

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Ohjelmointiparadigma

- **Ohjelmointiparadigma** on ohjelmointikielen taustalla oleva **tapa ajatella** ja **mallintaa** annettuun ongelmaan ratkaisu.
- Ohjelmointiparadigmat eroavat toisistaan siinä, millaisista eri osista ohjelma rakentuu (esim. tietorakenteista, luokista, olioista, funktioista, muuttujista jne.), miten kontrollin, tapahtumien, ohjelman eteneminen yms. esitetään.
- Usein ohjelmointikielen ajatellaan noudattavan yhtä paradigmaa, mutta kieli voi tukea useitakin paradigmoja, jolloin sen sanotaan olevan *moniparadigmainen*.
- Ohjelmointiparadigmoja:
 - Deklaratiivinen ohjelmointi, Epärakenteellinen ohjelmointi, Funktionaalinen ohjelmointi, Geneerinen ohjelmointi, Imperatiivinen ohjelmointi, Logiikkapohjainen ohjelmointikieli, **Olio-ohjelmointi**, Proseduraalinen ohjelmointi, Prototyyppipohjainen ohjelmointi, Reaktiivinen ohjelmointi, Rinnakkaisohjelmointi, Strukturoitu ohjelmointi ...

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Ohjelmointiparadigmoja

- Alla ohjelmointiparadigmoja linkkeineen.
- Deklaratiivinen ohjelmointi: https://fi.wikipedia.org/wiki/Deklaratiivinen_ohjelmointi
- Epärakenteellinen ohjelmointi: https://fi.wikipedia.org/wiki/Ep%C3%A4rakenteellinen_ohjelmointi
- Funktionaalinen ohjelmointi: https://fi.wikipedia.org/wiki/Funktionaalinen_ohjelmointi
- Geneerinen ohjelmointi: https://fi.wikipedia.org/wiki/Geneerinen_ohjelmointi
- Imperatiivinen ohjelmointi: https://fi.wikipedia.org/wiki/Imperatiivinen_ohjelmointi
- Logiikkapohjainen ohjelmointikieli: https://fi.wikipedia.org/wiki/Logiikkapohjainen_ohjelmointikieli
- **Olio-ohjelmointi:** <https://fi.wikipedia.org/wiki/Olio-ohjelmointi>
- Proseduraalinen ohjelmointi: https://fi.wikipedia.org/wiki/Proseduraalinen_ohjelmointi
- Prototyyppipohjainen ohjelmointi: https://fi.wikipedia.org/wiki/Prototyyppipohjainen_ohjelmointi
- Reaktiivinen ohjelmointi: https://fi.wikipedia.org/wiki/Reaktiivinen_ohjelmointi
- Rinnakkaisohjelmointi: <https://fi.wikipedia.org/wiki/Rinnakkaisohjelmointi>
- Strukturoitu ohjelmointi: https://fi.wikipedia.org/wiki/Strukturoitu_ohjelmointi

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

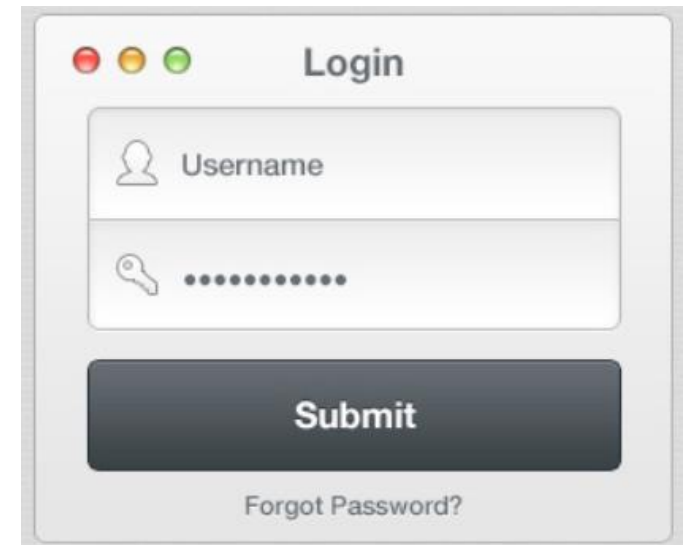
Olio-ohjelmointi

- Olio-ohjelmointi (engl. *Object-oriented programming*, OOP) on ohjelmoinnin lähestymistapa, jossa ohjelmointi jäsennetään olioiden **yhteistoimintana**.
- Olio-ohjelma muodostuu kokoelmasta **yhteistyössä** toimivia oliota, kun taas perinteinen proseduraalinen tietokoneohjelma on funktioiden, muuttujien ja tietorakenteiden joukko.
- Olio-ohjelmoinnissa **tieto** ja sitä käsittelevä **toiminnallisuus** kootaan luokkarakenteeksi. Luokista luodaan olioita.
- Olio käsittelee sisältämäänsä tietoa ja voi vastaanottaa viestejä ja lähettää tietoa muille olioille. Jokainen olio voidaan nähdä itsenäisenä pienenä koneena, jolla on tietty rooli tai vastuu.
- Suosittuja olio-ohjelmointia tukevia kieliä ovat mm. C++, C#, Java, Python ja Visual Basic.
- Olio-ohjelmoinnin neljä tärkeää peruseriaatetta: kapselointi, tiedon kätkeyminen, periytyminen ja polymorfismi.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: uudelleenkäytettävyys ja ylläpito

- Oikein käytettynä olio-ohjelmointi nopeuttaa kehitystyötä, vähentää redundanssia, ohjelmointivirheitä ja helpottaa merkittävästi ohjelmistojen ylläpitoa. Nykyaikaisten tietokoneohjelmien kehittämisessä olio-ohjelmointi on yksi tärkeimmistä ohjelmointikielistä.
- Esimerkiksi valmis ohjelmistokomponentti (**esim. kirjautumisliittymä**) voidaan ottaa mukaan mihin tahansa sovellukseen, kunhan ohjelmoija tietää miten komponentin luokkia, ja sen tietoja ja toimintoja käytetään. Komponentin sisäistä toteutusrakennetta ei ohjelmoijan välttämättä tarvitse tietää.
- Uudelleenkäytettävä komponentti tarjoaa **ulkoisen rajapinnan**, jonka kautta komponenttia voidaan käyttää.
- Komponentin/luokan toteuttaja ei puolestaan välttämättä tiedä, mihin luokkaa käytetään, hän vain toteuttaa luokan tietoineen, toimintoineen ja yhteyksineen.
- Uudelleenkäytettävien komponenttien käyttö nopeuttaa ja helpottaa testaamista ja ylläpitoa.



IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: luokka

- Luokka kuvaa mitä luokasta tehdyllä oliolla voi tehdä. *Luokka siis määrittelee olion tiedot, toiminnan mahdollisuudet ja yhteydet muihin luokkiin ja olioihin.*
- Luokka on yhteenliitetty joukko erilaisia **jäsenmuuttujia**, luokan toiminnallisuudesta vastaavia **jäsenfunktioita** ja lisäksi luokalla on **yhteyksiä muihin luokkiin** ja olioihin erilaisilla olio-ohjelmointiin kuuluvilla yhteysmuodoilla.
- Luokkien välille voidaan rakentaa yhteyksiä olio-ohjelmoinnissa eri tavoin. Tällä opintojaksolla keskitytään kahteen käytetyimpään yhteysmuotoon: vahva kooste ja periytyminen.
- Luokka ei tee sovelluksessa mitään, vaan luokkaan rakennetut tiedot, toiminnallisuuden ja yhteydet saadaan sovelluksessa tekemään asioita vain **luomalla luokasta olio**.
- Luokasta voidaan tarvittaessa tehdä useita olioita.
- Luokan koodi sijaitsee RAM-muistin koodialueen muistiosassa. Tämän voi todeta esim. debuggaamalla ohjelmaa. Debuggauksesta tulee lisää myöhemmin.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: luokan jäsenfunktio ja -muuttuja

- Luokan jäsenmuuttujien ja –funktioiden määrittelyn yhteydessä käytetään **tiedon kätkentään** liittyviä varattuja sanoja **public** (julkinen), **protected** (suojattu, liittyy perintään) ja **private** (yksityinen). *Niillä hallitaan luokan osien näkyvyyttä ohjelmassa luokkaa käytettäville olioille.*

Jäsenmuuttuja (=tietojäsen, attribuutti, muuttuja ...)

- Tietoabstraktion saavuttamiseksi luokan määrittelyssä käytetään tiedon kätkentää. Luokan jäsenmuuttujat ovat oletusarvoisesti **yksityisiä** (private), koska tietoja halutaan käsitellä **vain luokan jäsenfunktioissa**.
- Olio-ohjelmoinnin yksi perusperiaate on, että luokan jäsenmuuttujia käsitellään vain luokan jäsenfunktioissa.
- Muuttujat, joihin luokan olion on tarpeellista päästä käsiksi suoraan ilman jäsenfunktioita, määritellään julkisiksi (**public**). **Tätä käytetään harvoin!**

Jäsenfunktio (=metodi, operaatio, funktio ...)

- Luokan jäsenfunktioilla toteutetaan ohjelman toiminnallisuus ja käsitellään esimerkiksi jäsenmuuttujia.
- Luokan jäsenfunktiot ovat **oletusarvoisesti julkisia** (public), joita luokasta luodut oliot kutsuvat. Myös luokan kanssa yhteydessä olevat toiset luokat voivat kutsua **public** osassa olevia jäsenfunktioita.
- Perintä yhteyden ollessa kyseessä luokan jäsenfunktiot voivat olla **protected** tasolla.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

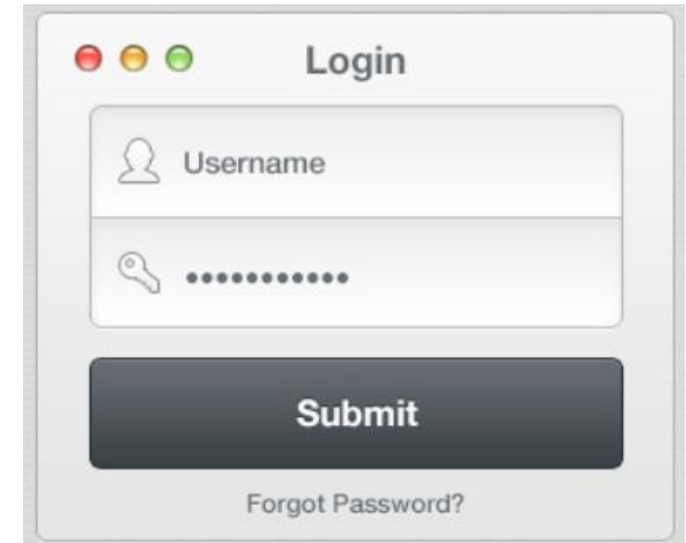
Olio-ohjelmointi: olio

- Jokainen olio kuuluu johonkin luokkaan.
- Olio on luokan ilmentymä, eli olio luodaan luokasta. Olio on se, joka suorittaa kaikki toiminnallisuudet sovelluksessa, käyttäen niitä rakenteita/määrittelyksiä joita luokassa on (tiedot, toiminnot, yhteydet ...).
- Olio käyttää siis luokan jäsenmuuttujia, -funktiota ja on yhteydessä eri yhteystavoilla (vahva kooste, periytyminen jne.) muihin olioihin.
- Luokasta luotu olio käyttää RAM muistin pino- tai kekomuistin muistiosaa sen mukaan, miten ohjelmassa olio luodaan. Tämän voi todeta esim. debuggaamalla ohjelmaa.
- Muistinhallintaan liittyvistä asioista tulee esimerkkejä, harjoituksia, koti- ja lisätehtäviä. Muistinhallintaan palataan myöhemmin.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: ohjelmistokomponentti

- Olio-ohjelmoinnissa ohjelmistokomponentti on itsenäinen ja uudelleenkäytettävä sovelluksen osa, esim. kirjautumisliittymä.
- Tarkoitus on siis tarjota käyttöön eräänlainen valmis ohjelmoinnin rakennuspalikka, jota voi uudelleenkäyttää eri sovellusten rakentamiseen.
- Komponentti on luokkien joukko.
- Komponentilla on aina tietty tehtävä, joka on luokkien avulla määritelty.
- Komponenttia käytetään komponentin luokista luoduilla olioilla.



Login

Username

.....

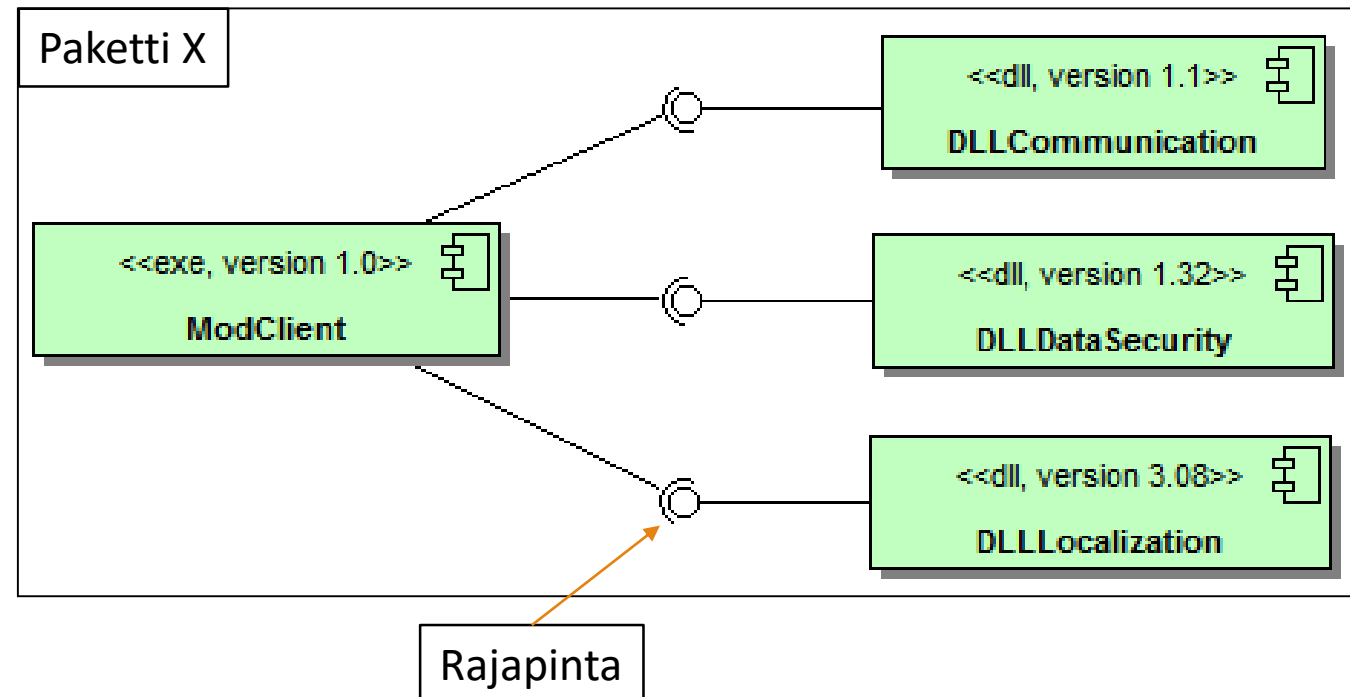
Submit

Forgot Password?

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: ohjelmistokomponentti, UML mallinnuskielen komponenttikaavio

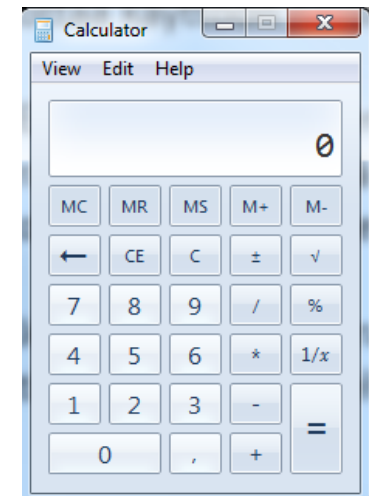
- **Komponenttikaavio** avulla kuvataan ohjelmistokomponentit ja niiden väliset suhteet.
- Kaavioon kuuluvat piirtoelementit ovat **paketit**, **komponentit** ja **rajapinnat**.
- Komponenttikaaviolla voidaan esittää ohjelmistokomponenttien sijainti eri **paketeissa**.
- Komponenttikaaviota voidaan käyttää mm.
 - ohjelmiston yleisen rakenteen hahmottamisessa.
 - suunnittelu- ja toteutusvaiheessa ohjelmiston komponenttirakenteen kuvaamisessa.
 - versionhallinnan työkaluna, kuvaamaan tiettyyn ohjelmistoversioon kuuluvat ohjelmistokomponentit.
 - eri testausvaiheissa, kun ohjelmiston joku ohjelmistoversio halutaan kasata testaukseen.



IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: luokan määritteleminen

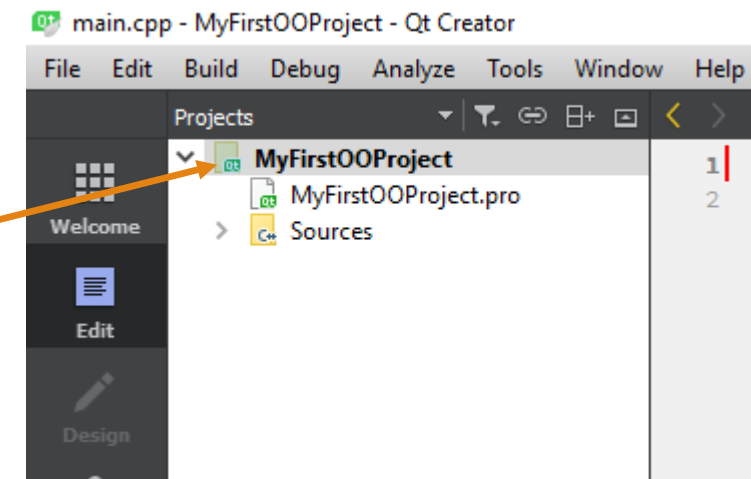
- Ohjelman tehtävä on yleensä ratkaista jokin todellista maailmaa kuvaava ongelma. Olio-ohjelmoinnissa voidaan ottaa käyttöön luokissa tietoja ja toimintoja, jotka vastaavat todellisuutta, ja nimetä asioita sen mukaan.
- Esimerkiksi jos rakennetaan jotain sulautettua järjestelmää johon kuuluu näppäimistö, anturit, erilaiset lukijalaitteet ja/tai näyttö, niin voidaan rakentaa/määritellä näitä vastaavat luokat, ja kapseloida käsiteltävä tieto eri käsitteiden omiin luonnollisiin osiin. Eli ohjelmassa rakennettaisiin luokat/luokkien kokonaisuus eri laitteille, ja näille omat tiedot, toiminnallisuudet ja yhteydet.
- Jos rakennetaan graafista käyttöliittymää, niin käyttöliittymän eri osat ovat omia luokkia.
- Esimerkiksi laskin sovelluksen eri graafiset elementit (painikkeet, valikot, näyttökenttä, ...) ovat eri luokista tehtyjä olioita, jotka suorittavat ohjelmassa toimintoja.
- Vastaavasti jos sovelluksessa on tietokanta, niin sovellukseen pitää rakentaa omat luokat ja oliot hoitamaan tietokannan käsittely.



IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: projektin ja luokan luonti QtCreatorissa

- Siirry **QtCreatoriin** ja sulje työkalussa oleva projekti ja kaikki editorissa olevat ikkunat (File->Close All Projects and Editors).
- Esimerkki 1: Luo uusi projekti nimeltä **MyFirstOOPProject**
 - Valitse **Qt Creator** työkalun päävalikosta **File->New File or Project ...**
 - Valitse projektiksi **Non-Qt Project->Plain C++ Application** ja klikkaa **Choose**
 - Anna projektille nimeksi **MyFirstOOPProject** ja valitse talletuskansio (esim. C:\data) ja klikkaa **Next**
 - Ikkunassa **Define Build System** klikkaa **Next** (riippuen versiosta tämä ikkuna tulee esille)
 - **Kit Selection** ikkunassa anna oletusvaihtoehdon olla valittuna ja klikkaa **Next**
 - **Project Management** ikkunassa klikkaa **Finish**, niin projekti luodaan
 - Työkaluun aukeaa **main.cpp** tiedosto. Poista **main.cpp** koodi kokonaan.
- Lisätään projektiin uusi luokka
 - Valitse projekti-ikkunassa **MyFirstOOPProject** aktiiviseksi klikkaamalla projektin nimen päällä hiiren vasenta painiketta
 - Tämän jälkeen klikkaa hiiren oikeaa ja valitse listalta **Add New ...**
 - Valitse **C++ -> C++ Class** ja klikkaa **Choose**
 - Kirjoita kohtaan **Class Name** luokalle nimeksi **MyFirstClass** ja klikkaa **Next**
 - Lopuksi klikkaa **Finish**, jonka jälkeen luokka lisätään projektiin



IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: luokan rakenne/määrittely (.h tiedosto)

- Avaa projekti-ikkunasta kohdat **Headers** ja **Sources**. Näistä hakemistoista löytyvät luokan tiedostot **myfirstclass.h** ja **myfirstclass.cpp**
- Tiedostossa **myfirstclass.h** on luokan **MyFirstClass** määrittely. Luokka kuvaa mitä luokasta tehdyllä oliolla voi tehdä. *Luokka siis määrittelee olion toiminnot (=jäsenfunktiot) ja käsiteltävän datan (=jäsenmuuttujat)!*
- Kaksoisklikkaa tiedostoa **myfirstclass.h**
 - Luokan määrittely alkaa aina **class** sanalla. Tämän jälkeen tulee luokan nimi **MyFirstClass**.
 - Luokan sisältö kuvataan aaltosulkeiden välissä ja sisällön määrittely päättyy puolipisteeseen.
 - Luokan määrittelyyn kuuluu jäsenfunktioiden ja –muuttujien esittely eri näkyvyys osissa (public, private, protected)
 - Tässä työkalussa siellä on valmiina vain **public** osa. Muut osat lisätään sinne tarvittaessa itse.
 - Poista **public** osan alta rivi **MyFirstClass();** (tyhjennetään luokka jäsenfunktioista harjoituksen vuoksi)
- Rivit **#ifndef**, **#define** ja **#endif** tulevat automaattisesti jokaiseen luokkaan
 - Luokan otsikkotiedosto saattaa esiintyä samassa käännösyksikössä **#include**-komennon yhteydessä useaan kertaan. Tällainen aiheuttaa saman luokan määrittelyn monta kertaa samassa käännösyksikössä. Saman tunnuksen voi esitellä monta kertaa, mutta määritellä vain kerran yhdessä käännösyksikössä.
 - Käännösvirheet voidaan estää esikäsittelijän direktiivillä (preprocessor directive) **#ifndef** (if not defined).
 - Kunkin luokan määrittelylle annetaan yksilöllinen tunniste **#define** direktiivillä. Tässä tapauksessa **#define MYFIRSTCLASS_H**
 - Esikäsittelijä tutkii onko määrittely jo sisällytetty käännettyyn koodiin. Jos ei ole, niin esikäsittelijä sisällyttää määrittelyn.
 - Määrittelyn loppuun on kirjoitettava **#endif**.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: luokan toteutuksen rakenne (.cpp tiedosto)

- Tiedostossa **myfirstclass.cpp** on luokan **MyFirstClass** jäsenfunktioiden toteutukset. Avaa tiedosto editoriin ja katso mitä sinne on valmiiksi rakentunut.
- Poista tiedostosta kaikki muut rivit paitsi **#include ...** rivi. *Nyt luokka on tyhjä määrittämisistä (.h) ja toteutuksista (.cpp)!*
- Luokan rakenteeseen (.h tiedosto) viittaava **#include** lause pitää olla luokkaa käyttävän **.cpp** tiedoston alussa

- Rakenne jäsenfunktioiden esittämiseen **.cpp** tiedostossa on alla olevan mukainen

<tietotyyppi> <luokannimi> <::> <jäsenfunktion nimi> <parametrit kaarisulkujen sisällä>

```
void MyFirstClass::myFirstFunction (short parameter)
{
... // Tälle funktion toteutus: algoritmit, palautuslause yms.
}
```

- Kun uusi luokka luodaan C++ -ohjelmointikielessä, niin aina kannattaa rakentaa **.h** ja **.cpp** tiedostot luokalle. Se selkeyttää ohjelman rakennetta ja helpottaa myöhemmin ylläpitoa. Monet työkalut jo itsessään ohjaavat tähän rakenteeseen.
- Tiedostot löytyvät resurssinhallintaohjelmalla projektin kansioista. Samasta paikasta missä **main.cpp** tiedosto on.
- **Käy katsomassa resurssinhallintaohjelmalla että tiedostot löytyvät!**

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: luokan jäsenmuuttujat ja –funktiot.

- Klikkaa auki projektin puuranteesta tiedosto **myfirstclass.h**
- Lisää tiedostoon ennen **class** sanaa alla olevat rivit

```
#include <iostream>          // tarvitaan, koska tulostusolio cout on määritelty tässä kirjastossa
using namespace std;        // tarvitaan, koska koko C++:n standardikirjasto on std-nimiavaruuden sisällä
                             // ja kun lause on kirjoitettu .h tiedostossa, niin ohjelmassa ei tarvitse käyttää
                             // käskyjen edessä etuliitettä std
```

- Lisää luokan rakenteeseen **public** osaan alla oleva jäsenfunktio (luokan jäsenfunktiot ovat **oletusarvoisesti julkisia**):

```
void myFirstMemberFunction();
```

- Klikkaa auki projektin puuranteesta tiedosto **myfirstclass.cpp**
- Lisää jäsenfunktion tyhjä runko tiedostoon. HUOM! *Tee aina välitallennus kun kirjoitat hiemankin uutta koodia!*

```
void MyFirstClass::myFirstMemberFunction()
{
}
```

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: luokan jäsenmuuttujat ja –funktiot.

- Klikkaa auki projektin puuranteesta tiedosto **myfirstclass.h**
- Lisää luokan rakenteeseen osa **private**: jäsenfunktion määrittelyn jälkeen
- Lisää **private** osaan alla oleva jäsenmuuttuja (luokan jäsenmuuttujat ovat **oletusarvoisesti yksityisiä**):

```
short myFirstMemberVariable;
```

- Klikkaa auki projektin puuranteesta tiedosto **myfirstclass.cpp**
- Kirjoita funktion **myFirstMemberFunction()** toteutukseen alla oleva koodi

```
myFirstMemberVariable = 10;
```

```
cout << "Jasenmuuttujan myFirstMemberVariable arvo =" << myFirstMemberVariable << endl;
```

- Ylläolevassa lauseessa oleva **endl** aiheuttaa rivinvaihdon näytöllä. Tämä selventää tulostusta.

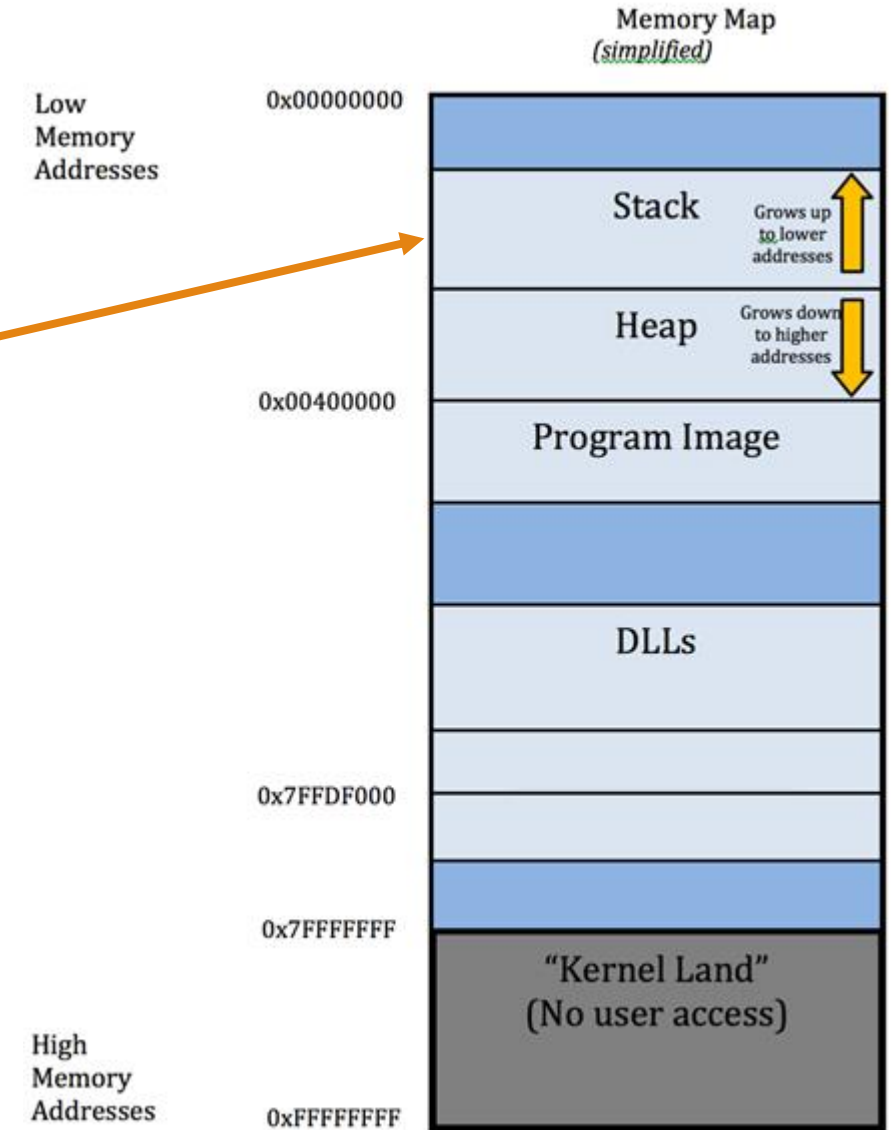
IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin iatkokurssi

Olio-ohjelmointi: automaattisen olion luonti luokasta

- Tehtävässä luodaan automaattinen olio: *käyttöjärjestelmä huolehtii olion muistinvarauksista ja vapauttamisista, olion luonnin ja tuhoamisen yhteydessä, ilman ohjelmoijan erillisiä toimenpiteitä*
- Automaattinen olio luodaan aina pinomuistiin (pinomuisti=Stack)
- Avaa **main.cpp** tiedosto ja kirjoita alla oleva koodi tiedostoon

```
#include "myfirstclass.h"
int main()
{
    MyFirstClass objectMyFirstClass;
    objectMyFirstClass.myFirstMemberFunction();
    return 0;
} // tämän rivin jälkeen olio objectMyFirstClass tuhoutuu pinomuistista
// automaattisesti käyttöjärjestelman toimesta.
```

- Tutustu koodiin huolella, suorita build ja aja ohjelmaa.
Mitä ohjelma tulostaa ohjelmaikkunaan?



IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: luokan jäsenmuuttujat ja –funktiot.

- Lisää luokkaan jäsenfunktio **void mySecondMemberFunction();** ja jäsenmuuttujat **short numberOne,numberTwo;**
- Kirjoita funktion **mySecondMemberFunction()** toteutus tiedostoon **myfirstclass.cpp** ja anna yllä lisätyille jäsenmuuttujille jotkut alkuarvot jäsenfunktiossa **mySecondMemberFunction();**
- Laske ja tulosta jäsenmuuttujien summa, erotus, tulo ja osamäärä jäsenfunktiossa **mySecondMemberFunction()**. Tulosta laskutoimitukset muodossa $5+2=10$
- Käytä **endl** käskyä selventämään tulostusta näytölle.
- Kutsu jäsenfunktia **mySecondMemberFunction()** funktiosta **main()** oliolla **objectMyFirstClass**.
- Suorita build ja aja ohjelmaa. Mitä ohjelma tulostaa ohjelmaikkunaan?

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: luokan jäsenmuuttujat ja –funktiot.

- Lisää luokkaan jäsenfunktio **short myThirdMemberFunction(short parameterOne);**
- Rakenna yllä luotuun jäsenfunktioon seuraava toiminnallisuus:
 - lisää jäsenfunktioon paikallinen muuttuja **short localVariable;**
 - jäsenfunktiolle välitetty parametri kerrotaan luvulla 10 ja tämä arvo asetetaan muuttujaan **localVariable**
 - jäsenfunktio palauttaa muuttujan **localVariable** arvon
- Avaa **main()** –funktio ja lisää sinne paikallinen muuttuja **short returnValue;**
- Kutsu **myThirdMemberFunction** jäsenfunktiota **main()** funktiossa siten, että jäsenfunktion palautusarvo asetetaan muuttujaan **returnValue;**
- Lisää luokkaan jäsenfunktio **void myFourthMemberFunction(short parameterOne);**
- Rakenna jäsenfunktioon toiminnallisuus, jossa tulostetaan näytölle jäsenfunktion parametrin arvo.
- Kutsu **myFourthMemberFunction** funktiota **main()** funktiossa luomallasi paikallisella muuttujalla.
- Suorita build ja aja ohjelmaa. Mitä ohjelma tulostaa ohjelmaikkunaan?

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: luokan jäsenmuuttujat ja –funktiot.

- Lisää luokkaan jäsenfunktio **void myFifthMemberFunction();**
- Rakenna jäsenfunktioon toiminnallisuus, jossa tulostetaan ensimmäiselle riville teksti: "Hei, taalla ollaan!", ja toiselle riville jäsenmuuttujien **numberOne** ja **numberTwo** arvot. *HUOM! Voit siis käyttää luokassa määriteltyjä jäsenmuuttujia jokaisessa luokan jäsenfunktiossa!*
- Jäsenfunktioita voidaan kutsua C++ kielessä aivan kuten C-kielessäkin, toisten funktioiden sisältä.
- Kutsu jäsenfunktioista **myFourthMemberFunction ()** jäsenfunktioita **myFifthMemberFunction()**
- Suorita build ja aja ohjelmaa. Mitä ohjelma tulostaa ohjelmaikkunaan?
- Voit sulkea projektin **MyFirstOOPProject** työkalusta (File->Close All Projects and Editors).

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: luokan muodostin- ja tuhoajafunktio

- Luokalla voi olla erityinen muodostinfunktio, jossa esimerkiksi luokan jäsenmuuttujat voidaan alustaa.
- Muodostinfunktiolle voidaan välittää parametreja aivan kuten muillekin jäsenfunktioille, mutta se ei palauta mitään arvoa, *koska sillä ei ole paluuarvon tyyppiä (=tietotyyppitön funktio).*
- Muodostinfunktio on **aina** saman niminen kuin itse luokka
- Muodostinfunktio suoritetaan (jos se siis on koodattu) aina kun **olio luodaan**
- Luokalla voi olla tuhoajafunktio, joka suoritetaan kun **olio tuhotaan**
- Tuhoajafunktio voi siivota olion jäljet kun olio lakkaa olemasta (esim. muuttujien nollaus, muistin vapautus etc.)
- Tuhoajafunktio on **aina** saman niminen kuin itse luokka, mutta funktion nimen eteen tulee merkki ~
- Sille ei välitetä parametreja eikä sillä ole paluuarvoa, koska sillä ei ole paluuarvon tyyppiä.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: luokan muodostin- ja tuhoajafunktio

- Luo uusi projekti nimeltä **MySecondOOPProject (Non-Qt Project->Plain C++ Application)** ja lisää projektiin luokka **MySecondClass**
- Avaa tiedosto **mysecondclass.h**
- Luokan **public**: osaan tulee automaattisesti muodostinfunktio **MySecondClass()**, jos se ei siellä ole, niin lisää muodostinfunktio sinne.
- Lisää luokkaan tuhoajafunktio **~MySecondClass()** ja jäsenfunktiot **void setValue(short parameterValue)** ja **short getValue()**
- Lisää luokan **private** osaan jäsenmuuttuja **short myVariable**, jonka jälkeen luokan rakenne pitäisi näyttää alla olevalta

```
#include <iostream>
using namespace std;

class MySecondClass
{
public:
    MySecondClass();// muodostinfunktio, joka suoritetaan aina kun olio luodaan
    ~MySecondClass(); // tuhoajafunktio, joka suoritetaan aina kun olio tuhotaan

    void setValue(short parameterValue);
    short getValue();
private:
    short myVariable;
};
```

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: luokan muodostin- ja tuhoajafunktio

- Avaa tiedosto **mysecondclass.cpp** ja kirjoita sinne alla oleva koodi

```
#include "mysecondclass.h"

MySecondClass::MySecondClass()
{
    cout << "Luokasta MySecondClass luotiin olio" << endl;
    myVariable = 10; // asetetaan muuttujan alkuarvo
}

MySecondClass::~~MySecondClass()
{
    myVariable = 0;
    cout << "Luokan MySecondClass olio tuhottiin" << endl;
}

void MySecondClass::setValue(short parameterValue)
{
    myVariable=parameterValue;
}

short MySecondClass::getValue()
{
    return myVariable;
}
```

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: luokan muodostin- ja tuhoajafunktio

- Avaa **main.cpp** tiedosto ja kirjoita alla oleva koodi tiedostoon

```
#include "mysecondclass.h"
```

```
int main()  
{
```

```
    MySecondClass objectMySecondClass; // kun olio luodaan kutsutaan muodostinfunktiota, jos sellainen on luokassa  
    cout << "Olion objectMySecondClass, jäsenuuttuja myVariable arvo=" << objectMySecondClass.getValue() << endl;
```

```
    objectMySecondClass.setValue(5); // asetetaan olion jäsenuuttujalle uusi arvo  
    cout << "Olion objectMySecondClass, jäsenuuttuja myVariable UUSI arvo=" << objectMySecondClass.getValue() << endl;
```

```
    return 0;  
} // kun olio tuhoutuu, niin kutsutaan tuhoajafunktiota, jos sellainen on luokassa
```

- Suorita build ja aja ohjelmaa. Mitä ohjelma tulostaa näytölle?

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: muodostinfunktion parametri

- Avaa luokan määrittely (mysecondclass.h) ja lisää määrittelyn **public** osaan muodostinfunktion määrittelyn jälkeen funktio **MySecondClass(short parameter);**
 - Eli tehdään toinen toteutus luokan muodostinfunktiosta = **muodostinfunktion ylikuormittaminen!**
- Avaa luokan toteutus (mysecondclass.cpp), lisää alla oleva muodostinfunktion toteutus olemassa olevan muodostinfunktion toteutuksen jälkeen

```
MySecondClass::MySecondClass(short parameter)
{
    cout << "Luokasta MySecondClass luotiin olio" << endl;
    cout << "Muodostinfunktion parametrin arvo = " << parameter << endl;
    myVariable = parameter;
}
```

- Muuta **main()** funktiossa oleva rivi **MySecondClass objectMySecondClass;** muotoon **MySecondClass objectMySecondClass(8);**
- Nyt oliota luotaessa kutsutaan sitä muodostinfunktiota, jolla on parametrimuuttuja.
- Suorita build ja aja ohjelmaa.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: useita olioita yhdestä luokasta

- Yhdestä luokasta voidaan luoda **useita** olioita. Olioiden **nimet** täytyy olla ohjelmassa **erit**.
- Jokainen olio on itsenäinen ja sillä on käytössä oma muistialue tehtävien tekemiseen (=jäsenfunktiot) ja muuttujien datan (=jäsenmuuttujat) säilyttämiseen ohjelmassa.
- Olion jäsenmuuttujat ja –funktiot ovat olion omia, eikä niitä jaeta muiden olioiden kesken. Jos halutaan luokasta luoduille olioille yhteisiä jäsenmuuttujia ja/tai –funktioita, tehdään niistä **staattisia**. *Tähän palataan myöhemmin.*
- Avaa **main()** funktio ja kirjoita alla olevat koodirivit ennen riviä **return 0;**

```
MySecondClass objectTest(99); // toinen olio luokasta MySecondClass  
cout << "Olion objectTest, jäsenmuuttuja myVariable arvo=" << objectTest.getValue() << endl;
```

- Suorita build ja aja ohjelmaa
- **Riippuen ohjelmasta voidaan kutsua siis kumpaa muodostinfunktiota tahansa! Kaikilla jäsenfunktioilla voi olla useita erilaisia toteutuksia ohjelman tarpeen mukaan = jäsenfunktion uudelleenmäärittely!**

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: useita olioita luokasta

- Lisää luokkaan **MySecondClass** jäsenfunktio **void showObjectMemoryAddress();**

- Kirjoita yllä olevan jäsenfunktion toteutukseen alla oleva koodi

```
cout << "Muuttujan myVariable muistiosoite pinossa: " << &myVariable << endl;
```

- Kirjoita **main()** funktioon ennen **return 0;** lausetta seuraavat rivit

```
objectMySecondClass.showObjectMemoryAddress();  
objectTest.showObjectMemoryAddress();
```

- Suorita build ja aja ohjelmaa. Mitä yllä oleva jäsenfunktio tulostaa näytölle?
- Jokainen luokasta luotu olio saa oman muistiavaruuden (tässä tapauksessa RAM muistin osasta PINO), minkä vuoksi jäsenmuuttujan muistipaikkoja tulostettaessa, tulostuu näytölle eri arvot pinosta.
- Luokka varaa RAM –muistia pinosta luokan jäsenmuuttujien määritysten (short, int, double..) mukaisesti.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

- Olio-ohjelmoinnin neljä tärkeää perusperiaatetta: kapselointi, tiedon kätkeyminen, periytyminen ja polymorfismi.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: kapselointi (encapsulation)

- Kapselointi on ohjelmointitekniikka, joka määrittää sekä ohjelmien modularisoinnin että tietojen ja toteutustavan piiloutuksen periaatteet. Sen tarkoitus on **uudelleenkäytettävyyden** lisääminen ja **ylläpidon** paikallistaminen luokkaan.
- **Kapseloinnin** avulla voidaan sovelluksessa koota yhdeksi kokonaisuudeksi (luokaksi/ohjelmistokomponentiksi) yhteen kuuluvat tiedot ja toiminnot.

*Encapsulation is all about binding the data variables and functions together in class.
It can also be said data binding.*

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: tiedon kätkeyttä (information hiding)

- Tiedon kätkeyttäällä tarkoitetaan sitä, että kapseloitua osaa (esim. luokkaa tai ohjelmistokomponenttia) voidaan käyttää tuntematta luokan/ohjelmistokomponentin sisäistä rakennetta.
- Luokan jäsenmuuttujien osalta tarkoitetaan sitä, että luokan jäsenmuuttujat tehdään sellaiseksi, että niitä voidaan käsitellä vain luokan jäsenfunktioissa.
- C++ -ohjelmoinnissa tiedon kätkeyttä toteutetaan luokassa jakamalla luokan näkyvyystasot osiin **public**, **protected** ja **private**.
- Alla tiedon kätkeyttään liittyviä periaatteita:
 - luokan jäsenmuuttujat – joita olio käyttää - tulisi olla kätkeyttynä eri tasoilla (public, protected ja private).
 - jäsenmuuttujat ovat lähtökohtaisesti **private** tasolla, näin jäsenmuuttujiin ei pääse käsiksi olion ulkopuolelta. Tämä tarkoittaa sitä, että vain luokan jäsenfunktioiden avulla voidaan muuttaa jäsenmuuttujien tietoja.
 - Jäsenfunktioiden avulla voidaan jäsenmuuttujien sisältöä muuttaa kontrolloidusti ja turvallisesti.
 - Jäsenfunktiot ovat lähtökohtaisesti julkisia (eli ovat **public** tasolla), mutta esim. perintä yhteyden ollessa kyseessä luokan jäsenfunktiot voivat olla **protected** tasolla.
- **protected** osa liittyy periytymiseen ja sitä käsitellään periytymiseen liittyvässä osassa.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: periytyminen

- Periytyminen on kahden luokan välinen binäärinen relaatio.
- Periytyminen mahdollistaa yliluokassa määriteltujen ominaisuuksien käyttämisen aliluokassa.
- C++ tukee kahdenlaista periytymistä
 - Yksinkertaista periytymistä, jossa periytetään vain yksi luokka.
 - Monikertaista periytymistä, missä luokalla on useampi kuin yksi suora kantaluokka samaan aikaan.
- Periytymisen myötä ohjelmaan muodostuu luokkahierarkia.
- Periytymisessä aliluokka perii yliluokan jäsenmuuttujat, -funktiot ja yhteydet muihin luokkiin. Aliluokassa voidaan yliluokasta perittyjä ominaisuuksia kumota eli määritellä uudestaan.
- Periytymistä käsitellään erikseen tarkemmin tulevassa opiskelumateriaalin osassa periytyminen.
- Periytymisen avulla koodin kirjoittaminen nopeutuu, koska voidaan käyttää olemassa olevia valmiita, toimivia ja testattua luokkia, eikä kaikkia tarvitse tehdä alusta alkaen itse.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: polymorfismi eli monimuotoisuus

- Monimuotoinen luokka mahdollistaa saman jäsenfunktion toteutuksen monella tavalla.
- Olio-ohjelmoinnissa monimuotoisuus ilmenee jäsenfunktioiden osalta kahdella eri tavalla:
jäsenfunktioiden ylikuormittamisena (overloading) ja metodien uudelleen määrittelemisenä (overriding)
- Käsite **overriding** liittyy C++ -ohjelmointikielessä periytymiseen. Se tarkoittaa että ylä- ja aliluokilla voi olla samannimisiä jäsenfunktioita. Tähän asiaan tutustutaan myöhemmin opiskelumateriaalin luvussa 6.
- Monimuotoisuus toteutetaan C++ -ohjelmointikielessä **virtuaalifunktioiden** ja käsitteen **dynaaminen sidonta** avulla. Tätä asiaa käydään läpi myöhemmin opiskelumateriaalin osassa 6, osan kotitehtävissä ja osan lisätehtävissä.

The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance. C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

- Seuraavalla sivulla esitellään käsite **overloading**.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Polymorfismi: Jäsenfunktion ylikuormittaminen (Overloading)

- Ylikuormittaminen on yksi polymorfismin ilmentymä olio-ohjelmoinnissa
- Ylikuormittamisessa funktion nimi on monimuotoinen, toisin sanoen samannimisellä funktiolla on **useita eri toteutuksia**, joista kutsuttaessa valitaan tilanteeseen sopiva. Tyypillisesti tämä tapahtuu siten, että eri toteutuksien parametrilistat ovat erilaiset ja kutsuttaessa metodia päätellään syöteparametrien määrästä ja tyypeistä, mikä toteutus valitaan. Valinta voidaan tehdä staattisesti eli käännösaikana, koska kaikki tarvittava informaatio kutsun sitomiseksi on jo silloin saatavana.
- Luokassa voisi olla ylikuormitettuna jäsenfunktio **void myOverloadedFunction()** vaikka seuraavasti:

```
void myOverloadedFunction();  
void myOverloadedFunction(short param);  
short myOverloadedFunction(short numberOne, numberTwo);
```

- Ylikuormittamista voidaan käyttää esimerkiksi silloin, kun luokan toiminnallisuuteen kuuluu piirtää erilaisia asioita erilaisten laitteiden näytöille. Luokka voi tällöin olla jäsenfunktio **void draw()**, joka ylikuormitetaan eri tavoin, ja jokaiselle jäsenfunktiolle kirjoitetaan oma toteutus.
- Ylikuormittamiseen liittyen on kotitehtävä.

Operator overloading is an important concept in C++. It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it. Overloaded operator is used to perform operation on user-defined data type.