## Olio-ohjelmointi: Aliluokan esittely ja määrittely

- Seuraavassa käydään läpi käsitteet **esittelyluokka** ja **määrittelyluokka**. Näitä käsitteitä tarvitaan kun opiskellaan C++ -ohjelmointikielessä olevaa käsitettä **monimuotoisuus** ja **dynaaminen sidonta**.

- Esittelyluokka tarkoittaa olion tunnuksen luokkaa. Määrittelyluokka tarkoittaa luokkaa, jonka mukaan olion jäsenmuuttujien tilanvaraus tapahtuu.

- Esittelyluokka voi olla joko määrittelyluokka tai jokin määrittelyluokan yliluokka. Aliluokan olion määrittelyluokka on aina aliluokka.

- Aliluokan oliolle varattavan tilan koko määräytyy yliluokassa ja aliluokassa määriteltyjen jäsenmuuttujien yhteenlasketusta tavumäärästä.  Aliluokan oliolla on siis kaikki yliluokassa ja aliluokassa määritellyt tiedot.

- Esittely- ja määrittelyluokka ovat samat, jos olio on automaattinen.

- Esittely- ja määrittelyluokka voivat poiketa toisistaan, jos olion tunnus on osoitin tai viittaus olioon.

- Seuraavalla sivulla on esimerkkejä asiaan liittyen.

## Olio-ohjelmointi: Aliluokan esittely ja määrittely

- Seuraavassa käydään läpi käsitteet **esittelyluokka** ja **määrittelyluokka**. Näitä käsitteitä tarvitaan kun opiskellaan C++ -ohjelmointikielessä olevaa käsitettä **monimuotoisuus** ja **dynaaminen sidonta**.

- Esittelyluokka tarkoittaa olion tunnuksen luokkaa. Määrittelyluokka tarkoittaa luokkaa, jonka mukaan olion jäsenmuuttujien tilanvaraus tapahtuu RAM-muistissa.

- Esimerkkejä alla

  **Aliluokka olioAliluokka;**

  - Yllä olevalla rivillä on aliluokan automaattisen olion esittely ja määrittely. Esittely- ja määrittelyluokka on sama.

  **Aliluokka *olioAliluokka = new Aliluokka;**

  - Yllä olevalla rivillä on dynaamisen olio esittely ja määrittely. Tässäkin esittely- ja määrittelyluokka on sama.

  **Yliluokka *olio = new Aliluokka;**

  - Yllä olevalla rivillä olion esittely- ja määrittelyluokka poikkeaa toisistaan. Olion tunnus on esitelty yliluokkaan ja olio on määritelty aliluokkaan. Olion tilanvaraus tapahtuu siis aliluokan perusteella.

## Olio-ohjelmointi: Aliluokan esittely ja määrittely

- Esittelyluokka voi olla joko määrittelyluokka tai jokin määrittelyluokan yliluokka. Aliluokan olion määrittelyluokka on aina aliluokka.

- Aliluokan oliolle varattavan tilan koko määräytyy yliluokassa ja aliluokassa määriteltyjen jäsenmuuttujien yhteenlasketusta tavumäärästä.  Aliluokan oliolla on siis kaikki yliluokassa ja aliluokassa määritellyt tiedot.

- Esittely- ja määrittelyluokka ovat samat, jos olio on automaattinen.

- Esittely- ja määrittelyluokka voivat poiketa toisistaan, jos olion tunnus on osoitin tai viittaus olioon.

## Olio-ohjelmointi: Staattinen sidonta ja dynaaminen sidonta

- Ajettavan jäsenfunktion valinta ja sitominen jäsenfunktion kutsuun voi tapahtua ohjelman käännöksen aikana tai ohjelman ajon aikana.

- Jäsenfunktion sitomista kutsuun käännöksen aikana sanotaan **staattiseksi sitomiseksi** (Static Binding is also called Early Binding or Compile-time Binding).

- Ajon aikana tapahtuvaa sitomista kutsutaan **dynaamiseksi sidonnaksi** (Dynamic Binding is also called Late Binding or Runtime Binding)

- Sidontatapaan vaikuttavat tekijät
  Olion esittely- ja määrittelyluokka
  * Staattinen sidonta: olion tunnuksen esittelyluokan perusteella
  * Dynaaminen sidonta: olion määrittelyluokan perusteella

  Kutsutun jäsenfunktion laji:
  * Staattinen sidonta: tavallinen jäsenfunktio
  * Dynaaminen sidonta: virtuaalinen jäsenfunktio

## Olio-ohjelmointi: Dynaaminen sidonta

- Virtuaalisten jäsenfunktioiden dynaaminen ja osoittimien käyttö toteuttavat C++ -kielessä monimuotoisuuden.
- Monimuotoisuus tarkoittaa eri olioiden kykyä reagoida samaan viestiin eri tuloksella.
- Monimuotoisuus on piirre, jonka ansiosta tunnus voi osoittaa eri hetkinä saman periytymishierarkian eri luokkiin määriteltyihin olioihin. Saman tunnuksen kautta on mahdollista kutsua samannimisen jäsenfunktion eri rinnakkaistoteutuksia.
- Järjestelmä etsii toteutusratkaisua osoittimen osoittaman olion määrittelyluokasta.
- Monimuotoisuus lyhentää luokkahierarkiaa käyttävää ohjelmakoodia ja helpottaa ylläpitoa.
- Seuraavilla sivuilla tehdään esimerkkiprojekti asiaan liittyen.

## Polymorphism, Static Binding (Early Binding…) and Dynamic Binding (Late Binding…)

### Polymorphism

- Polymorphism means having multiple forms of one thing. In inheritance, polymorphism is done, by method overriding, when both base and sub class have member function with same declaration but different definition.

### Function Overriding

- 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

### Requirements for Overriding

- Inheritance should be there. Function overriding cannot be done within a class. For this we require a derived class and a base class. Function that is redefined must have exactly the same declaration in both base and derived class, that means same name, same return type and same parameter list.

## Exercise 1: Function Call Binding with class Objects

- Create a new project in name **myBinding**
- Connecting the function call to the function body is called **Binding**. When it is done before the program is run, its called Early Binding or Static Binding or Compile-time Binding.
- Static Binding means the compiler (or linker) is able to directly associate the identifier name (such as a function or variable name) with a machine address.
- Remove your code in **main.cpp** and add code on the right to the editor.
- Build and run the code.
- In this example, we are calling the overrided function using Base class and Derived class object. Base class object will call base version of the function and derived class's object will call the derived version of the function.

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

class Base
{
public:
   void show()
   {
      cout << "Base class" << endl;
   }
};
class Derived : public Base
{
public:
   void show()
   {
      cout << "Derived Class" << endl;
   }
};

int main()
{
   Base b;        //Base class object
   Derived d;     //Derived class object
   b.show();     //Static Binding Occurs
   d.show();    //Static Binding Occurs
}
```

OUTPUT
Base class
Derived class

## Exercise 1: Function Call Binding using Base class Pointer

- When we use a Base class's pointer or reference to hold Derived class's object, then Function call Binding gives some **unexpected results**.
- In the example, although, the object is of Derived class, still Base class's method is called. This happens due to Static Binding .
- Compiler on seeing Base class's pointer, set call to Base class's show() function, without knowing the actual object type.
- Edit your code in QtCreator as code in the right.
- Build and run the code.

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

class Base
{
public:
   void show()
   {
   cout << "Base class" << endl;
   }
};

class Derived:public Base
{
public:
   void show()
   {
      cout << "Derived Class" << endl;
   }
};

int main()
{
   Base* b;      //Base class pointer
   Derived d;    //Derived class object
   b = &d;       //d address to b
   b->show();    //Static Binding Occurs
}
```

OUTPUT
Base class

## Exercise 2: Dynamic Binding (Late Binding…)

### Virtual Functions

- Virtual Function is a function in base class, which is **overrided** in the derived class, and which tells the compiler to perform Dynamic Binding on this function.

- Virtual Keyword is used to make a member function of the base class **Virtual**.

### Dynamic Binding

- In Dynamic Binding function call is resolved at **runtime**. Hence, now compiler determines the type of object at runtime, and then binds the function call. Dynamic Binding is also called Late Binding or Runtime Binding.

- In the next page there is an example how to use Dynamic Binding

## Exercise 2: Dynamic Binding

### Using Virtual Keyword

- We can make base class's methods virtual by using virtual keyword while declaring them.
- **Virtual** keyword will lead to Dynamic Binding of that method.
- Using Virtual keyword with Base class's function, Late Binding takes place and the derived version of function will be called, because base class pointer pointes to Derived class object.
- Edit your code in QtCreator as code in the right.
- Build and run the code.

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

class Base
{
public:
   virtual void show()
   {
      cout << "Base class" << endl;
   }
};
class Derived : public Base
{
public:
   void show()
   {
      cout << "Derived Class" << endl;
   }
};

int main()
{
   Base* b;       //Base class pointer
   Derived d;     //Derived class object
   b = &d;
   b->show();     //Dynamic Binding Occurs
}
```

OUTPUT
Derived class