

TDP004

OOA, Arv, Polymorfi

Eric Ekström

Institutionen för datavetenskap

Övning 1

Givet på kurshemsidan är två klasser. Skapa en lämplig basklass och bryt ut det som är gemensamt.

- 1 Polymorfi
- 2 Typkonvertering
- 3 UML-diagram

Härledda klasser

Vilken funktion kommer anropas (med statisk bindning)?

```
vector<Shape*> v {};  
  
v.push_back(new Shape{...});  
v.push_back(new Rectangle{...});  
v.push_back(new Circle{...});  
  
for (Shape* s : v)  
{  
    cout << s->area() << endl;  
}
```

Polymorfi

Vi vill att varje härledd klass ska ha sin egen specialiserade implementation av en funktion.

Vi vill att den härledda klassens implementation anropas automatiskt istället för basklassens.

Dynamisk bindning

```
class Shape
{
public:
    ...

    virtual float area() const
    {
    }
};
```

```
class Rectangle : public Shape
{
public:
    ...

    float area() const override
    {
        return x * y;
    }
};
```

Härledda klasser

Vilken funktion kommer anropas (nu med dynamisk bindning)?

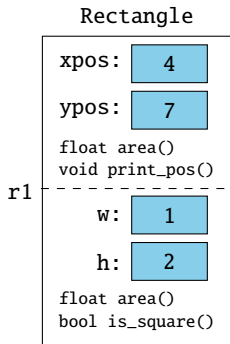
```
vector<Shape*> v {};  
  
v.push_back(new Shape{...});  
v.push_back(new Rectangle{...});  
v.push_back(new Circle{...});  
  
for (Shape* s : v)  
{  
    cout << s->area() << endl;  
}
```

Virtuell destruktör

```
Shape*      s {new Rectangle{...}};
Rectangle*  e {new Rectangle{...}};

delete s;
delete e;
```

```
class Shape
{
public:
    ...
    virtual ~Shape() = default;
    ...
};
```



Abstrakt klass

Problem:

- Ibland har en basklass ingen rimlig implementation av en funktion.

```
class Shape
{
public:
    Shape(int x, int y);

    virtual double get_area() const
    {
        // ???
    }

private:
    int x, y;
};
```

Abstrakt klass

Lösning:

- Sätt implementationen i basklassen till noll (pure virtual).
- En härledd klass måste implementera funktionen.
- Klassen är nu abstrakt. Det går inte att skapa instanser av en abstrakt klass.

```
class Shape
{
public:
    Shape(int x, int y);

    virtual double get_area() const = 0;

private:
    int x, y;
};
```

Interface

- En abstrakt klass med endast “pure virtual”-funktioner.
- Ett interface är som att ärva en designspecifikation, eftersom härledda klasser måste implementera alla “pure virtual”-funktioner.
- Samma princip som en abstrakt datatyp.

```
class Stack_Interface
{
public:
    virtual int top() const = 0;
    virtual void pop() = 0;
    virtual void push(int i) = 0;
    virtual int size() const = 0;
    virtual bool is_empty() const = 0;
};
```

using och delete

Det går att välja vilka medlemsfunktioner från basklassen som ska finnas i den härledda klassen.

- using - använd en implementation från basklassen.
- delete - ta bort en implementation.

```
class Point : public Shape
{
public:
    using Shape::Shape;

    float area() const = delete;
};
```

Exempel

- `std::ostream` är i själva verket basklassen till
 - `std::ostream`
 - `std::ofstream`
- `std::istream` är i själva verket basklassen till
 - `std::istream`
 - `std::ifstream`
- Möjliggör att återanvända samma kod för olika typer av strömmar.

Läs mer på <https://en.cppreference.com/w/cpp/io>

Övning 2

Givet på kurshemsidan är ett antal klasser. Modifiera programmet så att det fungerar som det är tänkt, och ger "Rätt" utdata.

Övning 2

Givet på kurshemsidan är ett antal klasser. Modifiera programmet så att det fungerar som det är tänkt, och ger "Rätt" utdata.

Tips: Programmet har slicing!

- 1 Polymorfi
- 2 **Typkonvertering**
- 3 UML-diagram

Explicit typkonvertering i C++

- `static_cast`
- `dynamic_cast`
- `const_cast`
- `reinterpret_cast`
- (C-style cast)

Statisk typkonvertering

```
int main()
{
    char c {'i'};
    int i { static_cast<int>(c) };

    float f { 3.14 };
    double d { static_cast<double>(f) };
}
```

Dynamisk typkontroll

När du behöver komma åt en medlemsfunktion som enbart finns i en viss härledd klass.

Funktionen är olämplig att konvertera till en virtuell funktion i basklassen.

Vi kan kontrollera om en basklasspekare i själva verket pekar på ett objekt av en härledd typ.

```
vector<Shape*> v {  
    new Rect    {1,2,3,4},  
    new Circle {1,2,5},  
    new Rect    {1,1,2,2}  
};  
  
for (Shape* s : v)  
{  
    Rect* r { dynamic_cast<Rect*>(s) };  
    if (r != nullptr)  
    {  
        cout << r->is_square() << endl;  
    }  
}
```

C-style cast

```
float f {};  
int i { (int) f };
```

Använd ej!!!!

Använd inte C-style casts!

Typkonverterande konstruktörer

En konstruktor som endast tar ett argument kan användas för typkovertering.

```
class Time
{
public:
    Time(std::string const& str);
};

void foo(Time const& t)
{
    std::cout << "En typkonvertering hände!";
}

int main()
{
    std::string str {};
    foo(str);
}
```

Typkonverterande konstruktörer

En konstruktor som endast tar ett argument kan användas för typkovertering.

```
class Time
{
public:
    explicit Time(std::string const& str);
};

void foo(Time const& t)
{
    std::cout << "En typkonvertering hände!";
}

int main()
{
    std::string str {};
    foo(str); // kompilerar ej!
}
```

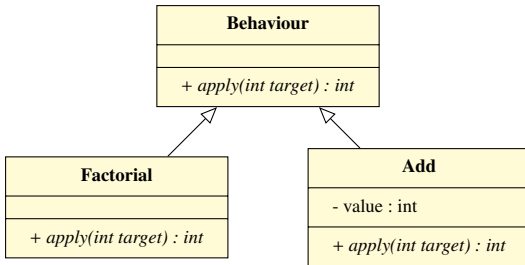
Typkonverterande operator

```
class Time
{
public:
    operator int()
    {
        return 5;
    }
};

void bar(int i)
{
    std::cout << "En annan typkonvertering hände!";
}

int main()
{
    Time t {};
    bar(t);
}
```

Övning 3



- `Behaviour::apply(int target)` har inget definierat beteende.
- `Factorial::apply(int target)` returnerar fakulteten av `target`
- `Add::apply(int target)` returnerar summan av `target` och datamedlemmen `value`

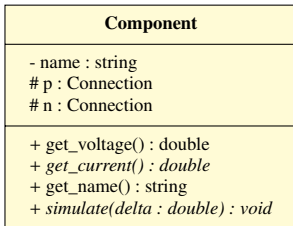
- 1 Polymorfi
- 2 Typkonvertering
- 3 UML-diagram

UML-diagram

- Visuell representation av projektets design
- Oberoende av programmeringsspråk
- Standardiserad representation
- Innehåller:
 - Klasser med alla medlemmar
 - Relationer mellan klasser

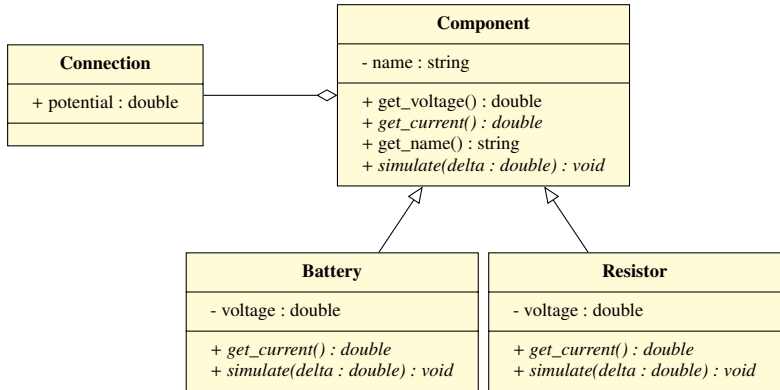
UML-diagram

Hur representeras en klass?

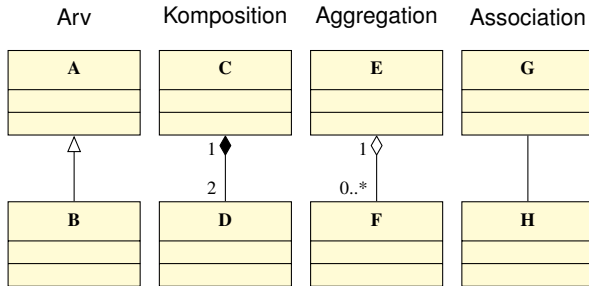


- + public
- - private
- # protected

UML-diagram



UML-diagram



“B är en A”

“C består av två D”

“E har av noll eller flera F”

“G känner till H”

UML-diagram i kod

```
class Component
{
public:
    Component() = default;

private:
    vector<Connection*> connections; // Aggregation
    std::string name;               // Komposition
};

class Battery : public Component // Arv
{
    //...
};
```

www.ida.liu.se/~TDP004/