Ville Salonen

Javasta C#:iin

Tietotekniikan
kandidaatintutkielma
21. maaliskuuta 2011

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Tekijä: Ville Salonen
Yhteystiedot: ville.salonen@iki.fi
Työn nimi: Javasta C#:iin
Title in English: From Java To C#
Työ: Tietotekniikan kandidaatintutkielma
Sivumäärä: 25
Tiivistelmä: Tutkimuksen tavoitteena on selvittää, miten ohjelmoijan Java-osaamista voidaan hyödyntää C#-kielen opiskelussa tai toisinpäin. Helppo siirtyminen antaisi muun muassa yrityksille enemmän joustavuutta rekrytointiin, koska palkattavan työntekijän tuntemus tietystä käytettävästä kielestä ei olisi niin tärkeää kuin käytettävän kielen nopea oppiminen valmiita taitoja hyödyntäen. Tutkimus tehtiin vertailemalla kielistä julkaistuja ohjelmointioppaita, rajanpintakuvauksia ja kielten teknisiä spesifikaatioita. Tutkimuksen tulosten perusteella kielet ovat monessa suhteessa samankaltaisia, mutta erojakin on. Erot eivät ole haastavia ja Javasta C#:iin siirryttäessä ne tehostavat ohjelmointityötä. Tulosten perusteella siirtymät Javasta C#:iin tai toisinpäin ovat helppoja ja nopeita.
English abstract: The objective of the thesis was to find out if you can use existing Java experience to ease the learning of C# or vice versa. Easy moving from language to another would benefit for example companies' hiring decisions as the candidate's knowledge of a specific language would not be as important as an ability to learn the language quickly by using existing experience of another language. The study was done by comparing published programming tutorials, application programming interfaces and technical specifications. Based on the results of the study, languages are similar in many ways but there are some differences. These differences are not challenging and when moving from Java to C# they increase the productivity of the programming. The results suggest that learning C# by using existing Java experience or vice versa is easy and fast.
Avainsanat: Java, C#, ohjelmointikielet, oppiminen.
Keywords: Java, C#, programming languages, learning.

Sisältö

Termilista

Konekoodi (engl. *machine code*) Ohjelmakoodia, jonka koneen prosessori osaa suorittaa.

Tavukoodi (engl. *bytecode*) Ohjelmakoodia, jota ei voi suoraan suorittaa koneen prosessorilla, vaan jota suoritetaan virtuaalikoneella.

Virtuaalikone (engl. *virtual machine*) Alusta, jonka päällä tavukoodia suoritetaan. Virtuaalikone tulkkaa tavukoodin konekoodiksi.

Common Language Runtime Microsoftin .NET-kielten yhteinen virtuaalikone. Esimerkiksi C#- tai Visual Basic -koodista käännetään samanlaista tavukoodia, joka suoritetaan Common Language Runtimessa.

Java Virtual Machine JVM-pohjaisten kielten yhteinen virtuaalikone. Javan lisäksi esimerkiksi Scala [8] ja Groovy [5] käyttävät JVM:ää tavukoodin suoritukseen.

1 Johdanto

Olio-ohjelmointikieliä on lukuisia erilaisia. Viime aikoina dynaamisesti tyypitetyt ohjelmointikielet kuten Python ja Ruby ovat saaneet huomiota, mutta staattiset tyypitetytkään kielet eivät ole katoamassa. Microsoftin kehittämä C# ja Sunin kehittämä Java ovat kaksi suosittua staattisesti tyypitettyä olio-ohjelmointikieltä.

Jyväskylän yliopistossa uusille opiskelijoille suunnatut Ohjelmointi 1- ja Ohjelmointi 2 -kurssit ovat viime vuosina järjestetty käyttäen Java-kieltä. Hyvä ohjelmoija ei kuitenkaan ole vain yhden kielen osaaja.

Tutkielmani tarkoitus on selvittää, miten Java-kieltä osaavat opiskelijat pystyisivät opettelemaan mahdollisimman helposti C#-kieltä. Kartoitan ensin kielten historiaa ja käyttötarkoituksia, jonka jälkeen tutkin, mitä yhteistä ja erilaista kielillä on. Teen tutkimukseni vertailemalla kielten opetusmateriaaleja ja ohjelmakoodiesimerkkejä.

2 Java ja C#

Sun Microsystemsin (myöhemmin Sun) työntekijä, James Gosling, aloitti Java-kielen kehityksen vuonna 1991. Kieli oli alunperin suunnattu käytettäväksi sittemmin unohdetussa kodin elektroniikkalaitteiden ohjaukseen tarkoitetussa *7-laitteessa (StarSeven). Jo alusta asti kielen tavoitteena oli olla prosessoririippumaton, jolloin samalla ohjelmointikielellä pystyttäisiin toteuttamaan sovelluksia useille eri alustoille. *7-laitteet suunniteltiin käytettäväksi digitaalikaapelitelevisioon liittyvissä interaktiivisissa palveluissa, joissa käyttäjät voisivat tuottaa sisältöä. Kaapelitelevisioyhtiöt vastustivat ajatusta, koska he halusivat pitää sisällön kontrolloinnin itsellään. Goslingin tiimissä oli jo 70 henkilöä, mutta selkeää käyttötarkoitusta kielelle ei ollut kaapelitelevisioyhtiöiden kieltäytymisen takia. John Gage, James Gosling, Patrick Naughton, Wayne Rosing ja Eric Schmidt kokoontuivat miettimään uutta käyttötarkoitusta ja he päätyivät tutkimaan mahdollisuuksia Javan käyttöön Internetissä. [2]

1990-luvun loppupuolelle asti Windows-ohjelmistot kehitettiin edelleen lähes yksinomaan C- ja C++-kielillä, joiden muun muassa vaikeammasta muistinkäsittelystä johtuva monimutkaisuus piti ohjelmistonkehityksen monille vaikeana ja hitaana prosessina. Lisäksi Microsoftin kirjastojen käyttäminen sitoi kehitetyt ohjelmat Windows-ympäristöön.

Jos ohjelmistoja pystyisi käyttämään millä tahansa käyttöjärjestelmällä, ei Windowsia voisi enää markkinoida ylivoimaisella ohjelmatarjonnalla ja Microsoft me-

nettäisi valta-asemansa. Estääkseen Javan kehittymisen merkittäväksi kilpailijaksi Microsoft teki läheisimmän yhteistyökumppaninsa, Intelin, kanssa yhteisen päätöksen kehittää kilpailija Javalle. Javan prosessoririippumattomuus oli uhka myös Intelille sen ollessa suurin prosessorivalmistaja. Microsoftin sisällä laadittiin toimintastrategia, jossa Javan kehityksestä pyrittiin tekemään vaikeampaa niin, ettei koodin kirjoittaminen yhdelle alustalle ja sen ajaminen kaikilla muilla enää onnistunutkaan. Tämä onnistui toteuttamalla Windowsille oma Java-virtuaalikone, joka oli Sunin virallista tehokkaampi, mutta käytti epästandardeja J++-nimellä kulkeneita laajennuksia. Näennäisesti Microsoft jatkoi edelleen Javan tukemista omalla alustallaan, mutta pyrki saamaan oman Windowsiin sidotun ratkaisunsa kehityksessä Javan ohi. Sun ei pitänyt Microsoftin aiheuttamasta uhasta tuotettaan kohtaan ja vuonna 1998 Sun haastoi Microsoftin oikeuteen. [3] Syytteen perusteluiksi Sun esitti Microsoftin rikkoneen lisenssiehtoja, joista Sun ja Microsoft oli sopineet vuonna 1995. [25]

Oikeustaistelu varjosti Microsoftin Java-toteutuksen tulevaisuutta ja Microsoftin suunnittelema uusi pohja omaa korvaavaa kieltä varten lopetti J++:n jatkokehityksen. Vuonna 1999 Microsoft perusti Anders Hejlsbergin johdolla kehitysryhmän C#-ohjelmointikieltä varten. Kieli otti vaikutteita muun muassa Pascalista ja Javasta. Erityisen paljon vaikutteita on antanut Delphi, koska Hejlsberg työskenteli edellisessä työpaikassaan Borlandilla yhtenä Delphin pääkehittäjistä. [6]

Oikeustaistelu sovittiin oikeussalien ulkopuolella vuonna 2004 ja Microsoft maksoi 1,95 miljardia dollaria korvauksia ja lisenssimaksuja Sunille. [4]

2.1 Käyttöalustat

Kuten edellisessä luvussa selvisi, Java on alusta asti suunnattu käytettäväksi useilla eri alustoilla. Näin ohjelmiston toteuttava organisaatio ei joudu sitoutumaan vain yhteen laitealustaan, jonka tulevaisuuteen organisaatio ei voi suoraan vaikuttaa. Kaikille alustoille Java-virtuaalikonetta ei ole kehitetty, mutta muun muassa Windowsin, Mac OS X:n, Linuxin ja BSD-varianttien tukeminen kattaa suuren osan maailman tietokoneista. Microsoftin virallinen tuki C#:lle kattaa Windows-työpöydän, Xbox 360 -pelikonsolin ja Windows Mobile -laitteet.

C# ja siihen olennaisesti liittyvät kirjastot eivät kuitenkaan ole sidottu pelkästään Windowsiin. Mono on Novellin johtama projekti, jonka tavoitteena on kehittää useilla eri käyttöjärjestelmillä toimivat avoimen lähdekoodin versiot näistä työkaluista. Kirjoitushetkellä Monosta on toteutettu viralliset versiot Windowsille, Linuxille, Mac OS X:lle ja Solarikselle. Koska projekti ei ole Microsoftin toteuttama,

kaikki Microsoftin kehittämät ominaisuudet eivät ole käytettävissä, mutta kattava osa näistä on kuitenkin jo toimivia. Myös Microsoftin Silverlight-web-teknologiasta on muilla alustoilla toimiva Mono-projektin alainen Moonlight-käännös. [24]

2.2 Kehitysympäristöt

Javalle ei ole yhtä virallista kehitysympäristöä, mutta hyvin suosituksi valinnaksi on osoittautunut alunperin IBM:n VisualAge-ohjelmiston pohjalta kehitetty Eclipse Foundationin Eclipse. Eclipse itsessään on kehitetty Javalla ja sillä on mahdollista kääntää sovelluksia jokaiselle Java-virtuaalikonetta tukevalle alustalle. Toinen suosittu kehitysympäristö on Sunin NetBeans. Ominaisuuksiltaan ja käytettävyydeltään nämä kaksi ovat hyvin lähellä toisiaan.

Visual Studio on Microsoftin tarjoama kehitysympäristö, jolla pystytään kehittämään muun muassa C#-ohjelmia. Siitä on olemassa kaupallinen ominaisuuksiltaan monipuolisempi versio, mutta myös harrastelijoille ja opiskelijoille suunnattu kevyempi, ilmainen Visual Studio Express. Visual Studio toimii vain Windows-ympäristössä, mutta sillä voidaan kehittää muillakin alustoilla toimivia sovelluksia käyttämällä Mono-kääntäjää.

3 Yhtäläisyydet

Java ja C# ovat molemmat staattisesti tyypitettyjä olio-ohjelmointikieliä. Ohjelma koostuu yhdestä tai useammasta luokasta. Luokat sisältävät olio- ja arvotyyppejä. Kun ohjelmakoodi on valmis, se käännetään kääntäjäohjelmalla tavukoodiksi. Tavukoodia ei pysty suoraan suorittamaan koneella, vaan se suoritetaan virtuaalikoneessa, joka tulkkaa tavukoodin käyttöjärjestelmälle sopivaksi binäärikoodiksi.

3.1 Oliotyypit

Oliot ovat tietorakenteita, joilla on tietokenttiä ja metodeja eli olion liittyviä funktioita. Oliotyyppiset muuttujat ovat viitteitä olion ensimmäiseen muistipaikkaan. Olion tietorakenteeseen tallennetaan olion tyyppi, arvotyyppiset muuttujat ja viitteet oliotyyppisiin muuttujiin. Lisäksi olio tarvitsee metodiensa sijainnin muistissa.

3.2 Arvotyypit

Javan ja C#:n arvotyyppiset muuttujat ovat keskenään hyvin samankaltaisia. Arvotyyppeihin säilötään totuusarvoja, yksittäisiä merkkejä ja kokonais- ja desimaalilu-

kuja. Arvotyypit on esitelty taulukossa 1 [11].

Java	**C#**	**Kuvaus**
`boolean`	`bool`	Kaksiarvoinen totuusarvomuuttuja
`char`	`char`	Unicode-merkki
`byte`	`sbyte`	8-bittinen kokonaisluku väliltä [-128, 127]
-	`byte`	8-bittinen positiivinen kokonaisluku väliltä [0, 255]
`short`	`short`	16-bittinen kokonaisluku väliltä [-32 768, 32 767]
-	`ushort`	16-bittinen positiivinen kokonaisluku väliltä [0, 65 535]
`int`	`int`	32-bittinen kokonaisluku väliltä [-2 147 483 648, 2 147 483 647]
-	`uint`	32-bittinen positiivinen kokonaisluku väliltä [0, 4 294 967 295]
`long`	`long`	64-bittinen kokonaisluku väliltä [-922 337 203 685 477 508, 922 337 203 685 477 507]
-	`ulong`	64-bittinen positiivinen kokonaisluku väliltä [0, 1 844 674 407 370 955 015]
`float`	`float`	32-bittinen desimaaliluku väliltä [-3,402823e38, 3,402823e38]
`double`	`double`	64-bittinen desimaaliluku väliltä [-1,79769313486232e308, 1,79769313486232e308]

Taulukko 1: Javan ja C#:n arvotyypit.

Primiitivien nimet ovat kielissä lähes identtiset. Eroina totuusarvotyyppi on Javassa nimeltään `boolean` ja C#:ssa `bool` sekä 8-bittinen kokonaisluku on Javassa `byte` ja C#:ssa `sbyte`.

Nimeämiserojen lisäksi C#:ssa on omat tyypit etumerkittömille (engl. *unsigned*) kokonaisluvuille. Javassa kaikki desimaali- ja kokonaisluvut voivat olla positiivisia tai negatiivisia. Jos haluamme käsitellä pelkkiä positiivisia lukuja, joudumme kirjoittamaan ohjelmakoodiin logiikan, joka tarkistaa lukujen positiivisuuden.

3.3 Ohjelmointisyntaksi

Hello World -esimerkki on todennäköisesti monille opiskelijoille tuttu. Java-kielellä se toteutetaan seuraavasti:

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.writeln("Hello World!");
  }
}
```

Luodaan `HelloWorld`-luokka, johon määritellään julkinen staattinen main-metodi. `main`-metodi ottaa parametrikseen merkkijonotaulukon, joka sisältää mahdolliset komentoriviltä annetut argumentit. Main-metodissa kutsumme Javan järjestelmäkirjastoista `System.out.writeln`-metodia, joka tulostaa rivin `Hello World`.

```
public class HelloWorld {
  public static void Main(string[] args) {
    System.Console.WriteLine("Hello World!");
  }
}
```

Java ja C# lainaavat suuren osan syntaksistaan C-kieleltä, joten Javasta C#:iin siirtyessä ei tarvitse opetella täysin uutta syntaksia. Koodiesimerkeistä voidaan huomata, miten ohjelmat ovat lähes identtisiä. Suurin ero on, että C#-ohjelmissa luokat määritellään nimiavaruuden alle ja Javassa ne määritellään pakettiin (engl. *package*). Nimiavaruudet ja paketit ovat käsitteinä samoja, mutta niiden toteutukset poikkeavat [14].

3.4 Virtuaalikoneet ja tavukoodi

Nykyajan tehokkaat tietokoneet ja korkean tason ohjelmointikielet piilottavat suuren osan siitä, mitä koneen sisällä tapahtuu ohjelmaa ajettaessa. Kun tietokoneet olivat huomattavasti nykyistä tehottomampia, ei ollut varaa tuhlata suorituksen kellojaksoja. Tällöin ohjelmat kirjoitettiin Assembly-kielellä, joka on hyvin matalan tason ohjelmointikieli. Se on tehokas, koska ohjelmoija kirjoittaa koodiin jokaisen operaation, joka ohjelman ajamisen aikana suoritetaan. Koska ohjelmointi tehdään laitetasolla, kirjoittaminen on hidasta ja virhealtista ja toiselle käyttöjärjestelmä- tai laitealustalle siirtyminen saattaa vaatia huomattavia muutoksia koodiin.

Koneiden nopeutuessa yksittäisten kellojaksojen arvo ei ollut enää yhtä suuri kuin aiemmin, joten ohjelmat voitiin kirjoittaa korkeamman tason kielillä, jotka piilottavat kehittäjältä osan ohjelman monimutkaisuudesta. Korkeamman tason kielellä kirjoitetut ohjelmat muutetaan kääntäjällä Assembly-koodiksi, jota tietokone osaa suorittaa. Nyt toiselle käyttöjärjestelmä- tai laitealustalle siirtyminen on helpompaa, koska ohjelmakoodi voidaan parhaassa tilanteessa kirjoittaa vain kerran. Kun ote-

taan käyttöön uusi alusta, kirjoitetaan sille sopiva kääntäjä. Kääntäjä kääntää aiemmin kirjoitetun ohjelmakoodin uuden alustan ymmärtämäksi konekoodiksi. Usein siirtyminen ei kuitenkaan aivan näin yksinkertaista ole, vaan esimerkiksi Windowsista Linuxiin siirryttäessä joudutaan muokkaamaan tiedostonkäsittelyyn liittyviä osia, koska Windows ja Linux käsittelevät tiedostojärjestelmää eri tavoin.

Kun ohjelmakoodi muutetaan kääntäjällä tietylle alustalle sopivaksi ohjelmaksi, sen kopioiminen toiselle alustalle ei onnistu. Jotta siirtyminen alustalta toiselle onnistuisi ilman uutta käännöstä, muun muassa C#:ia ja Javaa ei käännetä suoraan konekoodiksi, vaan tavukoodiksi. Tavukoodia ei suoraan voi suorittaa koneella kuten konekoodia, vaan väliin tarvitaan virtuaalikone, joka kääntää tavukoodin pyytämät operaatiot konekoodiksi. Kun ohjelmaa halutaan käyttää uudella alustalla, voidaan vanhaa ohjelmatiedostoa suorittaa uudelle alustalle kehitetyllä virtuaalikoneella.

4 Erot

Javasta on mahdollista siirtyä C#:iin kirjoittamalla lähes täysin samanlaista koodia kuin Javalla kirjoittaisi, mutta tällöin kielestä toiseen siirtyminen ei tuo merkittäviä etuja, koska C#:stä löytyy useita tekniikoita, jotka mahdollistavat lyhyempiä ja tehokkaampia tapoja kirjoittaa sama ohjelmalogiikka.

Javan versiota 6 ja C#:n versiota 3 vertaillessani ei löytynyt ainoatakaan eroa, jossa Javan toteutus olisi teknisesti tai ohjelmointityön helppouden kannalta ylivoimainen C#:n toteutukseen verrattuna. Tästä ei kuitenkaan voi vetää johtopäätöstä, ettei Javalla ole sijaa ohjelmistokehityksessä ja että C# olisi jokaiseen tilanteeseen parempi ratkaisu. Suurin osa eroista ovat pieniä ja suhteellisen helposti kierrettäviä suoritustehon tai käyttäjäystävällisyyden kustannuksella. Lisäksi ohjelmointikielen valintaan vaikuttavat lukuisat tapauskohtaiset yksityiskohdat, joten yleispätevän suosituksen antaminen ei ole järkevää.

4.1 Luokat ja tietueet

Javassa primitiivimuuttujien lisäksi on olemassa vain erilaisia luokkia. C#:ssa on tietueita ja luokkia. C#:in primitiivimuuttujat on toteutettu tietueilla [18]. Tietue on paremmin C-ohjelmointikielestä tuttu tietorakenne. C:ssä näihin pystyi tallentamaan vain primitiivimuuttujia: funktioita ei pystynyt kirjoittamaan suoraan tietueen sisälle. C#:ssa tietueisiin sopivat kuitenkin funktiot, konstruktorit, operaattorifunktiot ja tapahtumat. Tietueiden periminen ei ole mahdollista, mutta ne voivat kuitenkin toteuttaa rajapintoja samoin kuin luokatkin. [22]

Luokista poiketen tietueiden muisti varataan pinosta, eikä dynaamisen muistinhallinnan keosta. Lisäksi uuden tietueen luominen on uuden olion alustamiseen verrattuna kevyt operaatio.

4.2 Muuttujatyyppien yhdenmukaisuus

Javassa kaikki muuttujatyypit pystyttiin jaottelemaan primitiiveihin ja olioihin. Primitiivejä ovat kaikki, joiden muuttujatyyppi alkaa pienellä alkukirjaimella eli `short`, `int`, `long`, `float`, `double`, `boolean`, `char` ja `byte`. Primitiivimuuttujan muistiosoitteessa on suoraan muuttujan arvo. Kaikki muut muuttujatyypit ovat olioita, joiden muistiosoitteessa on viite olion sijaintiin.

Microsoftin mukaan C#:ssa muuttujia ei jaotella primitiiveihin ja olioihin, vaan kaikki periytyvät object-luokasta: näin esimerkiksi `int`-muuttujilta löytyy `ToString`-metodi, ja ne pystytään antamaan argumentteina metodille, joka hyväksyy argumentikseen `object`-tyyppisen olion. Tätä kielen ominaisuutta kutsutaan muuttujatyyppien yhdenmukaisuudeksi (engl. *unified type system*). [23]

Jos Javassa haluttaisiin toteuttaa esimerkiksi pino-olio, johon voidaan asettaa mitä tahansa muuttujia, primitiivimuuttujat tulisi muuttaa vastaaviksi oliomuuttujiksi:

```
public class Stack {
  public void push(Object input) { /* ... */ }
  public object pop() { /* ... */ }

  public void main(String[] args) {
    Stack stack = new Stack();
    stack.push(new Double(3.14));
    stack.push(new Integer(17));
  }
}
```

C#:ssa sama tapahtuisi seuraavasti [23]:

```
public class Stack
{
  public object Pop() { /* ... */ }
  public void Push(object o) { /* ... */ }

  public void main() {
    Stack stack = new Stack();
    stack.Push(3.14);
    stack.Push(17);
  }
}
```

Olioita luodessa on varattava dynaamisesti muistia, suoritettava mahdolliset konstruktorimetodit ja luotava viite luotuun olioon. Arvomuuttujia käyttäessä tarvitsee vain tallentaa arvo staattisesti varattuun muistipaikkaan. Jos kaikki arvomuuttujat olisivat todellisuudessa olioita, suorituskyky kärsisi merkittävästi.

Esimerkiksi tieteellinen laskenta C#:llä olisi hyvin hidasta, koska yhden silmukkakierroksen aikana saatetaan luoda tuhansia muuttujia. Primitiivimuuttujien tapaan tehtävä suora tallennus muuttujan muistiosoitteeseen olisi nopea operaatio, mutta uuden olion luominen on sen sijaan merkittävästi hitaampi. Suoritustehon kasvattamiseksi arvomuuttujien käsittely suoritetaan suoraan muistiosoitteita käyttämällä. Vasta asetettaessa arvomuuttujaa olioon sen arvo kopioidaan olion sisään ja asetetaan muuttuja viittaamaan luotuun olioon. Vastaavasti asetettaessa olioa takaisin arvomuuttujaan arvo kopioidaan olion sisältä ja asetetaan suoraan muistiosoitteeseen. Näitä operaatioita kutsutaan boxing- ja unboxing-operaatioiksi. [23]

4.3 Ominaisuudet

Lähes kaikilla olioilla on attribuutteja, joiden arvoja tulee pystyä lukemaan olion ulkopuolelta. Yksi tapa toteuttaa tämä on asettaa attribuutit täysin tai osittain julkisiksi `public`- tai `protected`-avainsanoilla.

Attribuuttien asettaminen julkiseksi on kuitenkin huono tapa, ellei muuttujia ole asetettu vakioksi Javan `final`- tai C#:in `const`-avainsanalla, koska muulloin niiden arvoja pystytään myös muuttamaan olion ulkopuolelta. Tällöin syötettäviä uusia arvoja ei pystytä tarkistamaan ennen asettamista, eikä kohdeolio saa itse tietoa arvojensa muuttumisesta. Tämä rikkoo myös enkapsulaatioperiaatetta, jonka perusteella ohjelman pitäisi koostua komponenteista, jotka tarjoavat muille komponenteille käytettävän rajapinnan. Jos muut komponentit käyttävät suoraan toisen komponentin sisäisiä arvoja, on niiden toteutus riippuvainen tietystä komponentin

toteutuksesta ja kyseisen komponentin vaihtaminen vaihtoehtoiseen toteutukseen olisi tarpeettoman työlästä.

Yleiseksi tavaksi on muodostunut asettaa attribuutti salaiseksi `private`-avainsanalla ja kirjoittaa olioon julkiset `getter`- ja `setter`-metodit, joilla arvo pystytään lukemaan ja haluttaessa myös muuttamaan. Tämä johtaa niin sanottuun boilerplate-koodiin, jossa vain pieni osa koodista liittyy varsinaiseen toteutukseen [15].

```
public class User {
  private String username;

  public getUsername() {
    return this.username;
  }

  public setUsername(String value) {
    this.username = value;
  }
}
```

Valitettavasti Javassa tähän ei ole parempaa keinoa ja ainoa tapa vähentää boilerplate-koodin kirjoittamista on käyttää esimerkiksi kehitysympäristön makroja. C#:ssa sen sijaan on käytettävissä ominaisuudet (engl. *properties*):

```
public class User {
  public string Username { get; set; }
}
```

Tällä tavoin luokkaan määritellään yhdessä lohkossa julkinen ominaisuus, jonka sisällä `get`- ja `set`-avainsanoilla kerrotaan, mitä ominaisuuden arvoa luettaessa ja kirjoittaessa tehdään. Lohkosta voidaan jättää jompikumpi operaatio pois, jolloin ominaisuudesta tulee vain luettava tai vain kirjoitettava.

Esimerkkikoodissa on hyvin yksinkertainen tapaus, jossa kaikki arvot hyväksytään ja arvon lukeminen on sallittua, eikä vaadi erikoismuotoiluja. Vaikeammissa tapauksissakin ominaisuuksista on hyötyä, koska tällöin attribuutin lukemiseen ja kirjoittamiseen liittyvä logiikka on koottuna yhdessä paikassa. Myöskään eri luokkien välillä ei ole eroja `getter`- ja `setter`-metodien nimissä: ei ole siis esimerkiksi `getName`- ja `writeName`-nimisiä metodeja, vaan arvo luetaan aina syntaksilla `olio.Ominaisuus` ja uusi arvo asetetaan syntaksilla `olio.Ominaisuus = uusiArvo`.

```
private string email;
public string Email {
  get { return this.email; }
  set {
    if ( Validator.validateEmail(value) ) {
      this.email = value;
    }
  }
}
```

Yllä olevassa esimerkissä ominaisuuden arvoa asettaessa sille tehdään tarkistus ja vasta sen läpäistyään argumenttina annettu arvo asetetaan ominaisuudelle. Yleisen tavan mukaisesti salainen attribuutti ja julkinen ominaisuus ovat saman nimisiä, mutta ominaisuuden nimi on kirjoitettu isolla alkukirjaimella.

4.4 Operaattoreiden ylikuormitus

Operaattoreiden ylikuormituksella (engl. *operator overloading*) kehittäjän on mahdollista määritellä, miten kahden olion väliset aritmeettiset operaatiot ja yhtäsuuruusvertailut toteutetaan. Javassa operaattoreiden ylikuormitus ei ole kehittäjän määriteltävissä, mutta Sun on toteuttanut erikoistapauksena merkkijonojen yhdistämisen +-operaattorilla.

Kuormittamisen puuttumisen takia esimerkiksi matriisien yhteen- ja kertolaskun toteuttaminen on Javassa tehtävä erityisillä, mielivaltaisesti nimetyillä metodeilla, jotka ottavat parametrikseen toisen matriisin. Jo yksinkertaisesta matemaattisesta lausekkeesta tulee monimutkainen.

```
public class Matrix {
  public Matrix add(Matrix matrix) { /* ... */ }
  public Matrix multiply(Matrix matrix) { /* ... */ }

  public void main(String[] args) {
    Matrix matrix1 = new Matrix(/* ... */);
    Matrix matrix2 = new Matrix(/* ... */);
    Matrix matrix3 = new Matrix(/* ... */);

    matrix1.add(matrix2).multiply(matrix3);
  }
}
```

C#:n kanssa operaattoreiden ylikuormitus onnistuu määrittelemällä luokkaan staattinen metodi `operatorX`, jossa `X` on ylikuormitettava operaattori. Lista mahdollisesti ylikuormitettavista operaattoreista löytyy Microsoft Developer Network -palvelusta [20].

```
public class Matrix {
  public static Matrix operator+(Matrix m1, Matrix m2) {
    /* ... */
  }

  public static Matrix operator*(Matrix m1, Matrix m2) {
    /* ... */
  }

  public void main() {
    Matrix matrix1 = new Matrix(/* ... */);
    Matrix matrix2 = new Matrix(/* ... */);
    Matrix matrix3 = new Matrix(/* ... */);

    matrix1 = (matrix1 + matrix2) * matrix3;
  }
}
```

Matriisilaskenta yllä olevalla koodilla ei ole tehokasta. Jokaista laskutoimitusta kohden luodaan uusi matriisiolio. Esimerkiksi `matrix1 + matrix2` luo uuden olion, joka kerrotaan `matrix3`-oliolla, jolloin syntyy jälleen uusi olio. Jos alkuperäisiä matriiseja ei ole tarvetta säilyttää, kannattaa esimerkiksi kertolaskun tulokset sijoittaa suoraan kerrottavaan matriisiin. Periaatteessa tämän voi toteuttaa operaattoreiden ylikuormituksellakin, jos kuormittavassa metodissa lasketaan lopputulos operandiin. Tämä tosin on epäloogista käyttäjän kannalta, koska hän ei pysty huomaamaan lausekkeesta `matrix1 * matrix2`, että `matrix2`:n arvo muuttuu. Koska operaattoreiden ylikuormituksella ei voi saavuttaa muuta kuin parempaa luettavuutta, olisi tällaisen ylikuormituksen tekeminen mieletöntä.

Sinänsä operaattoreiden ylikuormitus ei ole kuin kuorrutusta syntaksin päälle, mutta sillä on mahdollista yksinkertaistaa ja lyhentää monimutkaisia lausekkeita. Etenkin matemaattista ohjelmistoa kehittäessä laskukaavat näyttävät samankaltaisemmilta paperilla ja ohjelmakoodissa. Lisäksi vähempi koodimäärä ja loogisempi muotoilu edesauttaa virheettömyyttä.

Operaattoreita ylikuormittaessa on tärkeää muistaa tehdä vain loogisia ylikuormituksia: mitä toiminnallisuutta haluttaisiin esimerkiksi kuvata merkkijonojen

−-operaattoria kuormittaessa? Epäloogisilla ylikuormituksilla on helppo tehdä koodista vaikealukuista.

4.5 Parametrinvälitys

Javassa parametrinvälitys hoidetaan aina arvonvälitysmenetelmällä (engl. *pass by value*). Tämä tarkoittaa sitä, että kun metodia kutsuttaessa annetaan argumenttina jokin arvo, se kopioidaan metodin parametrille.

Arvomuuttujilla tilanne on yksinkertainen. Jos metodille annetaan argumenttina `3` ja metodissa tätä arvoa kasvatetaan, muutos näkyy ainoastaan metodin määrittelemällä näkyvyysalueella.

Olioiden kanssa tilanne on valitettavasti monimutkaisempi. Jos esimerkkimetodilla `changeReference` on parametri `greetings` ja kutsuvasta metodista annetaan argumentti `greetingsArgument`, parametriin kopioidaan arvoksi sama viite kuin argumentilla. Jos `changeReference`-metodissa viite asetetaan osoittamaan uuteen olioon, parametrin arvo vaihtuu, mutta argumentin arvo säilyy ennallaan, koska parametrin arvo oli vain kopio argumentista: ei siis sama viite. Jos esimerkkimetodissa `useReference` kutsutaan parametriolion `append`-metodia argumentilla `Thanks!`, sekä parametri että argumentti osoittavat kuitenkin edelleen samaan olioon, joten sekä suorittaja että vastaanottaja näkevät `append`-metodilla tehdyn muutoksen.

```
public class ReferenceTest {
  private static void changeReference(StringBuffer greetings) {
    greetings = new StringBuffer("Thanks!");
  }

  private static void useReference(StringBuffer greetings) {
    greetings.append(" Thanks!");
  }

  public static void main(String[] args) {
    StringBuffer greetingsArgument = new StringBuffer("Hello!");
    System.out.println(greetingsArgument.toString());
    changeReference(greetingsArgument);
    System.out.println(greetingsArgument.toString());

    StringBuffer goodbyeArgument = new StringBuffer("Goodbye!");
    System.out.println(goodbyeArgument.toString());
    useReference(goodbyeArgument);
    System.out.println(goodbyeArgument.toString());
  }
}
```

Ohjelman tulosteeksi saadaan:

```
Hello!
Hello!
Goodbye!
Goodbye! Thanks!
```

C#:ssa parametrinvälityksessä käytetään oletuksena samaa arvonvälitysmenetelmää, mutta tarvittaessa parametrit voidaan välittää myös viitteenvälitysmenetelmällä (engl. *pass by reference*). Tällöin kutsuttu metodi voi muuttaa annetun viitteen arvoa. Viitteenvälitys valitaan käyttöön `ref`-avainsanalla. `Ref`-avainsanaa on silloin käytettävä sekä metodin määrittelyssä että metodia kutsuttaessa, jottei metodi pääse yllättäen muuttamaan viitteen arvoa ilman kutsun suorittajan erillistä lupaa.

```
using System;
using System.Text;

public class ReferenceTest {
  public static void setThanks(StringBuilder input) {
    input = new StringBuilder("Thanks!");
  }

  public static void setThanksByReference
          (ref StringBuilder input) {
    input = new StringBuilder("Thanks!");
  }

  public static void Main(string[] args) {
    StringBuilder greetings = new StringBuilder("Hello!");
    System.Console.WriteLine(greetings.ToString());
    setThanks(greetings);
    System.Console.WriteLine(greetings.ToString());
    setThanksByReference(ref greetings);
    System.Console.WriteLine(greetings.ToString());
  }
}
```

Ohjelman tulosteeksi saadaan:

```
Hello!
Hello!
Thanks!
```

4.6 Delegaatit ja osoittimet

Javassa ja C#:ssa metodeihin viittaaminen on toteutettu eri tavalla. Esimerkiksi graafisia käyttöliittymiä ohjelmoitaessa komponentille määritetään, mikä metodi sen tulee ajaa tietyn tapahtuman ilmetessä. Javassa suoraan metodiin viittaaminen ei ole mahdollista. Tämä kierretään yleisesti ohjelmoimalla kuuntelijarajapinnan toteuttava olio, joka sisältää tapahtuman ilmetessä ajettavan metodin. Alla esimerkki, kuinka painikkeen painallustapahtumaan liitetään kutsuttava metodi:

```
loginButton.addMouseListener(new java.awt.event.MouseAdapter() {
  public void mouseClicked(java.awt.event.MouseEvent evt) {
    loginButtonMouseClicked(evt);
  }
});
```

C#:ssa funktioihin viittaaminen onnistuu delegaateilla. C#:ssa on viitteiden ja delegaattien lisäksi myös esimerkiksi C-kielestä tutut osoittimet. Osoittimilla pystytään viittaamaan suoraan muistipaikkaan, jossa on primitiivimuuttuja tai tietue, joka sisältää primitiivimuuttujia [21].

4.7 Osittaiset luokat

Osittaiset luokat (engl. *partial classes*) mahdollistavat luokan jakamisen useampaan eri lähdekooditiedostoon. Näin esimerkiksi graafisissa käyttöliittymissä itse käyttöliittymän asettelu- ja muotoilukoodi voidaan eriyttää sen toiminnallisuutta kuvaavasta koodista ja lopputuloksena on selkeämpi kokonaisuus, jolloin kehittäjä voi keskittyä kulloinkin olennaiseen osaan.

Javassa osittaisia luokkia ei ole. Siksi yhden ikkunan ohjelmakoodissa on sekä käyttöliittymä komponenttien sijoittelu ja muotoilu että ikkunan käyttöliittymän logiikkakoodi. Esimerkiksi NetBeansillä käyttöliittymää kehittäessä tätä koodin paljoutta yritetään piilottaa tekemällä sijoittelu ja muotoilu niiden tekemiseen tarkoitetulla apuohjelmalla, eikä suoraan koodilla. Oletuksena NetBeans piilottaa tämän apuohjelman luoman koodin ja estää sen muokkaamisen. Muotoilu- ja sijoittelukoodi on kuitenkin samassa luokassa käyttöliittymän logiikan kanssa ja luokan koko kasvaa.

C#:ssa muotoilu ja sijoittelu sijoitetaan omaan tiedostoonsa, jota käyttäjän ei pitäisi joutua muokkaamaan suoraan ollenkaan. Luokan toiseen osaan jää vain käyttöliittymän logiikkaan liittyvä koodi. Osittaiset luokat sopivat myös muualle. Käyttöliittymäohjelmointi on vain yksi esimerkki niiden käytöstä.

4.8 Lambda-lausekkeet

C#:n lambda-lausekkeet ovat anonyymejä funktioita. [19] Anonyymit funktiot ovat nimensä mukaisesti nimettömiä funktioita. Anonyymejä funktioita käytetään usein tilanteissa, joissa funktiota ei tarvitse pystyä kutsumaan muualta ja funktio on suhteellisen yksinkertainen. Esimerkki tällaisesta tilanteesta on listojen läpikäynti tilanteessa, jossa listasta etsitään tietyn ehdon täyttäviä alkioita tai halutaan järjestää alkiot jotenkin muuten kuin alkioiden luonnolliseen järjestykseen. Alla esimerkki,

miten kokonaislukutaulukosta voidaan laskea parittomien alkioiden lukumäärä:

```
public class LambdaExpressionExample {
  public static void Main(string[] args) {
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
    int oddNumbers = numbers.Count(n => n % 2 == 1);
    System.Console.WriteLine(oddNumbers);
  }
}
```

`Count`-metodille ei tarvitse antaa tyyppiparametriä. C#:in tyypin määrittely (engl. *type inference*) pystyy päättelemään geneerisen metodin tyypin argumenttien perusteella. [16] `Count`-metodi on määritelty muodossa `Enumerable.Count<TSource> Method (IEnumerable<TSource>, Func<TSource, Boolean>).` [17] Esimerkin lista `numbers` on tyyppiä `IEnumerable<int>`, joten anonyymin funktion geneerinen tyyppi `Func<TSource, Boolean>` pystytään päättelemään eksplisiittiseksi tyypiksi `Func<int, Boolean>`.

Javassa ei ole mahdollista tehdä lambda-lausekkeita. [10] Lähes samanlaisen toiminnallisuuden voi tehdä käyttämällä anonyymejä luokkia. Kuten yllä olevassa C#-esimerkissä, on myös Java-esimerkin `count`-metodi geneerinen eli samalla metodilla voitaisiin laskea esimerkiksi oliolistasta niiden olioiden lukumäärä, joiden attribuutit täyttävät halutun ehdon. C#:sta poiketen Javassa geneeriset tyypit voivat olla vain oliotyyppejä. [1]

```
interface Predicate<T> { public boolean apply(T type); }

public class AnonymousClassExample {
  public static <T> int count(T[] values, Predicate<T> predicate) {
    int amount = 0;
    for (T value : values) {
      if (predicate.apply(value)) {
        amount++;
      }
    }
    return amount;
  }

  public static void main(String[] args) {
    Integer[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
    int oddNumbers = count(numbers, new Predicate<Integer>() {
      public boolean apply(Integer number) {
        return number % 2 == 1;
      }
    });
    System.out.println(oddNumbers);
  }
}
```

4.9 Suoritusteho

Java ja C# ovat molemmat virtuaalikoneessa ajettavia kieliä. Tästä voisi nopeasti vetää johtopäätöksen, että molemmat ovat hitaampia C:hen ja C++:aan verrattuna, koska niillä kirjoitetut ohjelmat käännetään suoraan binäärikoodiksi. Tilanne ei kuitenkaan ole aivan niin yksinkertainen. Esimerkiksi IBM:n suorittamien testien perusteella Javan muistinvaraus on C++:n muistinvarausta nopeampaa, vaikka C++:ssa ohjelmoija voi päättää, varataanko tila keosta vai pinosta [9].

Vaikka korkeamman tason kielillä ohjelmoijan mahdollisuudet yksityiskohtaiseen suoritustehon virittelyyn ovatkin pienemmät, ei tämä ole välttämättä suora osoitus suoritustehon laskemisesta. C:täkin voitaisiin pitää Assemblyä hitaampana kielenä, koska C on välitaso ja ohjelmoijan kirjoittaessa suoraan Assembly-kieltä hän voi tehdä kaikki hyväksi toteamansa optimoinnit, joita kääntäjä ei osaa tehdä.

Kannattaa kuitenkin muistaa, että ohjelmien suoritustehovaatimukset vaihtele-

vat ohjelmien luonteesta riippuen. Esimerkiksi verkkokauppaa tehdessä suuri osa käyttäjän kokemasta hitaudesta johtuu HTTP-yhteyksien hitaudesta ja tähän ei voi ohjelmointikielen valinnan vaikuttaa. Donald Knuth, eräs tietotekniikan alan suurista tutkijoista, on todennut ennenaikaisen optimoinnin olevan kaiken pahan alku ja juuri. Hän teki tämän väitteen jo vuonna 1974 [13].

Harvaa ohjelmaa kirjoitetaan tilanteessa, jossa aikaa ja rahaa olisi tarjolla loputtomasti. Jos rupeaa alusta asti kirjoittamaan käyttöliittymärutiineja C:llä tai jopa Assemblyllä, venyy kehitysaika varmasti Javaan ja C#:iin verrattuna, koska näille on tarjolla hyvin tehokkaita käyttöliittymäkirjastoja ja -työkaluja. Jos epäilee, että suoritusteho ei korkeamman tason kielillä välttämättä riitä, kannattaa kirjoittaa ensin prototyyppi korkeamman tason kielillä ja ajaa tämä kattavien suoritustehotestin läpi. Todennäköisesti pahimmassakin tapauksessa lisäsuoritustehoa vaativat ohjelman osat ovat pieni osa kokonaisuudesta. Nämä osat voidaan sitten optimoida esimerkiksi hiomalla algoritmit huippuunsa ja jos tarvetta oikeasti vielä tässä vaiheessa on, kirjoittaa ne uudestaan alemman tason kielillä ja liittää osaksi muuta ohjelmaa.

5 Yhteenveto

Maailmassa on valtava määrä ohjelmointikieliä. Eri ohjelmointikielet on suunniteltu erilaisia tarpeita varten. Vanhoja ohjelmointikieliä korvataan jatkuvasti uusilla. Vain yhden ohjelmointikielen osaaminen ei siis ole kannattavaa.

Tutkimuksen tarkoituksena oli selvittää, voidaanko Java-ohjelmointikielen osaamista hyödyntää C#-ohjelmointikielen opiskelussa. Tutkimuksessa selvitettiin ohjelmointikielten yhtäläisyyksiä ja eroja ohjelmoijan näkökulmasta.

Yhtäläisyyksiä ja eroja selvitettiin vertaamalla etupäässä Microsoftin sekä aiemmin Sunin, mutta nykyään Oraclen tekemiä kielten ohjelmointioppaita, rajapintakuvauksia ja kielten teknisiä spesifikaatioita. Näiden lisäksi materiaalina käytettiin myös kolmansien osapuolten artikkeleita Javasta, C#:sta ja ohjelmoinnista yleisesti.

Tutkimuksen tulosten perusteella Javan osaamista voidaan käyttää hyvin C#:in opiskelussa. Molemmat kielet käyttävät olio-ohjelmoinnin paradigmaa, yhtälaista syntaksia ja virtuaalikonetta tavukoodin tulkitsemisessa käytettävän tietokoneen ymmärtämäksi konekoodiksi. Eroja kielten välillä ovat muun muassa C#:in käyttämät tietueet primiitimuuttujien kantaluokkana, virtuaalikoneiden saatavuus eri käyttöjärjestelmissä sekä C#:in tuki delegaateille ja lambda-lausekkeille.

Tutkimusten tulosten perusteella Javaa osaava ohjelmoija voi siirtyä helposti käyttämään C#:ia ja vastaavasti C#:ia osaava ohjelmoimaan Javaa. Tästä on hyötyä

erityisesti yritysmaailmassa, jossa on tätä kirjoittaessa pula tietoteknologian osaajista Tietotekniikan liiton toiminnanjohtaja Robert Serénin mukaan [12]. Esimerkiksi C#-osaajaa etsivän yrityksen ei tarvitse tutkimusten tulosten mukaan hylätä Java-osaajaa, vaan hakija pystyy todennäköisesti omaksumaan uuden kielen nopeasti.

Voin vahvistaa tutkimusten tulosten pitävän paikkaansa omakohtaisten kokemuksieni perusteella. Keväällä 2010 vaihdoin Java-ohjelmointitehtävistä C#:ia käyttävään yritykseen. Siirtyminen tuntui helpolta ja se onnistui nopeasti.

Lähteet

[1] Allen, E., *Diagnosing Java code: Java generics without the pain, Part 1*, saatavilla WWW-muodossa <URL: http://www.ibm.com/developerworks/java/library/j-djc02113.html>, 1.3.2011.

[2] Byous, J., *Java Technology: The Early Years*, saatavilla WWW-muodossa <URL: http://web.archive.org/web/20080530073139/http://java.sun.com/features/1998/05/birthday.html>, 24.1.2011.

[3] CNET News, *Microsoft's Holy War On Java*, saatavilla WWW-muodossa <URL: http://news.cnet.com/Microsofts-holy-war-on-Java/2009-1001_3-215854.html>, 23.9.2008.

[4] CNET News, *Sun settles with Microsoft, announces layoffs*, saatavilla WWW-muodossa <URL: http://news.cnet.com/Sun-settles-with-Microsoft,-announces-layoffs/2100-1014_3-5183848.html>, 7.12.2008.

[5] Codehaus Foundation, *Groovy - An agile dynamic language for the Java Platform*, saatavilla WWW-muodossa <URL: http://groovy.codehaus.org/, 24.1.2011.

[6] Computer World, *The A-Z Of Programming Languages: C#*, saatavilla WWW-muodossa <URL: http://www.computerworld.com.au/article/261958/-z_programming_languages_c?pp=1&fp=&fpid=>, 6.12.2008.

[7] DedaSys LLC, *Programming Language Popularity*, saatavilla WWW-muodossa <URL: http://langpop.com/>, 25.1.2011.

[8] École Polytechnique Fédérale de Lausanne, *The Scala Programming Language*, saatavilla WWW-muodossa <URL: http://www.scala-lang.org/node/25>, 24.1.2011.

[9] IBM developerWorks, *Java theory and practice: Urban performance legends, revisited*, saatavilla WWW-muodossa <URL: http://www.ibm.com/developerworks/java/library/j-jtp09275.html>, 18.1.2011.

[10] Java Community Process, *JSR 335: Lambda Expressions for the JavaTM Programming Language*, saatavilla WWW-muodossa <URL: http://jcp.org/en/jsr/detail?id=335>, 1.3.2011.

[11] Jones, A. & Freeman, A., 2003. *C# for Java Developers*. Redmond, Washington: Microsoft Press.

[12] Kaleva, *Oikeanlaisista it-osaajista ollut pulaa*, saatavilla WWW-muodossa <URL: http://www.kaleva.fi/uutiset/oikeanlaisista-it-osaajista-ollut-pulaa/889837, 21.2.2011.

[13] Knuth, D.E., *Structure Programming with go to Statements*, saatavilla WWW-muodossa <URL: http://www.univasf.edu.br/~marcus.ramos/pc-2008-2/p261-knuth.pdf>, 25.1.2011.

[14] Kurniawan B., *Comparing C# and Java*, saatavilla WWW-muodossa <URL: http://ondotnet.com/pub/a/dotnet/2001/06/07/csharp_java.html?page=1>, 24.1.2011.

[15] Lämmel, R. & Jones, S.P., *Scrap Your Boilerplate: A Practical Design Pattern for Generic Programming*, saatavilla WWW-muodossa <URL: http://homepages.cwi.nl/~ralf/tldi02.pdf>, 24.1.2011.

[16] Microsoft Developer Network, *C# Language Specification*, luku 7.5.2, saatavilla WWW-muodossa <URL: http://msdn.microsoft.com/en-us/library/ms228593.aspx>, 1.3.2011.

[17] Microsoft Developer Network, *Enumerable.Count<TSource> Method (IEnumerable<TSource>, Func<TSource, Boolean>)*, saatavilla WWW-muodossa <URL: http://msdn.microsoft.com/en-us/library/bb535181.aspx>, 1.3.2011.

[18] Microsoft Developer Network, *Int32 Structure*, saatavilla WWW-muodossa <URL: http://msdn.microsoft.com/en-us/library/system.int32.aspx>, 24.1.2011.

[19] Microsoft Developer Network, *Lambda Expressions*, saatavilla WWW-muodossa <URL: http://msdn.microsoft.com/en-us/library/bb397687.aspx>, 1.9.2009.

[20] Microsoft Developer Network, *Overloadable Operators*, saatavilla WWW-muodossa <URL: http://msdn.microsoft.com/en-us/library/8edha89s(VS.71).aspx>, 24.1.2011.

[21] Microsoft Developer Network, *Pointer Types (C#)*, saatavilla WWW-muodossa <URL: http://msdn.microsoft.com/en-us/library/y31yhkeb(VS.80).aspx>, 3.9.2009.

[22] Microsoft Developer Network, *struct (C#)*, saatavilla WWW-muodossa <URL: http://msdn.microsoft.com/en-us/library/ah19swz4(VS.71).aspx>, 1.12.2008.

[23] Microsoft Developer Network, *Type system unification*, saatavilla WWW-muodossa <URL: http://msdn.microsoft.com/en-us/library/aa664638(VS.71).aspx>, 7.1.2009.

[24] Mono Project, *Mono*, saatavilla WWW-muodossa <URL: http://www.mono-project.com/>, marraskuu 2008.

[25] SunWorld, *Microsoft licenses Java*, saatavilla WWW-muodossa <URL: http://sunsite.uakom.sk/sunworldonline/swol-12-1995/swol-12-microsoft.html, 2.5.2009.