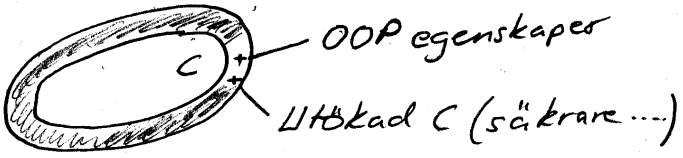
Objektorienterad programmering (OOP), 5p

Mål — OOP och C++



- händelsestyrd programmering i Windows

Medel — learning by doing

- stort projekt för att fördela arna med OOP syns

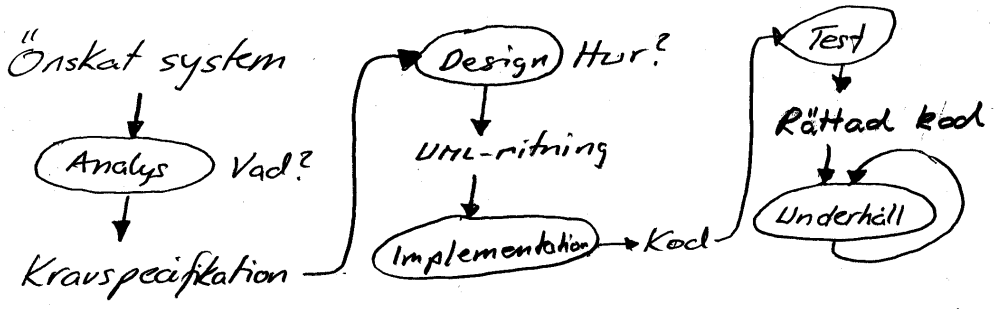
Litteratur — kompendium på nätet

- C++ Direkt, Skanstholm (Referens)

Kursinfo - på kurssidan på nätet

- delkurs 1, 3p (tentamen ger U, 3, 4 eller 5
- delkurs 2, 2p (projekt ger U, 3, 4 eller 5)
- slutbetyg blir lägsta betyg av tenta och projekt
- delkurs 2 ska vara inlämnad på nivå 3 senast på tentamensdagen. Kan senare kompletteras till högre betyg.

Kap 1, Objektorienterad konstruktion

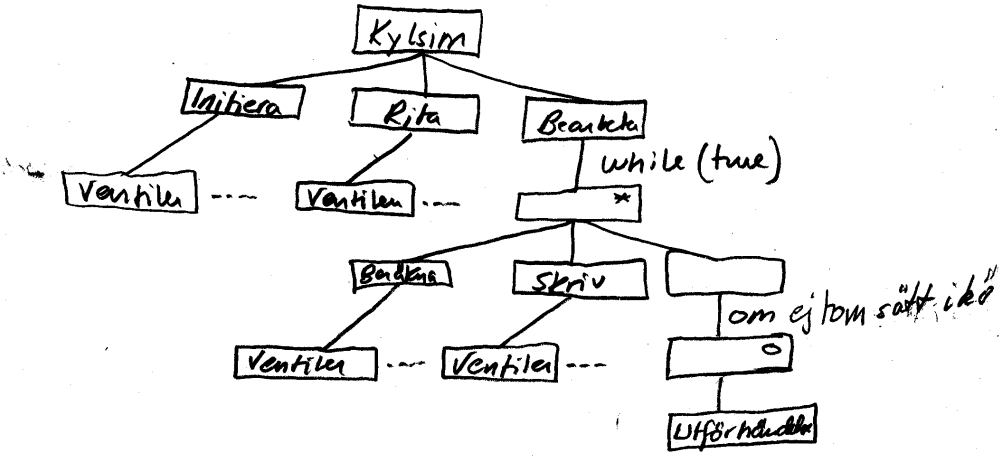


(Jfr med att bygga ett hus, lägenhet, skola ....)

UML = Unified Modelling Language

(Ex) Kylsim – kravspecifikation enligt kompendium

a) traditionell konstruktion (stegvis förfining)
- vad göra?
- vilka funktioner?



Nackdelar – utspridd data
– ingen bra modell av verkligheten

b) Objektorienterad konstruktion

- Vad hantera?
- vilka objekt?

① Leta efter objekt

Ventiler, filter, pump, noder...

② Data och operationer för objekten

Ventilen — flöde - beräkna ...
— öppning - stäng, öppna...

Noder — tryck — avläs, reglera ...

③ Relationer mellan objekt

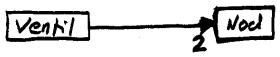
V1 frågar noderna n1 och n2 om trycket

V1 beräknar flödet

V1 skickar flödet till n1 och n2

n1 och n2 reglerar trycket så att summa flödet 0

Relation — ventilen ska känna till sina noder



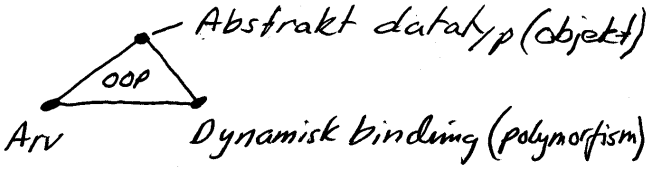
(UML)

④ Implementera — se nedan i C++

Fördelar — all data för ex. ventilerna finns på ett ställe
— modellen liknar verkligheten

Objektorienterade språk

Tre hörnstenar



Abstrakt datatyp – data + operationer (objekt)

(Ex) Läs in två personers namn och ålder och skriv ut personerna med den yngsta först.

a) **ANSI-C**

```
/* Person.h */
typedef struct
{
  char namn[20];
  int alder;
} person
```

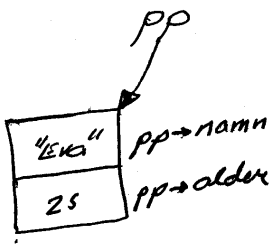
} data

```
void las_person(person *pp);
void skriv_person(person p);
int yngre_person(person p1, person p2);
```

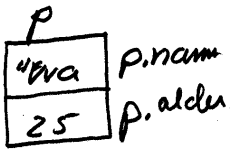
} Operationer

```
/* Person.c */
#include "Person.h"
#include <stdio.h>
```

```
void las_person(person *pp)
{
   printf("Ge personens namn: ");
   gets(pp->namn);
   printf("Ge personens ålder: ");
   scanf("%d", &pp->alder);
   getchar();
}
```

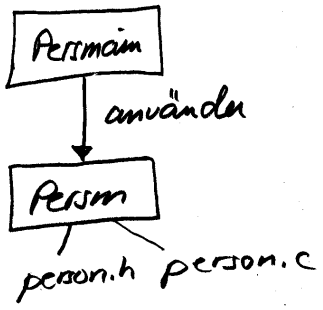



```
void skriv_person(person p)
{
   printf("Namn: %s\n", p.namn);
   printf("Ålder: %d\n", p.alder);
}
```



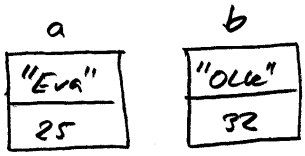

```
int yngre_person(person p1, person p2)
{
   return p1.alder < p2.alder;
}
```

```
/* Persmain.c */
#include "Person.h"
#include <conio.h>
```



```
void main()
{
   person a,b; /* Två personer a och b */

   las_person(&a);
   las_person(&b);
   if ( yngre_person(a,b))
   {
      skriv_person (a);
      skriv_person (b);
   }
   else
   {
      skriv_person(b);
      skriv_person(a);
   }
   getch();
}
```



OBS! Oskyddad data!
Man kan ex.
skriva a.alder++!

b) C++

```
//Specifikation -- person.h

class PERSON
{  private:
      char namn[20];
      int alder;
   public:
      void las();
      void skriv();
      bool yngre(PERSON P);
};  OBS!
    semikolon
```

} Data (medlemsvariabler)

} Operationer (medlemsfunktioner)

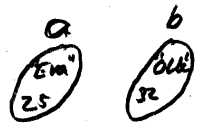
OBS! bool kan värdet
falsk eller true!

```
//Implementation -- Person.cpp
#include "Person.h"
#include <ioshecum>  //OBS! Inget .h då nya bibliotksfiler inkluderas. Använd istället
using namespace std;  namespace std och ta in gamla C-filer med <cmath>. Dina egna
                      filer ska fortfarande heta .h
void PERSON::Las()
{
    cout << "Ge personens namn: ";
    cin.getline(namn, sizeof(namn));
    cout << "Ge personens ålder: ";
    cin >> alder;
    cin.ignore();
}

void PERSON::skriv()
{
   cout << "Namn: " << namn << endl;
   cout << "Ålder: " << alder << endl;
}

bool PERSON::yngre(PERSON p)
{
   return alder < p.alder;  //OBS! Objektets ålder
}                           jämförs med
                            parameterns ålder!
```

```
// Huvudprogram -- Persmain.cpp
#include "Persm.h"
#include <conio.h> // Ingen std-modul
void main()
{
  PERSON a,b;  // Två objekt definieras
  a.Las();
  b.Las();
  if (a.yngre(b))
  {
    a.skriv();
    b.skriv();
  }
  else
  {
    b.skriv();
    a.skriv();
  }
  getch();
}
```

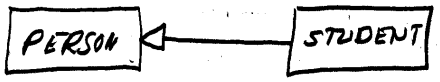


// OBS! objekt.operation!

// OBS! Skyddar data med privat!
Man kan inte öka ålder med
a.alder++;
utan man måste skriva en medlemsfunktion öka och sedan anropa den med
a.oka();
b.oka();

Arv

(Ex) En student är en person med poäng.



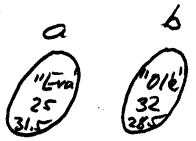
UML-diagram

```
// Specifikation -- Student.h
#include "Person.h"
class STUDENT : public PERSON
{  private:
      float poang;
   public:
      void las();
      void skriv();
};
// Implementation -- Student.cpp

#include "Student.h"
#include <iostream>
using namespace std;
VOID STUDENT::las()
{
    PERSON::las();
    cout << "Ge studentens poäng : ";
    cin >> poang;
    cin.ignore();
}
```

```
void STUDENT::skriv()
{
    PERSON::skriv();
    cout << "Poäng : " << poang << endl;
}

// Huvudprogram -- Studmain.cpp
#include "Student.h"
#include <conio.h>
void main()
{
    STUDENT a, b;
    a.las();
    b.las();
    if (a.yngre(b))
    {
        a.skriv();
        b.skriv();
    }
    else
    {
        b.skriv();
        a.skriv();
    }
    getch();
}
```



OBS! Funktionen yngre ärvs från PERSON-klassen och behöver ej skrivas i STUDENT-klassen!

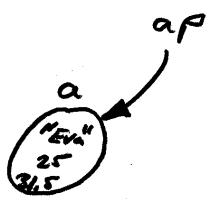
Vill man däremot jämföra poäng måste man skriva en mindre än funktion enligt:

```
//Specifikation -- Student.h
:
class STUDENT : public PERSON
{
    :
    public:
    :
    bool mindre(STUDENT s);
};
// Implementation -- Student.cpp
:
bool STUDENT::mindre(STUDENT s)
{
  return  poang < s.poang;
}

//Huvudprogram -- Studmain.cpp
:
void main()
{
   STUDENT a,b;
   a.las();
   b.las();
   if (a.mindre(b))
   {
      :
```

Dynamisk bindning

(Ex)
```
void main()
{
    STUDENT a;
    PERSON *ap;
    a.Las();
    ap = &a; // detta är en tillåten tilldelning
    a.skriv(); // Skriver ut namn, ålder och poäng
    ap->skriv(); // Skriver bara ut namn och ålder
                 eftersom ap är en pekare till
                 basklassen PERSON kör skriv-
                 funktion för PERSON-klassen!
```



Med dynamisk bindning får man basklass-pekaren att binda rätt funktion till sig och skriva ut namn, ålder och poäng.

```
;
class PERSON
{
  :
  virtual void skriv(); // Virtual markerar
                           att skriv-funktionen
  :                        ska bindas dynamiskt
};
```

Hemuppgift: Utöka PERSON-klassen med en medlemsfunktion fodelsedag som ökar medlemsvariabeln alder med ett samt med medlemsfunktionen lika-gamla som returnerar sant om samma ålder.