1. Johdanto

Harjoitusyössä ohjelmoidaan Java-kielellä tekstikokoelmia käsittelevä olioperustainen Last Oope Task (L.O.T.) -ohjelma. Tekstikokoelma eli tekstikorpus [1] koostuu tekstimuotoisista dokumenteista. Tekstidokumentti sisältää yleensä sanoja ja muuta ihmisen luontaisesti ymmärtämää tietoa. Dokumentissa on usein varsinaisen tiedon ohella tunniste ja muuta metatietoa, esimerkiksi päivämäärä.

Harjoitustyöohjelma lukee kokoelman dokumentit tekstitiedostosta ja säilöö dokumentit linkitetylle listalle. Ohjelma osaa tulostaa kokoelman tietoja eri tavoin, dokumentteja voi lisätä ja poistaa, kokoelmasta voidaan hakea dokumentteja ja kokoelman lajittelu on mahdollista samoin kuin esikäsittely. Esikäsittelyn tavoitteena on tehostaa tiedonhakua poistamalla dokumenteista välimerkkejä ja yleisiä sanoja. Esikäsittelystä on hyötyä myös siten, että kokoelman kokoa saadaan pienennettyä. Kaikki muutokset voidaan perua lataamalla kokoelma uudelleen.

Ohjelman rakenne ja toiminnallisuus on monella tapaa rajoittunut verrattuna todellisiin tekstikokoelmia käsitteleviin ohjelmiin. Tositoimiin tarkoitetuissa ohjelmissa käytetään tehokkaampia tietorakenteita, jolloin jopa miljoonien dokumenttien kokoisten kokoelmien käsittely on mahdollista. Myös esikäsittely on todellisuudessa laajempaa: tyypillinen tapa tehostaa hakua on normalisoida erityisesti taivutetut sanat niitä katkaisemalla tai perusmuotoistamalla. Yhdyssanat esikäsitellään yleensä ennen hakuja siten, että myös yhdyssanan yksittäisiä sanoja voidaan käyttää tiedonhaussa. Yleisten sanojen lisäksi voidaan poistaa harvinaisia sanoja.

Harjoitustyö tehdään itse ja omalla ajalla. Muiden kurssilaisten kanssa voi vaihtaa ideoita, mutta ei koodia. Kaikki vastuuopettajan julkaisema koodi on vapaasi käytettävissä. Muilta kuin kurssin sivuilta löytyneen lähdekoodin käyttö on kielletty.

Harjoitustyö on pakollinen ja keskeisin osa kurssia. Harjoitustyön voi tehdä valintansa mukaan laajassa tai suppeassa muodossa. Laajassa työssä on suppean työn ominaisuuksien lisäksi toiminnot frekvenssien tulostamiseen, kokoelman lajitteluun ja rivitettyyn tulostamiseen sekä opettajan antamien rajapintojen oletusmetodien korvaukset. Kurssin arvosanaan vaikuttavat työn laajuus, testaustulokset ja aikataulun noudattaminen. Suppealla työllä voi saada korkeintaan arvosanan 4.

2. Tekstikokoelmat ja sulkusanalista

Harjoitustyöohjelma osaa käsitellä kahdentyyppisiä tekstikokoelmia. Kokoelman dokumentit voivat olla joko uutisia tai vitsejä. Molemman tyyppisissä dokumenteissa on tekstin ohella kaksi metatietoa, joista dokumentin yksilöivä kokonaislukutunniste on yhteinen sekä vitseille että uutisille. Tieto vitsin lajista on ominaista pelkästään vitsidokumenteille. Uutisen sisältävälle dokumentille ominaista on puolestaan päivämäärä, jona uutinen on julkaistu. Dokumentit ovat aina englanniksi eikä niissä ole esimerkiksi skandinaavisten kirjainten tapaisia erikoismerkkejä.

Dokumentit on annettu tekstitiedostoissa, joissa yhden dokumentin tiedot ovat aina yhdellä rivillä. Dokumentin tiedot erotetaan toisistaan merkkijonolla "///". Vitsidokumentin sisältävän rivin rakenne on seuraava: tunniste, erotin, vitsin laji, erotin ja vitsin teksti. Uutisdokumentin sisältävän rivin tiedot ovat puolestaan: tunniste, erotin, uutisen päivämäärä, erotin ja uutisen teksti. Dokumentin esitystapa tiedostossa on täsmälleen sama kuin sitä vastaavan olion merkkijonoesitys ohjelmassa (luku 4.3). Dokumentit ovat tiedostossaan satunnaisessa järjestyksessä ja ne pitää lisätä listalle tunnisteensa mukaiseen nousevaan järjestykseen (luku 4.4).

Tiedostoissa ei ole metatietoa, josta voisi päätellä minkä typpisten dokumenttien kokoelman on kyseessä, vaan sisällön laatu päätellään tiedostojen nimistä. Vitsejä ja uutisia sisältävien tiedostojen nimet alkavat vastaavasti *jokes*- ja *news*-sanoilla.

Kuvassa 1 on esitetty ikivihreitä vitsejä sisältävän tiedoston *jokes\_oldies.txt* sisältö lähes kokonaisuudessaan. Tunnisteilla 9 ja 11 yksilöidyt vitsien rivit on katkaistu, koska ne eivät mahdu kuvaan kokonaisena. Näet nämä vitsit kokonaisuudessaan kurssin *Harjoitustyö*-kohdassa julkaistavasta tiedostosta. Tiedoston ensimmäisellä rivillä on tunnisteella 4 yksilöity ja lajiltaan sekalainen (miscellaneous) vitsi, jonka kuuluu: "This may be the wine talking, but I really, really, really, really love wine.". Valtaosa tiedoston sisältämistä vitseistä liittyy tietojenkäsittelyyn (computing).

Dokumentteihin viitataan jatkossa lyhyemmin ristikkomerkillä ja dokumentin tunnisteella. Esimerkiksi tunnisteella 9 yksilöity dokumentti on dokumentti #9.

Kuvassa 2 näkyy osa *news\_reuters.txt*-nimisestä tiedostosta, sisältää joitakin Reuters-uutistoimiston tuottamia uutisia vuodelta 1987. Kaikki rivit on katkaistu liian pitkinä. Esimerkiksi tunnisteella 20255 yksilöity uutinen on kokonaisuudessaan "U.S. Secretary of State George Shultz is likely to visit Moscow soon for talks following new Soviet arms control proposals, U.S. Officials said. The officials told Reuters no decision had been reached on when the trip might take place, but it was likely to be within a month. Kremlin leader Mikhail Gorbachev proposed on Saturday that talks on medium range missiles be separated from other arms issues. Schultz is currently visiting Peking.". Tämäkin tiedosto tulee kokonaisuudessaan saataville kurssisivuille.

Tekstin osien erotin on välilyönti. Muut välimerkit huomioidaan vasta esikäsittelyssä. Pelkkää välilyöntiä erottimena käytettäessä kaikki tekstin osat eivät ole oikeita sanoja. Esimerkiksi vitsin #4 tekstissä on sanojen lisäksi myös sanan ja erotinmerkin muodostamat osat "talking,", "really,", "really,", "really,", ja "wine." Sanoja ja tekstin muita osia kutsutaan jatkossa selvyyden vuoksi avaimiksi. Oikeissa järjestelmissä merkittävä osa avaimista ei ole oikeita sanoja, jos sanat normalisoidaan sanavartaloiksi poistamalla niiden lopusta pääteainesta, kunnes saavutetaan kaikille taivutusmuodoille yhteinen perusmuoto.

Kokoelmien sisältävien tiedostojen voi olettaa aina olevan kunnossa. Esimerkiksi tunnisteita ei puutu, dokumenteissa on aina tekstiä, rivien aluissa tai lopuissa ei ole välilyöntejä ja tiedostossa on aina vain yhden tyyppisiä dokumentteja.

Tiedonhaun kannalta yleisyytensä vuoksi merkityksettömät sanat annetaan niin sanotulla sulkusanalistalla. Sulkusanat välitetään ohjelmalle tekstitiedostossa, jossa kullakin rivillä on yksi sulkusana. Kuvassa 3 on annettu 319 sanaa sisältävän *stop\_word.tx*t-tiedoston viisi ensimmäistä sanaa. Listalla on muun muassa epämääräinen artikkeli "a", joka esiintyy lähes jokaisessa englanninkielisessä dokumentissa.

Kuva 1: Tiedoston *jokes\_oldies.txt* sisältöä.

4///miscellaneous///This may be the wine talking, but I really, really, really, really love wine.

11///computing///These two strings walk into a bar and sit down. The bartender says, "So what'll it be?" The first string...

9///computing///If you put a million monkeys at a million keyboards, one of them will eventually write a Java program. The...

3///miscellaneous///Nothing sucks like a Hoover.

5///computing///Q. What is the biggest lie in the entire universe? A. "I have read and agree to the Terms & Conditions."

10///computing///To understand what recursion is, you must first understand recursion.

8///computing///A SQL query goes into a bar, walks up to two tables and asks, "Can I join you?"

2///miscellaneous///Did you hear about the semi-colon that broke the law? He was given two consecutive sentences.

1///miscellaneous///Q. What's the difference between ignorance and apathy? A. I don't know and I don't care.

6///computing///Q. How does a computer get drunk? A. It takes screenshots.

7///computing///Q: How many programmers does it take to change a light bulb? A: None. It's a hardware problem.

12///computing///When your hammer is C++, everything begins to look like a thumb.

Kuva 2: Tiedoston *news\_reuters.txt* sisältö osittain.

20255///1.3.1987///U.S. Secretary of State George Shultz is likely to visit Moscow soon for talks following new Soviet arms...

17033///23.4.1987///Special Japanese envoy Shintaro Abe said in a brief interview with Reuters that the feeling in the U.S.... 22996///9.3.1987///Union Bank of Finland is issuing a 10 billion yen eurobond due March 30, 1992 paying five pct and priced...

21576///19.10.1987///Six black miners have been killed and two injured in a rock fall three km underground at a South... 3225///9.3.1987///The Ghana Cocoa Board said it purchased 1,323 tonnes of cocoa in the 21st week, ended February 26, of the... 13876///7.4.1987///Pyramid Technology Corp said it has reached an agreement to sell Oracle Corp's Oracle database and...

1023///3.3.1987///Sandoz AG said it planned a joint venture to produce herbicides in the Soviet Union. The company said it...

Kuva 3: Tiedoston *stop\_words.txt* alku

a

about

above

across

after

...

3. Ohjelman toiminnot

Seuraavassa esitellään ohjelman toiminnallisuutta pienten esimerkkien avulla. Kurssin kotisivuilla julkaistaan lisää esimerkkiajoja. Laajassa harjoitustyössä on oltava kaikki tässä luvussa kuvatut toiminnot. Suppeassa harjoitustyössä ei ole avainten frekvenssien tulostusta (*freqs*-komento, luku 3.8), lajittelua (*sort*-komento, luku 3.9) eikä sarakkeeseen tulostusta (*pprint*-komento, luku 3.10).

3.1. Ohjelman käynnistys

Ohjelma lataa käynnistyessään kokoelman ja sulkusanalistan tiedot tekstitiedostoista ohjelman tietorakenteisiin. Tiedostojen nimet välitetään ohjelmalle komentoriviparametreina, joista ensimmäinen kertoo kokoelman tiedoston nimen. Ohjelma käynnistetään kutsumalla ajoluokkaa *Oope2HT* sen lähdekoodin sisältävässä hakemistossa eli versionhallinnan työhakemistossa (luku 4.6).

Ohjelma käynnistetään seuraavalla komennolla:

java Oope2HT jokes\_oldies.txt stop\_words.txt

kun kokoelman dokumentit halutaan lukea tiedostosta *jokes\_oldies.txt* ja sulkusanat tiedostosta *stop\_words.txt*.

Tiedostoille voidaan määritellä polku. Polun käyttö on suositeltavaa, jotta testitiedostoja ei tulisi vahingossa lisättyä versionhallintaa. Tiedostot voitaisiin lukea esimerkiksi ylihakemistossa olevasta *kokoelmat*-hakemistosta:

java Oope2HT ..\kokoelmat\jokes\_oldies.txt ..\kokoelmat\stop\_words.txt

Mac- tai Linux-järjestelmässä polkuerottimena käytetään jakoviivaa /.

Uudelleenohjausta käytettäessä ajokomento voisi olla esimerkiksi:

java Oope2HT jokes\_oldies.txt stop\_words.txt < in.txt > out.txt

Standardisyötevirta liitetään yllä annetussa esimerkissä näppäimistön asemasta *in.txt*-tiedostoon, josta syötteet luetaan ohjelmaan rivi kerrallaan. Standarditulosvirta liitetään puolestaan näytön asemasta *out.txt*-tiedostoon, jolloin ohjelman tulosteet saadaan tallennettua tiedostoon.

Ohjelman tulostaa tervehdyksen (luku 3.1) jälkeen virheilmoituksen "Wrong number of command-line arguments!" ja jäähyväiset, jos komentoriviparametreja on väärä määrä. Jos komentoriviparametrit ovat esimerkiksi unohtuneet täysin:

java Oope2HT

niin ohjelma tulostaa:

Welcome to L.O.T.

Wrong number of command-line arguments!

Program terminated.

Ohjelma toimii lähes samoin, kun tiedostoa ei löydetä. Ainoa ero on virheilmoitus. Jos ohjelmaa kutsuttaessa esimerkiksi kirjoitetaan kokoelmatiedoston nimi väärin:

java Oope2HT jokes-oldies.txt stop\_words.txt

niin tulostetaan:

Welcome to L.O.T.

Missing file!

Program terminated.

3.1. Käyttöliittymä

Ohjelmalla on tekstipohjainen käyttöliittymä. Ohjelman käynnistyessä tulostetaan kuvassa 4 esitetyllä tavalla ensin tervehdysteksti ja sitten omalle rivilleen kehote. Tämän jälkeen jäädään odottamaan käyttäjän syötettä välittömästi kehoteriviä seuraavalle riville. Tervehdys tulostetaan vain kerran.

Kuva 4: Ohjelma käynnistyksen jälkeen.

Welcome to L.O.T.

Please, enter a command:

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 ja jäädään odottamaan uutta komentoa seuraavalle riville, jos komento oli oikeellinen.

Ohjelma vastaa virheelliseen komentoon tulostamalla omalle rivilleen "Error!" ennen uuden komennon lukemista. Virheeksi katsotaan tuntematon komento (kuva 5) tai tunnettu komento virheellisellä parametrilla. Tunnetulla komennolla voi olla myös väärä määrä parametreja ja parametrit voivat olla väärän tyyppisiä. Tyyppiin liittyvä virhe voi tapahtua vain annettaessa komento, jolla tulee olla kokonaislukutyyppisiä parametriarvoja. Voit olettaa, että komennossa ei ole ylimääräisiä välilyöntejä ennen komentoa, komennon jälkeen, komennon ja sen parametrin välissä tai komennon parametrien välissä.

Kuva 5: Tuntemattoman komennon aiheuttama virhe.

save

Error!

Please, enter a command:

Kaikkien tulosteiden muoto on kiinnitetty, jotta ohjelmien automaattinen, tulosteiden vertailuun perustuva tarkistus olisi mahdollista (luku 5). Tarkistusta automatisoimalla pyritään siihen, että kurssin opettajille jää enemmän aikaa työn rakenteen ja laadun arviointiin. Komennot luetaan Javan *Scanner*-luokan avulla, jotta automaattisessa testauksessa ei tapahtuisi yllättäviä virheitä.

Seuraavissa luvuissa esitellään kukin ohjelman tuntemista komennoista esimerkkien avulla. Ohjelma on käynnistetty komentoriviparametreilla *jokes\_oldies.txt* ja *stop\_words.txt*.

3.2. Dokumenttien tulostaminen

*Print*-komento tulostaa näytölle tietyn dokumentin merkkijonoesityksen (luku 4.3) tai kaikkien dokumenttien merkkijonoesitykset listan järjestyksessä. Tulostettava dokumentti valitaan sen tunnisteen avulla. Parametriarvoja voi olla vain yksi: esimerkiksi kolmen dokumentin tunnisteen antaminen parametriarvona on virhe. Parametriarvo voi olla vääräntyyppinen, esimerkiksi merkkijono "thirteen".

Koko kokoelmaa tulostettaessa kukin dokumentti tulostetaan omalle rivilleen.

Kuvassa 6 tulostetaan dokumentti #6. Seuraava komento epäonnistuu, koska kokoelmassa ei ole dokumenttia #13.

Kuva 6: Dokumenttien tulostamista.

print 6

6///computing///Q. How does a computer get drunk? A. It takes screenshots.

Please, enter a command:

print 13

Error!

Please, enter a command:

3.3. Dokumentin lisääminen kokoelmaan

*Add*-komento lisää kokoelman uuden dokumentin käyttöliittymän kautta. Lisättävä dokumentti annetaan komennon parametriarvoksi merkkijonona, jonka muoto on sama kuin tiedostossa ja olion merkkijonoesityksessä (luku 4.3).

Lisäys epäonnistuu, jos kokoelmassa on jo annetulla tunnisteella yksilöity dokumentti tai vitsien kokoelmaan yritetään lisätä uutista tai päinvastoin. Lisäys myöskään onnistu, jos parametria ei ole annettu tai niitä on enemmän kuin yksi.

Tehtävää on helpotettu siten, että parametriksi saadun merkkijonon voi olettaa olevan kokoelman sisällön tapaan aina oikeassa muodossa.

Kuvassa 7 kokoelmaan lisätään onnistuneesti dokumentti #13, joka tulostetaan näytölle heti lisäyksen jälkeen.

Kuva 7: Uuden dokumentin lisääminen kokoelmaan.

add 13///computing///This code was generated by a tool.

Please, enter a command:

print 13

13///computing///This code was generated by a tool.

Please, enter a command:

3.4. Kokoelmasta hakeminen

Kokoelmasta voidaan hakea tiettyjä avaimia sisältäviä dokumentteja *find*-komennolla. Hakuavaimet annetaan komennolle parametriarvona. Haku alkaa listan ensimmäisestä dokumentista ja etenee dokumentti kerrallaan listan loppuun. Komento tulostaa näytölle riveittäin niiden dokumenttien tunnisteet, jotka sisältävät kaikki komennolle annetut avaimet. Osittain hakua vastaavien dokumenttien tunnuksia ei tulosteta. Pienet ja suuret kirjaimet katsotaan eri merkeiksi.

Jos hakuavaimet ovat esimerkiksi "cat" ja "dog", niin molempien avainten on esiinnyttävä dokumentissa vähintään kerran, jotta dokumentin katsotaan vastaavan hakua. Esimerkiksi vain avaimen "cat" sisältävä dokumentti ei vastaa hakua. Haussa ei huomioida dokumentin avainten osajonoja. Esimerkiksi dokumentti, jossa on avaimet "catnip"ja "hotdog", mutta ei avaimia "cat" ja dog", ei vastaa avaimilla "cat" ja "dog" ja tehtävää hakua, koska hakuavaimet eivät esiinny dokumentissa sellaisenaan.

Ohjelma tulostaa pelkän kehotteen, jos kokoelmassa ei ole hakua vastaavia dokumentteja (kuva 9). Virheilmoitus tulostetaan vain, jos komennolle ei anneta parametreja.

Kuvassa 8 haetaan aluksi avaimella "The", joka löydetään vitseistä #9 ja #11. Haussa ei huomioida pienellä alkukirjaimella aloitettua "the"-avainta. Näin esimerkiksi vitsi #5 ei vastaa hakua. Vitsien tunnukset ovat tulosteessa käänteisessä järjestyksessä tiedostoon verrattuna (kuva 1), koska dokumentit ovat listalla tunnisteen mukaisessa kasvavassa järjestyksessä. Seuraava haku tehdään avaimilla "understand" ja "recursion". Hakua vastaa vain rekursiovitsi #10, jossa on molemmat avaimet. Viimeinen haku kertoo, että avain "bar" löydetään vitsistä #11. Haku ei vastaa vitsiä #8, koska sen avain on "bar,". Myöhemmin nähdään, että vitsi #8 löydetään, kun dokumentit esikäsitellään poistamalla avaimiin jääneet välimerkit.

Kuva 8: Vitsien kokoelmaan kohdistetuja hakuja.

find The

9

11

Please, enter a command:

find understand recursion

10

Please, enter a command:

find bar

11

3.5. Dokumentin poistaminen

Dokumentti poistetaan kokoelmasta *remove*-komennolla. Komennon parametriarvo on poistettavan dokumentin tunniste. Poisto epäonnistuu, jos kokoelmassa ei ole dokumenttia annetulla tunnisteella, parametriarvoja ei ole annettu, parametriarvoja on enemmän kuin yksi tai parametriarvo on vääräntyyppinen.

Kuvassa 9 kokoelmasta poistetaan dokumentti #11. Poisto onnistuu ja kokoelmassa ei enää ole avainta "bar" sisältäviä dokumentteja.

Kuva 9: Dokumentin poistaminen.

remove 11

Please, enter a command:

find bar

Please, enter a command:

3.6. Kokoelman esikäsittely

*Polish*-komento käynnistää kokoelman esikäsittelyn. Komento toimii vaiheittain siten, että dokumenteista poistetaan ensin parametrina annetut välimerkit. Poistettavat välimerkit annetaan ohjelmalle ne peräkkäin kirjoittamalla. ~~Välimerkit voidaan myös jättää antamatta, jolloin esikäsittelyn ensimmäisessä vaiheessa kokoelmaa ei muuteta mitenkään~~. Kokoelma jätetään ennalleen ~~myös~~, jos dokumenteissa ei ole parametrina annettuja merkkejä. Ohjelma tulostaa virheilmoituksen, jos parametreja ei ole annettu tai niitä on kaksi tai useampia.

Esikäsittelyn toisessa vaiheessa suuret kirjaimet muutetaan pieniksi.

Kolmannessa vaiheessa dokumenteista poistetaan kaikki sulkusanalistan sanojen esiintymät.

Esikäsittely tulee suorittaa edellä annetussa järjestyksessä, jotta lopputulos olisi sama kuin malliohjelmassa.

Kuvassa 10 esikäsittelykomentoa kutsutaan antamalla parametriksi merkkijono ",.:?"'". Avaimella "bar" haettaessa löydetään tällä kertaa vitsi #8, koska esikäsittely poisti avaimista muun muassa pilkut. Vitsin #8 tulostamalla nähdään, että välimerkkien lisäksi dokumenteista on poistettu yleisiä sanoja ja kaikki suuret kirjaimet on muutettu pieniksi. Esikäsittelyn helpottaa dokumenttien ohjelmallista käsittelyä, mutta tekee toisaalta dokumenteista hankalammin ymmärrettäviä. Esikäsitelty vitsi #8 on menettänyt parhaan teränsä.

Kuva 10: Vitsikokoelman esikäsittelyä.

print 8

8///computing///A SQL query goes into a bar, walks up to two tables and asks, "Can I join you?"

Please, enter a command:

polish ,.:?"'

Please, enter a command:

find bar

8

Please, enter a command:

print 8

8///computing///sql query goes bar walks tables asks join

Please, enter a command:

3.7. Muutosten peruminen

*Reset*-komento lataa dokumenttitiedoston uudelleen ja poistaa kaikki aiemmin tehdyt muutokset. Komennolla ei ole parametreja: ohjelma tulostaa virheilmoituksen, jos parametriarvoja on annettu. Kuvassa 11 perutaan aiemmat muutokset ja varmistetaan tilanne tulostamalla aiemmin tutkittu vitsi #8.

Kuva 11: Muutosten peruminen.

reset

Please, enter a command:

print 8

8///computing///A SQL query goes into a bar, walks up to two tables and asks, "Can I join you?"

Please, enter a command:

3.8. Frekvenssien tulostaminen

Kokoelman avainten absoluuttiset frekvenssit tulostetaan valinnaisella *freqs*-komennolla. Toteuta komento vain, jos teet harjoitustyön laajan version. Kukin avain ja sen frekvenssi tulostetaan omalle rivilleen yhdellä välilyönnillä erotettuna. Tulostus tehdään avainten aakkosjärjestyksessä, jonka määrää Javan API:n tietorakenne (luvut 4.2 ja 4.4). Viimeisellä rivillä kerrotaan avainten kokonaisfrekvenssi.

Komennolla voidaan tulostaa joko yksittäisen dokumentin tai koko kokoelman avainten frekvenssit. Dokumentti yksilöidään parametriarvona annettavalla tunnisteella. Ohjelma tulostaa virheilmoituksen, jos tunniste on arvoltaan tai tyypiltään virheellinen tai parametriarvoja on useampi kuin yksi kappale.

Kuvassa 12 tulostetaan vitsin #10 avainten frekvenssit. Alkuperäisessä dokumentissa on yhteensä 10 avainta, joista avain "understand" esiintyy kahdesti. Esikäsittelyn jälkeen vitsissä #10 on enää kaksi avainta. Molempien avainten frekvenssi on kaksi.

Kuva 12: Vitsin #10 avainten frekvenssit ilman esikäsittelyä.

print 10

10///computing///To understand what recursion is, you must first understand recursion.

Please, enter a command:

freqs 10

To 1

first 1

is, 1

must 1

recursion 1

recursion. 1

understand 2

what 1

you 1

A total of 10 words.

Please, enter a command:

polish ,.:?"'

Please, enter a command:

print 10

10///computing///understand recursion understand recursion

Please, enter a command:

freqs 10

recursion 2

understand 2

A total of 4 words.

Please, enter a command:

3.9. Kokoelman lajittelu

Kokoelman nousevaan järjestykseen lajitteleva *sort*-komento on valinnainen toiminto. Toteuta komento vain, jos teet harjoitustyön laajan version. Komennon parametrin arvo kertoo minkä tiedon mukaan lajittelu tehdään. Parametrilla "id" lajitellaan kokoelman tunnisteen mukaan. Vitsikokoelma voidaan lajitella vitsien lajin mukaan (parametriarvo "type") ja uutiskokoelma päivämäärän mukaan (parametriarvo "date").

Lajin mukainen lajittelu säilyttää vitsien keskinäisen järjestyksen lajittelun muodostamien lajiryhmien sisällä. Samoin päivämäärän mukaisessa lajittelussa samana päivänä julkaistut uutiset säilyttävät tunnisteen mukaisen järjestyksen.

Vitsin lajiin ja uutisen päivämäärään perustuvat lajittelut epäonnistuvat, jos ne kohdistetaan väärään tyyppiseen kokoelmaan. Ohjelma tulostaa virheilmoituksen myös, kun parametriarvo on jokin muu kuin jokin kolmesta edellä annetusta tai jos parametrille ei ole annettu arvoa tai jos arvoja on väärä määrä.

Esimerkki lajittelukomennon käytöstä on annettu kuvassa 13.

3.10. Tulostaminen rivittäen

*Pprint*-komento on ohjelman kolmas valinnainen toiminto. Toteuta komento vain, jos teet harjoitustyön laajan version. Komento tulostaa tunnisteella yksilöidyn dokumentin merkkijonoesityksen tai kokoelman jokaisen dokumentin merkkijonoesityksen annetun levyiseen sarakkeeseen. Sarakkeen leveys on ensimmäinen ja dokumentin tunniste toinen parametriarvo, kun komennolle annetaan kaksi parametriarvoa. Kokoelma tulostetaan listan määräämässä järjestyksessä listan alusta aloittaen.

Sarakkeeseen tulostaminen eli tulosteen rivittäminen tapahtuu siten, että aluksi tulostetaan dokumentin tunniste, erotin, dokumentin oma tieto, erotin ja dokumentin ensimmäinen avain. Ensimmäiselle riville tulostamista jatketaan tulostamalla välilyönnillä erotettuja avaimia niin pitkään kuin nykyinen avain mahtuu kokonaisena sarakkeeseen. Avainten tulostamista jatketaan toisella rivillä, kunnes toinen rivi on täynnä ja kolmannella rivillä, kunnes kolmas rivi on täynnä ja niin edelleen, kunnes kaikki avaimet on tulostettu.

Rivien loppuun ei tulosteta välilyöntejä tai muita tyhjemerkkejä, vaan kukin rivi päätetään heti sille viimeiseksi mahtuvan avaimen jälkeen. Huomaa, että sarake voi olla niin leveä, että rivittämistä ei tarvita, vaan koko dokumentin merkkijonoesitys mahtuu ensimmäiselle riville.

Tehtävää on helpotettu siten, että voit olettaa, että käyttäjä valitsevan sarakkeen leveyden aina siten, että rivin alku mahtuu sarakkeeseen ja että dokumentissa ei ole saraketta leveämpiä avaimia.

Ohjelma tulostaa virheilmoituksen, jos kokoelmasta ei löydy dokumenttia annetulla tunnisteella, tunnisteen parametriarvoa on väärää tyyppiä, parametriarvoja ei ole annettu tai parametriarvoja on useampi kuin kaksi kappaletta.

Varaa aikaa tämän komennon toteuttamiseen; se saattaa olla yllättävän vaikea toteuttaa.

Kuvassa 13 annettu esimerkki alkaa kokoelman uudelleenlatauksella, jota seuraa lajitelu vitsin lajin mukaiseen nousevaan järjestykseen, jonka seurauksena laskentaan (computing) liittyvät vitsit ovat ennen sekalaisia (miscellaneous) vitsejä. Dokumentit ovat molemmissa ryhmissä edelleen tunnuksen mukaisessa keskinäisessä järjestyksessä.

Lajiteltu kokoelma tulostetaan 50 merkin levyiseen sarakkeeseen vitsistä #5 aloittaen. Tulostus alkaa merkkijonolla "5///computing///Q." ja sitä voidaan jatkaa merkkijonoilla " What", " is", " the", " biggest", " lie", " in" ja " the". Seuraavan avaimen tulostaminen ensimmäiselle riville ei ole mahdollista, koska rivin pituus on jo 49 merkkiä. Ensimmäinen rivi päätetään ja tulostusta jatketaan seuraavalta riviltä siten, että hypätään välilyönti yli, jolloin rivi voidaan aloittaa avaimella "entire". Myös toiselle riville mahtuu 49 merkkiä. Tuloste päättyy kolmannelle riville, jolle yli jäänyt merkkijono " Terms & Conditions."" mahtuu kokonaisuudessaan. Myös kolmas rivi aloitetaan ilman välilyöntiä ja päätetään heti viimeisen merkin jälkeen rivinvaihtoon. Tulostusta jatketaan, kunnes kokoelman jokainen vitsi on tulostettu rivitettynä.

Kuva 13: Vitsin tyypin mukaan lajitellun kokoelman tulostaminen rivittäen.

reset

Please, enter a command:

sort type

Please, enter a command:

pprint 50

5///computing///Q. What is the biggest lie in the

entire universe? A. "I have read and agree to the

Terms & Conditions."

6///computing///Q. How does a computer get drunk?

A. It takes screenshots.

7///computing///Q: How many programmers does it

take to change a light bulb? A: None. It's a

hardware problem.

8///computing///A SQL query goes into a bar, walks

up to two tables and asks, "Can I join you?"

9///computing///If you put a million monkeys at a

million keyboards, one of them will eventually

write a Java program. The rest of them will write

Perl programs.

10///computing///To understand what recursion is,

you must first understand recursion.

11///computing///These two strings walk into a bar

and sit down. The bartender says, "So what'll it

be?" The first string says, "I think I'll have a

beer quag fulk boorg jdk^CjfdLk jk3s d#f67howe%^U

r89nvy owmc63^Dz x.xvcu." "Please excuse my

friend," the second string says, "He isn't

null-terminated."

12///computing///When your hammer is C++,

everything begins to look like a thumb.

1///miscellaneous///Q. What's the difference

between ignorance and apathy? A. I don't know and

I don't care.

2///miscellaneous///Did you hear about the

semi-colon that broke the law? He was given two

consecutive sentences.

3///miscellaneous///Nothing sucks like a Hoover.

4///miscellaneous///This may be the wine talking,

but I really, really, really, really love wine.

Please, enter a command:

3.11. Komentojen kaiuttaminen

*Echo*-komento käynnistää tai lopettaa komentojen kaiutuksen. Kaiutus tarkoittaa sitä, että ohjelma tulostaa näytölle saamansa komennon. Tästä toiminnosta on hyötyä, kun ohjelman testaustulokset ohjataan tekstitiedostoon. Kaiutus limittää ohjelman saamat komennot tulosteiden väliin, jolloin tekstitiedoston sisältö näyttää samalta kuin ohjelman ajo komentoikkunassa.

Ensimmäisen kerran annettu *echo*-komento aloittaa kaiutuksen, toisen kerran annettu komento lopettaa kaiutuksen, kolmas komento aloittaa kaiutuksen uudelleen ja niin edelleen. *Echo*-komento kaiuttaa aina itsensä: komento tulostetaan myös, kun kaiutus lopetetaan.

Kuvassa 14 käännetään aluksi kaiutus päälle, jolloin näytölle tulostetaan *echo*-komento itse ja vitsin #3 tulostava komento. Seuraavan *echo*-komennon seurauksena tulostetaan vielä *echo*, mutta ei enää muita komentoja.

Kuva 14: Komentojen kaiuttamista.

echo

echo

Please, enter a command:

print 3

print 3

3///miscellaneous///Nothing sucks like a Hoover.

Please, enter a command:

echo

echo

Please, enter a command:

print 3

3///miscellaneous///Nothing sucks like a Hoover.

Please, enter a command:

3.12. Ohjelman lopetus

Ohjelmasta poistutaan parametrittomalla *quit*-komennolla (kuva 15). Komennon jälkeen tulostetaan riviä vaihtaen "Program terminated."-teksti.

Kuva 15: Ohjelman lopetus.

quit

Program terminated.

4. Ohjelman rakenne

4.1. 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 kokoelman logiikasta vastaavaa koodia ei saa kirjoittaa sinne, vaan kokoelmalle on tehtävä oma *Kokoelma*-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 [2].

Erityisesti olio-ohjelmointiin liittyviä hyviä tapoja ovat luokkiin liitetyt yleiset kommentit, attribuuttien kommentointi, attribuuttien harkittu käyttö 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 näppäimistöltäpakatulla Javan API:n *Scanner*-luokalla.

4.2. Luokkahierarkia

Periytymistä käytetään harjoitustyössä erityisesti dokumenttien mallintamiseen. Ohjelmassa tulee olla abstrakti yliluokka *Dokumentti* ja siitä perityt konkreettiset *Uutinen*- ja *Vitsi*-luokat. Luokkien tiedot kätketään **private**-määreellä ja aksessorit nimetään harjoitusten tapaan attribuutin nimeä kuormittaen.

Yliluokalla on kaksi dokumenteille yhteistä tietoa. **Int**-tyyppinen *tunniste*-attribuutti on dokumentin yksilöivä tunnisteluku. Tunnisteen on oltava arvoltaan suurempi kuin nolla (> 0). *String*-tyyppinen *teksti*-attribuutti säilöö dokumentin varsinaisen tiedon eli tekstin. Attribuutin arvo ei saa olla **null** tai tyhjä merkkijono (""). Luokalla on parametrillinen rakentaja, jolla on järjestyksessä luetellen **int**- ja *String*-tyyppiset parametrit.

*Vitsi*-luokan omalle tiedolle tehdään *String*-tyyppinen *laji*-attribuutti, joka ei saa olla **null** tai tyhjä merkkijono. *Uutinen*-luokan omalle tiedolle tehdään puolestaan *LocalDate*-tyyppinen *päivämäärä*-attribuutti. *LocalDate* on Javan API:n *java. time*-pakkauksessa oleva luokka. Luokan *of*-metodi on hyödyllinen olioita luotaessa. Päivämäärä ei saa olla **null**.

Tee molemmille luokille parametrilliset rakentajat ja kutsu niistä yliluokan parametrillista rakentajaa. *Vitsi*-luokan rakentajan parametrit ovat **int**-, *String*- ja *String*-tyyppiset. Keskimmäinen parametri on luokan omalle tiedolle. *Uutinen*-luokan rakentajan parametrit ovat **int**-, *LocalDate*- ja *String*-tyyppiä.

Rakentajat – samoin kuin asettavat aksessorit – heittävät *IllegalArgumentException*-poikkeuksen, jos parametreissa havaitaan virhe. Poikkeus heitetään, jos tunnisteparametri on arvoltaan yhtä pienempi, viitetyyppinen parametri on **null**-arvoinen tai *String*-tyyppisen parametrin arvo on tyhjä merkkijono. Ohjelman testaus helpottuu, kun luot kunkin poikkeuksen antamalla rakentajalle poikkeuksen syytä valottavan viestin.

Aliluokkien parametrillisissa rakentajissa ei tulisi toistaa yliluokassa ja aliluokkien asettavissa aksessoreissa olevia virhetarkisteluja. Yliluokan rakentajan kutsu riittää tunnisteen ja tekstin tarkistukseen ja poikkeuksen heittämiseen. Aliluokan oman attribuutin arvon asettavaa metodia aliluokan rakentajassa kutsumalla saadaan aikaiseksi kaikki rakentajassa tarvittava virheenkäsittely. 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 yksi kolmesta opettajan valmiina antamasta rajapinnasta (luku 4.6). *Dokumentti*-luokka toteuttaa *Tietoinen<T>*-rajapinnan siten, että geneerinen tyyppi on kiinnitetty *T* tyypiksi *Dokumentti*. Abstraktit metodit toteutetaan sekä laajassa että suppeassa harjoitustyössä. *SanatTäsmäävät*-metodi on tarkoitettu käytettäväksi ohjelman hakutoiminnossa. Metodia kutsutaan *Kokoelma*-luokkaan tehtävästä metodista, joka toteuttaa hakutoiminnon (luku 3.4). S*iivoa*-metodia kutsutaan kokoelman esikäsittelykomennon (luku 3.6) toteuttavasta kokoelman metodista. *Siivoa*-metodin toteutus on helpointa *String*-luokan metodien avulla. Molemmissa kokoelman metodeissa tarvitaan silmukka, jonka avulla toteutettuja metodeja voidaan kutsua kunkin kokoelman dokumenttiolion kautta.

Rajapinnan *laskeFrekvenssit*-oletusmetodi toteutetaan vain laajassa harjoitustyössä. Metodi palauttaa dokumentin avaimet ja niiden frekvenssit Javan JDK:n *TreeMap<E>*-tyyppisessä sanakirjassa, joka löytyy *java.util*-pakkauksesta. Dokumentin avain ja sen frekvenssi muodostavat sanakirjan avain–arvo-parit. *TreeMap* järjestää parit avainten mukaiseen nousevaan järjestykseen. Metodia tarvitaan *Kokoelma*-luokan frekvenssit tulostavan komennon (luku 3.8) toteuttavassa metodissa, jossa parit tulostetaan näytölle samassa järjestyksessä kuin ne ovat sanakirjassa.

Rajapinnasta ja sen metodeista on kerrottu lisää kommenteissa.

Sijoitahierarkian luokat *harjoitustyo*.*dokumentit*-pakkaukseen (luku 4.6).

4.3. Dokumenttien merkkijonoesitykset ja vertailu

Korvaa *Object*-luokan *toString*-metodi luokkahierarkian jokaisella tasolla. Tee korvaus mielellään “ketjuttaen” siten, että kutsut aliluokkien *toString*-metodin korvauksissa yliluokan versiota **super**-attribuutin avulla.

*Dokumentti*-luokan *toString* palauttaa tunnisteen ja tekstin toisistaan merkkijonolla "///" erotettuna. Jos dokumentin tunniste on esimerkiksi 3 ja teksti "Nothing sucks like a Hoover.", paluuarvo on "3///Nothing sucks like a Hoover.". *Vitsi*-luokan *toString*-metodi lisää yliluokan metodin palauttaman merkkijonon keskelle tiedon vitsin lajista. Metodin paluuarvo voisi olla esimerkiksi "3///miscellaneous///Nothing sucks like a Hoover.".

*Uutinen*-luokassa korvatussa metodissa yliluokan tietojen merkkijonoesityksen keskelle lisätään päivämäärä Suomessa käytetyssä muodossa, jossa järjestysluvut annetaan järjestyksessä päivä, kuukausi ja vuosi ja erotetaan toisistaan pisteellä. Metodin paluuarvo voi olla esimerkiksi "20255///1.3.1987///U.S. Secretary of State George Shultz is likely to visit Moscow soon for talks following new Soviet arms control proposals, U.S. Officials said. The officials told Reuters no decision had been reached on when the trip might take place, but it was likely to be within a month. Kremlin leader Mikhail Gorbachev proposed on Saturday that talks on medium range missiles be separated from other arms issues. Schultz is currently visiting Peking.".

Dokumentteja vertaillaan vain niiden tunnisteiden avulla. Korvaa siksi *Object*-luokan *equals*-metodi vain *Dokumentti*-luokassa. Dokumentit katsotaan samoiksi, jos niiden tunnisteet ovat sama kokonaisluku. Toteuta myös *Comparable<T>*-rajapinnan *compareTo*-metodi *Dokumentti*-luokassa siten, että se vertaa tunnisteita. Palauta korvatusta metodista -1, 0 tai 1. Kiinnitä geneeriseksi tyypiksi *Dokumentti*.

4.4. OmaLista-luokka ja muut tietorakenteet

Harjoitustyössä tutustutaan linkitettyyn listarakenteeseen perimällä Javan *LinkedList<E>*-luokasta geneerinen *OmaLista<E>*-luokka, jota käytetään kokoelman dokumenttien säilömiseen. Listassa on toteuttava opettajan antama *Ooperoiva<E>*-rajapinta (luku 4.6). Älä kiinnitä tyyppiparametrin *E* arvoa, kun perit luokan ja korvaat rajapinnan.

Abstrakti *lisää*-metodi toteutetaan työn laajuudesta riippumatta. Metodi lisää alkion listalle siten, että alkio sijoittuu kaikkien itseään pienempien tai yhtä suurien alkioiden jälkeen ja ennen kaikkia itseään suurempia alkioita. Alkioiden järjestys päätellään vertaamalla lisättävää alkiota listan alkioihin *Comparable<T>*-rajapinnan *compareTo*-metodilla. Vertailussa tarvitaan vertailtaviin alkioihin liitettyjä *Comparable*-tyyppisiä apuviitteitä, koska tyyppiparametrilla *E* ei ole *Comparable<T>*-toteutusta. Käytä metodissa sen kommenteissa mainittua ja otsikossa annettua annotaatiota tyyppimuunnoksiin liittyvien kääntäjän varoitusten estämiseen. Älä käytä metodin toteutuksessa Javan API:n valmiita tai itse toteutettuja lajittelualgoritmeja, sillä listan pitää järjestyä alkioiden lisäämisen kautta.

Dokumentit lisätään listalle *lisää*-metodilla tiedostosta listalle ladattaessa (luku 3.1) tai kun ohjelmalle annetaan *add*-komento (luku 3.3). Dokumentteja sisältävä lista järjestyy automaattisesti tunnisteen mukaan nousevaan järjestykseen, kun dokumentit lisätään *lisää*-metodilla.

*Lajittele*-oletusmetodi toteutetaan vain laajassa harjoitustyössä. Varsinaiseen lajitteluun suositellaan käytettäväksi Javan API:n valmista lajittelualgoritmia, koska metodin oppimistavoite on keksiä, kuinka metodia kutsutaan lambdaa käyttäen.

Metodien tulee olla “puhtaita” yleiskäyttöisiä listaoperaatioita, joita voitaisiin käyttää myös muissa tehtävissä. *OmaLista*-luokassa ei saa siten esiintyä metodit harjoitustyön sitovia tunnuksia. Erityisesti tunnuksia *Dokumentti*, *Uutinen* ja *Vitsi* ei saa käyttää oman listaluokan metodeissa.

*Ooperoiva<E>*-rajapinnasta ja sen metodeista on kerrottu lisää kommenteissa.

Sulkusanalistan sanojen säilömiseen voi käyttää mitä tahansa tehtävään sopivaa tietorakennetta. Taulukko on huono valinta, koska sen koko ei voi muuttua linkitettyjen rakenteiden tapaan.

Koko kokoelman avainten frekvenssien laskevassa käytetään *TreeMap<E>*-rakennetta, jotta sanakirjan avainten järjestys olisi kiinnitetty yksikäsitteisesti myös kokoelman laajuisesti.

Sijoitaoma lista *harjoitustyo*.*omalista*-pakkaukseen (luku 4.6).

4.5. Kokoelma

*Kokoelma*-luokan vastuulla on nimensä mukaan kokoelman hallinnointi. Luokalla on kätketty *OmaLista<Dokumentti>*-tyyppinen *dokumentit*-attribuutti, joka sisältää viitteet kokoelmaan kuuluviin dokumenttiolioihin. *Kokoelma*-oliolla on näin osaolio, joka luomisesta ja hallinnoinnista kokoelma on vastuussa. Kirjoita oletusrakentaja, jossa *dokumentit*-attribuuttiin liitetään rakentajassa luotu tyhjä listaolio. Tee attribuutille lukeva aksessori.

Kokoelma toteuttaa komennot omaa listaa käsittelevinä metodeina. Älä nimeä *Kokoelma*-luokan metodeja suoraan komentojen nimiä käyttäen, vaan käytä pitempiä nimiä. Käytä komentojen nimistä suomenkielisiä vastineita, jos kirjoitat ohjelman suomenkielellä ja haluat sisällyttää komentojen nimet metodien tunnuksiin.

Toteuta *Kokoelma*-luokassa opettajan antama *Kokoava*-rajapinta (luku 4.6). *Lisää*-metodi toteuttaa käytännössä *add*-komennon (luku 3.3) ja *hae*-metodi *find*-komennon parametrillisen version (luku 3.4). Molemmat metodit on toteutettava työn laajuudesta riippumatta. Näitä metodeja tarvitaan oletusrakentajan ja aksessorin tapaan välipisteeseen liittyvässä WETO-testauksessa. Rajapinnasta ja sen metodeista on kerrottu lisää kommenteissa.

Sijoitakokoelma *harjoitustyo*.*kokoelma*-pakkaukseen (luku 4.6).

4.6. Koodin organisointi

Tee luokkahierarkian luokkien ja *OmaLista*-luokan lisäksi omat luokat kokoelmalle (*Kokoelma*, luku 4.5) ja käyttöliittymälle sekä erillinen *main*-metodin sisältävä ajoluokka *Oope2HT*. Ohjelmaan voi lisätä muita luokkia.

Ohjelman sovelluslogiikkaan ja vuorovaikutukseen riippuva koodi on pidettävä erillään. Kokoelmassa tulee olla vain kokoelman käsittelyyn liittyviä metodeja. Vuorovaikutus käyttäjän kanssa tapahtuu käyttöliittymäluokassa.

*Kokoelma*-luokassa ei saa lukea tietoja käyttäjältä. Tulostamisen pitäisi tapahtua mahdollisimman pitkälti käyttöliittymässä, jolloin logiikka ja vuorovaikutus on eriytetty vielä selvemmin eri luokkiin. Kokoelman metodien toteuttamiseen suositellaan tekniikkaa, jossa tulosteet palautetaan mahdollisuuksien mukaan metodien paluuarvona, Huomaa, että tulosteita merkkijonona palautettaessa merkkijonoon voi lisätä järjestelmäriippumattoman rivinvaihdon *System*-luokan *lineSeparator*-metodilla. Paluuarvoja ei tarvitse käyttää virhekoodien välittämiseen, jos virheisiin reagoidaan poikkeuksilla.

Älä välitä käyttäjän antamia komentoja kokoelman metodeille, vaan käsittele komennot käyttöliittymäluokassa, jotta kokoelman logiikkaa ei turhaan sidota käyttöliittymään.

Ideaalitilanteessa kokoelman ja vuorovaikutuksen tulee olla niin riippumattomia, että tekstipohjaisen käyttöliittymän esimerkiksi vaihtaa graafiseen käyttöliittymään.

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

Kaikki harjoitustyön luokat ja rajapinnat ovat *Oope2HT*-ajoluokkaa lukuun ottamatta *harjoitustyo*-nimisessä pakkauksessa. Ajoluokka on versionhallinnan työhakemistossa, josta löytyvät myös *.git*-hakemisto sekä *.gitignore*- ja *README.md*-tiedostot.

*Dokumentti*-, *Uutinen*- ja *Vitsi*-luokat ovat *harjoitustyo.dokumentit*-nimisessä alipakkauksessa, *OmaLista*-luokka on *harjoitustyo.omalista*-alipakkauksessa ja *Kokoelma*-luokka *harjoitustyo.kokoelma*-alipakkauksessa.

Hervannan kampuksen hallinnoimassa GitLab-palvelussa on annettu projektipohja, joka sisältää yllämainittujen pakkausten hakemistot sekä opettajan antamat *Tietoinen<T>*-, *Ooperoiva<E>*- ja *Kokoava*-rajapinnat *harjoitustyo.apulaiset*-alipakkauksessa [3]. Tähän pakkaukseen ei ole syytä tehdä muutoksia, koska WETO käyttää aina alkuperäistä apulaispakkausta. Harjoitustyö on helpointa aloittaa projektipohja lataamalla, vaikka oman koodin voi liittää pohjaan myös myöhemmin.

Voit tehdä halutessasi lisää alipakkauksia.

Kuvassa 16 on annettu harjoitustyön hakemistojen ja tiedostojen rakenne yllä kuvatulta osin. Pisteet symboloivat pakkauksen vapaavalintaista rakennetta.

Ohjelman kääntäminen komentoikkunassa on helpointa versionhallinnan työhakemistosta. Tällöin esimerkiksi *harjoitustyo.dokumentit*-alipakkauksen voi kääntää Windows-käyttöjärjestelmässä komennolla *javac -encoding utf-8 harjoitustyo\dokumentit\\*.java*. Koko ohjelman voi kääntää työhakemistosta käsin komennolla *javac -encoding utf-8 Oope2HT.java*. Mac- ja Linux-järjestelmissä polkuerotin on jakomerkki eikä lähdekoodin merkistöä tarvitse määritellä.

Komentoikkunassa kutsuttu Java-kääntäjä ei osaa aina päätellä oikein mitä kaikkea sen pitäisi kääntää, jotta viimeisimmät muutokset saataisiin mukaan ohjelmaan. Esimerkiksi ajoluokan kääntäminen ei välttämättä johda kaikkien muuttuneiden tiedostojen kääntämiseen. Kääntäjän voi pakottaa kääntämään kaiken lähdekoodin poistamalla tavukooditiedostot ennen käännöstä tai kääntämällä muutetut tiedostot erikseen. Ole varovainen, kun poistat tavukoodia. Käännösongelmia ei pitäisi ilmetä NetBeansin kaltaista kehitysympäristöä käytettäessä.

Kuva 16: Harjoitustyön pakkauksen rakenne.

.git  
.gitignore

Oope2HT.java  
README.md

+---harjoitustyo

|

+---apulaiset

| |---Kokoava.java

| |---Ooperoiva.java

| |---Tietoinen.java

|

+---dokumentit

| |---Dokumentti.java

| |---Uutinen.java

| |---Vitsi.java

|---kokoelma

| |---Kokoelma.java

|

+---omalista

| |---OmaLista.java

...

5. Palautus ja tarkistus

Harjoitustyössä on välipalautuspiste ennen lopullista palautusta. Harjoitustyöstä täytyy palauttaa sunnuntaihin 19.4. klo 23.55 mennessä Javalla toteutettu luokkahierarkia, *OmaLista<E>*-luokka ja *Kokoelma*-luokka ja viisi versionhallintaan tallennettua versiota. Välipalautuksessa tarvitsee toteuttaa vain opettajan antamien rajapintojen abstraktit metodit. Välipisteessä palauteta dokumentaatiota.

Valmis ohjelma ja sen dokumentaatio on palautettava viimeistään keskiviikkona6.5. klo 23.55.

Molemmat palautukset tehdään WETO-järjestelmään, joka tarkistaa ratkaisujen rakennetta ja toiminnallisuutta automaattisesti. Toiminnallisuutta arvioidaan pääosin vertailemalla mallivastauksen ja opiskelijoiden ratkaisujen tulosteita. Tästä syystä edellä annettuja tulostemäärittelyjä on seurattava merkilleen. Palautuksesta kerrotaan tarkemmin WETOssa.

Välipisteessä tarkistuksen pääroolissa on automaattinen testaus. Opettaja lukee koodia välipisteessä vain poikkeustapauksissa eikä välipalautuksen hyväksymisestä tai hylkäämisestä lähetetä erikseen viestiä. Välipalautus on hyväksytty, kun kaikki välipalautuksen testit menevät läpi. WETOn hylkäämä välipalautus on palautettava korjattuna viimeistään lopullisen palautuksen takarajaan mennessä. Lopullinen palautus katsotaan myöhästyneeksi, jos välipalautus on jäänyt korjaamatta.

Valmis ohjelma luetaan kokonaisuudessaan. Lopullisen palautuksen korjausaikaa on useimmiten viikko työn hylkäyksestä. Hylätyn työn voi palauttaa korjattuna opettajan tarkistettavaksi pääsääntöisesti kolme kertaa.

6. Dokumentointi

Harjoitustyöstä kirjoitetaan lopullisen palautuksen yhteydessä dokumentti, jossa tulee olla:

* Kansilehdellä tekijän nimi, opiskelijanumero ja sähköpostiosoite. Sivun keskellä tulisi olla suuremmalla fontilla dokumentin nimi
* 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, josta on kerrottu 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 on annettu 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.

7. Ohjaus

Välipalautukseen liittyvä mikroluokkaohjaus on peruttu COVID-19-pandemian vuoksi. Lopullisen palautuksen mikroluokkaohjausta ei voida todennäköisesti järjestää.

Ohjaus tapahtuu sähköpostilla ja muilla sähköisillä menetelmillä. Ota toistaiseksi yhteyttä kurssin vastuuopettajaan. Kurssin tuntiopettajien ohjauksesta kerrotaan myöhemmin.

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 tätä tehtävänantoa lukemalla ja tutustumalla kurssin verkkosivuilla julkaistaviin esimerkkiajoihin.

8. Arvostelu

Arvosana perustuu harjoitustyön työn laajuuteen ja sen toimintaan testeissä sekä palautusaikatauluun. Laajassa työssä on *freqs*-, *sort*- ja *pprint*-komennot ja opettajan antamien rajapintojen kokonaiset toteutukset eli myös oletusmetodien *laskeFrekvenssit*- ja *lajittele*-metodien korvaukset. Suppealla työllä 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. Pisteet lasketaan viimeisestä ennen takarajaa tehdystä palautuksesta. Korjauskierroksen kautta ei voi korottaa pistemääräänsä.

Kurssin arvosana perustuu alla olevan taulukon mukaan laskettuihin pisteisiin, joita voi saada 0–6 kappaletta. Pisteiden laskennasta on annettu tarkempaa tietoa kurssin WETO-sivujen *Harjoitustyö* | *Pisteet* -kohdassa.

|  |  |  |
| --- | --- | --- |
| 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 hyvästä syystä

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, Korpus, https://fi.wikipedia.org/wiki/Korpus, (Luettu viimeksi 8.4.2020.)
2. J. Laurikkala, Lausekielinen ohjelmointi I -kurssin luentorunko, luku 11, https://www.sis.uta.fi/~laki1/syksy-2019/luennot/luento07/ (Luettu viimeksi 8.4.2020.)
3. J. Laurikkala, Olio-ohjelmoinnin perusteet II -kurssin luentorunko, https://www.sis.uta.fi/~oope/oope2/kevat-2020/luennot/luento03/ (Luettu viimeksi 9.4.2020.)