JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Ohjelmointi 2 – Luento 5

Kokoelmat

Luennon sisältö

- Kokoelmat Javassa
- Listarakenteet
- Hakurakenteet
- Joukkorakenteet
- Pino- ja jonorakenteet
- Rekursio

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Kokoelmat

Kokoelma

- Tähän mennessä samantyyppisiä arvoja koostettu taulukkoihin ja listoihin
- Todellisuudessa tietorakenteiden käyttöön liittyy usein lisäehtoja, esimerkiksi:
  - Korttipelit: kortti monesti saa ottaa vain pakan päältä
  - Sisu: jokaisella opiskelijalla saa olla vain yksi arvosana samasta toteutuksesta
  - Sosiaaliset alustat: käyttäjä ei voi seurata toista käyttäjää kahdesti
- *Kokoelma* (engl. collection) on olio, joka *koostaa* samantyyppisiä arvoja yhteen kokonaisuuteen ja *määrittää tavat* käsitellä arvoja

Kokoelmat Javassa

Javassa kokoelmat käytetään yleensä seuraavasti:

(1) (2)

```
Collection<String> kokoelma = new ArrayList<>();
```

(1) Kokoelmarajapinnat

- Määrittävät metodeja, joiden kokoelmalla tulisi olla
- Määrittävät yleisesti, miten kokoelman tulisi toimia
- Voivat myös määrittää yleisiä ehtoja kokoelman alkioille (esim. järjestys, indeksointi)

(2) Kokoelmaluokat

- Toteuttavat jonkin kokoelman käytännössä
- Sisältävät muisti- ja nopeusehtoja
- Voivat sisältää lisärajoitteita (esim. ”ei null-alkioita”, ”ei voi lisätä uusia arvoja”)

Kokoelmat Javassa

Collection<T>-rajapinta: Kaikille kokoelmille ominaiset toiminnot

```
Collection<String> paikkoja = new ArrayList<>();

// 1. Alkion lisääminen
paikkoja.add("La Palma");
paikkoja.add("Madeira");

// 2. Alkion poistaminen
paikkoja.remove("Madeira");

// 3. Alkioiden lukumäärän tarkistaminen
IO.println("Paikkoja: " + paikkoja.size());

// 4. Alkion olemassaolon tarkistaminen
IO.println("Onko Lanzarote kokoelmassa: " + paikkoja.contains("Lanzarote"));

// 5. Kaikkien alkioiden läpikäynti
for (String paikka : paikkoja) {
    IO.println(paikka);
}
```

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Yleiset kokoelmatyypit

Preface: aikavaativuus

- Samojen kokoelmatyyppien toteutukset eroavat ensisijaisesti operaatioita toteuttavien algoritmien nopeuden ja vaadittavan muistin perusteella
- Algoritmeissa käytetään usein ns. Iso-O–merkintää kuvaamaan, kuinka monta vaihetta algoritmissa on
- Kokoelmien tapauksessa aikavaativuus kuvataan usein alkioiden lukumäärän suhteen

Merkintä	Yksinkertaistettu merkitys
O(1)	Algoritmi ei riipu alkioiden lukumäärästä, eli ns. vakioaika
O(n)	Algoritmi käy jokaisen alkion läpi (vrt. yksi for-silmukka)
O(n^2)	Jokaisen alkion kohdalla käydään jokainen alkio läpi (vrt. sisäkkäinen for-silmukka)
O(log(n))	Algoritmin jokaisessa vaiheessa käsitellään puolet alkioista kerrallaan (ns. puolitusmenetelmä)

Yksinkertaistetusti vaativuudet voidaan asettaa paremmuusjärjestykseen: O(1), O(log(n)), O(n), O(n^2)
Aikavaativuutta ja sen formaalia merkitystä käydään tarkemmin kurssilla ITKA201 Algoritmit 1

Yleiset kokoelmatyypit
Listat

- *Lista* (engl. list) on kokoelma, jossa alkioilla on järjestys
- Alkioiden järjestystä määrää numeerinen *indeksi*
- Alkioiden lisääminen ja poistaminen säilyttää alkioiden suhteellisen järjestyksen
- Alkioita voi lisätä ja poistaa indeksiä käyttäen
- Sama alkio voi esiintyä listassa monta kertaa
- Javassa listaa vastaa rajapinta List<T>

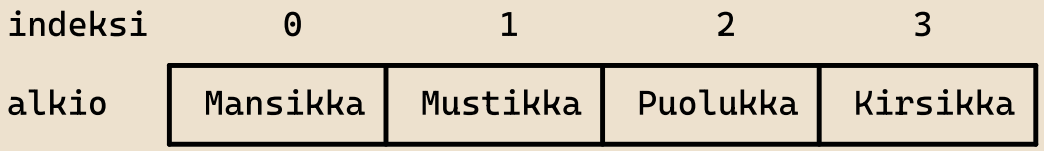
Yleiset kokoelmatyypit
Listat

- Kokeillaan toteuttaa yksinkertainen dynaaminen lista määritelmän perusteella
- Perusajatus: tehdään tarpeeksi suuri taulukko ja pidetään kirjaa, mihin indeksiin asti taulukko on ”täytetty”

```
class Lista<T> {
    T[] taulu;
    int taytettyLkm;

    Lista() {
        //noinspection unchecked: tyyppimuunnos (T[]) ei tee mitään ajon aikana
        taulu = (T[]) new Object[100];
        taytettyLkm = 0;
    }

    void set(int indeksi, T alkio) {}
    T get(int indeksi) {}
    void add(T alkio) {}
    void remove(T alkio) {}
    int size() {}
}
```

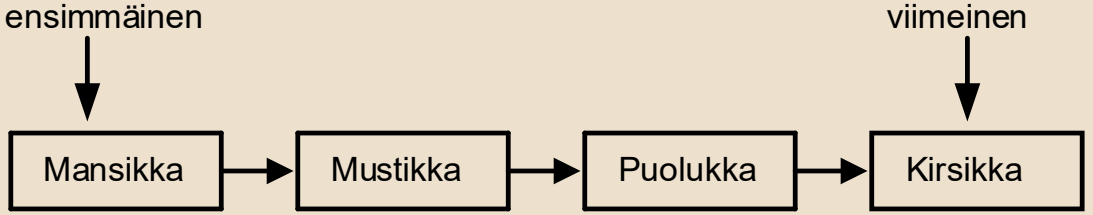
Yleiset kokoelmatyypit
Listat

ArrayList<T>



- Alkiot tallennettu yhteen taulukkoon
- Alkion hakeminen indeksin perusteella on *aina* O(1)
- Uuden alkion lisääminen on yleensä O(1), mutta pahimmillaan saattaa vaatia taulukon kopiointia, mikä on O(n)
- Alkion poistaminen vaatii alkioiden siirtämistä poistetun alkion tilalle, mikä on pahimmillaan O(n)

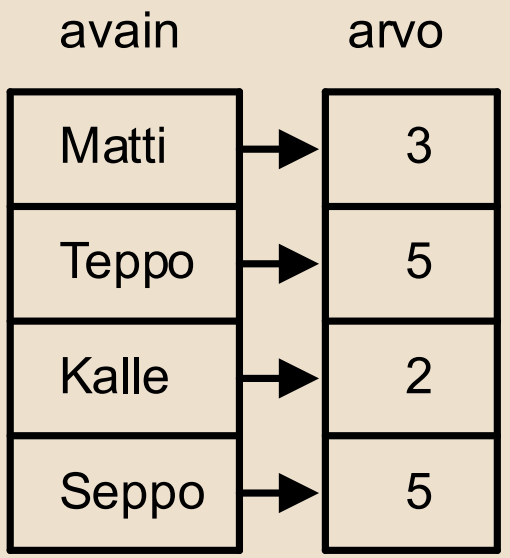
LinkedList<T>



- Listalla on viite ensimmäiseen ja viimeiseen alkioon; jokaisella alkiolla on arvo ja viite seuraavaan alkioon
- Alkion hakeminen indeksin perusteella on O(n)
- Uuden alkion lisääminen loppuun tai alkuun on *aina* O(1), mutta lisääminen tiettyyn indeksiin on pahimmillaan O(n)
- Alkion poistaminen alusta tai lopusta on aina O(1), mutta poistaminen tietystä indeksistä on O(n)

Yleiset kokoelmatyypit
Hakurakenteet

- *Hakurakenne* (engl. map) sisältää *avaimia* (engl. key) ja niitä vastaavia *arvoja* (engl. value)
- Indeksin sijaan arvoa vastaa jokin avainarvo, jonka voi päättää itse
- Alkioita voi lisätä ja poistaa avaimen perusteella
- Sama avain saa esiintyä vain kerran; arvot voivat esiintyä usean kerran
- Javassa hakurakenteelle on useampi rajapinta
  - `Map<K, V>` - Yleinen hakurakenne
  - `SequencedMap<K, V>` - Hakurakenne, jossa alkioilla on kiinteä järjestys
  - `SortedMap<K, V>` - Hakurakenne, jossa alkiot ovat aina järjestetty avaimen arvon perusteella
- Huom: `Map<K, V>` ei toteuta `Collection`-rajapintaa



Yleiset kokoelmatyypit
Hakurakenteet

```
Map<String, Integer> arvosanoja = new HashMap<>();
// Arvon tallentaminen avaimen perusteella
arvosanoja.put("Matti", 3);
arvosanoja.put("Teppo", 5);
arvosanoja.put("Kalle", 2);
arvosanoja.put("Seppo", 5);

// Arvon päivittäminen, ei lisää uutta arvoa
arvosanoja.put("Teppo", 3);
// Sama, mutta selkeämpi nimi
arvosanoja.replace("Teppo", 3);

// Haetaan Kallea avastaava arvosana
arvosanoja.get("Kalle");

// Arvon poistaminen avaimen perusteella
arvosanoja.remove("Seppo");

for (Map.Entry<String, Integer> nimiJaArvosana : arvosanoja.entrySet()) {
    IO.println(nimiJaArvosana.getKey() + " => " + nimiJaArvosana.getValue());
}
```

Yleiset kokoelmatyypit
Hakurakenteet

Hakurakenteille on useampi valmis toteutus, kolme ”hyvä tietää” toteutusta:

1. HashMap<K, V>
   - Perustuu avain-arvoparien tallentamiseen taulukkoon käyttäen avaimen hashCode-metodia
   - Arvon tallentaminen ja haku yleensä O(1), mutta pahimmillaan O(n) jos sisäinen taulu pitää laajentaa tai jos useammalla avaimella on sama hashCode
   - Ei anna takeita arvojen järjestyksestä
2. LinkedHashMap<K, V>
   - Perustuu avain-arvoparien tallentamiseen linkitettyyn listaan
   - Toteuttaa SequencedMap-rajapinnan; alkiot ovat aina lisäysjärjestyksessä
   - Hieman suurempi muistin käyttö

Yleiset kokoelmatyypit
Hakurakenteet

Hakurakenteille on useampi valmis toteutus, kolme ”hyvä tietää” toteutusta:

3. TreeMap<K, V>
   - Toteuttaa SortedMap-rajapinnan; alkiot ovat aina järjestetty avaimen mukaan
   - Ei käytä avaimen hashCode-metodia, mutta edellyttää, että avaimille on määritelty järjestys
   - Haku, lisäys ja poisto *aina* O(log(n)), eli hitaampi kuin O(1), mutta nopeampi kuin O(n)
   - Hyvä ”välimaasto”, kun halutaan, että alkiot ovat aina järjestyksessä

Yleiset kokoelmatyypit
Joukkorakenteet

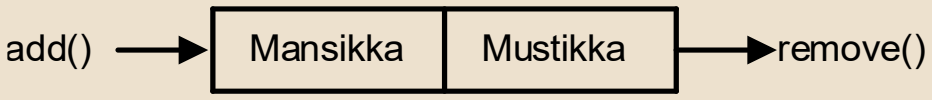
- *Joukko* (engl. set) on kokoelma, jossa sama alkio voi esiintyä vain yhden kerran
- Hyödyllinen toistuvien arvojen poistossa ja arvon olemassaolon tarkistuksessa
- Javan rajapinnat vastaavat hakurakenteita
  - Set<T> - yleinen rajapinta joukoille
  - SequencedSet<T> - joukko, jossa alkioilla on kiinteä järjestys
  - SortedSet<T> - joukko, jossa alkiot ovat aina järjestetty
- Vastaavasti toteutukset pääosin vastaavat hakurakenteita
  - HashSet<T> - taulukkoon perustuva; lisäys, haku ja poisto O(1), pahimmillaan O(n)
  - LinkedHashSet<T> - linkitettyyn listaan perustuva, toteuttaa SequencedSet
  - TreeSet<T> - alkiot aina järjestetty (SortedSet); lisäys, haku ja poisto aina O(log(n))

Yleiset kokoelmatyypit
Joukkorakenteet

```
Set<String> paikkoja = new HashSet<>();
paikkoja.add("La Palma");
paikkoja.add("Gran Canaria");
paikkoja.add("El Hierro");
paikkoja.add("Fuerteventura");

paikkoja.remove("Lanzarote");

IO.println("Onko La Palma mukana: " + paikkoja.contains("La Palma"));
IO.println("Onko Madeira mukana: " + paikkoja.contains("Madeira"));

for (String paikka : paikkoja) {
    IO.println(paikka);
}
```

Yleiset kokoelmatyypit
Jonorakenteet

- *Jono* (engl. queue) on kokoelma, joka on tarkoitettu käsittelyjonojen pitämiseen
- Alkoita voidaan lisätä jonon *häntään* ja ottaa pois jonon *päästä*
- Javassa kaksi oleellista rajapintaa jonoille
  - Queue<T> - Yksisuuntainen jono
  - Deque<T> - Kaksipäinen jono (engl. double ended queue), jossa alkioita voidaan lisätä ja poistaa vapaasti alusta ja lopusta



FIFO-jono (first-in-first-out)

LIFO-jono (last-in-first-out) eli pino

Yleiset kokoelmatyypit
Jonorakenteet

```
// FIFO-jono
Deque<Tehtava> tehtavat = new LinkedList<>();
// Lisää uusi tehtävä jonon loppuun
tehtavat.add(new Tehtava("Herää"));
tehtavat.add(new Tehtava("Pese hampaat"));
tehtavat.add(new Tehtava("Tee aamupala"));
tehtavat.add(new Tehtava("Syö aamupala"));

while (!tehtavat.isEmpty()) {
    // Ota ensimmäinen jonossa oleva tehtävä
    Tehtava tehtavaNyt = tehtavat.remove();
    tehtavaNyt.tee();
}
```

Yleiset kokoelmatyypit
Jonorakenteet

```
// LIFO-jono eli ns. pino
Deque<String> kortit = new LinkedList<>();
// "Työnnä" eli lisää uusi kortti pinoon päällimmäiseksi
kortit.push("Sininen 4");
kortit.push("Keltainen 4");
kortit.push("Keltaien 5");
kortit.push("Keltainen 1");

while (!kortit.isEmpty()) {
    // Ota päällimmäinen kortti pinolta
    String paallimmainen = kortit.pop();
    IO.println(paallimmainen);
}
```

Yleiset kokoelmatyypit
Jonorakenteet

- Huomaa, että LinkedList<T> toteuttaa Queue<T> sekä Deque<T>
- Lisäksi:
  - ArrayDeque<T> - taulukkoon perustuva kaksisuuntainen jono, muistin kannalta voi olla tehokkaampi kuin LinkedList
  - PriorityQueue<T> - jono, joka siirtää pienimmän alkion automaattisesti jonon päähän; antaa algoritmisesti nopean ratkaisun kysymykseen ”mikä on N. suurin/pienin alkio”

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Hieman rekursiosta

Rekursio

Yksinkertaistetusti rekursio on asian määrittäminen itsensä avulla

Kertoma n! = n * (n - 1)!

1! = 1

5! = 5 * 4!

= 5 * (4 * 3!)

= 5 * (4 * (3 * 2!))

= 5 * (4 * (3 * (2 * 1!)))

= 5 * 4 * 3 * 2 * 1 = 120

Listan summa summa(l) = l[0] + summa(l[1..])

summa([]) = 0

summa([4, 2, 5]) = 4 + summa([2, 5])

= 4 + (2 + summa([5]))

= 4 + (2 + (5 + summa([])))

= 4 + (2 + (5 + 0))

= 4 + 2 + 5 = 11

Rekursio
Rekursio ja pino

```
int kertoma(int n) {
    if (n == 1) return 1;
    return n * kertoma(n - 1);
}
```

Rekursio
Rekursio ja pino

```
class KertomaKehys {
    int n;
    int tulosNyt;

    KertomaKehys(int n, int tulosNyt) {
        this.n = n;
        this.tulosNyt = tulosNyt;
    }
}
```

```
int kertoma(int n) {
    Deque<KertomaKehys> pino = new LinkedList<>();
    pino.push(new KertomaKehys(n, 1));

    while (!pino.isEmpty()) {
        KertomaKehys nyt = pino.pop();
        if (nyt.n == 1) {
            return nyt.tulosNyt;
        }
        pino.push(new KertomaKehys(nyt.n - 1,
                                   nyt.n  * nyt.tulosNyt));
    }

    return -1;
}
```

Rekursio
Rekursio ja pino

```
int kertoma(int n) {
    KertomaKehys tila = new KertomaKehys(n, 1);

    while (true) {
        if (tila.n == 1) {
            return tila.tulosNyt;
        }
        tila.tulosNyt = tila.n * tila.tulosNyt;
        tila.n--;
    }
}
```

Mitä seuraavaksi

- Tee osan 5 harjoitustehtävät