TDP004

OOA, Arv, Polymorfi

Eric Ekström

Institutionen för datavetenskap

Handout version

Vektorer

- En vektor lagrar en sekvens av värden
- Vi måste ange datatypen för värden som ska lagras i sekvensen
- Alla värden har samma datatyp.

```
#include <vector>

int main()
{
  std::vector<int> v { 1, 5, 2 };
  v.push_back(8);
  std::cout << v.front() << v.at(1) << v.back() << std::endl;
  return 0;
}
```

Vad mer kan man göra med en vektor? Kolla cppreference

LINKÖPING UNIVERSITY

Argument till main

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

int main(int argc, char* argv[])
{
  cout << "nr of arguments: " << argc << endl;
  cout << argv[0] << " : " << argv[1] << endl;
  return 0;
}
```

```
$ ./a.out Hej!
nr of arguments: 2
./a.out : Hej!
```

Argument till main

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

int main(int argc, char* argv[])
{
  istringstream iss { argv{1} };
  double argument1 {};
  iss >> argument1;
  cout << argument1 << endl;

  int argument2 { stoi(argv[2]) };
  cout << argument2 << endl;
  return 0;
}
```

Fördelen med att använda strängströmmar istället för stoi är att inga undantag kan kastas, vilket gör felhantering enklare. Vi har också samma flexibilitet som för annan inläsning.

Argument till main

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

int main(int argc, char* argv[])
{
  if (argc != 2)
  {
    cerr << "Error: Fel antal argument" << endl;
    return 1;
  }

  istringstream iss { argv{1} };
  double argument1 {};
  if (not iss >> argument1)
  {
    cerr << "Error: Inargumentet måste vara ett flyttal" << endl;
    return 2;
  }
}
```

Objektorienterad analys

En metod för att designa stora system.

Utgår från en kravspecifikation eller projektbeskrivning.

1. Finn objekten
2. Klassificera objekten
3. Beskriv relationer
4. (Identifiera aktörer och användningsfall)

Objektorienterad analys - Exempel

“Vi vill bygga ett ritprogram likt MS-Paint. Man ska kunna rita ut olika former (tex cirklar och rektanglar) med olika storlek och färg. Programmet ska hålla koll på alla former som ritats ut (lite som lager i Photoshop), och man ska kunna plocka bort enstaka former från sin bild utan att det påverkar andra.”

Objektorienterad analys - Steg 1

- Finn all objekt som behöver representeras.

“Vi vill bygga ett ritprogram likt MS-Paint. Man ska kunna rita ut olika former (tex cirklar och rektanglar) med olika storlek och färg. Programmet ska hålla koll på alla former som ritats ut (lite som lager i Photoshop), och man ska kunna plocka bort enstaka former från sin bild utan att det påverkar andra.”

Ett förslag till hur man ska finna objekt är att läsa kraspecifikationen och notera förekommande substantiv.

- Rit-programmet
- Rektanglar
- Cirklar

Objektorienterad analys - Steg 2

- Klassificera objektet m.h.a CRC.

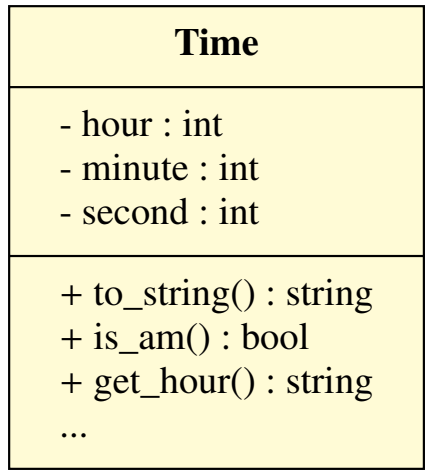
Objekt	Namn	Ansvar	Samarbete
Rit-programmet	Paint	Hålla koll på alla former Rita ut allt	Rect Circle
Rektanglar	Rectangle	Sin position och storlek Rita ut sig	Paint
Cirklar	Circle	Sin position och storlek Rita ut sig	Paint

Skriv ned en klass för varje identifierat objekt. Ett CRC-kort innehåller klassens namn, dess ansvar och vilka andra klasser den samarbetar med. Här gör vi CRC som en tabell istället. Samma innehåll, annat format.

Objektorienterad analys - Steg 3

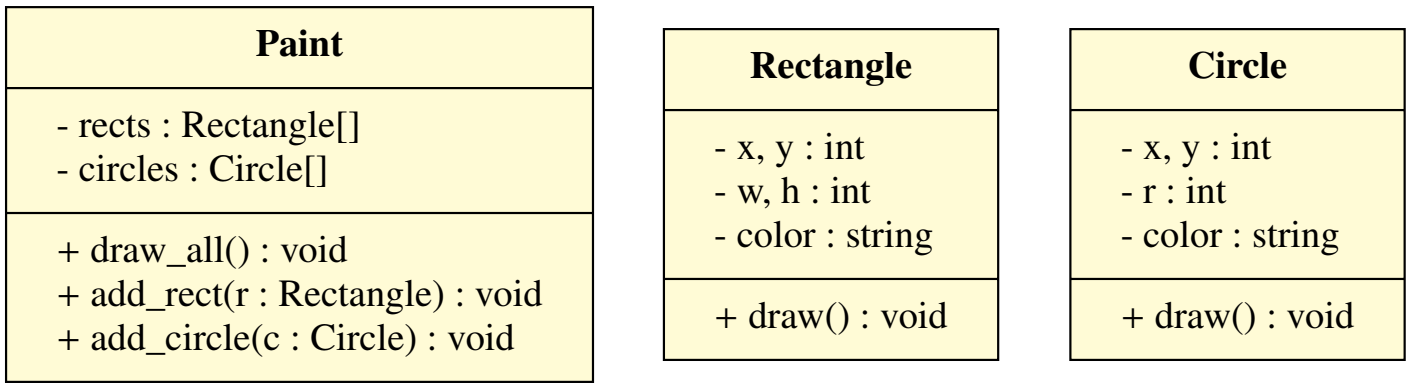
- Detaljbeskrivning av klasser och deras relationer

UML-diagram

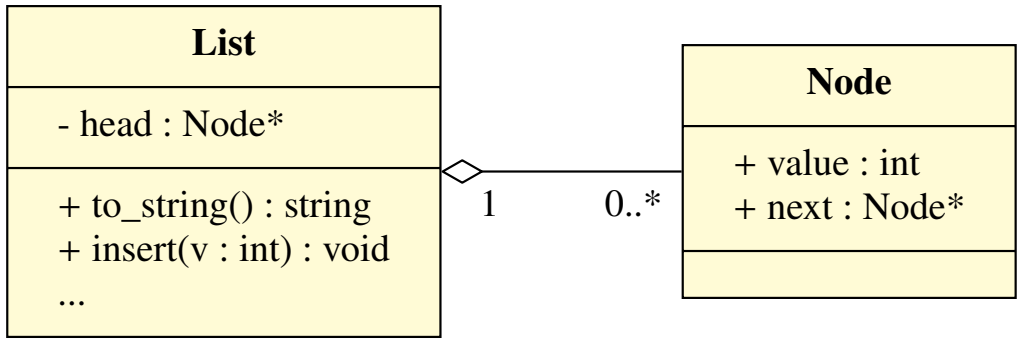

Time
- hour : int
- minute : int
- second : int
+ to_string() : string
+ is_am() : bool
+ get_hour() : string
...

Objektorienterad analys - Steg 3

- Detaljbeskrivning av klasser och deras relationer

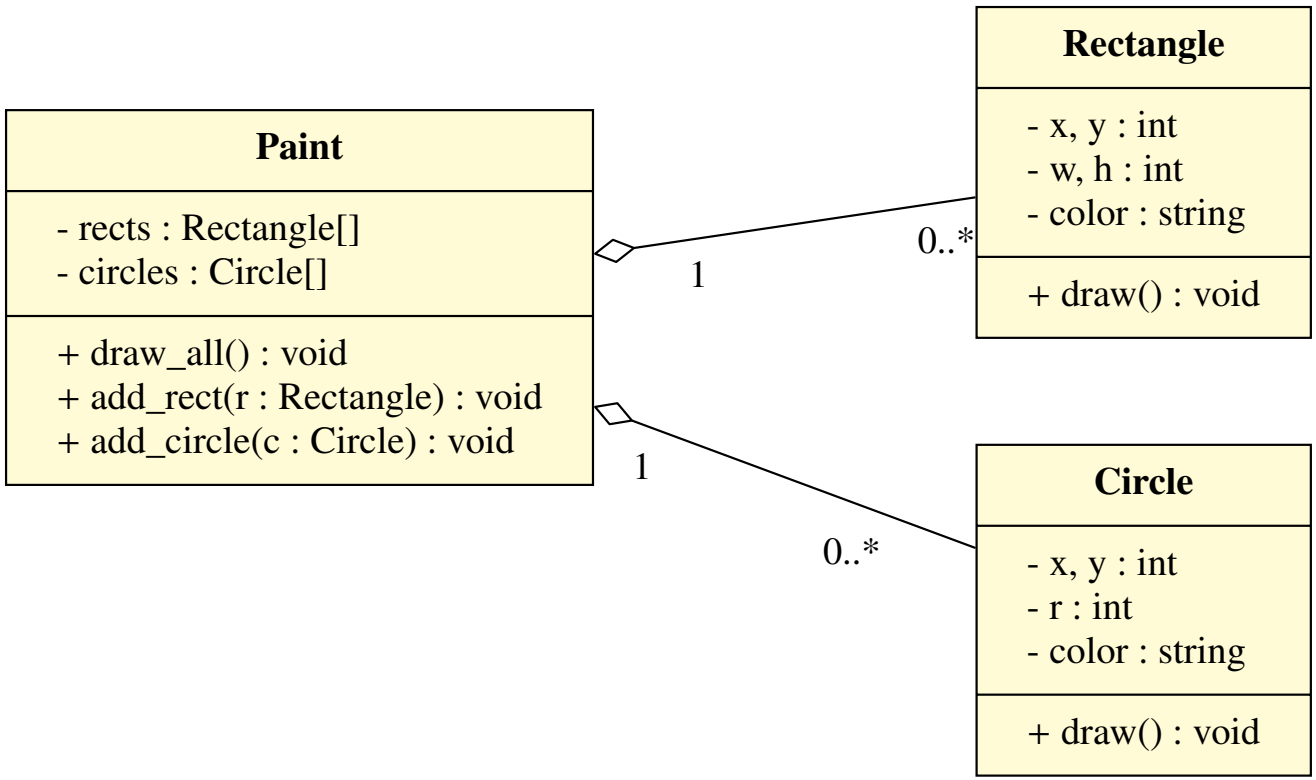


UML-diagram


List
- head : Node*
+ to_string() : string
+ insert(v : int) : void
...
1
0..*
Node
+ value : int
+ next : Node*

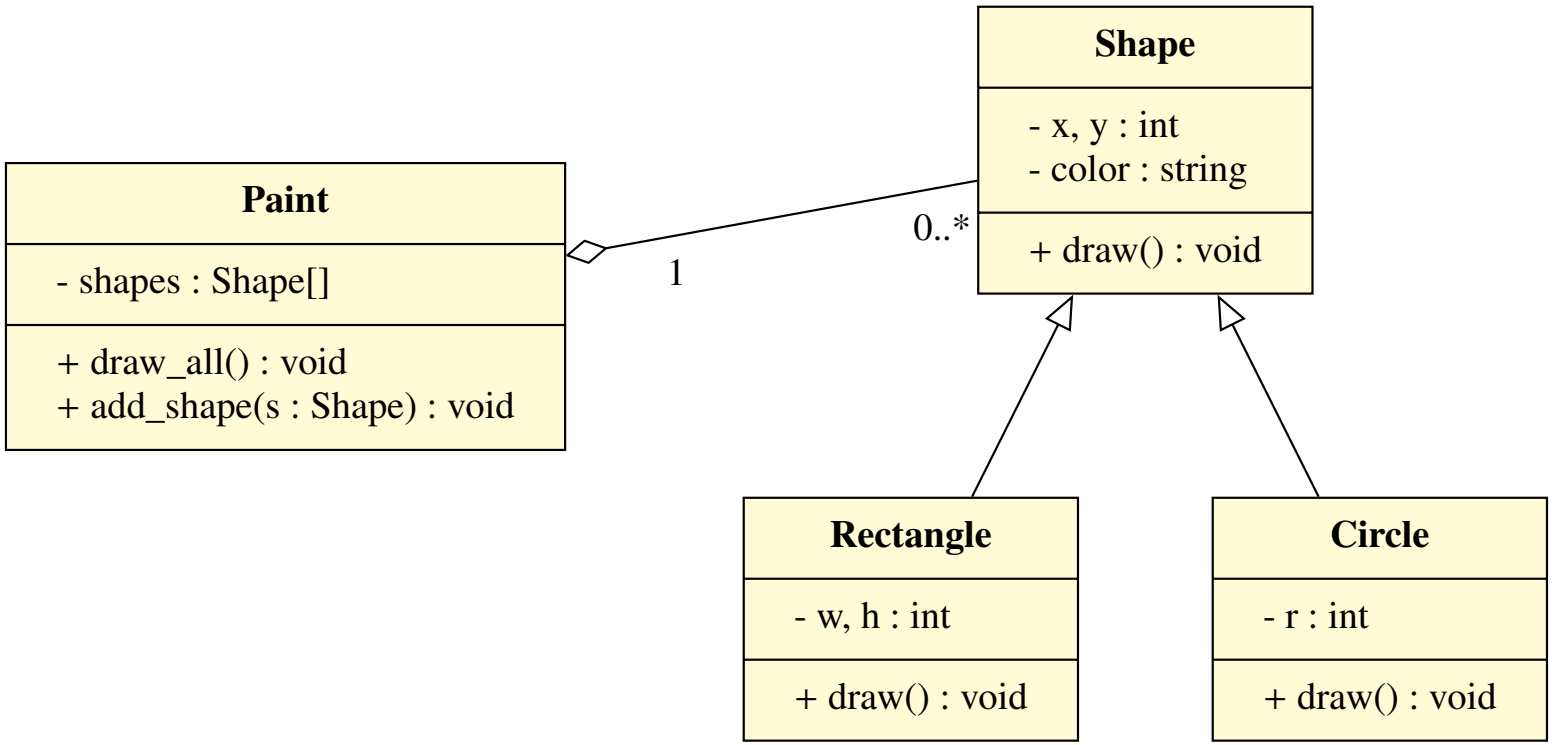
Objektorienterad analys - Steg 3

- Detaljbeskrivning av klasser och deras relationer



Objektorienterad analys - UML

- Komplett UML-diagram med arv



1 Listor i C++
2 Argument till main
3 Objektorienterad analys
4 Problembeskrivning
5 Specialisering (Arv)
6 Statisk binding
7 Synlighet

LINKÖPING UNIVERSITY

Objektorienterad analys - Steg 3

- Duplicering i klasser
- Omständligt att lägga till fler klasser
- Går inte att hantera alla klasser generellt

1 Listor i C++
2 Argument till main
3 Objektorienterad analys
4 Problembeskrivning
5 Specialisering (Arv)
6 Statisk binding
7 Synlighet

Specialisering - Två snarlika klasser

```
class Rectangle
{
public:
  Rectangle(int x, int y,
            int w, int h);

  float area() const;
  void print_pos() const;

private:
  int xpos;
  int ypos;
  int w;
  int h;
};
```

```
class Circle
{
public:
  Circle(int x, int y,
         float r);

  float area() const;
  void print_pos() const;

private:
  int xpos;
  int ypos;
  float r;

};
```

Dåliga alternativ:

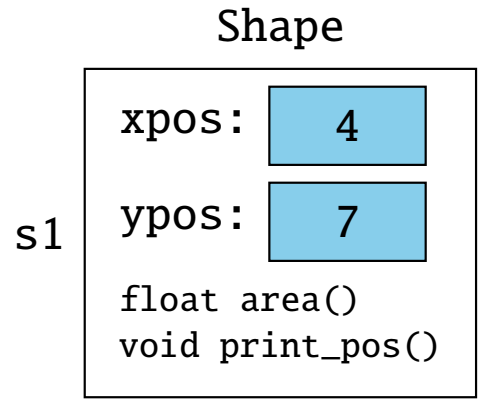
- Kopiera koden från `Rectangle`
- Lägg till den nya koden direkt i `Rectangle`
- Gör `Rectangle` till en medlem i `Circle`

Specialisering - Basklass

```
class Shape
{
public:
  Shape(int x, int y)
    : xpos {x}, ypos {y}
  {}

  float area() const;
  void print_pos() const;

private:
  int xpos;
  int ypos;
};
```



Allt som var gemensamt för Rectangle och Circle lägger vi istället i klassen Shape. Shape blir en basklass.

```
int main()
{
  Shape s1 {4, 7};
}
```

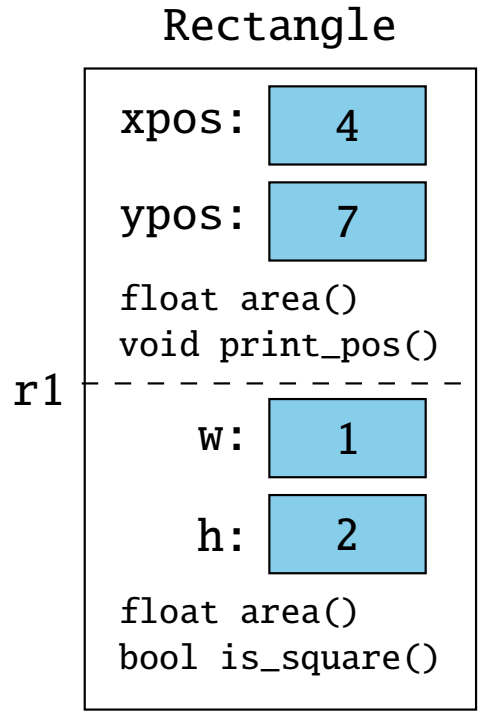
Specialisering - Basklass

```
class Rectangle : public Shape
{
public:
  Rectangle(int x, int y,
            int w, int h)
    : Shape {x, y}, w {w}, h {h}
  {}

  float area() const;
  bool is_square() const;

private:
  int w;
  int h;
};
```

```
int main()
{
  Rectangle r1 {4, 7, 1, 2};
}
```



`Rectangle` ärver från `Shape` och blir då en härledd klass.
Alla medlemmar som fanns i `Shape` är nu också medlemmar i `Rectangle`.
Funktioner i `Rectangle` kan ersätta implementationer från basklassen.

Specialisering - Härledd klass

- Härledda klassen ska alltid initiera sin basklassdel genom att anropa basklassens konstruktor

```
Rectangle(int x, int y, int w, int h)
  : Shape {x, y}
{}
```

- Vi kan göra explicita anrop till basklassens version av en funktion.

```
Shape::area()
```

- Basklassens privata medlemmar går inte att komma åt från den härledda klassen.

1 Listor i C++
2 Argument till main
3 Objektorienterad analys
4 Problembeskrivning
5 Specialisering (Arv)
6 Statisk binding
7 Synlighet

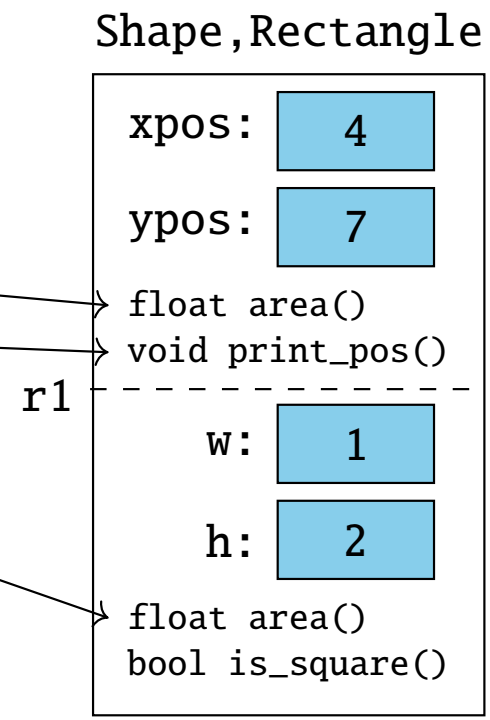
Statisk bindning

Hur vet kompilatorn vilken funktion som ska anropas?

```
Rectangle r1 {4, 7, 1, 2};

Shape& s1 {r1};

cout << s1.is_square() error
     << s1.area()
     << r1.is_square()
     << r1.print_pos()
     << r1.area();
```



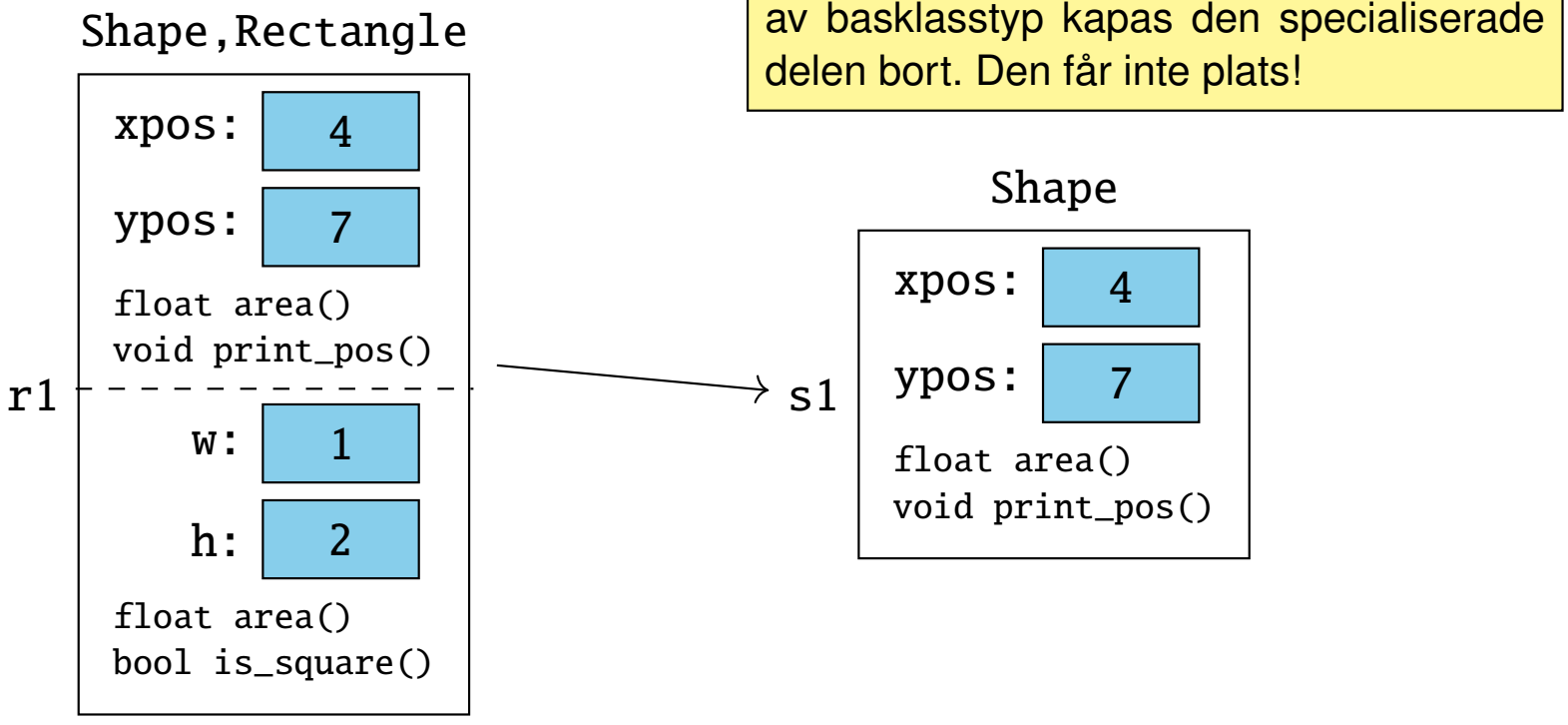
- Vilken funktion som anropas avgörs vid kompilering genom att titta på objektets deklaration i koden.
- Det spelar ingen roll vilken typ av objekt som faktiskt ligger i minnet.
- Standard i C++ om inget annat anges.

Slicing

```
Rectangle r1 {4, 7, 1, 2};

Shape s1 {r1}; // Kopia!
```

När ett härlett objekt kopieras till en varabel av basklasstyp kapas den specialiserade delen bort. Den får inte plats!



1 Listor i C++
2 Argument till main
3 Objektorienterad analys
4 Problembeskrivning
5 Specialisering (Arv)
6 Statisk binding
7 Synlighet

LINKÖPING UNIVERSITY

Synlighet

I en klass kan vi sätta synlighet på medlemmarna.

- public - tillgänglig utanför klassen
- private - endast tillgänglig inom klassen.
- protected - som private men tillgänglig från härledda klasser.

Hur blir det med arv?

Ibland vill vi att härledda klasser ska komma åt medlemmar, men ingen utanför klasshierarkin ska komma åt dem.

Synlighet

```
class Person
{
public:

  Person(string const& n,
         string const& p);
  string get_phone() const;
  void print_info(ostream& os) const;

protected:
  string name;
  string phone;

};
```

```
class Employee : public Person
{
public:

  Employee(string const& n,
           string const& p,
           string const& w,
           int s);

  string get_phone() const
  {
    return phone + work_phone;
  }

private:
  string work_phone;
  int salary;

};
```

Synlighet

Varför behövs `public` vid arv?

```
class Employee : public Person
```

Vad händer om vi skriver något annat?

- `public` - medlemmar i basklassen behåller deklarerat skydd i härledd klass.
- `protected` - publika medlemmar blir skyddade i härledd klass.
- `private` - alla medlemmar i basklassen blir privata i härledd klass.

Om inget deklareras väljs `private`.

www.ida.liu.se/~TDP004/