

## Demo 7 / 1.11

Tämän kerran aiheena on 2-ulotteiset taulukot ja keskipisteet pistejoukossa. Tähän liittyy kuva [pistejoukonkeskipisteet.png](#), jossa on arvottu tasoon joukko pisteitä ja laskettu pistejoukon painopiste (punainen piste), pistejoukon miidi (keskiarvoa lähin joukon alkio, vihreä ympyrä) sekä koordinaateittain laskettu miidi (sininen ympyrä). Voit tehdä nyt tehtävät 1-3 samaan luokkaan, koska ne liittyvät samaan tehtävään. Jos haluat tehdä tehtäviä eri tiedostoon esimerkiksi sen mukaan, että toiset piirtävät ja toiset vain laskevat (sekin järkevää), niin voit kutsua toisessa tiedostossa (vaikka `Laskut.java` olevaa metodia `miidi` toisesta tiedostosta seuraavasti:

```
double ym = Laskut.miidi(pisteet[1]);
```

Toinen tapa olisi laittaa alkuun staattinen `import`:

```
import static demo6.Lskut.miidi; // pitää olla paketissa demo6
...
double ym = miidi(pisteet[1]);
```

### Tehtävät

V1. Tee [Ville](#)-tehtävät: 5.5, 5.6, 5.8, 9.5, 9.6

T1. Jos tarkistat vähintään kahden funktion toiminnan automaattisella testillä ([ComTest](#) ja/tai [JUnit](#), ks. [Wikistä OhjOpetuksenTyopaja](#)), saat merkitä yhden lisäpisteen.

1. **M: 14. Taulukot:** Esitetään pisteet (eli lukuparit) 2-ulotteisessa taulukossa niin, että 1. rivissä (indeksi 0) on kaikki x-arvot ja 2. rivissä (indeksi 1) kaikki y-arvot. Tällöin `i:n` pisteen koordinaatit olisivat:

```
x = pisteet[0][i];
y = pisteet[1][i];
```

Piirrä itsellesi paperille kuva miltä taulukko näyttäisi (siis 2 riviä, paljon sarakkeita).

Tämä ei ole paras mahdollinen tapa esittää lukupareja, mutta tässä vaiheessa elämää kuitenkin käyttökelpoinen. Ongelmana on se, että tuollaisen taulukon jokaisen rivin ei tarvitse olla välttämättä yhtä pitkiä (saat vaatia ne yhtä pitkiksi omassa ratkaisussasi). Ota pohjaksi [LuvutGraafisesti.java](#) (muista että täällä oli `x=i` ja `y=luvut[i]`, eli vähän pitää muuttaa) ja tee tarvittavat aliohjelmat, jotta seuraava koodi piirtää kuvan [pistejoukonkeskipisteet.png](#) mustat ympyrät:

```
public static void main(String[] args) {
    int pisteita = 100;
    double min = 0;
    double max = 10;
    double r = (max-min)/100;

    double pisteet[][] = new double[2][];
    pisteet[0] = arvoLuvut(pisteita, min, max);
```

```

    pisteet[1] = arvoLuvut(pisteita, min, max);
    Window ikkuna = new Window(600,400);
    ikkuna.scale(min-1,min-1,max+1,max+1);
    ikkuna.add(new Axis(100,100,0));
    piirraKuva(ikkuna,pisteet,r);
    ikkuna.showWindow();
}

```

Vinkki: Demo6:ssa tehty suuri osa tarvittavista aliohjelmista: [Keskiluku.java](#).

2. **M: 15. Toistorakenteet, 14. Taulukot:** Kirjoita `lahimmanPaikka(pisteet, x, y)`, joka etsii missä kohti `pisteet` -taulukkoa on pistettä  $(x, y)$  lähin piste ja palauttaa tämän indeksin (ennätystehtailussa nyt siis ylläpidetään voittajan sijasta voittajan paikkaa, eli indeksiä). Muista että kahden 2D-pisteen etäisyys on tehty demossa 6. Voit testata vaikka aineistolla:

```

double[][] luvut = {
    {1,2,1,4 },
    {1,1,2,3 }
};
int i1 = lahimmanPaikka(luvut,0,0); // 0;
int i2 = lahimmanPaikka(luvut,10,10); // 3;
int i3 = lahimmanPaikka(luvut,2.1,2.1); // 1;

```

3. Nyt sitten yhdistä kaikki edellä oleva tietämys niin, että lisäät tehtävän 1 pääohjelmaan loppuun vielä lauseet:

```

double xk = keskiarvo(pisteet[0]); // Painopiste, x-keskiarvo
double yk = keskiarvo(pisteet[1]); // y-keskiarvo

int i = lahimmanPaikka(pisteet,xk,yk); // 2D-miidin paikka
double xlahin = pisteet[0][i];
double ylahin = pisteet[1][i];

ikkuna.add(new Marker(xk,yk,r*1.4)).setColor(Color.RED);
ikkuna.add(new Marker(xlahin,ylahin,r*1.2)).setColor(Color.GREEN);

double xm = miidi(pisteet[0]); // Miidi x-pisteistä
double ym = miidi(pisteet[1]); // Miidi y-pisteistä
ikkuna.add(new Marker(xm,ym,r2)).setColor(Color.CYAN);

```

4. **M: 14.4 Moniulotteiset taulukot:** Piirrä tekstitiedostoon (vaikka `ConText`-editorilla) kuva 3-ulotteisesta taulukosta mukaellen Ohj2:n luentomonisteen [kuvaa](#). Piirrä samaan tiedostoon myös kuva 2-ulotteisesta merkkijonotaulukosta mukaellen monisteen 2-ulotteisen taulukon ja [args-taulukon kuvaa](#). Vinkki: [14. luentovideon](#) lopussa (ajassa 1:23:00) on näytetty miten `ConTextia` voi käyttää.
5. **M: 14.4 Moniulotteiset taulukot:** Tee funktioaliohjelma, joka summaa yhteen kaksi matriisia vastinalkioittain. Funktio siis joutuu ensin luomaan oikeankokoisen tulosmatriisin. Mitä on syytä olettaa parametrimatriiseilta? Dokumentoi kommentteihin oletuksesi. Sitten funktio laskee tulosmatriisiin komponenteittain summan. Esimerkiksi:

```

public static void main(String[] args) {
    double mat1[][] = {{1,2,3},{2,2,2},{4,2,3}};
}

```

```

double mat2[][] = {{9,2,8},{1,2,5},{3,19,-3}};

double mat3[][] = summaa(mat1,mat2);
tulosta(mat3);
}

```

tulostaisi (tulosta on tehty luento-esimerkistä [Matriisit.java](#) double -matriisille muokaten, muista että %d on kokonaislukuja varten ja %f reaalityyppiä):

```

10,00  4,00 11,00
 3,00  4,00  7,00
 7,00 21,00  0,00

```

6. **M: 15. Toistorakenteet:** Tee funktioaliohjelma `laskeKirjaimet(jono, kirjain)`, joka laskee merkkijonossa olevien valittujen kirjainten lukumäärän. Testaa pääohjelmalla jossa on kutsuja tyyliin (keksi lisää testejä):

```

System.out.printf("%d\n", laskeKirjaimet("kissa", 's'));
System.out.printf("%d\n", laskeKirjaimet("kissa", 'k'));

```

7. Katso esimerkistä: [Bittikuvat](#) miten Graphicsilla piirretään kuva. Ota kuvaksi vaikka: [vesa.jpg](#). Kuva talletetaan Eclipseissä samaan paikkaan missä projekti tehdään (eli src-hakemiston yläpuolelle). Tee ensin ohjelma mikä piirtää kuvan sellaisenaan. Sitten muuta sitä niin, että kuva piirretään harmaasävykuvana (kopioi aliohjelma `convertGrayScale` ja kutsu sitä ennen bittikuvan lisäämistä ikkunaan). Tämän jälkeen tee vastaava aliohjelma poistaPunainen joka muuttaa harmaasävyksi kaikki ne pisteet, joissa punaista väriä on enemmän kuin 150. (ks: [Graphics](#) -dokumentaatio ja etsi [Bitmap](#)-kohdalta mm. funktioita joilla saat punaisen määrän)

8. Ota vaikka esimerkki: [Monimutkaisempikappale](#). Lisää ja toteuta tähän aliohjelma:

```

/**
 * Siirtää jokaista nuottia saman verran
 * @param mp soitin josta siirretään
 * @param siirto kuinka paljon siirretään
 * @param alkukanava mistä kanavasta siirto aloitetaan
 * @param loppukanava mihin kanavaan siirto lopetetaan
 */
public static void transponoi(BufferedMidiPlayer2 mp, int siirto,
                             int alkukanava, int loppukanava) {

```

Esimerkin kappaleessa alkukanava on 0 ja loppukanava (track) on 1. Kutsulla

```
List<Note> notes = mp.getNotes(k);
```

saat yhden raidan kaikki nuotit. Vastaavasti saat käytyä kaikki nämä nuotit läpi silmukalla:

```
for ( Note n: notes ) {
```

Nuotti koostuu nuotin arvosta, kestosta ja siitä kuinka nopeasti ”kosketin” lyödään alas. Katso Music -[dokumentaatiosta](#) miten saat [nuotin](#) arvon ja muuta sitä pyydetyllä määrällä ja aseta arvo takaisin nuottiin Tee tämä jokaiselle nuotille. Kutsu `trasponoi` aliohjelmaa ennen soiton käynnistämistä. Millä arvolla saat siirrettyä kokonaisen oktaavin?

- B1. **M: 14.4 Moniulotteiset taulukot:** Tee Java-ohjelma, jossa esittelet ja alustat 3-ulotteisen reaalityyppitaulukon. Esitele ja alusta myös 2-ulotteinen merkkijonotaulukko. Tee kaksi

tulosta -aliohjelmaa, joilla voit tulostaa em. taulukot.

## GURU-tehtävät

G1-2. **PNS:** Pienimmän neliösumman sovitus on eräs tapa laskea eräänlainen ”keskiluku” tai trendi aineistolle. Esimerkiksi meillä on havaintopisteitä, joiden periaatteessa pitäisi muodostaa ”suora”. Laskemalla PNS-suoran ( $y = a + b \cdot x$ ) kertoimet  $a$  ja  $b$  voimme piirtää aineistoa parhaiten kuvaavan suoran. Katso <http://mathworld.wolfram.com/LeastSquaresFitting.html> kertoimien laskukaavat (12) ja (14) ja tee ohjelma, joka piirtää aineiston ja sitä kuvaavan PNS-suoran. Aineisto piirretään kuten tehtävässä 1, eli voit käyttää siellä olevaa piirtoaliohjelmaa itse aineiston piirtämiseen (tai edellisen kerran Guru-tehtävän vastausta). Esimerkki: [pns.png](#).