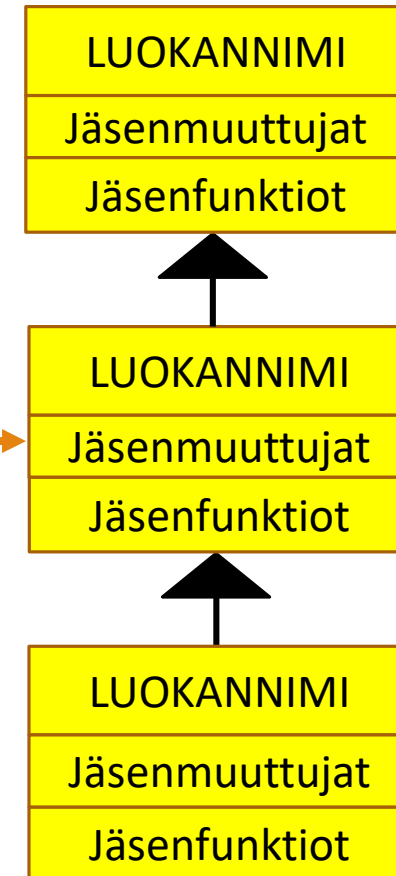


## Olio-ohjelmointi: Periytyminen, luokkien välinen yhteys

- Olio-ohjelmoinnin yksi perusperiaate on se, että ohjelmisto rakennetaan toimimaan olioiden yhteistyöhön perustuen.
- Periytyminen on kahden luokan välinen binäärinen relaatio. Se mahdollistaa yliluokassa (=kanta-/isäluokka) määriteltujen ominaisuuksien käyttämisen aliluokassa (=lapsiluokka). Periytymisessä aliluokka perii ominaisuudet myös kaikilta yliluokkansa esivanhemmilta (=transitiivisyys).
- C++ tukee kahdenlaista periytymistä
  - Yksinkertaista periytymistä, jossa periytetään vain yksi luokka.
  - Monikertaista periytymistä, missä luokalla on useampi kuin yksi suora yliluokka samaan aikaan.
  - Periytymisen myötä ohjelmaan muodostuu luokkahierarkia.
- Periytymisessä uusi luokka perii jäsenmuuttujat ja -funktiot toisesta luokasta. Myös perityn luokan periytymishierarkia peritään.
- Aliluokassa voidaan yliluokasta perittyjä ominaisuuksia kumota eli **määritellä uudestaan**.
- Perintä voidaan muodostaa julkiseksi (public), suojatuksi (protected) ja yksityiseksi (private). Muodostaminen tapahtuu luokan määrittelyssä, **class** sanan yhteydessä
- Perinnän syntaksi C++ -ohjelmointikielessä on: **class Aliluokka : public Yliluokka**




### Olio-ohjelmointi: Periytyminen

- Julkinen periytyminen (public)
  - *Aliluokka voi käyttää omassa toteutuksessa ylliluokan julkisia jäseniä (public) ja suojattuja jäseniä (protected).* Julkinen periytyminen määritellään lisäämällä perittävän luokan yhteyteen **public**-avainsana. Määritys tulee tehdä jokaiselle luokalle erikseen.  
**Julkinen perintä on yleisin.**
- Suojattu periytyminen (protected)
  - Suojatussa periytymisessä aliluokan oliota ei voida käyttää kuin se olisi ylliluokan olio. Suojatussa periytymisessä ylliluokan julkiset jäsenet ovat aliluokan suojattuja jäseniä. Aliluokka voi käyttää omassa toteutuksessa ylliluokan julkisia jäseniä sekä suojattuja jäseniä. Suojattu periytyminen määritellään lisäämällä perittävän luokan yhteyteen **protected**-avainsana. Tämä pitää tehdä jokaiselle luokalle erikseen.
- Yksityinen periytyminen (private)
  - Yksityisessä periytymisessä aliluokan oliota ei voidaan käyttää kuin se olisi ylliluokan olio. Yksityisessä periytymisessä ylliluokan julkiset ja suojatut jäsenet ovat aliluokan yksityisiä jäseniä. Aliluokka voi käyttää omassa toteutuksessa ylliluokan julkisia sekä suojattuja jäseniä. Periytymissuhde määritellä yksityiseksi lisäämällä perittävän luokan yhteyteen **private**-avainsana. Tämä pitää tehdä jokaiselle luokalle erikseen.

### Olio-ohjelmointi: Periytyminen, hyviä puolia

- Periytymisen ansiosta ohjelmoinnin **työmäärä vähenee**, kun aiemmin tehtyjä luokkia voidaan käyttää hyväksi vain tarpeelliset luokat lisäämällä.
- **Virheiden määrä pienenee**, koska käytetään luokkia joiden toiminnallisuus on jo aikaisemmin testattu.
- **Ylläpidettävyys helpottuu**, koska kaikkea ei tarvitse tehdä itse, vaan voi käyttää hyväkseen jonkun toisen luomia luokkakirjaston luokkia (esim. Qt luokkakirjaston luokat).
- **Testattavuus paranee**, koska voidaan keskittyä testaamaan vain uusia ominaisuuksia.
- **Koodin uudelleenkäyttö**. Periytymisen kenties yksinkertaisin käyttötarkoitus on olemassa olevan luokan kaikkien/tai osan toimintojen käyttöönottaminen ohjelmassa. Aliluokka ottaa käyttöön kaikki/tai osan ylliluokan tarjoamista toiminnoista ja sen lisäksi tuo toteutukseen mukaan omia toiminnallisuuksia. Tällainen periytymisen käyttö mahdollistaa **koodin uudelleenkäytön**, koska aliluokan ei tarvitse kirjoittaa uudelleen ylliluokan jo kertaalleen toteuttamia toimintoja.

## Olio-ohjelmointi: periytyminen

- Luokkakaaviossa luokkaa kuvatessa noudetaan oikealla olevaa rakennetta 
- Luokan kuvauksessa jäsenfunktioiden ja –muuttujien edessä oleva merkki +, # tai – tarkoittaa seuraavaa:

+ tarkoittaa **public** osaa  
# tarkoittaa **protected** osaa  
- tarkoittaa **private** osaa

Device
#short deviceId -short index
+void setDeviceID() +short getDeviceID(short deviceParameter)



```
class Device
{
protected:
    short deviceId;

private:
    short index;

public:
    void setDeviceID();
    short getDeviceID(short deviceParameter);
};
```

Luokan nimi
Jäsenmuuttujat
Jäsenfunktiot

# IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

## Olio-ohjelmointi: Periytyminen, projektin luonti

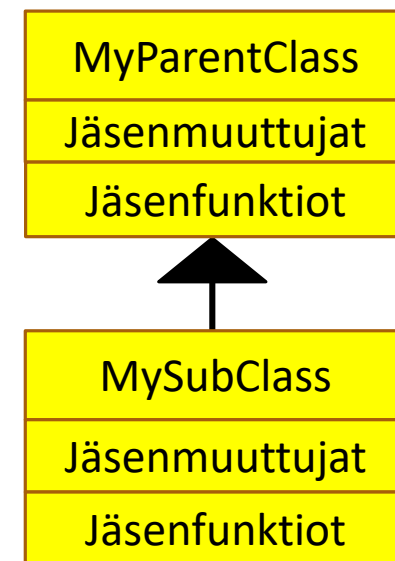
- Tee uusi projekti (tyyppi **Non-Qt Project>Plain C++ Application**) nimeltä **MyFirstInheritanceProject**
- Lisää projektiin luokka **MyParentClass** (tästä tulee yli-/kantaluokka). Lisää luokkaan tuhoajafunktio, jos se ei siellä automaattisesti ole.
- Lisää projektiin luokka **MySubClass** (aliluokka, joka perii luokan **MyParentClass** ). Lisää luokkaan tuhoajafunktio, jos se ei siellä automaattisesti ole.
- Avaa luokan **MySubClass** määrittely.
- Lisää alla olevat rivit luokan **MySubClass** määrittelyyn **#define** rivin jälkeen

```
// Aliluokassa täytyy aina esitellä yliluokan .h tiedosto, jotta käännös menee läpi.  
#include "myparentclass.h"
```

- Luokan määrittelyssä periytyminen kerrotaan kirjoittamalla luokan nimen jälkeen kaksoispiste, periytymistapa sekä yliluokan nimi. Muuta luokan määrittelyn **class** rivi alla olevan mukaiseksi

```
// Luokka MySubClass perii luokan MyParentClass  
class MySubClass : public MyParentClass
```

- Suorita **Build** toiminto (älä aja ohjelmaa) ja jatka seuraavalle sivulle, kun ohjelma kääntyy ilman virheitä.



### Olio-ohjelmointi: Periytyminen, muodostin- ja tuhoajafunktioiden suorittaminen periytymisessä

- Seuraavat rivit lisätään ohjelmaan, jotta yli- ja aliluokan muodostin- ja tuhoajafunktioiden **suoritusjärjestys** tulee esille.
- Lisää luokkien muodostin- ja tuhoajafunktioihin tulostuslauseet **cout** komennolla, esimerkiksi alla olevan mukaisesti:

```
cout << "MyParentClass muodostinfunktio" << endl;
```

- Koska kyseessä on perintäyhteys luokkien välillä niin riittää, että **MyParentClass** luokkaan (yliluokkaa) lisätään alla olevat rivit ennen luokan määrittelyä (siis **class** sanaa)

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

- Suorita **Build** toiminto (älä aja ohjelmaa) ja jatka seuraavalle sivulle, kun ohjelma kääntyy ilman virheitä.

## IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

### Olio-ohjelmointi: Periytyminen, muodostin- ja tuhoajafunktioiden suorittaminen periytymisessä

- Poista **main.cpp** tiedostossa oleva koodi ja kirjoita sinne alla oleva koodi

```
#include "mysubclass.h"
```

```
int main()
```

```
{  
    // Alla olevalla rivillä esitellään osoitin objectMySubClass joka on tyyppiä MySubClass ja varataan muistia keosta.  
    MySubClass *objectMySubClass = new MySubClass;  
  
    delete objectMySubClass;  
    objectMySubClass = nullptr;  
  
    return 0;  
}
```

- Käännä ja aja ohjelmaa. *Huomaa missä järjestyksessä muodostin- ja tuhoajafunktiot suoritetaan! Vertaa koosteeseen automattisilla oliolla tehtynä ja dynaamisella oliolla tehtynä!*
- Huomaa, että yliluokan oliota ei edes luoda, vaikka sen muodostin- ja tuhoajafunktiot suoritetaan! Periytyminen!***

## IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

### Olio-ohjelmointi: Periytyminen, julkinen (public) periytyminen

- *Aliluokka voi käyttää omassa toteutuksessa yliluokan julkisia jäseniä (public) ja suojattuja jäseniä (protected).*
- Lisää yliluokan määrittelyn **public** osaan jäsenfunktio **void myParentMemberFunction();**
- Lisää yliluokan määrittelyn **protected** osa ja sinne jäsenmuuttuja **short myParentMemberVariable;**
- Aseta muuttujan **myParentMemberVariable** alkuarvoksi 10 yliluokan muodostinfunktiossa.
- Lisää jäsenfunktion **void myParentMemberFunction()** toteuksen runko ja lisää toteutukseen alla olevat rivit

```
cout << "MyParentClass luokan jäsenfunktio myParentMemberFunction()" << endl;  
cout << "myParentMemberVariable jäsenmuuttujan arvo= " << myParentMemberVariable << endl;
```

- Lisää **main()** funktiossa olion luonnin jälkeen alla oleva rivi

```
objectMySubClass->myParentMemberFunction(); // aliluokan olio kutsuu yliluokan jäsenfunktiota. Periytyminen mahdollistaa!
```

- Aliluokan olio voi kutsua yliluokan osassa **public** esiteltyä jäsenfunktiota, koska luokkien **MySubClass** ja **MyParentClass** välille on muodostettu **julkinen periytyminen**.
- Suorita build ja aja ohjelmaa.



## IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

### Olio-ohjelmointi: Periytyminen, julkinen periytyminen

- Aseta muuttujan **myParentMemberVariable** arvoksi 200 **ALILUOKAN** muodostinfunktiossa.
  - Aliluokan jäsenfunktioissa voidaan käyttää yliluokan **protected** osassa esiteltyjä muuttujia/oliota. *Niitä käsitellään aliluokassa kuten ne olisivat aliluokan **private** osan muuttujia/olioita.*
- Suorita build ja aja ohjelmaa.
- Lisää aliluokan määrittelyn **private** osaan jäsenmuuttuja **short mySubMemberVariable;**
- Kirjoita aliluokan muodostinfunktion koodi alla olevan mukaiseksi

```
// Muodostinfunktiossa voidaan muuttujien alkuarvoja asettaa myös alla olevalla tavalla.  
// Muuttujia/oliota voi olla useita, ja ne kirjoitetaan peräkkäin pilkulla erotettuna.  
// Myös oliota voidaan luoda vastaavalla tavalla. Tasta esimerkkejä myöhemmin.  
MySubClass::MySubClass() : mySubMemberVariable(100)  
{  
    cout << "Luokan MySubClass muodostinfunktio" << endl;  
    cout << "mySubMemberVariable jäsenmuuttujan arvo=" << mySubMemberVariable << endl;  
    // myParentMemberVariable = 200;  
}
```

- Suorita build ja aja ohjelmaa.

### Olio-ohjelmointi: Periytyminen, julkinen periytyminen

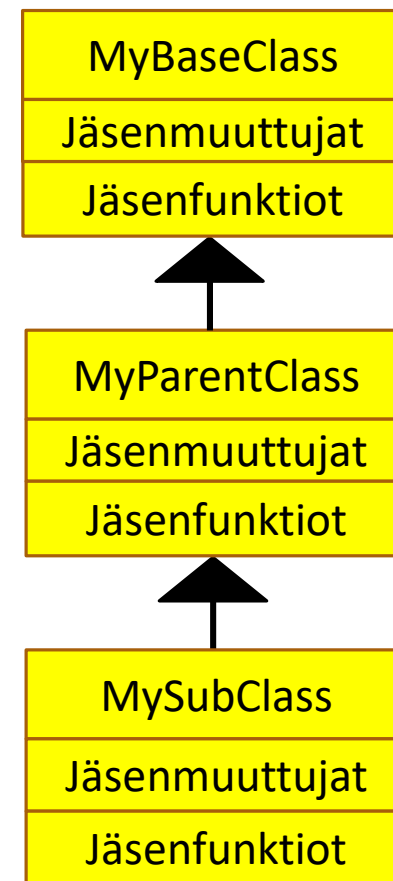
- Siirrä yliluokan jäsenmuuttuja **short myParentMemberVariable**, **private** osaan.
- Poista kommentit aliluokan muodostinfunktiossa, jossa yliluokan jäsenmuuttujalle annetaan alkuarvo.
- Suorita build (älä aja ohjelmaa). Miksi tulee käännösvirhe?
- Siirrä yliluokassa jäsenmuuttuja takaisin **protected** osaan, jotta ohjelma kääntyy.
- Suorita build ja aja ohjelmaa.

### Olio-ohjelmointi: Periytyminen, julkinen periytyminen

- Lisää ohjelmaan luokka **MyBaseClass**. Oikealla kuvassa ohjelman luokkahierarkiasta UML mallinnuskielen luokkakaavio.
- Lisää luokkaan **MyBaseClass** muodostin- ja tuhoajafunktiot ja niiden toteutukset siten, että toteutuksissa tulostetaan missä jäsenfunktiossa ollaan.
- Kommentoi **main()** funktiossa oleva rivi **objectMySubClass->myParentMemberFunction();**
- Laita luokka **MyParentClass** perimään luokka **MyBaseClass** julkisen perinnän mukaisesti
- Koska kyseessä on perintäyhteys luokkien välillä niin riittää, että **MyBaseClass** luokkaan lisätään alla olevat rivit ennen luokan määrittelyä (siis **class** sanaa). Voit siis poistaa/kommentoida vastaavat rivit luokan **MyParentClass** määrittelystä.

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

- Suorita build ja aja ohjelmaa. *Huomaa missä järjestyksessä muodostin- ja tuhoajafunktiot suoritetaan!*



### Olio-ohjelmointi: Periytyminen, julkinen periytyminen

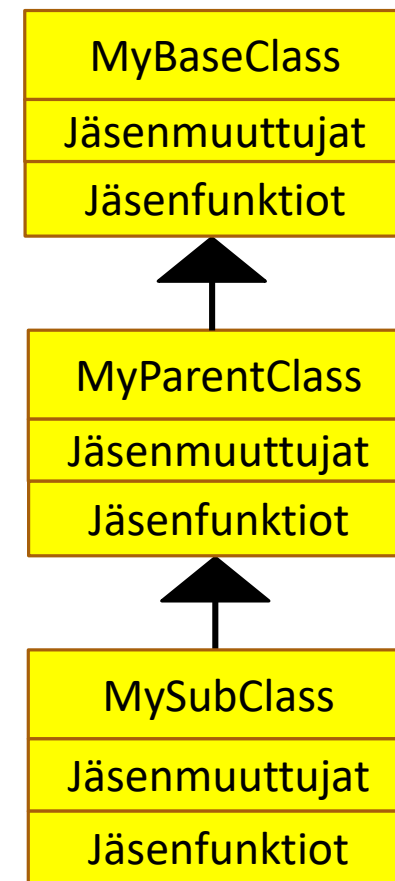
- Lisää luokkaan **MyBaseClass** jäsenfunktio **void myBaseFunctionOne();** jossa tulostetaan

```
cout << "MyBaseClass luokan funktio myBaseFunctionOne()" << endl;
```

- Lisää **main()** funktiossa olion luonnin jälkeen rivi

```
objectMySubClass->myBaseFunctionOne();
```

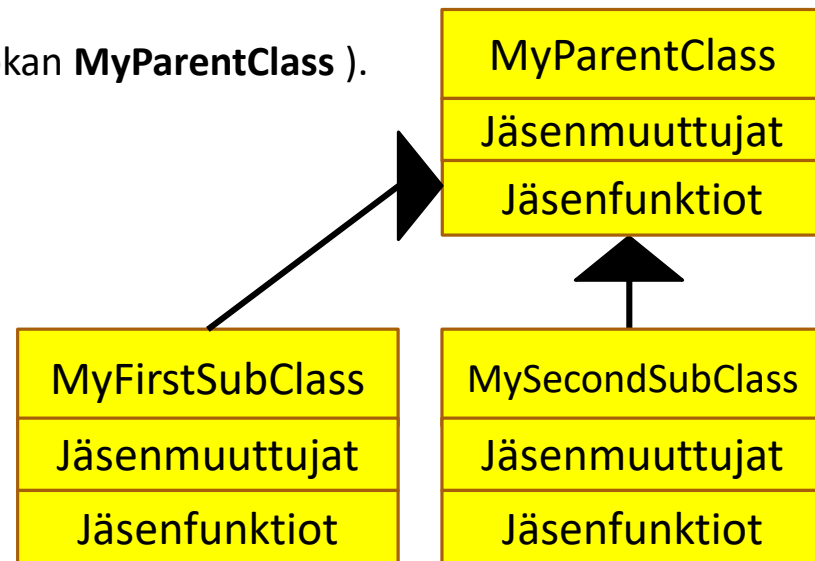
- Suorita build ja aja ohjelmaa.
- Kuvassa olevan luokkahierarkian mukaisesti **MySubClass** voi kutsua myös kantaluokassa (kantaluokka on luokkahierarkian ylin luokka) **MyBaseClass** olevia funktiota.



# IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

## Olio-ohjelmointi: Periytyminen, projektin luonti

- Seuraavassa esimerkissä huomaat, että kun kaksi luokkaa perii saman yliluokan, niin ne saavat yliluokan jäsenmuuttujat- ja funktiot käyttöönsä toisista riippumatta. Lisäksi esimerkin tarkoituksena on selventää mikä ero yliluokan eri näkyvyysosilla on aliluokille.
- Tee uusi projekti (tyyppi **Non-Qt Project>Plain C++ Application**) nimeltä **MySecondInheritanceProject**
- Lisää projektiin luokka **MyParentClass** (tästä tulee yli-/kantaluokka). Poista luokasta muodostinfunktio, sitä ei tässä esimerkissä tarvita.
- Lisää projektiin luokat **MyFirstSubClass** ja **MySecondSubClass** (aliluokat, joka perivät luokan **MyParentClass** ).
- Laita aliluokat koodissa perimään yliluokka julkisen perinnän mukaisesti.
- Suorita **Build** toiminto (älä aja ohjelmaa) ja jatka seuraavalle sivulle, kun ohjelma kääntyy ilman virheitä.



### Olio-ohjelmointi: Periytyminen, julkinen periytyminen

- Lisää yliluokan **private** osaan jäsenmuuttuja **short myParentPrivateMemberVariable;**
- Lisää yliluokkaan jäsenfunktio **void setValue(short paramValue)**. Lisää jäsenfunktioon alla oleva koodi, jossa asetetaan jäsenmuuttujaan funktion parametrin arvo

**myParentPrivateMemberVariable = paramValue;**

- Lisää yliluokkaan jäsenfunktio **short getValue();** Lisää jäsenfunktioon alla oleva koodi, jossa palautetaan jäsenmuuttujan arvo.

**return myParentPrivateMemberVariable;**

- Eli yliluokan jäsenfunktiot käsittelevät **private** osan jäsenmuuttujaa. Myöhemmin tehtävässä aliluokkien oliot kutsuvat yliluokan jäsenfunktioita, ja voivat niiden avulla käsitellä yliluokan **private** osassa olevaa jäsenmuuttujaa.  
***Periytyminen luokkien välillä mahdollistaa tämän!***

### Olio-ohjelmointi: Periytyminen, julkinen periytyminen

- Luo **main()** funktiossa oliot aliluokista käyttäen pinomuistia.
- Kutsu aliluokkien muodostinfunktioissa yliluokan jäsenfunktiota **setValue**. Aseta kutsuissa eri arvot parametrille.
- Kirjoita main() funktioon koodia ennen **return** lausetta, jossa aliluokan oliolla kutsutaan jäsenfunktiota **getValue()**, ja tulostetaan **cout** käskyllä jäsenfunktion palauttama arvo näytölle. Käytä molempia oliota.
- *Tässä esimerkissä huomaat, että kun kaksi luokkaa perii saman yliluokan, niin ne saavat yliluokan jäsenmuuttujat- ja funktiot käyttöönsä toisista riippumatta. Lisäksi aliluokan oliot pääsevät käsiksi yliluokan private osan jäsenmuuttujiin periytymisen ansiosta.*
- Suorita BUILD ja aja ohjelmaa.

## IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

- Lisää yliluokan **protected** osaan jäsenmuuttuja **short myParentProtectedMemberVariable**;
- Nyt kun jäsenmuuttuja on **protected** osassa, niin voit suoraan käsitellä sitä aliluokissa, aivan kuin tämä yliluokan jäsenmuuttujan olisi aliluokan **private** osan muuttuja.
- Aseta molempien aliluokkien muodostinfunktioissa arvot jäsenmuuttujaan **myParentProtectedMemberVariable** ja tulosta arvo **cout** käskyllä muodostinfunktiossa.
- Suorita BUILD ja aja ohjelmaa.
- *Tämän esimerkin tarkoituksena on selventää mikä ero yliluokan eri näkyvyysosilla on aliluokille!*