Pauli Laiho

Luokittelu ja perintä reaalimaailmasta oliosuuntautuneeseen ohjelmointiin

Tietotekniikan Kandidaatintutkielma

13. kesäkuuta 2022

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Pauli Laiho

Yhteystiedot: pauli.s.laiho@student.jyu.fi

Ohjaaja: Jonne Itkonen

Työn nimi: Luokittelu ja perintä reaalimaailmasta oliosuuntautuneeseen ohjelmointiin

Title in English: Classification and inheritance from reality to object-oriented programming

Työ: Kandidaatintutkielma

Sivumäärä: 25+0

Tiivistelmä: Oliosuuntautunut ohjelmointi on erittäin suosittu tapa mallintaa ohjelmistoja. Oliot ja oliomainen ajattelutapa on kuitenkin merkittävästi ohjelmointia vanhempia käsitteitä ja niitä on tutkittu laajasti sekä filosofiassa että muiden tieteenalojen sovelluksissa. Tässä kirjallisuuskatsauksessa käsitellään kahta olio-ohjelmoinnin perusrakennetta: luokittelua ja perintää. Tutkielmassa rajataan filosofiasta näihin relevantit ajatusmallit ja verrataan niitä olio-ohjelmoinnin eri toteutusmalleihin. Näin pyritään saavuttamaan laajempi ymmärrys olio-ohjelmoinnin perusideasta.

Avainsanat: oliosuuntautuneisuus, prototyyppipohjaisuus, olio, prototyyppi, luokka, luokittelu, perintä

Abstract: Object-oriented programming is a particularly popular way of modeling software. Objects and object-based thinking are however much older concepts and have been widely explored in both philosophy and in other branches of science. This literature review covers two basic elements of object-oriented programming: classification and inheritance. The thesis outlines relevant philosophical thought models to these elements and compares them to different implementations of object-oriented programming. In this way the thesis aims to provide a more comprehensive understanding on the basic concepts of object-oriented programming.

Keywords: object-oriented, prototype-based, object, prototype, class, classification, inheritance

Sisällys

1 Johdanto

Oliosuuntautunut ohjelmointi on laajasti käytetty ohjelmointiparadigma eli tapa mallintaa tietokoneohjelmaa. Oliosuuntautunut ohjelmointi perustuu olioihin, jotka ovat itsenäisiä kokonaisuuksia muistin eli attribuuttien, toiminnan eli metodien sekä identiteetin suhteen. Oliosuhtautuneelle ohjelmoinnille ei ole yhtä hyväksyttyä määritelmää ja useat lähteet antavat sille jopa jossain määrin keskenään ristiriitaisia määritelmiä. Lisäksi oliokielten välillä voi olla suuriakin eroja toimintaperiaatteiden ja ajattelutapojen suhteen. Lähtökohtaisesti oliokieleltä vaaditaan ainakin käytöksen jakamista ja käytöksen piilottamista. Käytöksen jakaminen tarkoittaa sitä, että samankaltaisten olioiden yhteinen käytös voidaan määritellä vain yhdessä paikassa, mikä yleensä, mutta ei aina, toteutetaan luokkien ja perinnän kautta. Käytöksen piilottamisella viitataan siihen, että oliot ovat itsenäisiä kokonaisuuksia ja niiden muistin tila on lähtökohtaisesti vain olion itsensä tiedossa. Oliot kommunikoivat keskenään lähettämällä toimintapyyntöjä viestien avulla toisilleen. Käytöksen piilottamisen vuoksi vastuu toiminnan oikeellisuudesta on ainoastaan vastaanottavalla oliolla.

Oliosuuntautunut ohjelmointi on ollut jo usean vuosikymmenen ajan merkittävä ja suosittu ohjelmointiparadigma. Oliosuuntautuneen ohjelmoinnin perusidea alkoi hahmottua 1960-luvun alussa muun muassa simulointiin tarkoitetun Simula-ohjelmointikielen ja graafiseen suunnitteluun tarkoitetun SketchPad-ohjelmiston muodossa. Merkittävä kehitysaskel olio-ohjelmoinnissa tapahtui 1970-luvulla Smalltalk-ohjelmointikielen myötä, jonka kehitysprosessista myös oliosuuntautunut ohjelmointi terminä juontuu. Sittemmin olio-ohjelmoinnin periaatteet on integroitu useihin ohjelmointikieliin, ja nykypäivänä monet suosituimmista ohjelmointikielistä voidaan määritellä olio-ohjelmointikieliksi. Nykyaikaiset kielet ovat usein moniparadigmaisia eli ne mahdollistavat usean eri tavan ohjelmoida.

On kuitenkin huomioitava, ettei yksikään ohjelmointiparadigma ole objektiivisesti muita parempi, vaan jokaisella on omat vahvuutensa, heikkoutensa sekä käyttötarkoituksensa. Olio-ohjelmoinnin vahvuuksia ovat ohjelmakoodin uudelleenkäytettävyys ja tiivistäminen esimerkiksi luokkien ja perinnän avulla sekä suurtenkin kokonaisuuksien hallitseminen ja jakaminen pienempiin osiin kapseloinnin kautta (Guimarães 1995). Olio-ohjelmointi on toimintaperiaatteensa ja ajattelumallinsa vuoksi esimerkiksi erityisen luonteva tapa mallintaa reaali-

maailmaa. Yksi oliosuuntautuneen ohjelmoinnin merkittävä heikkous on sen ymmärtämisen kompleksisuus; erityisesti luokan ja olion välinen suhde (Thomasson, Ratcliffe ja Thomas 2006; Xinogalos 2015) sekä siirtyminen proseduraalisesta ohjelmointimallista oliomaiseen (Sheetz ym. 1997) koetaan hankalaksi ymmärtää ja oppia.

Tässä tutkielmassa tutkitaan olioita, luokittelua ja perintää vertaamalla näiden ajatusmalleja samankaltaisiin filosofisen ajattelun malleihin kirjallisuuskatsauksen keinoin. Tutkielman tarkoituksena on hahmottaa oliosuuntautunutta ajatusmallia myös tietotekniikan ulkopuolella esiintyvän vanhemman ja laajemman tutkimuksen avulla, jotta voidaan tunnistaa yhtäläisyydet ja eroavaisuudet. Tunnistamalla yhteys ohjelmoinnin ja filosofisen ajattelun välillä voidaan löytää vaikutteita ja näkökulmia oliomaiseen ajatteluun, ja näiden avulla voidaan saavuttaa syvempi käsitteellinen ymmärrys oliosuuntautuneen suunnittelun ja ohjelmoinnin tueksi.

Luvussa 2 käsitellään olioiden, luokittelun ja perinnän historiallista kontekstia filosofisen näkökulman kautta tutkimalla ontologian kahta haaraa – käsiterealismia ja nominalismia. Ontologian lisäksi luvussa pohjustetaan syllogistista logiikkaa ja kognitiotieteisiin kuuluvaa prototyyppiteoriaa. Luvussa 3 käydään läpi olio-ohjelmoinnin rakennetta ja luokittelun eri toteutuskeinoja sekä verrataan niitä filosofiassa havaittuihin malleihin. Luvussa 4 esitellään perinnän periaatteita ja toteutuskeinoja sekä verrataan niitä niin ikään filosofisiin malleihin. Luku 5 sisältää yhteenvedon ja ehdotuksia mahdollisista jatkotutkimuksista.

2 Oliot ohjelmoinnin ulkopuolella

Yksi filosofian keskeisimmistä osa-alueista tutkii olemassaolon, yksilöiden ja kaikkeuden ominaisuuksia. Tämä filosofian haara tunnetaan yleisesti metafysiikkana ja se juontaa juurensa antiikin Kreikkaan. Metafysiikka perehtyy, muttei rajoitu pelkästään, ongelmiin olemisen, muotojen ja aineellisuuden luonteisiin ja suhteisiin, olioihin ja niiden ominaisuuksiin sekä kaikkeuden luonteeseen ja alkuun liittyen (van Inwagen ja Sullivan 2021). Olioihin ja olemassaolon luonteeseen perehtyvää metafyysikan haaraa kutsutaan ontologiaksi. Metafysiikalla on merkittävä rooli myös filosofian ulkopuolella eri tieteenaloilla, kuten esimerkiksi fysiikassa ja astronomiassa.

Tässä luvussa esitellään tiivistetysti joitain historiallisesti merkittäviä ontologian näkökulmia antiikin Kreikasta nykypäivään. Tutkielman tarkoituksena ei ole ottaa kantaa eri näkökulmien vahvuuksiin, vaan esitellä ne mahdollisimman selkeästi sekä objektiivisesti. Näkökulmia on pelkistetty käsittelemällä erityisesti olio-ohjelmoinnin suhteen mielenkiintoisiin seikkoihin. Lisäksi luvussa alustetaan ontologian ulkopuolelta kahta muuta tutkielman kontekstille merkittävää tieteenhaaraa: logiikkaa ja kognitiotieteissä esiintyvää prototyyppiteoriaa.

2.1 Platonin ideaoppi

Antiikin Kreikassa elänyttä Platonia voidaan pitää yhtenä historian merkittävimmistä filosofeista, ja hänen työnsä käsitteli hyvin laajasti niin politiikkaa, etiikkaa, tieto-oppia kuin metafysiikkaakin. Platonin teoksille ominaista on kaunokirjallinen, fiktiivisistä dialogeista ja vertauskuvista koostuvaa kirjoitustyyli. Platonin teokset ovat usein monitulkintaisia ja pyrkivät herättämään mielipiteitä ja keskustelua. Tässä tutkielmassa ei pyritä tulkitsemaan suoraan Platonin teoksia, vaan ajatusmaailma esitellään lyhyesti modernien tulkintojen kautta.

Platonin keskeisin metafysiikkaan liittyvä oppi on hänen ideaoppinsa, jonka avulla hän pyrki vastaamaan kysymykseen olemassaolon luonteesta. Krautin (2022) mukaan Platonin ideaoppi perustuu ajatukseen, että kaikkien reaalimaailmassa havainnoitavien asioiden taustalla on universaali, muuttumaton ajatus ja määritelmä, jota asia pyrkii jäljittelemään. Platonin uni-

versaalit nimetään käännöksestä riippuen joko **ideoiksi** tai **muodoiksi** (engl. *form*). Rickless (2020) täydentää, että ideat ovat muuttumattomia ja ikuisia, mutta ihmisen havaitseman reaalimaailman asiat eivät ole. Platonin mukaan täytyy olla olemassa erillinen ideamaailma, jossa universaalit ideat sijaitsevat. Asiat ovat riippuvaisia ideoista; ideamaailma toimii reaalimaailman pohjapiirustuksena. Tylman (2018) korostaa Platonin ideaopista kahdeksan sääntöä:

1. On olemassa sekä ideoita että asioita.
2. Ideat ovat ikuisia.
3. Asioita voidaan sekä luoda että tuhota.
4. Asiat ovat riippuvaisia ideoista.
5. Ideat ovat asioiden malleja.
6. Yksi idea vastaa useaa asiaa.
7. Ideat muodostavat hierarkian.
8. Hierarkian huipulla on yksi tietty idea.[1]

Ideaopin mukaan siis kaikki reaalimaailmassa aisteilla havainnoitavat asiat ovat väliaikaisia kopioita jostain ideasta, joka yhdistää samankaltaisia asioita yhden käsitteen alle. Idea taas on aistien ulkopuolella, vain järjellä ymmärrettävissä oleva käsite, joka toimii standardina kaikille sitä muistuttaville asioille.

2.2 Aristoteles

Platonin ohella toinen merkittävä antiikin filosofi oli Platonin oppilas Aristoteles Shields (2022). Aristoteles jatkoi Platonin työtä ontologian saralla ja sovelsi näitä ideoita vahvasti myös muihin tieteenaloihin, kuten biologiaan ja matematiikkaan. Lisäksi Aristoteles käsitteli teoksissaan muun muassa logiikkaa. Tässä tutkielmassa käsitellään Aristoteleen töitä sekä luokittelun että logiikan saralta.

2.2.1 Aristoteleen luokittelu

Aristoteles keskittyi tutkimaan luonnossa ilmentyviä asioita ja ilmiöitä, kuten kasveja ja eläimiä. Hän lajitteli reaalimaailman perusaineet eli substanssit luokkiin – substanssit kuuluvat

1. Platon mukaan hierarkian huipulla on *Hyvän* idea, joka tekee kaikista asioista arvokkaita ja hyödyllisiä.

samaan luokkaan, jos niillä on samat ominaisuudet. Perusaineen eli substanssin määrittäminen on oma aihealueensa filosofian tutkimuksessa (Robinson 2021), mutta tämän tutkielman kontekstissa substanssilla tarkoitetaan mitä tahansa itsenäistä asiakokonaisuutta, jonka olemassaolo ei ole riippuvainen mistään muusta asiasta.

Aristoteles (350 eaa./2000[a]) jakaa perusaineet ensisijaisiin substansseihin (engl. *primary substance*) sekä toissijaisiin substansseihin (engl. *secondary substance*). Ensisijaisia substansseja ovat kaikki asiat yksilöinä. Toissijaisella substanssilla Aristoteles tarkoittaa niitä piirteitä ja ominaisuuksia, jotka määrittelevät ensisijaisen substanssin olemuksen. Toissijaiset substanssit jaetaan edelleen kahteen eri haaraan, sukuihin (engl. *genus*) ja lajeihin (engl. *species*). Laji sisältää samankaltaisten yksilöiden yhteiset, määrittelevät piirteet, eli yksilöt kuuluvat tiettyyn lajiin. Suku puolestaan sisältää usean eri lajin tai suvun yhteiset piirteet.

Aristoteleen luokat voidaan siis määritellä yksilön ominaisuuksien lisäksi myös toisten luokkien ominaisuuksien mukaan; uusi luokka jakaa ylemmän luokan ominaisuudet ja erikoistaa sen lisäämällä uusia ominaisuuksia. Täten luokat muodostavat yksijuurisen puurakenteen, jonka huipulla on yksi korkein luokka, *olemassa oleva*. (Studtmann 2021).

2.2.2 Aristoteleen logiikka

Aristoteleen logiikka perustuu väitelauseisiin, joiden voidaan olevan yksiselitteisesti joko tosia tai epätosia. Aristoteles (350 eaa./2000[b]) määrittelee väitelauseen rakenteen yksinkertaisesti kahteen termiin, subjektiin ja predikaattiin, sekä termit yhdistävään liitossanaan. Lauseen subjekti voi olla joko yksilö tai universaali käsite, ja predikaatti on universaali käsite johon subjekti kuuluu tai jokin subjektin ominaisuus. Esimerkiksi väitelauseen *ihminen on kuolevainen* subjekti on universaali käsite *ihminen* ja predikaatti on universaali käsite *kuolevainen*, eli suku johon ihmiset kuuluvat. Väitelauseessa *Sokrates on ihminen* puhutaan tietystä henkilöstä eli subjekti on yksilö, ja predikaatti *ihminen* on laji, johon subjekti *Sokrates* kuuluu.

Väitelauseisiin perustuvaa loogista argumenttia kutsutaan myös syllogismiksi. Aristoteleen (350 eaa./2000[c]) määrittelemä syllogismi koostuu kahdesta, toisiinsa liittyvästä väitteestä eli premissistä ja niistä johdetusta päätelmästä. Ensimmäinen pääpremissi on kattavampi,

ja toinen alipremissi on suppeampi. Aristoteleen mukaan pääpremississä ja alipremississä on jokin yhteinen termi, joka yhdistää nämä väitteet toisiinsa, mutta joka ei esiinny johtopäätöksessä. Johtopäätös muodostuu niistä termeistä, jotka eivät ole premisseille yhteisiä. Esimerkki[2] syllogismista olisi:

Pääpremissi: Ihminen on kuolevainen.
Alipremissi: Sokrates on ihminen.
Johtopäätös: Sokrates on kuolevainen.

Esimerkissä yhteinen termi premissien välillä on *ihminen*, joka toimii ensimmäisen väitteen subjektina ja toisen väitteen predikaattina. Johtopäätös muodostuu toisen väitteen subjektista *Sokrates* ja ensimmäisen väitteen predikaatista *kuolevainen*. Huomioitavaa on, että syllogismi ei ota kantaa siihen, ovatko väitteet totta, vaan sen pätevyys perustuu pelkästään sen loogiseen muotoon eikä merkitykseen. Johtopäätös on totta vain silloin, jos molemmat premissit ovat totta (Smith 2020).

2.3 Nominalismi

Platonin ja Aristoteleen ajatukset olivat varhainen osa laajempaa, universaalien ongelmana tunnettua kysymystä. Universaalien ongelma hakee vastausta siihen, ovatko universaalit, eli yksilöiden jakamat yleiskäsitteet[3], oikeasti olemassa tai edes mahdollisia (Klima 2022). Platonin ja Aristoteleen ajatusmaailmat edustavat käsiterealismia, jonka mukaan universaalit ovat, ainakin jossain muodossa, todellisia. Yksi vaihtoehtoinen, keskiajalla muodostunut ajattelutapa on nominalismi.

Rodriguez-Pereyran (2019) mukaan nominalismi voidaan jakaa ainakin kahteen eri haaraan; toinen kieltää universaalien olemassaolon ja toinen kieltää abstraktien asioiden olemassaolon. Abstraktit asiat ovat keskeinen käsite metafyysikassa, mutta niille ei ole yhtä hyväksyttyä määritelmää ja varsinkin abstraktin ja konkreettisen asian välinen raja on häilyvä (Falguera, Martínez-Vidal ja Rosen 2021). Yleisesti abstrakteilla asioilla tarkoitetaan sellaisia

2. Esimerkki on yleisesti tunnettu syllogismi, jota pidetään yleisesti olevan lähtöisin Aristoteleelta, mutta oikea kirjallinen lähde lienee Mill (1843, s. 245)

3. vrt. Platonin idea, Aristoteleen toissijainen substanssi.

asioita, jotka eivät sijaitse aika-avaruudessa eli niillä ei ole fyysistä olomuotoa maailmankaikkeudessa. Täten abstrakteja asioita ei voida havaita aisteilla, eivätkä ne voi vaikuttaa mitenkään reaalimaailmaan.

Tämän tutkielman kontekstissa nominalismilla tarkoitetaan ajattelutapaa, joka kieltää universaalit. Kingin ja Arlingin (2018) mukaan ensimmäisenä merkittävänä nominalismin edustajana voidaan pitää keskiajalla elänyttä Peter Abelardia. Abelardin (n. 1120, teoksessa Spade 1994, s.37-56) mukaan mitään universaaleja ideoita ei ole olemassa, ja vaikka yksilöillä on samanlaisia piirteitä, ne eivät kuitenkaan ole jaettuja yksilöiden välillä. Abelardille universaalit ovat vain nimiä, joita käytetään kielellisesti tunnistamaan samankaltaisia asioita. Tylman (2018) yksinkertaistaa nominalismin perusajatuksen siihen, että reaalimaailmassa on nominalismin mukaan olemassa vain asioita yksilöinä. Universaalit ovat vain yleistyksiä asioiden samankaltaisuuksista, ja täten riippuvaisia yksilöiden olemassaolosta päinvastoin kuin käsiterealismin ajattelussa, jonka mukaan yksilöt perustuvat ja ovat täysin riippuvaisia universaaleista.

2.4 Prototyyppiteoria

Perinteinen, objektiivinen luokittelujärjestelmä – jota myös Aristoteleen malli edustaa – esittää, että on olemassa yksi oikea, universaali luokitteluhierarkia, jota reaalimaailma noudattaa (Lakoff 1990, s. 9-11). Perinteisen järjestelmän kategoriat selkeästi rajattuja ja yleispäteviä säiliöitä asioille, ja tietty asia yksiselitteisesti joko kuuluu tai ei kuulu tiettyyn kategoriaan. Lisäksi kaikki asiat ovat tasavertaisia kategorian sisällä, eli kaikki kategorian esiintymät edustavat kategoriaa yhtä selkeästi ja vahvasti. (Lakoff 1990, s. 40).

Roschin (1973) esittämä kognitio- ja kielitieteisiin kuuluva prototyyppiteoria on esimerkki toisenlaisesta ajattelumallista. Roschin mukaan tietyt asiat ilmentävät kategorioita paremmin kuin toiset, ja kategoriasta on olemassa paras mahdollinen esimerkkitapaus, jota kutsutaan prototyypiksi. Prototyypit ovat subjektiivisia – havainnoijan yksilölliset aistit, henkilökohtainen tausta ja alakohtainen kokemus vaikuttavat siihen, minkä asian havainnoija mieltää kategorian prototyypiksi. Prototyyppiteorian mukaisiin kategorioihin kuuluminen ei ole absoluuttista ja selkeää, vaan mitä enemmän asia muistuttaa prototyyppiä, sitä vahvemmin

sen voidaan katsoa kuuluvan kategoriaan (Osherson ja Smith 1981). Prototyyppiteoria pohjautuu vahvasti Wittgensteinin (2009) **perheyhtäläisyyteen**. Wittgensteinin (2009, s.36-38) mukaan joillain kategorioilla[4] ei ole yhtä selkeää piirrettä, joka esiintyy kaikissa kategorian asioissa ja jonka mukaan asian voidaan sanoa kuuluvan kategoriaan. Sen sijaan asioiden samankaltaiset ominaisuudet osuvat päällekkäin ja muodostavat laajan verkon, jonka ääripäillä ei välttämättä ole mitään yhteistä keskenään.

4. Wittgenstein (2009, s.36-38) nostaa esimerkiksi *pelin* käsitteen. Peleissä on paljon yhteisiä piirteitä, kuten viihdyttävyys, kilpailullisuus, taito tai tuuri, mutta mikään piirre ei esiinny kaikissa peleissä.

3 Olioiden luokittelu ohjelmoinnissa

Olio-ohjelmoinnin merkittävä vahvuus ohjelmakoodin uudelleenkäytössä ja tiivistämisessä perustuu käytöksen jakamisen periaatteeseen; olio-paradigma mahdollistaa sen, että useassa eri vaiheessa ohjelmiston suoritusta tapahtuvat samanlaiset toiminnot tarvitsee määritellä vain yhdessä paikassa (Rentsch 1982). Tämä edellyttää, että ohjelmoitaessa täytyy pystyä tunnistamaan ja esittämään samankaltaiset rakenteet yhtenä kokonaisuutena – toisin sanoen tarvitaan tapa luokitella rakenteita niiden ominaisuuksien mukaan. Huomioitavaa on kuitenkin, että olio-paradigman vastuulla on vain mahdollistaa ja tarjota työkalut luokittelun toteutukselle, mutta itse halutun rakenteen looginen suunnittelu ja toteutus ovat täysin ohjelmoijan vastuulla. Tässä luvussa esitellään kaksi eri tapaa luokitella ohjelmakoodin komponentteja ja jakaa niiden yhteinen käytös.

Olio on oliosuuntautuneen ohjelmoinnin perusyksikkö. Robson (1981) määrittää olion dataksi sekä kyseisen datan manipulaatioon käytettäviksi toiminnoiksi yhdessä paketissa. Sekä Robson (1981) että Wegner (1987) pitävät merkittävänä piirteenä juuri datan ja toiminnallisuuden yhdistämistä verrattuna perinteiseen proseduraaliseen tapaan ohjelmoida, jolloin data ja sen manipulointiin käytettävät funktiot pidetään selvästi erillään. Olio voidaan kuvitella mustana laatikkona, joka sisältää sekä muistin että käytöksen. Jokaisella oliolla on lisäksi yksilöivä identiteetti: vaikka kahdella oliolla olisi täsmälleen samat ominaisuudet, ne ovat erillisiä, itsenäisiä yksilöitä. Oliosuuntaisen ohjelmiston toiminta perustuu kokoelmaan useita olioita, jotka suorituksen aikana kommunikoivat keskenään lähettämällä ja vastaanottamalla viestejä.

3.1 Luokkapohjainen malli

Luokka on toinen merkittävä olio-ohjelmoinnin käsite. Luokka on kuvaus tai pohjapiirros siitä, minkälainen muisti ja käytös oliolla on, ja oliot luodaan luokan perusteella (Robson 1981). Jokainen olio on määritelty jossain luokassa, ja olion sanotaan olevan luokkansa ilmentymä eli instanssi. Yhdestä luokasta voidaan luoda useampia instansseja, eli luokka kokoaa yhteen samankaltaisten olioiden yhteiset ominaisuudet. (Wegner 1987; Robson 1981).

Budd (2001, s. 14-15) esittää – osittain virheellisesti Kayta (1993) lainaten – olio-ohjelmoinnille kuusi periaatetta:

1. Jokainen asia on olio.
2. Oliot vuorovaikuttavat keskenään lähettämällä ja vastaanottamalla viestejä.
3. Oliolla on toisista olioista koostuva muistitila.
4. Jokainen olio on luokan instanssi.
5. Luokka sisältää sen instanssien yhteisen käytöksen.
6. Luokat muodostavat puurakenteen, jolla on yksi juuriluokka.

Vaikka Kayn (1993) esittelemän listan kuudes periaate ei ole edellämainitun mukainen[1] ja Kayn periaatteet koskivat nimenomaan hänen kehittämäänsä Smalltalk-ohjelmointikieltä eivätkä olio-ohjelmointia yleisesti, Buddin määrittelemät periaatteet tiivistävät oliosuuntaisen ohjelmoinnin toimintaperiaatteen hyvin.

Kun verrataan edellä esitettyä Buddin (2001) listaa olio-ohjelmoinnin periaatteista alaluvussa 2.1 Tylmanin (2018) muotoilemaan listaan ideaopin periaatteista, samankaltaisuus on selkeä. Olio-ohjelmoinnin oliot vastaavat ominaisuuksiltaan Platonin reaalimaailmassa havaitsemia asioita, ja oliot määrittelevä luokka taas vastaa universaalia ideaa asiasta. Tarkka suhde olio-ohjelmoinnin sekä filosofian välillä ei kuitenkaan ole selvä. Vaikka Tylmanin mukaan ei voidakaan todistettavasti esittää olio-ohjelmoinnin ottaneen merkittävää inspiraatiota filosofiasta, Rayside ja Campbell (2000) toteaa ohjelmointipiireissä ilmaantuvan usein käsitys filosofian, erityisesti Platonin ja Aristoteleen ajatusmallien, merkityksestä olio-ohjelmoinnille. Myös Kay (1993, s. 3, 6) tiedostaa yhteyden Smalltalkin toimintalogiikan sekä Platonin filosofian välillä, mutta ei tee niiden välille suoraa vertausta.

On kuitenkin huomioitava, että edellä mainitut olio-ohjelmoinnin periaatteet kuitenkaan ole ehdottomia ja vaikka suuri osa periaatteista toteutuu suuressa osassa oliokielistä, jokainen periaate ei kuitenkaan välttämättä toteudu kaikissa kielissä. Esimerkiksi kuudennen periaatteen mukainen yhden juuriluokan hierarkia on toteutettu suurimmassa osassa[2], mutta ei

1. ”6. To eval a program list, control is passed to the first object and the remainder is treated as its message”. (Kay 1993, s. 19)

2. Esimerkiksi C#, Java, Python ja Smalltalk, joissa kaikissa juurena on *Object*-niminen luokka.

kuitenkaan kaikissa[3], olio-ohjelmointikielissä.

Tylman (2018), Rayside ja Campbell (2000) sekä Taivalsaari (1996a) vertaavat suoraan olio-ohjelmointia Platonin ideamaailmaan ja yleisesti käsiterealismiin sellaisten kielten tapauksessa, joissa luokkia käytetään säiliöimään olioiden yhteistä käytöstä. Monet oliokielet toteuttavat tämän kaltaisen eron luokan ja olion välille; luokat ovat rakenteellisesti eri asia kuin olio. Tämä ei kuitenkaan päde aivan kaikissa tapauksissa; esimerkiksi Smalltalkin tapauksessa Kay (1993) painottaa, että myös luokat ovat olioita. Smalltalkin toteutuksessa luokka on olio, jonka käytös on toisten olioiden määrittäminen. Smalltalkin toimintaperiaatteessa on täten paljon samaa Aristoteleen luokittelun kanssa; Aristoteles rinnasti vahvasti sekä yksilöt että lajit saman kategorian – substanssin – alle. Näissäkin tapauksissa ohjelmisto suunnitellaan Platonin näkökulman mukaisesti luokat ensin; luokkaan kerätään olioiden yhteiset ominaisuudet ja yksilölliset olio-instanssit luodaan tämän perusteella.

Vaikka luokkaa pidetään yleisesti erittäin keskeisenä käsitteenä olio-ohjelmoinnissa ja monet tahot, kuten Armstrong (2006) ja Wegner (1987), liittävät luokat olio-ohjelmoinnin määritelmään, oliosuuntautunut kieli on mahdollista toteuttaa myös täysin ilman luokkia. Rentsch (1982) määrittelee oliosuuntautuneen ohjelmoinnin luokkien sijasta käytöksen jakamisen perusteella. Käytöksen jakamisella Rentsch (1982) tarkoittaa sitä, että jonkin joukon yhteinen ominaisuus on määritelty vain yhdessä paikassa, ja joukon yksilöt voivat tarvittaessaan uudelleenmäärittää ominaisuuden omalla kohdallaan. Luokka on vain yksi mahdollinen tapa toteuttaa käytöksen jakaminen.

3.2 Prototyyppipohjainen malli

Yksi tapa toteuttaa käytöksen jakaminen on edustettuna prototyyppipohjaisessa ohjelmoinnissa[4]. Donyn, Malenfantin ja Cointen (1992) mukaan tärkein ero prototyyppipohjaisen ja perinteisen luokkapohjaisen toteutuksen välillä on, että käytöksen jakaminen toteutetaan konkreettisten olioiden tasolla luokkien konseptitason sijasta. Prototyyppipohjaisessa ohjelmoinnissa ei käytetä luokkia, vaan olio luodaan joko suoraan määrittämällä sen käytös

3. Esimerkiksi C++ ei toteuta lainkaan yksijuurista luokkahierarkiaa.
4. Prototyyppipohjaisia kieliä ovat esimerkiksi Smalltalkiin pohjautuva Self sekä JavaScript

olioon tai kopioimalla jo aikaisemmin luodusta oliosta uusi instanssi (Tylman 2018; Dony, Malenfant ja Cointe 1992). Oliota, josta luodaan kopio, kutsutaan **prototyypiksi**. Toisin kuin luokka, joka on vain abstrakti yleistys samankaltaisten olioiden piirteistä, prototyyppi on konkreettinen esimerkki siitä, minkälainen olio on (Dony, Malenfant ja Cointe 1992). Kun oliosta luodaan kopio, kopio jakaa käytöksen alkuperäisen olion kanssa, jollei kopion käytöstä muokata sopivaksi.

Sekä Dony, Malenfant ja Cointe (1992) että Taivalsaari (1996a) esittävät prototyyppipohjaisen ohjelmoinnin eduksi sen, että ihmiselle on luontevampaa ja yksinkertaisempaa ajatella uusi käsite varsinaisen reaalimaailman esimerkin mukaan, toisin kuin luokkapohjaisessa täytyy suunnitella abstrakti yleistys ennen olioita. Tämä vastaa luvussa 2.4 esitellyn prototyyppiteorian mukaista käsitystä siitä, miten ihmiset havainnoivat reaalimaailmaa. Lisäksi Donyn, Malenfantin ja Cointen (1992) mukaan luokat luovat liikaa käytännön rajoitteita olioille, sillä yksittäiselle olion ei voi antaa yksilöllistä käytöstä. Prototyyppipohjaisessa toteutuksessa olioiden ei tarvitse sopia absoluuttisesti tietynlaiseen muottiin, vaan voidaan myös yksilön tasolla lisätä sekä poistaa attribuutteja ja metodeja (Taivalsaari 1996a).

Tylman (2018) rinnastaa prototyyppipohjaisen ohjelmoinnin nominalismiin. Prototyyppipohjainen ohjelmointi on täysin ilman universaaleja luokkarakenteita ja ohjelma koostuu täysin yksilöllisistä olioista, mitä voidaan suoraan verrata nominalismin perusideaan. Taivalsaari (1996a) puolestaan perustelee prototyyppipohjaisen ohjelmoinnin prototyyppiteorian kautta. Prototyyppipohjaisen toteutuksen suunnittelulähtökohta on prototyyppiteorian mukainen: Ensin esitellään halutusta oliosta mahdollisimman hyvä prototyyppitapaus, josta jalostetaan kaikki muut tapaukset.

4 Perintä ohjelmoinnissa

Yleinen tapa olio-ohjelmoinnissa käytöksen jakamisen toteuttamiselle olioiden välillä on alaluvussa 3.1 esitelty luokkapohjainen toteutustapa. Olioiden samankaltaiset ominaisuudet on määritelty luokassa, toisin sanoen samankaltaiset oliot jakavat yhteiset ominaisuutensa keskenään luokan avulla. Myös keskenään samankaltaiset luokat voivat jakaa käytöksensä keskenään perinnän avulla. Perinnässä uusi luokka muodostetaan jo valmiiksi määritellyn luokan pohjalta, jolloin uusi luokka saa omien ominaisuuksiensa lisäksi käyttöönsä myös toisen luokan ominaisuudet[1].

4.1 Aliluokat ja yliluokat

Luokkaa, joka perii toisen luokan kutsutaan aliluokaksi ja niin ikään luokkaa, josta uusi luokka periytyy kutsutaan yliluokaksi (Robson 1981). Koska aliluokka sisältää yliluokan ominaisuuksien lisäksi myös omat lisäominaisuutensa, aliluokan sanotaan erikoistavan yliluokkansa, eli se on yksityiskohtaisempi ja tarkempi kuvaus mallinnettavasta oliosta. Samoin yliluokka on yleistys aliluokkien jaetuista ominaisuuksista, ja yliluokka on aliluokkiaan abstraktimpi ja pelkistetympi muoto.

Luokkapohjaisessa olio-ohjelmoinnissa käytetään paljon hyväksi **abstrakteja luokkia**. Abstrakti luokka on luokka, jossa osalle tai kaikelle luokan määritellylle käytökselle ei ole luotu toteutusta. Tällöin oletuksena on, että abstraktin luokan perivät aliluokat antavat toteutuksen puuttuville käytökselle (Taivalsaari 1996b). Taivalsaaren mukaan abstraktien luokkien pääasiallinen käyttötarkoitus on varmistaa, että aliluokat toteuttavat vaaditun käytöksen, eikä abstrakteista luokista ole tarkoituksenmukaista luoda olio-instansseja. Moni oliokieli estää poikkeuksetta olio-instanssien luomisen abstraktista luokasta, mutta esimerkiksi Smalltalkin tapauksessa kiinteää rajoitusta ei ole. Näissä tapauksissa päätäntävalta säännön noudattamisesta on täysin ohjelmoijan vastuulla.

Yli- ja aliluokan välinen suhde voidaan havainnoida *is-a* eli *on-kuin*-semanttisen yhteyden

1. Luokasta voidaan yleensä estää joko tiettyjen osien periytyminen kielestä riippuen esim. *private*-avainsanalla, tai kieltää luokan periminen kokonaan esim. *final*-avainsanalla.

avulla; jos kahden termin väliin voidaan luontevasti asettaa sana *on*, niin termien välillä on mahdollisesti perimäsuhde (Brachman 1983). Voidaan esimerkiksi sanoa, että *ihminen on kädellinen*, minkä perusteella luokka *ihminen* voidaan esittää luokan *kädellinen* aliluokkana. Brachmanin mukaan edellämainitulla tavalla voidaan myös esittää luokan ja sen instanssin välinen suhde: esimerkiksi lauseen *Aristoteles on ihminen* subjekti *Aristoteles* viittaa yksilöön eli olioon, joten se on luokan *ihminen* instanssi. Tällainen lausemuoto on suora esimerkki Aristoteleen väitelauseesta, ja jos väitteen voidaan sanoa olevan tosi, niin termien välillä oikeellinen suhde.

4.2 Perintähierarkia

Perittävä yliluokka voi myös olla jonkun kolmannen, vielä pelkistetymmän luokan aliluokka, jolloin alkuperäinen aliluokka saa käyttöönsä oman ensisijaisen yliluokkansa kautta myös tämän luokan yliluokan ominaisuudet. Luokat muodostavat täten puurakenteen muotoisen perintähierarkian. Ylempänä puussa on hyvinkin abstrakteja ja laaja-alaisia käsitteitä, joista haarautuu yksityiskohtaisempia luokkakuvauksia. Kuten luvussa 3.1 todetaan, monen oliokielen sisäänrakennettu *Object*-luokka toimii joko suoraan tai välillisesti kaikkien muiden luokkien yliluokkana. Tämän vuoksi koko ohjelmiston luokkarakenne muodostaa yhden, yksijuurisen perintähierarkian.

Vastaavasti yksijuurinen hierarkia esiintyy sekä Platonin että Aristoteleen maailmankuvissa. Tylman (2018) vertaa olio-ohjelmoinnin perintähierarkiaa Platonin ideamaailman hierarkiaan, joskaan vertaus ei ole aivan täydellinen; Platonin hierarkia rakentuu ennemmin asioiden arvon ja tärkeyden perusteella, eikä termien yleispätevyyteen kuten olio-ohjelmoinnissa. Rayside ja Campbell (2000) vertaa perintää Aristoteleen luokittelun lajeihin ja sukuihin; samoin kuin lajit ja suvut jakavat piirteensä ylemmän tason suvun kanssa, myös ali- ja yliluokat jakavat käytöksensä perinnän avulla.

Perustavanlaatuinen käsitteellinen ero Aristoteleen lajien ja sukujen sekä olio-ohjelmoinnin ali- ja yliluokkien on kuitenkin olemassa. Ali- ja yliluokka termeinä viittaavat kahden luokan väliseen suhteeseen, ja sama luokka voi kontekstista riippuen olla sekä ali- että yliluokka. Aristoteleen mukaan laji ja suku ovat konkreettisesti kaksi eri asiaa; laji koostuu pelkäs-

tään yksilöiden yhteisistä piirteistä ja suku puolestaan koostuu pelkästään lajien tai sukujen yhteisistä piirteistä. Jokainen yksilö on siis aina tietyn lajin edustaja, eikä suoraan minkään suvun edustaja. Toisin sanoen jotta olio-ohjelman luokkahierarkia vastaisi täysin tätä mallia, jokaisen yliluokan tulisi olla abstrakti, eli pelkästään niistä luokista, joilla ei ole aliluokka, voidaan luoda olio-instansseja. Hürsch (1994) soveltaa kyseistä **abstraktien yliluokkien sääntöä** olio-ohjelmiston suunnitteluun ja toteaa mallin olevan usein hyvä käytäntö noudattaa. Kuitenkin malli voi esimerkiksi rajoittaa ohjelmakoodin uudelleenkäytettävyyttä ja hankaloittaa kehitysvaiheen työskentelyä. Hürschin mukaan juuri tämän vuoksi oliokielien ei tulisi poikkeuksetta vaatia säännön noudattamista, vaan vastuu ja päätösvalta on hyvä pitää ohjelmiston suunnittelijalla.

4.3 Polymorfismi

Polymorfismi on olio-ohjelmoinnin käsite, joka mahdollistaa sen, että samalle toiminnallisuudelle voidaan määritellä erilaiset, tyypistä riippuvat toteutukset (Milner 1978). Cardelli ja Wegner (1985) esittävät useita eri tapoja polymorfismin toteutukselle eri ohjelmointiparadigmoilla, joista olio-ohjelmoinnille merkittävin on perinnän avulla saavutettava **sisällyttävä polymorfismi**. Liskov (1987) esittää, että luokkaan viitatessa ja käsiteltäessä voidaan samaa toiminnallisuutta käyttää suoraan myös kaikkiin luokan aliluokkiin. Tämä on mahdollista, koska kaikki yliluokan toiminnallisuus kuuluu myös aliluokalle joko sellaisenaan tai muokattuna. Voidaan siis olla varmoja, että aliluokalla on toteutus kaikelle käytökselle, joka on myös yliluokalla. Booch ym. (2007) pitää olio-ohjelmoinnin tehokasta ja laajaa polymorfismin hyödyntämistä merkittävänä olio-ohjelmoinnin vahvuutena.

Rayside ja Kontogiannis (2001) selittävät sisällyttävän polymorfismin klassisen Aristoteleen logiikan avulla. Sisällyttävä polymorfismin ajatus voidaan esittää suoraan perinteisenä syllogismina:

Yliluokka sisältää toiminnallisuuden.
Aliluokka perii yliluokan.
Aliluokka sisältää toiminnallisuuden.

Jos yliluokassa on määritelty jokin toiminnallisuus ja aliluokka perii luokan, voidaan siis

loogisesti olla varmoja, että myös aliluokalla on tämä toiminnallisuus. Rayside ja Kontogiannis esittävät, että Aristoteleen logiikan ymmärtäminen helpottaa selvästi myös perinnän sekä polymorfismin ymmärtämistä.

4.4 Prototyyppien perintä

Prototyyppipohjaisessa ohjelmoinnissa perintä tapahtuu suoraan olioiden välillä. Kun prototyyppinä käytettävästä oliosta luodaan kopio, kopio perii prototyypin käytöksen, joka voidaan erikoistaa muokkaamalla perittyä käytöstä ja lisäämällä uutta käytöstä (Lieberman 1986). Lieberman esittää kaksi käytännön tapaa toteuttaa prototyyppien perintä: joko kaikki käytös prototyypiltä kopioidaaan ja oliot pidetään täysin erillisinä toisistaan, tai luodaan linkki prototyypin ja kopion välille. Tällöin kaikki toiminnot, jolle kopiolla ei ole erikseen määriteltyä käytöstä, ohjataan edelleen prototyypille. Tätä prosessia kutsutaan myös **delegoinniksi**. Koska jostain toisesta oliosta tehty kopio voi toimia prototyyppinä edelleen useammalle oliolle, delegointi muodostaa puumuotoisia perintähierarkioita. Lisäksi joidenkin kielten, kuten esimerkiksi Javascriptin kohdalla tämä perintähierarkia on yksijuurinen; kielessä on sisäänrakennettu *Object*-olio, josta kaikki muut oliot periytyvät.

Rayside ja Campbell (2000) vertaavat prototyyppipohjaista perintämallia biologiassa esiintyvään Charles Darwinin evoluutioteoriaan. Samoin kuin biologisessa evoluutiossa, muutokset olioissa tapahtuvat yksilöiden tasolla. Olio mukauttaa prototyypiltään saamat piirteet sopimaan omaan toimintaansa, samoin kuin biologisessa evoluutiossa oliot kehittyvät sopeutumaan eliympäristönsä mukaisesti.

Lieberman (1986) esittää, että oikeanlaisella suunnittelulla myös delegoinnin avulla voidaan toteuttaa myös kaikki luokkapohjaisen perintämallin toiminnallisuus. Luomalla ensin toiminnallisuuden määrittelevä luokka-olio ja kopioilla halutut instanssit tästä oliosta saavutetaan tilanne, joka esiintyy myös joissain luokkapohjaisissa kielissä: myös luokat ovat olioita. Tätä suunnittelumallia voidaan taas verrata tulkinnasta riippuen joko Aristoteleen substanssiehin tai nominalistiseen näkökulmaan.

5 Yhteenveto

Tässä tutkielmassa tutkittiin luokkien ja perinnän käsitteitä olio-ohjelmoinnissa sekä esitettiin joitakin niitä vastaavia ajatusmalleja filosofiasta. Luokkapohjaista ohjelmointimallia vastaa parhaiten Aristoteleen ajattelu sekä luokittelun että logiikan saralta, ja olio-ohjelmoinnin luokan ja perinnän käsitteet voidaan havainnoida ja ymmärtää Aristoteleen töiden kautta. Vastaavasti prototyyppipohjaista olio-ohjelmointimallia voidaan parhaiten verrata prototyyppiteoriaan, vaikka yhtäläisyyksiä löytyy myös klassisen nominalismin kanssa. Jo tutkielmassa esitetty pelkistetty yleiskuva aihepiiristä on riittävä selittämään oliomainen ajattelumalli ohjelmoinnin kontekstissa. Lisäksi, vaikka oliokielten välillä on rakenteellisia eroavaisuuksia, kielten toteutus antaa hyvin myöten ja mahdollistaa ohjelmoinnin monella eri tapaa. Oliomaisen ajattelumallin soveltaminen näkyy ennen kaikkea ohjelmiston suunnitteluvaiheessa ja riippuu ennemmin kehittäjästä kuin käytettävästä ohjelmointikielestä.

Tutkielmassa pyrittiin pelkistämään luvussa 2 esitellyt teoreettiset näkökulmat tutkielman kontekstiin sopivaksi sekä tuomaan esille vain tutkielmalle relevantit osat. Riskinä on, että tämän vuoksi tutkielma ei anna kokonaisvaltaista kuvaa käsitellyistä näkökulmista. Lisäksi tutkielma käsittelee vain kahta olio-ohjelmoinnin rakennetta – luokittelua ja perintää. Tutkielmasta on jätetty käsittelemättä muiden olio-ohjelmoinnin keskeisten rakenteiden, kuten esimerkiksi kapseloinnin ja komposition, merkitys tutkimuskohteelle.

Tämä tutkielma osoittaa, että olio-ohjelmoinnin peruskäsitteet voidaan esittää filosofisen näkökulman kautta. Lisätutkimusta tarvitaan tunnistamaan, kuinka suuri osa olioparadigmaa käyttävistä ohjelmoijista tiedostaa tämän yhteyden ja millä tavalla tämä tietoisuus vaikuttaa sekä tuotantoprosessin sujuvuuteen että tuotetun ohjelmakoodin laatuun. Paljon tutkimusta on tehty olio-ohjelmoinnin oppimisesta, ja aloittelijoiden sekä opiskelijoiden keskuudessa olio-ohjelmoinnin yhdistäminen reaalimaailman ilmiöihin tuottaa erityisesti vaikeuksia (Eckerdal ja Thuné 2005; Xinogalos 2015). Lisätutkimuksen avulla voidaan selvittää, voidaanko olio-ohjelmoinnin koulutuksessa hyödyntää laajemmin filosofista näkökulmaa. Laajempaa tutkimusta poikkitieteelliseen kirjallisuuteen tarvitaan tunnistamaan tutkielmassa esitettyjen teorioiden lisäksi muita näkökulmia, joista voidaan löytää ratkaisuja ja inspiraatiota uusien oliokielten, sekä yleisemmin myös muiden tietotekniikan tutkimuskohteiden kehityksessä.

Lähteet

Aristoteles. 350 eaa./2000(a). *Categories.* Kääntänyt Ella Mary Edghill. (Alkuperäisteos julkaistu n. 350 eaa.) Infomotions, Inc.

———. 350 eaa./2000(b). *On Interpretation.* Kääntänyt Ella Mary Edghill. (Alkuperäisteos julkaistu n. 350 eaa.) Infomotions, Inc.

———. 350 eaa./2000(c). *Prior Analytics.* Kääntänyt A. J. Jenkinson. (Alkuperäisteos julkaistu n. 350 eaa.) Infomotions, Inc.

Armstrong, Deborah J. 2006. "The Quarks of Object-Oriented Development". *Commun. ACM* (New York, NY, USA) 49, numero 2 (helmikuu): 123–128. ISSN: 0001-0782. https://doi.org/10.1145/1113034.1113040.

Booch, Grady, Robert Maksimchuk, Michael Engle, Bobbi Young, Jim Conallen ja Kelli Houston. 2007. *Object-Oriented Analysis and Design with Applications.* Third. Addison-Wesley Professional. ISBN: 9780201895513.

Brachman, Ronald J. 1983. "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks". *Computer* 16 (10): 30–36. https://doi.org/10.1109/MC.1983.1654194.

Budd, Timothy A. 2001. *An Introduction to Object-Oriented Programming.* 3rd. USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201760312.

Cardelli, Luca, ja Peter Wegner. 1985. "On Understanding Types, Data Abstraction, and Polymorphism". *ACM Comput. Surv.* (New York, NY, USA) 17, numero 4 (joulukuu): 471–523. ISSN: 0360-0300. https://doi.org/10.1145/6041.6042.

Dony, Christophe, Jacques Malenfant ja Pierre Cointe. 1992. "Prototype-Based Languages: From a New Taxonomy to Constructive Proposals and Their Validation". *SIGPLAN Not.* (New York, NY, USA) 27, numero 10 (lokakuu): 201–217. ISSN: 0362-1340. https://doi.org/10.1145/141937.141954.

Eckerdal, Anna, ja Michael Thuné. 2005. “Novice Java Programmers’ Conceptions of "Object"and "Class", and Variation Theory”. *SIGCSE Bull.* (New York, NY, USA) 37, numero 3 (kesäkuu): 89–93. ISSN: 0097-8418. https://doi.org/10.1145/1151954.1067473.

Falguera, José L., Concha Martínez-Vidal ja Gideon Rosen. 2021. “Abstract Objects”. Teoksessa *The Stanford Encyclopedia of Philosophy,* toimittanut Edward N. Zalta. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/win2021/entries/abstract-objects.

Guimarães, José de Oliveira. 1995. “The Object Oriented Model and Its Advantages”. *SIGPLAN OOPS Mess.* (New York, NY, USA) 6, numero 1 (tammikuu): 40–49. ISSN: 1055-6400. https://doi.org/10.1145/209866.209872.

Hürsch, Walter L. 1994. “Should Superclasses Be Abstract?” (Berlin, Heidelberg), ECOOP ’94, 12–31. https://doi.org/10.1007/BFb0052174.

Kay, Alan C. 1993. “The Early History of Smalltalk”. *SIGPLAN Not.* (New York, NY, USA) 28, numero 3 (maaliskuu): 69–95. ISSN: 0362-1340. https://doi.org/10.1145/155360.155364.

King, Peter, ja Andrew Arlig. 2018. “Peter Abelard”. Teoksessa *The Stanford Encyclopedia of Philosophy,* toimittanut Edward N. Zalta. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/entries/abelard/.

Klima, Gyula. 2022. “The Medieval Problem of Universals”. Teoksessa *The Stanford Encyclopedia of Philosophy,* toimittanut Edward N. Zalta. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/entries/universals-medieval/.

Kraut, Richard. 2022. “Plato”. Teoksessa *The Stanford Encyclopedia of Philosophy,* toimittanut Edward N. Zalta. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/entries/plato/.

Lakoff, George. 1990. *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind.* The University of Chicago Press. ISBN: 9780226468044.

Lieberman, Henry. 1986. “Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems”. *SIGPLAN Not.* (New York, NY, USA) 21, numero 11 (kesäkuu): 214–223. ISSN: 0362-1340. https://doi.org/10.1145/960112.28718.

Liskov, Barbara. 1987. “Keynote Address - Data Abstraction and Hierarchy”. *SIGPLAN Not.* (New York, NY, USA) 23, numero 5 (tammikuu): 17–34. ISSN: 0362-1340. https://doi.org/10.1145/62139.62141.

Mill, John Stuart. 1843. *A System of Logic: Ratiocinative and Inductive.* London : J. W. Parker.

Milner, Robin. 1978. “A theory of type polymorphism in programming”. *Journal of Computer and System Sciences* 17 (3): 348–375. ISSN: 0022-0000. https://doi.org/https://doi.org/10.1016/0022-0000(78)90014-4.

Osherson, Daniel N., ja Edward E. Smith. 1981. “On the adequacy of prototype theory as a theory of concepts”. *Cognition* 9 (1): 35–58. ISSN: 0010-0277. https://doi.org/https://doi.org/10.1016/0010-0277(81)90013-5.

Rayside, D., ja Kostas Kontogiannis. 2001. “On the syllogistic structure of object-oriented programming”, 113–122. Kesäkuu. ISBN: 0-7695-1050-7. https://doi.org/10.1109/ICSE.2001.919086.

Rayside, Derek, ja Gerard T. Campbell. 2000. “An Aristotelian Understanding of Object-Oriented Programming”. *SIGPLAN Not.* (New York, NY, USA) 35, numero 10 (lokakuu): 337–353. ISSN: 0362-1340. https://doi.org/10.1145/354222.353194.

Rentsch, Tim. 1982. “Object Oriented Programming”. *SIGPLAN Not.* (New York, NY, USA) 17, numero 9 (syyskuu): 51–57. ISSN: 0362-1340. https://doi.org/10.1145/947955.947961.

Rickless, Samuel. 2020. “Plato’s Parmenides”. Teoksessa *The Stanford Encyclopedia of Philosophy,* toimittanut Edward N. Zalta. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/spr2020/entries/plato-parmenides/.

Robinson, Howard. 2021. “Substance”. Teoksessa *The Stanford Encyclopedia of Philosophy,* toimittanut Edward N. Zalta. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/fall2021/entries/substance/.

Robson, David. 1981. “Object-Oriented Software Systems”. *Byte Publications Inc.* 6 (8): 74–86.

Rodriguez-Pereyra, Gonzalo. 2019. “Nominalism in Metaphysics”. Teoksessa *The Stanford Encyclopedia of Philosophy,* toimittanut Edward N. Zalta. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/entries/nominalism-metaphysics/.

Rosch, Eleanor H. 1973. “Natural categories”. *Cognitive Psychology* 4 (3): 328–350. ISSN: 0010-0285. https://doi.org/10.1016/0010-0285(73)90017-0.

Sheetz, Steven D., Gretchen Irwin, David P. Tegarden, H. James Nelson ja David E. Monarchi. 1997. “Exploring the Difficulties of Learning Object-Oriented Techniques”. *Journal of Management Information Systems* 14 (2): 103–131. https://doi.org/10.1080/07421222.1997.11518167.

Shields, Christopher. 2022. “Aristotle”. Teoksessa *The Stanford Encyclopedia of Philosophy,* toimittanut Edward N. Zalta. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/spr2022/entries/aristotle/.

Smith, Robin. 2020. “Aristotle’s Logic”. Teoksessa *The Stanford Encyclopedia of Philosophy,* toimittanut Edward N. Zalta. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/fall2020/entries/aristotle-logic/.

Spade, Paul Vincent. 1994. *Five Texts On The Mediaeval Problem Of Universals: Porphyry, Boethius, Abelard, Duns Scotus, Ockham.* Hackett Publishing Company, Inc. ISBN: 9780872202504.

Studtmann, Paul. 2021. “Aristotle’s Categories”. Teoksessa *The Stanford Encyclopedia of Philosophy,* toimittanut Edward N. Zalta. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/entries/aristotle-categories/.

Taivalsaari, Antero. 1996a. “Classes vs. Prototypes - Some Philosophical and Historical Observations”. Teoksessa *Journal of Object-Oriented Programming,* 44–50. SpringerVerlag. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.56.4713.

———. 1996b. “On the Notion of Inheritance”. *ACM Comput. Surv.* (New York, NY, USA) 28, numero 3 (syyskuu): 438–479. ISSN: 0360-0300. https://doi.org/10.1145/243439.243441.

Thomasson, Benjy, Mark Ratcliffe ja Lynda Thomas. 2006. “Identifying Novice Difficulties in Object Oriented Design”. *SIGCSE Bull.* (New York, NY, USA) 38, numero 3 (kesäkuu): 28–32. ISSN: 0097-8418. https://doi.org/10.1145/1140123.1140135.

Tylman, Wojciech. 2018. “Computer Science and Philosophy: Did Plato Foresee Object-Oriented Programming?” *Foundations of Science* 23:159–172. https://doi.org/10.1007/s10699-016-9506-7.

van Inwagen, Peter, ja Meghan Sullivan. 2021. “Metaphysics”. Teoksessa *The Stanford Encyclopedia of Philosophy,* toimittanut Edward N. Zalta. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/win2021/entries/metaphysics/.

Wegner, Peter. 1987. “Dimensions of Object-Based Language Design”. *SIGPLAN Not.* 22:168–182. https://doi.org/10.1145/38807.38823.

Wittgenstein, Ludwig. 2009. *Philosophical Investigations.* Wiley-Blackwell. ISBN: 9781405159289.

Xinogalos, Stelios. 2015. “Object-Oriented Design and Programming: An Investigation of Novices’ Conceptions on Objects and Classes”. (New York, NY, USA) 15, numero 3 (heinäkuu). https://doi.org/10.1145/2700519.