INTERNATIONELLA
HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

OO Analys och Design

Objektorienterade språk och implementation

Ulf Seigerroth
Internationella Handelshögskolan
Jönköping

INTERNATIONELLA
HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

OO språk och implementation

- Om designfasen är ordentligt genomarbetad blir implementeringen i ett objektorienterat språk ganska enkelt.
- Generellt så stödjer de OO språken objekt, polymorfism och arv
- Från objektmodellen får vi främst deklarativa strukturer såsom
  - Specifikationer av klasser
  - Attribut
  - Arvshierarkier
  - Associationer
- Den dynamiska modellen specificerar
  - Kontrollstrategier
  - Typ av system (procedure, händelse eller multitasking)
- Den funktionella modellen definierar
  - Det som skall implementeras i metoderna

INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

- Det första som implementeras är objektmodellen genom att klasserna specificeras.

```
class Strang {              // Klassspecifikation
public:                     // Skyddsnivå
    Strang();               // Konstruktor
    Strang(char *init);     // Konstruktor med argument
    ~Strang();              // Destruktor
protected:                  // Skyddsnivå

private:                    // Skyddsnivå
char        *str;           // Intern datastruktur
};
```


INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

Skapande av objekt

- När objekt skapas skall de alltid ha
  - Tillstånd (attribut)
  - Operationer (metoder)
  - Identitet

- Objekt kan skapas och tas bort på två sätt
  - Statiskt
  - Dynamiskt

INTERNATIONELLA
HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

```
class Strang { // Klassspecifikation
public:
    Strang();
    Strang(char *init);
    ~Strang();
protected:
private:
char *str;
};

Strang::Strang(char *init) // Konstruktor
{
    str = new char[strlen(init) + 1];
    strcpy(str,init);
}

Strang::~Strang() // Destruktor
{
    delete str;
}
```


INTERNATIONELLA
HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

```
#include <iostrem.h>
void main(void)
{
    Strang      ObjEtt;    // Statiskt skapad förekomst utan innehåll
    Strang      Kalle(”Kalle”);// Statiskt skapad förekomst med innehåll

    Strang      *StrPek;  // En pekare för typen Strang
    StrPek = new Strang(”Olle”);      // Dynsamiskt skapad förekomst
    delete StrPek;                    // Avallokering av Strang
    StrPek = new Strang(”Pelle”);
    delete StrPek;
}
```

INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

Anrop av operationer

Två sätt att anropa operationer:

- Direkt mot en förekomst med punktnotation
- Med objektpekare genom (->)

INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

Samma specifikation som tidigare:

```
class Strang { // Klassspecifikation
public:
      Strang();
      Strang(char *init);
      ~Strang();
      void SkrivUt(void);// Metod för att skriva ut Strang
protected:
private:
char *str;
};


void Strang::SkrivUt(void)              // Implementation av SkrivUt
{
      --
      --
}
```

INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

```
#include <iostrem.h>
void main(void)
{
    Strang          Kalle(”Kalle”);
    Strang          *StrPek  = new Strang(”Olle”);

    Kalle.Skrivut(); // Utskrift av Kalle

    StrPek->Skrivut();          // Utskrift av Olle

    StrPek = &Kalle;            // Ställa pekaren mot Kalle
    StrPek->Skrivut();          // Utskrift av Kalle

    delete StrPek;   // Avallokera Olle
}
```


INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

This pekaren

- Varje objekt har sin egna pekare inom objektets fokus (this)
- I C++ kan man definiera friend funktioner och klasser som har full access i de klasser där de är definierade
- Om operationerna delas, hur vet då operationen vilket objekts datastruktur som skall bearbetas
  - Detta löses med this pekaren som alltid har fokus på aktuellt objekt (adressen till det egna objektet)

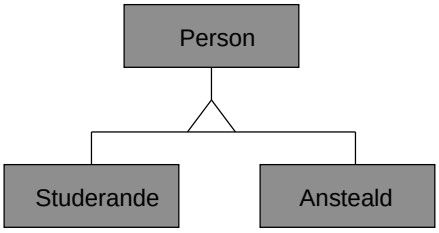
```
Screen& Screen::Clear(char Bakgrund)
{
    char *p = Cursor;
    while(*p) *p++ = Bakgrund;
    return *this;
}
```

INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

Arv

```
class Person {
public:
      Person();
      ~Person();
protected:
      char      PersNr[11];
      char      Namn[15];
private:
};
```



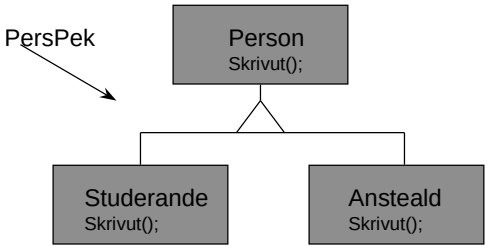

```
class Ansteald : public Person
{
public:
      Ansteald();
      ~Ansteald();
protected:
private:
      int       loen;
};
```

```
class Stederande : public Person
{
public:
     Studerande();
      ~Studerande();
protected:
private:
      char      Program[15];
};
```


INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

- En pekare till en basklass kan peka på samtliga subklasser i en arvshierarki



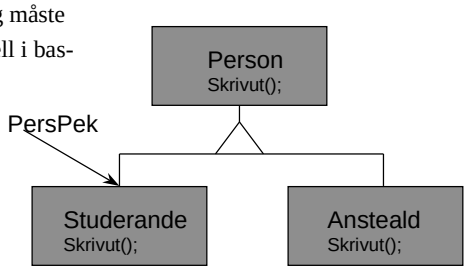

```
void main(void)
{
      Person     *PersPek;
      Anstead      Sven(.....);

      PersPek = &Sven;
      PersPek->SkrivUt(); // Persons skrivut kommer att anropas (Ej önskvärt!!)
}
```

- Lösningen på problemet heter dynamisk bindning
- Beroende på vilket objekt som vi pekar på så skall detta objekts implementation av skrivut() anropas

INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

- För att uppnå dynamisk bindning måste funktionen deklareras som virtuell i bas-klassen




```
class Person
{
public:
    virtual void SkrivUt(void);
    --
    --
};
```

- Om implementation ej finns i subklassen när vi pekar på den sker sökning uppåt
- I den klass där funktionen är deklarerad som virtuell måste det finnas en implementation (en målvakt)

INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

Användande av dynamisk bindning

```
void main(void)
{
    Person          *PersPek;

    Anstеald    Sven(.....);
    Studerande  Ulla(.....);

    PersPek = &Sven;
    PersPek->SkrivUt();     // Svens skrivut kommer att anropas (Yes!!)

    PersPek = &Ulla;
    PersPek->SkrivUt();     Ullas skrivut kommer att anropas
}
```

INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

Abstrakta basklasser

- När vi inte tänkt att klassen skall bli någon förekomst
- Abstrakta basklasser kan skapas på två sätt
  - En metod görs till äkta virtuell
  - Konstruktorn i klassen deklareras som protected

```
class Person {
public:
    virtual voidSkrivUt() = 0;
protected:
    Person();
};
```

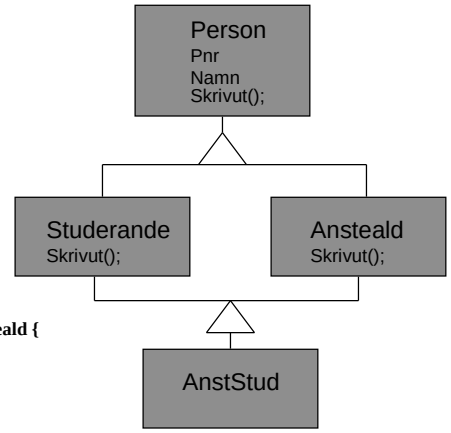
- SkrivUt() behöver ej implementeras i basklassen
- Tvingar subklasserna till implementation av SkrivUt()

INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

Multipelt arv

```
class Person {
--
};

class Studerande:public Person {
--
};

class Anstеald:public Person {
--
};

class AnstStud:public Studerande, public Anstеald {
--
};
```



INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

- Två problem
  - Namnkonflikt
  - Upprepat arv

- Lösningen på namnkonflikt är egen implementation
- Lösningen på upprepat arv är att låta Studerande och Anstеald ärva basklassen virtuellt

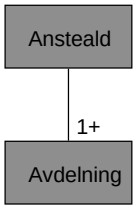
```
class Studerande:virtual public Person {
--
};

class Anstеald:virtual public Person {
--
};
```


INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

Associationer

- Associationer har inget direkt syntaktiskt stöd i C++
- Associationer implementeras i C++ med hjälp av pekare till andra typer av klasser




```
class Avdelning {
public:
    Avdelning(char *Namn);
    void SkrivAvd();
--
};

class Anstеald {
public:
    Anstеald(char *Namn, Avdelning *Avd)
    { // Association skapas
        TillhorAvd = Avd;
    }
private:
    Avdelning  *TillhorAvd;
};
```

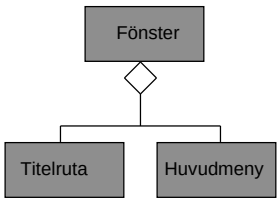
```
void main(void)
{
    Avdelning Tillv("Tillverkning");
    Anstеald Nisse("Nils Nilsson", &Tillv);
}
```

INTERNATIONELLA HANDELSHÖGSKOLAN
HÖGSKOLAN I JÖNKÖPING

Implementation

Aggregat

- Aggregatrelationen avspeglar att en klass eller objekt byggs upp av andra klasser eller objekt




```
class Fonster {
public:
    Fonster();
    ~Fonster();
private:
    Titelruta     Titel;   // Aggregat
    Huvudmeny     Hmeny;   // Aggregat
};
```

- När en förekomst av fönster skapas så skapas automatiskt först förekomsterna Titel och Hmeny