

TDP004

OOA, Arv, Polymorfi

Eric Ekström

Institutionen för datavetenskap

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

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](#)

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

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;
}
```

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;
    }
}
```

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

Objektorienterad analys

En metod för att designa stora system.

Utgår från en kravspecifikation eller projektbeskrivning.

Objektorienterad analys

En metod för att designa stora system.

Utgår från en kravspecifikation eller projektbeskrivning.

1. Finn objekten

Objektorienterad analys

En metod för att designa stora system.

Utgår från en kravspecifikation eller projektbeskrivning.

1. Finn objekten
2. Klassificera objekten

Objektorienterad analys

En metod för att designa stora system.

Utgår från en kravspecifikation eller projektbeskrivning.

1. Finn objekten
2. Klassificera objekten
3. Beskriv relationer

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.”

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.”

- Rit-programmet
- Rektanglar
- Cirklar

Objektorienterad analys - Steg 2

- Klassificera objektet m.h.a CRC.

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

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

Objektorienterad analys - Steg 2

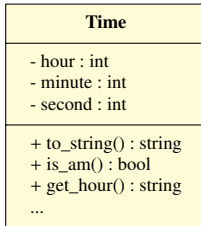
- Klassificera objektet m.h.a CRC.

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

Objektorienterad analys - Steg 3

- Detaljbeskrivning av klasser och deras relationer

UML-diagram



Objektorienterad analys - Steg 3

- Detaljbeskrivning av klasser och deras relationer

Objektorienterad analys - Steg 3

- Detaljbeskrivning av klasser och deras relationer

Rectangle
- x, y : int - w, h : int - color : string
+ draw() : void

Objektorienterad analys - Steg 3

- Detaljbeskrivning av klasser och deras relationer

Rectangle
- x, y : int - w, h : int - color : string
+ draw() : void

Circle
- x, y : int - r : int - color : string
+ draw() : void

Objektorienterad analys - Steg 3

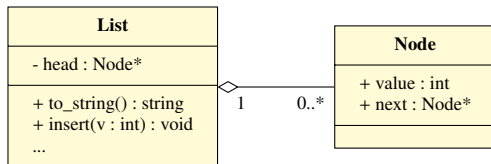
- Detaljbeskrivning av klasser och deras relationer

Paint
- rects : Rectangle[] - circles : Circle[]
+ draw_all() : void + add_rect(r : Rectangle) : void + add_circle(c : Circle) : void

Rectangle
- x, y : int - w, h : int - color : string
+ draw() : void

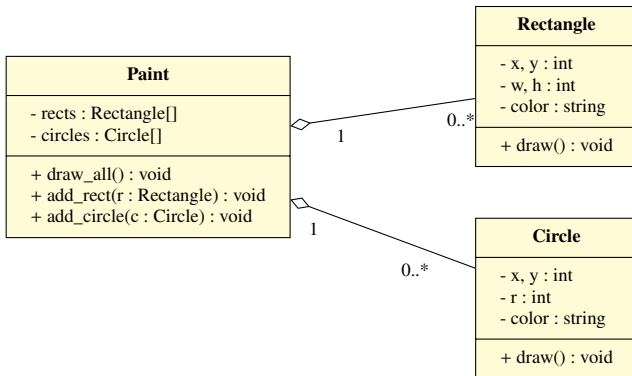
Circle
- x, y : int - r : int - color : string
+ draw() : void

UML-diagram



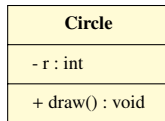
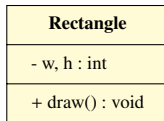
Objektorienterad analys - Steg 3

- Detaljbeskrivning av klasser och deras relationer



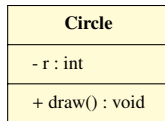
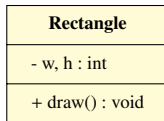
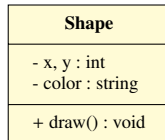
Objektorienterad analys - UML

- Komplett UML-diagram med arv



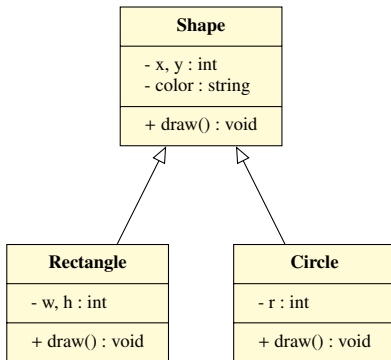
Objektorienterad analys - UML

- Komplett UML-diagram med arv



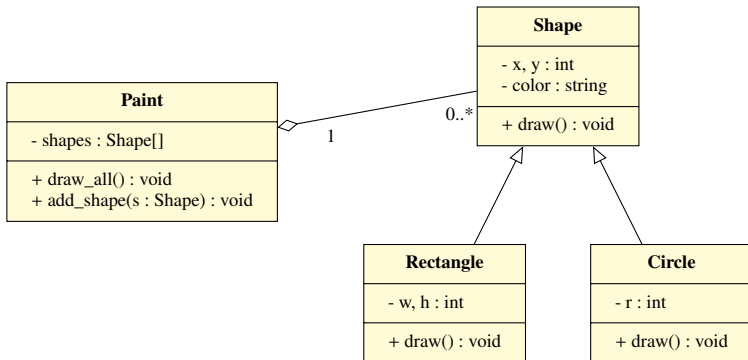
Objektorienterad analys - UML

- Komplett UML-diagram med arv



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

Objektorienterad analys - Steg 3

- Duplicering i klasser

Objektorienterad analys - Steg 3

- Duplicering i klasser
- Omständligt att lägga till fler klasser

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;
};
```

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;
};
```

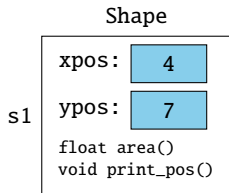
Specialisering - Basklass

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

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

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

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



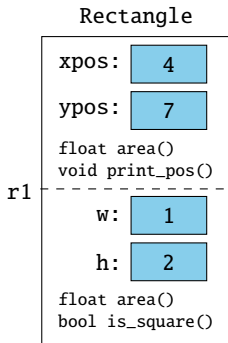
Specialisering - Basklass

```
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};
}
```



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}  
{}
```

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()
```

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

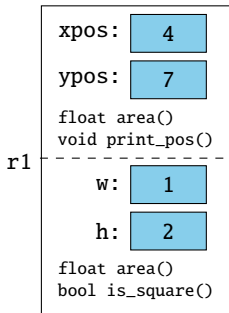
Hur vet kompilatorn vilken funktion som ska anropas?

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

Shape& s1 {r1};

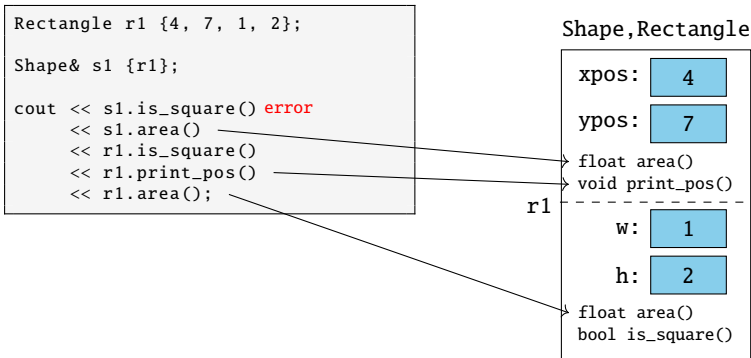
cout << s1.is_square()
      << s1.area()
      << r1.is_square()
      << r1.print_pos()
      << r1.area();
```

Shape, Rectangle



Statisk bindning

Hur vet kompilatorn vilken funktion som ska anropas?

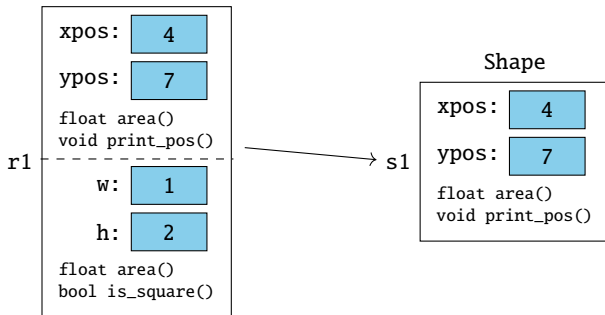


Slicing

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

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

Shape, Rectangle



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

Synlighet

I en klass kan vi sätta synlighet på medlemmarna.

- public - tillgänglig utanför klassen
- private - endast tillgänglig inom klassen.

Hur blir det med arv?

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

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.

Synlghet

```
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 deklarerats väljs `private`.

www.ida.liu.se/~TDP004/