Siirtyminen Visual Basic .NETistä C#:iin tietojenkäsittelyn koulutusohjelmassa

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

Visamäki, 4.10.2010

Juha Ijäs

HAMK
HÄMEEN AMMATTIKORKEAKOULU

OPINNÄYTETYÖ

Tietojenkäsittelyn koulutusohjelma
Hämeenlinna

Työn nimi	Siirtyminen Visual Basic .NET:stä C#:iin tietojenkäsittelyn koulutusohjelmassa
Tekijä	Juha Ijäs
Ohjaava opettaja	Tommi Saksa
Hyväksytty	_____._____.20_____
Hyväksyjä	

HAMK
HÄMEEN AMMATTIKORKEAKOULU

TIIVISTELMÄ

Hämeenlinna
Tietojenkäsittelyn koulutusohjelma

Tekijä Juha Ijäs **Vuosi** 2010

Työn nimi Siirtyminen Visual Basic .NET:stä C#:iin tietojenkäsittelyn koulutusohjelmassa

TIIVISTELMÄ

Opinnäytetyön toimeksiantajana oli Hämeen ammattikorkeakoulu. Toimeksiantaja halusi muuttaa tietojenkäsittelyn koulutusohjelmassa olevan Ohjelmoinnin perusteet -opintojakson käytettävän ohjelmointikielen Visual Basic .NET:stä C#:iin. Ohjelmointikielen muutos tarvittiin, koska opiskelijoiden tulisi osata ohjelmoida C#:lla tulevissa ohjelmointiopintojaksoissa. C#:sta on myös helpompi siirtyä tulevissa opintojaksoissa ohjelmoitavaan Javaan kuin Visual Basic .NETistä. Ohjelmointikielet ovat syntaksiltaan hyvin lähellä toisiaan.

Tutkimusongelmana opinnäytetyössä oli selvittää, kannattaako tietojenkäsittelyn koulutusohjelman siirtyä käyttämään C#-ohjelmointikieltä Visual Basic .NET:n sijaan. Lisäksi selvitettiin ohjelmointikielien eroja ja sekä niiden tulevaisuutta. Tutkimusmenetelmänä opinnäytetyössä oli kehitysprojekti. Opinnäytetyössä käytettiin kirjallisia ja sähköisiä lähteitä. Vanhan opintojakson Moodle-alustan perusteella rakennettiin opintojakso C#:lle. Myös uusi opintojakso toteutettiin Moodle-ympäristöön ja sieltä löytyvät kaikki opintojakson materiaalit ja harjoitteet.

Teoriaosuudessa käsiteltiin, miten .NET Framework toimii ja miksi Microsoft on kehittänyt .NET Frameworkin. Osuudessa käsiteltiin myös C#:n ja Visual Basic .NET:n eroavaisuuksia. Teoriaosuuden jälkeen käsiteltiin Ohjelmoinnin perusteet -opintojaksoa ja sen sisältöä. Lisäksi käytiin läpi ohjelmointiopintojaksoja tietojenkäsittelyn koulutusohjelmassa ja niiden muuttumista tulevaisuudessa. Lopuksi kerrottiin tiivistetysti tutkimustuloksista. Tutkimustulosten perusteella ei voitu antaa selvää vastausta ohjelmointikielien paremmuudesta. Tuloksista voitiin päätellä kuitenkin, että C# sopii opintojaksolle paremmin verrattaessa muita tietojenkäsittelyn opintojaksoja ja koko opetussuunnitelmaa.

Avainsanat C#, Visual Basic .NET, tietojenkäsittely, ohjelmointi

Sivut 34 s.

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Hämeenlinna
Degree Programme in Business Information Technology

Author Juha Ijäs **Year** 2010

Subject of Bachelor's thesis Changing over from Visual Basic .NET to C# in Degree Programme in Business Information Technology

ABSTRACT

This thesis was assigned by HAMK University of Applied Sciences. The client wanted to change the programming language from Visual Basic to C# in a study module called Programming Basics in the Degree Programme in Business Information Technology. The programming language change was needed because students should know how to program C# in advanced study modules. It is also easier to adapt from C# than Visual Basic .NET to Java, which is used in some study modules. The language syntax in C# is close to Java.

The research question in this thesis was to examine if the Degree Programme in Business Information Technology should switch from Visual Basic .NET to C# instead. The thesis also researched differences in these programming languages, and what kind of future those programming languages have. The research method in this thesis was constructive research. This thesis used written and electronic sources. The old study module in Moodle platform was used to build a new module in Moodle. All materials and exercises are in that module.

The theoretical part examined how .NET Framework works, and why Microsoft has developed .NET Framework. This section also compared Visual Basic .NET and C# differences in language syntax. After the theoretical part the thesis described how the Programming Basics study module is executed. Also other programming study modules in Programming Basics in the Degree Programme in Business Information Technology were briefly described. Summary in the end explained the research results of the thesis. The results could not show a clear answer to which programming language is better because of the similarities. C# is generally better for the Degree Programme in Business Information Technology because the similarities in advanced study modules.

Keywords C#, Visual Basic .NET, programming

Pages 34 p.

SANASTO

Assembly	Sisältää IL-ohjelmakoodia, ajettavan tiedoston sisällä. Tiedostoa ei voi ajaa, jollei mukana ole manifesti-tiedostoa. Assemblyssä voi olla useampi Managed Module, resurssitiedostoja ynnä muuta.
Common Language Runtime	Common Language Runtime on ajoympäristö .NET-ohjelmille.
Common Language Specification	Määrittelee, mitä ominaisuuksia on ohjelmointikielessä, mikä voidaan kääntää Intermediate Language -muotoon.
Common Type System	Common Type System kontrolloi ohjelman muuttujien tietoa. Se määrittelee käytetyt tietotyypit .NET-ohjelmassa.
Global Assembly Cache	.NET-kirjastot, joita käyttää useampi kuin yksi ohjelma samalta koneelta, tallennetaan global assembly cacheen.
Manifest	Assemblyjen mukana on manifest. Se sisältää tietoa, miten eri elementit assemblyssä liittyvät toisiinsa. Lisäksi manifest sisältää metatietoa assemblystä, kuten version, nimen ynnä muuta Manifest voi olla liitettynä dll tai exe-tiedostoon tai olla omana tiedostonaan.
Managed Module	Managed Module on ohjelmasta käännetty versio, jonka sisällä on IL-ohjelmakoodia ja metadataa.
Namespace	Namespace on avaruus, jonka alla on luokkia. Esimerkiksi Forms-namespacen alla on lomakkeisiin liittyviä luokkia.
Intermediate Language	Intermediate Language on välikieli. Kääntäjät kääntävät ohjelmat IL-muotoon .NET Frameworkia varten. CLR-ajoympäristö osaa tulkita IL-kieltä.
Just In Time	Kääntäjä, joka kääntää Intermediate Language –koodista prosessorin ymmärtämää konekieltä. Kääntäminen tapahtuu juuri ennen ohjelman ajoa.
PE32	PE32 on ajettavassa tai kirjasto-ohjelmassa oleva headeri, jossa on tietoa tiedostotyypistä, aikaleima käännöksen ajankohdasta ja tietoa mahdollisesta natiivista ohjelmakoodista.
WYSIWYG	What you see is what you get. Graafinen editori, jossa näkee suoraan, minkänäköinen tuotoksesta tulee.

SISÄLLYS

1 JOHDANTO

Ohjelmoinnin opetus on osa ammattikorkeakoulujen tietojenkäsittelyn opetusohjelmaa. On tärkeää, että ohjelmoinnin opetuksessa käytettävät välineet ja menetelmät ovat ajan tasalla. Opetuksen on pystyttävä vastamaan työelämän asettamiin haasteisiin. Jatkuva kehitys opetuksessa takaa valmistuville tietojenkäsittelyn tradenomeille hyvät mahdollisuudet selviytyä työelämässä.

Opinnäytetyön toimeksiantajana on Hämeen ammattikorkeakoulu (myöhemmin HAMK). HAMK tarjoaa monialaista koulutusta 24 eri koulutusohjelmalla. HAMKissa voi opiskella kokopäiväisesti tai työn ohessa. Opiskelijoita on noin 7000 ja henkilöstöä 800. Tietojenkäsittelyn opetuksessa ensikosketus ohjelmointiin tehdään tällä hetkellä Visual Basicilla. Opinnäytetyössä otetaan selville, kannattaako HAMKin siirtyä Visual Basicista Visual C#:iin. (Microsoft kutsuu ohjelmointikieltään Visual C#:ksi, mutta jatkossa tässä työssä käytetään pelkkää C#-nimitystä.)

Aikaisemmin HAMKilla ei ole ollut opintojaksoa, jossa opetellaan C#:a. Nyt sitä pitäisi opettaa Ohjelmoinnin perusteet -opintojaksolla. Ohjelmointi 1 muutetaan myös Visual Basicista C#:lle.

Kun Microsoft julkaisi .NET Frameworkin 2002, se päivitti myös Visual Basic 6.0:n toimimaan .NET Frameworkissa. Tällöin sai alkunsa Visual Basic .NET. Uutena ohjelmointikielenä .NET –ympäristöön tuli C#. Se on Visual Basiciin verrattuna syntaksiltaan helpompi oppia, jos on aikaisempaa kokemusta C++:sta tai Javasta. Toisaalta myös muihin ohjelmointikieliin siirtyminen on helpompaa. C# on enemmän olio-ohjelmointipainotteinen verrattuna Visual Basiciin, joka voi vaikeuttaa C#:n opettelua. C# pakottaa ohjelmoijan käyttämään enemmän olio-ohjelmointia kuin Visual Basic .NET. Visual Basic on BASIC-pohjainen ohjelmointikieli. Muita suosittuja samaan syntaksiin perustuvia ohjelmointikieliä ei ole.

C#:n käyttö on yleistynyt ja Visual Basicin hieman laskenut. Molempia ohjelmointikieliä tuetaan kuitenkin nykyisessä .NET 4.0-versiossa. Microsoftin mukaan Visual Basic .NET:ä tuetaan myös jatkossa. Kaikki ohjelmointikielet .NET-ympäristössä käännetään samanlaisiksi. Siitä huolehtii Common language runtime (CLR). Se mahdollistaa myös erilaisten skriptikielien käytön, muun muassa Pythonin tai Perlin.

Opinnäytetyössä otetaan selville kannattaako HAMKin tietojenkäsittelyn siirtyä käyttämää C#:a opetuksessa. Tutkimuskysymyksiksi nousivat tällöin seuraavat: Mitä eroja Visual Basicillä ja C#:lla on? Miten Ohjelmoinnin perusteet -opintojakson opetusta tulee muuttaa, jotta päästään opetussuunnitelmassa määriteltyihin tavoitteisiin? Millainen tulevaisuus on Visual Basicilla ja C#:lla?

1.1 Tutkimusmenetelmät

Opinnäytetyön tutkimusmenetelmänä on kirjallinen tutkimus. Työssä on käytetty erilaisia artikkeleita liittyen ohjelmoinnin opettamiseen, erityisesti ensimmäiseen ohjelmointiopintojaksoon.

Korkeakoulujen ensimmäisestä ohjelmointiopintojaksosta löytyy erilaisia tieteellisiä artikkeleita, jossa oli kyselyiden ja testien perusteella yritetty selvittää, miten opiskelijat oppivat parhaiten ohjelmointia. Käytin näitä artikkeleita lähteinä tutkiessani erilaisia vaihtoehtoja ohjelmoinnin opettamiseen.

Käytännön työssä käytetään tutkimusmenetelmänä myös kehittämisprojektia. Ohjelmoinnin perusteet -opintojakson materiaalia tehdessä tuli esille erilaisia kysymyksiä. Esimerkiksi missä vaiheessa opintojaksoa pitäisi käsitellä mikäkin asia.

2 MICROSOFT .NET FRAMEWORK

Microsoft .NET Framework on sovelluskehys .NET-ohjelmien ajamiseen. Sovelluskehys on alusta, jonka päällä toimivat varsinaiset ohjelmat, tässä tapauksessa .NET-ohjelmat. .NET Framework toimii hajautetussa ympäristössä, missä laskenta voidaan tehdä Internetin yli palvelimilla, mutta myös perinteiset työpyötäsovellukset toimivat. (Moghadampour 2009, 12.)

.NET Framework tarjoaa valmiita rajapintoja ja kirjastoja, joita jokainen sillä ohjelmoitu sovellus voi käyttää. Uusin versio on 4.0, joka julkaistiin 2010 keväällä. .NET Frameworkin ohjelmia voi tehdä millä tahansa sovelluskehittimellä. Sovelluskehitin on ohjelma, jolla on mahdollista ohjelmoida sovelluksia. Microsoft tarjoaa kuitenkin omaa versiotaan, joka on Visual Studio. Uusin versio siitä on 2010, se julkaistiin samaan aikaan, kun .NET Frameworkista tuli versio 4.0.

Microsoft Windows -käyttöjärjestelmiä on pidetty epävakaina ja monimutkaisina. Tämä maine on tullut monien syiden seurauksena. Yhtenä syynä ovat varmasti dynamic-link libraries -kirjastot (myöhemmin DLL). Kaikki ohjelmat käyttävät DLL-tiedostoja. Kirjastoja on saatavilla Microsoftilta tai kolmansilta osapuolilta. Koska ohjelma useimmiten käyttää monia eri kirjastoja, ei ohjelmoija voi olla aivan varma, käyttääkö jokin muu ohjelma samoja kirjastoja. DLL-ristiriitoja voidaan yrittää ratkaista ohjelman testausvaiheessa. (Richter 2010, 32.)

Käyttäjät voivat kuitenkin törmätä ongelmiin, kun jokin ohjelma päättää päivittää itsensä. Jos jokin muu ohjelma käyttää samaa kirjastoa, mitä toinen ohjelma päivittää, saattaa tulla ongelmia. Kirjastojen pitäisi olla taaksepäin yhteensopivia, mutta eivät sitä välttämättä aina ole. (Richter 2010, 32.)

Kyse on ohjelmistojen riippuvuusongelmista. Toinen ohjelmisto tarvitsee kirjastosta eri version kuin toinen samaa kirjastoa käyttävä ohjelmisto. Käyttäjälle tulee ongelmia, jos hän haluaa käyttää molempia ohjelmia. Ohjelmistoriippuvuusongelmaa on myös muilla käyttöjärjestelmillä kuin Microsoft Windowsilla.

Yksi Windowsin ongelmista on ohjelmistoasennusten monimutkaisuus. Kun käyttäjä asentaa ohjelmiston, asennusohjelma tekee muutoksia kaikkialle järjestelmään. Useat asennuspaketit kopioivat tiedostoja moneen eri hakemistoon, tekevät rekisterimuutoksia ja kopioivat pikakuvakkeet Käynnistä-valikkoon sekä Työpöydälle. (Richter 2010, 32.)

Ongelma on siinä, että ohjelman tiedostojen sijainti ei ole rajoitettu yhteen hakemistoon. Ohjelmaa ei voi siirtää helposti toiseen hakemistoon tai tietokoneeseen, koska rekisteriviittaukset osoittavat vanhaan hakemistoon. Ainoa mahdollisuus muuttaa ohjelmiston asennushakemistoa on poistaa se ja asentaa uudestaan. Ohjelmissa on useimmiten mukana poistamistyökalu, mutta ohjelmista voi silti jäädä jotakin tiedostoja tai rekisterimerkintöjä tietokoneelle. (Richter 2010, 32.)

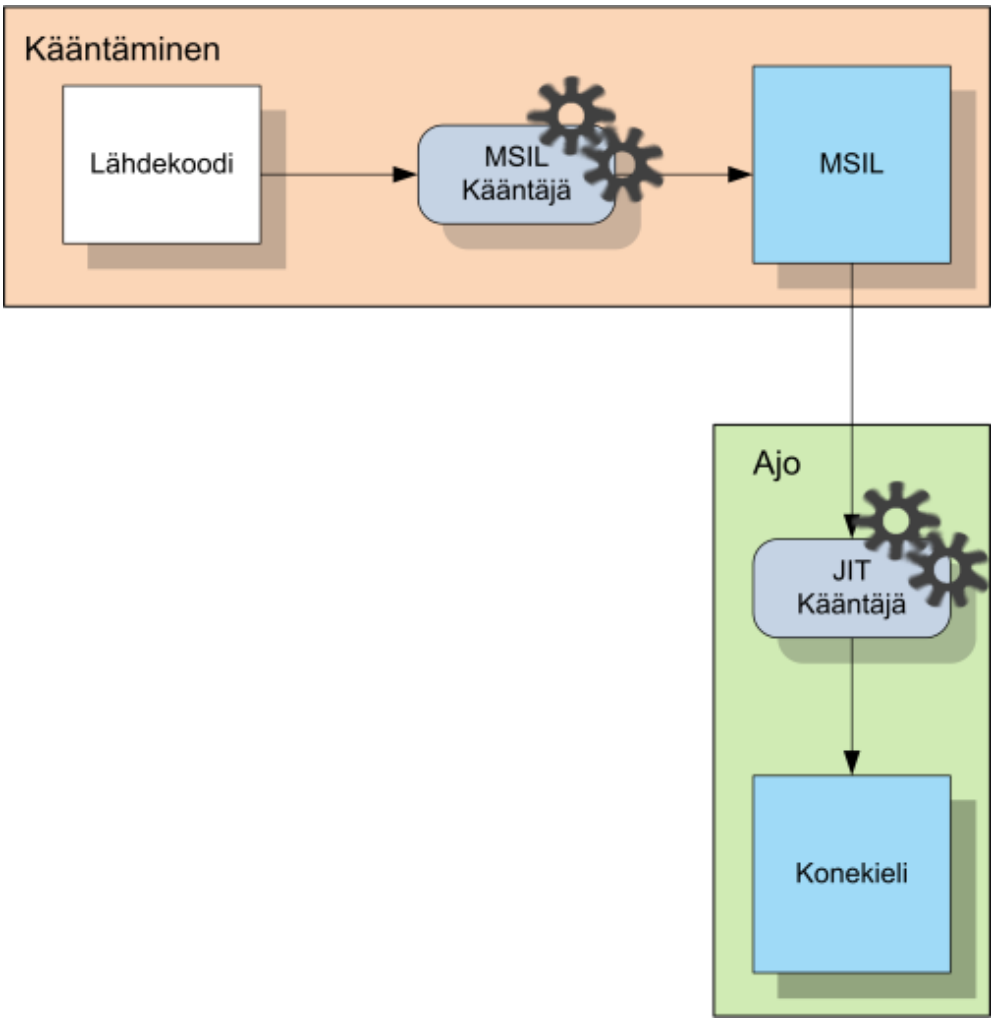
Tietoturvanäkökulmasta perinteisissä ohjelmistoissa on myös ongelmia. Kun ohjelmisto asennetaan, siinä tulee mukana monia erilaisia tiedostoja. Monet kirjastot ovat muiden kuin ohjelmiston kehittäjien ohjelmoimia. Web-sovelluksissa on myös tietoturvaongelmia. ActiveX-komponentit voivat tehdä mitä tahansa, kun käyttäjä on hyväksynyt komponentin asentamisen. ActiveX ei toimi minkäänlaisessa rajoitetussa ympäristössä. Käyttäjät voivat tuskastua kaikkien näiden tietoturvariskien ympäröiminä. (Richter 2010, 32.)

.NET Framework yrittää ratkaista näitä ongelmia. Ohjelmiston ei tarvitse välttämättä olla jakautuneena moneen eri hakemistoon. Kaikki tiedostot voivat olla samassa hakemistossa. Tietoturvaongelmaan .NET Framework tuo ratkaisuksi code access securityn. Normaali Windows-ohjelma toimii niillä käyttöoikeuksilla, mitä ohjelman käynnistäneelle käyttäjälle on annettu. Code access securityllä voidaan rajata ohjelman oikeuksia ilman käyttäjätunnusrajoituksia. Ohjelmasta oikeuksia voidaan esimerkiksi rajata niin, ettei se voi käyttää Internet-yhteyttä. (Richter 2010, 32–33.)

.NET rohkaisee ohjelmoijia olio-ohjelmointiin. Se tukee olio-ohjelmointia täysin. Kaikki tieto on olioiden sisällä .NETissä: muuttujien arvot, lähdekoodit ja käyttäjien laukaisemat tapahtumat. Kaikkeen tietoon päästään käsiksi olioiden kautta. (Patrick 2006, 3.)

.NET Framework on Windowsiin integroitu komponentti. Sillä on mahdollista ohjelmoida ja käyttää seuraavan teknologiasukupolven ohjelmia. Näissä ohjelmissa voidaan varsinainen prosessointityö tehdä serverillä. Käyttäjän operoimassa päätteessä näkyy vain ohjelman käyttöliittymä. Päätteen ja serverin välinen liikenne voi tapahtua Internetin yli tai lähiverkossa. Myös perinteisen työpöytäsovelluksen tekeminen onnistuu .NET Frameworkilla. Tällöin ohjel-

ma ajetaan samassa tietokoneessa, missä käyttöliittymä näkyy. (MSDN Library 2010a)



Kuva 1 Ohjelman kääntäminen .NET Framework –ympäristössä

.NET ei ole riippuvainen ohjelmointikielestä. Ohjelmoinnissa voidaan siis käyttää periaatteessa mitä tahansa ohjelmointikieltä, jos sille löytyy kääntäjä. Lähdekoodista ohjelma käännetään Intermediate Language (myöhemmin IL) -koodiksi. Tästä muodosta ohjelma käännetään vasta ajon aikana prosessorin ymmärtämäksi konekieleksi. Kun ohjelman on IL-muodossa .NET Framework tarjoaa ympäristön ja kirjastot ohjelmalle. Käännetty ohjelma voidaan ajaa .NET Frameworkin Common Language Runtime (CLR) -ajoympäristössä. Itse kääntämisen tekee Just-In-Time -ohjelma. (Moghadampour 2009, 12.)

Uusin versio .NET Frameworkista on 4.0. Se julkaistiin huhtikuussa 2010. Samalla julkaistiin Visual Studio 2010. Version myötä tuli paljon uudistuksia.

Näistä yksi on säikeistetyn ohjelmakoodin kirjoittamisen helpottaminen. Säikeistetyn sovelluksen tekeminen on perinteisesti ollut vaativaa. Säikeistys mahdollistaa monen eri tehtävän suorittamisen samaan aikaan. Versiossa 4.0 on myös uutena ominaisuutena dynamic language runtime (myöhemmin DLR). Se helpottaa skriptikielien käyttöä, jolloin lähdekoodista kääntäminen tehdään vasta ajon aikana. (MSDN Library 2010b)

2.1 Common Language Runtime

Common Language Runtime (myöhemmin CLR) tarjoaa ohjelmoijalle ajoympäristön. Ohjelmointiympäristö pakottaa ohjelmoijan käyttämään turvallista koodia. Se tarjoaa ohjelman käyttöön palveluja ajettavalle ohjelmalle. CLR:ssä on automaattinen muistinhallinta ja se hallitsee myös ohjelman eri säikeitä. (Moghadampour 2009, 13.)

CLR on .NET Frameworkin keskus. Kaikki .NET ohjelmointikielet käyttävät sen ajoympäristöä. Kaikki, mitä .NET-ohjelma tekee, tapahtuu CLR:n avulla. Se vastaa ohjelman toimivuudesta. Koska ohjelma ajetaan ajoympäristössä, on se hallinnoitua (turvallista) koodia. (Patrick 2006, 8.)

Normaalissa C/C++ -ohjelmassa muistinhallinta ja säikeistys jää ohjelmoijan huoleksi. Ohjelmoija saa päättää, miten käyttää muistia ja luoda säikeitä haluamallaan tavalla. Tämä tuo vapautta, mutta voi myös monimutkaistuttaa ohjelmointityötä. (Richter 2010, 1.)

Aikaisemmin prosessorien kellotaajuudet ovat kasvaneet ajan saatossa. Ohjelma pystyttiin ajamaan nopeammin uudemmalla (= suuremmalla kellotaajuudella olevalla) prosessorilla kuin vanhalla. Kellotaajuuksien nostaminen ei kuitenkaan onnistu nykyisellä tekniikalla loputtomiin. Prosessorin tuottamaa lämpöä ei saada siirrettyä pois riittävän nopeasti. Prosessorivalmistajat keskittyvätkin nykyään siihen, miten saada prosessorin sisältämistä transistoreista mahdollisimman pieniä. Tämän kehityksen johdosta yhteen mikropiiriin saadaan useampi prosessoriydin. (Richter 2010, 699.)

Ohjelma, jossa on useampi kuin yksi säie toimii nopeammin moniydinprosessoreissa. Laskentaa voidaan jakaa useammalle ytimelle. Ytimet laskevat rinnakkain nopeammin. Yksisäikeistä ohjelmaa voidaan ajaa vain yhdellä ytimellä.

CLR mahdollista minkä tahansa ohjelmointikielen käytön. Tosin kielelle on löydettävä kääntäjä. Kääntäjä kääntää ohjelman Intermediate Language -muotoon. CLR osaa käsitellä IL-muotoa. (Richter 2010, 2.)

2.2 Managed Module

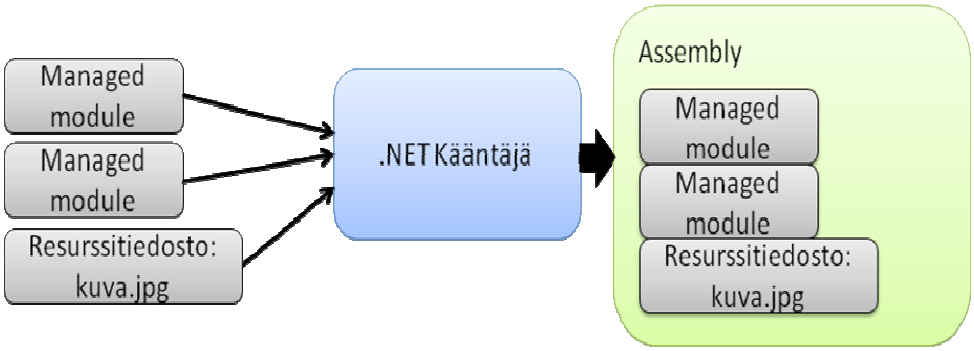
Managed Module on lähdekoodista käännetty moduuli, joka tarvitsee CLR:n ajoympäristökseen. Se sisältää PE32-headerin, CLR-headerin, metadataa ja IL-ohjelmakoodia. Normaali PE32-headeri voidaan ajaa 32 tai 64-bittisessä Windowsissa, PE32+ toimii vain 64-bittisessä. PE32-header sisältää tietoa siitä, onko tiedosto ajettava ohjelma vai kirjasto. Lisäksi mukana on tieto, milloin ohjelma on käännetty. Moduuleissa, joissa on pelkästään IL-koodia, suurin osa PE32-headerin tiedosta jätetään huomiotta. Jos moduulissa on valmiiksi prosessorille käännettyä konekieltä, tämä header sisältää tietoa siitä koodista. (Richter 2010, 3.)

CLR Headerissa on informaatiota, mikä tekee ohjelmasta Managed Modulen. CLR käyttää tätä informaatiota, voidakseen ajaa ohjelman. Header sisältää tiedon, mikä versio CLR:stä tarvitaan ajamaan tämä ohjelma. Lisäksi sieltä löytyy tieto, missä on ohjelman pääfunktio. Pääfunktio ajetaan, kun ohjelma käynnistetään. (Richter 2010, 3.)

Managed Modulen metadata sisältää tietoa ohjelman lähdekoodissa esiintyvistä sisäisistä tyypeistä ja luokkien ilmentymistä. Metadatassa on myös tietoa ulkoisista tyypeistä ja luokkien ilmentymät. Ulkoiset tyypit ja luokkien ilmentymät voivat kuulua johonkin kirjastoon. Ne eivät ole suoraan ohjelman lähdekoodissa. (Richter 2010, 3–4.)

Managed Modulen tärkein osa on IL-koodi. Se sisältää varsinaisen toiminnallisen ohjelmakoodin. Koodi on käännetty lähdekooodista IL-muotoon. Tämän koodin CLR kääntää ajon aikana prosessorin ymmärtämäksi konekieleksi. (Richter 2010, 3.)

Assembly on .NET Frameworkissa yleisimmin ohjelman käännetty versio. Se voi olla ajettava EXE-tiedosto tai DLL-kirjasto. On mahdollista jakaa assembly useammaksi tiedoksi, mutta yleensä assembly on yksi tiedosto. Assemblyyn kuuluu EXE:n tai DLL-tiedoston lisäksi manifesti. Assemblyissä, joissa on vain yksi tiedosto, manifesti sisältyy tiedostoon. Se voi olla myös omana tiedostonaan. (Patrick 2006, 15.)



Kuva 2 Esimerkki assemblystä

Kaikki tiedostot, jotka tulevat assemblyyn menevät kääntäjän läpi. Se kokoaa niistä assemblyn. Siinä voi olla useampi managed module tai resurssitiedosto. Kyseisessä esimerkkikuvassa assemblyyn on koottu kaksi managed modulea sekä kuva.jpg-resurssitiedosto.

Manifesti sisältää tietoa assemblystä. Manifestiin on tallennettu assemblyn nimi, versio, oletus alueasetus, tietoa sen käyttämistä ulkoisista assemblyistä ja tiedostolistaus koko assemblystä. CLR ei tunnista assemblyä ilman manifestia. (Patrick 2006, 15.)

Kun sovellus otetaan käyttöön, kaikki sen assemblyt tai muut sen tiedostot tulevat yleensä kaikki samaan hakemistoon. .NET Kirjastot, joita käyttää useampi kuin yksi ohjelma yhdeltä koneelta, tallennetaan global assembly cacheen (myöhemmin GAC). Assemblyt, jotka on sijoitettu GAC:iin, tarvitsevat strong namen. (Patrick 2006, 15.)

Strong namea käytetään suojaamaan Internetin läpi kuljetettua liikennettä. Siinä käytetään julkista ja henkilökohtaista avainta salaamaan tietoa. Salaukseen ja purkamiseen käytetään tällöin eri avainta. Tiedon oikeellisuuden tarkistamiseen käytetään tiivistettä, joka muodostuu lähetettävästä tiedosta. (Strong Names Explained 2004.)

Ohjelmoijalla on mahdollisuus parantaa ohjelmansa metadataa laittamalla ohjelman lähdekoodiin attribuutteja. Mikä tahansa .NET -ohjelmointiympäristö voi hakea attribuutit ohjelman metadatasta ja toimia niiden mukaan. Attribuutilla voidaan esimerkiksi merkitä jokin funktio vanhentuneeksi. Tällöin Visual Studio varoittaa, jos ohjelmoija käyttää vanhentuneeksi merkittyä funktiota. (Patrick 2006, 16.)

2.3 Common Language Specification

Common Language Specification (myöhemmin CLS) mahdollistaa eri ohjelmointikielien kommunikoinnin keskenään. CLS:n avulla voidaan tehdä samanlainen ohjelma eri ohjelmointikielillä. Tämä on mahdollista, koska CLS

sisältää standardit tyypeistä, metadatasta ja ajoympäristöstä. (Richter 2010, 25.)

Ohjelmointikielissä on monia eri poikkeavuuksia toisistaan. Esimerkiksi jotkin ohjelmointikielet käsittelevät pieniä ja suuria kirjaimia muuttujissa samanlaisina, rivinvaihtomerkkejä ei välttämättä ole ja niin edelleen. Muuttujien tyyppejä on paljon erilaisia, eivätkä ne kaikki vastaa täysin toisiaan eri ohjelmointikielissä.

Kun luodaan jollakin ohjelmointikielellä tyyppejä, joihin halutaan päästä käsiksi muista ohjelmointikielistä, pitää käyttää vain ominaisuuksia, jotka toimivat myös muissa ohjelmointikielissä. Jotta tämä olisi helpompaa, Microsoft on määritellyt CLR:ään tiedot, joita minimitoiminnallisuuksia pitää kääntäjän tukea. Kääntäjien pitää tukea näitä, jotta generoitua koodia on mahdollista käyttää muista CLS-yhteensopivista ohjelmointikielistä. Kaikessa muille näkyvässä koodissa (joka on määritelty public tai proctected), pitää käyttää vain CLS:ssä määriteltyjä ominaisuuksia. (Richter 2010, 25–26.)

Ohjelmointikieli joka täyttää vain minimi vaatimukset CLS:stä, ei välttämättä pysty käyttämään jonkin toisen .NET-ohjelmointikielen komponentteja, jos niihin on määritelty enemmän ominaisuuksia, mitä CLS:n minimimäärityksessä on (Patrick 2006, 10).

2.4 Common Type System

Common Type System (myöhemmin CTS) kontrolloi ohjelman muuttujien tietoja. CTS määrittelee käytetyt tietotyypit ja mekanismit, joita käytetään .NET ohjelmissa. Tämä koskee kaikkia numeerisia, alphanumeerisia tai boolean tietotyyppejä. Se myös määrittelee olion, jonne tieto tallennetaan. (Patrick 2006, 10.)

CTS jakaa kaikki oliot, jotka sisältävät tietoa, kahteen eri osioon. Ensimmäisessä osiossa on muuttujan tyyppi, toisessa on varsinainen tieto. Ensimmäisessä osiossa on myös tieto, mistä varsinainen sisältö löytyy. (Patrick 2006, 10.)

2.5 .NET luokkakirjasto

.NET:n ydinkomponentit, jotka koostuvat CLR:stä ja siihen liitetystä CTS:stä tarjoavat ohjelmoijalle perusohjelmatoiminnot. Jos ohjelmoija haluaa tehdä jotakin monimutkaisempaa, kuten järjestää listan, .NET kirjastosta löytyy siihen valmis ratkaisu. Ohjelmoijan ei tarvitse tehdä kaikkea itse. (Patrick 2006, 12.)

.NET sisältää kaksi eri kirjastoa: Base Class Library (myöhemmin BCL) ja Framework Class Library (myöhemmin FCL). BCL on pienempi ja sisältää kaikki ne ominaisuudet, joita ilman ohjelmoija ei tule toimeen. Näitä ominai-

suuksia tarvitaan, jotta ne perustoiminnot, joita ohjelmat vaativat, saadaan toimimaan. (Patrick 2006, 12.)

FCL on laajempi. Se sisältää kaikkea muuta, mitä saatatetaan tarvita ohjelmassa. Ne eivät ole kuitenkaan niin tärkeitä, että ne sisältyisivät BCL:ään. Jotta BCL:stä ja FCL:stä voidaan löytää tarvittava kirjasto, .NET käyttää namespaceja. (Patrick 2006, 13–14.)

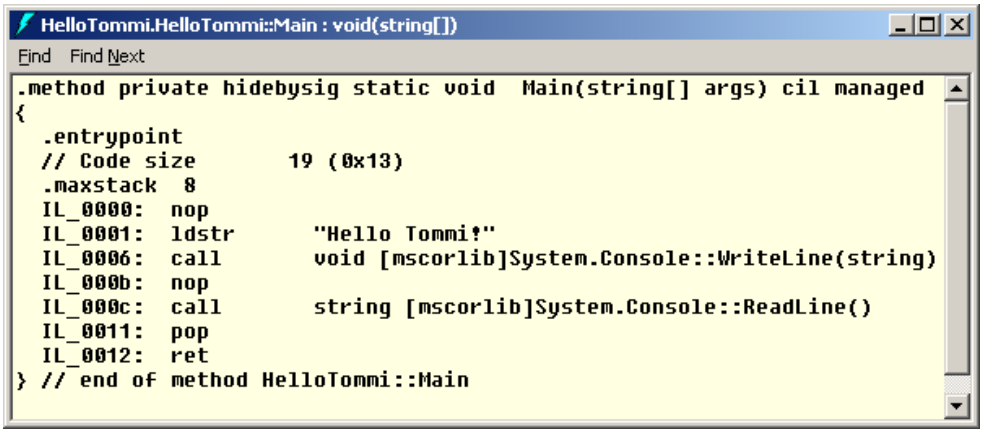
Namespacet kuuluvat kirjaston juureen. Microsoft on varannut omaan käyttöönsä namespacen nimeltä System. Tämän alta löytyvät luokat, jotka käsittelevät Windows-pohjaisia sovelluksia tai websovelluksia. Namespacen alta voi löytyä monta eri luokkaa. Namespacet ja luokat erotellaan pisteillä. Esimerkiksi Windows-lomake on System.Windows.Forms.Form.
(Patrick 2006, 13–14.)

Myös itse tehdyt sovellukset löytyvät hierarkiasta, esimerkiksi jos ohjelman nimi olisi Ohjelma ja sen lomakkeen nimi olisi Lomake1. Hierarkiassa Lomake1:een päästäisiin käsiksi polusta Ohjelma.Lomake1. Ohjelmoija voi luoda oman namespacen, jonka alle tekee ohjelmansa. Jos namespace olisi Yritys, jonka alla ohjelma olisi, tulisi koko poluksi Yritys.Ohjelma.Lomake1. (Patrick 2006, 14.)

2.6 Intermediate Language

Intermediate Language (myöhemmin IL) on käännettyä koodia, joka on voitu kääntää esimerkiksi C#-lähdekoodista. Lähdekoodi on helposti ihmisen luettavissa. Koodista näkee suoraan, mihin luokkaan koodirivi mahdollisesti viittaa.

IL lähdekoodi sisältää ohjeet lataukseen, tallennuksen, kutsumiseen, virheiden käsittelyyn, funktioiden kutsumiseen ja muihin operaatioihin. Näitä ohjeita CLR käyttää hyödyksi, kun se välittää IL-koodia Just In Time (myöhemmin JIT) -kääntäjälle. (MSDN Library 2010c)



Kuva 3 HelloTommi-ohjelman IL-koodi

Kyseisessä esimerkissä ohjelma tulostaa rivin ”Hello Tommi!” konsoliin, ja jää odottamaan käyttäjän enter-painallusta, jonka jälkeen ohjelma päättyy. Main-funktiossa on sana managed, joten koodi on hallinnoitua. Käännös tapahtuu vasta ohjelman ajamisen aikana. Kun ohjelman on IL-muodossa, voi CLR kääntää sen prosessorille sopivaksi konekieleksi.

2.7 Just In Time

Just In Time kääntää IL-koodista ja metadatasta konekielistä ohjelmakoodia ohjelmaa ajettaessa. IL ei ole prosessoririippuvaista kieltä. Konekielinen käännös tietylle prosessorille tehdään vasta ohjelman ajamisen aikana. Näin ollen koodi on optimoitu prosessorille, joka ajaa ohjelman. Normaalin C/C++ ohjelman kääntää yleensä kehittäjä, jolloin se käännetään toimimaan esimerkiksi kaikissa x86-arkkitehtuuriin perustuvissa prosessoreissa. Tällöin konekielinen koodi ei voi käyttää kaikkia nykyaikaisen prosessorin tarjoamia käskykantalaajennuksia, jotka nopeuttaisivat ohjelman toimintaa. (Richter 2010, 9.)

Koska IL:n kääntäminen konekieliseksi tapahtuu, kun ohjelman käynnistetään, ohjelman avautuminen voi kestää hieman pitempään kuin natiivin C/C++-ohjelman. Koko ohjelmaa ei käännetä, vain se, mitä ohjelman käynnistäminen vaatii. Yleensä käynnistyksessä ajetaan Main-funktio (tämän voi myös vaihtaa ohjelman asetuksista). Kun ohjelmasta käytetään kääntämättömiä funktioita, niistäkin tehdään ennen käyttöä konekielinen käännös. Ne ovat tallessa muistissa. Jos funktioon tarvitsee viitata uudestaan, käytetään jo valmista käännettyä funktiota muistista. Käännetty koodi on tallessa vain keskusmuistissa. Jos ohjelma sammutetaan, menetetään myös käännetyt konekieliset koodit. (Richter 2010, 11.)

3 VISUAL BASICIN JA C#:N EROT

Suurimpana erona C#:n ja Visual Basic .NETin (myöhemmin VB.NET) välillä on kielien rakenne. C# pohjautuu C++-kieleen, kun taas Visual Basic perustuu BASIC:iin. Ohjelmointikielten erot näkyvät lähdekoodia tarkastellessa helposti. C# käyttää enemmän symboleita kuvaamaan muuttujien alustuksia ja muita toimenpiteitä. VB.NET käyttää sanoja, koodiriveistä tehdään englanninkielen lauseita.

Kääntäjälle on yhdentekevää, kumpaa ohjelmointikieltä käytetään. Jos kirjoitetaan täysin samalla tavalla toimiva sovellus C# tai VB.NET:llä, kääntäjä kääntää periaatteessa ohjelmista täysin samanlaiset. Tekniseltä kannalta ohjelmointikielellä ei ole siis merkitystä toimittaessa .NET Frameworkissa. Ohjelmointikielien erot näkyvät vain ohjelmoijalle.

Monet ajattelevat, että C# voi tehdä monipuolisempia ohjelmia kuin Visual Basic .NET:llä. C# ja VB.NET ovat kuitenkin tasavertaisia ohjelmointikieliä. Jos jokin ohjelma on tehty C#:lla, sen voi tehdä myös VB.NET:llä. Joitain pieniä eroja ohjelmointikielissä on. VB.NET on tehty ehkä hieman käyttäjäystävällisemmäksi. C#:ssa on joitain ominaisuuksia, jotka puuttuvat VB.NET:stä. C# pystyy käsittelemään umanaged-ohjelmakoodia. (Ji & Gong 2002, 1.)

3.1 Muuttujat

VB.NET:ssä muuttujien nimissä ei ole merkitystä, kirjoitetaanko muuttuja pienellä vai isoilla kirjaimilla. De käsitellään samana muuttujana, kun taas C#:ssa, jos muuttuja on kirjoitettu toisaalla erikokoisilla kirjaimilla, se käsitellään eri muuttujana. Kirjainten koot pitää olla samat kuin alustettaessa muuttujaa. (Ji & Gong 2002, 2.)

Muuttujat alustetaan VB.NETissä eri lailla kuin C#:ssa. VB.NET:ssä määritellään muuttujan nimi ennen tietotyyppiä. Esimerkiksi VB.NET:ssä kokonaislukutyyppinen muuttuja määritellään `Dim i As Interger`, kun taas sama toiminta saadaan C#:ssa `int i`. (Ji & Gong 2002, 2.)

Taulukko 1 Muuttujatyypit Visual Basic .NET:ssä ja C#:ssa

Visual Basic .NET	C#
Boolean	bool
Byte	byte
Short	short
Integer	int
Long	long
Single	float
Double	double
Decimal	decimal
Date	System.DateTime
String	string
Char	char
Object	object

Suurin ero ohjelmointikielien välillä muuttujien alustuksessa on se, että muuttujat VB.NET:ssä kirjoitetaan isolla alkukirjaimella, kun taas C#:ssa kokonaan pienellä. Jos käytetään luokkia, niin käytetään isoa alkukirjainta. (Ji & Gong 2002, 3.)

3.2 Lausekkeet, lohkot ja operaattorit

VB.NET päättää lausekkeet automaattisesti rivinvaihtoihin ja jos riviä halutaan jatkaa, pitää käyttää alaviivaa. Kaksoispisteellä voidaan laittaa samalle riville useampi suoritus. C#:a lauseet erotellaan puolipisteellä. Lausekkeet voivat jatkua monelle riville, tai useampi lauseke voi olla samalla rivillä, kun vain käytetään puolipistettä erottimena. (Ji & Gong 2002, 3.)

C#:ssa lohkot erotellaan {}-merkeillä. VB .NET:ssä on taas avainsanoja, joilla lohkot loppuvat, esimerkiksi:

```
If A = 1 Then
          Console.WriteLine(”A on 1”)
End If
```

Esimerkissä `If` aloittaa lohkon ja `End If` lopettaa lohkon.

Sulkuja käytetään erilailla VB.NET:ssä ja C#:ssa. VB.NET:ssä ei käytetä hakasulkeita taulukoiden indekseihin vaan normaaleja sulkeita. Molemmissa ohjelmointikielissä käytetään aaltosulkeita, kun sijoitetaan tietoa taulukon soluihin, kuten `int [] = new int[3] {1, 2, 3}`. Normaaleja sulkeita käytetään, kun halutaan tuoda funktion sisään syötettävä arvo. Esimerkiksi: `funktio(tietoa).`

3.3 Funktiot

VB.NETissä funktion sisään tulevat muuttujat määritetään `ByRef` ja `ByVal` avainsanoilla. `ByVal` määrittelee, että funktio saa muuttaa muuttujan arvoa. `ByRef`:llä määritelty muuttujaan funktio ei voi tehdä muutoksia. Esimerkiksi:

```
Sub Funktio (ByVal a As String, ByRef b As String)
```

C#:ssa ei tarvitse merkitä, jos haluaa, että funktio pääsee muuttamaan muuttujan arvoa. `ByRef`:ä C#:ssa vastaa ref-avainsana. Samoin kuin VB.NET:ssä tällöin funktio ei voi tehdä muutoksia muuttujan arvoon. Yllä oleva esimerkki C#:lla:

```
void Funktio(string a, ref string b)
```

3.4 Manageroimaton koodi

Manageroimatonta koodia (unmanaged code) voidaan kirjoittaa vain C#:lla. Se on määriteltävä `unsafe`-avainsanalla, siksi ettei ohjelmoija vahingossa tekisi manageroimatonta koodia. Manageroimatonta koodia ei voida käyttää eiluotetussa ympäristössä. Esimerkiksi verkkolevyt eivät ole luotettuja, jollei niitä ole erikseen määritelty .NET Frameworkkiin luotettaviksi. (Ji & Gong 2002, 10.)

Manageroimaton koodi tarkoittaa sellaista koodia, joka ajetaan natiivina suoraan prosessorissa, eikä CLR:n kautta. Kääntäjä ymmärtää avainsanaa käytettäessä kääntää manageroimattoman koodin valmiiksi prosessorin ymmärtämään muotoon.

3.5 Avainsanat ja operaattorit

Avainsanat eroavat C#:n ja VB.NET:n välillä. Esimerkiksi luokkien eri perinnöllisyydet ovat VB.NET:ssä kuvaavampia kuin C#:ssa. C#:ssa luotetaan avainsanoihin, jotka ovat peräisin C:stä ja Javasta. Kaikki avainsanat VB.NET:ssä alkavat isolla alkukirjaimella, C#:ssa kirjoitetaan kaikki pienillä kirjaimilla.

Operaattoreissa on joitakin eroja C#:n ja VB.NET:n välillä. Suurimmat erot ovat loogisissa vertailuissa. Esimerkiksi VB.NET:ssä kirjoitetaan JA -looginen perusfunktio `And`, kun taas C#:ssa se kuvataan merkeillä `&&`. Monia eri operaattoreita kirjoitetaan englanniksi VB.NET:ssä. C#:ssa käytetään lähes kaikissa symboleita. (Ji & Gong 2002, 5.)

4 OHJELMOINNIN PERUSTEET -OPINTOJAKSO

Ohjelmoinnin perusteet -opintojakso on HAMKissa ensimmäisenä vuotena tietojenkäsittelyn koulutusohjelmassa. Opintojakso kuuluu perusopintoihin, joten se on pakollinen opintojakso HAMKin tietojenkäsittelyn opiskelijoille.

Opintojakson tavoite: ”Opiskelija hallitsee lukujärjestelmät ja logiikan alkeet. Opiskelija osaa toteuttaa yksinkertaisen työasemasovelluksen käyttäen perustietotyyppejä, ohjausrakenteita ja ohjelmasilmukoita ohjelman toiminnallisuuden toteuttamisessa. Hän osaa jakaa ohjelman rakenteen aliohjelmiin ja funktioihin. Opiskelija osaa ohjelmoida käyttöliittymäkomponenttien avulla lomakepohjaisen käyttöliittymän.” (SoleOPS 2008)

Opintojaksossa pääaiheina ovat C#-ohjelmointi, tietokannat ja bittimuunnokset. Suurin osa lähiopetuksesta käytetään C#-ohjelmointiin (noin 20 tuntia). Tietokantoihin käytetään 4 lähiopetustuntia ja 6 tuntia bittimuunnoksiin. 18 tuntia on varattu työpajatunneiksi. Näitä lähitunteja käytetään harjoitustyön tekemiseen. Nämä tunnit ovat ohjattua työntekoa. Itse harjoitustyön tekemiseen voi mennä kauemminkin. Työpajatunneilla opiskelijat voivat kysyä opettajalta, jos jotain ongelmia ilmentyy harjoitustyön tekemisessä.

Ohjelmoinnin perusteet -opintojaksolla ohjelmoinnin opetuksessa ei keskitytä ohjelmointikieleen. Opintojaksolla käytettävät ehtolauseet ja funktiot löytyvät monista eri ohjelmointikielistä. Kurssin tarkoituksena on saada opiskelijoille perustietämys ohjelmoinnista.

Ohjelmoinnin opetus aloitetaan kysymällä opiskelijoilta, mitä he jo tietävät ohjelmoinnista. Näillä kysymyksillä herätetään opiskelijat ajattelemaan, mitä he tietävät tai eivät tiedä ohjelmoinnista. Kysymykset liittyvät ohjelma ja ohjelmointi -käsitteisiin. Kysymyksissä ei käsitellä varsinaisesti mitään ohjelmointikieltä tai ohjelman lauserakenteita. (Moodle 2010)

Opetus on jaettu kahteen osaan: konsolisovelluksiin ja työpöytäsovelluksiin, vaikka konsolisovelluksissa opittuja asioita voi hyödyntää myös työpöytäsovelluksissa.

4.1 Konsolisovellukset

Ennen varsinaisten konsolisovellusten ohjelmointia opiskelijat pääsevät tutustumaan Visual Studio 2010:iin. Opetuksessa käydään läpi perustoiminnot Visual Studiossa sekä, miten luodaan uusi projekti ja tallennetaan se. Opiskelijoita suositellaan käyttämään omaa verkkolevyään tehtävien talletukseen. Ensimmäisessä sovelluksessa tulostetaan oma nimi.

Konsolisovellusvaiheessa aiheena ovat:

- konsoliin tulostus
- käyttäjän syötteen lukeminen
- ehtolauseet
- toistorakenteet
- tekstitiedoston lukeminen
- tekstitiedostoon tallentaminen
- merkkijonojen muotoilu
- virhetilanteista toipuminen
- taulukot
- ohjelman parametrit
- ohjelman kääntäminen.

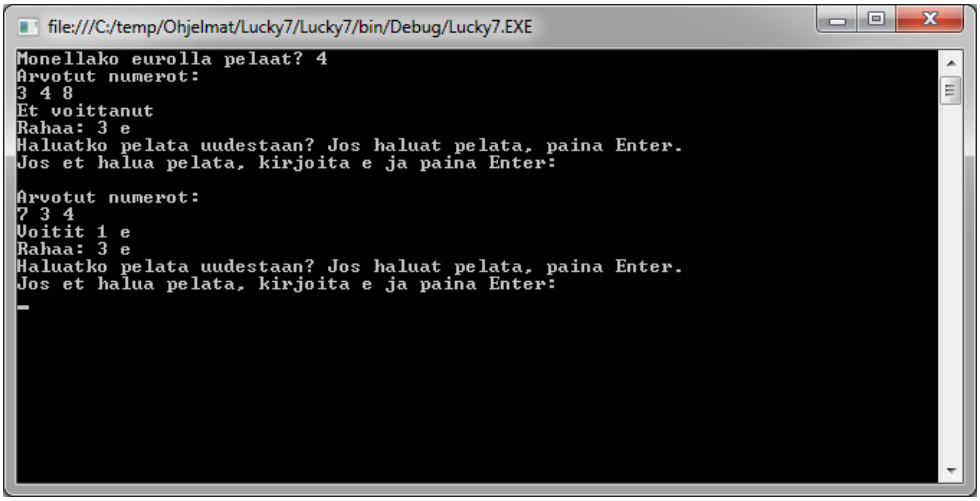
4.1.1 LuckySeven-harjoitustehtävä

Opiskelijoita yritetään orientoida ohjelmointiin pelillä. Pelissä on tarkoitus arpoa kolme numeroa välillä 1-10. Jos yksikin numero on 7, pelaaja voittaa. Ohjelmassa luodaan uusi olio `random = new Random()`. jota käytetään satunnaisnumeroiden luomisessa. Tässä vaiheessa ei kuitenkaan keskitytä olioihin. Tehtävä tehdään monessa eri vaiheessa, ensimmäisessä vaiheessa voidaan pelata vain yksi kierros, jonka jälkeen ohjelma sammuu. Harjoitustehtävän seuraavat vaiheet kehotetaan tekemään vasta, kun toistorakenteet on käsitelty.

Seuraavassa vaiheessa käyttäjällä on mahdollisuus määritellä, kuinka monella eurolla he pelaavat. Käyttäjä syöttää numeron, millä summalla hän haluaa pelata. Yhdellä eurolla voi pelata yhden kierroksen. Yhdellä kierroksella voi voittaa seitsemällä yhden euron. Myös muiden numeroiden tulos otetaan

huomioon. Jokaisesta kierroksesta voi siis voittaa 0 – 3 euroa. Voitot kartuttavat rahoja, jolloin pelivuoroja saa lisää. Ohjelma loppuu, kun pelivuorot loppuvat.

Kolmannessa vaiheessa peliin lisätään vaihe, joka kysyy käyttäjältä, haluaako hän jatkaa pelaamista. Käyttäjä jatkaa pelaamista painamalla enteriä tai poistuu sovelluksesta kirjoittamalla e ja painamalla enter-näppäintä.



Kuva 4 Lucky7-harjoitustehtävä

Esimerkkikuvassa on käyttäjä antanut pelattavaksi summaksi neljä euroa. Käyttäjä on pelannut kaksi vuoroa. Ensimmäisellä vuorolla käyttäjä ei voittanut mitään, mutta toisella voitti yhden euron, joka kasvatti pelirahoja.

4.1.2 Ehtolauseet

Seuraavaksi opintojaksolla opiskellaan ehtolauseet. Ehtolauseet opiskelijoiden pitää osata hyvin, niitä joutuu käyttämään monentasoisessa ohjelmoinnissa. Jos ehtolauseet on oppinut hyvin jo aikaisessa vaiheessa, on siitä hyötyä myöhemmin, kun ehtolauseet monimutkaistuvat ja mukaan tuodaan toistorakenteita.

Ehtolauseiden peruskäyttöä opiskellaan tekemällä ohjelma, joka kertoo iän perusteella, mitä mahdollisuuksia kyseessä olevalla henkilöllä on. Ikä määritellään valmiiksi ohjelmaan. `if`-lauseella tarkistetaan, onko muuttuja yli 18, joka on tässä tapauksessa ikä. Jos ikä on 18 tai yli, näytetään teksti, joka kertoo henkilön olevan aikuinen. Ohjelmaa laajennetaan niin, että se osaa kertoa, mitä ajoneuvoa voi ajaa, onnitella tasavuosikymmeninä ynnä muuta. Tarkoituksena on että opiskelijat ymmärtävät, miten `if`-lauseet toimivat ja kuinka niitä käytetään eri tilanteissa. Erilaisia ehtorakenteita harjoitteessa on paljon, jotta `if`-lauseiden käyttö tulisi tutuksi.

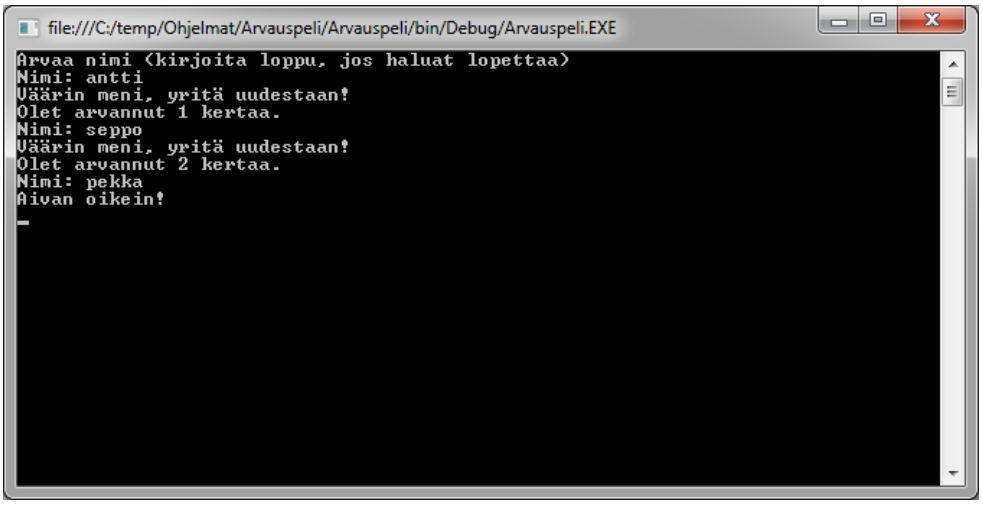
4.1.3 Toistorakenteet

Opetukseen kuuluu myös `for`, `foreach`, `while` ja `do-while` -toistolauseet. Opetuksessa käytetään esimerkiksi listan solujen lukemiseen mielummin while tai for -lausetta, jotta opiskelija ymmärtää, mitä ohjelmassa tapahtuu. Helpommin ja nopeammin ohjelman voi tehdä myös `foreach`-toistolauseella. Molemmilla toistolauseilla saadaan samanlainen toiminta. For-lausekkeessa opiskelija joutuu määrittämään muuttujan alustuksen, seuraavaan taulukon soluun siirtymisen ja ehdon, milloin toistolausekkeesta poistutaan.

Tarkoituksena on saada opiskelijalle käsitys, mihin toistolausekkeita voidaan käyttää helpottamaan ohjelmointi sekä missä tilanteissa on pakko käyttää toistorakenteita. Ohjelmoinnista on vältettävä turhaa asioiden toistoa. Tosin sitä tulee jossain määrin opiskelussa, koska esimerkkejä halutaan yksinkertaistaa. Opiskelija voikin miettiä, miten erilaisia esimerkkejä voisi parantaa.

Harjoituksissa tulostetaan moneen kertaan omaa nimeä, lisäksi arvotaan moneen otteeseen numeroita toistolausekkeiden avulla. Lisäksi tulostetaan kaikki numerot käyttäjän syöttämien lukujen väliltä.

Erillisessä tehtävässä käyttäjän pitää arvata nimeä. Nimi on aikaisemmin määritelty ohjelmaan. Ohjelma ajaa toistorakennetta niin kauan, kunnes arvaus osuu oikeaan. Arvaus luetaan käyttäjän syötteestä, jos käyttäjä arvaa oikein sovellus sulkeutuu ja käyttäjää onnitellaan. Käyttäjällä on mahdollisuus myös poistua sovelluksesta antamalla arvaukseksi ”loppu”.



Kuva 5 Nimenarvaustehtävä

Esimerkkiratkaisussa käyttäjä on arvannut nimen kaksi kertaa väärin, mutta arvaa nimen oikein kolmannella kerralla. Laskuri näyttää arvattuja kertoja, ennen kuin käyttäjä saa arvata seuraavan kerran.

4.1.4 Tekstitiedoston lukeminen ja kirjoittaminen

Osana konsolisovellusten opetusta on tekstitiedostoon kirjoittaminen ja tekstitiedoston sisällön lukeminen. Harjoitteissa kirjoitetaan tiedostoon tai luetaan tiedostosta useampi rivi tekstiä. Toistolausekkeen käyttäminen tiedostonlukemiseen on melkeinpä pakollinen, jos halutaan lukea kaikki tiedoston rivit, varsinkin jos ohjelmoija ei tiedä, kuinka monta riviä tiedostossa tulee olemaan.

Ennen kuin voidaan käyttää tiedoston luku- tai kirjoitusluokkia, pitää määritellä ohjelma käyttämään IO-luokkaa, se tapahtuu määrittelemällä ohjelman alkuun `using System.IO;`. Yksinkertaisimmillaan tiedostoa voidaan lukea yksi rivi ja tulostaa se konsoliin:

```
string tiedostonNimi  = "C:/temp/tervehdys.txt";
TextReader tr = new StreamReader(tiedostonNimi);
Console.WriteLine(tr.ReadLine());
tr.Close();
```

Ohjeissa neuvotaan, miten on mahdollista saada luettua tekstitiedostosta esimerkiksi `double`-tyyppinen muuttuja. Kun muuttuja luetaan suoraan tekstitiedosta, se käsitellään merkkijonona, joten se joudutaan jäsentämään `Double.Parse()`-funktion kautta. Kirjoittaminen menee helpommin, koska C# vaihtaa muuttujan tyypiksi `string` automaattisesti.

```
string tiedostonNimi = "C:/temp/hinta.txt";
double hinta;
TextReader tr = new StreamReader(tiedostonNimi);
hinta = Double.Parse(tr.ReadLine());
```

4.1.5 Merkkijonojen muotoilu

Tämän jälkeen opetellaan, miten merkkijonoja voi muotoilla. Moodlessa on taulukko, missä on kerrottuna, miten merkkijonoja muotoillaan. Esimerkeissä kerrotaan myös, miten `string`-tyyppinen tieto saadaan numeeriseksi esimerkiksi `double`-muuttujaan. Muuttujien tietotyyppien vaihtamisessa käytetään tietotyyppiluokkien `Parse()`-funktioita. Esimerkiksi merkkijonon saa kokonaisluvuksi seuraavalla lauseella `int luku = Int32.Parse(merkkijono)`. Myös `Convert`-luokka käy muuttamiseen, sama toiminta `Convert`-luokalla: `int luku = Convert.ToInt32(merkkijono)`.

4.1.6 Ohjelman virhetilanteet

Opiskelijoille on jo aikaisemmin puhuttu ohjelman erilaisista virhetilanteista. Ohjelmassa voi olla syntaksivirheitä, loogisia virheitä ja ajonaikaisia virheitä. Try-Catch -lauserakenteella yritetään estää ajonaikaisia virheitä. Sellainen voi olla esimerkiksi tapaus, jossa käyttäjä kirjoittaa numerokenttään kirjaimia. Kääntäjä huomaa poikkeustilanteen, joka on tässä tapauksessa FormatExcep-

tion. Ohjelma ei saa muunnettua kirjaimia numeeriseksi ja kaatuu. Tätä voidaan estää laittamalla muuttujaan sijoitus `try{}`-lohkon sisään. Jos `try`-lohkossa tapahtuu poikkeustilanne (exception), ohjelma suorittaa `try`-lohkon jälkeisin `catch`-rakenteen.

Esimerkissä näytetään, miten voidaan ottaa virhe kiinni, jos käyttäjä yrittää tallentaa int-muuttujaan arvon, joka ei ole kokonaisluku. Ohjelma ottaa kiinni kaikki `FormatException`-poikkeukset. Poikkeustilanne voi tulla `try`-lohkossa, kun `Int16.Parse()`-funktio yrittää muuttaa merkkijonosta 16-bittistä kokonaislukua.

```
int numero = 0;
Console.WriteLine("Kirjoita numero: ");
try
{
    // Tämä voi aiheuttaa FormatException-virheen
    numero = Int16.Parse(Console.ReadLine());
    // Tätä kohtaa ei suoriteta, jos käyttäjä ei anna numeroa
    // -> hypätään Catchiin
    Console.WriteLine("Kirjoitit " + numero);
}
catch (FormatException e)
{
    Console.WriteLine("Et kirjoittanut numeroa.");
}
```

4.1.7 Taulukot

Seuraavaksi opetetaan, kuinka taulukot toimivat C#:ssa. Ensimmäiseksi opiskelijoille kerrotaan pseudokoodilla, mikä on taulukkojen idea ja miten tieto varastoidaan taulukkoihin. Taulukon indeksin idea pitäisi tulla tutuksi, jotta sitä osattaisiin käyttää varsinaisessa ohjelmoinnissa. Opiskelijoille pitäisi tulla käsitys, miksi taulukkoja kannattaa käyttää ohjelmoinnissa. Taulukot ovat hyödyllisiä, jos samanlaista tietoa halutaan tallentaa useampi kappale. Esimerkiksi, jos haluttaisiin tallentaa autoja, niissä olisi monia samanlaisia tietoja. Taulukkoja on helpompi käsitellä kuin montaa samantyyppistä muuttujaa.

Opiskelijoille opetetaan, kuinka he voivat alustaa taulukon ja kirjoittaa taulukon yhteen soluun tietoa. Tututuksi tulee myös, kuinka taulukon solu saadaan myöhemmin luettua taulukosta. Lisäksi taulukon tietoa käydään läpi foreach ja `for`-loopeilla. `foreach`-looppia on helppo käyttää, mutta `for`-looppi selittää ehkä paremmin, miten taulukko käydään läpi. Lisäksi tutustutaan nopeasti, miten taulukon solut voi järjestää esimerkiksi aakkosittain.

Kun normaalit taulukot ovat tulleet tutuksi, opiskelijoille esitellään kaksiulotteiset taulukot. Kaksiulotteisissa taulukoissa voi olla useampi solu samalla rivillä. Kaksiulotteisen taulukon alustaminen on hieman erilainen kuin yksiulotteisen, pilkulla erotellaan eri ulottuvuudet. Samaa tapaa joutuu käyttämään aina, kun halutaan lukea tai käsitellä moniulotteisen taulukon solua. Opiskeli-

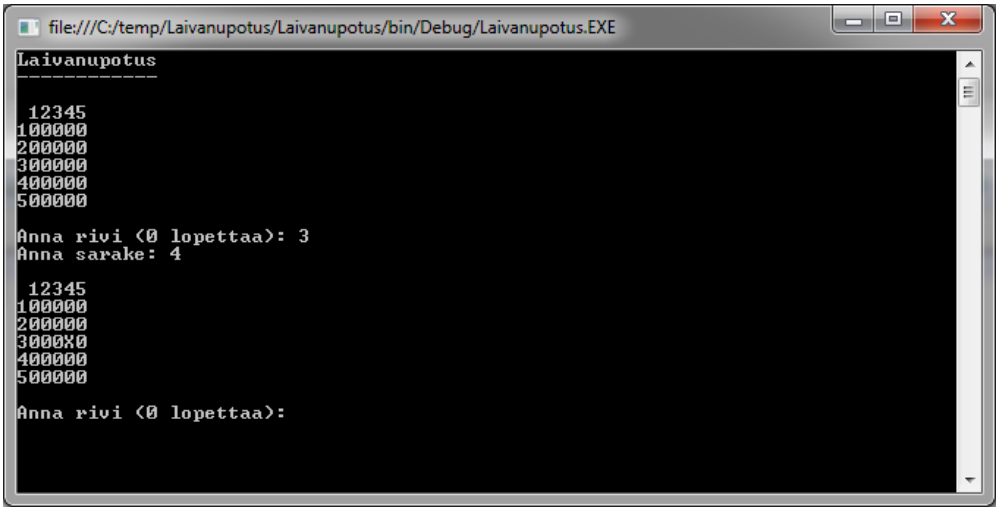
joille kerrotaan, miten on mahdollista sijoittaa tietoa kaksiulotteiseen taulukkoon. Lisäksi opiskellaan, miten taulukon voi lukea `for`-loopilla. Kaksiulotteisessa taulukossa joudutaan käyttämään kahta `for`-looppia. Ulompi `for`-looppi lukee rivejä, sisäkkäinen taas rivin soluja. Saman voi tehdä myös yhdellä `foreach`-loopilla, mutta siitä kaksiulotteisen taulukon idea ei käy niin hyvin selville.

```
for (int i = 0; i < 2; i++)
    for (int j = 0; j < 3; j++)
           Console.WriteLine("rivi: " + i + " sarake: " + j +
           " arvo: " + numerot[i,j]);
```

Taulukkojen käyttöä harjoitellaan tallentamalla yksiulotteiseen taulukkoon huonekalujen nimiä. Taulukon tiedot tulostetaan. Ohjelman tulostusta muokataan vaativammaksi eri vaiheissa. Ensin tulostetaan koko taulukko, sitten kaksi ensimmäistä taulukon arvoa. Lopuksi etsitään merkkijonoa ”sohva”, ja tulostetaan jos se löytyy taulukosta.

Toinen harjoitus taulukoilla on lottonumeroiden arvonta. Numerot arvotaan välillä 1—35 ja sijoitetaan yksiulotteiseen taulukkoon, jossa on soluja seitsemän kappaletta. Ohjelma myös tarkistaa, ettei samaa lukua tule kahteen kertaan taulukon sisään. Arvonnan jälkeen tulostetaan kaikki arvotutut numerot taulukosta.

Kaksiulotteisia taulukkoja harjoitellaan laivanupotuspelillä. Laivoja ei pääse upottamaan, ohjelma näyttää vain, mihin kohtaan ammukset menevät. Käyttäjä valitsee rivin ja sarakkeen, mihin hän haluaa ampua. Rivejä ja sarakkeita on viisi, joten osumapaikkoja tulee 25. Ohjelman voi lopettaa antamalla riviksi 0. Ohjelmassa näkyy merkkeinä, mihin ammus on osunut. X on merkkinä paikasta, mihin on ammuttu. 0 merkitsee kohtaa, mihin ei ole osunut ammusta.



Kuva 6 Laivanupotustehtävä

Esimerkkikuvassa ohjelma on käynnistetty. Käyttäjä on valinnut riviksi kolme ja sarakkeeksi neljä. Ohjelma näyttää taulukossa, mihin kohtaan taulukkoa on ammuttu. Tämän jälkeen käyttäjää pyydetään syöttämään uusi rivi ja sarake.

4.1.8 Ohjelman parametrit

Ohjelman parametrit käydään myös konsolisovelluksissa läpi. Esimerkkinä ohjelmalle voidaan antaa parametrina ”-h”, joka näyttää sovelluksen ohjeen. Oletuksena C#-konsolisovellukseen tulee `args[] string`-tyyppinen taulukko, mihin konsoliparametrit tulevat. Indeksiin numero 0 tulee ensimmäinen parametri, indeksiin 1 toinen parametri ja niin edelleen. Parametrit erotellaan ohjelmaa ajattaessa välilyönneillä. Opiskelijoille annetaan esimerkki, miten voidaan lukea kaikki parametrit, ja tulostaa ne.

Jotta konsoliparametreja päästään kunnolla testaamaan, pitää sovelluksesta kääntää IL-versio. Käännettyä ohjelmaan voidaan ajaa komentokehotteesta käsin. Harjoitteessa pyydetään avaamaan komentokehotteesta käsin ohjelman projektihakemisto, ja sieltä hakemistoon bin\debug. Siellä ohjelma pitää ajaa ja kirjoittaa sille parametreja. Parametrien pitäisi tulostua komentokehotteeseen.

Konsoliparametreja käytetään hyväksi seuraavassa harjoituksessa. Harjoitteessa annetaan ohjelmalla kaksi lukua ja laskutoimenpiteen symboli. Esimerkiksi ”Laskin 5 + 5”. Ohjelman pitää tulostaa annetut luvut ja tulos, kyseisessä esimerkissä siis ”5 + 5 = 10”. Laskimen tulee osata plus-, miinus-, kerto- ja jakolaskut.

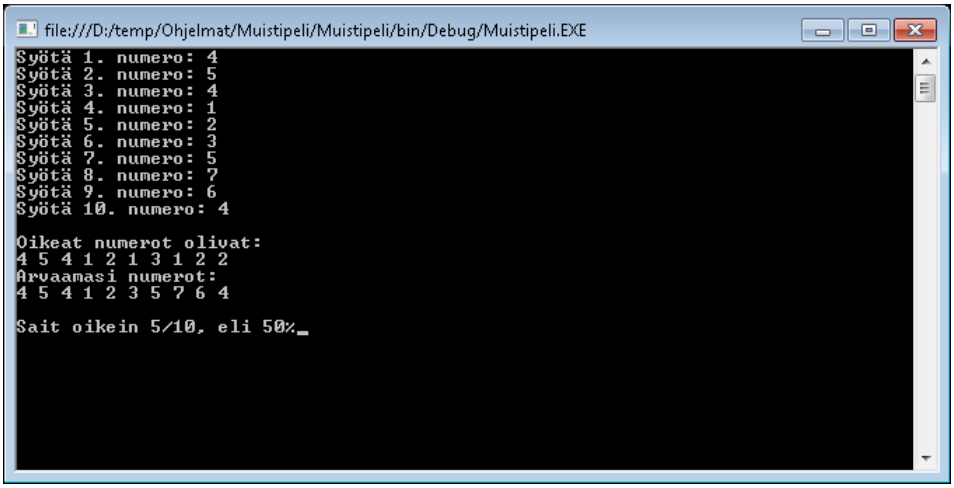
4.1.9 Ohjelman kääntäminen

Opiskelijat voivat tutusta materiaaliin, missä käsitellään C#-ohjelman kääntämistä. Asia ei varsinaisesti kuulu opetukseen, mutta siitä on dokumentti. Siinä kerrotaan, kuinka C#-lähdekoodista muodostuu IL-koodia ja kuinka IL-koodi käännetään konekieleksi ennen ohjelman ajamista. Ohjeessa neuvotaan, kuinka opiskelija voi avata kääntämänsä ohjelman IL Dasm -ohjelmalla. Opiskelijan tulisi huomata, kuinka helposti ohjelman IL-koodia pääsee kääntämään takaisin lähdekoodiksi. Ohjelman suojaamiseksi voi käyttää obfuscatoria, joka sekoittaa ohjelman käännettyä koodia vaikeammin tulkattavaksi.

4.1.10 Muistipeliharjoitus

Seuraavana tehtävänä on muistipeli, jossa on tarkoitus muistaa 10 numeroa, jotka voivat olla väliltä 1—5. Ohjelma näyttää numerot käyttäjälle viiden sekunnin ajan. Tämän jälkeen käyttäjän pitäisi syöttää näytetyt numerot yksi kerrallaan. Lopuksi ohjelma kertoo oikeat numerot ja arvatut numerot. Opiskelija vois myös laittaa ohjelmaan lisätoiminnon, joka kertoo kuinka monta

numeroa meni väärin ja kuinka monta prosenttia oli oikein. Ohjelman ohjeessa kerrotaan, miten on mahdollista tyhjentää konsolia ja jättää ohjelma odottamaan tietyksi aikaa. `Threading`-luokasta löytyy `Sleep()`-funktio, jolla voidaan määrittää millisekunteina, kuinka kauan ohjelma odottaa tekemättä mitään.

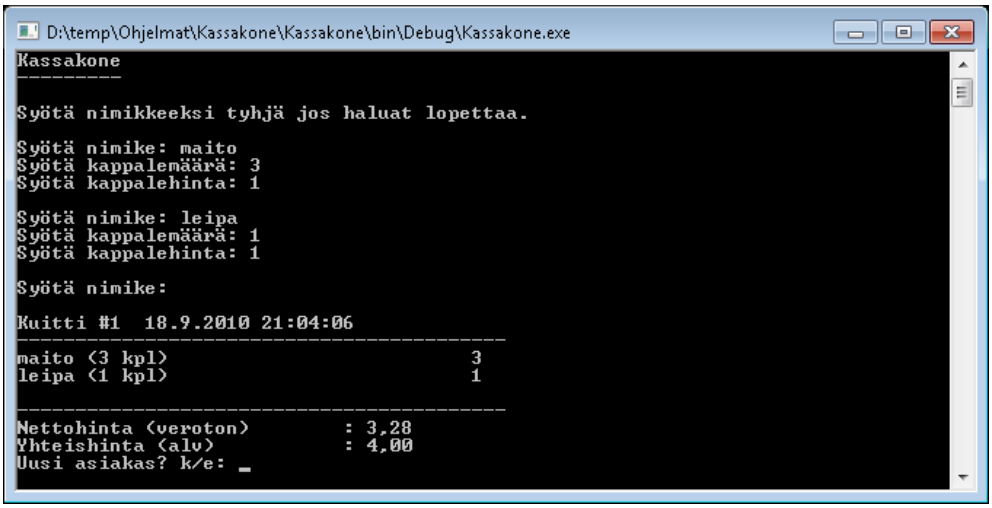


Kuva 7 Muistipelin esimerkki

Esimerkkikuvassa käyttäjä on muistanut puolet luvuista oikein. Ohjelma on aikaisemmin näyttänyt lukuja viisi sekuntia käyttäjälle.

4.1.11 Kassakoneharjoitus

Paljon tähän asti opittua hyödyntää kassakone-harjoitus. Palautettavassa harjoituksessa on tarkoitus tehdä kassakoneen tyyppinen ohjelma. Ohjelma tehdään monessa osassa. Ohjelma kysyy nimikettä, sen hintaa ja kuinka monta niitä ostetaan. Ostoksista tulostetaan kuitti tiedostoon. Tiedostossa on ostosten lisäksi loppusumma ja arvonlisävero eroteltuna. Kuittiin tulostetaan myös päivämäärä ja kellonaika. Ohjelmassa tulee olla virheentarkistus. Käyttäjä voi syöttää lukumääräksi esimerkiksi kirjaimia, eikä ohjelma kaadu. Ohjelmassa voi olla monta asiakasta peräkkäin. Asiakkaiden kuitit tulostetaan eri tekstitiedostoihin.



Kuva 8 Kassakone-esimerkki

Esimerkkiohjelmassa käyttäjä on ostanut kolme maitoa ja yhden leivän. Käyttäjä on määritellyt hinnoiksi yhden euron. Kokonaissummaksi tulee näin ollen neljä euroa. Ohjelma näyttää myös verottoman summan. Asiakastapahtuma on päättynyt, ohjelma odottaa vastausta, halutaanko ohjelmaan syöttää uuden asiakkaan ostokset.

4.2 Työpöytäsovellukset

Ennen kuin opiskelijat siirtyvät konsolisovelluksista työpöytäsovelluksiin, he voivat tehdä testin, joka testaa, kuinka hyvin he ovat oppineet kurssin tähän mennessä käsitellyt asiat. Ohjelmassa käyttäjää pyydetään syöttämään oma nimensä, lisäksi kysytään, kuinka monta kertaa nimi tulostetaan. Jos käyttäjä ei antanut nimeänsä, tulostuu teksti ”et antanut nimeäsi” ja ohjelma loppuu. Tulostus täytyy tehdä toistorakennetta käyttäen. Harjoitetta voidaan helpottaa siten, että käyttäjältä ei kysytä, kuinkamonta kertaa nimi tulostetaan, vaan nimi tulostetaan niin monta kertaa, kun on ennalta määrätty ohjelman lähdekoodissa.

Ensimmäinen työpöytäsovellus kurssilla on kuvagalleria. Se tehdään yhdessä tunnilla. Sovelluksessa on `PictureBox`, johon on laitettu kuva. Kuvaa voi vaihtaa neljäksi erilaiseksi neljällä eri painonapilla. Ensimmäisessä vaiheessa sovelluksessa on neljä eri `PictureBox`ia. Jokaisen `PictureBox`in näyttämiseen on oma nappinsa. Kun käyttäjä painaa nappia, kolme muuta `PictureBox`ia piilotetaan, ja yksi näytetään. Opiskelijan tehtäväksi jää miettiä, miten sovellusta voisi parantaa. Sovelluksen parantaminen onnistuu esimerkiksi tekemällä yhden `PictureBox`in ja muuttamalla sen parametria, mihin kuva määritetään, aina kun käyttäjä painaa painonappia.

Harjoituksen tarkoituksena on saada opiskelijoille kokemusta Visual Studion WYSIWYG-editorista, sekä yksinkertaisesta ohjelmarakenteesta. Myös pai-

nonappien tapahtumat (`Events`) ovat tärkeässä osassa ohjelmassa. Opiskelijan pitää kirjoittaa ohjelmakoodi painonappien Push-tapahtumiin.

Kun ensimmäinen työpöytäsovellus on opetettu, opetetaan opiskelijoille miten pääohjelmat ja aliohjelmat toimivat ja miten he voivat hyödyntää niitä ohjelmissaan. Opiskelijoille kerrotaan, miksi on hyvä käyttää aliohjelmia. Suurin syy aliohjelmien käyttöön on koodin helppo päivitettävyys. Jos samoja ohjaamakoodeja kopioidaan eri paikkoihin, päivitettävyys kärsii. Jos ohjelmakoodissa on jokin virhe, joudutaan se korjaamaan moneen eri paikkaan.

Aliohjelmiin tutustutaan ensi pseudo-ohjelmoinnilla. Kun opiskelija ymmärtää, miten ohjelma hyppisi pääohjelmien ja aliohjelmien välillä pseudo-ohjelmoinnissa siirrytään C#:iin. Opiskelijoille opetetaan, että he ovat todennäköisesti käyttäneet jo C#:n tarjoamia valmiita aliohjelmia. Esimerkiksi `MessageBox` on valmis aliohjelma. `MessageBox`illa voi tulostaa käyttäjälle viesti-ikkunoita.

Funktiota käytetään yksinkertaisessa ohjelmassa, missä on tekstikenttä sekä painonappi. Tekstikenttään kirjoitetaan salasana ja painonapista voidaan tarkistaa onko salasana oikein. Painonapin `button1_Click(object sender, EventArgs e)` -funktiossa käytetään salasanan tarkistusfunktiota. Jos funktio palauttaa arvon `true`, kerrotaan käyttäjälle että salasana on oikein, muissa tapauksissa se on väärin. Ohjelman lähdekoodi:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    bool tarkistasalasana(string salasana)
    {
        if (salasana == "qwerty")
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    private void button1_Click(object sender, EventArgs e)
    {
        if (tarkistasalasana(textBox1.Text))
            MessageBox.Show("Salasana oikein!");
        else
            MessageBox.Show("Salasana väärin!");
    }
}
```

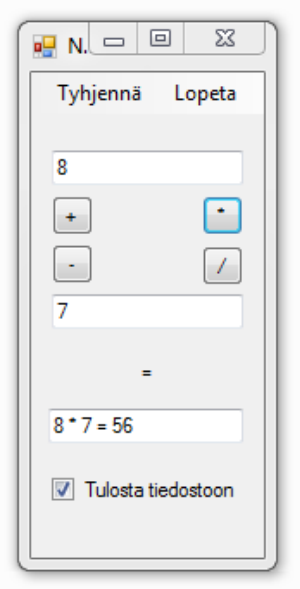
Toisella esimerkillä näytetään, miten muuttujat näkyvät vain aliohjelmien sisällä. Käytäntö lisää ohjelman tietoturvaa, sekä pienentää riskiä, että ohjelmassa on päällekkäin nimettyjä muuttujia. `tulostaTeksti()`-funktion perusmuuttuja näkyy vain funktion sisällä. Luokan sisälle määritelty `zuper`-muuttuja näkyy taas kaikkialla luokan sisällä. Ohjelman lähdekoodi:

```
public partial class Form1 : Form
{
    string zuper = "Zuperteksti";

    public Form1()
    {
        InitializeComponent();
    }
    void tulostaTeksti()
    {
        string perus = "terve";
        MessageBox.Show(perus);
        MessageBox.Show(zuper);
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        tulostaTeksti();
    }
}
```

Seuraavassa työpöytäsovellusharjoitteessa pitää tehdä nelilaskin. Laskimella voidaan laskea yhteen-, miinus-, kerto- ja jakolaskuja. Laskimessa on kaksi tekstikenttää, joihin voidaan kirjoittaa laskettavat luvut. Lisäksi sovelluksessa on yksi tekstikenttä tulokselle. Tuloskentässä tulee näkyä myös koko laskutoimitus. Sovelluksessa tulee olla valikko, mistä ohjelman kentät voi tyhjentää tai sulkea sovelluksen. Ohjelmaa sulkiessa sovelluksen pitää kysyä käyttäjältä varmistus. Sovelluksessa on hyödynnettävä aliohjelmaa. Aliohjelma kirjoittaa tiedostoon laskun tuloksen aina, kun painonappia painetaan. Samasta painonapista ohjelma näyttää lasketun tuloksen myös tekstikentässä. Ohjelmassa on myös valintaruutu, josta voidaan valita, kirjoittaako ohjelma tiedostoon.



Kuva 9 Nelilaskimen esimerkki

Esimerkissä käyttäjä on laskenut numeroiden kahdeksan ja seitsemän kertolaskun. Tulokseksi on tullut 56. Ohjelma on myös tallettanut laskutoimituksen ja tuloksen C:\laskin.txt -tiedostoon.

Seuraavaksi opintojaksolla opetetaan, miten yhdestä C#-ohjelmasta voidaan avata tai sulkea monia eri lomakkeita. Normaalisti, kun luo tyhjän C#-projektin Visual Studio luo yhden valmiin lomakkeen. Monimutkaisemmissa ohjelmissa halutaan kuitenkin käyttää monia lomakkeita. Monien lomakkeiden käyttö voi myös selkeyttää käyttöliittymää. Käyttöliittymän selkeyttämiseksi voidaan myös käyttää välilehtiä ynnä muuta.

Ensimmäiseksi opastetaan, miten Visual Studiossa voi tehdä uuden Windows Formin eli uuden lomakkeen. Tämän jälkeen lomake tulee avata vielä jo avatussa lomakkeessa (niin sanotussa päälomakkeessa). On myös mahdollista avata monta lomaketta, kun ohjelma käynnistetään, mutta sitä vaihtoehtoa ei käydä opetuksessa läpi. Tässä vaiheessa olio on määritelty vain funktion sisäiseksi (napin tapahtuma, jota painamalla lomake aukeaa).

```
private void buttonAvaaLomake2_Click(object sender,
EventArgs e)
{
    Lomake2 lomake2 = new Lomake2();
    lomake2.Show();
}
```

Koska monesti lomake halutaan sulkea ja avata uudestaan, opiskelijoille opetetaan, miten he voivat laittaa luokan muuttujaan olion talteen. Sieltä se voidaan ottaa käyttöön alilomakkeen sammutuksen jälkeenkin. Lomake voidaan joko vain piilottaa tai sitten hävittää kokonaan. Jos lomake on hävitetty, siihen

syötetyt tallentamattomat tiedot eivät jää talteen. Päälomakkeen luokkaan täytyy tällöin määritellä olio: `private Aliformi aliformi;`. Lomake voidaan ladata jo valmiiksi muistiin päälomakkeen latausfunktiossa

```
aliformi = new Aliformi();
```

Kun lomake halutaan näyttää, kutsutaan sen show()-funktiota

```
aliformi.Show();
```

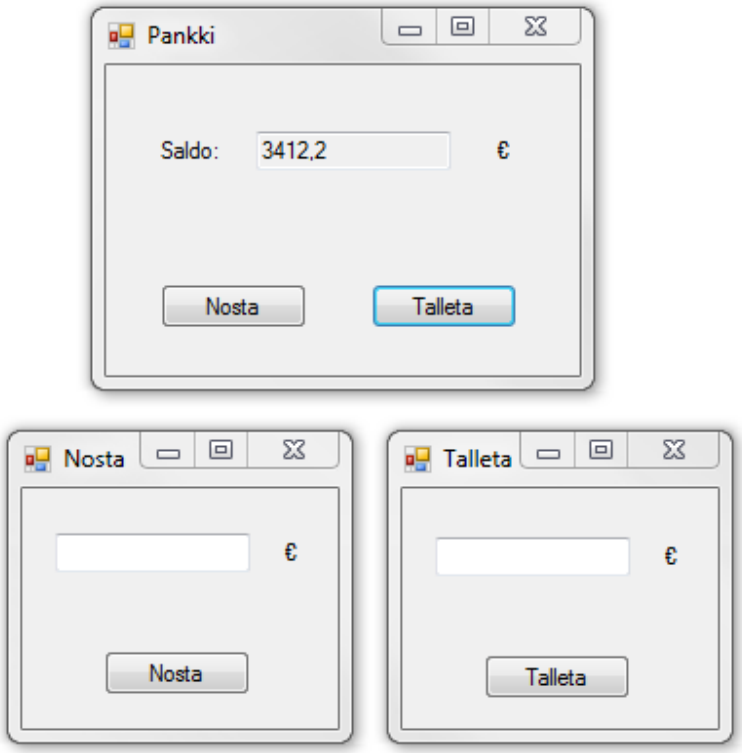
Kun alilomake suljetaan ja se halutaan avata uudestaan, tulee ongelma. Olio on kadonnut, koska se on hävitetty. Ongelman voi korjata monella tapaa. Yksi vaihtoehto on tehdä alilomakkeeseen `FormClosing`-tapahtuma, jossa ei hävitetä lomaketta, vain piilotetaan.

```
private void Aliformi_FormClosing(object sender,
FormClosingEventArgs e)
{
    this.Hide();
    e.Cancel = true;
}
```

Toinen mahdollisuus on tarkistaa päälomakkeesta, onko alilomake hävitetty ennen kuin sitä yritetään avata. Jos lomake on jo hävitetty, luodaan uusi. Tarkistus onnistuu helposti luokan sisäisellä funktiolla.

```
if (aliformi.IsDisposed)
    aliformi = new Aliformi();
```

Lomakkeiden käsittelyyn liittyvää olio-ohjelmointiteoriaa ei käsitellä kurssilla. Kurssimateriaalissa on kuitenkin pieni opiskelumateriaali olio-ohjelmoinnista. Materiaalissa käsitellään olio ja luokka käsitteitä, luokan muuttujia sekä näkyvyyttä. Kurssilla ei ole tarkoituksena opiskella olio-ohjelmointia, mutta jos opiskelija haluaa, hän voi tutustua olio-ohjelmointiin. Olio-ohjelmointia käydään läpi vasta Ohjelmointi 1 -opintojaksolla.



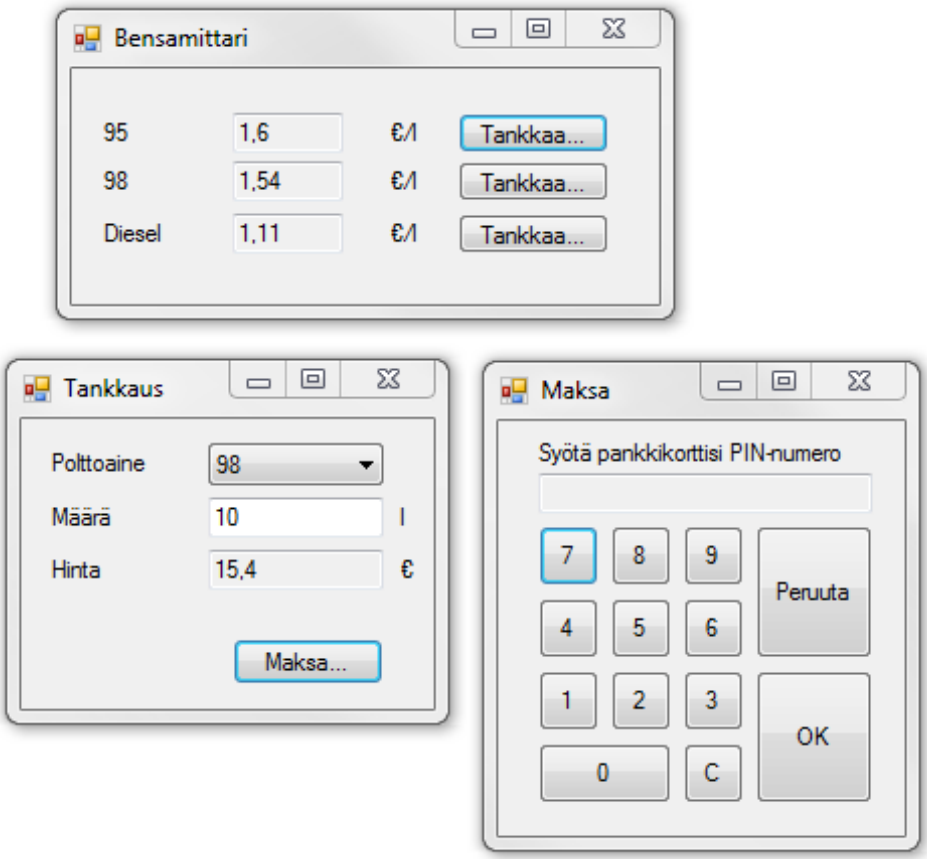
Kuva 10 Pankkitehtävän esimerkki

Opiskelijoiden tulee tehdä harjoitusohjelma, jossa on monta lomaketta. Päälomakkeella lukee pankkitilin saldo. Lisäksi lomakkeella on kaksi eri nappia, toisesta voidaan nostaa ja toisesta tallettaa rahaa. Molemmat avaavat oman alilomakkeensa. Alilomakkeissa voidaan kirjoittaa talletettava tai nostettava summa. Tämän jälkeen käyttäjän tulee painaa painonappia, joka vahvistaa siirron. Päälomakkeessa näkyy tämän jälkeen muuttunut saldo.

4.2.1 Bensa-asema -harjoitustyö

Viimeisenä työpöytäsovellusvaiheessa on jäljellä enää harjoitustyö. Se on bensa-aseman toimintaa jäljittelevä ohjelma. Ohjelma käsittelee aihetta tankkaajan, ylläpitäjän sekä valvojan kannalta. Ylläpitoa ja valvontaa varten ohjelma käyttää käyttäjätunnuksia. Tunnukset ja salasanat ovat tekstitiedostossa kryptattuna. Opiskelijat joutuvat itse opiskelemaan kryptauksen. Moodlessa on esimerkki MD5-kryptauksesta ja C#:n kirjastoista löytyvät valmiit luokat ja funktiot kryptausta varten. Ohjelmakoodiin ei tarvitse kuin kopioida vastaava kryptausfunktio.

Harjoitustyö on tarkoitettu kolmelle henkilölle. Jokaisella opiskelijalla on oma osa-alueensa, lopuksi toiminnot yhdistetään toimivaksi. Tankkaajan näkökulmasta ohjelmassa on bensamittari sekä maksuautomaatti. Ylläpitäjä ylläpitää hintoja ja mainostaulua. Bensiinitankkeja ylläpidetään myös erillisestä ohjelmasta.

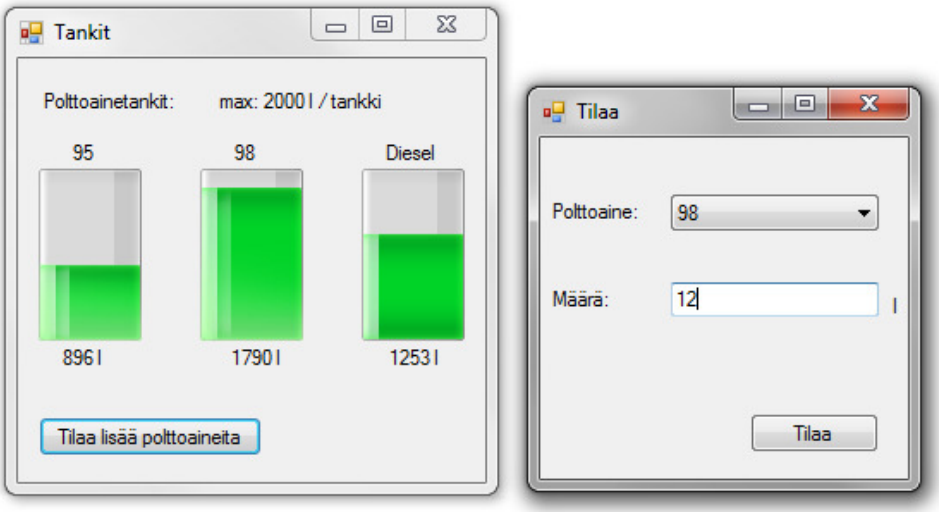


Kuva 11 Bensaharjoituksen tankkaus

Yhden ryhmän jäsenen tehtävänä on tehdä käyttöliittymä tankkaajalle. Ohjelmassa näkyvät polttoaineiden hinnat, jotka luetaan tekstitiedostosta. Ohjelmassa tulee valita litrat ja polttoainetyyppi. Ennen tai jälkeen tankkausta tulee asiakkaan maksaa tankkaus maksupäätellä. Päätteeseen näppäillään kortin kuvitteellinen PIN-koodi. Tämä PIN-koodi ei tarvitse vaihtua eri maksutapahtumien välillä. Tankkauksen jälkeen ohjelma vähentää tankatun litramäärän kyseistä polttoainetta bensa-aseman tankeista. Polttoainemäärät on talletettu tekstitiedostoon.

Kuva 12 Bensaharjoituksen kyltin ylläpito

Hintojen ja mainostaulun ylläpito on jaettu yhdelle ryhmän jäsenelle. Ohjelmassa voidaan päivittää hintoja, jotka näkyvät myös tankkaajalle. Samat hinnat näkyvät mainostaulussa. Ohjelmassa on myös mahdollista kirjoittaa vapaamuotoinen teksti mainostauluun. Ohjelmaan on kirjautuminen, tunnukset ovat tekstitiedostossa. Hinnat ovat myös talletettuna tekstitiedostoon. Opiskelijoita kannustetaan käyttämään jotakin salasanatiivistettä salasanan tallentamiseen.



Kuva 13 Bensaharjoituksen tankkien ylläpito

Huoltoasemien bensatankkien ylläpidosta tulee myös oma ohjelmansa. Tämänkin tekee yksi ryhmän jäsenistä itsenäisesti. Tankkiohjelmasta käyttäjä näkee, paljonko tankeissa on polttoainetta. Polttoaineitten määrät on tallennettu tekstitiedostoihin. Ohjelmasta on myös mahdollista tilata lisää polttoainetta. Ohjelma osaa varoittaa, jos polttoaine on lopussa jostakin tankista. Jokainen polttoainetilaus tallennetaan tekstitiedostoon. Ennen kuin sovelluksen saa auki, pitää käyttäjän kirjautua. Tällainen toiminta voidaan tehdä erillisellä lomakkeella.

Vaikka ohjelmat käyttävät samoja tekstitiedostoja, ei niitä tarvitse käyttää samaan aikaan. Jokainen ohjelma lukitsee tekstitiedostot omaan käyttöönsä, joten jos toinen yrittää niitä muokata, tulee ongelmia.

Harjoitustyötä arvioidaan palautuskeskustelussa, jossa ryhmä esittelee tuotoksensa opettajalle. Arvosana määräytyy koodin tasosta, lisäominaisuuksista ja ohjelman perustoiminnallisuuksista.

4.3 Opintojakson muu sisältö

Opintojaksossa käsitellään erilaisia lukujärjestelmiä. Opetuksessa muutetaan lukuja lukujärjestelmästä toiseen. Myös yksinkertaisia laskutoimituksia tehdään eri lukujärjestelmillä. Opintojaksolla opetetaan esimerkiksi, miten luku 5 muutetaan 10-järjestelmästä binäärimuotoon. Binääri- ja 10-järjestelmän lisäksi kurssilla käsitellään heksadesimaaleja sekä oktaalijärjestelmää. Heksadesimaalien kantaluku on 16 ja oktaalijärjestelmän 8. Binäärijärjestelmässä kantaluku on 2. Binäärilukujen kanssa käytetään loogisia funktioita (AND, OR, XOR). (Moodle 2010)

Opetuksessa käsitellään myös tietokantoja. Tietokantojen opetuksessa käytetään MySQL-tietokantaa. Opiskelijan tulee ymmärtää mitä relaatiotietokannat ovat ja miksi niitä käytetään. Opiskelijan tulee osata ymmärtää miten tietokantaa hyödynnetään ohjelmistoissa. Opiskelija tulee osata käyttää phpMyAdminia tietokannan taulun luontiin ja muokata sitä myöhemmin. Opiskelijan tulee osata kirjoittaa yksinkertaisia MySQL-kyselyitä, joita tarvitaan ohjelmassa. (Moodle 2010)

5 OHJELMOINTI-OPINTOJAKSOT TIETOJENKÄSITTELYSSÄ

Ohjelmointia käsitellään tietojenkäsittelyn opinnoissa Ohjelmoinnin perusteet ja Ohjelmointi 1 ja 2 -opintojaksoilla. Lisäksi ohjelmointia on enemmän systeemityön pääaineekseen valitsevilla. Ohjelmoinnin perusteet -opintojakso, jota tämä opinnäytetyö käsittelee, on kaikille tietojenkäsittelyn koulutusohjelmassa opiskeleville pakollinen.

Ohjelmointi 1 -opintojaksolla ohjelmointikielenä on C#. Opintojaksolla keskitytään enemmän työpöytäsovelluksiin ja olio-ohjelmointiin. Harjoitusohjelmien tulee käyttää myös tietokantaa tiedon säilytykseen.

Ohjelmointi 2 -opintojaksolla ohjelmointikielenä on Java. Opintojaksolla käsitellään perusteet Javasta. Opintojakson tavoitteena on, että opiskelija osaa tehdä web-sovelluksen, joka perustuu servletin, olioiden ja tietokantaobjektien käyttöön. Opiskelijan tulee osata tehdä ohjelmasta myös javadoc-dokumentaatio.

Systeemin toteutus -opintojakso on vain systeemityön pääaineekseen valitsevilla. Opintojaksolla on ohjelmointikielenä Java. Systeemin toteutus -opintojakso pohjautuu Systeemityön menetelmissä suunniteltuun ohjelmaan. Systeemityön menetelmissä suunnitellaan ohjelma, joka hyödyntää olio-ohjelmointia. Ohjelmasta pitää tehdä toimintakaavio, luokkakaavio ja tietokantakaavio. Suunnitelma tehdään 2-4 henkilön ryhmissä. Samat ryhmät toteuttavat ohjelmansa Javalla Systeemin toteutus -opintojaksolla.

Flash-toteutukset -opintojaksolla ohjelmoidaan Adobe Flash -animaatioita tai -pelejä ActionScript-ohjelmointikielellä. Opintojakso kuuluu eLearning ja multimedia -pääaineeseen, mutta myös systeemityön pääaineekseen valinneet voivat ottaa sen valinnaiseksi opintojaksoksi.

6 TUTKIMUSTULOKSET

On erittäin vaikeaa selittää yksi elementti ohjelmointikielestä (kuten `for`-looppi). Pakostakin joutuu puuttumaan ohjelmointikielen muihin elementteihin. Näistä elementeistä on myös viittauksia toisiin elementteihin ja niin edelleen. Ohjelmointikieleen kuuluu paljon erilaisia elementtejä, jotka kuitenkin kaikki liittyvät toisiinsa. (Robins 2010, 56.)

Tämä integrointi näkyy ohjelmoinnin opettamisessa. Ei ole selvää järjestystä, miten jokin ohjelmointikieli tulisi opettaa. Suuri kiistanalainen aihe on, pitäisikö oliot opettaa heti ensimmäiseksi vai ei. Ei ole mitään sovittua aihejärjestystä ohjelmoinnin opettamisessa. Yksinkertaisissakin ohjelmissa sivutaan monta eri aihealuetta. Ei ole mahdollista käsitellä kaikkea heti ensimmäisestä ohjelmasta alkaen. (Robins 2010, 56.)

Opimme parhaiten asioita, jos ne liittyvät jotenkin jo tuntemaamme tietoon. Oppiminen tapahtuu tehokkaimmin, jos opettelemanne tieto sivuaa jo opittua tietoa. Oppiminen tapahtuu jo oppimamme asian reunoilta. Oppiminen ei ole niin tehokasta, jos emme osaa liittää sitä johonkin oppimaamme tietoon. (Robins 2010, 56–57)

Yksinkertaisissa olio-ohjelmointiharjoitteissa on ongelmana se, ettei opiskelija välttämättä huomaa olioiden käytöstä mitään hyötyä. Olio-ohjelmoinnin hyödyt näkyvät konkreettisemmin vasta suuremmissa ohjelmakokonaisuuksissa.

Kehitin opintojakson materiaalin Visual Basic.NET -ohjeiden ja harjoitteiden perusteella Moodleen. Kehittämistyössä nousi esille paljon asioita, mitä tarvitsi huomioida opinnäytetyötä kirjoitettaessa. Itse opintojakson toteutus ei merkittävästi vaikuttanut opinnäytetyön sisältöön.

Ohjelmoinnin opetusta tullaan jatkamaan samaan tapaan, kuin aikaisemmin tietojenkäsittelyn koulutusohjelmassa. Opetustehtävät ja teoriat eivät paljon poikkea vastaavista Visual Basic .NET -materiaaleista. Jotain muutoksia kuitenkin on jouduttu tekemään, jotta harjoitukset toimisivat C#:lla.

Kirjallisista lähteistä ei löydy mitään selkeää vastausta, miten ohjelmointia pitäisi opettaa. Ehkä kuitenkin alkeisopintojaksossa on helpompaa, jos ensin opetellaan ohjelmoinnin perusteet, ennen kuin siirrytään olio-ohjelmointiin. Olio-ohjelmoinnissa tarvitsee opetella paljon uusia asioita, ennen kuin pääsee varsinaisesti ohjelmoijaan. Opiskelijalla ei välttämättä tule käsitystä, mihin olio-ohjelmoinnin luokka- ja oliorakenteita voi käyttää höydyksi itse sovelluksessa.

Opiskelijat voivat tutustua itsenäisesti Ohjelmoinnin perusteet -opintojaksolla olio-ohjelmointiin Moodlesta löytyvästä linkistä tai itse hakemillaan materiaaleilla. Olio-ohjelmointia käsitteillään vasta Ohjelmointi 1 -opintojaksossa. Opiskelijat kuitenkin käyttävät tietämättään olio-ohjelmointia, kun he käyttävät useampaa lomaketta tai monia C#:n sisäisiä luokkia. Teoriaa näistä toimenpiteistä ei kuitenkaan käsitellä. Pankki-harjoitustyössä opiskelijat joutuvat avaamaan useita lomakkeita toisesta lomakkeesta käsin, jolloin he käyttävät olioita.

Microsoft kertoo jatkavansa sekä C#:n että Visual Basic .NET:n tukemista jatkossakin. Ohjelmointikieli yritetään yhtenäistää, molemmissa ohjelmointikielissä pitäisi olla samat ominaisuudet. Lisäksi Microsoft kertoo panostavansa enemmän ihmisläheisempään ohjelmointiin. Ohjelmoijan ei tarvitsisi miettiä, miten ohjelma saadaan toimimaan ohjelmointirajapinnan tarjoamilla välineillä, vaan kertoa suoraan, mitä haluaa tehdä. Ohjelmien tulisi myös tehdä samaan aikaan monia eri toimenpiteitä. Nykyiset prosessorit ovat suurimmaksi osaksi moniydinprosessoreita. (Future Directions for C# and Visual Basic 2010)

Visual Basic on hakukoneille tehtyjen hakujen perusteella laskenut suosiota tasaisesti 2009 kesästä lähtien. Tuloksissa on erikseen Visual Basic .NET, joten tulkitseminen voi olla vaikeaa. C# on kokoajan kasvattanut suosiotaan, jos tuloksia katsotaan vuosittaisella tasolla. (TIOBE Programming Community Index 2010)

7 LOPPUYHTEENVETO

Päätavoitteena opinnäytetyössä oli selvittää, kannattaako tietojenkäsittelyn koulutusohjelman siirtyä käyttämään C#:a Visual Basic .NET:in sijaan. Aivan selvää vastausta kirjallisten lähteitten perusteella siihen ei löytynyt. Joissakin lähteissä mainittiin kuitenkin, että C# on kasvattamassa suosiotaan. Lisäksi C# on syntaksiltaan lähempänä Javaa, jota opiskellaan muilla jatkokursseilla Ohjelmoinnin perusteet -opintojakson jälkeen.

Opiskelijat todennäköisesti kokevat C#:n mielekkäämmäksi, jos heillä on ohjelmointitaustaa Javasta tai C/C++:sta. Kuitenkin kokemattomat ohjelmoijat tai Visual Basicillä ohjelmoineet voivat aloittaa ohjelmoinnin helpommin Visual Basic .NET:llä.

Ohjelmoinnin perusteet -opintojakson muuttaminen C#:lle sopii hyvin tietojenkäsittelyn koulutusohjelman opintosuunnitelmaan. Ohjelmointi 1 -opintojakso opetetaan syksyllä 2010 aloittaneille opiskelijoille myös C#:lla. Opiskelijoiden on myös helpompi siirtyä C#:sta Javaan Ohjelmointi 2- opintojaksolla. C#:n opiskelu tukee myös muita kuin ohjelmointiopintojaksoja. Myös Microsoftin toiminnanohjaus- ja asiakkuudenhallintajärjestelmiin ohjelmoidaan tulevilla opintojaksoilla lisäosia ja mukautuksia kuvitteellisen asiakkaan tarpeisiin.

Yleisesti ohjelmoinnin opettamiseen ei ole mitään valmiita vastauksia, kaikilla on omat mieltymyksensä opettamisessa. Eri asioita voidaan käsitellä eri vaiheessa opintojaksolla. Ei ole selkeästi parasta tapaa opettamiseen. Opiskelijat oppivat hyvin eri tavalla ohjelmoinnin. Ohjelmointi on kuitenkin käytännön työtä, opiskelijat oppivat tekemällä itse ohjelmia. Itse tekemässään ohjelmassa opiskelijat ymmärtävät, mitä ohjelmassa tapahtuu. Opintojaksolla onkin paljon harjoitteita sekä lopussa suurehko harjoitustyö. Opintojakson toteutus käyttää suurimmaksi osaksi tietojenkäsittelyn koulutusohjelmassa hyviksi todettuja opetustapoja ja harjoitteita.

LÄHTEET

Ala-Mutka, K. 2005. Ohjelmoinnin opetuksen ongelmia ja ratkaisuja. ReflekTori 2005. Viitattu 10.5.2010. http://opetuki2.tkk.fi/p/reflektori/verkkojulkaisu/pdf/c5_kirsi_ala-mutka.pdf

Albahari, J. & Albahari B. 2010. C# 4.0 in a Nutshell: The Definitive Reference. USA: O'Reilly Media.

Bronson, G. & Rosenthal, D. 2005. Introduction to programming with Visual Basic .NET. USA, Sudbury, MA : Jones and Bartlett Publishers.

Future Directions for C# and Visual Basic. 2010. Microsoft PDC10. Viitattu 13.9.2010. http://www.microsoftpdc.com/2009/FT11

Halvorson, M. 2008. Visual Basic 2008. Helsinki: Readme.fi.

Ji, M. & Gong, M. 2002. Differences Between Microsoft Visual Basic .NET and Microsoft Visual C# .NET. Microsoft 2002. Viitattu 9.9.2010. http://support.microsoft.com/kb/308470

Moodle. Ohjelmoinnin perusteet (TRTKNU10A3). Hämeen ammattikorkeakoulu. Viitattu 9.9.2010. https://moodle2.hamk.fi/course/view.php?id=2341

Moghadampour, G. 2009. C#-ohjelmointi. Jyväskylä: Docendo.

MSDN Library 2010a. Overview of the .NET Framework. Microsoft Corporation. Viitattu 12.6.2010.
http://msdn.microsoft.com/en-us/library/a4t23ktk%28v=VS.100%29.aspx

MSDN Library 2010b. .NET Framework 4. Microsoft Corporation. Viitattu 14.6.2010.
http://msdn.microsoft.com/en-us/library/w0x726c2%28v=VS.100%29.aspx

MSDN Library 2010c. Compiling to MSIL. Microsoft Corporatioin. Viitattu 12.9.2010.
http://msdn.microsoft.com/en-us/library/c5tkafs1%28v=VS.90%29.aspx

Patrick, T. 2006. Start-to-Finish Visual Basic 2005: Learn Visual Basic 2005 as You Design and Develop a Complete Application. USA: Addison-Wesley Professional

Richter, J. 2010. CLR Via C#. 3rd Edition. USA: Microsoft Press.

Research on Learning Programming. 2005. Viitattu 10.5.2010. http://www.cs.hut.fi/Research/COMPSER/ROLEP/ROLEP_paasivu.html

Robins, A. 2010. Learning edge momentum: a new account of outcomes in CS1. Computer Science Education. 20 (1), 37-71. Viivattu 10.5.2010. Saatavissa informaworld-tietokannassa: http://dx.doi.org/10.1080/08993401003612167

SoleOPS. Ohjelmoinnin perusteet –opintojakson kuvaus. Hämeen ammattikorkeakoulu. Viitattu 21.6.2010. https://portal.hamk.fi/opsnet/disp/fi/ops_ojYllapito/edi/tab/ops?ryhman_id=3614015&opinkohd=1814452&id2=3614042&stack=push

Strong Names Explained. 2004. The Code Project. Viitattu 12.9.2010. http://www.codeproject.com/KB/security/StrongNameExplained.aspx

TIOBE Programming Community Index. TIOBE. Viitattu 13.9.2010. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Wiley, J. & Hoboken, S. 2008. Visual Basic 2008 Programmer's Reference. NJ, USA: John Wiley & Sons