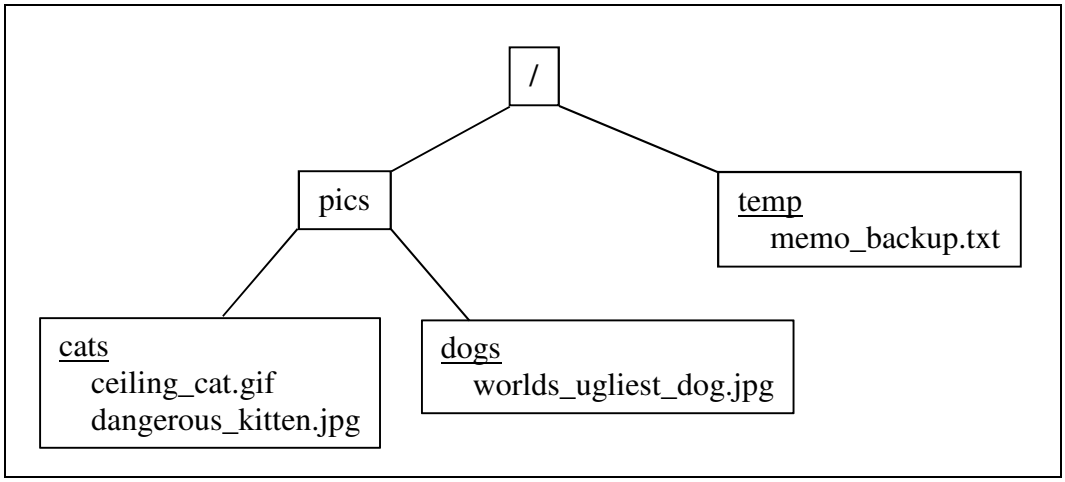
1. Johdanto

Komentoikkuna [1, 2] on teksuaalinen käyttöliittymä, jonka kautta käyttöjärjestelmää ohjaillaan komentoja kirjoittamalla. Komentoikkuna tunnetaan myös muun muassa komentorivinä (command prompt), konsolina (console), terminaalina (terminal), kuorena (shell) ja komentotulkkina (command interpreter). Tarkkaan ottaen komentotulkki on komentoikkunassa ajettava ohjelma, joka lukee käyttäjältä komennot ja muuntaa ne käyttöjärjestelmän ymmärtämään muotoon, mutta koska komentoikkunan avautuessa käynnistetään lähes aina jokin komentotulkki, ymmärretään komentotulkki käytännössä komentoikkunaksi.

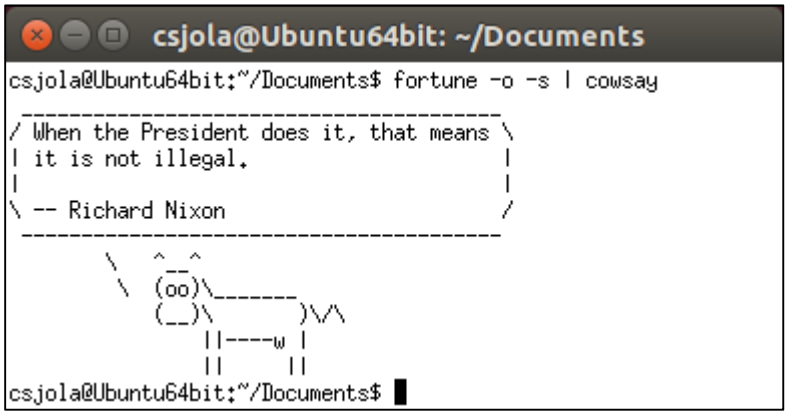
Suuri osa komentotulkissa saatavilla olevista komennoista liittyy tiedostojen ja hakemistojen käsittelyyn. Esimerkiksi hakemistosta toiseen siirtymiselle ja hakemiston sisällön listaamiselle on omat komennot. Komentotulkki tukeutuu muiden sovellusten tapaan massamuistin käsittelyssä tiedostojärjestelmänä [3] tunnetun käyttöliittymän osan tarjoamiin palveluihin. Tiedostojärjestelmä esittää massamuistissa olevat tiedot siten, että sovellusten ei tarvitse huolehtia hyvinkin monimutkaisesta alkuperäisestä esitystavasta. Tiedostojärjestelmä näyttäytyy tietokoneen käyttäjälle hakemistojen muodostamana puumaisena rakenteena, joka on yleensä käännetty. Juurihakemisto on ylimpänä ja puun lehtihakemistot alimpana. Kuvassa 1 on esitetty eräs hakemistopuu.



Kuva 1: Eräs hakemistopuu. Suorakaiteet symboloivat hakemistoja. Juurihakemiston nimenä käytetään kauttaviivaa. Viivat yhdistävät alihakemistot ja ylihakemistot.

Eri käyttöjärjestelmissä on samankaltaisia komentoja yleisemmille komennoille. Komennot on kuitenkin yleensä nimetty eri tavoin ja samannimisetkin komennot voivat toimia toisistaan poikkeavasti. Esimerkiksi hakemiston listaus tapahtuu Windows-järjestelmässä *dir*-komennolla, kun taas Linux- ja Mac-järjestelmissä käytetään *ls*-komentoa.

Hakemistopuun aktiivista hakemistoa kutsutaan nykyhakemistoksi tai työhakemistoksi. Komentoikkunassa on komentokehote, joka näyttää tyypillisesti juurihakemistosta nykyhakemistoon johtavan hakemistopolun. Komennot kirjoitetaan yleensä samalle riville kuin kehote ja syötetään komentotulkille rivinvaihtonäppäintä painamalla. Kuvassa 2 on annettu kaksi komentoa käyttäjän *csjola* kotihakemiston *Documents*-alihakemistossa.



Kuva 2: Erään Linux-käyttöjärjestelmän komentoikkuna ja siinä annettuja komentoja.

Harjoitustyössä on tehtävänä ohjelmoida Java-kielellä komentoikkunassa toimivaa komentotulkkia simuloiva olioperustainen Simple Oope Shell (SOS) -ohjelma. Tiedostojärjestelmä toteutetaan linkitettynä rakenteena, jossa hakemistojen sisällöt säilötään linkitetyille listoille (luku 3). Tiedostojärjestelmä tallennetaan keskusmuistiin, jolloin harjoitustyöohjelmassa annetut komennot eivät aiheuta muutoksia massamuistiin. Erityisesti tiedoston tai hakemiston poistava komento ei poista tietoja levyltä, vaan aiheuttaa muutoksen vain keskusmuistissa olevaan rakenteeseen.

Harjoitustyössä toteutettavan ohjelman tuntemat komennot ovat sekoitus yleisimmistä käyttöjärjestelmistä tuttuja komentoja. Pääpaino on kuitenkin Unixin kaltaisissa käyttöjärjestelmissä (*nix). Tästä syytä hakemistojen nimet erotetaan toisistaan kenoviivamerkillä ('/') ja tulkki on merkkikokoriippuvainen eli pienet ja suuret kirjaimet katsotaan eri merkeiksi.

Tulkin komentojen toiminnallisuutta on rajattu paljon, jotta tehtävä ei kävisi liian vaikeaksi, vaikka ohjelman käytettävyys kärsiikin rajoitteista. Komentoja voi kohdistaa lähinnä nykyhakemistoon, komennoilla ei ole niiden toimintaan vaikuttavia valintoja ja vain yksi komennoista toimii rekursiivisesti.

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

Harjoitustyö on pakollinen ja keskeisin osa kurssia. Harjoitustyön voi tehdä valintansa mukaan laajassa tai suppeassa muodossa. Laaja työ sisältää kopiointikomennon *cp* (luku 2.9) ja rekursiivisesti listaavan *find*-komennon (luku 2.10).

Kurssin arvosanaan (luku 7) vaikuttavat niin työn laajuus, testaustulokset kuin aikataulun noudattaminen.

2. Ohjelman toiminnot

Ohjelmassa on oltava toiminnot, jotka mahdollistavat uuden hakemiston ja tiedoston luomisen, hakemiston vaihdon, nykyisen hakemiston tietojen listaamisen eritavoin, tiedoston tai hakemiston poiston, tiedoston tai hakemiston uudelleen nimeämisen ja ohjelman lopettamisen. Harjoitustyön laajassa muodossa ohjelmassa on lisäksi toiminnot tietojen kopiointiin ja hakemiston rekursiiviseen listaukseen.

Seuraavassa esitellään ohjelman toiminnallisuutta pienten esimerkkien avulla. Laajempia esimerkkiajoja julkaistaan kurssin kotisivujen *Harjoitustyö*-kohdassa.

2.1. Ohjelman käynnistäminen

Ohjelma luo käynnistymisen yhteydessä pelkästään juurihakemiston sisältävän hakemistopuun. Juurihakemistoa ei luoda käyttäjän toimesta, vaan ohjelma on valmis ottamaan vastaan komentoja juurihakemistossa heti käynnistyttyään (kuva 3).

2.2. Käyttöliittymä

Ohjelmalla on oikean komentotulkin tapainen tekstipohjainen käyttöliittymä. Ohjelman käynnistyessä tulostetaan kuvassa 3 esitetyllä tavalla ensin tervehdysteksti "Welcome to SOS." ja omalle rivilleen komentotulkin kehote. Tämän jälkeen jäädään odottamaan käyttäjän syötettä samalle riville välittömästi suurempi kuin merkin jälkeen.

Kehote koostuu hakemistopolusta, jossa hakemistojen nimet on erotettu toisistaan kauttaviivalla. Kehote alkaa kauttaviivalla ja päättyy kauttaviivaan ja suurempi kuin merkkiin paitsi, jos ollaan juurihakemistossa, jolloin kauttaviivoja on yksi.

Kehote on aluksi "/>", koska tulkki käynnistyy juurihakemistoon.

```
Welcome to SOS.
/>
```

Kuva 3: Ohjelma käynnistyksen jälkeen.

Ohjelmaa käytetään komentoikkunan tapaan kirjoittamalla halutun toiminnon käynnistävä komento ja painamalla *Enter*-näppäintä. Komento ja sen mahdolliset parametrit erotetaan toisistaan yhdellä välilyönnillä. Kaikkien muiden kuin lopetuskomennon jälkeen tulostetaan kehote riviä vaihtamatta ja jäädään odottamaan uutta komentoa välittömästi kehotteen viimeisen merkin jälkeen, jos komento oli oikeellinen.

Ohjelma vastaa virheelliseen komentoon tulostamalla omalle rivilleen "Error!" ennen uuden komennon lukemista. Virheeksi katsotaan tuntematon komento (kuva 4) tai tunnettu komento virheellisellä parametrilla. Tulkki sallii tiedostojen ja hakemistojen nimissä vain merkit väleiltä a–z, A–Z ja 0–9 sekä alaviivamerkin ('_') ja pisteen. Nykyhakemistossa ei voi olla samannimistä tiedostoa ja hakemistoa. Tiedoston nimessä ei ole pakko olla siitä pisteellä erotettua päätettä. Nimi ei

saa koostua pelkistä pisteistä, mutta *cd*-komennon parametriksi voidaan antaa kaksi pistettä, joilla ilmaistaan kohdehakemiston olevan nykyhakemiston ylihakemisto (luku 2.4).

Osassa komennoista on virhe, jos parametrina annetun nimistä tietoa (tiedosto tai hakemisto) ei ole olemassa ja osassa komentoja virheen aiheuttaa olemassa oleva tieto. Esimerkiksi olemattomaan alihakemistoon ei voida siirtyä ja toisaalta alihakemistoa ei voida luoda, jos sellainen on jo olemassa. Komento voi epäonnistua myös molemmista syistä. Tietoa uudelleen nimettäessä on mahdollista, että vanha nimi on kirjoitettu väärin, jolloin nimettävää tietoa ei löydetä. Uudelleen nimettäessä voidaan myös havaita nykyhakemistossa olevan jo tieto kohdenimellä. Seuraavissa luvuissa kerrotaan mitä kukin komento vaatii parametrina saamiensa tietojen olemassaolon osalta.

Tunnetulla komennolla voi olla myös väärä määrä parametreja. Ylimääräiset välilyönnit ennen komentoa, komennon jälkeen, komennon ja sen parametrin välissä tai komennon parametrien välissä katsotaan virheeksi.

```
/>pwd
Error!
/>
```

Kuva 4: Tuntemattoman komennon aiheuttama virhe.

Kaikkien tulosteiden muoto on kiinnitetty, jotta ohjelmien automaattinen, tulosteiden vertailuun perustuva tarkistus olisi mahdollista (luku 4). Tarkistusta automatisoimalla pyritään siihen, että kurssin opettajille jää enemmän aikaa työn rakenteen ja laadun arviointiin.

Komennot pitää lukea *In*-apuluokan avulla, jotta ohjelmien tehtävänannon mukainen käyttäytymisen olisi todennäköisempää. Harjoitustyössä käytetään nimiseen pakkaukseen (luku 3.7) sijoitettua versiota *In*-luokasta, joka julkaistaan kurssin kotisivujen *Koodit*-kohdassa ja GitLab-palvelussa. Tässä pakkauksessa ovat myös opettajan antamat rajapinnat.

2.3. Hakemiston luominen

Nykyiseen hakemistoon luodaan uusi alihakemisto *md*-komennolla. Alihakemisto annetaan komennolle parametrina. Parametri on virheellinen, jos nykyhakemistossa on jo samanniminen alihakemisto tai tiedosto. Kuvassa 5 luodaan uusi hakemisto.

```
/>md pics
/>
```

Kuva 5: Luodaan juurihakemistoon *pics*-niminen alihakemisto.

2.4. Hakemiston vaihtaminen

Hakemistoa vaihdetaan *cd*-komennolla. Alihakemistoon siirryttäessä komennon parametriksi annetaan alihakemiston nimi. Ylihakemistoon siirrytään antamalla hakemiston nimeksi kaksi pistettä. Hakemistoa voidaan vaihtaa vain nykyhake-

miston välittömään yli- tai alihakemistoon. Näin hakemistopuussa voidaan liikkua yksi taso kerrallaan ylös tai alas. Tähän on poikkeuksena parametriton *cd*-komento, joka vaihtaa hakemistoksi juurihakemiston.

Parametri on virheellinen, jos alihakemistoa ei löydetä, parametriksi on annettu tiedoston nimi tai jos parametriksi on annettu kahden tai useamman hakemiston polku (esimerkiksi "../.." tai "pics/cats"). Virhe tapahtuu myös, jos juurihakemistossa yritetään siirtyä ylihakemistoon.

Hakemiston vaihdon onnistuessa komentokehotteen polku päivitetään. Ylihakemistoon siirryttäessä komentokehotteesta poistetaan nykyhakemisto. Alihakemistoon siirryttäessä kehotteeseen lisätään kohdehakemisto. Kuvassa 6 siirrytään ensin juuresta alihakemistoon *pics* ja sitten uuteen alihakemistoon *cats*.

```
/>cd pics
/pics/>md cats
/pics/>cd cats
/pics/cats/>
```

Kuva 6: Vaihdetaan juurihakemistosta *pics*-alihakemistoon. Luodaan uusi alihakemisto *cats* ja siirrytään tähän hakemistoon.

2.5. Tiedoston luominen

Tulkissa on poikkeuksellisesti komento tiedoston luomiseen nykyiseen hakemistoon (*mf*), koska simulaattorissa ei voi suorittaa esimerkiksi tekstieditoria. Komennolla ei voi luoda tiedostoja yli- tai alihakemistoihin. Komennolle annetaan parametreiksi tiedoston nimi ja tiedoston koko tavuina (≥ 0). Komento ei voi luoda tiedostoa, joka on jo hakemistossa tai jonka nimi on sama kuin alihakemiston nimi. Tiedoston luominen epäonnistuu myös, jos annettu nimi on virheellinen tai koko ei ole positiivinen kokonaisluku (≥ 0). Kuvassa 7 nykyiseen hakemistoon luodaan kaksi kuvatiedostoa.

```
/pics/cats/>mf ceiling_cat.gif 3677
/pics/cats/>mf dangerous_kiten.jpg 13432
/pics/cats/>
```

Kuva 7: Luodaan kaksi tiedostoa.

2.6. Uudelleen nimeäminen

Nykyhakemiston sisältämän tiedoston tai hakemiston uudelleen nimeäminen tapahtuu *mv*-komennolla, jonka ensimmäinen parametri on komennon kohteen nimi ja toinen on kohteen uusi nimi. Yli- tai alihakemistossa olevan tiedon nimeä ei voi vaihtaa. Nimen vaihto onnistuu, kun nykyisessä hakemistossa on ensimmäisellä parametrilla määritellyn niminen tiedosto tai alihakemisto ja toisena parametrina annettu nimi on oikeellinen eikä nimeä vastaavaa tiedostoa tai alihakemistoa vielä ole. Kuvassa 8 korjataan tiedoston nimeen tullut kirjoitusvirhe.

```
/pics/cats/>mv dangerous_kiten.jpg dangerous_kitten.jpg
/pics/cats/>
```

Kuva 8: Nimetään tiedosto uudelleen.

2.7. Hakemiston listaaminen

Nykyhakemiston listaamiseen käytetään *ls*-komentoa. Listattavaksi voidaan antaa yksittäinen tiedosto tai hakemisto. Listattavaksi voidaan määritellä myös useampia tietoja yksi tai kaksi jokerimerkkiä (asteriski) sisältävän ilmauksen avulla (luku 3.6). Ohjelma listaa nykyhakemiston koko sisällön, kun komennolle ei anneta parametreja tai parametri on pelkkä jokerimerkki. Listaus ei tulosta alihakemiston sisältöä. Listausta ei voi kohdistaa ylihakemistoon. Rekursiiviselle listaukselle on oma komento (luku 2.10).

Komento tulostaa löytämänsä tiedot nimen mukaisessa nousevassa aakkosjärjestyksessä. Komento ei järjestä hakemistoa, koska sen tiedot ovat aina oikeassa järjestyksessä (luku 3.4). Listauksessa tulostetaan kunkin tiedoston ja alihakemiston merkkijonoesitys (luku 3.3) omalle rivilleen.

Ohjelma tulostaa virheilmoituksen, jos nykyhakemistossa ei ole tietoa annetulla nimellä tai yksikään tieto ei vastaa annettua ilmausta. Tyhjään hakemistoon kohdistetun listauskomennon seurauksena ei tulosteta mitään, kun komennolla ei ole parametria tai parametri on pelkkä jokerimerkki. Muut tyhjän hakemiston listaukset tuottavat virheilmoituksen.

Kuvassa 9 tulostetaan kissakuvat sisältävän nykyhakemiston sisältö ja "jpg"-päätteiset tiedostot. Listauksista nähdään edellisessä kuvassa annetun komennon vaihtaneen tiedoston nimen. Listauksen jälkeen vaihdetaan *cd*-komennolla ylihakemistoon *pics*, luodaan hakemisto *dogs* ja siirrytään siihen. Uuden tiedoston luominen ei onnistu, koska heittomerkki ei kuulu nimissä sallittuihin merkkeihin (luku 2.2). Toinen luontiyritys onnistuu. Ilman parametria annettu *cd*-komento palauttaa tulkin juurihakemistoon, joka listaamalla nähdään alihakemistolla *pics* olevan kaksi tietoa.

```
/pics/cats/>ls
ceiling_cat.gif 3677
dangerous_kitten.jpg 13432
/pics/cats/>ls *.jpg
dangerous_kitten.jpg 13432
/pics/cats/>cd ..
/pics/>md dogs
/pics/>cd dogs
/pics/dogs/>mf world's_ugliest_dog.jpg 118088
Error!
/pics/dogs/>mf worlds_ugliest_dog.jpg 118088
/pics/dogs/>cd
/>ls
pics/ 2
/>
```

Kuva 9: Hakemiston listaus ja aiemmin esiteltyjä komentoja.

2.8. Poistaminen

Tiedosto tai hakemisto poistetaan nykyhakemistosta *rm*-komennolla. Poistettavaksi voidaan määritellä myös useita tietoja jokerimerkkejä sisältävän ilmauksen avulla (luku 3.6). Pelkästä jokerimerkkistä koostuvalla ilmauksella ohjelma tuho-

aa kaikki nykyhakemiston tiedot. Poistamista ei varmisteta käyttäjältä. Poisto epäonnistuu, jos parametrina annetulla nimellä tai ilmauksella ei löydetä poistettavaa tai hakemisto on tyhjä.

Kuvassa 10 luodaan uusi hakemisto, siirrytään siihen ja luodaan kaksi väliaikaista tiedostoa, joka poistetaan heti listauksen jälkeen yhdellä komennolla jokerimerkkien avulla.

```
/>md temp
/>cd temp
/temp/>mf temp_file.txt 0
/temp/>mf another_temp_file 1
/temp/>ls
another_temp_file 1
temp_file.txt 0
/temp/>rm *file*
/temp/>
```

Kuva 10: Tiedoston poisto.

2.9. Tiedoston kopiointi

Nykyhakemiston tiedostoja kopioiva *cp*-komento on valinnainen toiminto, jonka on oltava harjoitustyön laajassa versiossa. Hakemiston kopiointi katsotaan virheeksi, koska tehtävää on helpotettu rajaamalla kopiointi vain tiedostoihin. Komennon ensimmäinen parametri määrittelee kopioinnin lähteen (alla *x*) ja toinen parametri kopioinnin kohteen (alla *y*). Jo olemassa olevaa kohdetta ei voida korvata. Kopioitaessa *x*:stä tehdään syväkopio (luku 3.2). Kopiointikomentoa voidaan käyttää usealla eri tavalla. Alla oletetaan, että *x* on olemassa nykyhakemistossa.

a) Tiedoston *x* kopiointi nykyhakemistoon uudella nimellä *y*. Kopiointi onnistuu, jos *y* on oikeellinen tiedoston nimi eikä hakemistossa ole vielä tiedostoa nimellä *y*.

b) Tiedoston *x* kopiointi nykyhakemiston välittömään alihakemistoon tai ylihakemistoon *y*. Kopiointi onnistuu, jos *y* on olemassa eikä siinä ole vielä tiedostoa *x*:n nimellä.

c) Yhden tai kahden jokerimerkin avulla muodostettua ilmausta (luku 3.6) vastaavien tiedostojen x_i kopiointi nykyhakemiston välittömään alihakemistoon tai ylihakemistoon *y*. Kopiointi onnistuu, jos *y* on olemassa eikä hakemistossa ole vielä tiedostoa minkään kopioitavan tiedoston x_i:n nimellä. Hakemiston *y* sisältö jää entiselleen, jos kopiointi epäonnistuu.

Kuvassa 11 luodaan uusi tiedosto, tehdään siitä kopio *cp*-komennolla ja tarkistetaan tulos listaamalla. Sitten luodaan uusi hakemisto ja kopioidaan sinne alkuperäinen ja kopioitu tiedosto. Alkuperäistä tiedostoa ei voida enää kopioida uudelleen *backup*-hakemistoon, koska siellä on jo samanniminen kopio. Listauskomento näyttää, että uudessa hakemistossa on kaksi tiedostoa, jotka havaitaan oikeiksi hakemistoon siirtymällä ja listaamalla. Lopuksi siirrytään takaisin *temp*-hakemistoon ja poistetaan tietoja siten, että jäljelle jää vain ensimmäinen kopio ja palataan juureen.

```
/temp/>mf memo.txt 42
/temp/>cp memo.txt memo_backup.txt
/temp/>ls
memo.txt 42
memo_backup.txt 42
/temp/>md backup
/temp/>cp *.txt backup
/temp/>cp memo.txt backup
Error!
/temp/>ls
backup/ 2
memo.txt 42
memo_backup.txt 42
/temp/>cd backup
/temp/backup/>ls
memo.txt 42
memo_backup.txt 42
/temp/backup/>cd ..
/temp/>rm memo.txt
/temp/>rm backup
/temp/>ls
memo_backup.txt 42
/temp/>cd
/>
```

2.10. Rekursiivinen listaaminen

Rekursiivinen listaus on valinnainen toiminto ja tehdään *find*-komennolla. Komento listaa nykyhakemiston alihakemistoineen esijärjestyksessä (luku 3.5). Kuvassa 12 listataan tiedostojärjestelmän koko sisältö antamalla *find*-komento juurihakemistossa.

```
/>find
pics/ 2
cats/ 2
ceiling_cat.gif 3677
dangerous_kitten.jpg 13432
dogs/ 1
worlds_ugliest_dog.jpg 118088
temp/ 1
memo_backup.txt 42
/>
```

Kuva 12: Juurihakemistosta aloitettu rekursiivinen listaus.

2.11. Ohjelman lopetus

Ohjelmasta poistutaan parametrittomalla *exit*-komennolla (kuva 13). Komennon jälkeen tulostetaan riviä vaihtaen "Shell terminated."-teksti.

```
/>exit
Shell terminated.
```

Kuva 13: Ohjelman lopetus.

3. Ohjelman rakenne

3.1. Yleistä

Ohjelma on jaettava kapseloituihin luokkiin, joiden sisältö puolestaan jaetaan järkevän mittaisiin metodeihin. Luokissa on vältettävä saman koodin kopioimista eri paikkoihin, koska koodista voi kirjoittaa asianomaisista paikoista kutsuttavan metodin. Metodien korvaaminen, kuormittaminen ja monimuotoisuus parametrien välityksessä vähentävät tarvetta toistaa koodia eri metodien sisällä.

Luokat rakennetaan tiedon – ei toimintojen – ympärille. Luokkametodeista koostuvat, esimerkiksi *Math*-luokan tapaiset apuluokat ovat poikkeus tähän sääntöön, mutta tällaisia luokkia saa tehdä vain erittäin hyvin perustellusta syystä. Ohjelman toimintoja vastaavia luokkia ei saa tehdä, koska toiminnot eivät yleensä 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 tulkin logiikasta vastaavaa koodia ei saa kirjoittaa sinne, vaan tulkille on tehtävä oma *Tulkki*-niminen luokka.

Muista noudattaa hyvää ohjelmointitapaa: sisennä koodia johdonmukaisesti luettavuuden parantamiseksi, kommentoi riittävästi ja oikeissa paikoissa, liitä jokaiseen metodiin yleisluonteinen kommentti, nimeä vakiot, muuttujat ja metodit järkevästi, käytä vakioita, pidä rivit riittävän lyhyinä, käytä välejä lauseiden sisällä, erota loogiset kokonaisuudet toisistaan väliriveillä, pidä operaatiot järkevän mittaisia – operaation tulisi mahtua yhdelle A4-kokoiselle sivulle ja varaudu metodeissa virheellisiin parametrien arvoihin [4, 5].

Erityisesti olio-ohjelmointiin liittyviä hyviä tapoja ovat luokkiin liitetyt yleiset kommentit, attribuuttien kommentointi, attribuuttien harkittu käyttö [6] ja luokkien tai rajapintojen sijoittaminen omiin tiedostoihinsa.

Sisennä koodi välilyönnein. WETO ei hyväksy muilla tavoilla sisennettyä lähdekoodia, jotta koodi näkyisi opettajan editorissa täsmälleen opiskelijan aikomalla tavalla.

Lue tiedot näppäimistöltä pakatulla *In*-luokalla (luku 3.7).

3.2. Luokkahierarkia

Harjoitustyössä periytymistä käytetään erityisesti tiedostojärjestelmän sisällön mallintamiseen. Ohjelmassa tulee olla abstrakti yliluokka *Tieto* ja siitä perityt konkreettiset *Tiedosto*- ja *Hakemisto*-luokat. Luokkien tiedot kätketään **private**-määreellä ja aksessorit nimetään harjoitusten tapaan attribuutin nimeä kuormittaen.

Yliluokassa on *StringBuilder*-tyyppinen *nimi*-attribuutti tiedon nimelle. Luokalla on parametriton ja yksiparametrinen rakentaja. Parametrittomassa rakentajassa attribuuttiin liitetään *StringBuilder*-olio, jonka sisältää tyhjän merkkijonon "". Nimen asettava metodi ja parametrillinen rakentaja heittävät *IllegalArgumentException*-poikkeuksen, jos parametrina annettu nimi on **null**-arvoinen tai muodoltaan virheellinen (luku 2.2).

Tiedoston oma attribuutti *koko* kertoo tiedoston koon tavuina. Koko on **int**-tyyppinen ja sen tulee olla positiivinen (≥ 0). Tee luokalle oletusrakentaja, jossa kooksi asetetaan nolla. Tee myös kaksiparametrinen rakentaja, joka heittää *IllegalArgumentException*-poikkeuksen, jos parametrina annettu nimi tai koko on virheellinen. Rakentajan ensimmäinen parametri on nimi. Kutsu parametrillisessa rakentajassa yliluokan yksiparametrista rakentajaa. Heitä asetusmetodista *IllegalArgumentException*-poikkeus, jos parametri on virheellinen.

Hakemistolla on lista, josta on viitteet hakemiston sisältämiin tiedostoihin ja muihin hakemistoihin (luku 3.5). Lista-attribuutti on *OmaLista*<*Tieto*>-tyyppinen (luku 3.4) ja nimeltään *sisalto*. Hakemiston sisältö on järjestetty aina nimiensä mukaiseen nousevaan aakkosjärjestykseen. Listan lisäksi hakemistolla on *Hakemisto*-tyyppinen attribuutti *ylihakemisto*, joka viittaa hakemiston ylihakemistoon. Juurihakemiston ylihakemistoviite on **null**-arvoinen.

Kirjoita hakemistolle parametriton rakentaja, jossa *sisalto*-attribuuttiin liitetään tyhjä *OmaLista*<*Tieto*>-tyyppinen listaolio ja ylihakemiston viitteelle asetetaan **null**-arvo. Parametrillisella rakentajalla on *StringBuilder*-tyyppinen parametri nimelle (ensimmäinen parametri) ja *Hakemisto*-tyyppinen parametri ylihakemistolle. Kutsu rakentajassa yliluokan yksiparametrista rakentajaa. Hakemiston rakentaja heittää *IllegalArgumentException*-poikkeuksen, jos nimessä on virhe. Rakentajan jälkimmäinen parametri, samoin kuin ylihakemiston asettavan metodin parametri, saa olla **null**-arvoinen. Tässäkin rakentajassa luodaan tyhjä listaolio ja liitetään siihen attribuutti.

Aliluokkien parametrillisissa rakentajissa ei tulisi toistaa yliluokassa ja aliluokkien asettavissa aksesoreissa olevia virhetarkisteluja. Yliluokan rakentajan kutsu riittää nimiparametrin tarkistukseen ja poikkeuksen heittämiseen. Samoin aliluokan oman attribuutin asettavan metodin kutsumisella saadaan aikaiseksi kaikki tarvittava virheenkäsittelyyn liittyen. Aliluokkien rakentajissa ei ole tarpeen käyttää **try**-**catch**-lauseita, koska rakentajista kutsuttavat yliluokan rakentajat ja aliluokan omat asetusmetodit heittävät pyydetyn poikkeuksen, jolloin riittää, että poikkeuksen annetaan vain lähteä rakentajasta pois.

Luokkahierarkiassa on toteutettava kolme neljästä opettajan valmiina antamasta rajapinnasta (luku 3.7). *Tieto*-luokka toteuttaa *Tietoinen*-rajapinnan, jonka avulla tiedon nimeä voidaan verrata toisen tiedon nimeen tai mahdollisesti jokerimerkkejä sisältävään (luku 3.6). *Tiedosto*-luokan toteuttama *Syvakopioituva*<*T*>-rajapinta määrittelee kuinka tiedostot syväkopioidaan. *Hakemisto*-luokassa toteutettavat *Sailova*<*T*>-rajapinnan metodit mahdollistavat hakemistosta haun, hakemistoon lisäämiseen ja hakemistosta poiston muissa rajapinnoissa toteutettuihin metodeihin tukeutuen. Rajapinnoista ja niiden metodeista on kerrottu lisää kommenteissa.

Sijoita *Tieto*-, *Tiedosto*- ja *Hakemisto*-luokat omaa pakkaukseensa (luku 3.7).

3.3. Tietojen merkkijonoesitykset ja vertailu

Korvaa *Object*-luokan *toString*-metodi luokkahierarkian jokaisella tasolla. Tee korvaus "ketjuttaen" isten, että kutsut aliluokkien *toString*-metodin korvauksissa yliluokan versiota **super**-attribuutin avulla juuriluokka pois lukien, koska *Object*-luokan *toString*-metodia ei pidä kutsua tässä tehtävässä.

Tieto-luokka palauttaa nimensä *String*-muodossa. *Tiedosto*-luokan *toString*-metodi lisää yliluokassa korvatun metodin palauttamaan merkkijonoon välilyönnin ja tiedoston kokoon. Jos tiedosto-olion tiedot ovat esimerkiksi "ceiling_cat.gif" ja 3677, on metodin paluuarvo "ceiling_cat.gif 3677". *Hakemisto*-luokan *toString*-metodin palauttama merkkijono koostuu yliluokan metodin antamaan merkkijonoon tiedostoerottimen, välilyönnin ja kokonaisluvusta, joka kertoo montako tiedostoja ja alihakemistoa hakemisto sisältää. Jos hakemisto-olion nimi on "pics" ja hakemiston sisältö koostuu kahdesta alihakemistosta, on metodin palauttama merkkijonoesitys "pics/ 2". *Tiedosto*- ja *Hakemisto*-luokkien merkkijonoesitykset tulostetaan, kun hakemisto listataan.

Tietoja vertaillaan vain niiden nimien avulla. Korvaa siksi *Object*-luokan *equals*-metodi vain *Tieto*-luokassa. Tiedot katsotaan samoiksi, jos niiden nimet ovat merkilleen samat. Pienet ja suuret kirjaimet katsotaan eri merkeiksi. Toteuta myös *Comparable*-rajapinnan *compareTo*-metodi *Tieto*-luokassa siten, että se vertaa nimiä. Kiinnitä geneeriseksi tyypiksi *Tieto*.

Älä tee kumpaankaan metodiin omaa vertailualgoritmia nimille, vaan hyödynnä tyyppimuunnoksen kautta käytettävissä olevaa *String*-luokan valmista koodia. (*StringBuilder*-luokka ei korvaa itse *equals*-metodia eikä toteuta *Comparable*-rajapintaa.) Joudut tekemään pienen muutoksen *compareTo*-metodin paluuarvolle, koska *String*-luokassa korvattu metodi palauttaa negatiivisen arvon, nollan tai nollaa suuremman arvon ja *Tieto*-luokan korvauksen tulee palauttaa jokin arvoista -1, 0 tai 1.

3.4. OmaLista-luokka ja muut tietorakenteet

Harjoitustyössä tutustutaan linkitettyyn listarakenteeseen perimällä Javan *LinkedList<E>*-luokasta geneerinen *OmaLista<E>*-luokka. Listaan on kirjoitettava ainakin opettajan antamassa *Ooperoiva<E>*-rajapinnassa (3.7) määritellyt uudet listaoperaatiot. Yksi rajapinnan operaatioista lisää alkion listalle siten, että alkio sijoittuu kaikkien itseään pienempien tai yhtä suurien alkioiden jälkeen ja ennen kaikkia itseään suurempia alkioita. Alkioita vertaillaan lisättäessä *Comparable*-rajapinnan *compareTo*-metodilla (luku 3.3), jolloin tiedostoja ja hakemistoja sisältävä lista järjestyy automaattisesti nousevaan aakkosjärjestykseen. Voit halutessasi tehdä muitakin kuin rajapinnan määrittelemiä listaoperaatioita.

Omaa listaa käytetään ohjelman pääasiallisena tietorakenteena. Luokan tärkein käyttökohde on hakemiston sisällön säilöminen (luvut 3.2 ja 3.5).

Operaatioiden tulee olla "puhtaita" yleiskäyttöisiä listaoperaatioita, joita voitaisiin käyttää myös muissa tehtävissä. *OmaLista*-luokassa ei saa siten esiintyä operaatiot harjoitustyön sitovia tunnuksia. Erityisesti nimiä *Tieto*, *Tiedosto*, *Hakemisto*, *String* tai *StringBuilder* ei saa käyttää oman listaluokan operaatioissa.

3.5. Hakemistopuu

Harjoitustyössä tiedostojärjestelmä näyttäytyy hakemistopuuna (luku 1.1), joka koostuu keskinäisillä viitteillä yhdistetyistä *Hakemisto*-luokan olioista. Oliot muistuttavat jossain määrin yhteen suuntaan linkitetyn listan solmuja, joista kustakin on viite seuraavaan solmuun. Pääasiallinen ero listan solmuihin on se, että hakemistosta voi lähteä useita viitteitä seuraavin hakemistoihin (alihakemistot) ja että hakemistosta on myös viite edelliseen hakemistoon (ylihakemisto). Yli- ja alihakemiston välisten viitteiden avulla puussa voidaan liikkua ylös ja alas viitteitä seuraamalla.

Tehtävää on helpotettu siten, että tiedoston ei tarvitse tuntea hakemistoaan. Hakemisto viittaa tiedostoonsa, mutta tiedostolla ei ole takaisinviitettä hakemistoonsa.

Hakemisto-luokka säilöö tiedostoihinsa ja alihakemistoihinsa liittyvät viitteet *Omalista*<*Tieto*>-tyyppiselle listalle (luku 3.4), joka on luokan attribuutti. Hakemisto-oliolla on näin osaolio, joka luomisesta ja hallinnoinnista hakemisto on vastuussa. Hakemistoilla, samoin kuin tiedostoilla, on myös *StringBuilder*-tyyppinen osaolio tiedon nimelle.

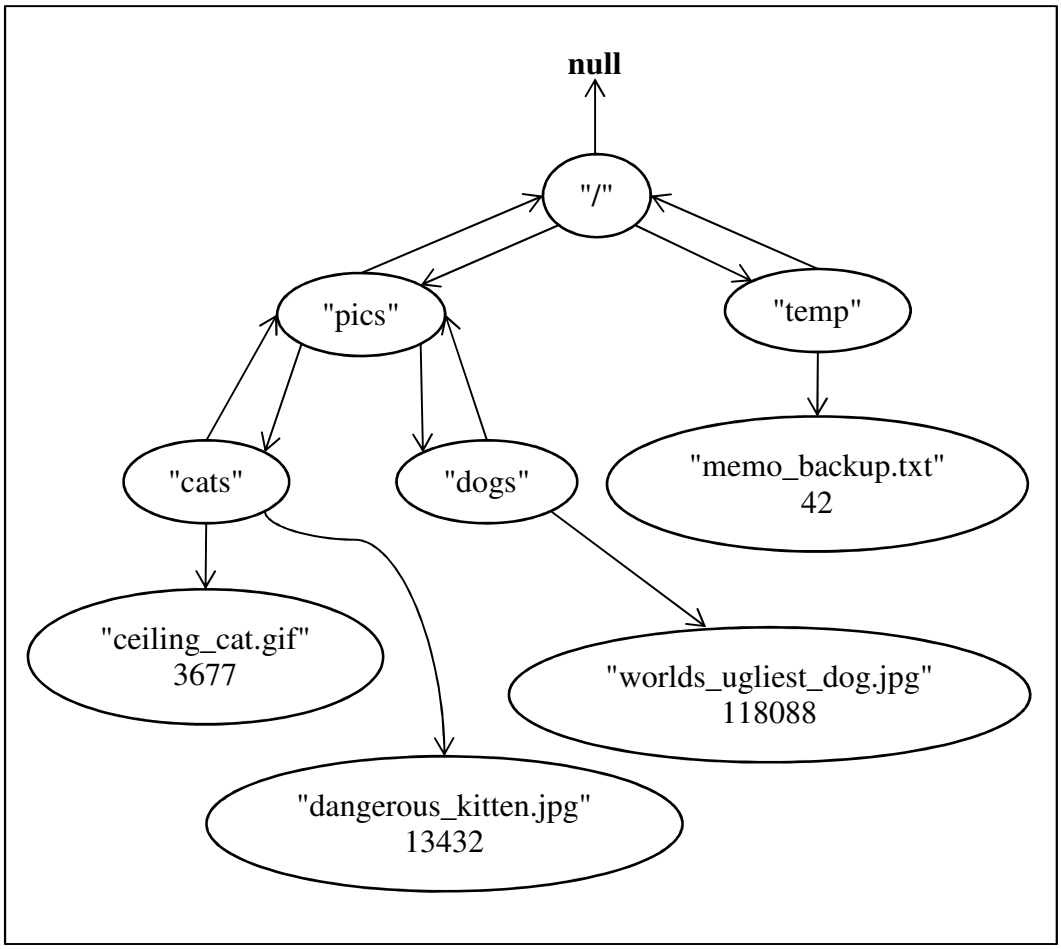
Kuvassa 1 annettu hakemistopuu on piirretty olioiden ja viitteiden muodostamana rakenteena kuvassa 14. Kuvaan ei ole piirretty näkyviin hakemisto-olioiden osaolioita. Juurihakemiston ylihakemistoviite on **null**-arvoinen. Kuvassa 15 on annettu *pic*-niminen hakemisto-olio osaolioineen ja tähän hakemistoon liittyvät hakemisto-oliot.

Puu on rakennettu luomalla ensin automaattisesti juurihakemisto ja liitämällä sitten sen alle käyttäjän antamien *md*-komentojen mukaan muut hakemistot joko suoriksi tai epäsuoriksi alihakemistoiksi.

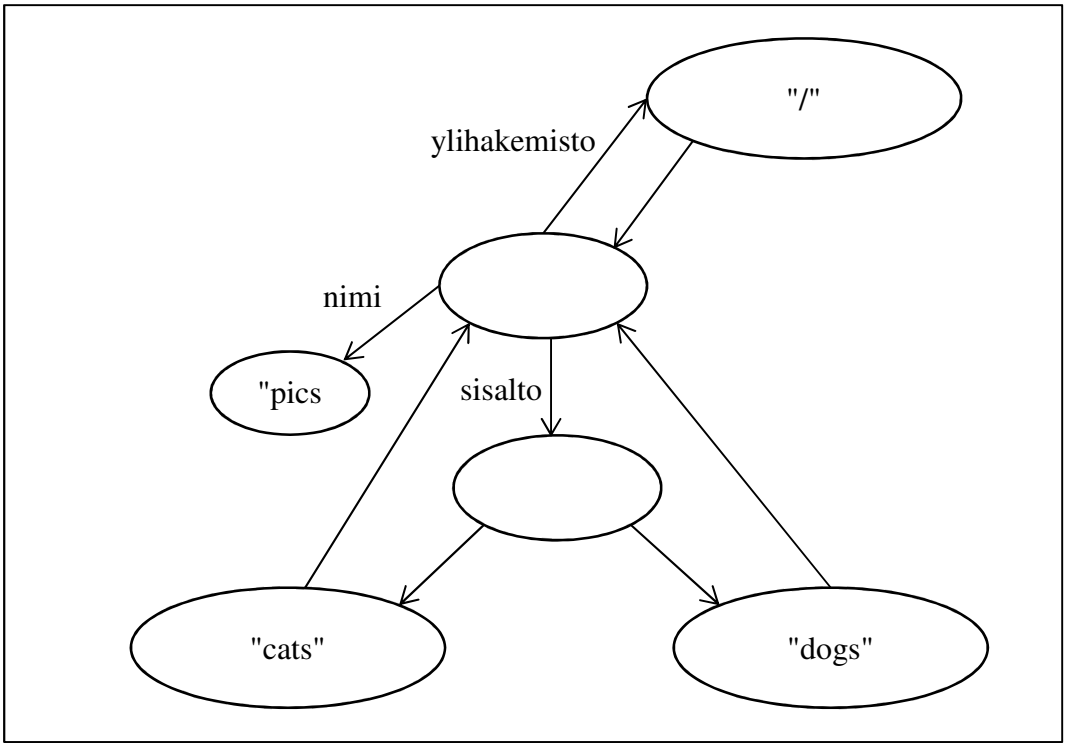
Kunkin alihakemiston lisääminen on tehty kahdessa vaiheessa. Ensin alihakemistosta on asetettu viite ylihakemistoon. Sitten ylihakemiston listalle on lisätty viite alihakemistoon. Liitos alihakemistosta ylihakemistoon muodostetaan antamalla alihakemistoa luotaessa rakentajalle viite ylihakemistoon. Rakentaja kutsuu puolestaan asetusmetodia:

```
public void ylihakemisto(Hakemisto ylla) {
   ylihakemisto = ylla;
}
```

joka kiinnittää alihakemisto-olion *ylihakemisto*-attribuutin ylihakemisto-olioon. Päinvastainen liitos saadaan aikaiseksi lisäämällä ylihakemisto-olion listalle alihakemisto-olioon liittyvä viite. Tähän käytetään *Hakemisto*-luokassa toteutettua *Sailova*<*T*>-rajapinnan *lisaa*-metodia.



Kuva 14: Hakemistopuu viitteillä yhdistettyinä olioina.



Kuva 15: *Pics*-niminen hakemisto-olio, sen osaoliot ja olioiden ja ympäristö.

Alla on esitetty kuinka kuvan 14 hakemistorakenne voidaan luoda manuaalisesti eli ilman *Tulkki*-luokan metodeja *Hakemisto*-luokan rakentajan ja luokan toteuttaman metodin avulla.

```
// Luodaan juurihakemisto.
Hakemisto juuri = new Hakemisto();
// Luodaan juuren alihakemistot ja liitetään ne juureen.
Hakemisto kuvat = new Hakemisto(new StringBuilder("pics"), juuri);
Hakemisto temppi = new Hakemisto(new StringBuilder("temp"), juuri);
// Liitetään juuri alihakemistoihinsa.
juuri.lisaa(kuvat);
juuri.lisaa(temppi);
// Luodaan kuvahakemiston alihakemistot ja liitetään ne
// kuvahakemistoon.
Hakemisto koirat = new Hakemisto(new StringBuilder("dogs"), kuvat);
Hakemisto kissat = new Hakemisto(new StringBuilder("cats"), kuvat);
// Liitetään kuvahakemisto alihakemistoihinsa.
kuvat.lisaa(kissat);
kuvat.lisaa(koirat);
```

Harjoitustyössä on valinnainen *find*-toiminto hakemistopuun sisällön rekursiiviseen tulostukseen nykyhakemistosta alkaen (luku 2.10). Tulostus tapahtuu esijärjestyksessä (preorder). Hakemistopuussa tämä tarkoittaa sitä, että ylihakemisto tulostetaan ennen alihakemistojaan.

Alla on annettu algoritmi, joka kulkee puun läpi esijärjestyksessä ja lisää samalla parametrilistalle viitteet puun tietoihin. Algoritmissa on hahmoteltu rekursiivisen käsittelyn keskeisin idea. Algoritmin voi toteuttaa myös siten, että tietojen viitteet sisältävä lista palautetaan paluuarvona.

```
Algoritmi esijärjestys(Hakemisto hakemisto, Lista tiedot) {
   // Asetetaan apuviite hakemiston tiedot säilövään listaan.
   sisältö ← hakemisto.sisältö();
   // Käydään hakemiston sisältö läpi yksi tieto kerrallaan.
   ind ← 0;
   while (ind < sisältö.koko()) {
      // Asetetaan viite hakemiston tietoon.
      tieto ← sisältö.alkio(i);
      // Lisätään hakemiston tieto listalle.
      tiedot.lisaaLoppuun(tieto);
      // Lisätään alihakemiston tiedot listalle rekursiivisesti.
      if (tieto on hakemisto) {
         esijärjestys(tieto, tiedot);
      }
      ind++;
   }
}
```

Tee *Hakemisto*-luokalle *Iterator<E>*-rajapinnan toteuttava *HakemistoIteraattori<E>*-iteraattoriluokka, jos päätät toteuttaa *find*-komennon. Toteuta *Hakemisto*-luokassa *Iterable*-rajapinnan *iterator*-metodi siten, että metodi palauttaa viitteen *HakemistoIteraattori<E>*-tyyppiseen olioon. Kiinnitä tyyppiparametriksi *Tieto*-tyyppi. Käytä esijärjestyksen luovaa algoritmia apuna hakemiston iteraattoriluokassa siten, että tuotat ensin algoritmilla esijärjestyksessä olevan viitteiden listan ja tulostat sitten listan sisällön iteraattorin avulla. Sijoita *HakemistoIteraattori*-luokka samaan pakkaukseensa (luku 3.7).

3.6. Jokerimerkit

Pakolliset *ls*- ja *rm*-komennot sekä valinnainen *cp*-komento voidaan kohdistaa tietyn tiedoston tai hakemiston lisäksi useisiin tietoihin antamalla komennon parametriksi yhden tai kahden jokerimerkin (asteriski) avulla muodostettu ilmaus. *Cp*-komennossa ilmaus on komennon ensimmäinen parametri.

Tässä työssä jokerimerkki voi aloittaa tai lopettaa ilmauksen ja ilmaus voi myös alkaa jokerimerkillä ja päättyä sellaiseen. Yhdestä jokerimerkistä muodostuva ilmaus on yleisin mahdollinen, sillä sitä vastaa mikä tahansa nimi. Kaikki muunlaiset jokerimerkkejä sisältävät ilmaukset, myös **, katsotaan virheellisiksi. Ilman jokerimerkkiä ilmaus on fraasi eli tiedon nimi sellaisenaan.

Ilmauksen aloittavalla jokerimerkillä valitaan käsiteltäväksi tiedot, joiden nimien loppu vastaa jokerimerkkiä seuraavia merkkejä. Jokerimerkki jättää tässä tapauksessa huomiotta nolla tai useampia ilmaukseen verrattavan nimen merkkejä ennen lopun vertailua. Esimerkiksi ilmausta **a* vastaavat muun muassa nimet *a*, *ba* ja *zebra*. Aloittavaa jokerimerkkiä käytetään usein tietyn päätteen omaavien tietojen valintaan. Esimerkiksi ilmaus **.jpeg* valitsee tiedot, joiden nimissä on nolla tai useampi merkki ennen merkkijonoa ".jpeg". Muun muassa nimet *grumpy_cat.jpeg* ja *cannot_be_unseen.jpeg* vastaavat tätä ilmausta.

Ilmauksen lopussa oleva jokerimerkki valitsee tiedot, joiden nimien alku vastaa ennen jokerimerkkiä annettuja merkkejä. Tässä tapauksessa nimien lopuista jätetään huomiotta nolla tai useampia merkkejä. Esimerkiksi ilmausta *a** vastaavat muun muassa nimet *a*, *ab* ja *ant*.

Ilmauksen alussa ja lopussa olevilla jokerimerkeillä ohitetaan nimeä ilmaukseen verratessa niin nimen alusta kuin lopusta nolla tai useampia merkkejä. Esimerkiksi ilmaus **a** vastaa nimiä, joiden nimissä on ennen ja jälkeen pienen *a*-kirjaimen nolla tai useampia merkkejä. Tällaisia nimiä ovat muun muassa *a*, *ba*, *ab*, *ant*, *cat*, *saab* ja *zebra*.

3.7. Koodin organisointi

Tee luokkahierarkian luokkien lisäksi omat luokat komentojen tulkille (*Tulkki*) ja käyttöliittymälle ja erillinen *main*-metodin sisältävä ajoluokka *Oope2HT*. Ohjelmaan voi lisätä näiden lisäksi muita omia luokkia.

Hakemistopuun hallinnointi on tulkkiluokan vastuulla. Tulkkiluokka toteuttaa ohjelman tuntema komennot hakemistopuuta käsittelevillä metodeilla. Kullekin komennolle on pääsääntöisesti oma metodi, joka hyödyntää mahdollisimman paljon *Hakemisto*-luokassa toteutettuja *Sailova<T>*-rajapinnan metodeja. Esimerkiksi *rm*-komento voidaan toteuttaa *Tulkki*-luokassa metodina, jossa kutsutaan nykyhakemiston olion kautta *Hakemisto*-luokan hakumetodia, jonka jälkeen löydetyt tiedot voidaan poistaa yksi kerrallaan *Hakemisto*-luokan poistometodilla. *Sailova<T>*-rajapinnan metodit puolestaan tukeutuvat muiden rajapintojen toteutettuihin metodeihin.

Älä nimeä *Tulkki*-luokan metodeja suoraan komentojen nimiä käyttäen. Esimerkiksi *rm* on liian lyhyt ja epämääräinen metodin nimeksi.

Tulkkiluokassa on *Hakemisto*-tyyppinen attribuutti hakemistopuun juurelle, jonka kautta voidaan edetä mihin tahansa hakemistoon. Nykyisen hakemiston hakemisto-olioon liittyvä attribuutti helpottaa komentojen toteutusta. Kaikki käyttäjän antamat tiedot luetaan käyttöliittymässä ja käyttöliittymässä tehdään mahdollisimman suuri osa tulostuksista. Huomaa, että *Tulkki*-luokan listaavat metodit voivat palauttaa hakemistolistauksen rivinvaihtoja sisältävänä merkkijonona, jos virheisiin reagoidaan poikkeuksilla.

Pääsilmukan luonteva paikka on käyttöliittymän luokka. Pääsilmukasta kutsutaan komentoja vastaavia tulkkiluokan metodeja *Tulkki*-luokan olion kautta joko suoraan tai apumetodien avulla.

Sijoita kaikki luokkasi *harjoitustyo*-nimiseen pakkaukseen. Ajoluokka on pakkauksen juurihakemistossa. Tee *Tieto*-, *Tiedosto*- ja *Hakemisto*-luokille *harjoitustyo.tiedot*-niminen alipakkaus. Harjoitustyössä käytetään *harjoitustyo.apulaiset*-alipakkaukseen sijoitettua *In*-luokkaa tietojen lukuun näppäimistöltä. Samaan alipakkaukseen on liitetty myös opettajan antamat *Tietoinen*-, *Syvakopioituva<T>*-, *Sailova<T>*- ja *Ooperoiva<E>*-rajapinnat. Tähän pakkaukseen ei ole syytä tehdä muutoksia, koska WETO käyttää aina alkuperäistä apulaispakkausta. *Harjoitustyo.apulaiset*-pakkaus julkaistaan kurssisivujen *Opetus | Harjoitustyö* -kohdassa ja GitLab-palvelussa. Sijoita mahdollisesti toteuttamasi *HakemistoIteraattori<E>*-luokka *harjoitustyo.iteraattorit*-pakkaukseen. Voit tehdä halutessasi lisää alipakkauksia. Kuvassa 16 on annettu *harjoitustyo*-pakkauksen rakenne.

```
+---harjoitustyo
               |---Oope2HT.java
               |
               +---apulaiset
               |           |---In.java
               |           |---Ooperoiva.java
               |           |---Sailova.java
               |           |---Syvakopioituva.java
               |           |---Tietoinen.java
               |
               +---iteraattorit
               |              |--- HakemistoIteraattori.java
               +---tiedot
               |         |---Hakemisto.java
               |         |---Tiedosto.java
               |         |---Tieto.java
               |
               ...
```

Kuva 16: Harjoitustyön pakkauksen rakenne. Valinnaiset osat on kursivoitu.

Ohjelman kääntäminen komentoikkunassa on helpointa hakemistosta, jolla on välittömänä alihakemistona *harjoitustyo*-hakemisto. Tällöin esimerkiksi *harjoitustyo.tiedot*-alipakkauksen voi kääntää Windows-käyttöjärjestelmässä komennolla *javac harjoitustyo\tiedot*.java*. Koko ohjelman voi kääntää tästä hakemistosta käsin Windowsissa komennolla *javac harjoitustyo\Oope2HT.java* ja suori-

tus tapahtuu komennolla *java harjoitustyo.Oope2HT*. Java-kääntäjä ei osaa aina päätellä oikein mitä kaikkea sen pitäisi kääntää, jotta viimeisimmät muutokset saataisiin mukaan ohjelmaan. Kaikki tavukooditiedostot ennen käännöstä poistamalla voit varmistaa, että lähdekooditiedostot käännetään varmasti. Ole varovainen, kun poistat tavukoodia.

4. Palautus ja tarkistus

Harjoitustyössä on välipalautuspiste ennen lopullista palautusta. Välipalautuksen tavoitteena on saada kurssilaiset aloittamaan harjoitustyön teko ajoissa. Harjoitustyöstä täytyy palauttaa perjantaihin 12.4.2019 klo 12.00 (keskipäivä) mennessä Javalla toteutettu luokkahierarkia (luvut 3.2 ja 3.3) ja *OmaLista<E>*-luokka (luku 3.4). Välipisteessä ei tarvitse palauttaa dokumentaatiota.

Valmis ohjelma ja sen dokumentaatio on palautettava viimeistään maanantaina 29.4.2019 klo 12.00 (keskipäivä).

Molemmat palautukset tehdään WETO-järjestelmään, joka tarkistaa automaattisesti ratkaisujen rakennetta ja toiminnallisuutta. Välipisteessä tarkistuksen pääroolissa on WETOn automaattinen testaus. Opettaja lukee koodia välipisteessä vain poikkeustapauksissa. Valmis ohjelma luetaan kokonaisuudessaan.

WETO tarkastelee ohjelman rakennetta ja toiminnallisuutta. Toiminnallisuutta arvioidaan pääosin vertailemalla mallivastauksen ja opiskelijoiden ratkaisujen tulosteita. Tästä syystä edellä annettuja tulostemäärittelyjä on seurattava merkilleen.

Ennen palautusta on syytä varmistaa, että ohjelma toimii varmasti oikein viimeisimpien muutosten jälkeen.

5. Dokumentointi

Harjoitustyöstä kirjoitetaan lopullisen palautuksen yhteydessä dokumentti, jossa tulee olla:

- Kansilehdellä tekijän nimi, opiskelijanumero ja sähköpostiosoite. Sivun keskellä tulisi olla suuremmalla fontilla dokumentin nimi. Kurssin kotisivuilla julkaistaan esimerkinomainen kansilehti.

- UML-luokkakaavio ja sen lyhyt sanallinen kuvaus. Itse tehtyjen luokkien luokkasymboleissa annetaan luokan kaikki vakiot, attribuutit ja metodit.

- 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.

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 symboleiden 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.

Kaavioiden piirto on helpompaa erityisesti UML-kaavioiden piirtoon tehtyjä ohjelmistoja käyttäen. 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 esimerkiksi UMLet-ohjelma, joista kerrotaan lisää kurssisivujen *Harjoitustyö* | *UML* -kohdassa. NetBeans- ja Eclipse-ohjelmistojen kaltaisiin kehitysympäristöihin on saatavilla takaisinmallinnuksen hallitsevia lisäosia.

Koodi dokumentoidaan lopullisessa palautuksessa kirjoittamalla lyhyet Javadoc-kommentit jokaiselle attribuutille, metodille ja luokalle. Rakentajia ja aksessoreita ei tarvitse kommentoida, mikäli ne ovat toiminnoiltaan tavanomaisia. Javadoc on Java-ympäristön tarjoama menetelmä koodin puoliautomaattiseen dokumentoimiseen. Javadoc-kommentoinnista annetaan erilliset ohjeet kurssin verkkosivujen *Harjoitustyö* | *Javadoc* -kohdassa. Javadoc-kommentit voi halutessaan kirjoittaa jo välipalautuksen yhteydessä.

Muista edelleen selittää metodien sisällä normaaliin tapaan metodien keskeiset ja vaikeaselkoiset osuudet tavanomaisia kommentteja käyttäen.

6. Ohjaus

Mikroluokkaohjausta tarjotaan välipalautusta ennen ja sen jälkeen. Ohjausten ajat ja paikat ilmoitetaan myöhemmin kurssin kotisivuilla. Ohjaukseen voi tulla omasta harjoitusryhmästä riippumatta: voit käydä ohjattavana aikana, joka sopii sinulle parhaiten. Luuppi tarjoaa ohjausta harjoitustyöhön koodauspajassaan. On varmempaa, että vastuuopettaja on paikalla huoneessaan, kun sovit tapaamisajan etukäteen.

Pienempiä kysymyksiä voi esittää ohjaajille sähköpostitse. Harjoitustyön pääasiallinen sähköpostiohjaaja on mikroharjoitusryhmäsi vetäjä. Näet ryhmäsi WETO-järjestelmän *Students*-välilehdeltä. Kysy neuvoa sähköpostitse sopivimmaksi katsomaltasi opettajalta, jos et ole valinnut ryhmää. Suuremmat kysymykset on helpointa kysyä mikroluokkaohjauksissa.

Kysymyksiä ja niihin annettuja vastauksia kerätään kurssin verkkosivujen *Harjoitustyö*-kohtaan. Ennen kysymistä kannattaa siis tarkistaa kurssin verkkosivut, koska 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.

7. Arvostelu

Arvosana perustuu harjoitustyön työn laajuuteen ja sen toimintaan testeissä sekä palautusaikatauluun. Suppealla työllä (ei *cp*- ja *find*-komentoja) voi saada korkeintaan arvosanan 4.

Lopullisen palautuksen testauksessa on julkinen ja salainen osuus. Huomaa, että salaisissa testeissä käytetään eri syötteitä kuin kotisivuilla annetuissa esimerkeissä. Ohjelman oikeellinen toiminta esimerkkien osalta ei siksi takaa salaisten testien läpäisyä. Testauksen pisteytys perustuu aina ensimmäiseen palautukseen. Korjauskierroksen kautta ei voi korottaa pistemääräänsä.

Kurssin arvosana perustuu alla olevan taulukon mukaan laskettuihin pisteisiin, joita voi saada 0–6 kpl.

Arvioinnin kohde	Toteutunut	Pisteet
Työn laajuus	laaja	1
	suppea	0
Välipisteen palautus ajoissa	kyllä	1
	ei	0
Välipisteen testit	onnistuivat	1
	epäonnistuivat	0
Lopullinen palautus ajoissa	kyllä	1
	ei	0
Lopullisen palautuksen julkiset testit	onnistuivat	1
	epäonnistuivat	0
Lopullisen palautuksen salaiset testit	onnistuivat	1
	epäonnistuivat	0

Alla olevassa taulukossa on annettu edellisestä taulukosta laskettujen pisteiden mukaiset arvosanat.

Arvosana	Pisteet	Selitys
5	6	Työ on laaja. Välipalautuksen testit ovat onnistuneet. Lopullisen palautuksen julkiset testit ovat onnistuneet. Lopullisen palautuksen salaiset testit ovat onnistuneet. Molemmat palautukset ovat olleet ajoissa.
4	5	Esimerkiksi: - Työ on laaja. Välipalautuksen testit ovat onnistuneet. Lopullisen palautuksen julkiset testit ovat onnistuneet. Lopullisen palautuksen salaisissa testeissä on ilmennyt ongelmaa. Molemmat palautukset ovat olleet ajoissa. - Työ on suppea. Välipalautuksen testit ovat onnistuneet. Lopullisen palautuksen julkiset testit ovat onnistuneet. Lopullisen palautuksen salaiset testit ovat onnistuneet. Molemmat palautukset ovat olleet ajoissa.
3	4	Nolla pistettä kahdesta arvioinnin kohteesta.
2	2 tai 3	Nolla pistettä kolmesta tai neljästä arvioinnin kohteesta.
1	0 tai 1	Nolla pistettä viidestä tai kuudesta arvioinnin kohteesta.

Aikavaatimuksen osalta voidaan joustaa äärimmäisen hyvästä syystä, jollaisia ovat esimerkiksi pitkä ulkomaan matka tai kokopäivätöiden aiheuttamat ylitsepääsemättömät esteet.

Hylätyn työn voi palauttaa korjattuna pääsääntöisesti kolme kertaa. Välipisteessä hylätyn työn korjaukseen on aikaa muutama päivä. Lopullisen palautuksen korjausaikaa on useimmiten viikko työn hylkäyksestä.

Arvosana ei alene, jos työtä täytyy korjata sen rakenteen tai ohjelmointitavan osalta, kun lopullisen palautuksen julkinen tai salainen testaus on onnistunut.

Plagiointiin liittyy sanktio. Sanktio ei koske vain opiskelijaa, joka on plagioinut, vaan myös opiskelijaa, joka on sallinut työnsä plagioinnin. Toiselta opiskelijalta tämän tietämättä kopioidun koodin käyttö johtaa kopioijan koko kurssisuorituksen hylkäämiseen.

Lähteet

1. Wikipedia-yhteisö, Command-line interface, https://en.wikipedia.org/wiki/Command-line_interface (Luettu viimeksi 11.3.2019.)

2. J. Laurikkala, Lausekielinen ohjelmointi I -kurssin kotisivut https://coursepages.uta.fi/tiep1/syksy-2018/ohjelmointivalineita/komentoikkuna/ (Luettu viimeksi 11.3.2019.)

3. Wikipedia-yhteisö, File system, https://en.wikipedia.org/wiki/File_system (Luettu viimeksi 11.3.2019.)

4. J. Laurikkala, Lausekielinen ohjelmointi I -kurssin luentorunko, luku 14, https://coursepages.uta.fi/tiep1/syksy-2018/luennot/ (Luettu viimeksi 11.3.2019.)

5. J. Laurikkala, Lausekielinen ohjelmointi II -kurssin luentorunko, luku 7, https://coursepages.uta.fi/tiep5/syksy-2018/luennot/ (Luettu viimeksi 11.3.2019.)

6. J. Laurikkala, Olio-ohjelmoinnin perusteet I -kurssin luentorunko, luku 3, https://coursepages.uta.fi/tiea2-1a/kevat-2019/luennot/ (Luettu viimeksi 11.3.2019.)