Olio-ohjelmointi: Periytyminen ja virtuaalifunktiot

- Joskus aliluokan olion on tarpeen suorittaa yliluokasta perimänsä toiminnallisuus hieman yliluokasta poikkeavalla tavalla.
- Olio-ohjelmointi tarjoaa aliluokalle mahdollisuuden tarjota oman toteutuksensa yliluokalta perimälleen jäsenfunktiolle, jos kyseinen jäsenfunktio on yliluokassa määritelty virtuaaliseksi. Tällaista jäsenfunktiota kutsutaan **virtuaalifunktioksi**.
- Luokasta tulee monimuotoinen (=polymorfismi), kun luokassa on yksikin virtuaalinen jäsenfunktio.
- Ohjelman luokkia määriteltäessä tulisi aina miettiä, voidaanko jotain ohjelman luokkaa käyttää jossain monimuotoisena yliluokkana
- Lähtökohtaisesti kannattaa tuhoajafunktio ja muut mahdollisesti aliluokissa korvattavat jäsenfunktiot määritellä virtuaalisiksi.

> ***Polymorphism*** *means having multiple forms of one thing. In inheritance, polymorphism is done, by method* ***overriding****, when both super and sub class have member function with same declaration bu different definition. If we inherit a class into the derived class and provide a definition for one of the base class's function again inside the derived class, then that function is said to be overridden, and this mechanism is called* ***Function Overriding****.*

Olio-ohjelmointi: Periytyminen ja virtuaalifunktiot

- Joskus aliluokan olion on tarpeen suorittaa yliluokasta perimänsä toiminnallisuus hieman yliluokasta poikkeavalla tavalla. Aliluokan jäsenfunktion sanotaan **uudelleenmäärittelevän** (**override**) sen yliluokan jäsenfunktion, mikäli aliluokassa on toteutettu jäsenfunktio, jolla on täsmälleen sama parametrilista ja paluuarvo kuin yliluokan jäsenfunktiolla.

- Olio-ohjelmointi tarjoaa aliluokalle mahdollisuuden tarjota oman toteutuksensa yliluokalta perimälleen jäsenfunktiolle, jos kyseinen jäsenfunktio on yliluokassa määritelty virtuaaliseksi. Tällaista jäsenfunktiota kutsutaan **virtuaalifunktioksi**.

- Jäsenfunktio määritellään yliluokan esittelyssä virtuaaliseksi lisäämällä jäsenfunktion eteen avainsana **virtual**.
  Tämän jälkeen yliluokasta periytettävillä aliluokilla on kaksi mahdollisuutta:

  - Hyväksyä yliluokan tarjoama jäsenfunktion toteutus. Tällöin aliluokan ei tarvitse tehdä mitään eli yliluokan toteutus periytyy automaattisesti myös aliluokkaan.
  - Kirjoittaa oma toteutuksensa perimälleen jäsenfunktiolle. Tässä tapauksessa aliluokan esittelyssä esitellään jäsenfunktio uudelleen, ja sen jälkeen aliluokan toteutuksessa kirjoitetaan jäsenfunktiolle uusi toteutus aivan kuin normaalille aliluokan jäsenfunktiolle. Aliluokan esittelyssä avainsanan **virtual** toistaminen ei ole pakollista, mutta kylläkin hyvän ohjelmointityylin mukaista.

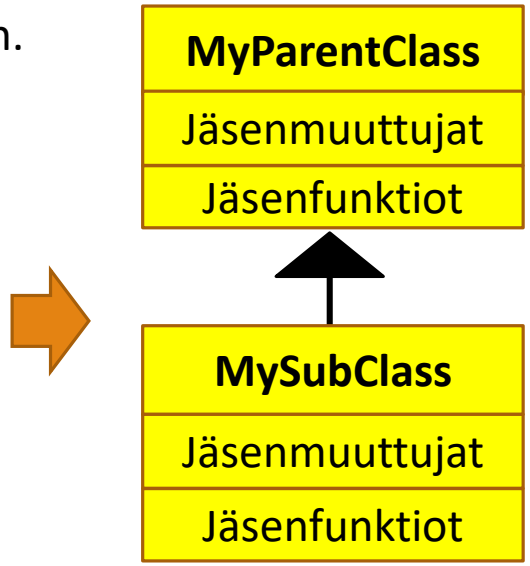
- Luokasta tulee monimuotoinen (=polymorfismi), kun luokassa on yksikin virtuaalinen jäsenfunktio.

- Ohjelman luokkia määriteltäessä tulisi aina miettiä, voidaanko jotain ohjelman luokkaa käyttää jossain monimuotoisena yliluokkana.

- Lähtökohtaisesti kannattaa tuhoajafunktio ja muut mahdollisesti aliluokissa korvattavat funktiot määritellä virtuaalisiksi.

Olio-ohjelmointi: Periytyminen ja virtuaalifunktiot

- Tee uusi projekti nimeltä **MyVirtualfunctionProject** (**Non-Qt Project>Plain C++ Application**)
- Lisää projektiin luokat **MyParentClass** ja **MySubClass**. Lisää luokkiin tuhoajafunktiot, jos niitä ei siellä automaattisesti ole.
- Lisää muodostin- ja tuhoajafunktioihin **cout …** lauseet kertomaan, missä jäsenfunktiossa ollaan.
- Suorita **Build** toiminto (älä aja ohjelmaa). Jos kääntäjä antaa varoituksen, niin ei tässä vaiheessa välitetä siitä.
- Kirjoita tarvittavat koodirivit, jotta ohjelmassa **MySubClass** luokka perii luokan **MyParentClass** julkisen (public) perinnän mukaan.
- Suorita **Build** toiminto (älä aja ohjelmaa). Jos kääntäjä antaa varoituksen, niin ei tässä vaiheessa välitetä siitä.

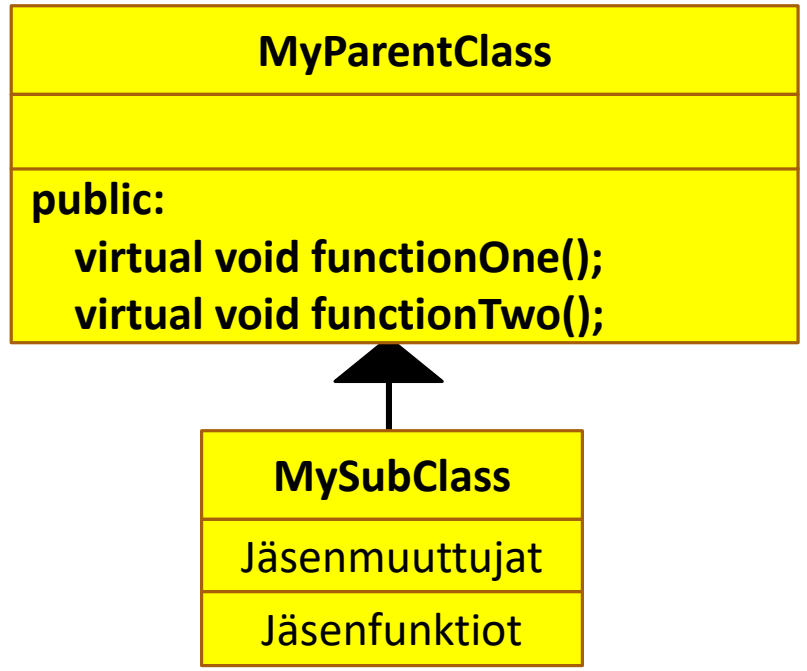


Olio-ohjelmointi: Periytyminen ja virtuaalifunktiot

- Lisää yliluokan määrittelyn **public** osaan jäsenfunktiot **virtual void functionOne()** ja **virtual void functionTwo()**

- Kirjoita jäsenfunktioiden toteutukset alla olevan mukaisesti

```
// virtual sana ei tule funktion toteutukseen mukaan.
void MyParentClass::functionOne()
{
   cout << "MyParentClass virtual functionOne()" << endl;
}

// virtual sana ei tule funktion toteutukseen mukaan.
void MyParentClass::functionTwo()
{
   cout << "MyParentClass virtual functionTwo()" << endl;
}
```



- Suorita **Build** toiminto (älä aja ohjelmaa). Jos kääntäjä antaa varoituksen, niin ei tässä vaiheessa välitetä siitä.

Olio-ohjelmointi: Periytyminen ja virtuaalifunktiot

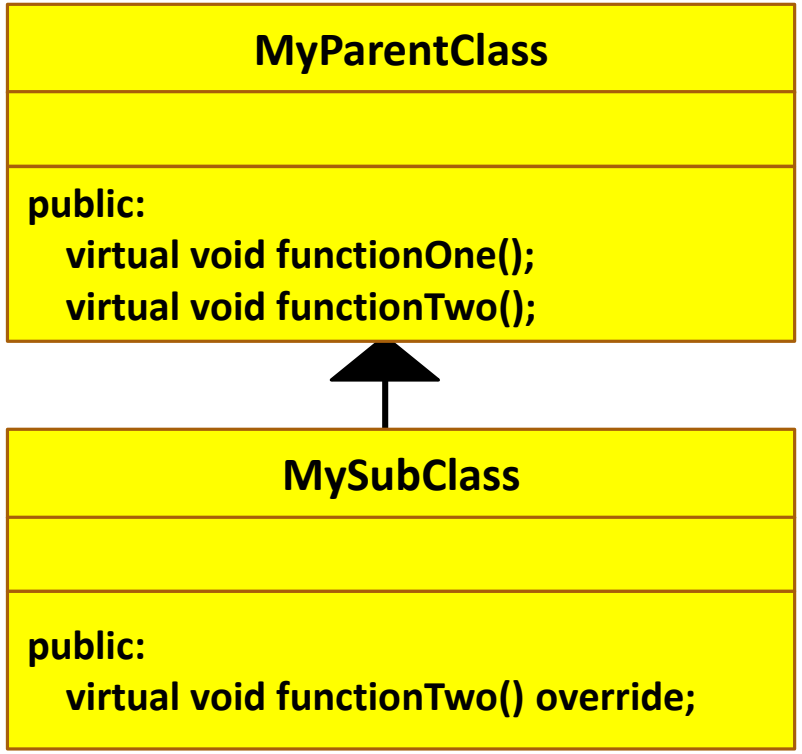
- *Aliluokassa voidaan yliluokan jäsenfunktio korvata* **(uudelleenmäärittely, override)***, jos* ***aliluokkaan*** *halutaan* ***erilainen toiminnallisuus*** *kuin yliluokalla.*

- Lisää aliluokan määrittelyn **public** osaan jäsenfunktio **virtual void functionTwo() override;**

- Lisää jäsenfunktion toteutus alla olevan mukaisesti

```
void MySubClass:: functionTwo()
{
    cout << "MySubClass virtual functionTwo()" << endl;
}
```

- **override** (=syrjäyttää): kerrotaan kääntäjälle, että uudelleenmääritellään aliluokassa yliluokan jäsenfunktio



Olio-ohjelmointi: Periytyminen ja virtuaalifunktiot

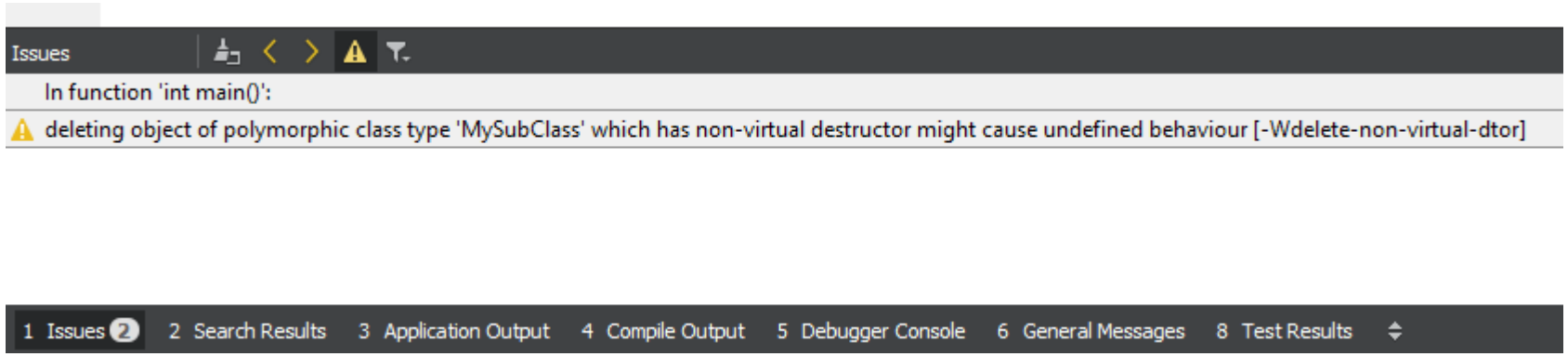
- Kommentoi **main()** funktion koodi ja kirjoita **main()** funktioon alla oleva koodi

```
#include "mysubclass.h"

int main()
{
   MySubClass *objectMySubClass = new MySubClass;

   objectMySubClass->functionOne();
   objectMySubClass->functionTwo();

   delete objectMySubClass;
   objectMySubClass = nullptr;

   return 0;
}
```

- Suorita build ja aja ohjelmaa. Mitä tulostuu näytölle?

Olio-ohjelmointi: Periytyminen ja virtuaalifunktiot

- Jos **Build** operaatio antaa alla olevan varoituksen (saat sen myös uudelleen esille valitsemalla päävalikosta **Build->Clean Project …** ja tämän jälkeen suoritat **Build->Rebuild Project …** toiminnon.)



- …. niin se tarkoittaa, että yliluokan tuhoajafunktio on määriteltävä virtuaaliseksi

- Kun olet määritellyt yliluokan tuhoajafunktion virtuaaliseksi, niin valitse päävalikosta **Build->Clean Project …** ja tämän jälkeen suorita **Build->Rebuild Project …** toiminnon uudelleen (voi olla että **Build** myös riittää)

- Varoitusta ei pitäisi enää tulla.

- Asiaa käsitellään lisää tämän luvun kotitehtävissä. Lisäksi tämän osan kotitehtävässä käydään läpi käsitteet **aikainen sidonta** (Early Binding) ja **myöhäinen sidonta** (Late Binding), jotka liittyvät koodin käännösprosessiin ja jäsenfunktioiden suorittamiseen ohjelman ajon aikana. Nämä käsitteet eivät tässä vaiheessa ole kovin olennaisia, mutta on kuitenkin syytä ymmärtää niiden tarkoitus.

Olio-ohjelmointi: Periytyminen ja virtuaalifunktiot

- Jos ohjelmassa jostain syystä kuitenkin halutaan ***aliluokan olion avulla*** kutsua ***yliluokan jäsenfunktiota*** tehdään se seuraavasti

  - Lisää **main()** funktioon ennen riviä **delete objectMySubClass;** rivi **objectMySubClass->MyParentClass::*functionTwo*();**

- Suorita build ja aja ohjelmaa.

- Virtuaalifunktio –käsitettä käydään lisää läpi myös luvussa 8. Luvussa tulee lisää esimerkkejä ja soveltava harjoitus asiaan liittyen.