)), aikakompleksisuus on odotetusti *O*(1).
  - Alkion lisäys tiettyyn paikaan (*add*(**int**, *E*)), poistaminen (remove) ja hakeminen (*contains*, *indexOf*) ovat hitaampia (*O*(*n*)).

Sanakirja (HashMap<K,V>)

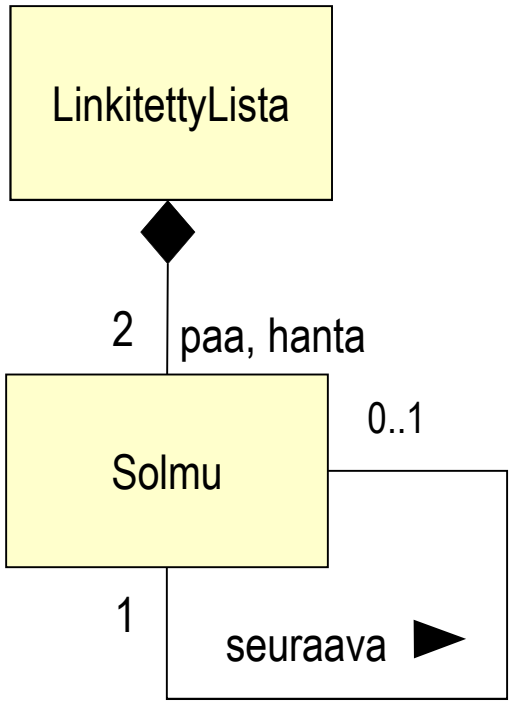
- Hyödyllinen tietorakenne avain–arvo-parien säilömiseen.
  - Sanakirjaan voidaan tallentaa esimerkiksi pelinumeron ja pelaajan nimen muodostamia pareja, joissa numero on avain ja nimi tieto tai päinvastoin.
- *HashMap*-luokka on toteutettu hajautustaulun (hash table) avulla.
  - **public class** HashMap<K,V> ... missä *K* on avaimen ja *V* tiedon tyyppiparametri.
  - Avainten tulisi olla muuttumattomia olioita.
  - Ei takaa lisäysjärjestyksen säilymistä.
  - Lisäyksen (*put*(*K*,*V*)) ja haun (*get*(*Object*)) kompleksisuus *O*(1).
  - Hyödyntää avainten *hashCode*-metodia.

Linkitetty lista

- Linkitetty lista (linked list) on tietorakenne, joka koostuu toisiinsa viittaavista solmuista (node).
- Solmu viittaa tietoalkioon.
- Listan alkioilla on järjestys ja alkioihin voidaan yleensä myös viitata nollasta alkavan indeksin avulla.
- Operaatiot määritellään lähteestä riippuen hieman eri tavoin ja toteutukseenkin on useita lähestymistapoja.

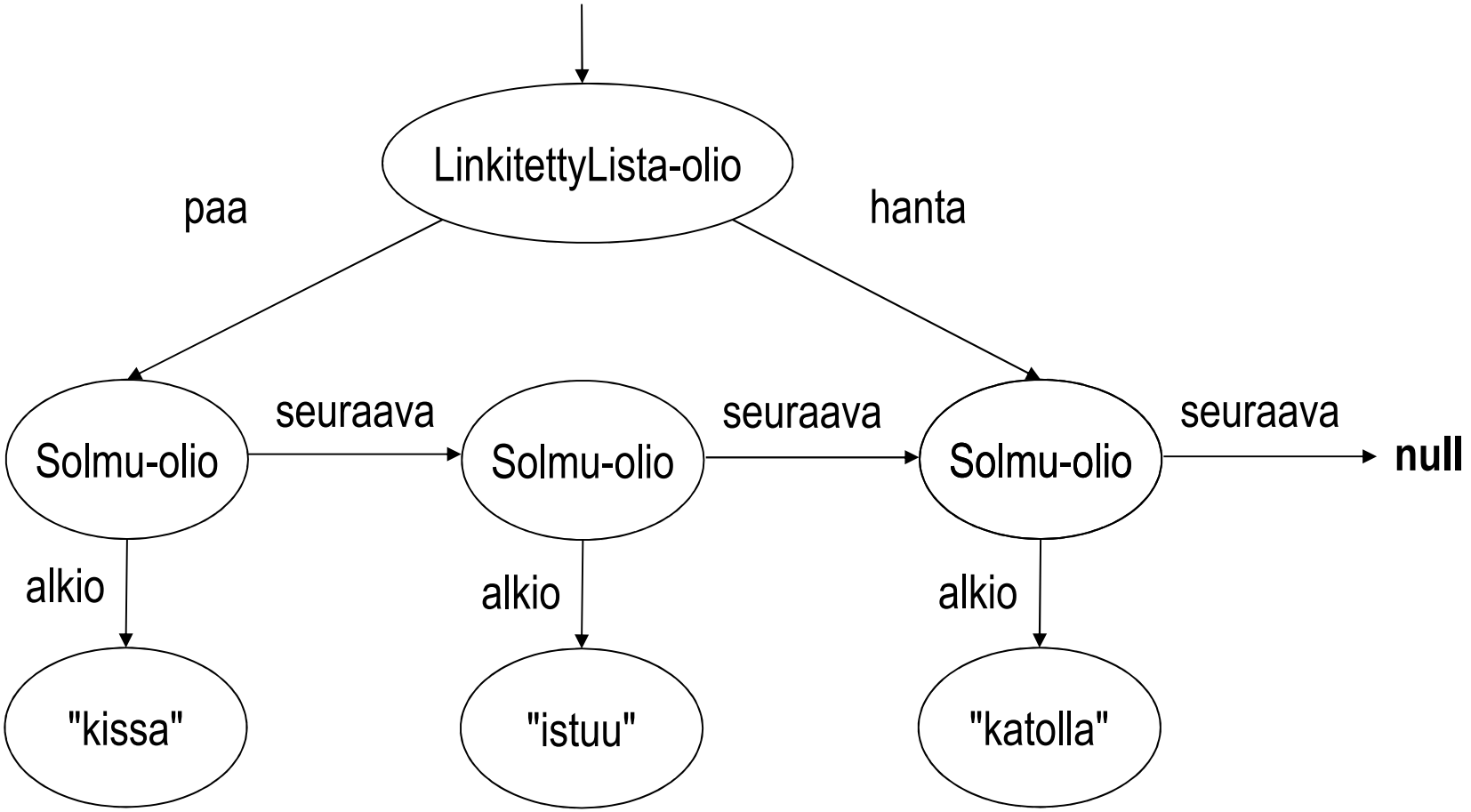
Yhteen suuntaan linkitetty lista

- Linkitetyn listan yksinkertaisin muoto.
  - Solmu viittaa aina seuraavaan solmuun.
  - Viimeinen viite liittyy **null**-arvoon.
- Pää (head) on listan ensimmäinen ja häntä (tail) on listan viimeinen solmu.
  - Pää ja häntä ovat listan osia: Listan toteuttavalla luokalla on *Solmu*-tyyppiset attribuutit päälle ja hännälle.
- Listalla voidaan kulkea siirtymällä attribuutin avulla ensimmäiseen solmuun ja seuraamalla solmuviitteitä, kunnes ollaan oikeassa paikassa.



Yhteen suuntaan linkitetty lista

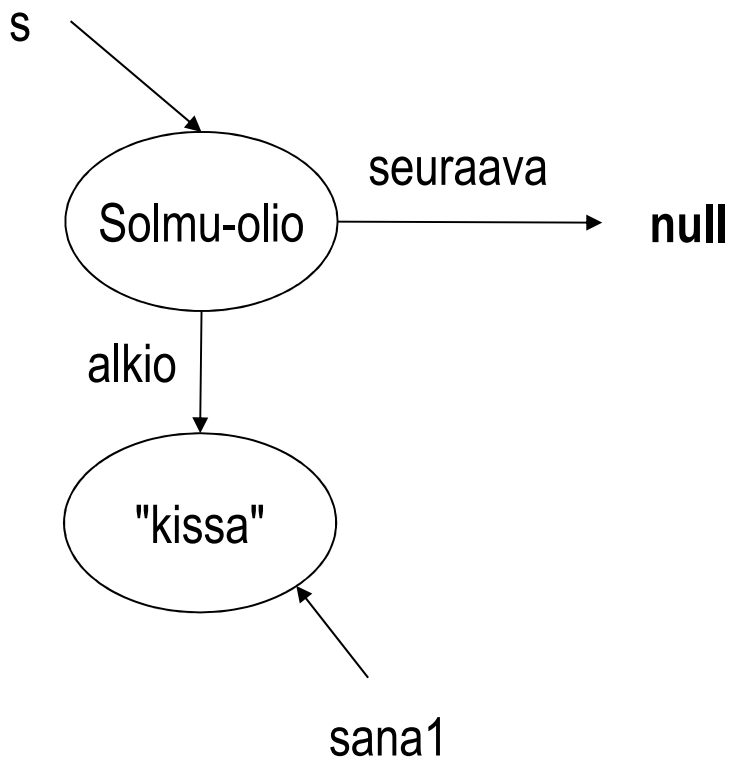
- Kolmisolmuinen lista, jonka tietoalkiot ovat merkkijonoja.



Yhteen suuntaan linkitetty lista

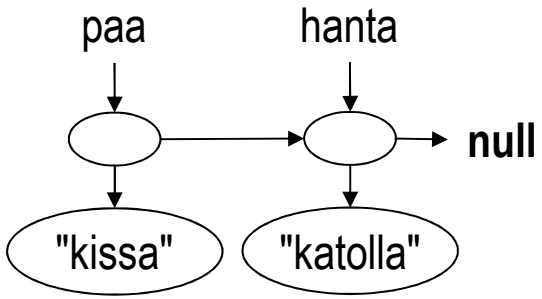
- **public class** Solmu {
  private Object alkio;
  private Solmu seuraava;
  public Solmu(Object uusi) {
    alkio = uusi;
    seuraava = **null**;
  }
  public Solmu seuraava() {
    return seuraava;
  }
  public void seuraava(Solmu s) {
    seuraava = s;
  } ...

- String sana1 = **new** String("kissa");
  Solmu s = **new** Solmu(sana1);

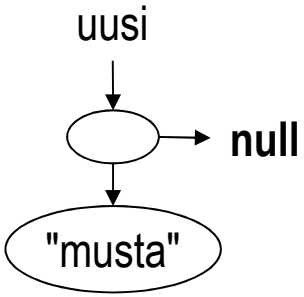


Listan alkuun lisääminen

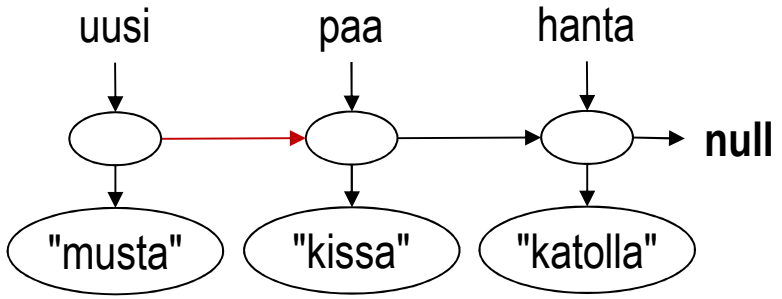
- Lista ennen lisäystä.



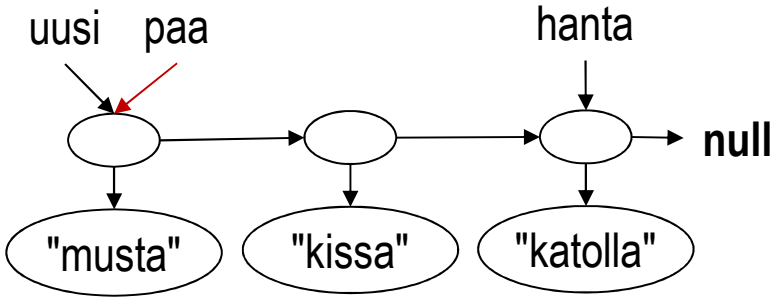
- // Luodaan uusi solmu.
  Solmu uusi = **new** Solmu(alkio);



- // Uusi solmu osoittamaan päähän.
  uusi.seuraava(paa);



- // Uusi solmu listan pääksi.
  paa = uusi;



LinkedList<E>-luokka

- Geneerinen luokka, joka toteuttaa kahteen suuntaan linkitetyn listan.
  - Solmuista viite sekä solmua edeltävään että sitä seuraavaan alkioon, jolloin lista on tehokkaampi, koska listaa voidaan kulkea sekä eteen että taakse päin.
- Alkioihin voidaan viitata nollasta alkavalla indeksiarvolla.
- Tietyn alkion arvon lukeminen ja muuttaminen on hitaampaa ($O(n)$) kuin taulukkolistassa.
- Listan alun ja lopun käsittely on yhtä nopeaa ($O(1)$).
- Lista käyttää aina täsmälleen tarvittavan määrän muistia toisin kuin taulukkolista.

Oma tietorakenneluokka

- Collections-kirjaston sisältöä voi hyödyntää oman tietorakenneluokan toteuttamisessa.
- Periytymis- ja koostesuhde ovat mahdollisia.
- Periytymisen avulla saadaan käyttöön esivanhempien metodit.
- Koostetta käyttämällä tietorakenne voidaan kätkeä ja luokan liittymä määritellä täysin itse.
- Jos tavoitteena on tehdä pelkästään uusi operaatio, on periytymissuhde luontevampi, koska metodin toteutukseen ja luokan käyttöön tarvittavat metodit ovat suoraan saatavilla.

OmaLista<E> ja oma listaoperaatio

- Yliluokka on otettava käyttöön **import**-lauseella.
- Periytymisessä on huomioitava geneerinen tyyppimääre.
- // Javan LinkedList-luokasta peritty oma listaluokka.

```
import java.util.LinkedList;
public class OmaLista<E> extends LinkedList<E> {
// Hakee listalta annetun arvon toisen esiintymän paikkaa.
public int toisenEsiintymanIndeksi(Object o) {
  // Haetaan ensimmäisen esiintymän indeksiarvo
  // yliluokasta perityllä metodilla.
  int ekaInd = indexOf(o);
  ...
```