Tietorakenteet, laskuharjoitus 7, 14.-19.3.

1. "Tira meets software engineering, osa 1"

   Lue luentomonisteen kalvot 233-236. Toteuta luokka `Opiskelijarekisteri` joka tarjoaa seuraavat palvelut:

   - opiskelijoiden lisääminen rekisteriin
   - opiskelijan tietojen haku nimen perusteella
   - opiskelijoiden tietojen haku numeron perusteella
   - kaikkien opiskelijanumeroiden tulostus, tulostus tapahtuu suuruusjärjestyksessä
   - aakkosjärjestyksessä tapahtuva nimien tulostus

   Molempien hakuoperaatioiden täytyy toimia ajassa $O(\log n)$ jos rekisterissä on n opiskelijaa. Tämän takia rekisterin sisäisessä toteutuksessa tulee käyttää Javan `TreeMap`-olioita. Tulostusoperaatioiden tulisi toimia ajassa $O(n)$ eli niissä ei saa tehokkuussyistä kutsua järjestämisoperaatiota.

   Huom: jos `Opiskelijarekisteri`-luokan koodi uhkaa kasvaa yli 40 riviseksi tai jonkun metodisi pituus alkaa ylittää 5 rivitä, taidat olla keksimässä pyörää uudelleen. Eli muista luottaa Java API:in!

   Seuraavassa testipääohjelma:

```
public static void main(String[] args) {
    Opiskelijarekisteri rekisteri = new Opiskelijarekisteri();

    rekisteri.lisaaOpiskelija(new Opiskelija("Wikla", "011"));
    rekisteri.lisaaOpiskelija(new Opiskelija("Lokki", "012"));
    rekisteri.lisaaOpiskelija(new Opiskelija("Ukkonen", "013"));
    rekisteri.lisaaOpiskelija(new Opiskelija("Niklander", "014"));
    rekisteri.lisaaOpiskelija(new Opiskelija("Mikkola",  "015"));
    rekisteri.lisaaOpiskelija(new Opiskelija("Smart", "016"));
    rekisteri.lisaaOpiskelija(new Opiskelija("Harjulehto", "017"));

    System.out.println("haetaan nimen perusteella");
    while ( true ){
        String haettava = lukija.nextLine();
        if ( haettava.equals("") ) break;
        Opiskelija haettu = rekisteri.haeNimenPerusteella(haettava);
        System.out.println( haettu );
    }

    System.out.println("haetaan opiskelijanumeron perusteella");
    while ( true ){
        String haettava = lukija.nextLine();
        if ( haettava.equals("") ) break;
        Opiskelija haettu = rekisteri.haeNumeronPerusteella(haettava);
        System.out.println( haettu );
    }
```

```
        rekisteri.tulostaNimetAakkosjarjestyksessa();
        rekisteri.tulostaOpiskelijanumerot();
    }
```

Luokan Opiskelija koodi:

```
import java.util.ArrayList;
import java.util.Random;

public class Opiskelija {
    private String nimi;
    private String opNro;
    private ArrayList<String> arvosanat;
    private static String[] kurssit = { "JTKT", "OhPe", "OhJa" };

    public Opiskelija(String nimi, String opNro) {
        this.nimi = nimi;
        this.opNro = opNro;
        arvosanat = new ArrayList<String>();

        Random r = new Random();
        for ( String k :kurssit ) {
            arvosanat.add(k + " "+ r.nextInt(6) );
        }

    }

    public String getOpNro() {
        return opNro;
    }

    public String getNimi() {
        return nimi;
    }

    public String toString() {
        String arvosMj = "";
        for ( String arvos : arvosanat ) {
            arvosMj += " "+ arvos + "\n";
        }
        return nimi + " ("+opNro+")\n" + arvosMj;
    }
}
```

2. "Tira meets software engineering osa 2"

   Yritys maksaa kuukausittaista korvausta yrityksessä töissä oleville sekä yrityksen alihankkijoille. Alihankkijat ovat normaaleja yrityksiä, eli niiden palkkalistoilla on henkilöitä ja lisäksi niillä voi olla alihankkijoita.

   Toteuta tietorakenneratkaisu, jonka avulla yrityksen koko palkkasumma saadaan selville. Palkkasumma koostuu työntekijöiden palkoista sekä alihankkijoiden palkkasummista. Alihankkijoiden palkkasummat taas koostuvat niiden työntekijöiden tai alihankkijana olevien yritysten palkkasummasta.

   Määrittele rajapinta `Palkittava` jonka sekä Työntekijät että Yritykset toteuttavat:

```
public interface Palkittava {
    int palkka();
}
```

   Ohje: älä keksi pyörää uudestaa vaan lue Ohjelmoistojen mallintamisen monisteen http://www.cs.helsinki.fi/u/mluukkai/ohmas10/ohma.pdf luku 3.8 "Esimerkki monimutkaisemman rakenteen mallintamisesta".

   Seuraavassa esimerkki, joka näyttää miten tietorakenneratkaisun tulisi toimia:

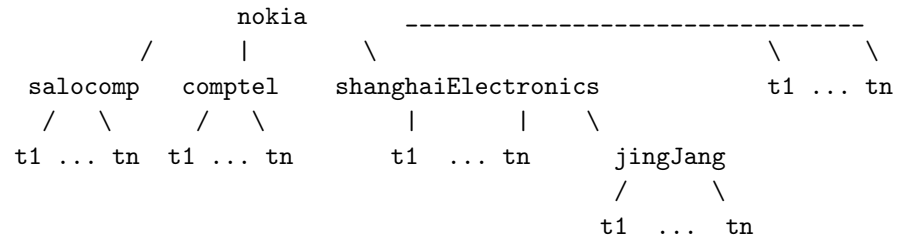
```
public static void main(String[] args) {
   Yritys nokia = new Yritys();

   nokia.lisaaPalkollinen(new Tyontekija(2000));
   nokia.lisaaPalkollinen(new Tyontekija(3000));
   nokia.lisaaPalkollinen(new Tyontekija(3000));
   nokia.lisaaPalkollinen(new Tyontekija(4500));
   nokia.lisaaPalkollinen(new Tyontekija(10000));

   Yritys comptel = new Yritys();
   comptel.lisaaPalkollinen(new Tyontekija(1800));
   comptel.lisaaPalkollinen(new Tyontekija(1850));
   comptel.lisaaPalkollinen(new Tyontekija(3000));

   nokia.lisaaPalkollinen(comptel);

   Yritys salocomp = new Yritys();
   salocomp.lisaaPalkollinen(new Tyontekija(1700));
   salocomp.lisaaPalkollinen(new Tyontekija(1750));
   salocomp.lisaaPalkollinen(new Tyontekija(2200));

   nokia.lisaaPalkollinen(salocomp);

   Yritys shElectr = new Yritys();
   for (int i = 0; i < 100; i++) {
       shElectr.lisaaPalkollinen(new Tyontekija(15));
   }
```

```
        Yritys jingJang = new Yritys();
        for (int i = 0; i < 1000; i++) {
            jingJang.lisaaPalkollinen(new Tyontekija(3));
        }

        shElectr.lisaaPalkollinen(jingJang);

        nokia.lisaaPalkollinen(shElectr);

        System.out.println("Palkkasummat");
        System.out.println("Comptel: " + comptel.palkka() + " euroa");
        System.out.println("Salocomp: " + salocomp.palkka() + " euroa");
        System.out.println("Shanghai Electronics: " + shElectr.palkka() + " euroa");
        System.out.println("JingJang: " + jingJang.palkka() + " euroa");

        System.out.println("Nokian palkkasumma: " + nokia.palkka() + " euroa");
    }
```

Tulostuu:

```
Palkkasummat:
Comptel: 6650 euroa
Salocomp: 5650 euroa
Shanghai Electronics: 4500 euroa
JingJang: 3000 euroa
Nokia: 39300 euroa
```

Huomaa, miten nokian palkolliset muodostavat esimerkissä puumaisen rakenteen. Osa puun solmuista on normaaleja työntekijöitä, osa taas edustaa alihankkijayrityksiä.



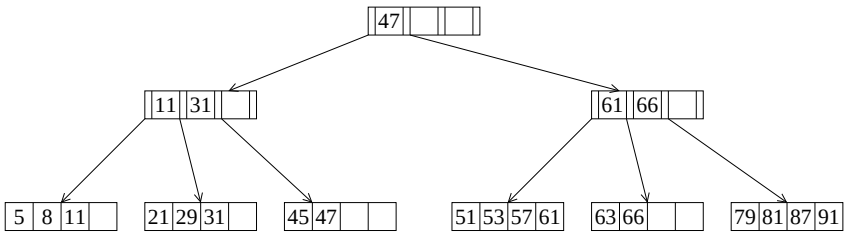
Ideana tietorakenteessa on, että sen käyttäjän ei tarvitse tuntea puun "solmuina" olevien olioiden tarkkaa tyyppiä vaan ainoastaan se, että jokaisella on metodi `palkka()`. Myös puun juuri käyttäytyy samalla tavalla. Eli käyttäjän kannalta oli yrityksen henkilöstöä edustavassa puussa miten monta alihankkijaa ja työntekijää tahansa, koko puu toimii samoin.

Vastaava tapa eli ns. *composite pattern* on hyvin yleinen olio-ohjelmoinnissa. Esim. Swing-käyttöliittymät noudattavat samaa periaatetta. Kaikki käyttöliittymäelementit totauttavat rajapinnan `Component`. Yksi elementti esim. koko sovelluksen GUI:ta

vastaava `JFrame` voi sisältää monia rajapinnan `Component` toteuttavia elementtejä, esim. nappeja, tekstikenttiä ym. Kun `JFrame`-elementille kutsutaan esim. metodia `setLocation`, siirtyy JFrame uuteen kohtaan näyttöä ja samalla siirtyvät "rekursiivisesti" kaikki JFramen sisältämät komponentit sopivasti ilman että sovelluksen tarvitsee niitä erikseen siirrellä.

Seuraavia tehtäviä varten joudut lukemaan osoitteesta olevan B+-puita esittelevän monisteen osoitteesta `http://www.cs.helsinki.fi/group/java/k11/tira/b.pdf`

3. Esitä tärkeimmät välivaiheet, kun allaolevaan B^+-puuhun lisätään avaimet 33, 25, 19, 43 ja 49.



4. Aluksi tyhjään B^+-puuhun lisään avaimet 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ja 12 (tässä järjestyksessä). Esitä lopputulos ja tärkeimmät välivaiheet, kun solmun maksimikoko on $2t = 4$.

5. Esitä tärkeimmät välivaiheet kun tehtävän 3 B^+-puusta poistetaan avaimet 5, 8 ja 47.

6. Poista vielä avain 45 edellisen tehtävän jälkeen tuloksena olevasta puusta. Näytä jälleen kaikki mielenkiintoiset välivaiheet