OOP-II
Arv

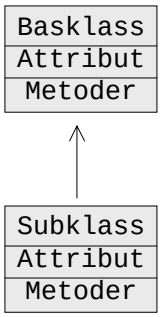
- Specialisering av en basklass
- UML-form
- En slags återanvändning av kod och datastrukturer
- Åtkomst mellan basklass/subklass
- Multipelt arv
- Abstrakta basklasser

Specialisering

- Nedåt i hierarkin - specialisering
- Uppåt - generalisering
- Exempel
  Fordon - Motorfordon - Bil

UML-form

- Basklassen (superklassen) överst (oftast)
- Pilen mot basklassen



Återanvändning

- En subklass kan *ärva* både attribut och metoder från sin basklass - då behöver den inte definiera dem
- En subklass kan *lägga till* attribut och metoder
- En subklass kan *definiera om* attribut och metoder
  - En Fordonsklass kan ha attributet *hastighet*
  - En subklass Bil ärver automatiskt hastighet, och kan lägga till t.ex. *antalPassagerare*
  - Om basklassen har metoden *tanka* så kan subklassen byta ut den mot en metod som tankar något speciellt (men den har samma signatur - omdefinition)
  - Förväxla inte med överlagring - olika signatur

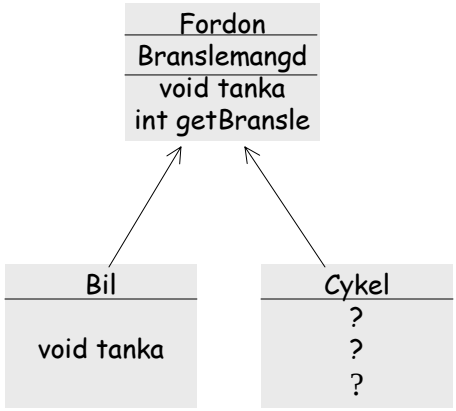
Arv i C++

```
class Fordon
{  private:
     int bransleMangd;
   public:
     void tanka(int liter);
     int getBransle();
};

class Bil:public Fordon
{
   private:
     int antalPass;
};
```


Omdefinition

```
class Fordon
{  private:
     int bransleMangd;
   public:
     void tanka(int liter);
     int getBransle();
};

class Bil:public Fordon
{
   private:
     int antalPass;
   public:
     void tanka(int liter);
};
```

Problem



Mer om arv

- När ej basklassens privata data om de är private

```
double Bil::berakna_rackvidd()
{
  return bransleMangd / 0.9; // Går ej!
}
```

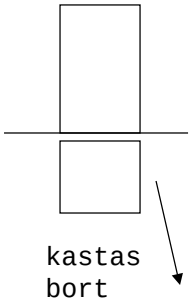
- Men kan anropa basklassens operationer

```
double Bil::berakna_rackvidd()
{
  return Fordon::getBransle() / 0.9; //Ok
}
```

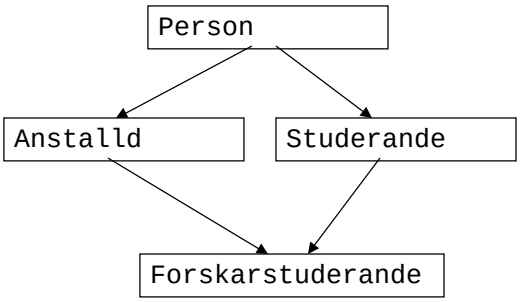
Typkonverteringar

- Konverteras om nödvändigt till basklass, "slicing"

```
Fordon f;
Bil b;

f = b;      // ok, 'slicing'
b = f;      // ej ok !
            // Odefinierat utrymme
```



Multipelt arv




```
class Forskarstuderande : public
Anstalld, public Studerande
{
};
```

Problem med multipelt arv

- Omdiskuterat behov
- Namnkonflikter
- Dubbla data (löses med virtuellt arv)
- Undvik!

Abstrakt basklass

- Fångar gemensamma egenskaper
- Påtvingar ett gränssnitt
- Med äkta virtuell funktion:

```
class Figur
{
public:
virtual double getArea() = 0;
};
```

- Eller med `protected` konstruktor