

# Demo 10 / 17.11

## Tehtävät

1. **Ville:** Tee Villellä (ks. <https://trac.cc.jyu.fi/projects/ohj1/wiki/ville>) tehtävät: Järjestysalgoritmit. Muuta tekemiesi tehtävien määrä suhteeksi välille [0,1] ja laita pistemääräksi 0.2 tarkkuudella pyöristetty luku. Esim. jos teit 120/195 tehtävää = 0.615, niin laita 0.6 pistettä. Jos teit 140/195 = 0.72, laita 0.8 tehtävää).

2. **Kokonaislukulistat:** Tee lotto-ohjelma, joka arpoo 7 numeroa ja 3 varanumeroa 39:stä. Vinkki:

```
public static void main(String[] args) {
    List<Integer> pallot = new ArrayList<Integer>();
    // täytä pallot numeroilla 1-39, esim: pallot.add(4);
    // sotke pallot (ks. Collections)
    // tulosta 7 ekaa "palloa"
    // tulosta 3 seuraavaa palloa
}
```

3. **Silmukat:** Erilaisten 7 numeroa sisältävien lottorivien määrä saadaan binomikertoimen kaavasta:

Katso: <http://fi.wikipedia.org/wiki/Kombinaatio>

$$\begin{array}{ccc} \text{kaava 1} & \text{kaava 2} & \text{kaava 3} \\ \binom{39}{7} = \frac{39!}{7! * (39-7)!} & = & \frac{33*34*35*36*37*38*39}{1*2*3*4*5*6*7} = 15380937 \end{array}$$

Tulos voidaan laskea käyttämällä `long` -tyyppisiä lukuja. Tee funktio

```
long nYliK(int n,int k)
```

jonka kutsulla `nYliK(39, 7)` saat mainitun tuloksen. Tulosta ei voi laskea keskimmäisestä kaavasta 2, koska `39!` ylittäisi reilusti pitkienkin (`long`) kokonaislukujen lukualueen.

Vinkki: Tässä tehtävässä ei tarvita listoja tms., pelkästään kerto- ja jakolaskuja sekä silmukoita. Vastaus on nätimpi jos avuksi kirjoittaa yhden pienen aliohjelman.

4. **Listat:** Edelliselläkin kaavalla (3) `long` -lukualue voi ylittyä jaettavassa, jos käsitellään paljon isompia lukujoukkoja kuin 39 (nyt raja on 515). Tällöin yksi mahdollisuus on ennen laskuja sieventää mahdollisimman paljon jaettavaa ja jakajaa. Sievennä ensin käsin mahdollisimman paljon kaavan 3 laskua. Mieti mitä teit. [Maara.java](#) -tiedostossa on valmiita aliohjelmien esittelyjä ja testejä avuksi siihen, miten laskua voisi tehdä, jos jaettava ja jakaja pidettäisiin kumpikin listana kertoimista. Toteuta metodit `tayta`, `kerro`, ja `poista`. Metodien pitää toimia kuten kommentteissa olevissa testeissä on kuvattu. Koska supistaminen/sieventäminen ei yksittäisten alkioiden kohdalta ole yksikäsitteistä, testeihin

saa halutessaan tehdä seuraavanlaisia muutoksia:

```
Testien jotka ovat tyyliin:
jaettava.toString() === "[2, 3, 2, 5, 3, 7]";
tilalle saa supistus/sievennys metodeissa laittaa testin,
joka tutkii, että osamäärän tekijä on oleellisesti sama, eli
kerro(jaettava) === 2*3*2*5*3*7;
```

Pyri ajamaan ComTest -testit ja varmistu sillä tavalla, että aliohjelmasi toimivat oikein.

Vinkki: ConcurrentModificationException tarkoittaa että iterointi (Olistan alkioden läpikäynti) silmukalla for (int alkio:lista) on johtanut ristiriitaan sen kanssa, että listaa on muutettu kesken iteroinnin (läpikäynnin). Tällöin pitää valita toisenlainen tapa listan läpikäyntiin. Kannattaa piirtää kuva itselle siitä, mitä alkion poistaminen taulukkomaisesta listasta aiheuttaa.

### 5. Listat: Toteuta edelliseen vielä metodi

```
int supista(List<Integer> jaettava, List<Integer> jakaja, int supistaja)
```

totuttamaan kommentteissa kuvatut vaatimukset. Näin voit laskea maksimissaan

```
nYliK(1733,7) = 9202167919706100768L
```

### 6. Poikkeukset: Kirjoita Double.parseDouble ja poikkeuksia käyttäen funktio muutaJono(String s, double oletus) jota voidaan käyttää seuraavasti:

```
public static void main(String args[]) {
    double d1 = muutaJono("12.3",0.0);
    double d2 = muutaJono("12.3e",0.0);
    System.out.printf("%5.2f %5.2f",d1,d2); // 12.30 0.00
}
```

### B1. Tutki miten Javassa toimii BigInteger -luokka ja tee nYliK(int n, int k) sen avulla. Tee siis tehtävää 3 vastaava toteutus erityyppisellä kokonaisluvulla.

## GURU-tehtävät

### G1. Mieti miten nYliK(int n, int k) ( kirjoitetaan jatkossa C(n, k) ) voidaan lausua rekursiivisesti, eli:

```
C(n+1,k) = C(n,k) * f(n,k) // mietä millainen on f(n,k):n lauseke.
C(k,k) = 1
```

Kirjoita sitten tämän avulla iteratiivinen (silmukkaan, ei rekursioon) perustuva ratkaisu, jossa kertolaskut eivät ”helposti” ylitä lukualueen rajaa. Tällä algoritmilla päästään tuloksiin, jotka ovat  $< \text{Long.MAX\_VALUE}/n$ . Kaavan 3 versiolla päästiin tuloksiin  $< \text{Long.MAX\_VALUE}/(k!)$ .

### G2. Toteuta ja testaa Maara.java -luokkaan metodit:

```
public static int sievenna(List<Integer> jaettava, List<Integer> jakaja)
public static int supista(List<Integer> jaettava, List<Integer> jakaja)
```

