Java-opas aloitteleville ohjelmoijille

Anni Seppänen

Haaga-Helia ammattikorkeakoulu

Tradenomi

Opinnäytetyö

2024

Tiivistelmä

Tekijä(t)
Anni Seppänen

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Java-opas aloitteleville ohjelmoijille

Sivu- ja liitesivumäärä
22 + 41

Opinnäytetyö Java-opas aloitteleville ohjelmoijille syntyi tarpeesta täyttää aukko nykyisissä ohjelmoinnin itseopiskelumateriaaleissa, joissa usein oletetaan lukijalla olevan aiempaa kokemusta ohjelmoinnista. Työssä käytettiin erittäin suosittua Java-ohjelmointikieltä, joka on edelleen laajasti käytössä sen monipuolisuuden sekä siirrettävyyden ansiosta. Opinnäytetyön tavoite oli laatia opas, jossa ohjelmointia lähestytään helposti ymmärrettävällä tavalla sekä näytetään ohjelmoinnin yhteys arkielämään konkreettisesti. Työn kohderyhmäksi valikoitui ohjelmoinnista kiinnostuneet toisen asteen opiskelijat tai toiselta asteelta valmistuneet. Työtä rajattiin sisältämään vain perusteet Java-ohjelmoinnista.

Työn tietoperustassa käsiteltiin aluksi oppimista yleisesti, ja tämän jälkeen erityisesti ohjelmoinnin oppimista. Java-ohjelmointikielen perusteita käsiteltiin esimerkiksi esittelemällä, miten kirjoitetaan yksinkertainen Hello World -ohjelma. Lisäksi tietoperustassa käytiin läpi Javan tieto- ja ohjelmointirakenteita, sekä virheiden korjaamista. Työn tuottaminen sisälsi suunnittelu-, tutkimus-, tuotanto-, testaus- ja arviointivaiheet, jotka yhdessä muodostivat selkeän prosessin. Kehittämistyön menetelminä käytettiin lähteiden analyysia, haastattelua, vertailuanalyysia sekä iteratiivista kehittämisprosessia.

Opinnäytetö onnistui tavoitteissaan tuottaa käytännönläheinen sekä helposti lähestyttävä oppimateriaali, jota on mahdollista hyödyntää itseopiskelussa. Työn aikana tunnistettiin myös jatkokehittämismahdollisuuksia ja prosessin aikana ilmenneitä haasteita pystyttiin ratkaisemaan rakentavalla tavalla. Työ tarjosi tekijälle arvokkaan oppimismahdollisuuden sekä ammatillista kehittymistä ohjelmoinnin parissa.

Asiasanat
Ohjelmointi, Java, itseopiskelumateriaali

Sisällys

1 Johdanto

Nykypäivänä koodia on kaikkialla. Arkipäiväiset laitteet kuten tietokoneet, puhelimet ja autot kaikki käyttävät jonkinlaista ohjelmakoodia toimiakseen. Ohjelmoinnin opiskelu auttaa ymmärtämään näiden arkipäiväisten laitteiden toimintaa ja siten myös koko ympäröivän maailman digitalisaatio muuttuu helpommin lähestyttäväksi.

Ohjelmoijilla on suuri kysyntä työmarkkinoilla ja ohjelmoinnin opiskelu avaa monia erilaisia mahdollisuuksia erilaisiin työtehtäviin ja -rooleihin eri aloilla (Itewiki 2024). Tällaisia ovat esimerkiksi ohjelmistokehitys, tietotekniikka ja tekoäly. Ohjelmointi on luovaa ongelmanratkaisua ja se kehittää niin loogista päättelykykyä kuin uusien ideoiden innovointitaitoja. Ohjelmointi on hauskaa pulmanratkaisua ja se on lisäksi erittäin palkitsevaa. Konkreettisempaa palkintoa omasta työstä on vaikea edes kuvitella – se hetki, kun koodin saa vihdoin toimimaan halutulla tavalla, on ainutlaatuisen tyydyttävä.

Tämä opinnäytetyö syntyi tarpeesta tarjota selkeä sekä kattava opas ihmisille, jotka ovat kiinnostuneet IT-alasta ja koodaamisesta, mutta eivät oikein tiedä, mistä aloittaa. Nykyiset ohjelmointioppaat ja -materiaalit ovat suunnattu pääsääntöisesti ihmisille, joilla on jo jonkin verran osaamista. Niissä oletetaan usein lukijan esimerkiksi tietävän, mikä on koodieditori ja miten sitä käytetään. Aloittaminen voi tuntua hankalalta, jos ei edes tiedä, mihin koodia tulisi kirjoittaa. Tästä syntyi tarve täyttää aukko nykyisissä oppimateriaaleissa ja luoda opas, joka on sekä kattava että helposti lähestyttävä aloitteleville ohjelmoijille.

Tässä opinnäytetyössä keskitytään Java-ohjelmointikieleen, joka on yksi maailma suosituimmista ja laajimmin käytetyistä ohjelmointikielistä (Stack Overflow 2024). Java syntyi 1990-luvulla vastauksena tarpeeseen kehittää kieli, joka olisi riippumaton käytettävästä laitteistosta ja käyttöjärjestelmästä. Ennen Javaa monet ohjelmointikielet, kuten C ja C++, olivat vahvasti sidoksissa laitteistoon, mikä teki ohjelmien siirrettävyydestä haastavaa. Javan ”write once, run everywhere”-periaate toi merkittävän parannuksen tähän, ja se on edelleen ominaisuus, joka selittää sen suosiota vielä tänäkin päivänä. Javaa käytetään erityisesti suurissa yrityssovelluksissa sekä mobiilikehityksessä (esim. Android-sovelluksissa), mikä tekee siitä relevantin ja hyödyllisen ohjelmointikielen oppia. (Roberts S, 2023).

1.1 Työn tavoitteet, rajaukset ja sisältö

Opinnäytetyön tavoitteena on laatia ohjelmoinnin itseopiskelumateriaali, joka tarjoaa selkeän ja motivoivan polun ohjelmoinnin maailmaan. Oppaan lukijalle pyritään näyttämään, että ohjelmoinnin

opiskelu ei ole pelkästään teoreettista puurtamista, vaan se avaa ovia luovaan ongelmanratkaisuun ja innovointiin. Lukijassa pyritään herättämään uteliaisuutta ja innostusta ohjelmointia kohtaan esimerkiksi motivointitekstillä sekä käyttämällä käytännön esimerkkejä. Ohjelmointitaitoja voi käyttää monilla elämän eri osa-alueilla oli kyse sitten työelämästä, harrastuksista tai omien projektien toteuttamisesta. Oppaan esimerkkien tavoitteena on osoittaa tätä yhteyttä arkielämään konkreettisesti. Yleisesti tavoitteena on luoda sellainen opas, jota olisin itse halunnut lukioikäisenä lukea.

Kohderyhmä opinnäytetyössäni ovat toisella asteella opiskelevat tai jo valmistuneet, jotka ovat ainakin hieman kiinnostuneet ohjelmoinnista. Tällöin voidaan esimerkiksi olettaa, että matemaattinen osaaminen on sellaisella tasolla, että matematiikan alkeet voidaan jättää tämän oppaan ulkopuolelle. Matemaattinen osaaminen antaa myös hyvän pohjan ongelmanratkaisu- ja loogisen päättelykyvyn kehittymiselle. Työtä myös rajattiin kattamaan ohjeet pelkästään Windows-käyttöjärjestelmälle, sillä muuten oppaasta olisi tullut liian pitkä.

Opas toteutetaan pdf-muodossa sisältäen havainnollistavia kuvia. Oppaan alussa on motivointiteksti, jossa kerrotaan, mitä ohjelmointi on ja miksi se on tärkeää sekä hauskaa. Lisäksi käydään läpi oppimista sekä erityisesti ohjelmoinnin oppimista, ja esitellään erilaisia oppimistekniikoita. Sen jälkeen esitellään Java-kielen perusteet ymmärrettävässä muodossa ja käsitellään esimerkiksi ohjelmointirakenteita, kuten ehtolauseita, silmukoita ja funktioita, sekä niiden käyttöä ohjelmoinnissa. Kaikki, mitä oppaassa opiskellaan, on perusteltu. Tällöin lukijalle jää epäselvyyttä siitä, miksi jotain opiskellaan.

Opinnäytetyön tietoperustassa käsitellään erilaisia opiskelumenetelmiä ja -strategioita, jotka voivat auttaa omaksumaan uutta tietoa tehokkaammin ja joita sovelletaan opasta laatiessa. Oppaan tarkoituksena on tarjota lukijalle vaihtoehtoja perinteiselle pänttäämiselle ja rohkaista lukijaa löytämään oman tapansa oppia. Työn tietoperustassa käydään myös läpi Javan perusteita ja tätä tietoperustaa käytetään pohjana oppaan luomisessa.

Oppaan luettuaan lukija on oppinut perusteet ohjelmoinnista, edistänyt ongelmanratkaisu- ja innovointikykyään kuin myös lisännyt ymmärrystään nykypäivän digitalisoitunutta maailmaa kohtaan. Yhteiskunnallisesti ohjelmointitaitojen lisääminen edistää esimerkiksi teknologista kehitystä, mistä voi olla hyötyä eri aloilla kuten terveydenhuollossa, teollisuudessa tai viihteessä, esimerkiksi pelisuunnittelussa.

Keskeiset käsitteet

Ohjelmointi

Sellaisten ohjeiden kirjoittamista, joita tietokone ymmärtää ja voi suorittaa.

Ohjelma

Joukko ohjeita, jotka tietokone suorittaa jonkin tietyn tehtävän suorittamiseksi. Esimerkiksi laskin on ohjelma.

Java

suosittu ohjelmointikieli, jolla ohjelmia kirjoitetaan.

Koodieditori

ohjelmisto, jolla kirjoitetaan, muokataan ja tallennetaan koodia. Esimerkiksi VSCode.

Ohjelmointirakenteet

koodin organisoimista siten, että se on selkeää sekä loogista. Esimerkiksi ehtolauseet ja silmukat.

Tietorakenteet

tapa varastoida ja organisoida dataa siten, että sitä voi käyttää tehokkaasti. Esimerkiksi listat, jonot ja hajautustaulut.

2 Tietoperusta

Opinnäytetyön tietoperusta koostuu yleisesti oppimisen sekä tarkemmin ohjelmoinnin oppimisen teorioista ja käytännöistä. Tässä luvussa käsitellään oppimista yleisesti, sekä syvennytään ohjelmoinnin ja erityisesti Java-ohjelmoinnin oppimiseen. Tämä tieto auttaa ymmärtämään, millaisista lähtökohdista opinnäytetyö on suunniteltu ja toteutettu. Lisäksi käsitellään Java-kielen perusteita, joiden pohjalta opas on toteutettu.

2.1 Oppiminen yleisesti

Oppiminen on monimutkainen prosessi, joka vaatii aikaa, kertausta ja keskittymistä. Muistijälkien syntyminen aivoihin ei tapahdu automaattisesti, vaan oppijan on aktiivisesti työskenneltävä uuden tiedon omaksumiseksi. Tämä prosessi on yksilöllinen ja siihen vaikuttavat monet tekijät, kuten tarkkaavaisuus, motivaatio ja tunteet. Oppimisessa toistot ovat keskeisessä roolissa: tieto jää mieleen paremmin, kun sitä käsitellään useaan kertaan eri yhteyksissä. Myös esimerkiksi liikkuminen, toisten kanssa keskustelu, päiväunet ja melun vähentäminen ovat asioita, jotka parantavat oppimista. (Brendtson, 17.4.2024).

Oppimisen tukemiseksi on tärkeää hyödyntää erilaisia oppimisstrategioita ja tekniikoita. Yksi tunnetuimmista on Pomodoro-tekniikka, jossa opiskellaan intensiivisesti 25 minuutin jaksoissa, joiden välillä pidetään lyhyitä, viiden minuutin taukoja. Näitä ns. tomaatteja (Pomodoro) toistetaan neljä kertaa, jonka jälkeen pidetään 20–30 minuutin pidempi tauko. Tämä auttaa ylläpitämään keskittymiskykyä ja vähentää prokrastinaation eli tehtävien lykkäämisen riskiä. (Licence to Fail 2.5.2020, 1:10-2:06 min)

Prokrastinaatio tarkoittaa tehtävien tai asioiden siirtämistä myöhemmälle, vaikka se olisi monessa tilanteessa parempi tehdä heti. Ihmisen aivot suosivat sellaisia asioita, joista ne saavat nopeasti mielihyvää. Usein vasta siinä vaiheessa, kun viimeinen palautuspäivä lähestyy uhkaavasti, saa ihminen itsensä aloittamaan tehtävän. (Clear s.a.) Prokrastinaatio voi johtaa siihen, että tehtävien aloittaminen lykkääntyy niin kauan, että niiden suorittaminen jää viime hetkeen, jolloin oppiminen on usein vähemmän tehokasta.

Oppimistyyleistä on keskusteltu paljon, mutta niiden soveltaminen ei aina takaa parasta oppimistulosta. Vaikka tietyt oppimistyylit voivat tuntua luonnollisemmilta, tehokkaampi oppiminen voi edellyttää toisenlaisten menetelmien käyttöä. Opiskeluvinkkeinä suositellaan, että opiskeluaika jaetaan pienempiin osiin, tehdään hyviä muistiinpanoja ja opetetaan opittua asiaa muille tai ääneen itselle.

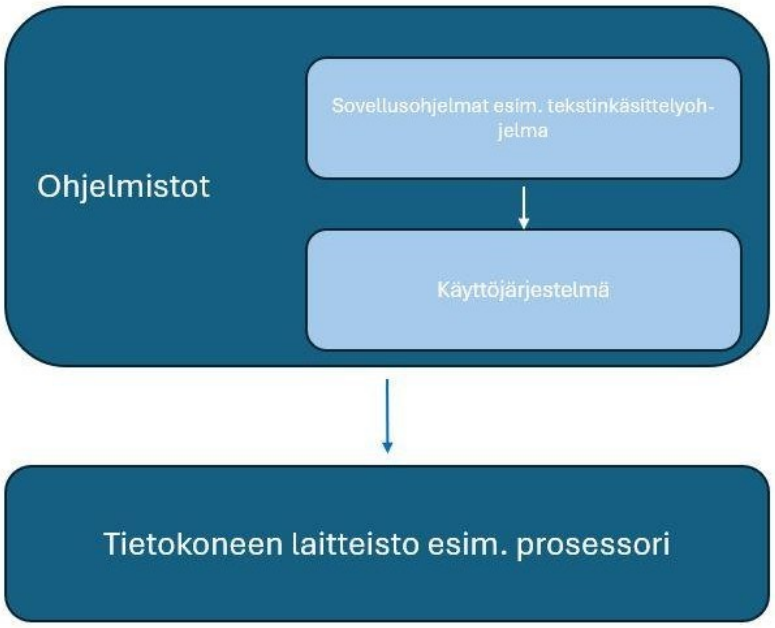
Näillä keinoilla varmistetaan, että opittu tieto siirtyy paremmin pitkäkestoiseen muistiin ja on siten myöhemmin hyödynnettävissä. (Opiskelukoulu s.a.).

2.2 Ohjelmoinnin oppiminen

Ohjelmoinnin opiskeluun kuuluu koodin kirjoittamisen lisäksi jonkinlainen ymmärrys siitä, miten tietokoneet toimivat. Tässä luvussa käydään ensin läpi perusasioita tietokoneista, jonka jälkeen siirrytään Java-ohjelmoinnin teoriaosuuteen.

Tietokoneen perusosia ovat prosessori, väliaikainen muisti, kovalevy sekä ulkoiset laitteet, kuten näyttö ja hiiri. Prosessoria pidetään usein tietokoneen aivoina. Se suorittaa suurimman osan tietokoneen prosesseista, joita tietokone tarvitsee toimiakseen. Väliaikainen muisti säilyttää tietoa vain sen ajan, kun tietokone on päällä. Kovalevy taas sisältää tietoja, jotka täytyy säilyttää myös silloin, kun tietokone suljetaan. (GeeksForGeeks 2023).

Käyttöjärjestelmät ovat ohjelmistoja, jotka mahdollistavat kommunikaation tietokoneen sovelluksien ja laitteiston välillä. Ne suorittavat yksinkertaisia tehtäviä, kuten tiedostojen- ja muistinhallintaa. Suosittuja käyttöjärjestelmiä ovat esim. Windows, MacOS ja Linux. (Tutorialspoint s.a.) Kuvassa 1 on havainnollistettu käyttöjärjestelmän ja tietokoneen laitteiston yhteyttä.



Kuva 1. Tietokoneen eri osien yhteys

Käyttöliittymällä tarkoitetaan sitä osaa, jonka avulla käyttäjä voi syöttää tai vastaanottaa tietoja. Graafisessa käyttöliittymässä käyttäjä suorittaa erilaiset toiminnot hiirellä klikkailemalla esimerkiksi erilaisia kuvakkeita. (Bautomo s.a.). Merkkipohjaisella käyttöliittymällä komennot annetaan tekstimuodossa komentoriviltä.

Komentorivi, toiselta nimeltään esim. command line, terminal tai console, on tärkeä ohjelmoijan työkalu. Monille ohjelmille, joita ohjelmoijat käyttävät, ei välttämättä ole olemassa graafista käyttöliittymää, ja usein merkkipohjaisen käyttöliittymän käyttäminen on tehokkaampaa. (FreeCodeCamp 2022). Myös esimerkiksi versionhallintaa tehdään usein komentoriviltä. Tämän takia ohjelmoijien on tärkeää tietää perusasiat komentorivistä.

Koodia voi kirjoittaa periaatteessa millä vain tekstinkäsittelyohjelmalla, esimerkiksi notepadilla. Koodieditorit kuitenkin mahdollistavat monia hyödyllisiä ominaisuuksia, jotka helpottavat koodin kirjoittamista. Ne esimerkiksi tarkistavat syntaksivirheet (“kirjoitusvirheet”), tekevät koodista luettavampaa esimerkiksi sisentämällä koodia automaattisesti sekä antavat ehdotuksia päätteistä. Esimerkiksi kun kirjoitat “Sys”, koodieditori luultavasti ehdottaa “System”. Nämä kaikki nopeuttavat koodin kirjoitusta. (Noble Desktop s.a.).

Visual Studio Code on koodieditori, joka on ilmainen ja helppo ottaa käyttöön nopeasti. Sillä voi kirjoittaa koodia monella eri kielellä, Javan lisäksi esim. Python ja JavaScript. VSCode korostaa avainsanoja eri väreillä, mikä auttaa hahmottamaan erilaisia koodaamisen malleja ja nopeuttaa oppimista. Se myös korjaa yleisiä virheitä automaattisesti. (Visual Studio Code s.a.).

2.3 Java

Java on erityisen sopiva aloittelijoille sen yksinkertaisuuden ansiosta. Verrattuna esimerkiksi C:hen, on Javan syntaksi helpompilukuista. Tämä tekee ohjelmoinnin perusteiden oppimisesta sujuvampaa. Java on myös vahvasti tyypitetty kieli, ja tämä auttaa havaitsemaan virheitä helpommin. Myös esimerkiksi laaja dokumentaatio sekä yhteisön tuki ovat tekijöitä, jotka tekevät Javasta erityisen soveltuvan aloittelijalle. Kysymyksiin löytyy helposti vastaus verkosta, mikä auttaa oppimaan tehokkaammin. (Medium 2020).

Java on edelleen yksi suosituimmista ohjelmointikielistä huolimatta siitä, että sitä pidetään ehkä joskus vanhanaikaisena ja joidenkin mielestä käytöstä poistuneena. Java on ollut käytössä 1990-luvulta lähtien, joten se on kypsä, mutta jatkuvasti kehittyvä sekä päivitettävä kieli. Java on alustariippumaton, eli koodin voi kirjoittaa kerran ja ajaa useilla eri alustoilla ilman uudelleenkääntämistä,

mikä tekee siitä joustavan kielen. (UNext 2022). Näistä syistä Java on edelleen sekä aloittelijaystävällinen että relevantti kieli, jota opiskella.

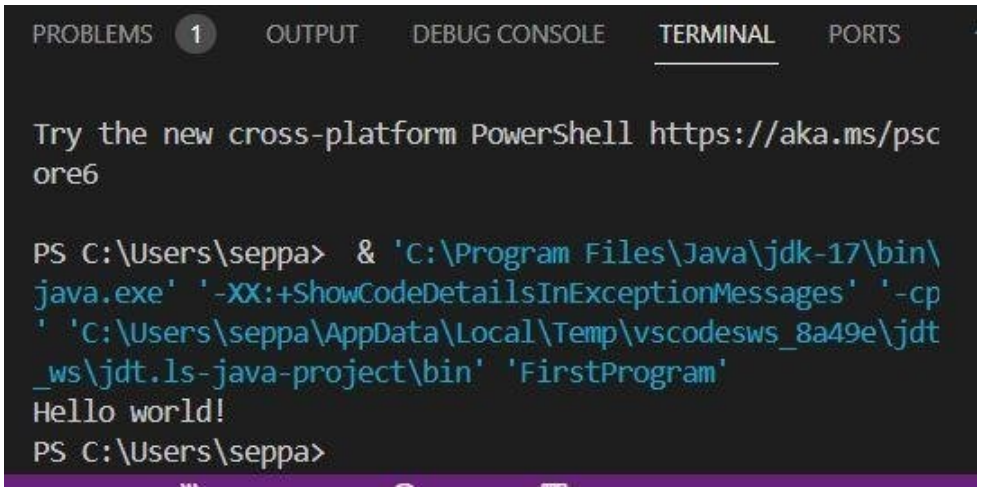
Seuraavissa alaluvuissa käydään läpi, miten rakennetaan yksinkertainen Java-ohjelma. Tämän jälkeen käsitellään tietorakenteita, perusohjelmointirakenteita sekä virheiden korjaamista. Opas on kirjoitettu tämän tietoperustan pohjalta.

Oppaan sisällysluettelon ja rakenteen ideoinnissa on hyödynnetty ChatGPT 3.5 -kielimallia. Syötteenä käytettiin: ”Keksi sisällysluettelo Java-oppaaseen, joka on suunnattu aloittelijoille”.

2.3.1 Hello World

Kuvassa 2 on esimerkki yksinkertaisesta Java-ohjelmasta. Rivillä 1 määritellään luokka ja se on nimeltään FirstProgram. Luokkien nimet kirjoitetaan sopimuksen mukaan isolla alkukirjaimella. Toisella rivillä määritellään päämetodi (main method). Päämetodi on ohjelman aloituspiste, josta ohjelman suorittaminen aloitetaan. Päämetodin sisälle kirjoitetaan komentoja. Tässä ohjelmassa on yksi komento System.out.println, joka tulostaa koodieditorin konsoliin halutun tekstin. Tässä tapauksessa teksti, joka tulostetaan, on ”Hello world!”. Teksti, joka halutaan tulostaa, tulee kirjoittaa sulkuihin, ””-merkkien väliin. Lopuksi vielä lisätään puolipiste (;), joka päättää jokaisen komennon. (Liang 2015, 12-13).

Kuva 2. Yksinkertainen Java-ohjelma



Kuva 3. Tuloste Visual Studio Coden konsolissa

Ohjelman sisälle voisi vielä lisätä kommentteja, jotka helpottavat koodin ymmärtämistä. Kommentit lisätään Javassa //-merkkien jälkeen, jos kyseessä on yksirivinen kommentti. Jos halutaan lisätä pidempi kommentti, joka on enemmän kuin yhden rivin pituinen, tulee kommentti lisätä /* */-merkkien väliin. Kuvassa 4 on havainnollistettu erilaisten kommenttien käyttöä. (Liang 2015, 12-13).

Kuva 4. Java-ohjelman erilaiset kommentit.

2.3.2 Tietorakenteet Java-ohjelmoinnissa

Tietorakenteet ovat keskeinen osa ohjelmointia. Tietorakenteilla varastoidaan ja organisoidaan dataa, jotta sitä voidaan käyttää ja päivittää tehokkaasti. Eri tietorakennetyyppejä ovat esimerkiksi muuttujat, taulukot, listat, hashmapit ja hashsetit.

Muuttujat ovat ohjelmoinnissa käytettäviä säilöjä, joihin tallennetaan tietoa, kun ohjelmaa suoritetaan. Java-ohjelmoinnissa muuttujilla on aina tietotyyppi, joka tarkoittaa, että se voi tallentaa vain tietynlaista tietoa, esimerkiksi kokonais- (int) tai liukulukuja (double), merkkejä (char), merkkijonoja (string) tai totuusarvoja (boolean). (Machinet 2024).

Taulukko (array) on saman tyyppisten alkioiden kokoelma, joka sijaitsee peräkkäisissä muistipaikoissa. Taulukko tallentaa vain ennalta määrätyn määrän alkioita, eli sillä on määrätty koko, joka ei voi muuttua. Javan taulukot ovat indeksipohjaisia, eli ensimmäinen alkio sijaitsee indeksissä 0, toinen indeksissä 1 jne. Taulukon etuna on tehokas tiedon haku ja järjestäminen, mutta haittapuolena on se, että taulukko ei voi kasvaa ajon aikana. (Tpoint Tech s.a.).

Lista (arraylist) edustaa järjestettyä arvojoukkoa, jossa sama arvo voi esiintyä useammin kuin kerran. Se voi myös kasvaa tai kutistua dynaamisesti, kun siihen lisätään tai sieltä poistetaan alkioita. Samoin kuin taulukoissa, listan alkioihin pääsee käsiksi niiden indekseillä, alkaen nollasta. (Baeldung 2024).

Hashset ja hashmap ovat tietorakenteita, jotka käyttävät hajautusta tietojen tallentamiseen ja hakemiseen. Hajauttamisen ansiosta, eli tietojen tallentaminen hash-koodin perusteella, lisääminen, poistaminen ja hakeminen on tehokasta. Hashsetillä ja hashmapilla on kuitenkin merkittäviä eroja. Hashset on kokoelma, joka tallentaa ainoastaan uniikkeja elementtejä, eli se ei salli kaksoiskappaleita. Hashmap tallentaa tiedot avain-arvo-pareina, jolloin tietoa voi hakea avaimen perusteella. Hashmapissa avaimet ovat uniikkeja, mutta arvot voivat toistua. (Medium 2023).

2.3.3 Javan perusohjelmointirakenteet

Ohjelmoinnissa operaatiota, ehtolauseita ja silmukoita käytetään erilaisten ongelmien ratkaisemiseen. Operaatioita ovat esimerkiksi aritmeettiset, loogiset ja vertailevat operaatiot. Ehtolauseita ovat esimerkiksi if-, else if- ja else -lauseet, ja niiden avulla voidaan tehdä päätöksiä ohjelmien sisällä. Silmukat kuten for- ja while-silmukka mahdollistavat toistuvien tehtävien suorittamisen tehokkaasti. Näiden elementtien hallinta on tärkeää sujuvan ohjelmakoodin kirjoittamisessa.

Aritmeettisia operaatioita ovat esimerkiksi yhteenlasku (+), erotus (-), kertolasku (*) ja jakolasku (/) (Liang 2015, 46). Javassa loogisia operaatiota ovat esimerkiksi AND ja OR. AND-operaattori palauttaa tosi, kun molemmat ehdot ovat tosia. Muuten se palauttaa epätosi. AND-operaatiota merkataan Javassa notaatiolla &&. OR-operaattori palauttaa tosi, jos yksikin annetuista ehdoista on tosi. Se palauttaa epätosi vain, jos kaikki ehdot ovat epätosia. OR-operaatiota merkataan notaatiolla ||. (Scaler, 2024). Vertailevia operaatioita ovat yhtäsuuruus (==), erisuuruus (!=), pienempi (tai yhtä suuri) kuin (< ja <=) ja suurempi (tai yhtä suuri) kuin (> ja >=) (W3Schools s.a.).

Ehtolauseet mahdollistavat ohjelman ohjaamisen eri lohkoihin, mikäli tietty ehto toteutuu. Jos ehtolauseessa oleva ehto toteutuu, toteutetaan ehtolauseen sisällä oleva koodi. Jos taas ehto ei toteudu, siirrytään seuraavaan ehtoon (else if) ja mikäli sekään ei toteudu, siirrytään else-lohkoon. Else-lohko suoritetaan aina siinä tapauksessa, jos mikään edellä olevista ehdoista ei toteudu. (Liang, 78-83).

Silmukat mahdollistavat saman koodinosan toiston useita kertoja ilman, että sitä täytyy kirjoittaa monta kertaa uudestaan. For-silmukka toistaa tiettyä lohkoa yhtä monta kertaa, kun sille alussa kerrotaan (Liang, 158). While-silmukka taas toistaa lohkoa, kunnes alussa määritelty ehto on epätosi. (Liang, 170-171).

2.3.4 Virheiden korjaaminen

Virheenkorjaus kuuluu ohjelmistokehitykseen olennaisesti. On tärkeää oppia lukemaan erilaisia virhetuloksia ja ymmärtää, miten niitä korjataan.

Javassa on sekä virheitä (errors), että poikkeuksia (exceptions), ja ne eroavat toisistaan merkittävästi. Virheet ovat erilaisia ongelmia, joiden ilmaantuessa koodia ei voi enää suorittaa. Tällaisia ovat esimerkiksi syntaksivirheet, ”SyntaxError”. Poikkeukset taas ovat tapahtumia, jotka johtuvat ohjelmakoodin ongelmista, kuten virheellisistä syötteistä. Koodi voi silti jatkaa pyörimistään, jos poikkeus käsitellään oikein. Esimerkiksi ”IllegalArgumentException” huomaa, jos syöte on virheellinen. Tämä on mahdollista ohittaa try-catch-lohkolla, joka ilmoittaa käyttäjälle, että poikkeus on tapahtunut. (Shiksha Online, 2024).

3 Empiirinen osa

Tämä opinnäytetyö tähtää aloittelijoille suunnatun ohjelmointioppaan tuottamiseen. Vaikka työllä ei ollut ulkoista toimeksiantajaa, se palvelee laajaa yleisöä. Esimerkiksi ohjelmoinnin korkeakouluopiskelijat, itseoppijat ja mahdolliset ohjelmoinnin peruskursseja tarjoavat oppilaitokset, kuten lukiot, voivat hyötyä tästä työstä. Tämä työ toimii sekä omana oppimisprojektinani samoin kuin konkreettisena oppimateriaalina ohjelmointia aloitteleville henkilöille.

3.1 Ongelmat ja tarpeet, joihin opinnäytetyö vastaa

Java on yksi maailman suosituimmista ohjelmointikielistä, mutta sen opiskelu voi olla haastavaa erityisesti aloittelijoille, jotka eivät ole aiemmin ohjelmoineet. Tähän tarpeeseen opinnäytetyö vastaa tarjoamalla selkeän, strukturoidun ja pedagogisesti mielekkään oppaan, joka opastaa alusta lähtien Java-ohjelmoinnin maailmaan. Tämänkaltaisille oppaille on jatkuvaa kysyntää, sillä ohjelmoinnin kurssit ovat suosittuja sekä toisella asteella että korkeakouluissakin. Opinnäytetyö vastaa myös tarpeeseen saada oppimateriaalia, joka on kirjoitettu selkeällä kielellä ja joka tarjoaa konkreettisia arkielämän esimerkkejä oppimisen tueksi.

3.2 Kohderyhmä

Työn kohderyhmä on toisen asteen käyneet tai toisella asteella opiskelevat, ohjelmoinnista kiinnostuneet ihmiset. Tein tämän rajauksen varmistaakseni, että matematiikan alkeet ovat hallussa, jolloin ne voidaan jättää opinnäytetyön ulkopuolelle. Matemaattinen osaaminen antaa myös hyvän pohjan ongelmanratkaisu- sekä loogisen päättelykyvyn kehittymiselle, joita molempia tarvitaan ohjelmoinnissa.

3.3 Tuottamisen rajoittavat tekijät

Työn tuottamiseen liittyy tiettyjä rajoittavia tekijöitä. Esimerkiksi aika on aina rajallista. Huomasin nopeasti työn edetessä, että rajaukseni oli aivan liian laaja ja että sitä tulisi kiristää merkittävästi. Kaikkea oppimaani ei millään saa yhden opinnäytetyön kokoiseksi kokonaisuudeksi, joten esimerkiksi tehtävät jäivät kokonaan opinnäytetyön ulkopuolelle.

Tuottamista rajoitti ajan lisäksi esimerkiksi eri käyttöjärjestelmien eroavuudet. Totesin, että tämä opinnäytetyö on pelkästään Windows-käyttöjärjestelmälle, sillä ohjeiden tekeminen myös esimerkiksi MacOS:lle olisi tehnyt oppaasta liian laajan.

3.4 Laatukriteerit

Koen, että lopputuotos on hyvä, kun se on selkeä, ymmärrettävä, käytännönläheinen, ajantasainen, monipuolinen, kattava ja pedagogisesti hyvin rakennettu. Nämä kriteerit varmistavat työn laadun ja hyödyllisyyden sekä vastaavuuden siihen tarpeeseen, johon se on suunniteltu.

Oppaan tulee olla selkeästi ja ymmärrettävästi kirjoitettu. Tämä tarkoittaa, että asiat on esitetty loogisessa järjestyksessä ja jokainen luku ja osa-alue esitetään siten, että aloittelija ymmärtää sen helposti. Käytännönläheisyys saavutetaan esittelemällä käytännön esimerkkejä, joiden avulla lukija näkee ohjelmoinnin yhteyden arkielämän ongelmien ratkaisuun. Oppaan tulee olla ajan tasalla, ja tämä on otettu huomioon esimerkiksi huomioimalla Javan uusin versio sekä nykyaikaiset ohjelmointikäytännöt. Olen pyrkinyt rajauksenkin jälkeen tekemään oppaasta monipuolisen ja kattavan, jotta lukija saa hyvän pohjan mahdollisille jatko-opiskeluille. Tärkeimmät peruskäsitteet, kuten muuttujat, tietorakenteet ja perusohjelmointirakenteet, on käyty läpi. Opas on laadittu siten, että se etenee vaiheittain vaikeusasteen kasvaessa, jolloin se on myös pedagogisesti hyvin rakennettu. Tämä auttaa lukijaa kehittämään taitojaan asteittain ja vahvistaa oppimista.

3.5 Tuottamisen vaiheet

Tuottaminen koostui suunnittelu-, tutkimus-, tuotanto-, testaus- ja arviointivaiheesta. Nämä vaiheet esitellään yksityiskohtaisemmin seuraavissa aliluvuissa.

3.5.1 Suunnitteluvaihe

Suunnitteluvaiheeseen kuului tarpeiden määrittelyä, kohderyhmän analyysiä sekä oppaan rakenteen suunnittelua.

Tarpeiden määrittely alkoi kohderyhmän ja oppaan tavoitteen määrittelyllä. Koska oppaan kohderyhmä koostuu aloittelijoista, tarvittiin yksityiskohtainen suunnitelma siitä, mitä perustaitoja he tarvitsevat Java-ohjelmoinnissa. Tarpeiden määrittelyyn kuului myös päätös siitä, että oppaan tulee olla helposti lähestyttävä, selkeä ja sen tulee sisältää käytännön esimerkkejä.

Kohderyhmän analyysissä tutkittiin, millaiset lähtötiedot aloittelijoilla on ja miten heidän oppimistaan voidaan parhaiten tukea. Tämä auttoi määrittelemään, kuinka perusteellisesti eri aiheita tulisi käsitellä. Esimerkiksi päätettiin, että Visual Studio Coden asennus ja käyttöohjeet ovat tarpeellisia, koska monille aloittelijoille ympäristön asennus voi olla haaste.

Suunnitteluvaiheen lopuksi luotiin oppaan rakenteellinen runko, jossa aihealueet järjestettiin loogiseen ja pedagogisesti järkevään järjestykseen. Tämä rakenteellinen suunnittelu loi perustan oppaan myöhemmille tuotantovaiheille. Tämän vaiheen aikana syntyi ensimmäinen luonnos oppaan sisällysluettelosta.

3.5.2 Tutkimusvaihe

Tutkimusvaihe koostui materiaalin keruusta, ohjelmointiympäristön sekä pedagogisen rakenteen valinnasta.

Materiaalin keruuseen sisältyi kirjallisuuden ja muiden Java-ohjelmoinnin oppimateriaalien tutkiminen. Näiden lähteiden analysointi auttoi varmistamaan, että oppaan sisältö oli ajan tasalla ja vastasi parhaita käytäntöjä. Tutkin esimerkiksi sitä, millaisia esimerkkejä käytetään yleisesti ohjelmoinnin perusoppaissa ja miten niitä voisi soveltaa tässä oppaassa.

Visual Studio Code valittiin oppaassa käytettäväksi ohjelmointiympäristöksi, sillä se on suosittu, ilmainen ja helppokäyttöinen IDE, joka sopii hyvin aloittelijoille. Lisäksi ympäristö tukee laajasti eri ohjelmointikieliä ja tarjoaa runsaasti dokumentaatiota.

Pedagogisen rakenteen valintavaiheessa päätettiin oppaan pedagogisesta rakenteesta. Valittiin vaiheittainen lähestymistapa, jossa aloittelija siirtyy perusasioista kohti monimutkaisempia käsitteitä.

3.5.3 Tuotantovaihe

Tuotantovaihe koostuu sisällön tuotannosta, esimerkkien luonnista sekä sisällön tarkistuksesta ja editoinnista. Nämä vaiheet menivät osittain päällekkäin, sillä oli hyvä välillä tarkistaa ja editoida sisältöä ennen kuin se oli kokonaan valmis.

Sisällön tuottaminen alkoi ensimmäisen luonnoksen kirjoittamisella. Jokainen aihealue käsiteltiin järjestyksessä, ja huomio kiinnitettiin siihen, että selitykset olivat selkeitä ja esimerkit helppoja ymmärtää. Tässä vaiheessa syntyi ensimmäinen versio oppaan tekstisisällöstä.

Jokaisen luvun yhteyteen lisättiin esimerkkejä, joiden avulla lukija voi harjoitella oppimaansa. Näiden esimerkkien valinta perustui käytännön soveltuvuuteen ja niiden helppoon ymmärrettävyyteen. Oppaan ajan käytettiin samaa budjetointisovellusesimerkkiä, jolloin lukija hahmottaa, millaisista erilaisista osista tällainen sovellus voidaan rakentaa.

Kun oppaan ensimmäinen versio oli valmis, se käytiin huolellisesti läpi ja muokattiin. Tässä vaiheessa opasta rajattiin merkittävästi, sillä se oli paisumassa liian laajaksi. Lisäksi tarkistettiin esimerkkien toimivuus ja varmistettiin, että sisältö oli johdonmukaista ja loogista. Sisältöä tarkistettiin ja editoitiin myös muissa vaiheissa, iteratiivisesti.

3.5.4 Testausvaihe

Opasta testattiin kahdella kohderyhmään kuuluvalla henkilöllä. Henkilöt lukivat opasta, asensivat koodieditorin ja kirjoittivat koodia editorissa. Kerättiin palautetta siitä, miten hyvin opas vastasi heidän tarpeisiinsa ja missä kohdin oppaassa oli vielä parantamisen varaa.

Positiivista palautetta sai asennusohje, jonka avulla henkilöt saivat molemmat helposti asennettua koodieditorin tietokoneelleen. Myös esimerkkien johdonmukaisuudesta pidettiin. Opas oli kohderyhmän mielestä kokonaisuutena hyvä ja selkeä.

Oppaan kieli sai osittain kritiikkiä. Joissain kohdissa asiat oli selitetty liian monimutkaisesti. Saadun palautteen perusteella huomattiin, että on selkeämpää jättää kertomatta tiettyjä asioita, mikäli ne eivät ole relevantteja opittavan asian kannalta. Esimerkiksi luokkien ja metodien määritelmät jätettiin oppaasta pois, sillä palautteen perusteella ne hämmensivät enemmän kuin auttoivat ymmärtämään kokonaisuutta. Loppujen lopuksi huomattiin, että nämä asiat eivät mahdu tämän opinnäytetyön kokoiseen kokonaisuuteen, joten ne jätettiin kokonaan rajauksen ulkopuolelle.

3.5.5 Arviointivaihe

Lopuksi arvioitiin oppaan laatua suhteessa asetettuihin laadullisiin kriteereihin. Tarkasteltiin, kuinka hyvin opas onnistui saavuttamaan tavoitteensa ja miten hyvin se palveli kohderyhmäänsä. Arvioinnin perusteella tehtiin viimeiset korjaukset ennen lopullista julkaisua.

3.6 Kehittämistyön menetelmät

Kehittämistyön menetelminä käytettiin lähteiden analyysia (Ojasalo, Moilanen & Ritalahti 2015, alaluku Dokumenttianalyysi), vertailuanalyysia, haastattelua (Ojasalo ym. 2015, alaluku Haastattelu) ja iteratiivista kehittämisprosessia.

Lähteiden analyysi auttoi varmistamaan, että oppaan sisältö on tieteellisesti ja pedagogisesti perusteltua. Analyysin avulla luotiin pohja oppaan sisällölle ja löydettiin asiat, joita oppaassa tulisi käsitellä.

Vertailuanalyysi eli benchmarking-menetelmä tarkoittaa oman työn vertailua muihin vastaaviin tuotoksiin (Ojasalo ym. 2015, 186). Tämän pohjalta olen suunnitellut ja muokannut työtäni ottaen huomioon jo olemassa olevat oppaat ja materiaalit.

Haastattelulla kerättiin palautetta kohderyhmältä, mikä auttoi tunnistamaan mahdolliset heikkoudet oppaassa. Tämä menetelmä mahdollisti käytettävyyden parantamisen ennen julkaisua. Tästä menetelmästä kerrotaan lisää kohdassa 3.5.4 Testausvaihe.

Iteratiivinen kehittäminen tarkoittaa, että oppaan tuotanto eteni vaiheittain ja jokaisessa vaiheessa sisältöä tarkistettiin ja paranneltiin (Martins, 14.1.2024). Tämä tapa varmistaa, että oppaan lopullinen versio on laadukas ja täyttää käyttäjien tarpeet.

4 Pohdinta

Pohdinta-luvussa käydään läpi opinnäytetyön tuotoksen ajankohtaisuutta, tarpeellisuutta sekä hyödynnettävyyttä, työn onnistuneisuutta, haasteita sekä jatkojalostusmahdollisuuksia. Lisäksi pohditaan esimerkiksi omaa oppimista prosessin aikana.

4.1 Tuotoksen ajankohtaisuus, tarpeellisuus ja hyödynnettävyys

Opinnäytetyön tuotos on ajankohtainen, sillä ohjelmointitaitojen kysyntä kasvaa jatkuvasti monilla eri aloilla. Digitalisaatio ja teknologian nopea kehitys ovat tehneet ohjelmointiosaamisesta tärkeän taidon monille, ei pelkästään IT-alan ammattilaisille. Tämä tekee oppaasta relevantin ja ajankohtaisen erityisesti niille, jotka haluavat päästä nopeasti ohjelmoinnin maailmaan.

Java on edelleen ohjelmointikielenä relevantti, mikä lisää oppaan ajankohtaisuutta ja hyödynnettävyyttä. Vaikka uusia ohjelmointikieliä on kehitetty Javan jälkeen, on se säilyttänyt asemansa yhtenä maailman suosituimmista ja käytetyimmistä kielistä.

Oppaan tarpeellisuus perustuu siihen, että monet aloittelevat ohjelmoijat kohtaavat haasteita ohjelmoinnin alussa, koska saatavilla oleva materiaali voi olla liian laajaa tai syvällistä heidän tarpeisiinsa nähden. Tämä opas täyttää tämän selkeän aukon tarjoamalla tiiviin ja käytännönläheisen johdatuksen koodaamisen perusteisiin, mikä auttaa lukijoita pääsemään helposti kiinni asiaan. Se on suunniteltu niin, että käyttäjät voivat oppaan avulla päästä suoraan ohjelmoinnin käytäntöön.

4.2 Työn onnistuneisuus

Empiirisessä osassa on mainittu työn onnistuneisuuden kriteereitä. Niitä olivat selkeys ja ymmärrettävyys, käytännönläheisyys, ajantasaisuus, monipuolisuus ja kattavuus sekä pedagogisesti hyvä rakenne. Tässä luvussa arvioidaan näiden kriteereiden toteutumista.

Opas on selkeästi ja loogisesti kirjoitettu, mikä varmistaa, että myös aloittelijat voivat helposti seurata ja ymmärtää sisältöä. Tämä on tärkeää, sillä monimutkaisempien ohjelmointikonseptien selittäminen on usein haasteellista. Koen, että asiat onnistuttiin selittämään yksinkertaisella kielellä.

Käytännön esimerkit ovat olennainen osa oppaan sisältöä, ja ne tukevat teoreettisten käsitteiden omaksumista. Oppaassa käytettiin budjetointisovellusta esimerkkinä, ja sama esimerkki pysyi läpi oppaan, jolloin lukija huomaa, miten ohjelmointia voidaan käyttää arkielämän ongelmien

ratkaisemiseen. Tämä lähestymistapa tekee oppaasta paitsi opetuksellisesti tehokkaan, myös mielestäni motivoivan lukijalle.

Oppaan ajantasaisuus on varmistettu huomioimalla Javan uusin versio ja nykyiset ohjelmointikäytännöt. Tämä on erityisen tärkeää, sillä vanhentunut tieto voi nopeasti muuttua ohjelmoinnissa hyödyttömäksi. Oppaaseen onnistuttiin sisällyttämään viimeisimmät käytännöt, jolloin se on luotettava sekä ajankohtainen materiaali.

Vaikka oppaan sisältöä jouduttiin rajaamaan, on se silti monipuolinen ja kattava, ottaen huomioon sen, mitä aloittelijan tulisi hallita. Tärkeimmät peruskäsitteet, kuten muuttujat, tietorakenteet ja perusohjelmointirakenteet, on käsitelty perusteellisesti. Tämä antaa lukijalle vahvan perustan jatko-opinnoille näin halutessaan.

Oppaan pedagoginen rakenne on onnistunut, sillä se etenee vaiheittain vaikeusasteen kasvaessa. Tämä mahdollistaa oppimisen asteittain ja auttaa lukijaa kehittymään systemaattisesti. Tämä rakenne tukee oppimista ja lukija saa selkeän käsityksen ohjelmoinnin perusteista, edeten askel askeleelta monimutkaisempiin aiheisiin.

Koen, että näiden havaintojen pohjalta työ on onnistunut täyttämään sille asetetut laadulliset kriteerit. Se on selkeä, käytännönläheinen, ajankohtainen, monipuolinen ja pedagogisesti hyvin rakennettu. Näiden kriteerien pohjalta opas vastaa hyvin siihen tarpeeseen, johon se on suunniteltu.

4.3 Jatkojalostamismahdollisuudet

Oppaan kirjoittamisprosessin aikana mieleen nousi jatkojalostusmahdollisuuksia, joista kerrotaan lisää tässä aliluvussa.

Opasta voisi jatkojalostaa lisäämällä siihen harjoitustehtäviä, joiden avulla lukijat voisivat konkreettisesti soveltaa oppimiaan asioita. Tämä lisäisi oppaan käytännön arvoa ja tarjoaisi mahdollisuuden syventää osaamista. Koen, että tämä laajennus olisi kokonaisen opinnäytetyön kokoinen prosessi, johon joku voisi ryhtyä kiinnostuessaan aiheesta.

Toinen jatkojalostusmahdollisuus olisi laajentaa opasta kattamaan vaikeampia ohjelmoinnin käsitteitä, kuten olio-ohjelmointia, versionhallintaa ja testausta. Näin opas voisi toimia ns. jatkokurssin materiaalina tai laajempana resurssina ohjelmoinnin opintojen syventämiseen. Myös tämä olisi mielestäni kokonaisen opinnäytetyön laajuinen projekti.

4.4 Ongelmat, epäonnistumiset ja ratkaisut haasteisiin

Yksi isoimmista ongelmista prosessin aikana oli työn laajuuden hallitseminen. Huomasin, että minulla oli paljon enemmän tietoa ja ideoita, kuin mitä pystyin realistisesti sisällyttämään oppaaseen. Tämä johti siihen, että jouduin tekemään merkittäviä rajauksia, mikä tuntui aluksi ikävältä. Olisin halunnut käsitellä monia asioita, kuten objektiorientoitunutta ohjelmointia ja testausta, mutta ymmärsin lopulta, että oppaan tarkoitus on toimia johdantona, ei kattavana käsikirjana.

Toinen haaste oli löytää tasapaino yksityiskohtaisuuden ja ymmärrettävyyden välillä. Kun esittelee teknisiä asioita aloittelijoille, on helppo yksinkertaistaa asioita liikaa tai toisaalta käsitellä niitä liian syvällisesti, jotka voivat molemmat hämmentää lukijaa. Tämä vaati sekä itsekritiikkiä että kohderyhmältä saadun palautteen hyödyntämistä.

Ratkaisuksi rajaamisen ongelmaan päätin keskittyä oppaan selkeyteen ja käytännönläheisyyteen pitäen mielessä kohderyhmän tarpeet. Tämä vaati aiheiden karsimista ja priorisointia. Päätin jättää syvällisemmän käsittelyn esimerkiksi monimutkaisemmista ohjelmointikäsitteistä oppaan ulkopuolelle, mutta toin esille, että nämä aiheet ovat tärkeitä jatko-opiskelun kannalta. Ohjasin myös lukijaa kattavampien materiaalien pariin ja toin esille, mihin kannattaa oppaan lukemisen jälkeen jatkaa, mikäli aihe kiinnostaa enemmän.

Tasapainon löytämiseksi yksityiskohtaisuuden ja ymmärrettävyyden välillä käytin esimerkkejä, joiden avulla pystyin havainnollistamaan abstrakteja käsitteitä konkreettisella tavalla. Testautin myös materiaalia kohderyhmään kuuluvilla henkilöillä, ja heidän palautteensa perusteella tein tarvittavia muutoksia selkeyden varmistamiseksi.

Jos voisin tehdä jotain toisin, alkaisin rajata ja jäsentää sisältöä aiemmin prosessissa. Tämä olisi säästänyt aikaa ja resursseja sekä mahdollistanut syvällisemmän keskittymisen niihin aiheisiin, jotka päätin lopulta sisällyttää oppaaseen.

4.5 Oma oppiminen ja ammatillinen kehittyminen

Opinnäytetyön aikana oma oppimiseni ja ammatillinen kehittymiseni olivat merkittäviä. Alusta alkaen ymmärsin, että tiedon hallitseminen ja sen tiivistäminen selkeäksi ja käyttökelpoiseksi oppaaksi on haastavaa. Tämä prosessi vahvisti viestintätaitojani, jotka ovat keskeisiä alalla kuin alalla. Opin, kuinka esittää monimutkaiset asiat selkeästi ja ymmärrettävästi.

Prosessi opetti myös paljon projektinhallinnasta, erityisesti ajan ja resurssien käyttöä. Opin, kuinka tärkeää on pystyä priorisoimaan ja tekemään päätöksiä, jotka tukevat projektin yleisiä tavoitteita,

vaikka se tarkoittaisi joidenkin mielenkiintoisten aiheiden jättämistä pois. Lisäksi kehitin analyyttistä ajattelukykyäni, erityisesti siinä, miten osaan arvioida materiaalin relevanssia ja soveltuvuutta kohderyhmälle.

Lähteet

Baeldung 2024. Guide to the Java ArrayList. Luettavissa: https://www.baeldung.com/java-arraylist. Luettu: 13.8.2024.

Bautomo s.a. Käyttöliittymä. Luettavissa: https://bautomo.com/sanastoa/kayttoliittyma/. Luettu: 2.6.2024.

Berndtson, T. 17.4.2024. Mitä psykologia tietää oppimisesta. Psykologilehti. Luettavissa: https://psykologilehti.fi/mita-psykologia-tietaa-oppimisesta/. Luettu: 1.6.2024.

Clear, J. s.a. Procrastination: A Scientific Guide on How to Stop Procrastinating. Luettavissa: https://jamesclear.com/procrastination#Make%20the%20Consequences%20of%20Procrastination%20More%20Immediate. Luettu: 2.6.2024.

FreeCodeCamp 2022. Command Line for Beginners – How to Use the Terminal Like a Pro [Full Handbook]. Luettavissa: https://www.freecodecamp.org/news/command-line-for-beginners/. Luettu: 2.6.2024.

GeeksForGeeks 2023. What is computer. Luettavissa: https://www.geeksforgeeks.org/a-simple-understanding-of-computer/. Luettu: 1.6.2024.

Itewiki 2024. Ohjelmistokehittäjistä suuri pula – osaajavajeen ratkaiseminen vaatii mielikuvamuutoksen, uskoo Rakettitieteen Juha Huttunen. Luettavissa: https://www.itewiki.fi/blog/2022/11/ohjelmistokehittajista-suuri-pula-osaajavajeen-ratkaiseminen-vaatii-mielikuvamuutoksen-uskoo-rakettitieteen-juha-huttunen/. Luettu: 2.5.2024.

Liang, D 2015. Introduction to Java programming, Comprehensive version. 10. painos. Pearson. Lontoo.

Licence to Fail 2.5.2020. Opi keskittymään: Pomodoro-tekniikka. Video. Katsottavissa: https://www.youtube.com/watch?v=9pp7AY2DgmA. Katsottu: 4.8.2024.

Machinet 2024. Ymmärtäminen ja käyttö: muuttujat Java-ohjelmoinnissa. Luettavissa: https://www.machinet.net/tutorial-fi/muuttujat-java-ohjelmoinnissa. Luettu: 13.8.2024.

Martins, J. 14.1.2024. Understanding the iterative process, with examples. Asana. Luettaavissa: https://asana.com/resources/iterative-process. Luettu: 6.9.2024.

Medium 2020. Why Java is the best Programming language for Beginners? Luettavissa: https://medium.com/javarevisited/why-java-is-the-best-programming-language-to-learn-coding-for-beginners-cba79aed1271. Luettu: 26.8.2024.

Medium 2023. HashMap vs HashSet in Java. Luettavissa. https://medium.com/javarevisited/hashmap-vs-hashset-in-java-3df49c867ee8. Luettu: 13.8.2024.

Noble Desktop s.a. What is code editors? Luettavissa: https://www.nobledesktop.com/learn/code-editors/what-is-code-editors. Luettu: 16.6.2024.

Ojasalo, K., Moilanen, T., Ritalahti, J. 2015. Kehittämistyön menetelmät – Uudenlaista osaamista liiketoimintaan. 3.-4. painos. Sanoma Pro. Helsinki. E-kirja. Luettu: 6.9.2024.

Opiskelukoulu s.a. Oppimistyylit-myytti ei pidä paikkaansa. Luettavissa: https://opiskelukoulu.fi/oppimistyylit/. Luettu: 1.6.2024.

Roberts, S 4.4.2023. A Brief History of Java Programming Language. The Knowledge Academy. Luettavissa: https://www.theknowledgeacademy.com/blog/history-of-java-programming-language/. Luettu: 6.9.2024.

Scaler 2024. Logical Operators in Java. Luettavissa: https://www.scaler.com/topics/java/logical-operators-in-java/. Luettu: 16.8.2024.

Shiksha Online 2024. Difference Between Errors and Exceptions in Java. Luettavissa: https://www.shiksha.com/online-courses/articles/difference-between-errors-and-exceptions-in-java-blogId-155937. Luettu: 16.8.2024.

Stack Overflow 2024. Most used programming languages among developers worldwide as of 2024. Statista. Luettu: 26.8.2024.

Tutorialspoint s.a. Operating system overview. Luettavissa: https://www.tutorialspoint.com/operating_system/os_overview.htm. Luettu: 1.6.2024.

Tpoint Tech s.a. Java Arrays. Luettavissa: https://www.javatpoint.com/array-in-java. Luettu: 13.8.2024.

UNext 2022. History of Java Programming Language. Luettavissa: https://u-next.com/blogs/java/history-of-java/. Luettu: 28.8.2024.

Visual Studio Code s.a. Learn to code with Visual Studio Code. Luettavissa: https://code.visualstudio.com/learn. Luettu 16.6.2024.

W3Schools s.a. Java Operators. Luettavissa: https://www.w3schools.com/java/java_operators.asp. Luettu: 16.8.2024.

Liitteet

Liite 1. Java-opas aloitteleville ohjelmoijille

2024

Java-opas aloitteleville ohjelmoijille

ANNI SEPPÄNEN

Sisällysluettelo

Johdanto

Nykypäivänä koodia on kaikkialla. Olipa kyse tietokoneista, älypuhelimista tai autoista, ohjelmakoodiin törmää jokapäiväisessä elämässämme. Ohjelmoinnin opiskelu ei ainoastaan auta ymmärtämään näiden arkipäiväisten laitteiden toimintaa vaan avaa myös ovia ympäröivän maailman digitalisaation ymmärtämiseen.

Ohjelmoijilla on suuri kysyntä työmarkkinoilla, ja ohjelmoinnin opiskelu avaa monia erilaisia mahdollisuuksia eri aloilla. Ohjelmistokehitys, tietotekniikka ja tekoäly ovat vain muutamia esimerkkejä urapoluista, joille voi päätyä ohjelmointia opiskelemalla. Ohjelmointi on luovaa ongelmanratkaisua, joka kehittää niin loogista päättelykykyä kuin innovointitaitoja.

Mikä ohjelmoinnissa sitten oikein viehättää, ja miksi se saa ihmiset jopa koukuttumaan? Se hetki, kun useiden eri kokeiluiden, epäonnistumisten ja yritysten jälkeen koodi toimii juuri niin kuin se pitääkin, ja testit näyttävät vihreää, on ainutlaatuisen tyydyttävä. Tämä hetki ei ole pelkästään palkitsevaa, vaan se kehittää myös kärsivällisyyttä ja sinnikkyyttä – arvokkaita ominaisuuksia kaikilla elämän osa-alueilla.

Tämä opas on syntynyt tarpeesta tarjota selkeä ja kattava ohje ohjelmoinnista kiinnostuneille, jotka eivät välttämättä tiedä, mistä aloittaa. Usein nykyiset ohjelmointioppaat ja -materiaalit on suunnattu jo jonkin verran osaaville. Tässä oppaassa ei oleteta, että lukijalla on aiempaa kokemusta ohjelmoinnista. Perusasiat käydään läpi alusta alkaen.

Materiaali on suunniteltu toisella asteella opiskeleville tai sieltä valmistuneille, joilla on ainakin hieman kiinnostusta ohjelmointia kohtaan. Matematiikan alkeet oletetaan olevan hallussa, ja niitä ei käsitellä tässä oppaassa. Ohjeet on kirjoitettu Windows-käyttöjärjestelmälle.

Tavoitteena on näyttää, että ohjelmoinnin opiskelu ei ole pelkästään teoreettista puurtamista ja se avaa ovia luovaan ongelmanratkaisuun. Ohjelmointitaitoja voi käyttää monilla elämän eri osa-alueilla, olipa kyse sitten työelämästä, harrastuksista tai omista projekteista. Oppaan esimerkkien tarkoituksena on osoittaa yhteys arkielämään konkreettisesti.

Yksi arkielämän tapaus, joka on mahdollista ohjelmoinnin avulla ratkaista, on oman talouden budjetointi. Omien tulojen ja menojen suunnittelu on monen mielestä hankalaa ja aikaa vievää ja se jää usein kokonaan tekemättä. Ohjelmoinnin avulla voidaan helposti rakentaa sovellus, joka helpottaa kulujen seuraamista, tavoitteiden asettamista ja joka auttaa tekemään parempia taloudellisia päätöksiä. Tässä oppaassa käydään läpi, minkälaisista osista ratkaisu lähtee liikkeelle ja miten sitä voisi ohjelmoinnin keinoin lähestyä. Tehtävien avulla harjoitellaan pienissä osissa ohjelmoinnin

eri osa-alueita, ja lopuksi kirjoitetaan kokonainen budjetointisovellus. Ohjelmointitaitojen karttumisen lisäksi huomataan, että arkipäiväisiä ongelmia voidaan ratkaista ohjelmoinnin avulla.

Tartu haasteeseen ja lähde matkalle Java-ohjelmoinnin maailmaan. Tämä opas on tehty juuri sinua varten – toivottavasti löydät siitä inspiraatiota ja iloa ohjelmoinnin opiskeluun!

Ohjelmoinnin opiskelusta

Ohjelmoinnin oppiminen ei ole helppoa, ja se vaatii systemaattista lähestymistapaa ja monipuolisia opiskelumenetelmiä. Tässä luvussa käydään läpi, miksi opiskelutekniikat ovat tärkeitä ja miten niitä voi soveltaa ohjelmoinnin opiskeluun.

Erilaiset opiskelutekniikat ovat tärkeitä monista syistä. Ne auttavat esimerkiksi tekemään oppimisprosessista tehokkaamman. Hyvien opiskelutekniikoiden avulla on mahdollista oppia ja muistaa tietoa nopeammin ja perusteellisemmin. Tämä tarkoittaa, että uusien taitojen ja tietojen omaksuminen on mahdollista vähemmällä vaivalla ja ajalla, mikä on erityisen hyödyllistä kiireisessä arjessa.

Opiskelutekniikat auttavat parantamaan keskittymiskykyä ja motivaatiota. Opiskeluaika kannattaa jaotella hallittaviin jaksoihin, mikä tekee pitkäkestoisista oppimissessioista vähemmän uuvuttavia. Pienemmät, hallittavat tehtävät ja säännölliset tauot auttavat ylläpitämään keskittymistä ja ne vähentävät stressiä. Tämä voi myös vähentää prokrastinaatiota, koska tehtävien jakaminen pienempiin osiin tekee niistä vähemmän pelottavia ja helpommin aloitettavia.

Tehokkaat opiskelutekniikat auttavat myös säilyttämään ja muistamaan opitun tiedon paremmin. Muistiinpanojen tekeminen ja säännöllinen kertaus auttavat vahvistamaan muistijälkiä. Tämä on erityisen tärkeää pitkäaikaisen oppimisen kannalta, sillä se varmistaa, että tieto ei katoa heti lukemisen jälkeen vaan säilyy käytettävissä myös tulevaisuudessa.

Opiskelutekniikat voivat lisäksi parantaa oppimisen laatua. Kun käyttää erilaisia menetelmiä, kuten opettaminen muille, ajatusten ääneen selittäminen tai monipuolisten resurssien hyödyntäminen, syvennät ymmärrystäsi ja pystyt yhdistämään uutta tietoa aiemmin oppimaasi. Tämä tekee oppimisesta kokonaisvaltaisempaa ja auttaa soveltamaan tietoa monipuolisemmin käytännön tilanteissa.

Opiskelutekniikoiden käyttäminen myös tukee elinikäistä oppimista. Nykymaailmassa tiedon ja taitojen jatkuva päivittäminen on välttämätöntä. Hyvien opiskelutaitojen avulla voit itseohjautuvasti oppia uusia asioita ja soveltaa niitä nopeasti muuttuvissa tilanteissa. Tämä tekee sinusta joustavamman ja paremmin varustautuneen vastaamaan tulevaisuuden haasteisiin.

Opiskelutekniikoita

Kun opit uutta asiaa, kirjoita muistiinpanoja. Ota ylös tärkeimmät käsitteet ja omat havaintosi. Hyvät muistiinpanot auttavat sinua kertaamaan ja muistamaan opitut asiat myöhemmin.

Opeta oppimiasi asioita muille. Opettamalla muille tai selittämällä asioita ääneen itsellesi, voit syventää ymmärrystäsi. Tämä metodi auttaa sinua havaitsemaan aukkoja omassa tiedossasi ja selkeyttämään käsitteitä.

Toisto on avain oppimiseen. Säännöllinen kertaaminen vahvistaa muistijälkiä ja tekee tiedosta pysyvämpää. Käytä erilaisia toistotekniikoita, kuten kertauskysymyksiä, flash-kortteja ja kertauskäyntejä muistiinpanoihisi. Toisto kannattaa ajoittaa säännöllisesti, esimerkiksi päivittäin tai viikoittain, jotta opittu tieto pysyy tuoreena mielessä.

Ajan hallinta ja keskittyminen ovat keskeisiä tekijöitä tehokkaassa oppimisessa. Pomodoro-tekniikka on erinomainen tapa parantaa keskittymiskykyäsi. Työskentele 25 minuutin jaksoissa, joiden jälkeen pidä 5 minuutin tauko. Neljän jakson jälkeen pidä pidempi, 15 – 30 minuutin tauko. Tämä menetelmä auttaa sinua pitämään keskittymisen korkealla ja estää uupumisen.

Älä unohda lepoa ja liikuntaa. Päiväunet ja fyysinen aktiivisuus voivat merkittävästi parantaa oppimiskykyä ja keskittymistä. Lyhyet päiväunet voivat virkistää mieltäsi ja auttaa jäsentelemään oppimaasi tietoa. Säännöllinen liikunta puolestaan edistää aivojen terveyttä ja parantaa kognitiivisia toimintoja.

Googlaaminen on tärkeä taito nykyaikaisessa oppimisessa. Se saattaa tuntua itsestäänselvältä ja helpolta, mutta sitäkin taitoa tulee harjoitella. Kun kohtaat ongelmia, käytä Googlea löytääksesi vastauksia. Usein muut ohjelmoijat ovat kohdanneet samoja ongelmia, ja heidän ratkaisunsa löytyvät Googlesta.

Kannattaa myös opetella hyödyntämään monipuolisia oppimisresursseja netissä. Ilmaiset verkkokurssit ja oppimateriaalit, kuten FreeCodeCamp ja GeeksForGeeks, tarjoavat arvokasta tietoa ja käytännön harjoituksia. Tee harjoitustehtäviä säännöllisesti, sillä käytännön harjoittelu on avain oppimiseen. Muista, että oppiminen vaatii aikaa, kärsivällisyyttä ja toistoa. Pidä motivaatiosi korkealla asettamalla itsellesi pieniä, saavutettavissa olevia tavoitteita ja palkitse itsesi niiden saavuttamisesta. Näin voit nauttia oppimisprosessista ja saavuttaa parempia tuloksia Java-ohjelmoinnissa.

Esimerkiksi näitä opiskelutekniikoita soveltamalla ohjelmoinnin opiskelusta saa innostavaa, motivoivaa ja pitkäjänteistä. Onnea opiskeluun!

Tietokoneen perusasiat aloittelijoille

Ohjelmoinnin opiskeluun kuuluu muutakin kuin pelkän koodin kirjoittaminen. On tärkeää ymmärtää hieman siitä, miten tietokoneet toimivat ja miten ne käyttävät resurssejaan. Näiden perusasioiden ymmärtäminen auttaa siirtymään sujuvasti Java-ohjelmoinnin pariin.

Tietokoneen tärkeimmät osat

Tietokoneen prosessori suorittaa laskutoimitukset ja tehtävät, joita ohjelma vaatii. Väliaikainen muisti (RAM) säilyttää tiedot ohjelman ajon aikana, kun taas kovalevy tallentaa tiedot pysyvästi. Käyttöjärjestelmä, kuten Windows, MacOS tai Linux, mahdollistaa ohjelmien ja laitteiston välisen kommunikaation, hoitaen esimerkiksi tiedostojen ja muistin hallinnan.

Käyttöliittymät ja komentorivi

Käyttäjät voivat kommunikoida tietokoneen kanssa käyttöliittymien avulla. Graafisessa käyttöliittymässä (GUI) toiminnot suoritetaan hiirellä klikkailemalla kuvakkeita. Komentorivi (CLI) taas on tekstipohjainen työkalu, jota monet ohjelmoijat käyttävät tehokkuutensa vuoksi. Komentoriviltä voidaan hallita versionhallintaa ja suorittaa muita tehtäviä, jotka ovat tärkeitä ohjelmoinnissa.

Koodieditori

Vaikka koodia voi kirjoittaa millä tahansa tekstinkäsittelyohjelmalla, koodieditorit, kuten Visual Studio Code, tarjoavat hyödyllisiä ominaisuuksia, kuten syntaksivirheiden tarkistamisen ja koodin automaattisen sisentämisen. Nämä työkalut tekevät koodin kirjoittamisesta sujuvampaa ja helpottavat oppimista.

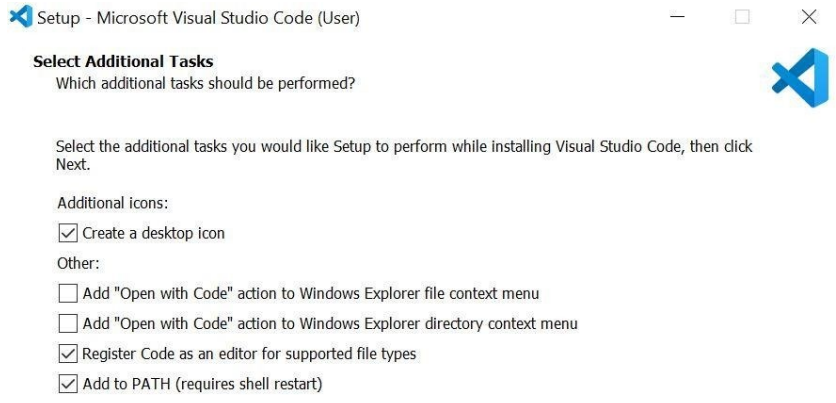
Java-ohjelmoinnin aloittaminen

Ohjelmoinnin opiskelun aloittamisessa on muutamia vaiheita, jotka täytyy tehdä, ennen kuin koodia pääsee kirjoittamaan ja ajamaan. Koodia on mahdollista kirjoittaa periaatteessa millä tahansa tekstinkäsittelyohjelmalla, kuten Notepadilla, mutta on myös kehitetty sovelluksia, jotka helpottavat ja nopeuttavat ohjelmointia huomattavasti. Näitä sovelluksia kutsutaan koodieditoreiksi. Yksi helppokäyttöinen sekä ilmainen koodieditori on Visual Studio Code (VSC). Seuraavaksi asennetaan VSC.

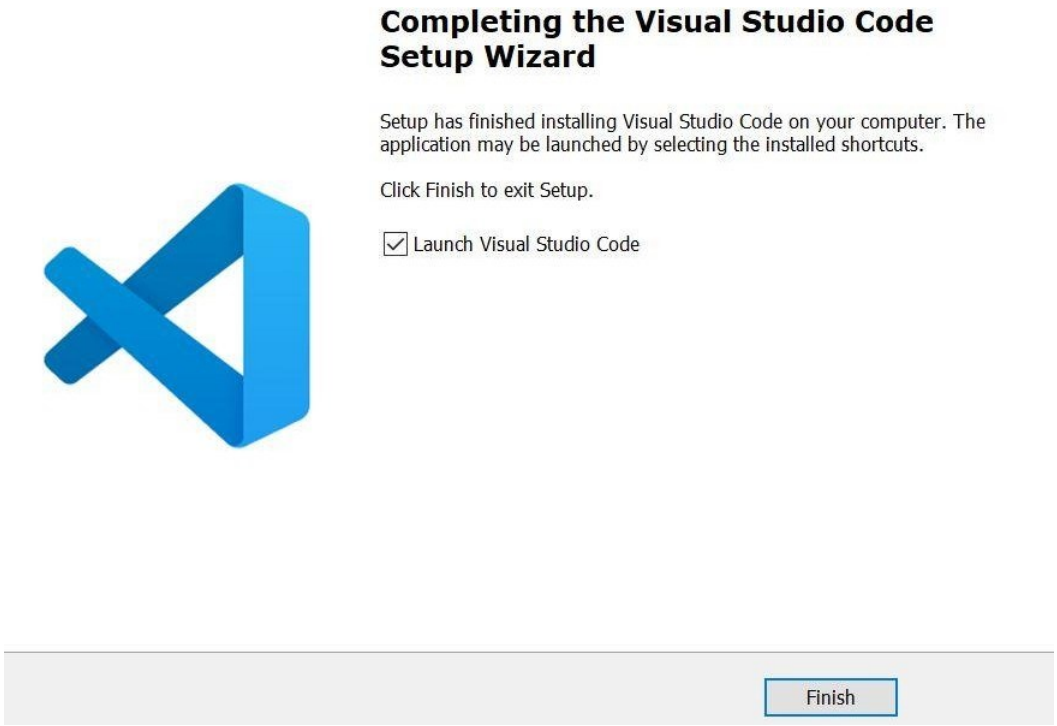
Visual Studio Coden asennus

Avaa Visual Studio Coden lataussivu (https://code.visualstudio.com/#alt-downloads). Vieritä ruutua alas ja lataa Windows-käyttöjärjestelmään sopiva asennuspaketti.

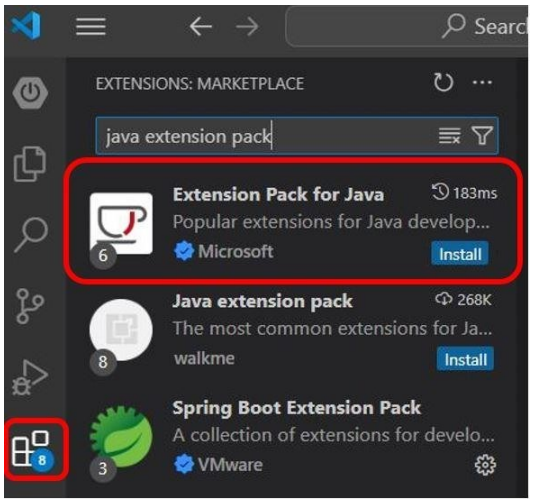
Avaa ladattu asennuspaketti tuplaklikkaamalla (todennäköisesti se sijaitsee tietokoneesi Ladatut tiedostot -kansiossa (Downloads)) ja seuraa asennusohjeita. Valitse alla olevassa kuvassa olevat vaihtoedot, kun tämä ikkuna tulee näkyville.



Kun asennus on valmis, eli näytöllä näkyy alla olevan kuvan mukainen ikkuna, avaa VSC ruksaamalla Launch Visual Studio Code -kohta ja painamalla Finish. Voit myös avata VSC:n työpöytäkuvakkeesta painamalla.



Asenna seuraavaksi laajennukset, joita tarvitaan Java-ohjelmointia varten. Siirry VSC:ssä Extensions-osioon vasemmassa sivupalkissa, joka on kehystetty alla olevan kuvan vasemmassa alanurkassa.
Etsi Java Extension Pack, valitse alla olevassa kuvassa kehystetty vaihtoehto (Extension Pack for Java) ja paina Install.

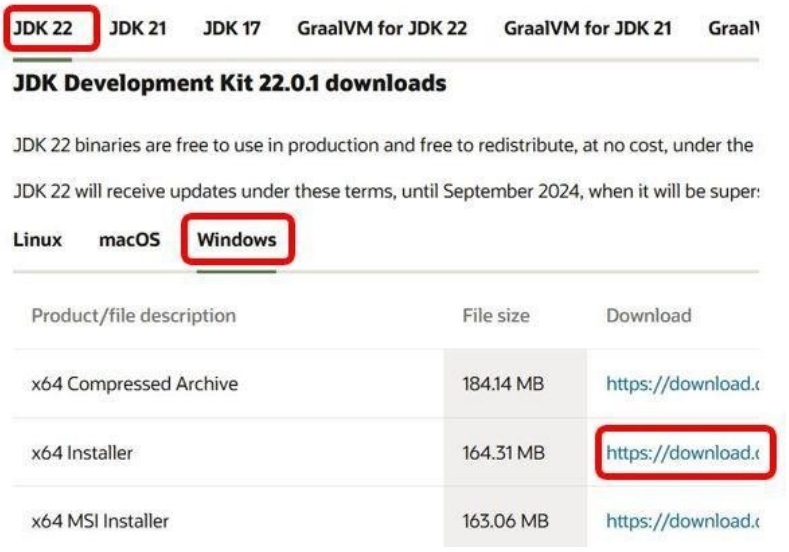


Ikkuna näyttää tältä, kun asennus on valmis.

Javan asennus

Koodieditorin asentamisen lisäksi tulee asentaa Java Development Kit (JDK), jotta tietokone osaa suorittaa kirjoitetun ohjelman. Javan asentamisessa on vaiheita, jotka saattavat vaikuttaa hankalilta. Nämä asiat käydään läpi oppaassa myöhemmin.

Avaa Oracle JDK:n lataussivu (https://www.oracle.com/java/technologies/javase-jdk11-downloads.html).
Valitse uusin versio ja lataa asennuspaketti. Opasta kirjoittaessa uusin versio on 22.

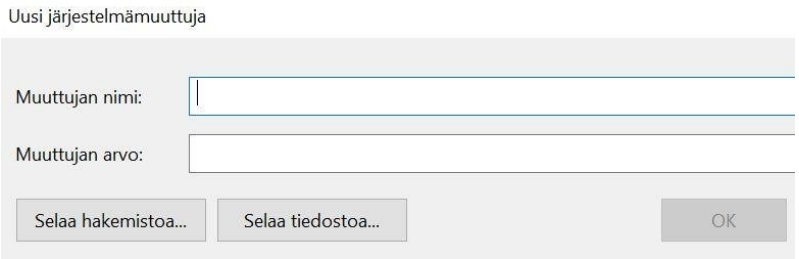


Avaa ladattu asennuspaketti ja seuraa asennusohjeita. Ota ylös polku, johon JDK asennetaan. Alla olevassa kuvassa polku on C:\Program Files\Java\jdk-22\. Asennuspolkua tarvitaan asennuksen seuraavassa vaiheessa.

Lisätään seuraavaksi ympäristömuuttujat. Avaa Ohjauspaneeli / Control Panel, klikkaa Järjestelmä ja suojaus / System and Security ja klikkaa vielä Järjestelmä / System.

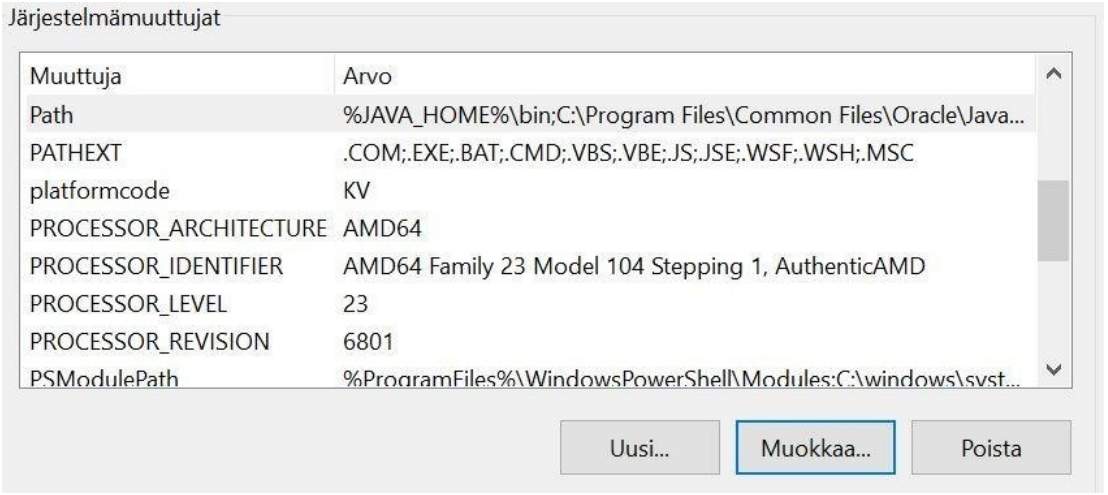
Vieritä ruutua alas, paina järjestelmäasetukset ja sen jälkeen paina ympäristömuuttujat.

Klikkaa Uusi / New Järjestelmämuuttujat / System variables -osiossa.
Luo uusi muuttuja.
Muuttujan nimi / Variable name: JAVA_HOME
Muuttujan arvo / Variable value: JDK:n asennushakemisto, eli aikaisemmin ylös otettu polku (esim. C:\Program Files\Java\jdk-22\)

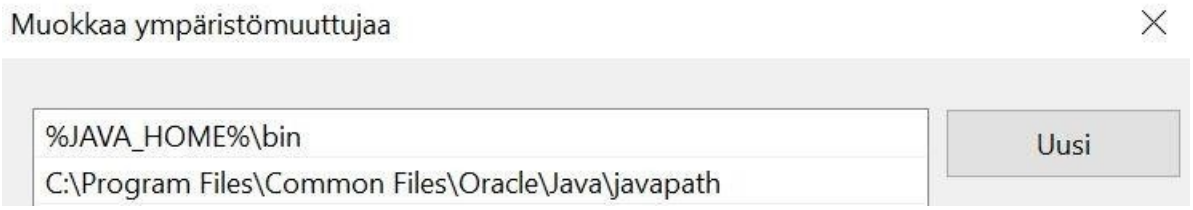


Paina OK.

Etsi "Path" Järjestelmämuuttujat / System variables -osiosta, klikkaa sitä ja klikkaa Muokkaa / Edit.



Klikkaa Uusi / New ja lisää uusi polku: %JAVA_HOME%\bin.



Avaa Komentokehote / Command prompt, kirjoita java –version ja paina enter. Mikäli näytölle ilmestyy jotain samankaltaista kuin alla olevassa kuvassa, asennus on onnistunut. Jos taas tulostus on esim. “'java' is not recognized as an internal or external command, operable program or batch file.” niin asennuksessa on mennyt jokin pieleen. Suosittelen tässä vaiheessa aloittamaan asennuksen alusta.

```
C:\Users\seppa>java --version
java 17.0.9 2023-10-17 LTS
Java(TM) SE Runtime Environment (build 17.0.9+11-LTS-201)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.9+11-LTS-201, mixed mode, sharing)
```

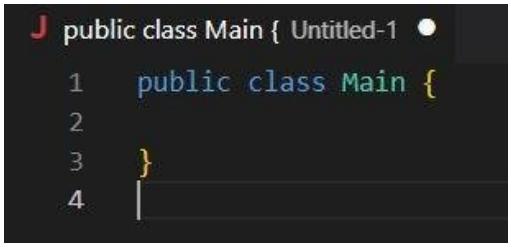
Kun Java on onnistuneesti asennettu, voidaan ensimmäisen ohjelman kirjoittaminen viimein aloittaa!

(Ympäristömuuttujat tulee lisätä esimerkiksi sen takia, että käyttöjärjestelmä tietää, mistä etsiä suoritettavia tiedostoja, kun komentoja ajetaan komentoriviltä. Se myös lisää helppokäyttöisyyttä, kun Java-komentoja voi ajaa mistä tahansa komentoriviltä, eikä tiettyyn hakemistoon tarvitse aina navigoida. Nämä ovat tässä vaiheessa vain lisätietoja ja perusteluja niistä kiinnostuneille.)

Ensimmäinen Java-ohjelma

Ensimmäinen askel Java-ohjelmoinnin oppimisessa on yksinkertaisen “Hello World” -ohjelman kirjoittaminen ja suorittaminen. Tämä on luultavasti yleisin ensimmäinen ohjelma, jonka aloittelevat koodarit kirjoittavat. Tämä ohjelma yksinkertaisesti tulostaa näytölle tekstin “Hello World”.

Aloitetaan avaamalla Visual Studio Code. Viedään hiiri ylävasemmalla sijaitsevan File-tekstin päälle, jonka jälkeen alas avautuvasta listasta klikataan New File. Tämän jälkeen valitaan tiedostotyypiksi New Java file ja ruudulle ilmestyy alla oleva näkymä.



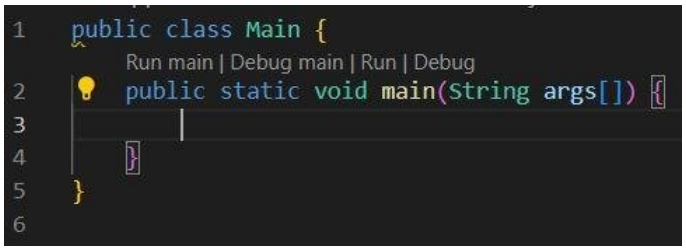
VSC on automaattisesti luonut Java-luokan (englanniksi class), jonka nimi on Main. Palataan luokkiin myöhemmin syvällisemmin.

Ohjelmatiedosto kannattaa tallentaa heti alussa. Tämä tapahtuu joko painamalla File>Save tai lyhyemmin painamalla Ctrl+s. Tiedoston nimi tulee olla sama kuin luokan nimi. Tässä tilanteessa tiedoston nimeksi tulee siis antaa Main.

Kiinnitä erityistä huomiota isoihin ja pieniin kirjaimiin koodia kirjoittaessasi. Javassa main ja Main ovat eri asioita. Myös sisennykset ovat Javassa tärkeitä, sillä ne helpottavat koodin lukua, mutta niiden ”vääränlainen käyttö” ei aiheuta virhetilanteita. On kuitenkin hyvä alusta asti opetella niin sanottuja koodauksen hyviä tapoja. Sisennyksen saa tehtyä tab-näppäimellä.

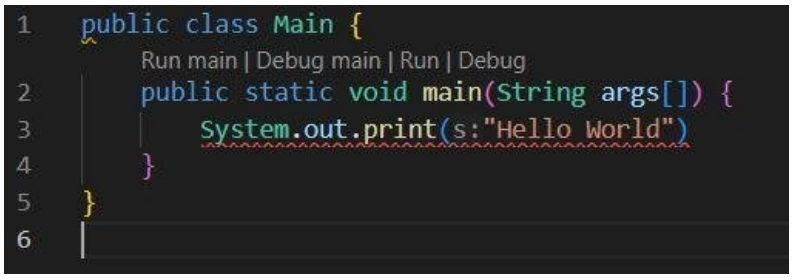
Luokan sisälle, riville numero 2, lisätään päämetodi (englanniksi main-method). Myös metodeihin palataan myöhemmin. Siirrytään riville 2, tehdään sisennys, mikäli kursori ei ole automaattisesti sisennetty, ja kirjoitetaan public static void main(String args[]) {}. VSC lisää sulkeiden oikean

puolen automaattisesti, joten älä hämäänny tästä. Siirrytään rivillä 2 olevien aaltosulkeiden väliin ja painetaan enter: nyt päämetodi on määritelty, ja sen sisään voi kirjoittaa lisää koodia. Kuvassa näkyvästä harmaalla tekstillä olevasta rivistä ”Run main | Debug main | Run | Debug” ei tarvitse välittää. VSC lisää sen automaattisesti päämetodin kirjoittamisen jälkeen.



Rivistä public static void main(String args[]) {} ei vielä tarvitse ymmärtää muuta kuin se, että päämetodi määritellään aina näin.

Seuraavaksi kirjoitetaan metodin sisään, riville 3, itse ohjelma. Siirrytään riville 3, tehdään sisennys (mikäli kursori ei ole automaattisesti sisennetty) ja kirjoitetaan System.out.print(”Hello World”).



Nyt huomataan kuitenkin, että VSC on alleviivannut punaisella koodin System.out.print(”Hello World”). Ajetaan ohjelma yläreunassa olevasta play-napista ja katsotaan, mitä tapahtuu.

Nyt näytön alareunaan avautuvaan terminaaliin tulee seuraava teksti:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Syntax error, insert ";" to complete BlockStatements

        at Main.main(Main.java:3)
```

Kyseessä on ensimmäinen virhe! Ohjelmoinnissa kohdataan usein erilaisia virheitä ja on alusta asti hyvä oppia lukemaan ja ymmärtämään niitä. Tässä ”Syntax error, insert ”;” to complete BlockStatements” viittaa siihen, että koodissa on syntaksi- eli kirjoitusvirhe ja tarkennetaan, että koodiin tulee lisätä puolipiste (;). Java-koodissa voi nyrkkisääntönä pitää, että kaikkien muiden rivien päätteeksi

tulee puolipiste, paitsi luokan tai metodin lopun määrittelevän aaltosulkeen } jälkeen. Lisätään puolipiste oikeaan paikkaan.

```
1   public class Main {
        Run main | Debug main | Run | Debug
2       public static void main(String args[]) {
3           System.out.print(s:"Hello World");
4       }
5   }
6
```

Ohjelmoinnissa tulee olla erittäin tarkkana oikeinkirjoituksen kanssa. VSC on onneksi hyvä huomaamaan erilaisia virheitä, jolloin ne on helpompaa korjata. Mikäli ohjelmaa ajaessa tulee jokin muu virhe, suosittelen tarkistamaan ohjelman kirjoitusvirheiltä ja katsomaan, että jokainen iso ja pieni kirjain on oikein. Ajetaan nyt koodi ja katsotaan tuloste terminaalista.

```
PROBLEMS 7   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

        Syntax error, insert ";" to complete BlockStatements

        at Main.main(Main.java:3)
PS C:\Users\seppa> ^C
PS C:\Users\seppa>
PS C:\Users\seppa>  & 'C:\Program Files\Java\jdk-17\bin\java.exe'
_6e6d5\jdt_ws\jdt.ls-java-project\bin' 'Main'
Hello World
PS C:\Users\seppa>
```

Nyt terminaalissa on rivi Hello World. Ohjelma siis toimii!

Kommenttien käyttö

Kommentit ovat koodirivejä, jotka lisätään koodiin sen selkeyttämiseksi, mutta niitä ei suoriteta. Kommentit auttavat ymmärtämään, mitä koodi tekee, ja niiden avulla on helppo palata vanhan koodin pariin, jonka toimintaa ei välttämättä muista. Niiden merkitys kasvaa, kun koodia kirjoittaa useampi kuin yksi henkilö: tällöin on erittäin tärkeää, että kaikki ymmärtävät, mitä koodi tekee. Mikäli kommentointi on tehty hyvin, se myös tehostaa työskentelyä, sillä joskus pelkästä koodista voi olla vaikeaa ymmärtää nopeasti, mistä on kyse. Kommentit lukemalla koodin hahmottaminen on mahdollista yhdellä vilkaisulla.

Yksiriviset kommentit alkavat kahdella kauttaviivalla (//). Samalla rivillä oleva teksti kauttamerkkien jälkeen jätetään siis huomiotta ohjelmaa suorittaessa.

```java
// Tämä on yksirivinen kommentti
public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        System.out.print(s:"Hello World");

    }
}

```

Moniriviset kommentit alkavat merkeillä /*, joiden jälkeen kirjoitetaan itse kommentti ja lopuksi lisätään */ -merkit. Tällöin kaikki teksti näiden merkkien välillä jätetään huomiotta, vaikka se olisi usealla eri rivillä.

```java
public class Main {
    /* Tämä
    on monirivinen
    kommentti */
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        System.out.print(s:"Hello World");

    }
}

```

Kommentointi saattaa ohjelmoinnin alussa tuntua turhalta, ja usein ehkä ajatellaan, että kyllähän minä muistan, mitä koodini tekee. On kuitenkin hyvä muistaa kommenttien olemassaolo ja niiden hyöty, varsinkin kun ohjelmien koko alkaa kasvaa.

Tietorakenteet

Jotta arkielämän asioita saataisiin muutettua tietokoneen ymmärtämään muotoon, tarvitaan erilaisia tietorakenteita. Miten Javassa voisi mallintaa esimerkiksi tuloja, menoja ja budjettia? Miten tilanne eroaa, kun halutaankin mallintaa tekstiä? Mitä jos näitä tietoja on useita, ja ne halutaan tallentaa yhdeksi listaksi? Näihin kysymyksiin vastataan seuraavissa aliluvuissa.

Muuttujat

Muuttujat ovat tietokoneen muistissa olevia nimettyjä säilytyspaikkoja, joihin voi tallentaa arvoja. Niiden avulla tietoja voidaan käsitellä eri tavoilla. Muuttujilla on aina tietotyyppi, joka määrittää, millaisia arvoja muuttuja voi saada ja toisaalta, millaisia arvoja muuttujalle ei voi antaa. Yleisiä tietotyyppejä Javassa ovat:

integer (kokonaisluku): esim. -1, 0, 20
double (desimaaliluku): esim. -0.001, 2.2, 3.14
String (merkkijono): esim. “Hei”, “123”, “Mitä kuuluu?”
boolean (totuusarvo): true (tosi) tai false (epätosi)

Javassa esimerkiksi muuttujalle, jonka tietotyyppi on kokonaisluku (integer), ei voi antaa arvoksi desimaalilukua (double).

Muuttujien nimeäminen aloitetaan Javassa pienellä kirjaimella. Mikäli muuttujan nimi on pidempi kuin yksi sana, kirjoitetaan ensimmäinen sana edelleen pienellä, mutta loput sanoista aloitetaan isolla kirjaimella, esimerkiksi onkoBudjettiYlittynyt. Yleensä muuttujat nimetään englanninkielisillä sanoilla, mutta tässä vaiheessa ymmärtämisen helpottamiseksi käytetään suomenkielisiä muuttujien nimiä.

Budjettisovelluksessa voisi olla esimerkiksi muuttuja budjetti, jonka arvo on 1000, muuttuja onkoBudjettiYlittynyt, jonka arvo on false ja muuttuja viesti, jonka arvo on ”Budjetti on ylittynyt”. Alla olevassa kuvassa on havainnollistettu, miten muuttujat määritellään Javassa.

```
1  public class Main {
       Run main | Debug main | Run | Debug
2      public static void main(String args[]) {
3          int budjetti = 1000;
4          boolean onkoBudjettiYlittynyt = false;
5          String viesti = "Budjetti on ylittynyt";
6      }
7  }
8
```

Huomataan, että muuttuja määritellään kirjoittamalla ensin muuttujan tietotyyppi, muuttujan nimi, yhtä suuri kuin -merkki (=), muuttujan arvo ja puolipiste. Kokonaisluvulle (integer) tietotyyppi lyhennetään int, totuusarvossa tietotyyppi on boolean ja merkkijonossa String (huomaa iso alkukirjain).

Muuttujia voidaan tulostaa samalla tavoin, kuin teimme aikaisemmin merkkijonolle ”Hello World”. Esimerkiksi muuttuja budjetti voidaan tulostaa kirjoittamalla System.out.print(budjetti);.

```
1   public class Main {
        Run main | Debug main | Run | Debug
2       public static void main(String args[]) {
3           int budjetti = 1000;
4           boolean onkoBudjettiYlittynyt = false;
5           String viesti = "Budjetti on ylittynyt";
6
7           System.out.print(budjetti);
8       }
9   }
10
```

Ajetaan ohjelma play-nappia painamalla ja katsotaan tulostus terminaalista.

```
PS C:\Users\seppa>  &
\Temp\vscodesws_d37e6\
1000
```

Käyttäjän syötteen lukeminen

Java tarjoaa valmiiksi useita luokkia, joita voidaan käyttää ohjelmissa erilaisiin tarkoituksiin. Yksi tärkeimmistä luokista aloittelijalle on Scanner. Sillä voidaan lukea käyttäjän syötettä esimerkiksi näppäimistöltä. Käyttäjän syötteen käsittely on keskeinen osa monia ohjelmia, koska se tekee ohjelmista interaktiivisia.

Scanner-luokan saa käyttöön lauseella import java.util.Scanner, jonka jälkeen tulee luoda Scanner-olio, joka lukee syötettä. Import-lause on alla olevassa kuvassa rivillä 2, ja olion luominen on rivillä 6. Seuraavaksi luetaan käyttäjän syöte. Tähän voidaan käyttää erilaisia metodeja, riippuen käyttäjän syötteen tietotyypistä. Esimerkiksi nextLine() lukee tekstiä, nextInt() lukee kokonaislukuja ja nextDouble() lukee liukulukuja. Alla olevassa esimerkissä siis halutaan käyttäjän kertovan budjetin kokonaislukumuodossa.

```java
import java.util.Scanner;

public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        Scanner lukija = new Scanner(System.in);

        System.out.println(x:"Kerro budjetti: ");
        int budjetti = lukija.nextInt();

        System.out.println("Budjettisi on " + budjetti);
    }
}
```

Kun koodi ajetaan, terminaaliin ilmestyy teksti ”Kerro budjetti: ”, jonka jälkeen budjetin voi kirjoittaa suoraan terminaaliin. Sen jälkeen tulostetaan ”Budjettisi on ” + budjetti.

```
Kerro budjetti:
100
Budjettisi on 100
```

Taulukot ja listat

Taulukko (array) on kiinteän kokoinen tietorakenne, eli sen koko ei voi määrittelyn jälkeen muuttua. Taulukoilla on tietty tietotyyppi, samalla tavoin kuin muuttujillakin, eli tietyn tietotyypin taulukko voi saada vain tämän tietotyypin arvoja. Taulukot ovat hyödyllisiä, jos tiedetään, kuinka monta arvoa taulukkoon halutaan tallentaa. Määritellään seuraavaksi taulukko, jossa on viiden eri päivän kulut.

```java
public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        int[] kulut = {20, 50, 10, 100, 10};
    }
}
```

Huomataan, että taulukko määritellään kirjoittamalla ensin tietotyyppi (int), jonka jälkeen tulee hakasulut [], taulukon nimi, tässä kulut, yhtä suuri kuin -merkki ja aaltosulut {}, joiden sisään annetaan arvot.

Taulukon voi myös määritellä alussa tyhjäksi, mutta sillä tulee olla koko. Alla olevassa kuvassa määritellään taulukko, jonka koko on viisi.

```
1  public class Main {
       Run main | Debug main | Run | Debug
2      public static void main(String args[]) {
3          int[] kulut = new int[5];
4      }
5  }
6
```

Jos halutaan määritellä tyhjä taulukko, se määritellään aluksi samalla tavalla kuin taulukko, jolla on jo arvoja, eli int[] kulut =, mutta sen jälkeen tulee lisätä new int, ja sitten hakasulkeisiin [] taulukon koko. Notaatiolla new int[5] itseasiassa määritellään siis uusi tyhjä taulukko-objekti, mutta palataan objekteihin myöhemmin.

Lista on samankaltainen tietorakenne kuin taulukkokin, mutta erona on se, että listalla ei ole määrättyä kokoa. Listan koko voi siis muuttua, eikä määrittelyvaiheessa tarvitse siis tietää, minkäkokoinen lista tulee olemaan. Jotta listatietorakennetta voi käyttää, tulee se ensin hakea java.util-paketista lauseella import java.util.ArrayList;. Tämä lause kirjoitetaan heti ylimmäksi, ennen luokan määrittelyä (kuvassa rivi 2). Lista määritellään aina ensin tyhjäksi, eli luodaan tyhjä listaobjekti kirjoittamalla ArrayList, <>-merkit, joiden sisään kirjoitetaan tietotyyppi (tässä tilanteessa Integer), listan nimi, yhtä suuri kuin -merkki, new ArrayList<>();. Tietotyyppi on tässä kohtaa Integer (eikä int), koska lista ottaa arvoikseen vain objekteja, mutta tästä ei tarvitse vielä ymmärtää sen enempää.

```
1
2  import java.util.ArrayList;
3
4  public class Main {
       Run main | Debug main | Run | Debug
5      public static void main(String args[]) {
6          ArrayList<Integer> kulut = new ArrayList<>();
7      }
8  }
9
```

Jos listaan haluttaisiin kokonaislukujen sijaan tallentaa esimerkiksi merkkijonoja, olisi määrittely tämännäköinen ArrayList<String> nimi = new ArrayList<>().

Taulukoiden ja listojen operaatioita

Hakeminen

Kun taulukosta tai listasta halutaan esimerkiksi tulostaa jokin tietty arvo, tarvitaan tämän arvon indeksi eli paikka taulukossa tai listassa. Taulukoiden ja listojen indeksointi alkaa nollasta, eli taulukon kulut = {20, 50, 10, 100, 10} kohdassa 0 on arvo 20, kohdassa 1 on arvo 50 ja niin edelleen.

Taulukon viimeisen arvon indeksi on joko taulukon pituus – 1 tai vain -1. Taulukon kulut ensimmäinen arvo saadaan komennolla kulut[0], toinen arvo saadaan komennolla kulut[1] ja viimeinen arvo saadaan komennolla kulut[-1] tai kulut[4]. Määritellään nyt muuttuja ensimmäinenArvo, jonka arvoksi haetaan taulukosta kulut ensimmäinen arvo ja tulostetaan se.

```
1
2   public class Main {
        Run main | Debug main | Run | Debug
3       public static void main(String args[]) {
4           int[] kulut = {20, 50, 10, 100, 10};
5           int ensimmäinenArvo = kulut[0];
6           System.out.print(ensimmäinenArvo);
7       }
8   }
9
```

Terminaalissa tulostus näyttää tältä.

```
\Temp\vscodesws_
20
```

Lisääminen

Taulukoihin ja listoihin voidaan lisätä arvoja, mutta se tapahtuu eri tavoilla taulukoissa ja listoissa. Käydään ensin läpi taulukkoon lisääminen.

Koska taulukolla on tietty koko, täytyy lisättävien arvojen paikka tietää. Jos esimerkiksi haluamme lisätä taulukon ensimmäiseen kohtaan arvon, tapahtuu se alla olevalla tavalla.

```
1
2   public class Main {
        Run main | Debug main | Run | Debug
3       public static void main(String args[]) {
4           int[] kulut = new int[5];
5           kulut[0] = 20;
6       }
7   }
8
```

Tulostetaan vielä koko taulukko. Javassa taulukoita ei voi suoraan tulostaa komennolla System.out.print(), vaan joudutaan hakemaan paketista java.util luokka Arrays, jossa on metodi toString(), joka muuttaa taulukon merkkijonoksi. Komennolla import java.util.Arrays; (kuvassa rivi 2), haetaan luokka Arrays, ja rivillä 9 System.out.print(Arrays.toString(kulut)); käytetään tästä luokasta haettua toString()-metodia.

```java
import java.util.Arrays;


public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        int[] kulut = new int[5];
        kulut[0] = 20;
        System.out.println(Arrays.toString(kulut));
    }
}
```

Nyt terminaalissa on tulostus.

```
\Temp\vscodesws_a268d
[20, 0, 0, 0, 0]
```

Huomataan, että ensimmäinen arvo on 20, ja loput ovat 0.

Listaan voidaan lisätä arvoja metodilla add.

```java
import java.util.ArrayList;

public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        ArrayList<Integer> kulut = new ArrayList<>();
        kulut.add(e:20);
        System.out.println(kulut);
    }
}
```

Terminaalissa tulostus näyttää nyt tältä.

```
\Temp\vscodesws_
[20]
```

Poistaminen

Koska taulukot ovat kiinteän kokoisia, ei niistä voi suoranaisesti poistaa arvoja. On kuitenkin mahdollista muuttaa arvo takaisin nollaksi, lyhyesti kulut[halutun arvon indeksi] = 0.

Listasta voidaan poistaa arvoja metodilla remove(), joka ottaa parametrikseen indeksin, josta tieto halutaan poistaa.

```
1
2   import java.util.ArrayList;
3
4   public class Main {
        Run main | Debug main | Run | Debug
5       public static void main(String args[]) {
6           ArrayList<Integer> kulut = new ArrayList<>();
7           kulut.add(e:20);
8           kulut.remove(index:0);
9           System.out.println(kulut);
10      }
11  }
12
```

Tässä ensin lisätään listaan kulut arvo 20, ja sen jälkeen se poistetaan. Nyt kun lista tulostetaan, se on tyhjä.

Muokkaaminen

Taulukon arvoja voi muokata samalla tavoin kuin aikaisemmin, eli kulut[muokattavan arvon indeksi] = haluttu uusi arvo.

Listoissa muokkaaminen tapahtuu metodilla set(), joka ottaa ensimmäiseksi arvokseen indeksin, jossa muokattava arvo sijaitsee, ja toiseksi uuden arvon, joka haluttuun indeksiin halutaan laittaa. Alla ensiksi lisätään listaan numero 20, jonka jälkeen se muutetaan numeroksi 50.

```
1
2   import java.util.ArrayList;
3
4   public class Main {
        Run main | Debug main | Run | Debug
5       public static void main(String args[]) {
6           ArrayList<Integer> kulut = new ArrayList<>();
7           kulut.add(e:20);
8           kulut.set(index:0, element:50);
9           System.out.println(kulut);
10      }
11  }
12
```

Nyt tulostus näyttää tältä.

```
\Temp\vscodesws_
[50]
```

HashMap

HashMap on tietorakenne, jossa tiedot tallennetaan avain-arvo-pareina. Se on hyödyllinen, kun halutaan nopeasti hakea arvo tietyn avaimen perusteella. HashMapin avulla voidaan lisätä, poistaa ja hakea tietoja tehokkaasti.

Budjetointisovelluksessa HashMapia voidaan käyttää tulojen ja menojen tallentamiseen. Avain-arvo-parit mahdollistavat tietynlaisten tulojen ja menojen nopean käsittelyn.

Jotta HashMap-tietorakennetta voi käyttää, tulee se hakea java.util-paketista komennolla import java.util.HashMap. Määrittely on samankaltainen kuin listankin, mutta <>-merkkien sisään tulee lisätä sekä avaimen että arvon tietotyyppi.

```
1
2   import java.util.HashMap;
3
4   public class Main {
        Run main | Debug main | Run | Debug
5       public static void main(String args[]) {
6           HashMap<String, Integer> budjetti = new HashMap<>();
7           
8       }
9   }
10
```

HashMapin operaatioita

HashMapiin lisätään tietoja komennolla put, joka ottaa ensimmäiseksi parametrikseen avaimen ja toiseksi parametrikseen arvon, joka siihen avaimeen halutaan liittää. Lisätään budjetti-HashMap-piin erilaisia kuluja ja tuloja.

```
1
2   import java.util.HashMap;
3
4   public class Main {
        Run main | Debug main | Run | Debug
5       public static void main(String args[]) {
6           HashMap<String, Integer> budjetti = new HashMap<>();
7           budjetti.put(key:"Vuokra", value:500);
8           budjetti.put(key:"Ruoka", value:200);
9           budjetti.put(key:"Palkka", value:2000);
10          
11      }
12  }
13
```

Nyt siis esimerkiksi avaimen Vuokra arvo on 500.

HashMapista haetaan tietoja komennolla get. Haetaan budjetti-HashMapista erilaiset kulut ja tallennetaan ne muuttujaan kulut. Haetaan myös tulot ja tallennetaan ne muuttujaan tulot.

```
1
2   import java.util.HashMap;
3
4   public class Main {
        Run main | Debug main | Run | Debug
5       public static void main(String args[]) {
6           HashMap<String, Integer> budjetti = new HashMap<>();
7           budjetti.put(key:"Vuokra", value:500);
8           budjetti.put(key:"Ruoka", value:200);
9           budjetti.put(key:"Palkka", value:2000);
10
11          int kulut = budjetti.get(key:"Vuokra") + budjetti.get(key:"Ruoka");
12          int tulot = budjetti.get(key:"Palkka");
13      }
14  }
15
```

HashMapista voidaan poistaa tietoja komennolla remove, joka ottaa arvokseen poistettavan avain-arvo-parin avaimen.

```
1
2   import java.util.HashMap;
3
4   public class Main {
        Run main | Debug main | Run | Debug
5       public static void main(String args[]) {
6           HashMap<String, Integer> budjetti = new HashMap<>();
7           budjetti.put(key:"Vuokra", value:500);
8           budjetti.put(key:"Ruoka", value:200);
9           budjetti.put(key:"Palkka", value:2000);
10
11          int kulut = budjetti.get(key:"Vuokra") + budjetti.get(key:"Ruoka");
12          int tulot = budjetti.get(key:"Palkka");
13
14          budjetti.remove(key:"Vuokra");
15
16          System.out.println(budjetti);
17      }
18  }
19
```

Tulostus näyttää nyt tältä.

```
\Temp\vscodesws_a268d\jdt_
{Ruoka=200, Palkka=2000}
```

HashSet

HashSet on tietorakenne, joka tallentaa ainutlaatuisia arvoja, eli se ei salli päällekkäisiä alkioita. HashSetin avulla voidaan nopeasti tarkistaa, onko tietty arvo jo joukossa, sekä lisätä ja poistaa arvoja.

HashSet on hyödyllinen esimerkiksi, kun halutaan kategorisoida erilaisia tuloja ja menoja, sillä ei haluta, että samannimisiä kategorioita on useita. Kategorioita voisivat olla esimerkiksi palkka, sijoitustuotot, vuokra, ruoka ja niin edelleen.

Määritellään HashSet samankaltaisesti kuin listakin. HashSet täytyy myös hakea java.util-paketista.

```
1
2   import java.util.HashSet;
3
4   public class Main {
        Run main | Debug main | Run | Debug
5       public static void main(String args[]) {
6           HashSet<String> kategoriat = new HashSet<>();
7
8       }
9   }
10
```

HashSetin operaatioita

HashSetiin voidaan lisätä arvoja komennolla add. Lisätään kategorioiksi palkka, vuoka ja ruoka.

```
1
2   import java.util.HashSet;
3
4   public class Main {
        Run main | Debug main | Run | Debug
5       public static void main(String args[]) {
6           HashSet<String> kategoriat = new HashSet<>();
7
8           kategoriat.add(e:"Palkka");
9           kategoriat.add(e:"Vuokra");
10          kategoriat.add(e:"Ruoka");
11
12      }
13  }
14
```

Lisätään vielä uudestaan Palkka, ja tulostetaan kategoriat.

```java
import java.util.HashSet;

public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        HashSet<String> kategoriat = new HashSet<>();

        kategoriat.add(e:"Palkka");
        kategoriat.add(e:"Vuokra");
        kategoriat.add(e:"Ruoka");
        kategoriat.add(e:"Palkka");

        System.out.println(kategoriat);
    }
}
```

```
\Temp\vscodesws_a268d\jdt_
[Ruoka, Palkka, Vuokra]
```

Tulostuksesta huomataan, että vaikka palkka lisättiin kahdesti, se on HashSetissä vain kerran.

HashSetistä voidaan poistaa arvoja komennolla remove.

```java
import java.util.HashSet;

public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        HashSet<String> kategoriat = new HashSet<>();

        kategoriat.add(e:"Palkka");
        kategoriat.add(e:"Vuokra");
        kategoriat.add(e:"Ruoka");
        kategoriat.remove(o:"Palkka");
    }
}
```

Perusohjelmointirakenteet

Jotta arkielämän ongelmat voitaisiin muuttaa tietokoneen ymmärtämään muotoon, tarvitaan erilaisia ohjelmointirakenteita. Tällaisia ovat esimerkiksi ehtolauseet ja silmukat.

Ehtolauseiden avulla on mahdollista luoda päätöksentekoon perustuvia ohjelmistoja. Niiden avulla voidaan määritellä, mitä tapahtuu eri tilanteissa ja reagoida erilaisiin syötteisiin tai tapahtumiin. Budjetointisovelluksessa ehtolauseita voidaan käyttää esimerkiksi tarkistamaan, onko budjetti ylittynyt. Silmukat puolestaan mahdollistavat toistuvien toimintojen suorittamisen tehokkaasti ilman, että samaa koodia tarvitsee kirjoittaa uudelleen ja uudelleen. Silmukoiden avulla voidaan esimerkiksi käydä läpi kaikki kuukauden ostokset tai laskea keskimääräiset kuukausimenot.

Operaatiot

Java-kielessä on paljon erilaisia operaatioita, joita tarvitaan muuttujien käsittelyyn ja vertailuun. Yleisiä operaatioita ovat:

+ (yhteenlasku), esim. 1+2
- (erotus), esim. 4-3
* (kertolasku), esim. 5*6
/ (jakolasku), esim. 7/8
== (yhtäsuuruus), esim. a == b tarkistaa, ovatko a ja b yhtä suuria
!= (erisuuruus), esim. a != b tarkistaa, ovatko a ja b erisuuria
> (suurempi kuin), esim. a > b tarkistaa, onko a suurempi kuin b
< (pienempi kuin)
>= (suurempi tai yhtä suuri kuin), esim. a <= b tarkistaa, onko a suurempi tai yhtä suuri kuin b
<= (pienempi tai yhtä suuri kuin)
&& (ja), esim. a > b && b > c tarkistaa, että a on suurempi kuin b ja b on suurempi kuin c
|| (tai), esim. a > b || b > c tarkistaa, että a on suurempi kuin b tai b on suurempi kuin c

Ehtolauseet

Ehtolauseita tarvitaan, kun ohjelman sisällä halutaan tehdä päätöksiä. Ne mahdollistavat toimintojen ohjaamisen perustuen erilaisiin ehtoihin. Budjetointisovelluksessa voisi olla hyödyllistä esimerkiksi tarkistaa, onko budjetti ylittynyt, jonka jälkeen tulostetaan joko ”Budjetti on ylittynyt” tai ”Olet budjetissa”.

Käytetään edellisessä esimerkissä olleita muuttujia budjetti ja onkoBudjettiYlittynyt, mutta määritellään uusi muuttuja kulut. Annetaan kuluille arvo 900.

```
1   public class Main {
        Run main | Debug main | Run | Debug
2       public static void main(String args[]) {
3           int budjetti = 1000;
4           int kulut = 900;
5           boolean onkoBudjettiYlittynyt = false;
6       }
7   }
8
```

If-lauseet

If-lauseet ovat yksinkertaisimpia ehtolauseita. Ne suorittavat tietyn koodilohkon, jos ehto on tosi. Tarkistetaan nyt, ovatko kulut suuremmat kuin budjetti ja jos ovat, vaihdetaan muuttujan onkoBudjettiYlittynyt arvo todeksi (true). Käytetään operaattoria suurempi kuin (>).

```
1   public class Main {
        Run main | Debug main | Run | Debug
2       public static void main(String args[]) {
3           int budjetti = 1000;
4           int kulut = 900;
5           boolean onkoBudjettiYlittynyt = false;
6
7           if(kulut > budjetti) {
8               onkoBudjettiYlittynyt = true;
9           }
10      }
11  }
12
```

If-lause kirjoitetaan muodossa if, sulut (), joiden sisälle itse ehto ja aaltosulkeet {}, joiden sisälle kirjoitetaan koodi, mikä tapahtuu, jos ehto on tosi. Ehto kulut > budjetti tarkoittaa siis, että jos kulut ovat suuremmat kuin budjetti, niin toteutetaan rivi 8 eli onkoBudjettiYlittynyt = true. Nyt, koska kulut (900) ovat pienemmät kuin budjetti (1000), ei riviä 8 toteuteta, eli onkoBudjettiYlittynyt on edelleen epätosi (false). Tulostetaan muuttuja onkoBudjettiYlittynyt ja tarkistetaan, että riviä 8 ei ole toteutettu.

```java
public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        int budjetti = 1000;
        int kulut = 900;
        boolean onkoBudjettiYlittynyt = false;

        if(kulut > budjetti) {
            onkoBudjettiYlittynyt = true;
        }

        System.out.print(onkoBudjettiYlittynyt);
    }
}
```

```
\Temp\vscodesws_d37e6\
false
```

Terminaalissa on ohjelman ajamisen jälkeen siis arvo false.

Else-if- ja else-lauseet

Else-if- ja else-lauseet ovat if-lauseen jälkeen käytettäviä lauseita. Mikäli if-lauseen ehto ei toteutunut, siirrytään else-if-ehtoon/ehtoihin, ja mikäli nekään eivät täyty, siirrytään else-lohkoon. Else-lohkossa ei enää tarkisteta mitään ehtoa, vaan sen sisältö toteutetaan aina, mikäli if- tai else-if-lauseiden ehdot eivät täyty.

Määritellään uudet merkkijonomuuttujat, joiden nimet ovat budjettiYlittynytViesti ja oletBudjetissaViesti.

```java
public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        int budjetti = 1000;
        int kulut = 900;
        boolean onkoBudjettiYlittynyt = false;
        String budjettiYlittynytViesti = "Budjetti on ylittynyt";
        String oletBudjetissaViesti = "Olet budjetissa";

        if(kulut > budjetti) {
            onkoBudjettiYlittynyt = true;
        }

        System.out.print(onkoBudjettiYlittynyt);
    }
}
```

Käytetään onkoBudjettiYlittynyt-muuttujaa ja tarkistetaan, onko sen arvo tosi vai epätosi. Mikäli arvo on tosi, tulostetaan viesti ”Budjetti on ylittynyt”, ja mikäli arvo on epätosi, tulostetaan ”Olet budjetissa”.

```
public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        int budjetti = 1000;
        int kulut = 900;
        boolean onkoBudjettiYlittynyt = false;
        String budjettiYlittynytViesti = "Budjetti on ylittynyt";
        String oletBudjetissaViesti = "Olet budjetissa";

        if(kulut > budjetti) {
            onkoBudjettiYlittynyt = true;
        }

        if(onkoBudjettiYlittynyt == true) {
            System.out.print(budjettiYlittynytViesti);
        } else if(onkoBudjettiYlittynyt == false) {
            System.out.print(oletBudjetissaViesti);
        }

    }
}

```

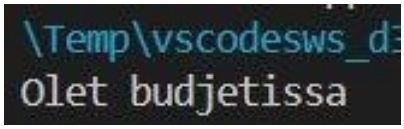
Terminaalissa tulostus näyttää siis tältä.

```
\Temp\vscodesws_d3
Olet budjetissa
```

Huomataan, että koodia voitaisiin hieman lyhentää, sillä onkoBudjettiYlittynyt-muuttujan on mahdollista saada vain kaksi erilaista arvoa. Jos siis onkoBudjettiYlittynyt on tosi, tulostetaan budjettiYlittynytViesti, ja kaikissa muissa tilanteissa (eli kun onkoBudjettiYlittynyt on epätosi) tulostetaan oletBudjetissaViesti. Nyt voidaan käyttää lyhyemmin vain if- ja else-lausetta, kuten alla.

```
public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        int budjetti = 1000;
        int kulut = 900;
        boolean onkoBudjettiYlittynyt = false;
        String budjettiYlittynytViesti = "Budjetti on ylittynyt";
        String oletBudjetissaViesti = "Olet budjetissa";

        if(kulut > budjetti) {
            onkoBudjettiYlittynyt = true;
        }

        if(onkoBudjettiYlittynyt == true) {
            System.out.print(budjettiYlittynytViesti);
        } else {
            System.out.print(oletBudjetissaViesti);
        }

    }
}

```

Tulostus on sama kuin ennen.



Else-if-lauseita voi myös olla useita peräkkäin. Tehdään seuraavaksi ohjelma, joka tulostaa erilaisia viestejä sen perusteella, ollaanko budjetissa vai onko budjetti ylittynyt vähän vai paljon. Jos budjetti on ylittynyt 0-10 %, tulostetaan viesti ”Budjetti on ylittynyt vähän”, jos budjetti on ylittynyt 11-15 %, tulostetaan viesti ”Budjetti on ylittynyt jonkin verran” ja jos budjetti on ylittynyt yli 15 %, tulostetaan viesti ”Budjetti on ylittynyt paljon”. Määritellään uusi muuttuja viesti ja lisätään sille arvo sen perusteella, kuinka paljon budjetti on ylittynyt tai alittunut. Muuttuja viesti voidaan määritellä alussa tyhjäksi ja vasta tarkistuksen jälkeen annetaan sille arvo.

```java
public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        int budjetti = 1000;
        int kulut = 900;
        boolean onkoBudjettiYlittynyt = false;
        String viesti;

        if(kulut > budjetti) {
            onkoBudjettiYlittynyt = true;
        }

        if(onkoBudjettiYlittynyt && kulut <= budjetti * 1.1) {
            viesti = "Budjetti on ylittynyt vähän";
        } else if(onkoBudjettiYlittynyt && kulut <= budjetti * 1.15){
            viesti = "Budjetti on ylittynyt jonkin verran";
        } else if (onkoBudjettiYlittynyt && kulut > budjetti * 1.15) {
            viesti = "Budjetti on ylittynyt paljon";
        } else {
            viesti = "Olet budjetissa";
        }

        System.out.print(viesti);
    }
}
```

Nyt täytyy tarkistaa siis kaksi ehtoa kerrallaan, onko budjetti ylipäätään ylittynyt, ja sen jälkeen täytyy vielä tarkistaa, kuinka paljon se on ylittynyt. Käytetään hyväksi aiempaa onkoBudjettiYlittynyt-muuttujaa. Sen jälkeen lisätään operaatio && (ja) ja lasketaan, ovatko kulut pienemmät kuin 110 % budjetista, eli kulut <= budjetti*1.1. Jos molemmat ehdot ovat tosia, toteutetaan ehtolauseen sisällä oleva lohko. Mikäli toinen tai molemmat eivät ole tosia, siirrytään seuraavaan lohkoon. Jos yksikään ehdoista ei ole tosi, toteutetaan else-lohko, eli viesti saa arvon ”Olet budjetissa”. Lopuksi vielä tulostetaan viesti.

Silmukat

Silmukoita (englanniksi loop) käytetään, jos halutaan toistaa tiettyä koodia useita kertoja.

For-silmukka

For-silmukkaa käytetään sellaisissa tilanteissa, joissa tiedetään, kuinka monta kertaa silmukka täytyy suorittaa. Se sopii esimerkiksi tilanteeseen, jossa taulukon koko tiedetään ja halutaan käydä läpi kaikki taulukon arvot.

Määritellään taulukko, joka sisältää kuluja, ja tulostetaan ne silmukan avulla. For-silmukan määrittely alkaa sanalla for, jonka jälkeen tulee sulut (), joiden sisälle kirjoitetaan alustus, ehto ja päivitys. Alustuksessa annetaan muuttuja, jota käytetään silmukan hallintaan. Alla olevassa esimerkissä siis alustetaan muuttuja i = 0, eli kun silmukka alkaa, i:n arvo on 0. Ehto on looginen, ja se tarkistetaan aina ennen silmukan suoritusta: jos ehto on tosi, silmukan suorittaminen jatkuu. Alla ehto on i<6, eli joka silmukan kierroksella tarkistetaan, että onko i pienempi kuin 6 ja jos on, jatketaan suoritusta. Viimeinen silmukan parametri on päivitys, ja se suoritetaan aina jokaisen silmukan kierroksen jälkeen. Alla olevassa esimerkissä siis lisätään jokaisen kierroksen jälkeen i:hin 1. Tämä on lyhennetty muotoon i++.

```
1
2   public class Main {
        Run main | Debug main | Run | Debug
3       public static void main(String args[]) {
4           int[] kulut = {500, 600, 700, 1000, 600, 1000};
5
6           for(int i = 0; i < 6; i++) {
7               System.out.println(kulut[i]);
8           }
9       }
10  }
11
```

Tulostus näyttää tältä.

```
500
600
700
1000
600
1000
```

For-silmukkaa voi budjetointisovelluksessa käyttää myös esimerkiksi kokonaiskulujen laskemiseen. Määritellään muuttuja kokonaiskulut, lasketaan silmukan avulla sille tulos ja tulostetaan se.

```
1
2  public class Main {
       Run main | Debug main | Run | Debug
3      public static void main(String args[]) {
4          int[] kulut = {500, 600, 700, 1000, 600, 1000};
5          int kokonaiskulut = 0;
6
7          for(int i = 0; i < 6; i++) {
8              kokonaiskulut += kulut[i];
9          }
10
11         System.out.println(kokonaiskulut);
12     }
13 }
14
```

Tulostus näyttää tältä.

```
4400
```

While-silmukka

While-silmukkaa käytetään toistamaan tiettyä koodilohkoa niin kauan kuin määritelty ehto on tosi. Tällainen silmukka on hyödyllinen silloin, kun ei tiedetä etukäteen, kuinka monta kertaa toistoa tarvitaan.

While-silmukan syntaksi on yksinkertainen. Määritellään while-silmukka, joka tulostaa numerot 0-4.

```
1
2  public class Main {
       Run main | Debug main | Run | Debug
3      public static void main(String args[]) {
4          int i = 0;
5          while (i < 5) {
6              System.out.println(i);
7              i++;
8          }
9      }
10 }
```

Ehto-kohtaan kirjoitetaan lauseke, joka arvioidaan joka ”kierroksella” todeksi tai epätodeksi. Niin kauan kuin ehto on tosi, silmukassa oleva koodi suoritetaan uudelleen. Kun ehto muuttuu

epätodeksi, silmukan suoritus päättyy. Yllä olevassa esimerkissä siis tarkistetaan joka kierroksella, onko i pienempi kuin 5, ja joka kierroksen lopussa i:hin lisätään yksi. Tulostus näyttää nyt tältä.

```
0
1
2
3
4
```

While-silmukka on hyödyllinen esimerkiksi sellaisessa tilanteessa, jossa käyttäjältä halutaan kysyä jonkinlaista syötettä, kunnes se on hyväksyttävä. Tällöin voidaan käyttää komentoa break, jolla silmukasta voidaan poistua, kun haluttu ehto täyttyy. Määritellään nyt muuttujat lukija, kokonaismenot ja budjetti. Kysytään käyttäjältä kuluja ja lisätään ne kokonaismenoihin, kunnes budjetti ylittyy ja silmukan suorittaminen päättyy. Lopuksi vielä tulostetaan kokonaismenot.

```java

import java.util.Scanner;

public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        Scanner lukija = new Scanner(System.in);
        int kokonaismenot = 0;
        int budjetti = 1000;


        while (true) {
            System.out.println(x:"Syötä kulu: ");
            int kulu = lukija.nextInt();
            kokonaismenot += kulu;

            if (kokonaismenot > budjetti) {
                System.out.println(x:"Budjetti on ylittynyt");
                break;
            }
        }

        System.out.println("Kokonaismenot: " + kokonaismenot);
    }
}
```

Tulostus voi näyttää esimerkiksi tältä.

```
Syötä kulu:
1100
Budjetti on ylittynyt
Kokonaismenot: 1100
```

While-silmukan ehdon kanssa tulee olla varovainen. On mahdollista, että silmukan ehto ei ole koskaan epätosi, jolloin silmukan suoritus jatkuu ikuisesti. Jos silmukan sisälle esimerkiksi unohdetaan kirjoittaa i++ tai i--, saattaa silmukka pyöriä ikuisesti. Myös esimerkiksi silloin, jos ehdon asettaa i > 0 ja silmukan sisällä i:tä kasvatetaan joka kierroksella, on silmukka loputon.

Oma pieni Java-sovellus

Tässä osiossa yhdistetään kaikki aiemmin opitut taidot ja rakennetaan pieni Java-sovellus. Tämä sovellus tulee hyödyntämään muuttujia, tietorakenteita, ehtolauseita ja silmukoita. Harjoitus auttaa ymmärtämään, miten eri ohjelmointikonseptit liittyvät toisiinsa ja miten niitä voidaan käyttää yhdessä todellisen ongelman ratkaisemiseksi.

Otetaan esimerkiksi yksinkertainen budjetinseurantasovellus. Se seuraa käyttäjän budjettia sekä kuluja. Käyttäjä voi syöttää kuukausittaisen budjetin sekä yksittäiset kulut, ja sovellus laskee, kuinka paljon rahaa on jäljellä. Aloitetaan hyvin yksinkertaisesta sovelluksesta, jonka jälkeen laajennetaan sitä vaihe vaiheelta monimutkaisemmaksi.

Budjetin ja kulujen määrittely

Aloitetaan määrittelemällä budjetti ja kulut, lasketaan jäljellä oleva budjetti ja tulostetaan lopuksi se.

```
1
2
3   public class Main {
        Run main | Debug main | Run | Debug
4       public static void main(String args[]) {
5           double budjetti = 1000.0;
6           double kulu1 = 100.0;
7           double kulu2 = 500.0;
8           double kulu3 = 150.0;
9
10          double budjetista_jäljellä = budjetti - (kulu1+kulu2+kulu3);
11
12          System.out.println("Kuukausibudjetti: " + budjetti);
13          System.out.println("Kulut yhteensä: " + (kulu1+kulu2+kulu3));
14          System.out.println("Jäljellä oleva budjetti: " + budjetista_jäljellä);
15      }
16  }
```

Tulostus näyttää tältä.

```
Kuukausibudjetti: 1000.0
Kulut yhteensä: 750.0
Jäljellä oleva budjetti: 250.0
```

Käyttäjän syötteen käsittely

Lisätään seuraavaksi sovellukseen mahdollisuus käyttäjän syötteen käsittelyyn. Luetaan ensin käyttäjältä budjetti, jonka jälkeen kysytään kulut kolme kertaa. Lopuksi tulostetaan samat asiat kuin aiemminkin.

```java
import java.util.Scanner;
public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        Scanner lukija = new Scanner(System.in);

        System.out.println(x:"Syötä kuukausibudjetti: ");
        double budjetti = lukija.nextDouble();

        System.out.println(x:"Syötä ensimmäinen kulu: ");
        double kulu1 = lukija.nextDouble();

        System.out.println(x:"Syötä toinen kulu: ");
        double kulu2 = lukija.nextDouble();

        System.out.println(x:"Syötä kolmas kulu: ");
        double kulu3 = lukija.nextDouble();

        double budjetista_jäljellä = budjetti - (kulu1+kulu2+kulu3);

        System.out.println("Kuukausibudjetti: " + budjetti);
        System.out.println("Kulut yhteensä: " + (kulu1+kulu2+kulu3));
        System.out.println("Jäljellä oleva budjetti: " + budjetista_jäljellä);
    }
}
```

Jatkuvan syötteen käsittely silmukalla

Lisätään vielä while-silmukka, joka mahdollistaa kulujen syöttämisen, kunnes budjetti on käytetty. Kysytään alussa kuukausibudjetti samoin kuin edellä, mutta käytetään sen jälkeen while-silmukkaa kulujen syöttämiseen. Silmukan sisällä päivitetään kokonaiskuluja (kulut) ja budjettia ja tulostetaan aina joka kierroksen lopussa jäljellä olevan budjetin määrä. Mikäli käyttäjä syöttää 0 tai budjetti ylittyy, silmukan suoritus keskeytyy. Mikäli budjetti ylittyy, merkataan kuitenkin, että jäljellä oleva budjetti on 0 eikä esimerkiksi -200.

```java
import java.util.Scanner;
public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String args[]) {
        Scanner lukija = new Scanner(System.in);
        System.out.println(x:"Syötä kuukausibudjetti: ");
        double budjetti = lukija.nextDouble();
        double kulut = 0.0;

        while(budjetti > 0) {
            System.out.println(x:"Syötä kuluerä tai 0 lopettaaksesi: ");
            double kulu = lukija.nextDouble();
            if(kulu == 0) {
                break;
            }
            kulut += kulu;
            budjetti -= kulu;
            if(budjetti < 0) {
                budjetti = 0;
            }
            System.out.println("Jäljellä oleva budjetti: " + budjetti);
        }

        System.out.println("Kulut yhteensä: " + kulut);
        System.out.println("Jäljellä oleva budjetti: " + budjetti);
    }
}
```

Tulostus näyttää esimerkiksi tältä.

```
Syötä kuukausibudjetti:
1000
Syötä kuluerä tai 0 lopettaaksesi:
500
Jäljellä oleva budjetti: 500.0
Syötä kuluerä tai 0 lopettaaksesi:
600
Jäljellä oleva budjetti: 0.0
Kulut yhteensä: 1100.0
Jäljellä oleva budjetti: 0.0
```

HashMapin käyttö

Laajennetaan sovellusta vielä lisää ja käytetään Hashmapeja kuvaamaan kulujen jakautumista. Avaimina käytetään erilaisia kategorioita, kuten vuokra tai ruoka, ja arvoikseen HashMap saa

kuhunkin kategoriaan kulutetun yhteissumman. Lopussa käydään läpi, miten for-silmukalla voidaan käydä läpi HashMap-rakenne.

Määritellään HashMap nimeltä kulutKategorioittain, joka sisältää kaikkien eri kategorioiden kulut summattuna yhteen. Jos käyttäjä syöttää esimerkiksi kategoriaan ”Ruoka” enemmän kuin yhden kuluerän, se lisätään avaimen ”Ruoka” arvoon.

```
1  import java.util.HashMap;
2  import java.util.Scanner;
3  public class Main {
       Run main | Debug main | Run | Debug
4      public static void main(String args[]) {
5          Scanner lukija = new Scanner(System.in);
6          System.out.println(x:"Syötä kuukausibudjetti: ");
7          double budjetti = Double.parseDouble(lukija.nextLine());
8          double kulut = 0.0;
9          HashMap<String, Double> kulutKategorioittain = new HashMap<>();
10         while(budjetti > 0) {
11             System.out.println(x:"Syötä kategorian nimi tai lopeta: ");
12             String kategoria = lukija.nextLine();
13             if(kategoria.equalsIgnoreCase(anotherString:"lopeta")){
14                 break;
15             }
16             System.out.println("Syötä kategorian " + kategoria + " kulu: ");
17             double kulu = Double.parseDouble(lukija.nextLine());
18             kulut += kulu;
19             budjetti -= kulu;
20             if(!kulutKategorioittain.containsKey(kategoria)) {
21                 kulutKategorioittain.put(kategoria, kulu);
22             } else {
23                 double uusisumma = kulutKategorioittain.get(kategoria) + kulu;
24                 kulutKategorioittain.replace(kategoria, uusisumma);
25             }
26             if(budjetti < 0) {
27                 budjetti = 0;
28             }
29             System.out.println("Jäljellä oleva budjetti: " + budjetti);
30         }
```

Rivillä 20 käytetään ehdossa kahta uutta asiaa, joita ei vielä olla käyty materiaalissa läpi. Huutomerkki ehtoa ennen tarkoittaa, että if-lauseen alapuolella oleva lohko toteutetaan, mikäli ehto on epätosi. ContainsKey-metodi hakee kaikki HashMapin avaimet, ja tarkistaa, löytyykö kategoria avaimista. Mikäli siis ei löydy, lisätään kategoria sekä sen kulut HashMapiin. Jos kulu löytyi jo HashMapista, haetaan ensin vanha kulujen summa HashMapista metodilla get() ja lisätään siihen sen jälkeen uusi kulu. Tämän jälkeen vaihdetaan HashMapin kategorian summa uudeksi summaksi.

```
29                 System.out.println("Jäljellä oleva budjetti: " + budjetti);
30             }
31             System.out.println(x:"Kulut kategorioittain: ");
32             for(String kategoria : kulutKategorioittain.keySet()) {
33                 System.out.println(kategoria + ": "+ kulutKategorioittain.get(kategoria));
34             }
35             System.out.println("Kulut yhteensä: " + kulut);
36             System.out.println("Jäljellä oleva budjetti: " + budjetti);
37         }
38 }
```

Rivillä 32 on uudenlainen for-silmukka, jolla läpikäydään HashMap-tietorakennetta. Ensimmäisenä argumenttina annetaan tietotyyppi sekä nimi iteroitavalle asialle, jonka jälkeen kirjoitetaan puolipiste. Tämän jälkeen kirjoitetaan iteroitavan HashMapin nimi, ja koska tässä tilanteessa iteroidaan vain avaimia, käytetään metodia keySet(). Nyt jokaisella for-silmukan kierroksella käsitellään eri kategoriaa ja voidaan täten tulostaa eri kategorioiden kulujen summat yksitellen.

Tämän sovelluksen avulla harjoiteltiin oppaassa käytyjä asioita yhdistäen ne käytännölliseksi kokonaisuudeksi. Sovellus osoittaa sen, miten Javaa ja sen konsepteja voidaan hyödyntää todellisten ongelmien ratkaisemiseksi. Käytiin myös muutamia uusia asioita läpi, jotka näyttivät, mitä muuta Javalla voi tehdä, ja jotka toivottavasti motivoivat opiskelemaan Javaa vielä lisää. Sovellus rakennettiin vaiheittain, jotta on helpompaa nähdä, miten lisäominaisuudet ja monimutkaisuus tuodaan ohjelmaan hallitusti. Samalla tämä harjoitus on hyvä pohja mahdolliselle jatkokehitykselle.

Jatko-opintomahdollisuudet

Tämä opas on tarjonnut perustan Java-ohjelmointiin, mutta ohjelmoinnin maailma on laaja ja monimutkainen. Oppaan jälkeen on vielä monia muita osa-alueita, joita kannattaa opiskella syventääkseen osaamista ja kehittääkseen taitoja edelleen. Kannattaa hyödyntää erilaisia nettilähteitä, kuten GeeksForGeeks, Baeldung, KhanAcademy tai W3Schools. Alla on lueteltu erilaisia aihealueita, joihin kannattaa perehtyä tämän oppaan lukemisen jälkeen.

Oppaassa ei käsitelty olio-ohjelmoinnin peruskonsepteja, kuten luokkia, objekteja, perintää tai polymorfismia. Olio-ohjelmointi on keskeinen osa Javaa ja monia muita ohjelmointikieliä. Sen opiskelu auttaa ymmärtämään ohjelmointia syvällisemmin.

Testaus ja virheenkäsittely jätettiin myös tämän oppaan rajauksen ulkopuolelle. Testauksen oppiminen auttaa varmistamaan ohjelman toimivuuden erilaisissa tilanteissa. Virheenkäsittely taas mahdollistaa ongelmien käsittelyn ajon aikana hallitusti.

Versionhallinta on keskeinen osa ohjelmistokehitystyötä, erityisesti tiimityöskentelyssä. Git on suosittu versionhallintajärjestelmä, joka kannattaa opetella. Gitiä käytetään usein komentoriviltä, joten perusteet siihen liittyen kannattaa myös opiskella.

Paljon onnea jatko-opintoihin!