OULUN AMMATTIKORKEAKOULU

Tatu Piippo

KOOSTEOPINNÄYTETYÖ: C-, C++- JA C#-OHJELMOINTIKIELIEN EROT SEKÄ TILAUSTIEDOSTOJEN LUKEMINEN JA KIRJOITTAMINEN C#-SERVICE-OHJELMALLA

KOOSTEOPINNÄYTETYÖ: C-, C++- JA C#-OHJELMOINTIKIELIEN EROT SEKÄ TILAUSTIEDOSTOJEN LUKEMINEN JA KIRJOITTAMINEN C#-SERVICE-OHJELMALLA

Tatu Piippo
Opinnäytetyö
Kevät 2018
Tieto- ja viestintätekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tieto- ja viestintätekniikan tutkinto-ohjelma, ohjelmistotuotanto

---

Tekijä: Tatu Piippo
Opinnäytetyön nimi: Koosteopinnäytetyö: C-, C++- JA C#-ohjelmointikielien erot sekä tilaustiedostojen lukeminen ja kirjoittaminen C#-service-ohjelmalla
Työn ohjaaja(t): Veijo Väisänen, Pekka Alaluukas
Työn valmistumislukukausi ja -vuosi: Kevät 2018 Sivumäärä: 62 (9 + 2 liitettä)

Tämä on kooste opinnäytetyö, joka toteutettiin kahdessa eri osassa. Ensimmäinen osa kertoo C-, C++- ja C#-ohjelmointikielten eroavaisuuksista käytännössä kooditasolla sekä se myös sisältää pohdintoja kielten käyttötarkoituksista. Toisessa opinnäytetyöosassa toteutetaan C#-kielellä service-ohjelma. Service-ohjelmat ovat käyttöjärjestelmän taustalla pyöriviä ohjelmia, joilla ei ole minkäänlaista käyttöliittymää. Toisen osan ohjelma toteutettiin alun perin Procomp Solutions -yritykselle, mutta opinnäytetyö varten siitä on jätetty pois Procompin järjestelmiä sekä sen asiakkaita koskevat kohdat.

Molemmissa opinnäytetyön osissa on käytetty Microsoft Visual Studiota. Ohjelmia tehdessä on hyödynnetty Microsoftin omia dokumentaatioita sekä omia ohjelmointikokemuksia. Ensimmäisen osan ohjelmointikielten käyttötarkoituksia on tutkittu eri lähteiden avulla.

Molempien osien toteutus oli mielenkiintoista. On hyvä myös tietää, mitä ohjelmointikieltä käyttäisi missäkin tilanteessa. C-kielet ovat myös yleisessä käytössä ja niiden tuntemus on erittäin tärkeää. Toisen osan service-ohjelma oli kokonaan uusi sovellustyyppi, josta ei ollut aiempaa tietämystä. Service-ohjelmien tietämys avaa paljon uusia mahdollisia erilaisia projekteja varten.

Asiasanat: C#, C++, C, ohjelmointi, toiminnanohjausjärjestelmät

ABSTRACT

Oulu University of Applied Sciences
Degree programme in Information Technology, software developing

---

Author(s): Tatu Piippo
Title of thesis: Compilation thesis: Differences of C-, C++ And C#-programming languages, and Reading and writing order files with a C#-service program
Supervisor(s): Veijo Väisänen, Pekka Alaluukas
Term and year when the thesis was submitted: Spring 2018 Number of pages: 62 (9 + 2 attachments)

This is a compilation thesis that was done in two parts. The first part is about the differences of the C-, C++- and C# -programming languages. It looks for the differences in the basic code structures and in theoretical stand point on what language you should use on different types of projects. The second thesis part is about programming a service program that read and writes order files for communicating with external systems without direct access. Service-programs are programs that run in the background and they don't have any user interface. The service program was originally done for Procomp Solutions -company, but for this thesis I left out all things concerning the company or its clients. It is presented in a more basic standpoint.

Microsoft Visual Studio was used in both thesis parts. In making of the programs Microsofts documentation was used as well as my own experiences in programming. The theoretical standpoint of the first thesis part was researched with different sources.

Doing both parts of the thesis was very interesting. It is good to know which programming languages I should use on which occasions. C-languages are also very commonly used so knowing more about them is extremely useful. Developing the second thesis parts service-program opened up a whole new program type for me. I didn't have any previous knowledge about service-programs beforehand.

Keywords: C#, C++, C, programming, enterprise resource planning

SISÄLLYS

1 JOHDANTO

Tämä opinnäytetyö suoritettiin kahdessa osassa, yhdessä viiden opintopisteen osassa sekä toisessa kymmenen opintopisteen osassa. Ensimmäinen osa tehtiin vuoden 2016 kevätlukukaudella ja toinen osa vuoden 2018 kevätlukukaudella. Tämän koostedokumentin tarkoituksena on esitellä molemmat opinnäytetyön osat yhdessä dokumentissa.

Ensimmäisessä osassa tutustuttiin C-, C++- ja C#-ohjelmointikielten eroavaisuuksiin käytännössä sekä pohdittiin, mihin kukin kieli sopisi parhaiten. Valitsin tämän aiheen sillä voisin uskoa, että moni aloitteleva ohjelmoija miettii, mikä näiden kielten ero on. Itseäni kiinnosti tietää, millaisia puutteita perus-C-kielessä on näihin kahteen muuhun verrattuna sekä se mihin itse voisin kieliä käyttää.

Toisessa osassa toteutettiin tilaustenkäsittelyohjelma C#-kielellä. Ohjelma toteutettiin service-ohjelmana. Service-ohjelmat tai palveluohjelmat ovat tietokoneen taustalla ajettavia ohjelmia, joilla ei ole omaa käyttöliittymää. Valitsin tämän aiheen, sillä tein samantyyppisen ohjelman Procomp Solutions -yritystä varten. Opinnäytetyön toteutuksessa jätin kuitenkin Procomp Solutionsin ja sen asiakkaita koskevat osat pois.

2 ENSIMMÄISEN OSAN ESITTELY

Opinnäytetyön ensimmäisessä osassa (liite 1) tutustuttiin C-, C++- sekä C#-kieliin ja niiden eroavaisuuksiin. Kielten eroavaisuuksia tutkittiin käytännössä tekemällä samankaltaisen ohjelman kullakin kielellä. Ohjelmaan tehtiin yksinkertainen luokkarakenne, jossa on myös aliluokka. Tässä tulee esille, kuinka ohjelmoinnin perustoiminnot eroavat kussakin kielessä. Kielten erovaisuuksia tutkittiin myös eri lähteiden avulla. Osassa keskityttiin C-kielten eroavaisuuksiin, eikä näin ollen sisällä kaikkia C-kielten yhteisiä ominaisuuksia taikka C-kielten vertailua muihin kieliin.

3 TOISEN OSAN ESITTELY

Toisessa opinnäytetyön osassa (liite 2) toteutettiin C#-service-ohjelma, joka automaattisesti luki tiedostoja ja kirjoitti niiden sisällön kantaan sekä luki kannassa olevaa tietoa ja kirjoitti ne tiedostoksi. Tällainen sovellus toteutettiin Procomp Solutions -yritystä varten kahden eri järjestelmän kommunikointiin. Tätä opinnäytetyön osaa varten kuitenkin sovelluksen toteutuksesta poistettiin Procomp Solutionsin järjestelmiä ja sen asiakkaita koskevat kohdat, joten se on toteutettu yleisemmältä näkökannalta.

4 YHTEENVETO

Tämä opinnäytetyö suoritettiin kahdessa eri osassa yhden osan sijaan. Ensimmäinen osa käsitteli C-, C++- sekä C#-ohjelmointikielten eroavaisuuksia. Toisessa osassa toteutettiin C#-service-ohjelma, joka automaattisesti luki tilaustiedostoja tietokantaan sekä kirjoitti niitä tietokannasta löytyvillä tiedoilla.

Opinnäytetyön suorittaminen kahdessa eri osassa oli kätevää. Työpaineen pystyi helpommin tasoittamaan, kun ei tarvinnut tehdä yhtä suurta opinnäytetyötä kerralla. Opinnäytetyön ei myöskään tarvinnut olla samasta aiheesta, jos se oli tehty osissa. Tämäkin sinällään helpotti opinnäytetyön kirjoittamista.

LIITTEET

Liite 1 C-, C++- JA C#-Ohjelmointikielien erot

Liite 2 Tilaustiedostojen lukeminen ja kirjoittaminen C#-service-ohjelmalla

LIITE 1

OULUN AMMATTIKORKEAKOULU

Tatu Piippo

C-, C++- JA C#-OHJELMOINTIKIELIEN EROT

C-, C++- JA C#-OHJELMOINTIKIELIEN EROT

Tatu Piippo
Opinnäytetyö osa 1
Kevät 2016
Tieto- ja viestintätekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistotuotanto

Tekijä: Tatu Piippo
Opinnäytetyön nimi: C-, C++- JA C#-Ohjelmointikielien erot
Työn ohjaaja(t): Veijo Väisänen, Tuula Hopeavuori
Työn valmistumislukukausi ja -vuosi: Kevät 2016 Sivumäärä: 15 + 3 liitettä

Työn tarkoituksena oli tutkia, mitä eroja on C-, C++- ja C#-ohjelmointikielillä koodi tasolla sekä pohtia ja selvittää millaisiin käyttötarkoituksiin mikäkin kieli sopii. Vertailussa keskitytään C-kielten eroavaisuuksiin eikä sisällä kaikkia C-kielten yhteisiä ominaisuuksia eikä niitä myöskään verrata muihin ohjelmointikieliin.

Ohjelmointikielien eroa tutkitaan tekemällä samankaltainen ohjelma C-, C++- sekä C#-ohjelmointikielillä. Kielten hyviä ja huonoja puolia sekä mahdollisia käyttötarkoituksia selvitetään myös eri lähteiden avulla.

C-kielen vertaaminen uudempiin olio-ohjelmointiin perustuviin kieliin oli kiinnostavaa. C-kieltä koodatessa yllätyin, kuinka hyvin siinä pystyi suorittamaan olio-ohjelmoinnin kaltaista ohjelmointia, vaikka siinä ei ole luokkia. Samalla selvisi mukavia pieniä oikoteitä C++ ja C# koodaukseen ja myös niidenkin eroavaisuuksia.

Asiasanat: C#, C++, C, ohjelmointi

ABSTRACT

Oulu University of Applied Sciences
Degree programme in Information Technology, software developing

---

Author(s): Tatu Piippo
Title of thesis: Differences of C-, C++ And C#-programming languages, and Reading and writing order files with a C#-service program
Supervisor(s): Veijo Väisänen
Term and year when the thesis was submitted: Spring 2018 Number of pages: 15 + 3 attachments

---

The point of this project is to research the differences of C-, C++ and C#-programming languages in a basic code level and also to think what each would be best suited for. In the comparison we will be focusing on the differences of the C-languages, so it will not include things common to the three or comparisons to other programming languages.

The differences will be compared by programming a similar program on each of the three programming languages. We will also research the good and bad parts of the languages with different sources.

Comparing the three programming languages was interesting. I was surprised how well C-language could do object-oriented programming even though it does not have any classes. It was also nice to get to know a few neat tricks with C++- and C#- languages.

Keywords: C#, C++, C, programming, enterprise resource planning

SISÄLLYS

SANASTO

Argumentti – Muuttuja joka syötetään funktiolle sitä kutsuttaessa.

Funktio – Toiminnon tai toimintoja sisältävä kokonaisuus.

Getter – Hakee luokan muuttujan arvon.

Muodostin – Oliota tehdessä käytetty funktio, jossa syötetään oliolle muuttujat.

Setter – Asettaa luokan muuttujan arvon.

Structi – Muuttujia sisältävä rakenne.

1 JOHDANTO

Tässä dokumentissa selvitetään C-, C++ sekä C#-ohjelmointikielien eroavaisuuksia sekä käyttötarkoituksia. Dokumentissa verrataan myös toiminnaltaan samanlaisen ohjelman kirjoittamista kullakin kielellä.

Dokumentissa keskitytään C-kielten eroavaisuuksiin, eikä näin ollen sisällä kaikkia C-kielten yhteisiä ominaisuuksia taikka C-kielten vertailua muihin kieliin.

Valitsin kielet aiheeksi, koska voin uskoa, että moni aloitteleva ohjelmoija miettii, mikä näiden kielten ero on. Itseäni kiinnostaa tietää mitä puutteita perus C-kielessä on näihin kahteen muuhun verrattuna sekä se, mihin voisin itse kieliä käyttää.

2 C++ JA C#, C-KIELEN JÄLKELÄISINÄ

C-kieltä on käytetty monen ohjelmointi kielen perustana kuten juurikin näissä C++ ja C# kielissä. C++-kieli kehiteltiin olemaan C-kieli luokkien kanssa. C++-kieli toi mukanaan luokka systeemin, luokkien perimisen, inline-funktiot, funktioiden oletus argumentit sekä muita ominaisuuksia. Näistä ehkä tärkein on luokat. (1, s. 11.)

Luokat ovat kuin paranneltuja C-kielen structeja. Luokat voivat sisältää muuttujia, funktiota, muodostimia, destructorin sekä perimisen. C-kielen structit sen sijaan sisältävät vain muuttujia. Luokkien puutteiden takia C-kielessä ei pysty toteuttamaan olio-ohjelmointia kunnolla. C++-kielessä yhtenä tärkeänä ominaisuutena on myös muistin käsittely. Pointer tyyppisten muuttujien avulla pystytään viittaamaan muistissa kohtaan, jossa muuttuja tai objekti sijaitsee. Pointtereiden avulla ohjelmoija voi vapauttaa muistia kun pointterilla merkittyä muuttujaa tai objektia ei enää tarvitse.

C# luotiin C++- ja Java-kielten myötä. C#-kielen tärkeimpänä ominaisuutena C++-kieleen verrattuna on sen automatisoitu muistin käsittely. Automatisoidun muistinkäsittelyn myötä C#-kielessä ei voi käyttää pointtereita eikä vapauttamaan muistia käsin. (2, s. 11.)

3 KIELTEN VERTAILU KÄYTÄNNÖSSÄ

Vertailua varten tein saman ohjelman C-, C#- ja C++-kielillä. Ohjelmassa on eläin, koira sekä susi. Eläin sisältää yleiset ominaisuudet eläimelle kuten ikä ja nimi muuttujan sekä funktiot tietojen tulostamiselle ja nimen vaihtamiselle. Koira sisältää kaiken mitä eläimessäkin on sekä rodun, haukutoiminnon ja tietojentulostus-toiminnon. Susi sisältää tiedon onko tämä lauman johtaja vai ei sekä ulvo-toiminnon ja oman tulostus-toiminnon. Koodit löytyvät myös liitteenä dokumentin lopussa.

3.1 Luokan luominen

Ensimmäiseksi katsotaan miten eläin on luotu C-, C#- ja C++-kielissä.

```
struct Elain {
    char nimi[10];
    int ika;
};
```

```
class Elain
{
    public string nimi;
    protected int ika;
```

```
class Elain {

protected:
    string nimi;
    int ika;
```

Kuva 1. Eläimen määrittely C-, C# ja C++-kielillä

Kuvan vasemmanpuoleinen osio on tehty C-kielellä, keskimmäinen osio on tehty C#-kielellä ja oikean puoleinen C++-kielellä.

Kuvassa ensimmäisellä rivillä näkyy eläimen määrittely luokaksi paitsi C-kielessä, jossa ei luokkia ole, se on määritelty structiksi. Structit ovat vähän kuin alkeellisia luokkia, ne voivat sisältää muuttujia muttei funktioita.

Structiin sekä luokkiin on määritelty kaksi muuttujaa nimi sekä ikä. Nimi muuttuja voi sisältää useamman eri merkin, joten se on luotu string muuttujalla C#- ja C++-kielissä. C-kielessä ei string muuttujia ole, joten siinä on täytynyt käyttää char-taulukkoa. Char-taulukolle täytyy määritellä valmiiksi sen pituus (esimerkissä 10). Tämä voi tuottaa ongelmia myöhemmin ohjelmaa ajettaessa jos haluasikin syöttää siihen pitemmän merkkijonon. Taas jos taulukolle määrittelee liian suuren pituuden valmiiksi, se varaisi itsellensä turhaan ylimääräistä muistia.

C#- ja C++-kielissä muuttujat ovat myös määritelty protected tyyppisiksi paitsi C# kielen nimi muuttuja, josta kerrotaan myöhemmin lisää. Protected tyypin määrittäminen mahdollistaa sen että aliluokat voivat käyttää niitä. C-kielessä ei voi protected ominaisuutta määritellä vaikka siinä voi periyttää yhden structin muuttujat toiselle. Huomaa myös että C++-kielellä tämän määrittelyn voi

tehdä monelle muuttujalle kerralla yhdellä rivillä laittamalla protected määrittelyn perään kaksoispiste. C#-kielessä tämä määrittely täytyy tehdä jokaiselle muuttujalle ja funktiolle erikseen.

3.2 Muodostimen luonti

Seuraavaksi katsomme muodostimen luomista. C-kielessä ei pysty luomaan oikeita muodostimia structeille, joten sen sijaan on käytetty tavallista funktiota.

C

```c
//Asettaa nimen ja iän eläimelle
void elainConstructori(struct Elain* elain, char nimi[10], int ika) {
    strcpy(elain->nimi, nimi);
    elain->ika = ika;
}
```

C#

```csharp
//Constructori joka asettaa nimen ja iän
public Elain(string nimi = "Ei nimea", int ika = 0)
{
    this.nimi = nimi;
    this.ika = ika;
}
```

C++

```cpp
public:
    //Contructori joka asettaa nimen ja iän
    Elain(string nimi = "Ei nimea",int ika = 0) : nimi(nimi),ika(ika) {}
```

Kuva 2. Muodostimen luominen eri C-kielillä

C-kielen structeihin ei voi suoraan liittää funktioita, siksi muodostimen funktiossa täytyy antaa argumenttina eläin, jolle tiedot halutaan asettaa. C#- ja C++-kielissä tätä argumenttia ei tarvitse antaa sillä muodostinta kutsutaan muutenkin jo eläin-objektia luodessa tai kutsu tehdään objektin kautta. C#- ja C++-kielten muodostimissa on myös asetettu oletus argumentit, jota ei C-kielessä voi tehdä. Esimerkiksi jos luodaan uusi eläin, mutta sille ei anneta nimeä eikä ikää, muodostin asettaa automaattisesti eläimelle nimeksi ”Ei nimea” ja iäksi ”0”.

C

```c
//Luodaan eläin (elukka1), jonka jälkeen annetaan sille nimeksi "Elukka" ja iäksi 10
struct Elain elukka1;
elainConstructori(&elukka1, "Elukka", 10);
```

C#

```csharp
//Luodaan uusi eläin (elukka1) ja annetaan sille nimeksi Elukka ja iäksi 10
Elain elukka1 = new Elain("Elukka",10);
```

C++

```cpp
//Luodaan uusi eläin (elukka1) ja annetaan sille nimeksi Elukka ja iäksi 10
Elain *elukka1 = new Elain("Elukka",10);
```

Kuva 3: Muodostimen kutsu C-, C# ja C++-kielillä

Muodostimen kutsumisessakin näkee että C-kielessä ei oikeasti kutsuta muodostinta vaan funktiota. Funktiolle täytyy antaa argumenttina eläin, jolle nimi ja ikä asetetaan. Funktiota ei myöskään voi kutsua suoraan eläimen luonnin yhteydessä. C#- ja C++-kielten muodostimen kutsu ei poikkea toisistaan juuri ollenkaan. Ainoana erona on että C++-kielessä luon eläimen pointterina.

3.3 Getter ja Setter

Getterit ja setterit ovat yleisiä ominaisuuksia luokissa, joiden avulla voidaan hakea muuttujan arvo tai asettaa sille uusi arvo.

C

```c
//Setter nimi muuttujalle
void setNimi(struct Elain* elain, char nimi[10]) {
    strcpy(elain->nimi, nimi);
}

//Getter nimi muuttujalle
char getNimi(struct Elain* elain) {
    return elain->nimi;
}
```

C#

```csharp
public string nimi;
```

C++

```cpp
//Setter nimi muuttujalle
void setNimi(string nimi)
{
    this->nimi = nimi;
}

//Getter nimi muuttujalle
string getNimi()
{
    return nimi;
}
```

Kuva 5: Setter ja getter nimi muuttujalle

C-kielessä näillekin täytyy antaa argumenttina eläin, jonka tieto päivitetään tai haetaan. C#- ja C++-kielten setterit, getterit ja muut luokan funktiot sijaitsevat luokan sisällä. Niiden kutsuminen tapahtuu luodun objektin kautta ja siksi argumenttina ei tarvitse antaa objektia, jonka tietoja ollaan muuttamassa.

C#-kielen getteri ja setteri luodaan automaattisesti jos vain asettaa luokassa olevan muuttujan public tyyppiseksi eikä muuttujille näin ollen tarvitse erikseen kirjoittaa setteri ja getteri funktiota. Siksi kuvassakin näkyy C#-kielen kohdalla vain muuttujan määrittely. Tietenkin jos getteriin tai setteriin haluaa muitakin ominaisuuksia, ne voidaan myös kirjoittaa itse.

3.4 Aliluokan luominen

Nyt katsomme miten aliluokka luodaan ja miten se onnistuu C-kielen structeilla

C

```c
//Koiran luonti, koiralle periytetään Elain structi ja annetaan uusi muuttuja
struct Koira {
    struct Elain;
    char rotu[20];
};
```

C#

```csharp
//Koira luokan määrittely, jolle periytetään Elain luokka
class Koira : Elain
{
    string rotu;
```

C++

```cpp
//Koira luokan luonti, jolle periytetään Elain
class Koira: Elain {

    string rotu;
```

Kuva 5: Aliluokan luominen C-, C#- ja C++kielillä

C-kielen Koira-structiin periytetään Eläin-structi, joka tapahtuu samalla tavalla kuin muuttujan tekeminen structille. C#- ja C++-kielissä luokan periminen tapahtuu lisäämällä ”: Elain” koira-luokan nimen määrittelyn jälkeen.

C

```c
//Koiran constructori
void koiraConstructori(struct Koira* koira, char nimi[10], int ika, char rotu[20]) {
    strcpy(koira->nimi, nimi);
    koira->ika = ika;
    strcpy(koira->rotu, rotu);
}
```

C#

```csharp
//Koira luokan constructori
public Koira(string nimi, int ika, string rotu) : base(nimi, ika)
{
    this.rotu = rotu;
}
```

C++

```cpp
public:
    //Koira-luokan constructori
    Koira(string nimi,int ika,string rotu) : Elain(nimi,ika), rotu(rotu){}
```

Kuva 6: Aliluokan muodostin C-, C#- ja C++-kielillä

C#- sekä C++-kielen muodostimissa pystytään viittaamaan suoraan ylemmän luokan muodostimeen. C#-kielessä kohta ”base(nimi, ika)” ja C++-kielessä kohta ”Elain(nimi,ika)” kutsuvat suoraan Elain-luokan muodostinta.

4 KIELIEN KÄYTTÖTARKOITUKSIA

C-kielen luokka puutteiden myötä sekä sen yksinkertaisuudesta johtuen sillä ei voisi tehdä helposti suuria monimutkaisia ohjelmia ilman että koodi menisi kauhean monimutkaiseksi. Siksi C-kieli soveltuisi paremmin esimerkiksi yksinkertaisten laitteiden ohjelmointiin tai johonkin mihin ei tarvitsisi kauhean isoa ohjelmaa.

C++-kielessä suurten ja monimutkaisten ohjelmien käsittely on helpompaa, siksi sitä voi paremmin käyttää esimerkiksi työpöytä sovellusten ohjelmointiin tai pelien ohjelmointiin. C++-kieli antaa myös hyvät mahdollisuudet hallinnoida itse muuttujien ja objektien elinaikaa poistamalla ne koodissa silloin kun niitä ei enää tarvitse. Tämä voi auttaa pienentämään ohjelman RAM-muisti vaatimuksia. C++-kieli on myös käytetyin ohjelmointikieli pelimoottoreissa. (3, s. 11.)

C#-kieli sopii myös hyvin suuriin ja monimutkaisiin ohjelmiin. Erona on että C#-kielessä ohjelmoijan ei tarvitse päättää milloin poistetaan ja mitä poistetaan, koska se hoituu automaattisesti C#-kielen roskankeruu ominaisuudella, joka itsestään vapauttaa tilaa poistamalla tarpeettomia muuttujia ohjelmasta. C#-kieli on myös melko yleinen pelimoottoreissa käytetty kieli.

5 YHTEENVETO

C-kielen vertaaminen uudempiin olio-ohjelmointiin perustuviin kieliin oli kiinnostavaa. C-kieltä koodatessa yllätyin kuinka hyvin siinä pystyi suorittamaan olio-ohjelmoinnin kaltaista ohjelmointia vaikka siinä ei ole luokkia. Samalla selvisi mukavia pieniä oikoteitä C++ ja C# koodaukseen ja myös niidenkin eroavaisuuksia.

LÄHTEET

1. History of C++. Saatavissa: http://www.cplusplus.com/info/history/. Hakupäivä 3.3.2016.
2. C# and it's features. Saatavissa: http://www.c-sharpcorner.com/UploadFile/ggaga-nesh/CSfeatures11082005232713PM/CSfeatures.aspx. Hakupäivä 3.3.2016.
3. When and why you should use C++. Saatavissa: http://in-sights.dice.com/2013/07/25/who-uses-c-and-why/. Hakupäivä 21.3.2016.

LIITTEET

Liite 1 C-koodi

Liite 2 C#-koodi

Liite 3 C++-koodi

```c
#include <stdio.h>
#include <string.h>

#define bool int
#define true 1
#define false 0

//Eläimen constructorit funktiot yms
struct Elain {
    char nimi[10];
    int ika;
};

//Asettaa nimen ja iän eläimelle
void elainConstructori(struct Elain* elain, char nimi[10], int ika) {
    strcpy(elain->nimi, nimi);
    elain->ika = ika;
}

//Tulostaa eläimen nimen sekä iän
void tulostaTiedot(struct Elain* elain) {
    printf(" Nimi: %s\n Ika: %d\n", elain->nimi, elain->ika);
}

//Setter nimi muuttujalle
void setNimi(struct Elain* elain, char nimi[10]) {
    strcpy(elain->nimi, nimi);
}

//Getter nimi muuttujalle
char getNimi(struct Elain* elain) {
    return elain->nimi;
}



//Koiran luonti, koiralle periytetään Elain structi ja annetaan uusi muuttuja
struct Koira {
    struct Elain;
    char rotu[20];
};

//Koiran constructori
void koiraConstructori(struct Koira* koira, char nimi[10], int ika, char rotu[20]) {
    strcpy(koira->nimi, nimi);
    koira->ika = ika;
    strcpy(koira->rotu, rotu);
}

//Tulostaa koirantiedot. Tämä funktio ei syrjäytä aiemin tehtyä tulostaTiedot-funktiota
void tulostaKoiranTiedot(struct Koira* koira) {
    printf("\n Nimi: %s\n Ika: %d\n rotu: %s\n\n", koira->nimi, koira->ika, koira->rotu);
}

void hauku(struct Koira* koira) {
    printf("\n %s sanoo hau hau!\n", koira->nimi);
}
```

```c
//Suden constructorit funktiot yms
struct Susi {
    struct Elain;
    bool laumanjohtaja;
};

//Suden constructori
void susiConstructori(struct Susi* susi, char nimi[10], int ika,bool laumanjohtaja) {
    strcpy(susi->nimi, nimi);
    susi->ika = ika;
    susi->laumanjohtaja = laumanjohtaja;
}

//Tulostaa sudentiedot. Tämä funktio ei syrjäytä aiemin tehtyä tulostaTiedot-funktiota
void tulostaSudenTiedot(struct Susi* susi) {
    char laumanjohtaja[5] = "";
    if (susi->laumanjohtaja) {
        strcpy(laumanjohtaja, "On");
    }
    else{
        strcpy(laumanjohtaja, "Ei ole");
    }
    printf("\n Nimi: %s\n Ika: %d\n %s laumanjohtaja \n", susi->nimi, susi->ika,laumanjohtaja);
}

void ulvo(struct Susi* susi) {
    printf("\n %s ulvoo!\n", susi->nimi);
}


int main(void)
{
    //Luodaan eläin (elukka1), jonka jälkeen annetaan sille nimeksi "Elukka" ja iäksi 10
    struct Elain elukka1;
    elainConstructori(&elukka1, "Elukka", 10);

    struct Koira koira1;
    koiraConstructori(&koira1, "Musti", 5, "Husky");

    struct Susi susi1;
    susiConstructori(&susi1, "Hopeanuoli", 4, true);

    //Tulostetaan eläimien tiedot ja kokeillaan ulvomista sekä haukkumista
    tulostaTiedot(&elukka1);
    tulostaKoiranTiedot(&koira1);
    tulostaSudenTiedot(&susi1);

    ulvo(&susi1);
    hauku(&koira1);

    scanf("");
    return 0;
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CsExample
{
    class Program
    {
        class Elain
        {
            public string nimi;
            protected int ika;

            //Constructori joka asettaa nimen ja iän
            public Elain(string nimi = "Ei nimea", int ika = 0)
            {
                this.nimi = nimi;
                this.ika = ika;
            }

            ~Elain()
            {
                Console.WriteLine("\n Destroyed: " + nimi);
            }

            //Tulostaa eläimen nimen sekä iän
            public virtual void tulostaTiedot()
            {
                Console.WriteLine("\n Nimi: " + nimi + "\n ika: " + ika + "\n");
            }
        }


        //Koira luokan määrittely, jolle periytetään Elain luokka
        class Koira : Elain
        {
            string rotu;

            //Koira luokan constructori
            public Koira(string nimi, int ika, string rotu) : base(nimi, ika)
            {
                this.rotu = rotu;
            }

            //Tulostaa koiran tiedot, funktio korvaa Elain-luokan tulostaTiedot-funktion
            public override void tulostaTiedot()
            {
                Console.WriteLine("\n Nimi: " + nimi + "\n ika: " + ika + "\n Rotu: " + rotu + "\n");
            }


            public void hauku()
            {
                Console.WriteLine("\n " + nimi + " sanoo hau hau.");
            }

        }
```

```
        //Susi luokan määrittely, Susi-luokalle periytetään Elain luokka
        class Susi : Elain
        {
            Boolean laumanjohtaja;

            //Susi-luokan constructori
            public Susi(string nimi, int ika, Boolean laumanjohtaja) : base(nimi, ika)
            {
                this.laumanjohtaja = laumanjohtaja;
            }

            //Tulostaa suden tiedot, funktio korvaa Elain-luokan tulostaTiedot-funktion
            public override void tulostaTiedot()
            {
                string onkoJohtaja;
                if (laumanjohtaja)
                {
                    onkoJohtaja = "On laumanjohtaja";
                }
                else {
                    onkoJohtaja = "Ei ole laumanjohtaja";
                }

                Console.WriteLine("\n Nimi: " + nimi + "\n ika: " + ika + "\n " + onkoJohtaja + "\n");
            }

            public void ulvo()
            {
                Console.WriteLine("\n " + nimi + " ulvoo!");
            }
        }
```

```
        static void Main(string[] args)
        {
            //Luodaan uusi eläin (elukka1) ja annetaan sille nimeksi Elukka ja iäksi 10
            Elain elukka1 = new Elain("Elukka",10);

            Koira koira1 = new Koira("Musti",5,"Husky");

            Susi susi1 = new Susi("Hopeanuoli",4,true);

            //Tulostetaan eläimien tiedot ja kokeillaa ulvomista ja haukkumista
            elukka1.tulostaTiedot();
            koira1.tulostaTiedot();
            susi1.tulostaTiedot();

            susi1.ulvo();
            koira1.hauku();
            string a = Console.ReadLine();
        }
    }
}
```

```cpp
#include "stdafx.h"
#include <iostream>
#include <string>

using namespace std;

class Elain {

protected:
    string nimi;
    int ika;

public:
    //Contructori joka asettaa nimen ja iän
    Elain(string nimi = "Ei nimea",int ika = 0) : nimi(nimi),ika(ika) {}

    ~Elain()
    {
        cout << "Destroyed " << nimi;
    }
    //Tulostaa eläimen nimen sekä iän
    void tulostaTiedot() {
        cout << "\n Nimi: " << nimi << "\n Ika: " << ika << "\n";
    }

    //Setter nimi muuttujalle
    void setNimi(string nimi)
    {
        this->nimi = nimi;
    }

    //Getter nimi muuttujalle
    string getNimi()
    {
        return nimi;
    }

};

//Koira luokan luonti, jolle periytetään Elain
class Koira: Elain {

    string rotu;

public:

    Koira(string nimi,int ika,string rotu) : Elain(nimi,ika), rotu(rotu){}

    //Tulostaa koiran tiedot. Funktio syrjäyttää Elain-luokan tulostaTiedot-funktion
    void tulostaTiedot() {
        cout << "\n Nimi: " << nimi << "\n Ika: " << ika << "\n Rotu: " << rotu << "\n";
    }
    void hauku() {
        cout << "\n " << nimi << " sanoo hau hau!\n";
    }
};
```

```
//Susi-luokan määrittely, jolle periytetään Elain-luokka
class Susi : Elain {

    bool laumanjohtaja;

public:
    //Susi-luokan constructori
    Susi(string nimi, int ika, bool laumanjohtaja) : Elain(nimi, ika), laumanjohtaja(laumanjohtaja) {}

    //Tulostaa suden tiedot. Funktio syrjäyttää Elain-luokan tulostaTiedot-funktion
    void tulostaTiedot() {

        string onkoJohtaja;

        if (laumanjohtaja)
        {
            onkoJohtaja = "On";
        }
        else
        {
            onkoJohtaja = "Ei ole";
        }

        cout << "\n Nimi: " << nimi << "\n Ika: " << ika << "\n " << onkoJohtaja << " laumanjohtaja.\n";
    }

    void ulvo() {
        cout << "\n " << nimi << " ulvoo!";
    }
};

int main()
{
    //Luodaan uusi eläin (elukka1) ja annetaan sille nimeksi Elukka ja iäksi 10
    Elain *elukka1 = new Elain("Elukka",10);

    Koira *koira1 = new Koira("Musti",5,"Husky");

    Susi *susi1 = new Susi("Hopeanuoli",4,true);


    //Tulostetaan eläimien tiedot ja testataan ulvo ja hauku toimintoja
    elukka1->tulostaTiedot();
    koira1->tulostaTiedot();
    susi1->tulostaTiedot();

    susi1->ulvo();
    koira1->hauku();

    int a;
    cin >> a;
    return 0;
}
```

LIITE 2

OULUN AMMATTIKORKEAKOULU

Tatu Piippo

TILAUSTIEDOSTOJEN LUKEMINEN JA KIRJOITTAMINEN C#-SERVICE-OHJELMALLA

TILAUSTIEDOSTOJEN LUKEMINEN JA KIRJOITTAMINEN C#-SERVICE-OHJELMALLA

Tatu Piippo
Opinnäytetyö, osa 2
Kevät 2018
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistotuotanto

Tekijä: Tatu Piippo
Opinnäytetyön nimi: Tilaustiedostojen lukeminen ja kirjoittaminen C#-service-ohjelmalla
Työn ohjaaja: Pekka Alaluukas
Työn valmistumislukukausi ja -vuosi: Kevät 2018 Sivumäärä: 28

Tilaustenkäsittelyohjelma luotiin alun perin Procomp Solutions -yritystä varten, mutta tätä opinnäytetyötä varten siitä on poistettu Procomp Solutionsin asiakkaita koskevat tiedot sekä Procompin sisäisiä järjestelmiä käsittelevät kohdat. Tämä työ on siis toteutettu yleisemmältä näkökannalta.

Tavoitteena oli luoda C#-service-ohjelma, joka pystyy lukemaan sekä kirjoittamaan määrämittaisia sekä XML-tiedostoja. Määrämittaisellatiedostolla tarkoitetaan tekstitiedostoa, jonka tiedot ovat tietyn merkkimäärän mittaisia. Tiedostot sisältävät tilaustietoja, joiden tarkoitus on välittää tietoa kahden eri järjestelmän välillä.

Serviceä tehdessä hyödynnettiin Microsoftin dokumentaatiota service-ohjelman luomisesta, Procomp Solutionsissa opittuja tilauksenkäsittelymenetelmiä sekä Visual Studiossa olevia valmiita kirjastoja.

Service-ohjelmia voidaan hyödyntää monessa muussakin tarkoituksessa kuin pelkästään kahden järjestelmän väliseen tietojen siirtoon tiedostojen avulla. Servicejä voidaan myös hyödyntää komponentin tavoin niin, että jokin muu sovellus käyttää servicen toiminnallisuuksia kutsumalla sen funktioita.

Asiasanat: C#, ohjelmointi, toiminnanohjausjärjestelmät

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, software developing

---

Author: Tatu Piippo
Title of thesis: Reading and writing order files with a C#-service program
Supervisor: Pekka Alaluukas
Term and year when the thesis was submitted: Spring 2018 Number of pages: 28

The purpose of this thesis was to write a service-program that can read and write order files that are of specific length or in XML-format. This type of program was original done for the Procomp Solutions company but for the thesis, I removed customer specific information and any references to the Procomp Solutions own inner systems. So this was done in more of a general view of a file writing and reading program. The files contain order information and are used to convey information between two different systems.

When writing this program, micorosofts documentation was utilized as well as things learnt in Procomp Solutions and also Visual Studios programming libraries.

Service programs can be used in many other ways than just communicating between two systems with files. Services can also be used as a dedicated module for other programs with their own specific functions.

Keywords: C#, programming, enterprise resource planning

SISÄLLYS

1 JOHDANTO

Service-ohjelmat tai palveluohjelmat ovat tietokoneen taustalla ajettavia ohjelmia, joilla ei ole omaa käyttöliittymää. Service-ohjelmat voidaan myös laittaa käynnistymään automaattisesti tietokoneen käynnistyessä. Tämä tekee service-ohjelmista erityisen hyödyllisiä esimerkiksi palvelimille tai pitkään ajettaville sovelluksille, jotka eivät häiritse muiden käyttäjien työskentelyä. (Introduction to Windows Service Applications. 2017.)

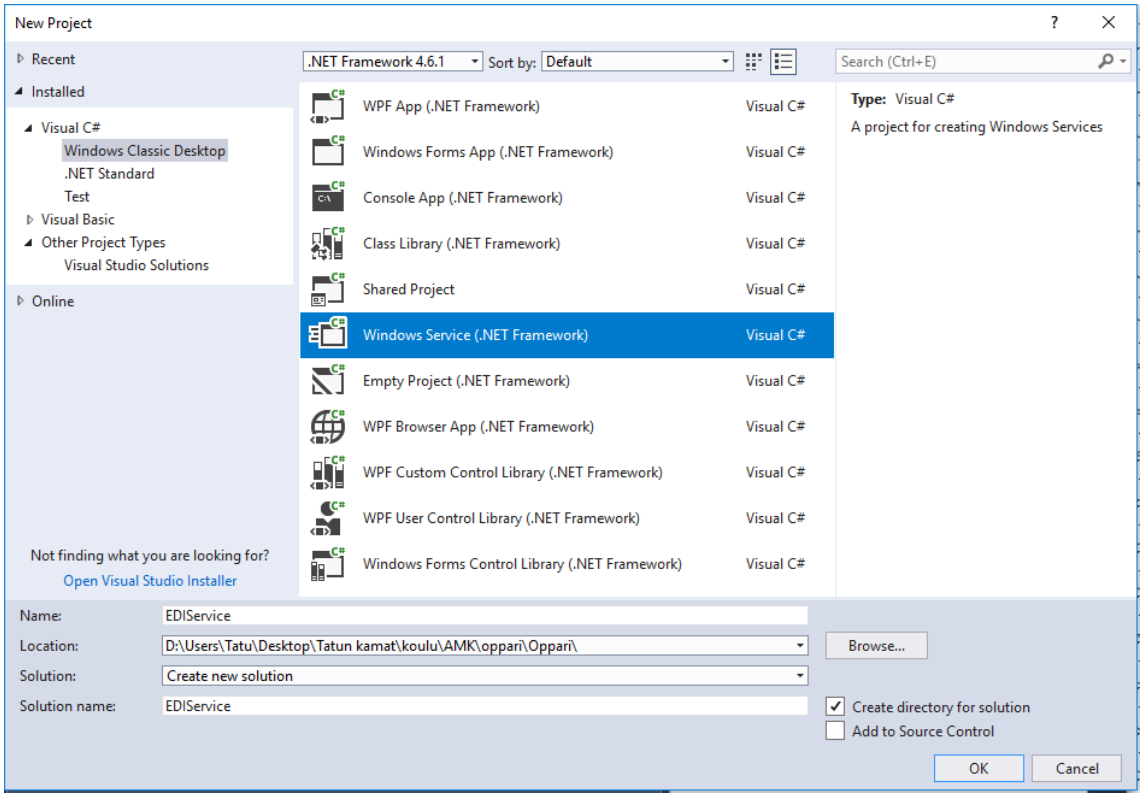
Service-ohjelmia pystytään hallitsemaan muiden ohjelmien avulla tai service-ohjelma voi olla täysin itsenäinen. Jos service-ohjelma on täysin itsenäinen, se voidaan käynnistää tai sammuttaa Windowsin omalta asennettujen palveluiden listalta. Listan pääsee näkemään kirjoittamalla Windowsin hakuun ”Services” tai ”Palvelut” riippuen Windowsin kieliasetuksista. Asennetut servicet voi myös nähdä tehtävienhallinnassa omalla välilehdellä.

Työn tarkoituksena on tutustua service-ohjelmiin sekä toteuttaa Windows service -ohjelma C#-ohjelmointikielellä. Service-ohjelmalle toteutetaan toiminnallisuus tiedostojen tietojen kirjoittaminen tietokantaan sekä tiedostoon kirjoittaminen kannasta löytyvillä tiedoilla. Tiedostojen luku- sekä kirjoitustoiminnallisuus toteutetaan määrämittaiselle tiedostolle sekä XML-formaatissa olevalle tiedostolle. Service-ohjelma katsoo omat asetuksensa tiedostosta, jossa määritellään tietokanta-asetukset sekä tiedoston luku- ja kirjoituskansiot. Luetut ja kirjoitetut tiedostot sisältävät tilaus tietoja. Service toteutetaan Microsoft Visual Studiolla.

Toteutin saman tyyppisen ohjelman Procomp Solutions -yritykselle, mutta opinnäytetyötä varten jätän poissa asiakaskohtaiset asiat sekä Procompin sisäisiä järjestelmiä käsittelevät asiat, joten tässä toteutettu service-ohjelma on kuvattu yleisemmältä näkökulmalta.

2 SERVICE-OHJELMAN ALUSTUS

Visual Studiosta löytyy valmis projektityyppi servicen tekemiseen (kuva 1). Kun valitsee Windows Service -projektityypin, Visual Studio luo automaattisesti tiedostot Program.cs sekä Service1.cs. Service1.cs-tiedostossa löytyy servicen rakentaja funktio sekä OnStart- ja OnStop-funktiot, joita kutsutaan, kun service käynnistetään tai sammutetaan. Program.cs tiedostossa määritetään ne ominaisuudet, joita tarvitaan service-ohjelmassa.

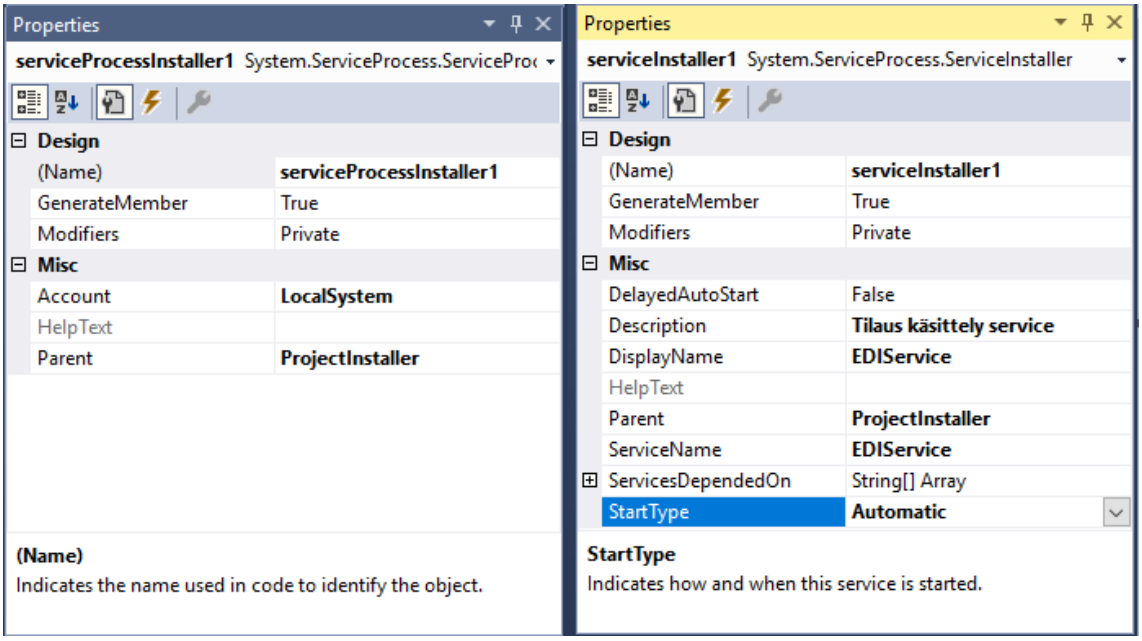


KUVA 1. Windows Service -pohjan valitseminen

2.1 Servicen asentaminen ja käynnistäminen

Kun pohja on luotu servicelle, katsotaan, miten se asennetaan ja käynnistetään. Servicestä luodaan .exe-tiedosto niin kuin tavallisesta ohjelmasta Visual Studion Build-valikon alta. Tätä .exe-tiedostoa ei kuitenkaan pystytä ajamaan suoraan, vaan se täytyy asentaa ensin. Service voidaan asentaa käyttämällä Windowsin InstallUtil-ohjelmistoa tai tekemällä servicelle erillinen asentajaohjelma. Tässä opinnäytetyössä ei tehdä erillistä asentajaohjelmaa.

Ennen kuin servicen pääsee asentamaan edes InstallUtil-ohjelmalla, täytyy sille lisätä asentaja projektiin. Asentaja sisältää servicen näytettävän nimen, kuvauksen, käynnistystavan sekä käyttäjän. Asentaja lisätään avaamalla Service1.cs designer-näkymässä ja valitsemalla hiiren oikeanpuoleisen menun alta ”Add installer”. Asentaja aukeaa automaattisesti design muodossa, josta löytyy ”serviceProcessInstaller1” sekä ”serviceInstaller1”. ServiceProcessInstaller1:n ominaisuuksista voidaan valita, millä käyttäjällä service asentuu. Tämä vaikuttaa siihen, mitä oikeuksia servicellä on. ServiceInstaller1:n ominaisuuksista voidaan valita servicen nimen, kuvauksen sekä käynnistystavan. (Kts. Kuva 2).

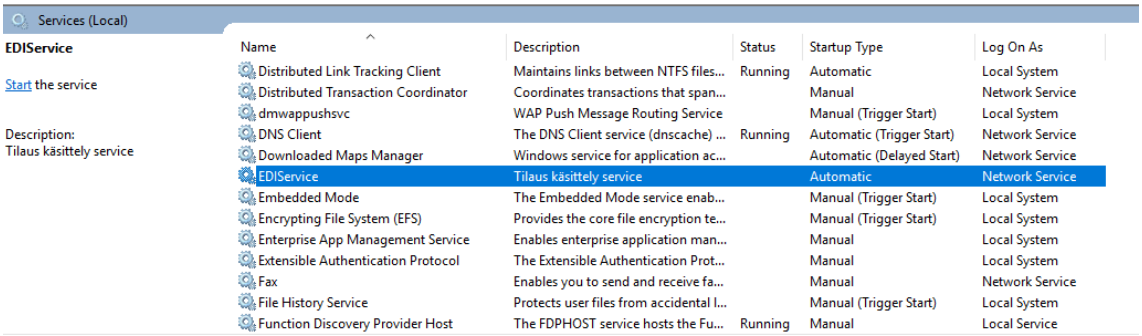


KUVA 2. Servicen ominaisuudet

Tietokantaan kirjoittamista varten service tarvitsee oikeudet internetin käyttämiselle. Ne saadaan asettamalla servicen käyttäjäksi joko NetworkService tai LocalSystem. Tässä työssä toteutetun servicen tulisi käsitellä tilauksia automaattisesti ilman, että siihen tarvitsee asentamisen jälkeen enää koskea, joten servicen käynnistystyypiksi valitaan ”Automatic”. Automatic -käynnistystyypillä service käynnistyy aina Windowsin käyttöjärjestelmän mukana. Servicelle on myös hyvä antaa kuvaava nimi ja selitys. Tämän jälkeen servicestä voidaan tehdä exe-tiedosto sekä asentaa se InstallUtil-ohjelmalla.

Servicen asentaminen InstallUtil-ohjelmalla tapahtuu Windowsin komentokehotteella tai kehittäjien komentokehotteella, joka tulee Visual Studion mukana. Käytetään kehittäjän komentokehotetta, sillä se ei vaadi sijaintipolkua InstallUtil-ohjelman ajamiseen.

Navigoidaan komentokehotteella kansioon, jossa exe-tiedosto sijaitsee, ja annetaan komento ”installutil EDIService.exe”. Mikäli virheitä ei tullut, service asentui onnistuneesti. Asennetun servicen löydät joko tehtävänhallinnasta tai palveluista(services).



KUVA 3. Asennettu service

2.2 Servicen testaaminen

Jos service ajetaan Visual Studiosta, Windows antaa virheen ja kertoo, että serviceä ei voida ajaa virheiden jäljittäjällä(debugger) vaan service pitäisi asentaa ensin. Tämä vaikeuttaa servicen testaamista erittäin paljon, mutta se on kuitenkin ohitettavissa.

2.2.1 Testaaminen ilman asentamista

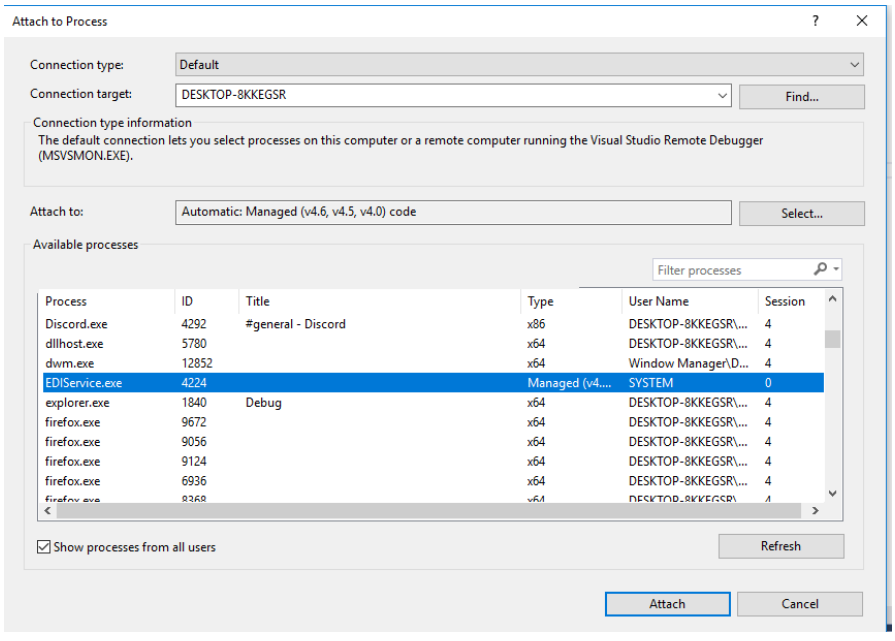
Virheilmoitus voidaan ohittaa muokkaamalla Program.cs-tiedostoa, niin että serviceä ei käynnistetä servicenä, vaan luodaan se normaalin objektin tavoin ja tehdään sille uusi funktio, jota kutsutaan (kuva 4). Tällä tavalla voidaan testata servicestä haluttua funktiota ilman, että sitä tarvitsee asentaa. Tässä tulee huomioida se, että servicen sisään rakennettuja Start- ja Stop-funktioita ei voida testata suoraan. Tätä keinoa ei mainita Microsoftin dokumentaatioissa, vaikka tämä on mielestäni kätevin keino testata service-ohjelman toimintoja.

```
namespace EDIService
{
    0 references
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        0 references
        static void Main()
        {
            /*ServiceBase[] ServicesToRun;
            ServicesToRun = new ServiceBase[]
            {
                new Service1()
            };
            ServiceBase.Run(ServicesToRun);*/
            Service1 service = new Service1();
            service.DoThing();
        }
    }
}
```

KUVA 4. Muokattu Program.cs

2.2.2 Testaaminen asentamisen jälkeen

Servicen asentamisen jälkeen sitä voidaan testata yhdistämällä Visual Studion virheiden jäljittäjä käynnissä olevaan serviceen. Tälläkään keinolla ei pystytä testaamaan servicen OnStart-funktiota, sillä servicen täytyy olla käynnissä, ennen kuin virheiden jäljittäjän voi siihen yhdistää.



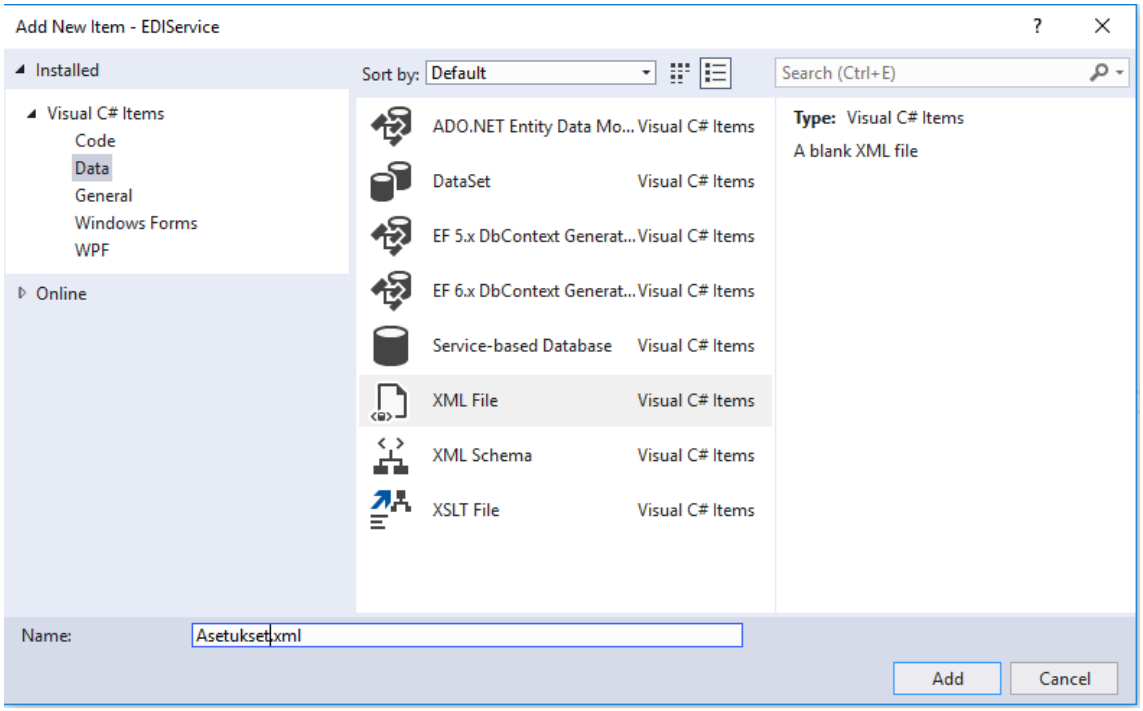
KUVA 5. Virheiden jäljittäjän yhdistäminen servicen prosessiin

3 TIEDOSTON LUKEMINEN

Tarkoituksena olisi, että service lukisi omat asetuksensa tiedostosta, sen lisäksi että se lukisi tilaustietoja. Asetukset määritellään XML-tiedostossa, josta ne puretaan talteen servicen käynnistyessä. Tilaustiedostot tulevat ulkoisesta järjestelmästä ja sisältävät tilauksen otsikkotiedot sekä eri määriä tilausrivejä, joissa ovat tuotteiden tiedot.

3.1 Asetusten lukeminen XML-tiedostosta

Luodaan projektille XML-tiedosto, josta asetukset löytyvät. XML-tiedostoja voi luoda tavallisella tekstieditorilla. Visual Studiossa niitä kuitenkin voidaan luoda suoraan niin, että ne näkyvät projektinäkymässä. (Kts. Kuva 6)



KUVA 6. XML-tiedoston lisäys

XML-tiedostoissa arvot laitetaan tagien sisälle. Tagi kertoo, mikä arvo on kyseessä, ja sen sisälle kirjoitetaan itse arvo esimerkiksi näin; <tagi>arvo</tagi>. Tagi-arvokokonaisuuksia kutsutaan nodeiksi. Tagi voi myös sisältää muita tageja sisällään. XML-tiedostoon lisätään kaikki, mitä mah-

dollisesti halutaan muuttaa helposti ilman, että tarvitsee itse koodiin koskea. Laitetaan sinne serviceä varten tietokanta-asetukset, kuinka usein service ajetaan, sekä kansiopolut tilaustiedostojen lukemista ja kirjoittamista varten.

Asetukset.xml | Service1.cs | Program.cs

```
<?xml version="1.0" encoding="utf-8" ?>
<Initialization>
  <!--Server where the database is located-->
  <Server>111.11.11</Server>
  <!--Name of the database-->
  <Database>Testi</Database>
  <!--User and password for the database-->
  <User>user</User>
  <Password>password</Password>
  <!--Determines if we are using the interval or the time of the day-->
  <UseInterval>1</UseInterval>
  <!--How long the service will wait before running again (minutes)-->
  <RunInterval>1</RunInterval>
  <!--Time of the day when the service will run-->
  <RunTime>12:42</RunTime>
  <!--Read file path -->
  <ReadFilePath>C:\Tilaus\luku</ReadFilePath>
  <!--Write file path-->
  <WriteFilePath>C:\Tilaus\kirjoitus</WriteFilePath>
</Initialization>
```

KUVA 7. XML-tiedoston rakenne

Valmiista asetustiedostosta luetaan siellä olevat arvot muuttujiin itse ohjelmassa. Visual Studiossa on valmiina kirjasto XML-tiedostojen lukua varten. Tämä helpottaa XML-tiedostojen lukemista huomattavasti.

Haetaan koodissa asetustiedosto samasta sijainnista, missä itse service sijaitsee. Ei siis laiteta suoraa kansiopolkua vaan haetaan polku sen mukaan, missä ohjelman exe-tiedosto on ajohetkellä. Tallennetaan tiedosto XmlDocument-objektiin, joka tulee Visual Studion XML-kirjaston mukana. Haetaan tästä XmlDocument-objectista kaikki ”Initialization”-tagin alla olevat nodet (tagi-arvokokonaisuudet) ja tallennetaan niiden arvot muuttujiin.

```
1 reference
private void ReadSettings()
{
    string configFilePath = Path.GetDirectoryName(Process.GetCurrentProcess().MainModule.FileName);

    XmlDocument configDoc = new XmlDocument();
    configDoc.Load(configFilePath + "\\" + "Asetukset.xml");

    //Gets the settings like server, database and filepaths
    XmlNodeList settings = configDoc.GetElementsByTagName("Initialization");
    foreach (XmlNode node in settings[0].ChildNodes)
    {
        string nodeValue = node.InnerText;
        switch (node.Name.ToLower())
        {
            case "server":
                server = nodeValue;
                break;
            case "database":
                database = nodeValue;
                break;
            case "user":
                user = nodeValue;
                break;
            case "password":
                password = nodeValue;
                break;
            case "useinterval":
                if (nodeValue.ToLower() == "true" || nodeValue == "1")
                    useInterval = true;
                else
                    useInterval = false;
                break;
            case "runinterval":
```

KUVA 8. Asetusten lukeminen

3.2 Määrämittaisen tilaustiedoston lukeminen

Servicen lukemat tilaustiedostot ovat määrämittaisia ja niissä on otsikkotason tiedot sekä yksi tai useampi tilausrivi. Otsikkotasolla sekä riveillä on oma alkutunniste, jotta tiedetään, milloin alkaa otsikkotaso ja milloin tilausrivi. Tunnisteen jälkeen tulee kyseisen tilausotsikon tai rivin tiedot ja ne luetaan siitä ottamalla tietyn verran merkkejä tietystä kohtaa. Nämä merkkien sijainnit sekä pituudet voisi myös tallentaa asetukset-tiedostoon.

Tyypillisen tilausotsikon tasolta löytyy ainakin tilauksen numero, tilauksen päivämäärä, asiakas, asiakkaan puhelinnumero sekä osoitetiedot. Tilausriveiltä löytyy tuote, tilattu määrä sekä hinta. Nämä eivät ole täysin samat tiedot kuin mitä Procomp Solutionsille tehty service luki vaan, nämä tiedot ovat yleiseltä näkökannalta, mitä tilaustiedot voisivat sisältää.

Toteutetaan omat luokat tilauksen otsikkotason tiedoille sekä tilausriveille, joihin voidaan sitten tallentaa luetut tiedostot. Laitetaan tilauksen otsikkotason luokkaan tavallisten arvojen lisäksi lista tilausriveistä. Näin saadaan suoraan yhdistettyä tilausrivit omiin tilauksiinsa.

```
1 reference
class Order
{
    public string orderNumber, orderDate, customer, customerPhoneNumber,
           town, street, postNumber;
    public List<OrderRow> orderRows;

    0 references
    public Order()
    {
        orderRows = new List<OrderRow>();
    }
}
```

KUVA 9. Tilauksen otsikkotason luokka

Määrämittaisen tiedoston lukeminen voidaan toteuttaa purkamalla tiedoston teksti string-muuttujaan ja ottamalla tästä string-muuttujasta tietty osa tekstiä jokaista arvoa varten. Olisi myös mahdollista, että tiedostossa olisi jokin merkki, joka erottaisi jokaisen arvon toisistaan, mutta tässä tilanteessa arvoilla ei ole erottajaa ne vaan ovat tietynmittaisia.

```
0 references
private void ReadFiles(string path)
{
    //Creates the folder if it doesnt exist
    if (!System.IO.Directory.Exists(path))
        System.IO.Directory.CreateDirectory(path);

    //Gets all the files that need to be read
    string[] filelist = Directory.GetFiles(path);

    if (filelist.Length > 0)
    {
        foreach (string file in filelist)
        {
            string fileText = System.IO.File.ReadAllText(file);
            Order order = GetOrderFromText(fileText);
            WriteOrderToDatabase(order);
        }
    }
}
```

KUVA 10. Tiedoston tekstin hakeminen

Kuvassa 11 etsitään tiedoston tekstistä ensin kohta, josta tilauksen otsikko alkaa käyttäen tilausotsikon tunnistetta. Kun tunniste on löydetty, voidaan todeta, että sen jälkeen tulee tilauksen tiedot. Puretaan siis substring-komennoilla tekstistä tietyt osiot niille kuuluville muuttujille. Näiden sijainnit ja pituudet voisi myös määritellä servicen asetukset-tiedostossa. Tilaustiedosto on määrämittainen, joten arvot eivät välttämättä ole niin pitkiä kuin niille on varattu merkkejä. Jos arvo ei täytä sille

määrättyä tilaa, sen loppuosa on täytetty tyhjillä merkeillä. Nämä tyhjät merkit saadaan karsittua pois Trim-komennolla.

```
private Order GetOrderFromText(string fileText)
{
    //Identifiers and lengths for the structures
    string OrderIdentifier = "$TO";
    string OrderRowIdentifier = "$TR";
    int orderStructureLength = 101;
    int orderRowStructureLength = 33;

    //Each file consists of one order and it's orderRows
    if (fileText.IndexOf(OrderIdentifier, 0) != -1)
    {
        string orderText = fileText.Substring(fileText.IndexOf(OrderIdentifier, 0), orderStructureLength);
        fileText = fileText.Remove(fileText.IndexOf(OrderIdentifier, 0), orderStructureLength);
        Order order = new Order();
        order.orderNumber = orderText.Substring(3, 6).Trim();
        order.orderDate = orderText.Substring(9, 6).Trim();
        order.customer = orderText.Substring(15, 20).Trim();
        order.customerPhoneNumber = orderText.Substring(35, 16).Trim();
        order.town = orderText.Substring(51, 20).Trim();
        order.postNumber = orderText.Substring(71, 10).Trim();
        order.street = orderText.Substring(81, 20).Trim();

        //Gets all the orderRows for the order
        while (fileText.Length > 0)
        {
            if (fileText.IndexOf(OrderRowIdentifier, 0) != -1)
            {
                string orderRowText = fileText.Substring(fileText.IndexOf(OrderRowIdentifier, 0), orderRowStructureLength);
                fileText = fileText.Remove(fileText.IndexOf(OrderRowIdentifier, 0), orderRowStructureLength);
                OrderRow orderRow = new OrderRow();
                orderRow.product = orderRowText.Substring(3,20).Trim();
                orderRow.amount = int.Parse(orderRowText.Substring(23,2));
                orderRow.price = double.Parse(orderRowText.Substring(25,8));
                order.orderRows.Add(orderRow);
            }
            else
            {
                break;
            }
        }
        return order;
    }

    return null;
}
```

KUVA 11. Tiedoston tekstin purkaminen objekteihin

4 TILAUKSEN KIRJOITTAMINEN TIETOKANTAAN

Kun tilaus on luettu ohjelman sisälle, tulisi se kirjoittaa tietokantaan. Tietokantaan kirjoittamista ja lukemista varten täytyy ensin luoda yhteys tietokantaan. Tietokanta-asetukset määriteltiin asetukset-tiedostossa, joten käytetään sieltä haettuja arvoja, kun yhdistetään tietokantaan. (Kts. Kuva 12).

```
SqlConnection connection;
SqlCommand cmd;
SqlTransaction transaction;
0 references
private void connectToServer()
{
    connection = new SqlConnection();
    connection.ConnectionString = "Data Source=" + server + ";Initial Catalog=" + database +
                                  ";User id=" + user + ";Password=" + password + ";";
    cmd = new SqlCommand();
    cmd.Connection = connection;
}
```

KUVA 12. Tietokantayhteyden luominen

SqlCommand-objektilla hoidetaan kaikki tietokantakyselyt, kuten esimerkiksi tietojen hakeminen tietokannasta tai sinne kirjoittaminen. SqlCommand-objektille voidaan myös määritellä transaktio, jolloin voidaan perua tietokantaan tehdyt muutokset, mikäli jotain menee pieleen.

4.1 Tilauksen kirjoittaminen tietokantaan

Tietokantaan kirjoittamisen voi tehdä kahdella tavalla käyttäen SqlCommand-objektia. Yksi tapa on kirjoittaa komento suoraan objektille arvojen kanssa. Tämä tapa altistaa kyselyn arvoille, jotka saattavat rikkoa kyselyn. Jos jokin arvoista esimerkiksi sisältää heittomerkin tai kaksi miinusmerkkiä, kysely menisi rikki. Toinen tapa kirjoittaa tietokantaan on antaa arvot kyselylle parametreina, jolloin ne automaattisesti menevät sql-kyselylle sopivaan muotoon. Huonompaa tapaa voi käyttää semmoisissa tilanteissa, joissa arvo ei voi sisältää merkkejä, jotka hajottaisivat kyselyn. Jos arvo on suoraan koodissa kirjoitettu eikä se tule ohjelman ulkopuolelta, siinä tilanteessa voisi käyttää tätä huonompaa tapaa.

```
string query = "INSERT INTO TilausTaulu (Tilausnumero,Tilauspvm,Asiakas,Puhnro,Paikkakunta,Osoite,Postinro) " +
               "VALUES (" + order.orderNumber +"," + order.orderDate + "," + order.customer + "," +
               order.customerPhoneNumber + "," + order.town + "," + order.street + "," + order.postNumber + ")";
cmd.CommandText = query;
cmd.ExecuteNonQuery();
```

KUVA 13. Huonompi tapa kirjoittaa tietokantaan arvoja

```
private bool WriteOrderToDataBase(Order order)
{
    //Opens the connection and begins a transaction
    connection.Open();
    transaction = connection.BeginTransaction("OrderTransaction");
    cmd.Transaction = transaction;
    string query = "INSERT INTO TilausTaulu (Tilausnumero,Tilauspvm,Asiakas,Puhnro,Paikkakunta,Osoite,Postinro) " +
                   "VALUES (@OrderNumber,@OrderDate,@Customer,@CustomerPhoneNumber,@Town,@Street,@PostNumber)";
    cmd.CommandText = query;
    cmd.Parameters.Clear();
    cmd.Parameters.AddWithValue("@OrderNumber",order.orderNumber);
    cmd.Parameters.AddWithValue("@OrderDate", order.orderDate);
    cmd.Parameters.AddWithValue("@Customer", order.customer);
    cmd.Parameters.AddWithValue("@CustomerPhoneNumber", order.customerPhoneNumber);
    cmd.Parameters.AddWithValue("@Town", order.town);
    cmd.Parameters.AddWithValue("@Street", order.street);
    cmd.Parameters.AddWithValue("@PostNumber", order.postNumber);

    int result = cmd.ExecuteNonQuery();

    //If no result are received something went wrong with the insert command
    if (result <= 0)
    {
        transaction.Rollback();
        connection.Close();
        return false;
    }
    if (WriteOrderRowsToDatabase(order))
    {
        transaction.Commit();
        connection.Close();
        return true;
    }
    else
    {
        transaction.Rollback();
        connection.Close();
        return false;
    }
}
```

KUVA 14. Tilausotsikon kirjoittaminen tietokantaan parametrien avulla

Ennen kuin tietokantaan kirjoittaminen aloitetaan, määritellään SqlCommand-objektille transaktio. Kyselyn suorituksen jälkeen tarkistetaan, että kysely palauttaa enemmän kuin 0. Tämä kertoo, että uuden rivin kirjoittaminen onnistui. Jos päivitettäisiin jo tietokannassa olevia arvoja, kysely palauttaisi muuttuneiden rivien määrän.

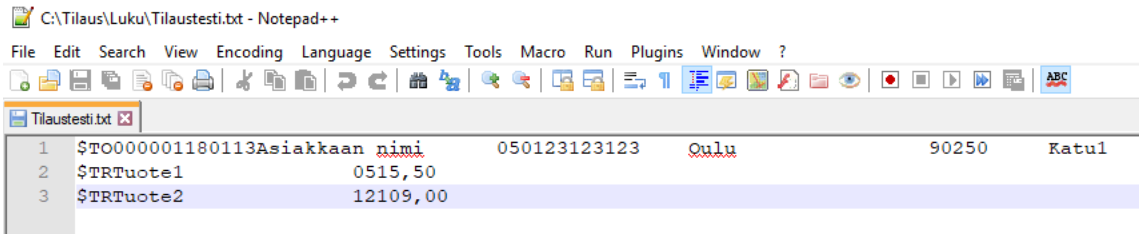
Jos kirjoittaminen ei onnistunut, transaktio voidaan perua, jolloin tietokantaan tehdyt muutokset peruuntuvat. Tätä voidaan hyödyntää myös tilausrivien kirjoittamisessa. Jos esimerkiksi tietokantaan on kirjoitettu tilausotsikko ja viisi tilausriviä, mutta kuudennessa tilausrivissä tapahtuukin virhe, voidaan perua tilausotsikon sekä viiden aiemman tilausrivin kirjoittamisen.

```
private bool WriteOrderRowsToDatabase(Order order)
{
    //Goes through all the rows in the order
    foreach (OrderRow orderRow in order.orderRows)
    {

        string query = "INSERT INTO TilausriviTaulu (Tilausnumero,Tuote,Maara,Hinta) " +
                       "VALUES (@OrderNumber,@Product,@Amount,@Price)";
        cmd.CommandText = query;
        cmd.Parameters.Clear();
        cmd.Parameters.AddWithValue("@OrderNumber", order.orderNumber);
        cmd.Parameters.AddWithValue("@Product", orderRow.product);
        cmd.Parameters.AddWithValue("@Amount", orderRow.amount);
        cmd.Parameters.AddWithValue("@Price", orderRow.price);

        int result = cmd.ExecuteNonQuery();

        //If no result are received something went wrong with the insert command
        if (result <= 0)
            return false;
    }
    //If all the rows were written successfully returns true
    return true;
}
```

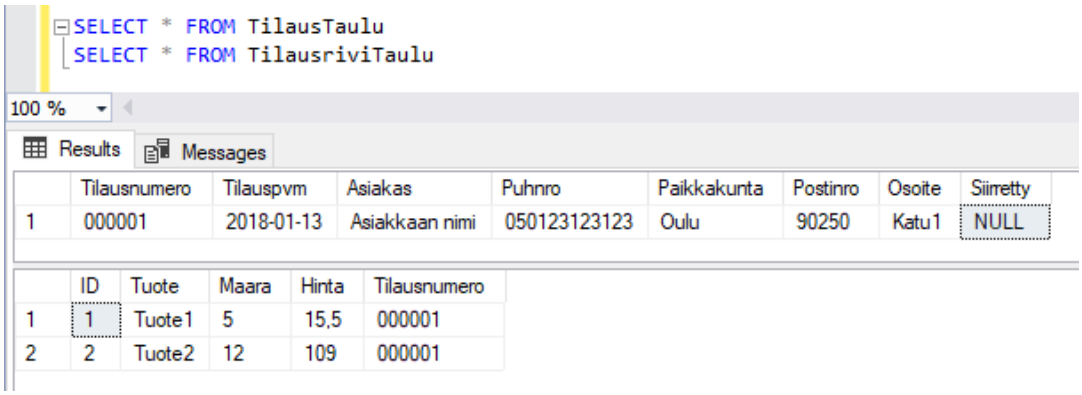
KUVA 15. Tilausrivin kirjoittaminen parametrien avulla

4.2 Esimerkkitiedoston kirjoitus tietokantaan

Kuvan 16 esimerkkitilaustiedostossa on tilaus, jolla on kaksi tilausriviä. Esimerkissä näkyy paljon tyhjää tilaa. Esimerkiksi Oulun jälkeen on runsaasti tyhjää. Tämä on sitä varten, että jos tulisi pidempiä kaupunkien nimiä, niitä ei tarvitsisi lyhentää.



Kuva 16. Esimerkkitilaustiedosto



	Tilausnumero	Tilauspvm	Asiakas	Puhnro	Paikkakunta	Postinro	Osoite	Siirretty
1	000001	2018-01-13	Asiakkaan nimi	050123123123	Oulu	90250	Katu1	NULL

	ID	Tuote	Maara	Hinta	Tilausnumero
1	1	Tuote1	5	15,5	000001
2	2	Tuote2	12	109	000001

KUVA 17. Esimerkkitilaus tietokannassa

5 TIEDOSTOON KIRJOITTAMINEN

Seuraavaksi katsotaan, miten tilaus saadaan kirjoitettua tiedostoksi. Esimerkin vuoksi kirjoitetaan XML-tiedosto sekä määrämittainen tekstitiedosto.

5.1 Tilausten hakeminen tietokannasta

Ennen kuin tilauksia kirjoitetaan tiedostoksi, täytyy niiden tiedot hakea tietokannasta. Tietojen hakemista varten ei tarvitse laittaa transaktiota päälle, sillä tietokannan tietoihin ei tule muutoksia. Tiedot voidaan siis suoraan hakea SELECT-kyselyllä.

Kuvassa 18 haetaan tilaustaulusta kaikki tilaukset, joiden siirretty kentän arvo on joko määrittelemätön (Null) tai 0. Tämä kenttä päivitetään tiedoston kirjoituksen onnistuttua ykköseksi, jotta voidaan tunnistaa, mitkä tilaukset on jo kirjoitettu tiedostoon. Kun tilaukset on haettu, haetaan jokaiselle tilaukselle sen omat tilausrivit käyttäen tilauksen numeroa kyselyn ehdoissa.

```
private List<Order> GetOrdersFromDatabase()
{
    //Gets all ordesr that have not been transferred
    List<Order> orders = new List<Order>();
    string query = "SELECT Tilausnumero, Tilauspvm, Asiakas, Puhnro, Paikkakunta, Osoite, Postinro " +
                   "FROM Tilaustaulu WHERE Siirretty IS NULL OR Siirretty = 0";
    cmd.CommandText = query;

    if (connection.State == ConnectionState.Closed)
        connection.Open();
    SqlDataReader reader = cmd.ExecuteReader();
    while (reader.Read())
    {
        Order order = new Order();
        order.orderNumber = reader["Tilausnumero"].ToString();
        order.orderDate = reader["Tilauspvm"].ToString();
        order.customer = reader["Asiakas"].ToString();
        order.customerPhoneNumber = reader["Puhnro"].ToString();
        order.town = reader["Paikkakunta"].ToString();
        order.street = reader["Osoite"].ToString();
        order.postNumber = reader["Postinro"].ToString();
        orders.Add(order);
    }
    reader.Close();
    //Gets the rows for the order
    foreach(Order order in orders)
    {
        string getRows = "SELECT Tuote, Maara, Hinta FROM TilausriviTaulu WHERE Tilausnumero = " + order.orderNumber;
        cmd.CommandText = getRows;
        reader = cmd.ExecuteReader();
        while (reader.Read())
        {
            OrderRow orderRow = new OrderRow();
            orderRow.product = reader["Tuote"].ToString();
            orderRow.amount = Int32.Parse(reader["Maara"].ToString());
            orderRow.price = Double.Parse(reader["Hinta"].ToString());
            order.orderRows.Add(orderRow);
        }
        reader.Close();
    }

    return orders;
}
```

KUVA 18. Tilausten hakeminen tietokannasta

5.2 Määrämittaisen tiedoston kirjoitus

Määrämittaisen tiedoston kirjoittaminen on melkeinpä helpompaa kuin sellaisen lukeminen. Tilauksen tiedot kirjoitetaan yhteen riviin, kannasta haettujen tietojen perusteella. Jos jokin arvoista on liian lyhyt, sille lisätään tyhjiä merkkejä, kunnes se täyttää sille varatun tilan. Taas jos arvo on liian pitkä, se pitää katkaista lyhyemmäksi. Tiedon katkaiseminen on tietenkin huono ratkaisu, joten tiedoille varatut tilat tulisi olla tarpeeksi suuret, jotta tiedot mahtuvat niihin riippumatta siitä, kuinka pitkiä ne ovat.

Kuvassa 19 kaikki tilausotsikon tiedot tallennetaan yhteen string-muuttujaan. Merkkijonoihin lisätään tyhjiä merkkejä PadRight-komennolla ja katkaistaan Substring-komennolla. Tilauksen rivien kirjoittamisessa voidaan käyttää samaa tapaa. Itse tiedoston kirjoittamiseen käytetään StreamWriter-luokkaa, joka on Visual Studiossa oleva valmis luokka tekstitiedoston kirjoittamista varten.

```
/// <summary>
///  Converts order data into one line of text with the correct length
/// </summary>
1 reference
private string ConvertOrderDataToText(Order order)
{
    string orderText = "";
    orderText += order.orderNumber.PadRight(6);
    orderText += order.orderDate;
    orderText += order.customer.PadRight(20).Substring(0, 20);
    orderText += order.customerPhoneNumber.PadRight(16);
    orderText += order.town.PadRight(20).Substring(0, 20);
    orderText += order.postNumber.PadRight(10).Substring(0, 10);
    orderText += order.street.PadRight(20).Substring(0, 20);
    return orderText;
}
```

KUVA 19. Tilausotsikon merkkijonon kirjoittaminen

```
private void WriteOrderFile(Order order, string fullFilePath)
{
    using (StreamWriter sw = System.IO.File.CreateText(fullFilePath))
    {
        //Creates the order header text and writes it to the file
        string orderText = OrderIdentifier;
        orderText += ConvertOrderDataToText(order);
        sw.WriteLine(orderText);

        //Creates the order rows for the orders
        foreach (OrderRow orderRow in order.orderRows)
        {
            string orderRowText = OrderRowIdentifier;
            orderRowText += ConvertOrderRowDataToText(orderRow);
            sw.WriteLine(orderRowText);
        }
    }
}
```

KUVA 20. StreamWriter-objektilla kirjoittaminen

Ensiksi aloitetaan käyttämään StreamWriter-luokkaa, luomalla se halutulle tiedostopolulle. Polku sisältää myös itse kirjoitettavan tiedostonnimen. Kutsutaan sitten StreamWriter-objektin WriteLine-komentoa tilauksen merkkijonolla sekä jokaisella tilausrivillä, jolloin niiden tiedot tulevat tiedostoon omiksi riveikseen.

5.3 XML-tiedoston kirjoitus

XML-tiedoston etuna määrämittaiseen tiedostoon on se, että ei tarvitse asettaa kiinteää rajaa arvojen pituuksille. Saman edun saisi myös lisäämällä erottimet tekstitiedostossa oleville arvoille.

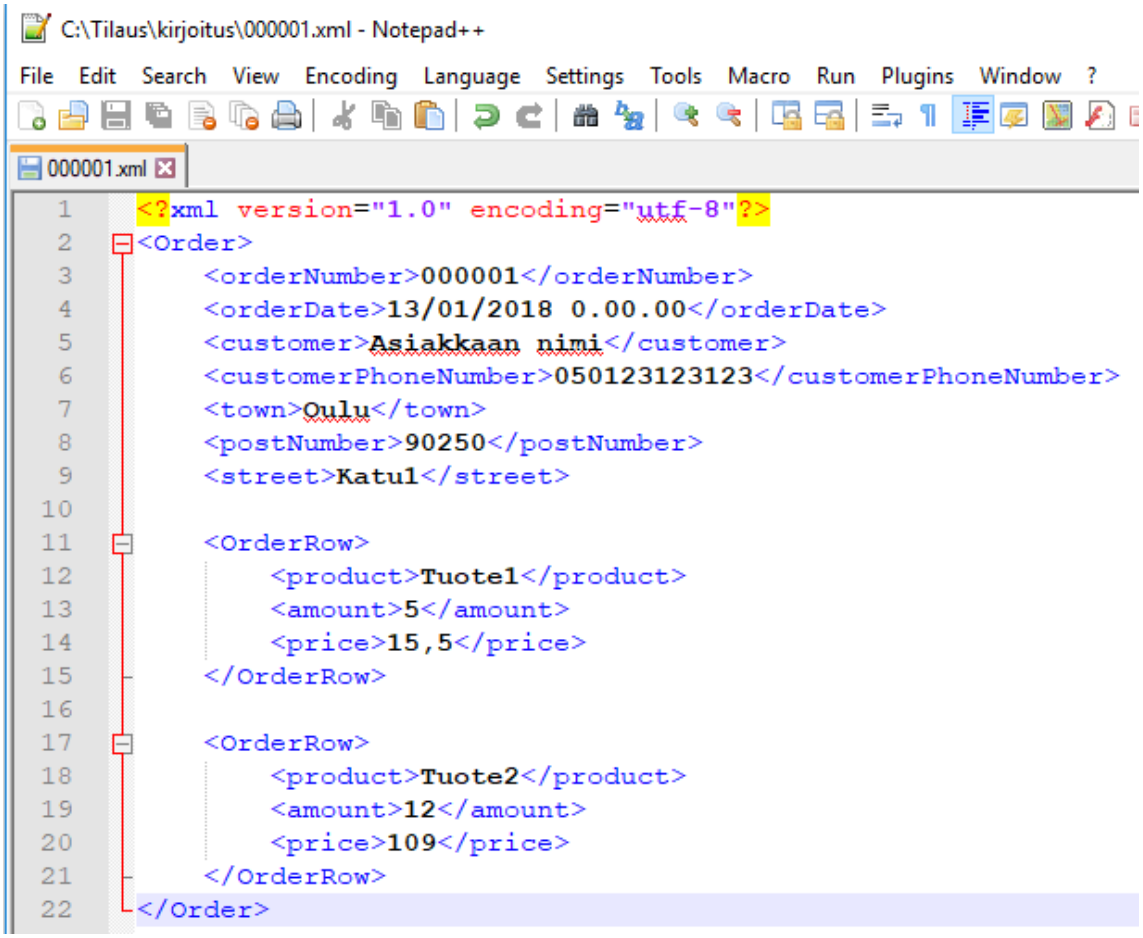
XML-tiedosto koostuu tageista, niin kuin aiemmin mainittiin, ja arvon loppumisen voi todeta siitä, että tagi suljetaan.

XML-tiedostoon kirjoittamiseen voidaan käyttää XmlWriter-luokkaa. Tagit voisi myös käsin kirjoittaa tavallisella tiedoston kirjoittajalla, jota käytettiin määrämittaisen tiedoston kirjoittamiseen, mutta XmlWriter-luokalla voidaan kätevämmin lisätä tagit arvoille.

Kirjoitetaan ”Order”-tagin (element koodissa) sisälle tilauksen otsikkotason arvot eli tilauksen numero, asiakas yms. Otsikkotason jälkeen luodaan uudet tagit tilausriveille. Tilausrivi tagien sisälle sitten niiden tiedot.

```
private void WriteXmlFile(Order order, string fullFilePath)
{
    XmlWriter xmlWriter = XmlWriter.Create(fullFilePath);

    xmlWriter.WriteStartDocument();

    xmlWriter.WriteStartElement("Order");//Order tag starts

    //Writing the values of the order
    xmlWriter.WriteElementString("orderNumber", order.orderNumber);
    xmlWriter.WriteElementString("orderDate", order.orderDate);
    xmlWriter.WriteElementString("customer", order.customer);
    xmlWriter.WriteElementString("customerPhoneNumber", order.customerPhoneNumber);
    xmlWriter.WriteElementString("town", order.town);
    xmlWriter.WriteElementString("postNumber", order.postNumber);
    xmlWriter.WriteElementString("street", order.street);

    //Writing the order rows
    foreach (OrderRow orderRow in order.orderRows)
    {
        xmlWriter.WriteStartElement("OrderRow");
        xmlWriter.WriteElementString("product", orderRow.product);
        xmlWriter.WriteElementString("amount", orderRow.amount.ToString());
        xmlWriter.WriteElementString("price", orderRow.price.ToString());
        xmlWriter.WriteEndElement();
    }

    xmlWriter.WriteEndElement(); //Order tag ends
    xmlWriter.WriteEndDocument();
    xmlWriter.Close();
}
```

KUVA 21. XML-tiedoston kirjoittaminen



KUVA 22. XML-tiedosto rivitetty luettavammaksi

6 SERVICEN AJAMINEN TIETYIN AIKAVÄLEIN

Kun servicelle on toteutettu kaikki sille haluttu toiminnallisuus, automatisoidaan sen ajaminen. Servicen katsoo asetuksistaan, kuinka usein sen toiminnot suoritetaan. Asetuksissa määritellään joko kellonaika päivästä tai minuuttiväli, milloin service tulisi ajaa.

```
0 references
protected override void OnStart(string[] args)
{
    //Sets the timers time depending if its a time of date or just interval in minutes
    if (useInterval)
    {
        timer.Interval = runInterval * 60000;
    }
    else
    {
        scheduleTime = DateTime.Parse(runTime);
        if (scheduleTime < DateTime.Now)
            scheduleTime = scheduleTime.AddDays(1);
        timer.Interval = scheduleTime.Subtract(DateTime.Now).TotalSeconds * 1000;
    }
    timer.Elapsed += new ElapsedEventHandler(ReadAndWriteOrders);
    timer.Start();
}
```

KUVA 23. Ajastimen asettaminen servicen käynnistyttyä

Ajastukseen käytetään timer-objektia. Timer-objektiin voidaan määritellä millisekunteina, milloin se ajaa halutun funktion, joten asetuksissa annetut minuutit kerrotaan luvulla 60 000, jotta saadaan haluttu odotusaika.

Kellonajan mukaan ajaminen vaatii hieman monimutkaisemman asettamisen. Ensiksi puretaan asetuksissa määritelty kellonaika DateTime-tyyppiseen muotoon. DateTime-objektiin tallentuu kellonaika millisekunteina. Verrataan DateTime-objektin kellonaikaa tämän hetkiseen kellonaikaan ja asetetaan millisekunnit sen mukaan. Kun ohjelma on yhdesti ajettu kellonaika-asetuksella, asetetaan millisekunnit uudestaan niin, että ohjelma ajetaan päivän välein (Kts kuva 24).

```
1 reference
private void ReadAndWriteOrders(object source, ElapsedEventArgs e)
{
    //Reads the files
    try
    {
        ReadFiles(readFilePath);
        List<Order> orders = GetOrdersFromDatabase();
        foreach (Order order in orders)
        {
            WriteFile(order);
        }
    }
    catch (Exception exp)
    {
        Debug.Print(exp.Message);
    }

    //Sets the timer to elapse the next day on the same time
    if (!useInterval && timer.Interval != 24 * 60 * 60 * 1000)
    {
        timer.Interval = 24 * 60 * 60 * 1000;
    }
    scheduleTime = DateTime.Now;
}
```

KUVA 24. Ajastimen kutsuma funktio

7 YHTEENVETO

Service-ohjelmien tekeminen on hieman erilaista tavallisiin ohjelmiin nähden. Esimerkiksi serviceistä puuttuu oma käyttöliittymä kokonaan. Niissä on kuitenkin omat etunsa tavalliseen ohjelmaa verrattuna. Service-ohjelmat sopivat hyvin esimerkiksi palvelimille tai taustalla ajettaviksi ohjelmiksi, jotka eivät vaadi käyttäjältä toimenpiteitä.

Tässä opinnäytetyössä toteutettua serviceä voidaan käyttää kahden eri järjestelmän väliseen kommunikointiin, ilman että ne ovat suoraan toisiinsa yhteyksissä. Tätä dokumenttia voidaan myös käyttää opastuksena service-ohjelman tekemiseen, vaikka lopullinen toiminnallisuus olisikin poikkeava. Tässä toteutettuja tiedostojen luku- ja kirjoitusominaisuuksia sekä tietokannan käsittelyä voidaan hyödyntää muissakin ohjelmissa, joko käyttämällä koodia pohjana tai tekemällä servicestä moduulin, jonka toiminnallisuuksia muut ohjelmat voisivat sitten kutsua.

Servicen toteuttaminen onnistui hyvin. Jokaisen servicen eri toiminnallisuuden toteuttamisessa tuli opittua uusia asioita. Näistä suurimpana kuitenkin oli service-ohjelmista oppiminen yleisesti. Aiempaa käsitystä serviceistä ei ollut ja niiden ymmärtäminen avaa monia uusia mahdollisuuksia eri järjestelmien kehitystä varten.

LÄHTEET

Introduction to Windows Service Applications. 2018. Microsoft. Saatavissa: https://docs.microsoft.com/en-us/dotnet/framework/windows-services/introduction-to-windows-service-applications. Hakupäivä 20.12.2017