1. Johdanto

1.1. Tehtävä

Muinaiset muukalaiset ovat jättäneet jälkeensä sokkelon, jossa on salaperäisiä esineitä ja vaarallisia robotteja. Toteuta Java-kielellä peli, jossa mönkijää ohjaavan pelaajan tehtävänä on kerätä mönkijän varastoon kaikki sokkelossa olevat esineet pääasiassa robotteja vältellen ja joskus niiden kanssa otellen.

Sokkelon ajatellaan olevan $n \times m$ -kokoisen ($n \geq 5$, $m \geq 5$) kaksiulotteisen taulukon kaltainen rakenne, joka koostuu osista. Rakenneosa on joko sokkelon seinää tai käytävää. Sokkelon kukin osa voidaan tunnistaa rivi-indeksin r ($0 \leq r <$ n) ja sarake-indeksin s ($0 \leq s < m$) määrittelemän paikan (r, s) avulla. Indeksit numeroituvat kaksiulotteisen taulukon tapaan. Sokkelon ensimmäinen paikka (0, 0) on sokkelon vasemmassa yläkulmassa, toinen paikka on ensimmäisen rivin toisessa sarakkeessa (0, 1) ja viimeinen paikka (n - 1, m - 1) on sokkelon oikeassa alakulmassa. Sokkelossa ei ole ulos- tai sisäänkäyntejä.

Rakenteiden lisäksi sokkelossa on käytäväosiin liittyvää sisältöä. Sisällön osista mönkijä ja robotit voivat liikkua käytävillä. Esineet pysyvät paikoillaan käytävälle paitsi, jos ne ovat mönkijän varastossa. Käytäväpaikka voi olla tyhjä. Käytävällä voi olla myös yksi tai useampi esine tai robotti. Esineet eivät estä robottien liikkumista: paikassa voi olla samaan aikaan esineitä ja robotteja. Mönkijä kerää automaattisesti paikassaan olevat esineet varastoonsa. Sokkelo sisältää pelin alussa mönkijän, vähintään yhden robotin ja vähintään yhden esineen.

Robotti ei salli mönkijän liikkua ohitseen. Samaan paikkaan sattunut robotti ja mönkijä kamppailevat kunnes jompikumpi häviää. Mönkijä kamppailee yhden robotin kanssa kerrallaan, jos paikassa on useampia robotteja. Kamppailun tulos määräytyy energian (≥ 0) avulla. Enemmän energiaa omaava kamppailija voittaa. Jos energioiden tasot ovat samat, mönkijän katsotaan voittavan. Myös esineillä on energiaa ja mönkijä voi muuntaa varastoimiensa esineiden energiaa omakseen. Mönkijän voimistamisen on kuitenkin tapahduttava ennen kamppailun alkua.

Käyttäjä liikuttelee mönkijää. Mönkijä voi liikkua pääilmansuuntiin (pohjoinen, itä, etelä ja länsi). Mönkijä siirtyy annettuun suuntaan, jos valitussa suunnassa on käytävää. Liikkuminen tapahtuu paikka kerrallaan. Mönkijä voi siirtyä vain johonkin naapuripaikoistansa, jotka ovat välittömästi mönkijän paikan vieressä pohjoisessa, idässä, etelässä ja lännessä olevat sokkelon paikat. Robotteja liikutellaan satunnaisesti apuluokan ja kahden rajapinnan avulla (luku 4.5).

Pelissä on kolme tulostuskomentoa. Näistä ensimmäinen tulostaa näytölle mönkijän tiedot ja listaa mönkijän varaston. Toinen komento tulostaa sokkelon karttamuodossa, jossa kukin paikka esitetään yhdellä merkillä. Seinä tulostetaan aina omalla merkillään. Käytävän merkki riippuu sen sisällöstä. Tyhjä käytävä esitetään omalla merkillään. Käytävällä oleva esine peittää käytävän merkin. Mönkijä ja robotti peittävät käytävän ja esineen merkin. Mönkijä ja robotti eivät peitä toisiaan, koska tietyssä paikassa voi kamppailun lopuksi olla vain mönkijä tai yksi tai useampi robotti. Kolmas komento tulostaa annetussa pääilmansuunnassa sijaitsevan mönkijän naapuripaikan.

Sokkelo ja sen sisältö on mahdollista säilöä tekstitiedostoon ja lukea tiedostosta keskusmuistiin. Apuluokan alustamiseen käytettävä siemenluku ([-999, 9999]) ja sokkelon rivien ja sarakkeiden lukumäärät luetaan tiedostosta ja tallennetaan tiedostoon.

Peli etenee käyttäjän ja pelin vuoroja vaihdellen. Pelaaja aloittaa pelin. Ohjelman siirtää omalla vuorollaan robotteja apuluokan avulla (luku 4.5), mutta käyttää vuoronsa vain, jos käyttäjä liikuttaa mönkijää tai ohittaa vuoronsa. Näin robotit pysyvät paikallaan, jos käyttäjä vaikkapa päättää tulostaa mönkijän naapuripaikan sisällön.

Peli loppuu, kun kaikki esineet on kerätty, mönkijä häviää kamppailun tai kun käyttäjä päättää ohjelman erillisellä komennolla. Peliä ei voi lopettaa sokkelosta poistumalla, koska sokkelon ulkoseinä on rikkumaton.

1.2. Työn organisointi

Harjoitustyö tehdään **itse ja lähinnä omalla ajalla**. Muiden kurssilaisten kanssa voi vaihtaa ideoita, mutta koodin kopiointi ei tietenkään ole sallittua. Samoin muualta kuin kurssin verkkosivulta löytyneen koodin kopiointi omaan ohjelmaan on kielletty.

1.3. Pakollisuus ja korvaavuudet

Harjoitustyö on pakollinen kurssin 10 opintopisteen laajuisena suorittaville opiskelijoille. Tällaisia kurssilaisia ovat erityisesti kaikki tietojenkäsittelytieteiden tutkinto-ohjelman omat opiskelijat. Ota yhteyttä tutkinto-ohjelmasi opintokoordinaattoriin tai kurssin vastuuopettajaan, jos et tiedä tuleeko sinun suorittaa kurssi 5 vai 10 opintopisteen laajuisena.

Harjoitustyön voi korvata A) muiden oppilaitosten opinnoilla, B) edellisellä Olio-ohjelmoinnin perusteet -kurssilla hyväksytyllä harjoitustyöllä tai C) edellisten kohtien tapaisella painavalla syyllä. Kohdan A perusteella on annettu kaikki pyydetyt korvaavuudet. Kohdan B perusteella annettavista harjoitustyökorvaavuuksista on tarkempia tietoja kurssin kotisivuilla. B-kohdan osalta on tärkeintä muistaa, että harjoitustyö korvautuu vain, jos ottaa yhteyttä kurssin vastuuopettajaan. Tähän mennessä tulleet yhteydenotot ja sopimukset on kirjattu ylös eikä uusia yhteydenottoja näiltä osin tarvita.

2. Sokkelon säilöminen

2.1. Tietorakenteet

Ohjelman pääasiallisena tietorakenteena käytetään luennoilla ja harjoituksissa käsitellystä *LinkitettyLista*-luokasta perittyä *OmaLista*-luokkaa (luku 4). Kunkin käytäväpaikan sisältö säilytetään *OmaLista*-tyyppisellä listalla. Myös mönkijän varastoimat esineet pidetään *OmaLista*-tyyppisellä listalla.

Sokkelon rakenne on helpointa esittää kaksiulotteisena taulukkona, jonka kussakin alkiossa on viite seinää tai sokkeloa mallintavaa luokkaa edustavaan olioon.

Huomaa, että *fi.uta.csjola.oope.lista*-pakkauksen sisältöä ei saa muuttaa millään tavoin. Älä lisää mönkijän, robottien, esineiden tai minkään muun luokan tietoja attribuuteiksi listan *Solmu*-luokkaan. Mönkijän, robottien ja esineiden on oltava solmujen tietoalkioina, jotta lista olisi edelleen yleiskäyttöinen. *Lista*-pakkaus on saatavilla kurssin kotisivujen *Koodit*-kohdasta.

Javan API:n tietorakenteita (esimerkiksi *ArrayList*-luokka) saa käyttää vain pienissä avustavissa tehtävissä. Kysy neuvoa harjoitustyön ohjaajalta, mikäli olet epätietoinen tietorakenteiden käyttöön liittyvissä kysymyksissä.

2.2. Tekstitiedosto

Harjoitustyössä tutustutaan laajemmin **tiedostojen käsittelyyn**. Tiedostojen avulla tietoja voidaan säilöä tietokoneen keskusmuistia pysyvämmin. Ohjelma ja sen tiedot ovat olemassa tietokoneen keskusmuistissa vain niin kauan kuin koneessa on virta päällä, mutta massamuistilaitteelle luoduissa tiedostoissa ohjelma ja tiedot ovat melko hyvässä tallessa seuraavaa käyttökertaa varten. Tiedoston sisältö jaetaan usein **tietueiksi** ja tietueet edelleen **kentiksi**. Kenttä sisältää yksittäisen tiedon, jonka esittämiseen käytetään yhtä tai useampaa merkkiä.

Huomaa, että tiedostot ovat eri asia kuin tietokantajärjestelmä. Java tarjoaa valmiit toiminnot tiedostojen käsittelyyn, jolloin tiedostoja voidaan lukea ja kirjoittaa ilman erillistä ohjelmistoa. Tiedostot soveltuvat pienimuotoiseen ja nopeaan tietojenkäsittelyyn, kun taas tietokannat ovat omimmillaan suurten tietomäärien järjestelmällistä hallintaa vaativissa tehtävissä. Javan voi liittää myös tietokantaan.

Tekstitiedostojen lukeminen ja tallentaminen on esitelty Lausekielinen ohjelmointi II -kurssin luentomateriaalissa [1]. Harjoitustyössä tarvittava tiedostojen käsittely onnistuu luentomateriaalin esimerkkejä suoraviivaisesti soveltamalla. Löydät tarvittaessa lisätietoja verkosta ja useimmista Java-kieltä käsittelevistä kirjoista. Esimerkiksi kurssikirjallisuuteen kuluvassa lähteessä [2] on annettu perusteellinen selvitys tiedostoista ja Java-kielestä. Huomaa, että tuoreimmissa lähteissä tiedostoja käsitellään Javan uusimpien versioiden ($\geq$ 1.5) mahdollistamin tavoin. *Scanner*-luokka on hyödyllinen myös tiedostonkäsittelyssä. (Katso neljännen viikkoharjoituksen neljäs tehtävä.)

Harjoitustyössä tiedostoon tallennetaan apuluokan siemenluku, sokkelon rivien ja sarakkeiden lukumäärät ja sokkelon rakenne- ja sisältöosat. Tiedoston **otsikkotietue** koostuu kolmesta kentästä, joissa on siemenluku, kentän rivien lukumäärä ja kentän sarakkeiden lukumäärä. Kukin **osatietue** sisältää omat kentät osan luokalle, paikalle ja mahdollisille luokan omille tiedoille. Mönkijän esineiden tietueet seuraavat mönkijän tietuetta ja käytävän sisällön tietueet seuraavat käytävän tietuetta. Näin mönkijällä ei ole esineitä, jos seuraava tietue ei ole esinetietue ja käytävä on tyhjä, jos käytävätietuetta seuraa seinän tai käytävän tietue.

Kentän paikkojen sen rakenneosille määräämä keskinäinen järjestys säilyy tiedostossa. Rakenneosat talletetaan kentän rivejä alusta loppuun kulkemalla, jolloin ensimmäisen rakenneosan (seinä paikassa (0, 0)) tiedot muodostavat ensimmäisen osatietueen ja viimeisen rakenneosan (seinä paikassa (n -1, m - 1)) tiedot tallentuvat viimeiseksi osatietueeksi.

Sokkelon tiedot tallennetaan *sokkelo.txt*-nimiseen tiedostoon. Tiedot esitetään siten, että yhdellä tiedoston rivillä on aina yksi tietue. Otsikkotietue on tiedoston ensimmäisellä rivillä. Tiedoston loput rivit ovat sokkelon osien tietueet.

Otsikkotietueen kenttien arvo esitetään kokonaislukuina. Ohjelmassa näiden kenttien tiedot ovat **int**-tyyppisiä. Osatietueen kolme ensimmäistä kenttää ovat kaikille osille yhteisiä:

1. Osan luokka. Tiedostossa jokin merkkijonoista "Kaytava", "Seina", "Monkija", "Robotti" ja "Esine". Ohjelmassa rakenne- tai sisältöosan olion tyyppi.
2. Osan rivi-indeksi. Ohjelmassa **int**-tyyppinen kokonaisluku.
3. Osan sarakeindeksi. Ohjelmassa **int**-tyyppinen kokonaisluku.

Mönkijällä, roboteilla ja esineillä on neljäs kenttä, jossa on sisältöosan energia, joka on ohjelmassa **int**-tyyppinen kokonaisluku. Mönkijällä ja robotilla on vielä viides kenttä, jossa on suunnan ilmaiseva merkki: 'p' (pohjoinen) 'i' = itä, 'e' = etelä ja 'l' = länsi. Suunta on ohjelmassa **char**-tyyppinen.

Luokan kenttä on yhdeksän merkin mittainen. Muissa kentissä on neljä merkkiä. Jos luokka on esimerkiksi "Monkija", on luokan kentässä merkkijono " Monkija ", jossa merkkijonon loppuun lisätyt kaksi välilyöntiä täydentävät kentän määrämittaiseksi. Kentät erotetaan toisistaan ja rivi päätetään yhdellä putkimerkillä ('|'). Kuvassa 1 on esitetty erään sokkelon sisältävä tiedosto.

Ensimmäisellä rivillä (tietueessa) määritellään, että siemenluku on 1 (ensimmäinen kenttä), rivien lukumäärä on 5 (toinen kenttä) ja sarakkeiden lukumäärä on 5 (kolmas kenttä). Tiedostosta on poistettu sokkelon kolmannen ja viidennen rivin osat. Koko tiedosto on saatavilla kurssin kotisivuilla.

Sokkelon ensimmäinen rivi koostuu seinästä. Toinen rivi alkaa seinällä, jota seuraa käytävä paikassa (1, 1). Tässä paikassa on myös mönkijä, jonka energia on 100 ja suunta etelään. Mönkijällä ei ole varastossaan esineitä, koska seuraavana on vuorossa käytävä paikassa (1, 2). Käytävällä on esine, jonka energia on 50. Toisen rivin viimeiset osat ovat käytävä ja seinä. Kolmas rivi alkaa seinästä ja loppuu seinään. Välissä on käytävä, jolla on esine (energiaa 500), seinä ja tyhjä

käytävä. Neljännellä rivillä on seinä, esineen (energia 100) sisältävä käytävä, seinä, robotin (energia 300 ja suunta pohjoinen) sisältävä käytävä ja seinä. Viides rivi koostuu ensimmäisen rivin tapaan seinästä.

```
1   |5   |5   |
Seina    |0   |0   |
Seina    |0   |1   |
Seina    |0   |2   |
Seina    |0   |3   |
Seina    |0   |4   |
Seina    |1   |0   |
Kaytava  |1   |1   |
Monkija  |1   |1   |100 |e   |
Kaytava  |1   |2   |
Esine    |1   |2   |50  |
Kaytava  |1   |3   |
Seina    |1   |4   |
Seina    |2   |0   |
Kaytava  |2   |1   |
Esine    |2   |1   |500 |
Seina    |2   |2   |
Kaytava  |2   |3   |
Seina    |2   |4   |
Seina    |3   |0   |
Kaytava  |3   |1   |
Esine    |3   |1   |100 |
Seina    |3   |2   |
Kaytava  |3   |3   |
Robotti  |3   |3   |300 |p   |
Seina    |3   |4   |
Seina    |4   |0   |
Seina    |4   |1   |
Seina    |4   |2   |
Seina    |4   |3   |
Seina    |4   |4   |
```

Kuva 1: Tiedoston s*okkelo.txt* sisältö.

Testeissä käytettävien tiedostojen sisällön oletetaan olevan **aina kunnossa**. Tietoja on oikea määrä, ne ovat oikeassa järjestyksessä, oikean tyyppisiä ja arvoiltaan laillisia. Tiedostossa on aina otsikkotietue ja vähintään viiden rivin ja viiden sarakkeen kokoisen sokkelon tiedot. Tiedostossa on aina mönkijä, vähintään yksi robotti ja yksi esine. Käytävän ja mönkijän varaston sisältö on aina järjestetty energian mukaan nousevaan järjestykseen. Kurssin kotisivuilla julkaistaan lisää esimerkkitiedostoja.

Vinkki: Tietuerivin pilkkominen kentiksi onnistuu *String*-luokan *split*-metodilla [1]. Kentät voidaan muuttaa kokonaisluvuiksi ja totuusarvoiksi kääreluokkien metodeilla. Esimerkiksi koko saadaan **int**-tyyppiseksi arvoksi *Integer*-luokan *parseInt*-metodilla.

3. Ohjelman toiminnot

Ohjelmassa on oltava toiminnot, jotka mahdollistavat sokkelon lataamisen tiedostosta keskusmuistiin (erityisesti listoille), erilaiset tietojen listaamiset (mönkijän tiedot, kartta ja naapuripaikan katsominen), mönkijän liikuttamisen, kerättyjen esineiden energiaksi muuttamisen, pelaajan vuoron ohittamisen, sokkelon tallentamisen listalta tiedostoon ja ohjelman lopettamisen. Seuraavassa esitellään ohjelman toiminnallisuutta pienten esimerkkien avulla. Laajempia esimerkkiajoja julkaistaan kurssin kotisivujen *Harjoitustyö*-kohdassa.

3.1. Ohjelman käynnistäminen

Ohjelman on käynnistyttävä komennolla:

```
java Oope2016HT
```

Näin ollen *main*-metodin sisältävän pääluokan on oltava aina nimeltään *Oope2016HT* ja pääluokan lähdekoodin on oltava *Oope2016HT.java*-tiedostossa.

Ohjelma lataa käynnistymisen yhteydessä tiedot tiedostosta keskusmuistiin aivan kuin ohjelmalle olisi annettu latauskomento (luku 3.3).

3.2. Käyttöliittymä

Ohjelmalla on yksinkertainen tekstipohjainen käyttöliittymä. Ohjelman käynnistyessä tulostetaan kuvassa 2 esitetyllä tavalla kehystetty teksti sekä omalle rivilleen teksti "Kirjoita komento:". Kehystämisessä käytetään reunamerkkinä asteriskia ('*') ja tekstin rivillä tekstin ja reunamerkin väliin tulostetaan yksi välilyönti. Tämän jälkeen jäädään odottamaan käyttäjän syötettä.

```
***********
* SOKKELO *
***********
Kirjoita komento:
```

Kuva 2: Ohjelma käynnistyksen jälkeen.

Ohjelmaa käytetään kirjoittamalla halutun toiminnon käynnistävä komento ja painamalla *Enter*-näppäintä. Komento ja sen mahdolliset parametrit erotetaan toisistaan yhdellä välilyönnillä. Kaikkien muiden kuin lopetuskomennon jälkeen pyydetään uusi komento siten, että tulostetaan "Kirjoita komento:" ja jäädään odottamaan syötettä omalla rivillä. Ohjelma toimii siis hieman komentoikkunan tapaan.

Kaikkien **tulosteiden muoto on kiinnitetty**, jotta ohjelmien automaattinen, tulosteiden vertailuun perustuva tarkistus olisi mahdollista (luku 6). Tarkistusta automatisoimalla pyritään siihen, että kurssin opettajille jää enemmän aikaa työn rakenteen ja laadun arviointiin.

Ohjelma vastaa virheelliseen komentoon tulostamalla omalle rivilleen "Virhe!" ja pyytämällä uutta syötettä. Virheeksi katsotaan tuntematon komento tai tunnettu komento virheellisillä parametreilla. Parametreja voi olla väärä määrä, niiden tyyppi voi olla virheellinen tai niiden arvot voivat olla arvoalueensa ulkopuolella.

Komennot pitää lukea *In*-apuluokan avulla, jotta ohjelmien tehtävänannon mukainen käyttäytymisen olisi todennäköisempää. Harjoitustyössä käytetään *apulaiset*-nimiseen pakkaukseen sijoitettua versiota *In*-luokasta, joka on saatavilla kurssin kotisivujen *Koodit*-kohdassa. Tässä paketissa on myös robottien liikuttelussa tarvittava luokka ja rajapinnat (luku 4.5).

3.3. Tietojen lukeminen tiedostosta

Komento "lataa" lukee *sokkelo.txt*-nimiseen tekstitiedostoon tallennetut tiedot. Siemenluvun avulla alustetaan apuluokka, joka liikuttelee robotteja paikasta toiseen. Sokkelon koko on tarpeen varsinkin, jos sokkelon rakenneosat ovat taulukossa. Komennolla ei ole parametreja.

Listat täytetään aina siten, että listan sisältö samassa järjestyksessä kuin tiedostossa. Lataus poistaa listalla mahdollisesti olevat aikaisemmat tiedot. Kuvassa 3 luetaan aiemmin esitellyn tiedoston (kuva 1) sisältämät tiedot keskusmuistiin. Latauskomento alustaa apuluokan siemenluvun avulla.

```
lataa
Kirjoita komento:
```

Kuva 3: Sokkelon tietojen lukeminen tiedostosta.

3.4. Mönkijän tietojen tulostaminen

Mönkijän ja sen mahdollisesti varastoimien esineiden tiedot tulostetaan "inventoi"-komennolla. Mönkijän ja sen varaston esineiden tiedot tulostetaan kukin omalle rivilleen samassa muodossa kuin ne esitetään tiedostossa (luku 2.2). Kuvassa 4 on listattu mönkijän tiedot latauksen jälkeen (luku 3.3). Mönkijä on paikassa (1, 1). Mönkijän energia on 100 ja suunta etelä. Mönkijän varastossa ei ole esineitä.

```
inventoi
Monkija  |1   |1   |100 |e   |
Kirjoita komento:
```

Kuva 4: Mönkijän tiedot.

3.5. Sokkelon tulostaminen

Sokkelo tulostetaan karttamuodossa "kartta"-komennolla. Seinän symboli on piste ('.'). Tyhjää käytävää symboloi välilyönti (' '). Mönkijä tunnistetaan isosta m-kirjaimesta ('M'), robotti isosta r-kirjaimesta ('R') ja esine isosta e-kirjaimesta ('E'). Symbolit peittyvät luvussa 1.1 annettujen sääntöjen mukaan. Kuvassa 5 on esitetty sokkelo tietojen lataamisen (luku 3.3) jälkeisessä lähtötilanteessa.

```
kartta
.....
.ME .
.E. .
.E.R.
.....
Kirjoita komento:
```

Kuva 5: Sokkelo karttamuodossa.

3.6. Naapuripaikan tulostaminen

Mönkijä naapuripaikka tulostetaan komennolla "katso x", missä *x* on pieni p-kirjain (pohjoinen), pieni i-kirjain (itä), pieni e-kirjain (etelä) tai pieni l-kirjain (länsi). Virheellinen parametri aiheuttaa virheilmoituksen. Komento tulostaa nähdyt osat samassa muodossa kuin ne ovat tiedostossa. Kuvassa 6 on esitetty, että mönkijä näkee eteläisessä naapurissaan käytävää, jolla on voimakas esine.

```
katso e
Kaytava  |2   |1   |
Esine    |2   |1   |500 |
Kirjoita komento:
```

Kuva 6: Naapuripaikan sisällön tulostaminen näytölle.

3.7. Mönkijän liikuttaminen

Mönkijää voidaan liikuttaa paikasta toiseen komennolla "liiku x", missä *x* on pieni p-kirjain (pohjoinen), pieni i-kirjain (itä), pieni e-kirjain (etelä) tai pieni l-kirjain (länsi). Virheellinen parametri aiheuttaa virheilmoituksen. Mönkijä siirtyy uuteen paikaan, jos siellä ei ole seinää. Seinää päin liikuttaessa ohjelma tulostaa viestin "Kops!" ja pitää mönkijän paikallaan.

Törmäys ja onnistunut liikkuminen antavat pelivuoron ohjelmalle, joka siirtää robotteja satunnaisesti. Robotti lisätään uudessa paikassaan energiansa määräämään kohtaan listalla (luku 4.3) siten, että robottien liikuttamisen jälkeen käytävien listojen sisältö on edelleen järjestetty energian mukaiseen nousevaan järjestykseen. Ohjelman vuoron jälkeen tulostetaan sokkelon kartta.

Paikkaan siirryttäessä siinä mahdollisesti olevat esineet siirtyvät automaattisesti mönkijän varastoon. Uusien esineiden lisäämisessä huomioidaan jo varaston listalla olevien esineiden energia (luku 4.3). Esineiden lisäyksen tulee tapahtua siten, että varaston sisältö järjestetty käytävän sisällön tapaan energian määräämään nousevaan järjestykseen.

Mönkijä joutuu ottelemaan kaikkia uudessa paikassa olevia robotteja vastaan. Ohjelma liikuttaa robotteja siirron jälkeen, jolloin on mahdollista, että mönkijän uuteen paikkaan siirtyy yksi tai useampi robotti. Myös tässä tapauksessa käydään otteluita, kunnes voittaja on selvillä. Jos mönkijä voittaa, jää uuteen paikkaan mönkijän symboli. Muussa tapauksessa uudessa paikassa on robotin symboli ja peli loppuu (luku 3.11).

Kuvassa 7 mönkijää siirretään askeleen etelään, jolloin varastoon siirtyy vahva esine (energia 500). Siirron seurauksena robotti on puolestaan liikkunut askeleen pohjoiseen omasta lähtöpaikastaan. Uusi askel etelään tuo varastoon toisen esineen (energia 100), joka sijoittuu listalla ennen ensimmäistä esinettä, jotta esineet olisivat listalla energian määräämässä kasvavassa suuruusjärjestyksessä. Ohjelma vastaa liikuttamalla robottia uudelleen pohjoiseen. Kolmas siirto etelään ei onnistu, mutta ohjelma ei anna armoa ja siirtää robotteja. Onneksi robotti palaa etelään.

3.8. Esineiden muuntaminen

Mönkijää voidaan voimistaa "muunna n"-komennolla, jolla *n* kappaletta mönkijän varastossa olevista esineistä muunnetaan energiaksi, joka lisätään mönkijän energiaan. Esineet valitaan muunnettaviksi listan alusta alkaen. Negatiivinen tai liian suuri *n* aiheuttaa virheilmoituksen.

Kuvassa 8 mönkijää siirretään kahdesti pohjoiseen kohti viimeistä esinettä. Toisen siirron jälkeen robotti ja esine ovat samassa paikassa, jolloin robotti peittää esineen (luku 1.1). Robotin nähdään olevan mönkijää voimakkaampi ja siten pieni dobing on paikallaan. Käyttäjä muuntaa molemmat esineet, jolloin mönkijän energia nousee 700:n (100 + 100 + 500) ja esineet katoavat varastosta.

```
liiku e
.....
. E .
.M.R.
.E. .
.....
Kirjoita komento:
liiku e
.....
. ER.
. . .
.M. .
.....
Kirjoita komento:
inventoi
Monkija  |3   |1   |100 |e   |
Esine    |3   |1   |100 |
Esine    |3   |1   |500 |
Kirjoita komento:
liiku e
Kops!
.....
. E .
. .R.
.M. .
.....
Kirjoita komento:
```

Kuva 7: Mönkijän ja robotin peliliikkeitä.

```
liiku p
.....
. ER.
.M. .
. . .
.....
Kirjoita komento:
liiku p
.....
.MR .
. . .
. . .
.....
Kirjoita komento:
katso i
Kaytava  |1   |2   |
Esine    |1   |2   |50  |
Robotti  |1   |2   |300 |1   |
Kirjoita komento:
muunna 2
Kirjoita komento:
inventoi
Monkija  |1   |1   |700 |p   |
Kirjoita komento:
```

Kuva 8: Esineiden muuntaminen energiaksi.

3.9. Vuoron ohittaminen

Pelaaja voi ohittaa oman vuoronsa "odota"-komennolla. Ohjelma käyttää pelaajan odottaessa oman vuoronsa samoin kuin, jos käyttäjä olisi liikuttanut mönkijää.

Kuvassa 9 käyttäjä päättää odottaa robotin siirtoa. Robotti päätyy mönkijän paikkaan ja ottelu alkaa. Mönkijä voittaa väännön energisempänä ottelijana, mutta menettää samalla energiaansa robotin energian verran (700 – 300 = 400).

Ohjelma tulostaa onnistumisen merkkinä viestin "Voitto!" ennen karttaa. Jos mönkijä olisi hävinnyt kamppailun, tulostuisi näytölle "Tappio!" ja ohjelma loppuisi (luku 3.11).

```
odota
Voitto!
.....
.ME .
. . .
. . .
.....
Kirjoita komento:
Monkija  |1   |1   |400 |p   |
Kirjoita komento:
```

Kuva 9: Vuoron ohittaminen.

```
tallenna
Kirjoita komento:
```

Kuva 10: Tietojen tallentaminen tiedostoon.

3.10. Tietojen tallentaminen tiedostoon

Otsikkotiedot ja listan sisältö tallennetaan parametrittomalla "tallenna"-komennolla *lotkot.txt*-tiedostoon. Vanha tiedosto korvataan uudella ilman varmistuskyselyjä. Kuvassa 10 on annettu esimerkki tallentamisesta. Tallennusmuoto on kuvattu luvussa 2.2. Sokkelon sisältö tallennetaan paikkojen määräämässä järjestyksessä. Kuva 11 esittää tiedoston sisällön tallennuskomennon jälkeen. Kuvasta nähdään, että robotti on kadonnut ja samoin esineet viimeistä esinettä lukuun ottamatta.

```
1   |5   |5   |
Seina    |0   |0   |
Seina    |0   |1   |
Seina    |0   |2   |
Seina    |0   |3   |
Seina    |0   |4   |
Seina    |1   |0   |
Kaytava  |1   |1   |
Monkija  |1   |1   |400 |p   |
Kaytava  |1   |2   |
Esine    |1   |2   |50  |
Kaytava  |1   |3   |
Seina    |1   |4   |
Seina    |2   |0   |
Kaytava  |2   |1   |
Seina    |2   |2   |
Kaytava  |2   |3   |
Seina    |2   |4   |
Seina    |3   |0   |
Kaytava  |3   |1   |
Seina    |3   |2   |
Kaytava  |3   |3   |
Seina    |3   |4   |
Seina    |4   |0   |
Seina    |4   |1   |
Seina    |4   |2   |
Seina    |4   |3   |
Seina    |4   |4   |
```

Kuva 11: Tiedoston *sokkelo.txt* uusi sisältö.

3.11. Ohjelman lopetus

Ohjelmasta poistutaan parametrittomalla "lopeta"-komennolla (kuva 12). Komennon jälkeen tulostetaan sokkelo ja lyhyet jäähyväiset ("Ohjelma lopetettu."-teksti). Samoin toimitaan, jos kaikki esineet on kerätty tai mönkijä häviää kamppailun.

```
lopeta
.....
.ME .
. . .
. . .
.....
Ohjelma lopetettu.
```

Kuva 12: Ohjelman lopetus komennolla.

4. Koodista

4.1. Yleistä

Ohjelma on jaettava **kapseloituihin luokkiin**, joiden sisältö puolestaan jaetaan järkevän mittaisiin metodeihin. Muistathan, että luokat rakennetaan tiedon – ei toimintojen – ympärille. Luokkametodeista koostuvat, esimerkiksi *Math*-luokan tapaiset apuluokat ovat poikkeus tähän sääntöön. Ohjelman toimintoja vastaavia luokkia (esimerkiksi *Lataus* ja *Tallennus*) **ei saa tehdä**, koska toiminnot eivät ole luokkia itsessään, vaan luokkien metodeja.

Olio-ohjelmoinnissa vain keskeisimmät käsitteet mallinnetaan. Luokkia tarvitaan vain sen verran, että ohjelma saadaan toimimaan olioperustaisesti edellä määritellyllä tavalla. Toisaalta kullakin luokalla tulisi olla yksikäsitteinen vastuu. Luokkien lukumäärää ei saa pienentää antamalla luokalle useita sille kuulumattomia vastuita. Erityisesti käyttöliittymäluokan sisältöä on syytä pitää silmällä. Käyttöliittymä vastaa vain ihmisen ja koneen vuorovaikutuksesta ja näin ollen sokkelon logiikasta vastaavaa koodia ei saa kirjoittaa sinne.

Noudata edelleen hyvää ohjelmointitapaa [3, 4]:

- Sisennä koodia johdonmukaisesti luettavuuden parantamiseksi.
- Kommentoi riittävästi ja oikeissa paikoissa.
- Käytä vakioita.
- Pidä metodit järkevän mittaisia (metodin tulisi mahtua yhdelle A4-kokoiselle sivulle).
- Kirjoita attribuutteihin, metodeihin ja luokkiin yleisluonteiset kommentit (luku 5).
- Nimeä vakiot, muuttujat, attribuutit, metodit ja luokat järkevästi.
- Käytä attribuutteja harkiten [5].
- Varaudu metodeissa virheellisiin parametrien arvoihin (erityisesti **null**-arvoinen viite), mikäli on todennäköistä, että metodille annetaan virheellisiä arvoja.
- Sijoita vain yksi luokka tai rajapinta yhteen lähdekooditiedostoon.

Sisennä koodi välilyönnein. Älä käytä sisentämiseen tabulaattoria, jotta koodisi näkyisi samanlaisena myös ohjaajan editorissa. Välilyöntejä ja tabulaattorimerkkejä ei saa käyttää “sekaisin”, koska tällöin on varmaa etteivät sisennykset näy ajatellulla tavalla.

4.2. Sokkelon osien luokkahierarkia

Periytyminen on harjoitustyössä keskeisessä roolissa. Periytymisessä samoja piirteitä sisältäville luokille tehdään yliluokka, jonne yhteiset piirteet siirretään. Harjoitustyötä ei saa toteuttaa siten, että sokkelon kaikki osat esitetään yhtä luokkaa käyttäen. Ohjelmassa tulee olla abstrakti luokka, jotta käytetään juuriluokkana sekä sokkelon rakenneosille että sokkelon sisältöosille. Juuressa on kaikille osille yhteiset tiedot eli osan paikan rivi- ja sarakeindeksit.

Peri tästä luokasta joko suoraan tai abstraktin väliluokan kautta sokkelon rakenneosia mallintavat konkreettiset *Kaytava*- ja *Seina*-luokat. Tee abstrakti luokka sisältöosille ja peri tästä luokasta konkreettiset aliluokat *Monkija*, *Robotti* ja *Esine*, joiden yhteinen tieto on energia.

Olion merkkijonoesityksestä on paljon iloa tässäkin työssä. Korvaa siksi *Object*-luokan *toString*-metodi luokkahierarkian jokaisella tasolla. Tee korvaus "ketjuttaen": kutsu aliluokkien *toString*-metodin korvauksissa yliluokan versiota **super**-attribuutin avulla juuriluokka pois lukien, koska *Object*-luokan *toString*-metodia ei pidä kutsua.

Luokkahierarkiassa ei saa olla attribuuttia luokan nimelle. Selvitä olion luokka hierarkian juuriluokassa metaolion palveluiden avulla.

Kaytava- ja *Monkija*-luokilla on *OmaLista*-tyyppinen (luku 4.3) attribuutti. Käytävän listalle säilöään käytävän mahdollinen sisältö (mönkijä, robotit ja esineet), kun taas mönkijän listalla ovat mönkijän mahdollisesti varastoimat esineet.

Toteuta *Comparable*-rajapinnan *compareTo*-metodi mönkijän, robotin ja esineen yliluokassa siten, että metodissa vertaillaan energioita. Käytä metodia mönkijän ja robotin vertailuun kamppailussa sekä lisäyspaikan hakemiseen *OmaLista*-luokan operaatiossa (luku 4.3).

4.3. OmaLista-luokka

Käytä *LinkitettyLista*-luokasta perimääsi *OmaLista*-luokkaan sisällön säilömiseen luvuissa 1.1 ja 4.2 kuvatuilla tavoilla. Harjoitustyössä tutustutaan linkitettyyn listarakenteeseen myös omia operaatioita tekemällä. Operaatioiden tulee olla "puhtaita" listaoperaatioita, joita voitaisiin käyttää myös muissa tehtävissä. Näin *OmaLista*-luokassa ei saa esiintyä harjoitustyön luokkien tunnuksia. Esimerkiksi nimiä *Monkija*-, *Robotti*- ja *Esine* ei saa käyttää oman listaluokan operaatioissa.

Tee *OmaLista*-luokkaan vähintään metodi, joka lisää listalle alkion siten, että alkio sijoittuu kaikkien itseään pienempien tai yhtä suurien alkioiden jälkeen ja ennen kaikkia itseään suurempia alkioita. Vertaile alkioita *Comparable*-rajapinnan *compareTo*-metodilla. Lisää robotit käytävän listalle ja esineet mönkijän listalle tällä metodilla, jolloin listan alkiot pysyvät energiansa mukaisessa nousevassa järjestyksessä. Metodin otsikko voisi olla seuraava:

```
/** Listan alkiot säilyttävät kasvavan suuruusjärjestyksen,
  * jos lisäys tehdään tällä operaatiolla.
  *
  * @param alkio viite olioon, jonka esivanhempi tai luokka
  * on toteuttanut Comparable-rajapinnan.
  * @throws IllegalArgumentException, jos oliolla ei ole
  * Comparable-rajapinnan toteutusta.
  */
 @SuppressWarnings("unchecked") // Estetään kääntäjän varoitus.
 public void lisaa(Object alkio) {
```

Metodin otsikkoa edeltää *SuppressWarnings*-annotaatio, jolla estetään kääntää varoittelemassa turhaan *Comparable*-rajapinnan vaarallisesta käytöstä.

Muita hyödyllisiä *OmaLista*-luokan metodeja ovat erilaiset listalta alkioita poistavat metodit, joihin on tutustuttu seitsemännen harjoituksen toisessa ja kolmannessa tehtävässä.

Muista, että *lista*-pakkauksen sisältöä (esimerkiksi *Solmu*-luokkaa) ei saa muuttaa.

4.4. Koodin organisointi

Main-metodin sisältävän ajoluokan tulee olla *Oope2016HT.java*-nimisessä lähdekooditiedostossa. Ajoluokassa ei saa olla pakkausmääreitä, jotta ohjelman automaattinen testaus WETO-järjestelmässä onnistuu. Ohjelman on käännyttävä suoraan ilman komentorivillä annettavia lisämääreitä komennolla *javac Oope2016HT.java*. Samoin ohjelman ajamisen on tapahduttava suoraan komennolla *java Oope2016HT*.

Harjoitustyössä käytetään *apulaiset*-pakkaukseen sijoitettua *In*-luokkaa tietojen lukuun näppäimistöltä. Tähän pakkaukseen on liitetty myös *Automaatti*-luokka sekä *Paikallinen*- ja *Suunnallinen*-rajapinnat (luku 4.5), joita tarvitaan robottien liikutteluun sokkelossa. *Apulaiset*-pakkausta ja sen luokkia ei saa muuttaa millään tavalla. *Apulaiset*-pakkauksen tulee olla ohjelman ajoluokan sisältävän hakemiston alihakemistona. Löydät *apulaiset*-pakkauksen kurssisivujen *Opetus* | *Harjoitustyö* -kohdasta.

Tee sokkelon osien luokkahierarkialle oma pakkaus. Myös tämä pakkaus on ajoluokan sisältävän hakemiston alihakemisto. Voit tehdä halutessasi lisää pakkauksia. Nämäkin pakkaukset ovat ajoluokan hakemiston alihakemistoja.

Käytä mahdollisimman vähän *File*-luokan palveluja, koska WETO-järjestelmän tietoturvamääritykset on säädetty tiukoiksi. WETO ei esimerkiksi hyväksy *delete*-metodia käyttävää koodia.

4.5. Robottien liikuttelu

Apulaiset-pakkaus (luku 4.4) sisältää *Automaatti*-luokan, jota käytetään robottien liikutteluun. Luokka olettaa, että se voi kutsua *Paikallinen*- ja *Suunnallinen*-rajapintojen toteutettuja metodeja.

Koska *Paikallinen*-rajapinta määrittelee paikkaan liittyviä aksessoreita, on rajapinnan luonteva toteutuspaikka sokkelon osien luokkahierarkian juuriluokka, jossa on attribuutit osan rivi- ja sarakeindeksille. *Suunnallinen*-rajapinnan toteutus tulee samaan luokkahierarkiaan, mutta matalammalle tasolle. Kenties paras paikka tämän rajapinnan toteutukseen ovat suuntansa tuntevat *Monkija*- ja *Robotti*-luokat.

Luokalla on kaksi julkista luokkametodia (luokan 3. kohta):

- *Alusta*-metodi on tarkoitettu satunnaislukugeneraattorin alustamiseen. Anna tiedostosta luettu siemenluku metodin parametriksi latauksen (luku 3.3) yhteydessä ennen *Automaatti*-luokan toisen metodin kutsumista.

- *PaivitaPaikat*-metodi saa ensimmäisenä parametrinaan listan, jolla on viitteet robottiolioihin ja toisena parametrinaan kaksiulotteisen taulukon, jonka alkioista on viitteet sokkelon rakenneosiin eli seinä- ja käytäväolioihin. Metodin käyttö on helppoa, kun sokkelon rakenneosat ovat suositellulla tavalla kaksiulotteisessa taulukossa (luku 2.1). Muunlaisten rakenteiden käyttäjien pitää tehdä metodi, joka tuottaa *paivitaPaikat*-metodin vaatiman taulukon. Metodi olettaa, että listan viitteisiin liittyvät oliot ovat luokasta, joka on toteuttanut *Paikallinen*- ja *Suunnallinen*-rajapinnat tai perii näiden rajapintojen toteutukset. Taulukon osalta oletetaan, että taulukon viitteisiin liittyvien olioiden luokissa on toteutettu *Paikallinen*-rajapinta tai että luokat perivät tämän rajapinnan toteutuksen. Metodi muuttaa *Paikallinen*-rajapinnan metodien avulla robottien paikkoja ja vaihtaa mahdollisesti robottien suuntia *Suunnallinen*-rajapinnan metodeja käyttäen. Metodi ei muuta taulukon sisältöä. Metodi heittää poikkeuksen, jos satunnaislukugeneraattoria ei ole alustettu *alusta*-metodia kutsumalla tai parametreissa havaitaan virhe.

Metodeihin on syytä tutustua tarkemmin lukemalla metodien otsikot ja metodien yleiset kommentit. Molemmat metodit ovat **static**-määreellä esiteltyjä luokkametodeja, joita kutsutaan *In*- ja *Math*-luokkien metodien tapaan luokan nimen kautta. Esimerkiksi *alusta*-metodin kutsu voisi olla:

```
Automaatti.alusta(siemen);
```

Yllä *siemen* on tiedostosta luetun siemenluvun sisältävä muuttuja.

Älä käytä muita menetelmiä robottien liikutteluun, jotta ohjelmasi toimii varmasti samalla tavoin kuin malliratkaisu. Älä muuta *Automaatti*-luokkaa millään tavalla. Älä kopioi luokan metodeja ohjelmaasi.

5. Dokumentointi

Harjoitustyöstä kirjoitetaan dokumentti, jonka tulee sisältää seuraavat asiat:

1. Kansilehdellä tekijän nimi, opiskelijanumero ja sähköpostiosoite. Sivun keskellä tulisi olla suuremmalla fontilla dokumentin nimi. Kurssin kotisivuilla on annettu esimerkinomainen kansilehti.

2. Luokkakaavio ja sen lyhyt sanallinen kuvaus.

3. Omia ajatuksia. Esimerkiksi: Oliko työ helppo, sopiva tai vaikea? Miksi koit työn helpoksi, sopivaksi tai vaikeaksi? Mitä uutta opit? Oliko työstä mitään hyötyä tekijälleen? Paljonko aikaa käytit työhön?

Dokumentin leipäteksti kirjoitetaan 12 pisteen fontilla ja yhdellä rivinvälillä. Valmiin tekstin lukeminen pariin otteeseen ei ole huonompi idea. Dokumentin kirjoitus tekstinkäsittelyohjelmalla sekä ohjelmasta löytyvän oikolukutoiminnon käyttäminen tekstin tarkistamiseen on myös suotavaa.

Luokkakaavio piirretään UML-notaatiota käyttäen. Ohjelmaa ei tarvitse suunnitella UML:ia käyttäen. Riittää, että toteutettu ohjelma mallinnetaan UML:lla. Kussakin luokkasymbolissa esitetään luokan kaikki attribuutit ja julkiset metodit. Jos kaavion piirtoon käytetty työväline ei osaa piilottaa metodeja, niin kätketyt (**private**) metodit voi jättää näkyviin.

Luokkakaaviossa esiintyvien luokkien väliset suhteet mallinnetaan luokkien välisinä suhteina. Piirrä periytymis- ja toteuttamissuhteiden lisäksi näkyviin luokkien väliset **assosiaatiot**.

Jos jossakin itse kirjoitetussa luokassa *A* on attribuutti, jonka tyyppi on toinen oma luokka *B*, niin tätä attribuuttia ei esitetä luokan *A* laatikossa, vaan *A*- ja *B*-luokkien laatikoiden välisenä assosiaatioviivana. Anna kaikille assosiaatioille nimet, suunta sekä rooli ja kertautumiset.

Piirrä luokkakaavioon myös luokkien väliset riippuvuussuhteet. Riippuvuussuhde on assosiaatiota heikompi suhde luokkien ilmentymien välisen hetkellisen yhteistyön ilmaisemiseen. Jos oma luokka *C* hyödyntää toisen oman luokan *D* palveluja ilman attribuuttia, niin luokkalaatikoiden välille voi piirtää nuolen (varsi katkoviivaa) luokasta *C* luokkaan *D* riippuvuussuhteen merkiksi.

Vahvahermoiset kurssilaiset saavat Office- ja OpenOffice-ohjelmistoilla aikaiseksi siedettävän näköisiä luokkakaavioita. Kaavioiden tuotto on kuitenkin helpompaa erityisesti UML-kaavioiden piirtoon tehtyjä ohjelmistoja käyttäen. Dia (https://wiki.gnome.org/Apps/Dia) on ilmainen ohjelma, jolla voi piirtää vuo- ja ER-kaavioiden lisäksi myös UML-kaavioita.

Jotkin ohjelmat osaavat muodostaa osan luokkakaaviosta automaattisesti lähdekoodin perusteella. Tällainen on UMLet-ohjelma, joista kerrotaan lisää kurssisivujen *Opetus* | *Harjoitustyö* | *UML* -kohdassa. NetBeans- ja Eclipse-ohjelmistojen kaltaisiin kehitysympäristöihin on saatavilla takaisinmallinnuksen hallitsevia lisäosia.

Koodi dokumentoidaan kirjoittamalla lyhyet yleisluonteiset **Javadoc-kommentit** jokaiselle attribuutille, metodille ja luokalle. Javadoc on Java-ympäristön tarjoama menetelmä koodin puoliautomaattiseen dokumentoimiseen. Javadoc-kommentoinnista annetaan erilliset ohjeet kurssin verkkosivujen *Opetus* | *Harjoitustyö* | *Javadoc* -kohdassa.

Huomaa, että rakentajia ja aksessoreita ei tarvitse kommentoida, mikäli ne ovat toiminnoiltaan tavanomaisia. Muista edelleen selittää **metodien sisällä** normaaliin tapaan metodien keskeiset ja vaikeaselkoiset osuudet **tavanomaisia kommentteja** käyttäen.

5. Ohjaus

Harjoitustyön pääasiallinen ohjaaja on mikroharjoitusryhmäsi vetäjä. Näet ryhmäsi WETO-järjestelmän *Students*-välilehdeltä. Kysy neuvoa sopivimmaksi katsomaltasi opettajalta, jos et ole valinnut ryhmää, nimesi ei ole listalla tai et ole käynyt alun perin valitsemassasi ryhmässä.

Henkilökohtainen ohjaus on aluksi luokkakaavioiden pohdintaa. Vastuuopettaja tarjoaa kommentteja luokkakaavioista ja ohjelman rakenteesta. Varsinainen ohjelmointiin liittyvä mikroluokkaohjaus alkaa myöhemmin. Ohjausten ajat ja paikat ilmoitetaan myöhemmin kurssin kotisivuilla. Luuppi tarjoaa ohjausta harjoitustyöhön koodauspajassaan. Vastuuopettajaa voi käydä tapaamassa myös muina aikoina. Tapaamisaika kannattaa sopia etukäteen, jos haluaa varmistaa, että vastuuopettaja on paikalla huoneessaan.

Pienemmät kysymykset kannattaa esittää ohjaajille sähköpostitse. Kysymyksiä ja niihin annettuja vastauksia kerätään kurssin verkkosivujen *Harjoitustyö*-kohtaan. Ennen kysymistä kannattaa siis tarkistaa kurssin verkkosivut – vastaus voi hyvinkin olla jo sivuilla. Monet ongelmat selviävät myös tehtävänantoa lukemalla ja tutustumalla verkkosivuilta löytyviin esimerkkiajoihin.

Kysyminen kannattaa erityisesti, kun on epävarma suuremman mittakaavan kysymyksissä. On pienempi vaiva selvittää epäselvä kohta ajoissa ohjaajan kanssa, kuin korjata valmista ohjelmaa jälkikäteen tehtävänantoa vastaavaksi.

6. Palautus ja tarkistus

Ohjelma ja dokumentti täytyy palauttaa viimeistään **tiistaina 5.4.2016 klo 16.00** WETO-järjestelmään. Aluksi ajateltua takarajaa on siten venytetty noin puolellatoista vuorokaudella, koska tarkan tehtävän annon julkaisu viivästyi suurin piirtein tämän verran. Tarkemmat palautusohjeet julkaistaan myöhemmin kurssin verkkosivuilla.

WETO-järjestelmä tarkistaa harjoitustöiden toiminnallisuuden vertailemalla automaattisesti mallivastauksen ja opiskelijoiden ratkaisujen tulosteita. Tästä syystä **edellä annettuja tulostemäärittelyjä on seurattava merkilleen**. Automaattinen vertailu vähentää rutiininomaista testaustyötä, jolloin opettajille jää enemmän aikaa mielekkäämpään työhön eli ohjelman rakenteen ja tyylin tutkimiseen. Opiskelijat hyötyvät tästä perusteellisempien kommenttien muodossa.

Lisäaikaa työn tekoon voi saada muutaman päivän hyvästä syystä. Lisäajasta on sovittava ohjaajan kanssa ajoissa eli viimeistään päivää tai paria ennen palautuksen takarajaa. Lisäaikana harjoitustyön ohjaaja ei välttämättä ole tavattavissa henkilökohtaisesti.

Ennen palautusta on syytä varmistaa, että dokumentissa on mukana kaikki edellä mainitut kohdat. Lisäksi kannattaa tarkistaa, että ohjelma toimii varmasti oikein viimeisimpien muutosten jälkeen.

7. Arvostelu

Arvostelu perustuu ensisijaisesti ohjelman oikeellisuuteen: ohjelman on toimittava testeissä tehtävänannossa määritellyllä tavalla. Harjoitustyö voidaan joko hylätä tai hyväksyä, jolloin työ arvostellaan asteikolla 0–4 pistettä. Pisteet lisätään harjoitushyvityspisteiden tapaan tenttipisteisiin vain, mikäli tentistä on saanut vähintään vaaditut 12 pistettä.

Maksimipisteet eli 4 pistettä voi saada vain, mikäli työ on palautettu ilman lisäaikaa. Tämän vaatimuksen osalta voidaan joustaa äärimmäisen hyvästä syystä, jollaisia ovat esimerkiksi pitkä ulkomaan matka tai kokopäivätöiden aiheuttamat ylitsepääsemättömät esteet. Täydet pisteet (4 pistettä) saa vain, jos ohjelma toimii oikein sekä julkisissa että salaisissa testeissä. **Ilman edellä mainittua todella hyvää syytä lisäajalle menneestä työstä ei voi saada hyvityspisteitä**.

Huomaa, että salaisissa testeissä käytetään eri syötteitä kuin kotisivuilla annetuissa esimerkeissä. Ohjelman oikeellinen toiminta esimerkkien osalta ei takaa vielä salaisten testien läpäisyä. Toisaalta salaisten testien läpäisystä ei ole mitään toivoa, jos ohjelma ei toimi oikein julkisten testien osalta. Siksi on erittäin toivottavaa, että kaikki testaavat ohjelmansa perusteellisesti.

Hylkäyksen perusteena voi olla tehtävänannon noudattamatta jättäminen, ohjelman toimimattomuus, harjoitustyön teossa kiellettyjen Javan ominaisuuksien käyttö, huono dokumentti, hyvän ohjelmointitavan noudattamatta jättäminen ja/tai plagiointi. Plagiointiin liittyy sanktio, joka koskee molempia opiskelijoita. (Toiselta opiskelijalta tämän tietämättä kopioidun koodin käyttö johtaa kopioijan koko kurssisuorituksen hylkäämiseen.) Hylätyn työn voi palauttaa korjattuna kerran tai pari. Korjausaikaa on pääsääntöisesti viikko työn hylkäyksestä. Pisteytys perustuu aina ensimmäiseen palautukseen. Korjauskierroksen kautta ei voi korottaa pistemääräänsä.

Lähteet

1. J. Laurikkala, Lausekielinen ohjelmointi II -kurssin luentorunko, luku 8, http://www.uta.fi/sis/tie/laki2/opetus/luennot.html (Luettu viimeksi 2.3.2016.)

2. H. Deitel ja P. Deitel, *Java, How to Program*, Prentice Hall.

3. J. Laurikkala, Lausekielinen ohjelmointi I -kurssin luentorunko, luku 14, http://www.uta.fi/sis/tie/laki1/opetus/luennot.html (Luettu viimeksi 3.3.2016.)

4. J. Laurikkala, Lausekielinen ohjelmointi II -kurssin luentorunko, luku 6, http://www.uta.fi/sis/tie/laki2/opetus/luennot.html (Luettu viimeksi 3.3.2016.)

5. J. Laurikkala, Olio-ohjelmoinnin perusteet -kurssin luentorunko, luku 4, http://www.uta.fi/sis/tie/oope/opetus/luennot.html (Luettu viimeksi 3.3.2016.)