Demo 10 / 16.11

Tehtävät

V1. Tee Ville-tehtävät: 8.1, 8.2, 8.3, 9.7, 9.9

1. **M: 19. Rekursio:** Katso mallia rekursiivisesta kolmion piirtämisestä. Valitse neljästä vaihtoehdosta se, jonka parhaiten ymmärrät (käytännössä Kolmiot.java -malli riittää kunhan ei piirretä alle pikselin kokoisia oksia). Tee mallia muokaten ohjelma, joka piirtää puu.png -kuvan kaltaisen puun rekursiosta:

```
/**
 * Puun yksi oksa ja kolme muuta rekursiivisesti
 * @param x oksan alkupiste
 * @param y oksan alkupiste
 * @param d oksan suunta radiaaneina
 * @param l oksan pituus
 */
oksa( x, y, d, l)
    viiva pisteestä (x,y) pisteeseen (x+l*cos(d),y+l*sin(d))
    jos l < 2 lopeta
    oksa(x+l*cos(d),y+l*sin(d),d+0.6,l*0.6);
    oksa(x+l*cos(d),y+l*sin(d),d-0.6,l*0.6);
    oksa(x+l*cos(d),y+l*sin(d),d,l*0.3);
```

Ikkuna kannattaa aluksi skaalat niin, että origo on vasemmalla alhaalla. Hyvä alkukulma piirtämiseen on vaikkapa `pi/2.` ja pituus vajaa puolet ikkunan korkeudesta.

Kokeile vaihdella suuntaa ja oksan pituutta muuttavia vakioita ja katso miten ne vaikuttavat kuvaan.

2. **M: 20. Dynaamiset tietorakenteet:** 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. **M: 15. Toistorakenteet:** Erilaisten 7 numeroa sisältävien lottorivien määrä saadaan binomikertoimen kaavasta:

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

kaava 1 kaava 2 kaava 3

$$\binom{39}{} = \frac{39!}{} = \frac{33*34*35*36*37*38*39}{} = 15380937$$

```
7       7! * (39-7)!          1*2*3*4*5*6*7
```

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.

4-5. **M: 20. Dynaamiset tietorakenteet, 22. Tiedostot:** Ota selaimeen nettisivu:

http://www1.nordea.fi/s/merita/kurssit/val-tilival_eur.asp

ja katso sen lähdekoodia (Firefoxissa `Ctrl-U`). Tee pääohjelma ja tarvittavat aliohjelmat tyyliin:

```
lista = lueNetistaListaan(osoite);
valuuttalista = etsiVainValuutat(lista); // ottaa <pre> </pre> välin
while ( true) {
   valuutta = kysy käyttäjältä valuutan nimi
   if ( valuutta == "" ) break;
   tulostaValuutta(valuuttalista,valuutta); // etsii ja tulostaa
}
```

6. **M: 23. 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.

B2. **M: 20. Dynaamiset tietorakenteet:** Jopa 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 kommenteissa 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 (listan alkioiden 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.

B3. **M: 20. Dynaamiset tietorakenteet:** Toteuta edelliseen vielä metodi

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

totuttamaan kommenteissa kuvatut vaatimukset. Näin voit laskea maksimissaan

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

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)   // mieti 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 < `Long.MAX_VALUE/n`. Kaavan 3 versiolla päästiin tuloksiin< `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)
```