Ohjelmistojen mallintaminen, syksy 2010, laskuharjoitus 5

1. Seuraavassa on lueteltu käsitteitä, jotka ovat jossain suhteessa keskenään. Mitä suhdetyyppiä (normaali yhteys, kompositio, perintä, riippuvuus, ...) kussakin tapauksessa kannattaa käyttää (siis jokaiseen palloon oma suhde). Piirrä esim. pieni luokkakaavio jokaisesta tilanteesta.

   - Ruokaileva filosofi käyttää haarukkaa
   - Unixissa tiedosto on joko tavallinen tiedosto tai hakemisto
   - Tiedosto muodostuu tavuista
   - Monikulmion kulmat ovat järjestetty jono pisteitä
   - Opiskelija hallitsee Java-kielen
   - Ihmisellä on syntymävuosi
   - Hiiri, näppäimistö ja Näyttö ovat I/O-laitteita (eli tietokoneen syötelaite)
   - Polku yhdistää kaksi kylää
   - Monopolissa on 2-8 Pelaajaa. Pelaaja voi olla yksittäinen henkilö tai joukkue. Joukkue koostuu yksittäisistä henkilöistö
   - Kaikki tiet vievät roomaan (Vihavaisen laatima tehtävä)

2. Tee luokkakaavio ohjelmoinnin jatkokurssin viikon ylimääräisestä tehtävästä "Maidon elämä". Metodien nimien merkitseminen ei ole oleellista.

   Tehtävämäärittely löytyy osoitteesta:
   http://www.cs.helsinki.fi/u/avihavai/ekstraa.html

   Pystyt tekemään luokkakaavion tehtäväkuvauksen perusteella vaikka et olisikaan ohjelmoinut Maidon elämää.

   Maidon elämän Java-koodi tulee viimeistään pe 26.11. osoitteeseen
   http://wiki.helsinki.fi/pages/viewpage.action?pageId=63743590

3. Kuvaa sekvenssikaaviona mitä Maidon elämässä tapahtuu kun seuraava koodi suoritetaan:

```
Maatila maatila = new Maatila("Esko", new Navetta(new Maitosailio()));
// voit ajatella edellisen sekvenssikaaviossa seuraavanlaiseksi:
// Maitosailio maitosailio = new Maitosailio();
// Navetta navetta = new Navetta(maitosailio);
// Maatila maatila = new Maatila( navetta );

Lypsyrobotti robo = new Lypsyrobotti();
maatila.asennaNavettaanLypsyrobootti(robo);

Lehma lehma1 = new Lehma();
Lehma lehma2 = new Lehma();
Lehma lehma3 = new Lehma();

maatila.lisaaLehma(lehma1);
maatila.lisaaLehma(lehma2);

maatila.eleleTunti();

maatila.hoidaLehmat();

System.out.println(maatila);
```

4. Laajennetaan edellisellä viikolla aloitettua monopolin luokkamallia. Ota pohjaksi viime viikon ratkaisusi tai esimerkkivastaus.

   Voit olettaa, että kyseessä on tietokonepelinä toteutettava monopoli.

   Seuraavassa asioita, jotka pitäisi tulla mallissa esille.

   Ruutuja on useampaa eri tyyppiä:

   - aloitusruutu
   - vankila
   - sattuma ja yhteismaa
   - asemat ja laitokset
   - normaalit kadut (joihin liittyy nimi)

   Monopolipelin täytyy tuntea sekä aloitusruudun että vankilan sijainti.

   Jokaiseen ruutuun liittyy jokin toiminto.

   Sattuma- ja yhteismaaruutuihin liittyy kortteja, joihin kuhunkin liittyy joku toiminto.

   Toimintoja on useanlaisia. Ei ole vielä tarvetta tarkentaa toiminnon laatua.

   Normaaleille kaduille voi rakentaa korkeintaan 4 taloa tai yhden hotellin. Kadun voi omistaa joku pelaajista. Pelaajilla on rahaa.

   Tässä vaiheessa ei vielä liitetä peliin metodeja. Niiden aika tulee seuraavalla viikolla.

5. Ohjelmoinnin jatkokurssilla on tutustuttu sangen hyödylliseen luokkaan ArrayList. ArrayList-luokan olioden voi ajatella olevan automaattisesti itseään kasvattavia vaihtelevanmittaisia taulukoita, joihin voidaan säilöä muita olioita. Lyhyesti ilmaistuna ArrayList-oliot ovat "oliosäiliöitä".

   Etsi Java-API:sta (http://download.oracle.com/javase/6/docs/api/), ArrayList-luokka.

   Tutkitaan ArrayListin ominaisuuksia. Tällä kertaa ei kiinnitetä huomiota ArrayListin metodeihin vaan sen sijaintiin Java-API:n luokkahierarkiassa. Eli mikä on ArrayList:in yläluokka, mitä sisarluokkia (eli yliluokan muita aliluokkia) sillä on, mikä on ArrayListin yliluokan yliluokka, mitä serkkuluokkia ArrayListilla on, jne...

   Erityisesti on syytä selvittää mitä "sopimuksia" eli rajapintoja ArrayList mainitsee toteuttavansa? Mitä noissa rajapinnoissa luvataan?

   Piirrä tilanteesta luokkakaavio. Jos kaavio kasvaa liian isoksi, älä tunge kaikkea kuvaan. Metodinimiä ei kannattane luokkakaavioon ainakaan montaa merkitä.

   ArrayList ja muut oliosäiliöt liittyvät läheisesti kurssin Tietorakenteet aihepiiriin. Ammattimaisessa Java-ohjelmoinnissa tämä erilaisten "oliosäiliöiden" luokkaperhe on erittäin hyödyllinen. Kuten huomataan, on erilaisia säiliöluokkia todella suuri määrä. Joskus onkin haastavaa löytää parhaiten omiin tarpeisiin soveltuva luokka.

   Ohjelmoinnin jatkokurssin viikon 2 tehtävässä 4.2 tutstuttiin hieman ArrayListin tapaan toimivaan HashSet-luokkaan. **Miten HashSet-mukaan sijoittuu Javan luokkahierarkiaan ArrayList:in suhteen? Mikä selittää sen, että HashSet:iä ja ArrayList:iä voi käyttää monin paikoin toistensa tilalla? Mitä eroavaisuuksia löytyy?**

6. Luennoilla ja kalvoissa on mainittu 3 oliosuunnittelun periaatetta:

   - **Single responsibility principle**

     Luokilla tulisi olla vain yksi selkeä vastuu. Vastuulla oikeastaan tarkoitetaan potentiaalista syytä muutokselle. Ei ole esimerkiksi viisasta laittaa käyttäjän syötteen lukemista minkään sovellusolion tehtäväksi. Syötteen lukeminen on oma selkeä vastuunsa ja sitä varten tulisi olla oma luokka. Sovellusoliot sitten tarpeen vaatiessa kutsuvat syötteenlukijaolioita.

- **Program to interfaces, not to concrete implementations**

  Älä ohjelmoi siten, että olet riippuvainen konkreettisista luokista, on järkevämpi olla riippuvainen ainoastaan rajapinnasta. Esim. lypsyrobootista (ks. tehtävä 2) tulee monikäyttöisempi kun se määrittelee lypsettävän olevan joku rajapinnan Lypsava toteuttava olio:

  ```
  public void lypsa(Lypsava lypsettava) {
      // ...
      double maitoa = lypsettava.lypsa();
      sailio.lisaaSailioon(maitoa);
  }
  ```

  Tämä periaate kannattaa myös pitää mielessä kun käyttää edellisestä tehtävästä tuttuja säiliöluokkia. Itseasiassa tärkeää säiliöluokkien suhteen ovat juuri niiden toteuttamat rajapinnat, käyttäjän kannalta säiliöluokan sijainnilla perintähierarkiassa ei ole mitään merkitystä

- **Favor composition over inheritance**

  Rakenna toiminnallisuutta monimutkaisen perintähierarkian sijaan liittämällä yhteen joukko selkeän vastuun omaavia "pieniä" olioita. Hyvänä esimerkkinä tästä ohjelmoinnin jatkokurssin viikon 3 MuistavaTuotevarasto jota käsiteltiin luennolla

  Tämän periaatteen soveltaminen on siis yksi keino, jonka avulla saadaan oliot noudattamaan single responsibility -periaatetta.

- Muitakin periaatteita on olemassa, joihinkin niistä palataan kurssin aikana

Tehtävät:

(a) Toteuttaako Maidon elämä kaikkia em. periaatteita? Jos ei niin mikä rikkoutuu?

(b) Etsi ohjelmoinnin perusteiden ja jatkokurssin sekä tämän kurssin materiaaleista (luentomateriaalista ja laskareista) esimerkkejä (ainakin 3 esimerkkiä) jotka rikkovat jonkun ym. periaatteista. Näytä miten korjaisit tilanteen.

(c) Minkä takia em. periaatteet ovat tärkeitä? Saako niitä rikkoa? Missä tilanteissa?