

Repetition fra 1. semester

OOP – Lektion 1

Nogle forskelle mellem C og C++

C	C++
int eller char som boolean	bool som boolean (ny basic 8-bit datatype)
void myFunction(void)	void myFunction()
#define SIZE 5 int a[SIZE];	const int SIZE = 5; int a[SIZE];
char [] / char *	string
printf / scanf – %.3f – %8d	cout << / cin >> – setprecision(3) – setw(8)

Datatypesen string – 1

- ▶ string er *ikke* en *standard* datatype (som int, float, double, char og bool)
- ▶ string er en "brugerdefineret" datatype (baseret på en klasse) – så en string "variabel" er et objekt !!!
- ▶ Du skal derfor inkludere biblioteket `<string>` (*ikke* C biblioteket `<string.h>`)
- ▶ Til gengæld får du en helt masse faciliteter til behandling af strenge til rådighed

Datatypes string – 2

► Eksempel:

```
#include <string>

int main()
{
    string firstName = "Hans";
    string lastName, name;

    lastName = "Vildstrup";

    name = firstName + " " + lastName;

    if( firstName == "Kurt" )
    {
        .
    }
}
```

Objektorienterede sprog – 1

- ▶ Hvad kan du så i C++ som du ikke kan i C?

MEGET !!!

- ▶ Mange (men ikke alle) af de ekstra og smarte features vil du lære om i dette semester
- ▶ *Først og fremmest* kan koden struktureres MEGET bedre i et objektorienteret sprog
- ▶ Princippet hedder *indkapsling* og implementeres vha. begreberne *klasser* og *objekter*

Objektorienterede sprog – 2

- ▶ Hvad var det nu begreberne betød?
 - Indkapsling ?
 - Klasse ?
 - Objekt ?
 - Information hiding?

Objektorienterede sprog – 3

- ▶ I objektorienterede sprog kan du altså vha. klasser:
 - Definere dine *egne* datatyper (Rektangel, Motor, Sensor, Person osv. osv.).
 - Definere den funktionalitet der skal hører til din nye datatype.
 - Strukturere din kode langt bedre.
- ▶ Det skal lære du meget mere om i kurset OOP

Definition af en klasse – 1

- ▶ Klassens "skelet":

```
class Circle  
{
```

```
    // Heri defineres de variable som definerer en  
    // cirkel og den funktionalitet vi ønsker at  
    // tilknytte
```

```
};
```

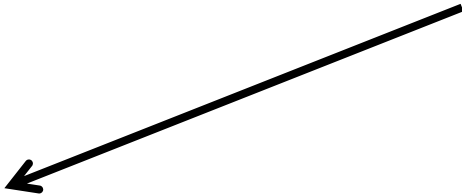

Definition af en klasse – 2

- ▶ Klassens medlemsdata/attributer

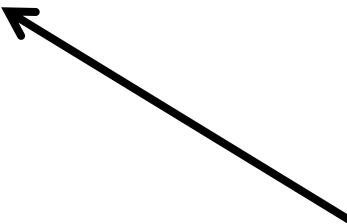
```
class Circle  
{
```

```
private:  
    double radius_;  
};
```

Disse er ALTID
private (skjulte)



Dette kaldes en
medlemsdata
eller en attribut



Definition af en klasse – 3

- ▶ Klassens medlemsfunktioner/metoder

```
class Circle
{
public:
    double getRadius( );
    void setRadius( double );

private:
    double radius_;
};
```

Disse er normalt offentlige (kan være private)

Dette kaldes medlemsfunktioner eller metoder

Definition af en klasse – 4

► Klassens constructors

```
class Circle
{
public:
    Circle();
    Circle( double );
    double getRadius( );
    void setRadius( double );
private:
    double radius_;
};
```

Dette er en
default-constructor

Dette er en
explicit-constructor

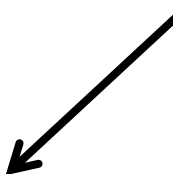
Constructorer hedder
ALTID samme som
klassen og er offentlige

Definition af en klasse – 5

► Eller:

```
class Circle
{
public:
    Circle( double = 1 );
    double getRadius( );
    void setRadius( double );
private:
    double radius_;
};
```

Dette er en
kombineret
default/explicit-
constructor



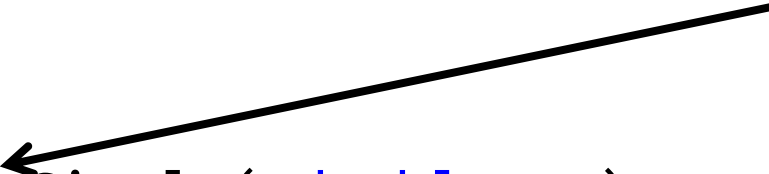
Nu er klassen
defineret

Dette kode skrives
I header-filen Circle.h

Implementering af en klasse – 1

- ▶ Dette gøres i source-filen Circle.cpp
 - Constructoren:

Dette er
vigtigt !!!



```
Circle::Circle( double r )  
{  
    if( r > 0 )  
        radius_ = r;  
    else  
        radius_ = 1;  
}
```

Implementering af en klasse – 2

De øvrige metoder:

```
double Circle::getRadius( )  
{  
    return radius_;  
}
```

Her er det igen!!!

```
void Circle::setRadius( double r )  
{  
    if( r > 0 )  
        radius_ = r;  
    else  
        radius_ = 1;  
}
```

Objekter af en klasse

- ▶ Nu er vi klar til at anvende vores klasse – dvs. instantiere objekter af typen Circle

```
int main()  
{
```

```
    int r;
```

```
    Circle myCircle1;
```

```
    Circle myCircle2( 7.3 );
```

```
    myCircle1.setRadius( 5 );
```

```
    r = myCircle2.getRadius();
```

```
    return 0;
```

```
}
```

Default objekt

Explicit objekt

Metode kald

#pragma once – 1

- ▶ I bogen bruger de følgende:

```
#ifndef
```

```
#define
```

```
.
```

```
.
```

```
.
```

```
.
```

```
.
```

```
.
```

```
.
```

```
#endif
```


#pragma once – 2

- ▶ Dette *kan* erstattes af:

#pragma once

-
-
-
-
-
-
-

En af delene skriver du **øverst** i **ALLE** header-filer !!!

Validering – 1

- ▶ Som nævnt er begrebet "information hiding" MEGET vigtigt
- ▶ Det betyder at du gør klassens medlemsdata "usynlige" udefra (ved at erklære dem private)
- ▶ Det gør du for at DU kan styre hvordan de tildeles værdier
- ▶ Mao. for at sikre at de KUN kan tildeles *gyldige* værdier
- ▶ Du **SKAL** derfor VALIDERE modtagne værdier
- ▶ Dette gælder **ALLE** metoder som tildeler værdier til klassens data (constructorer, set-metoder)

Validering – 2

► Eksempel:

```
void Circle::setRadius( double r )
{
    if( r > 0 )
        radius_ = r;
    else          // der SKAL være en else
        radius_ = 1;
}
```

◦ Alternativ:

```
void Circle::setRadius( double r )
{
    radius_ = ( r > 0 ? r : 1 ) ;
}
```

Validering – 3

► Eksempel:

```
Circle::Circle( double rad )  
{  
    if( rad > 0 )  
        radius_ = rad;  
    else  
        radius_ = 1;  
}
```

◦ Alternativ:

```
Circle::Circle( double rad )  
{  
    setRadius( rad );  
}
```

Forskellige typer metoder – 1

▶ Constructors

- Metoder som kun kaldes når et objekt erklæres – constructors kaldes automatisk.
- Vi har set på default- og explicit-constructors

▶ Destructors

- Disse kaldes ligeledes automatisk når et objekt nedlægges – hører I nærmere om i dette kursus

▶ Mutators

- Metoder som modificerer/ændrer på værdierne af medlemsdata
- Eksempel: alle set-metoder

Forskellige typer metoder – 2

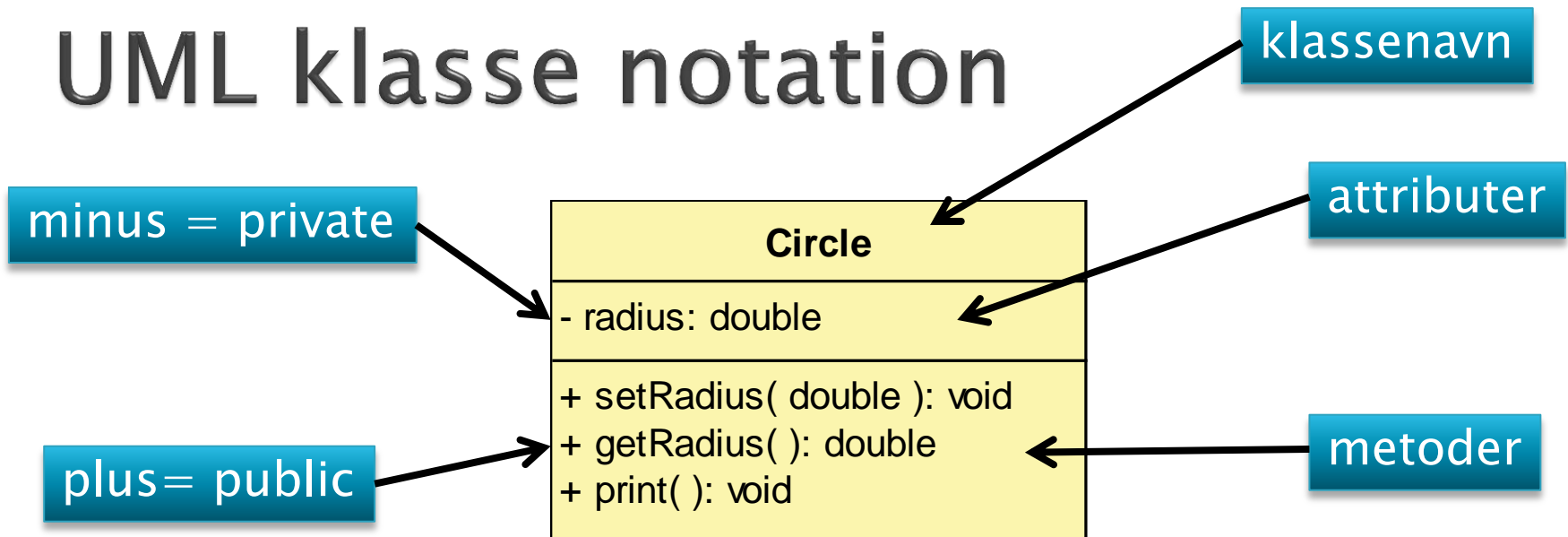
▶ Accessors

- Metoder som tilgår/læser værdier af medlemsdata
- Eksempel: alle get-metoder og print-metoder

▶ Utilities

- Hjælpe-metoder til andre metoder
- Disse er normalt private, da de kun bruges af de andre metoder

UML klasse notation

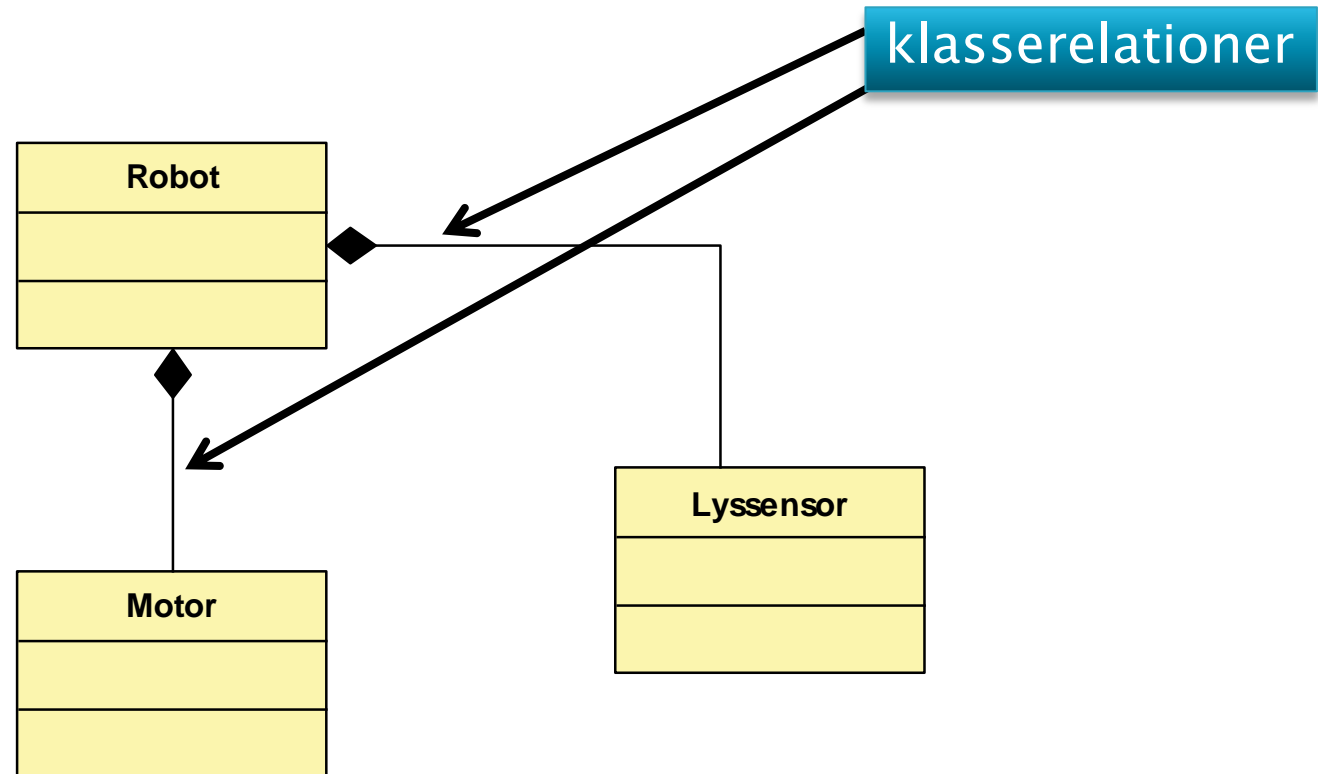


- ▶ **Bemærk:** constructorer vises normalt *ikke* i klassenotationen – fordi *alle* klasser har *minimum* en constructor
- ▶ **Men:** constructorer *skal* beskrives i den efterfølgende klassebeskrivelse

Eksempel

Time
- hour: int - minute: int - second: int
+ setTime(int, int, int): void + print(): void

UML klasse diagram



- ▶ De her viste relationer kaldes *komposition*

Klasserelationer

- ▶ Der findes 4 typer klasserelationer
 - **Komposition**
 - En "*har* en/et"-relation med ejerskab
 - Eksempel: En Bil *har* en Motor
 - Kender du fra 1. semester
 - **Aggregering**
 - En "*har* en/et"-relation uden ejerskab
 - Eksempel: Et Kursus *har* en Studerende
 - **Association**
 - En "anvender/bruger/aflæser/osv."-relation
 - Eksempel: En Sensor *skriver til* en Log
 - **Arv**
 - En "*er* en/et"-relation
 - Eksempel: En Sportsvogn *er* en Bil

Komposition – 1

- ▶ Komposition er en "har en/et"–relation
 - En Bil *har* en Motor
 - En Cirkel *har* et Punkt
 - Et Kontrolpanel *har* en Knap (eller flere)
 - En Lejlighed *har* et Værelse (eller flere)
- ▶ Man kan også sige "består af"
 - En Lejlighed *består af* Værelser
 - Et Værelse er *en del* af en Lejlighed
- ▶ Der er tale om "ejerskab"
 - En Lejlighed *ej*er Værelset (værelset kan ikke være en del af mere end en lejlighed)

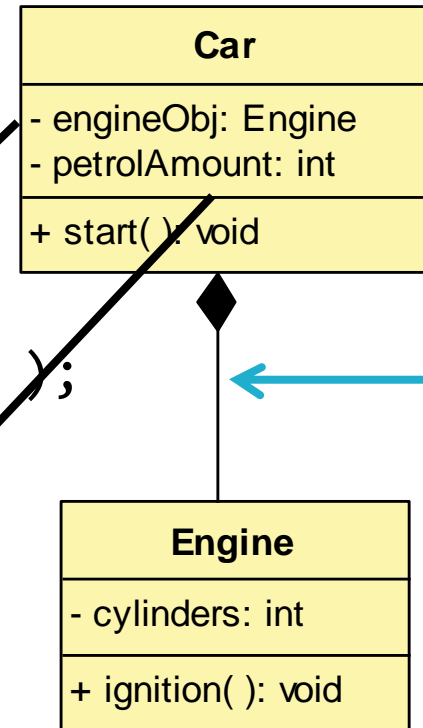
Komposition – 2

- ▶ Levertiderne for objekter i en komposition er ens
 - Hvis man nedlægger en Lejlighed nedlægger man også de Værelser som lejligheden består af.
- ▶ Komposition er derfor den stærkeste relationstype
- ▶ Komposition implementeres som "forventet"
 - Klassen Bil *har* et Motor objekt som privat attribut
 - Klassen Lejlighed *har* et (eller flere) Værelse objekter som privat attribut(er)

Komposition – 3

► Eksempel:

```
class Car
{
public:
    Car( int pa=0, int cyl=0 );
    void start();
private:
    Engine engineObj_;
    int petrolAmount_;
};
```

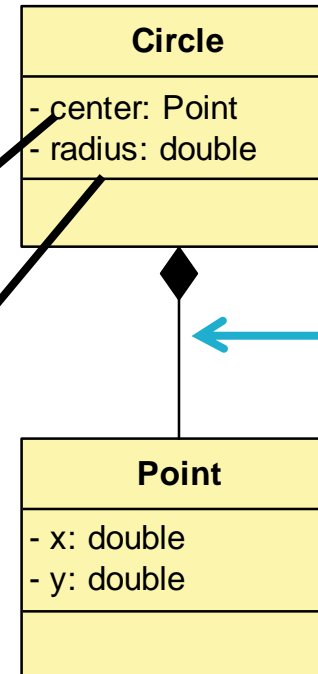


komposition

Komposition – 4

► Eksempel:

```
class Circle
{
public:
    .
    .
    .
private:
    Point center_;
    double radius_;
};
```

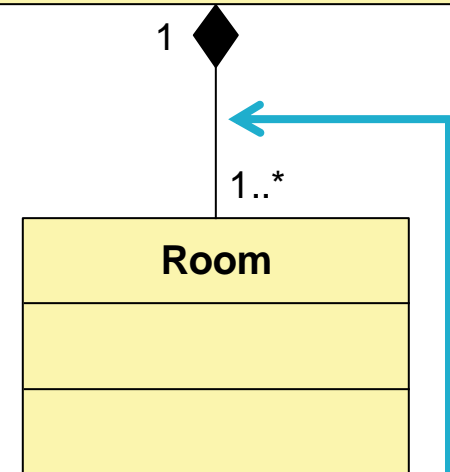
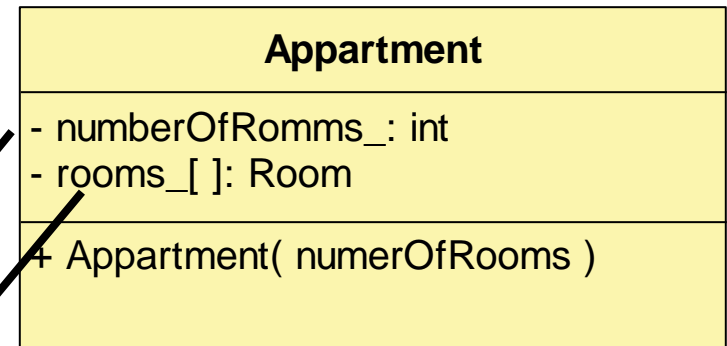


komposition

Komposition – 5

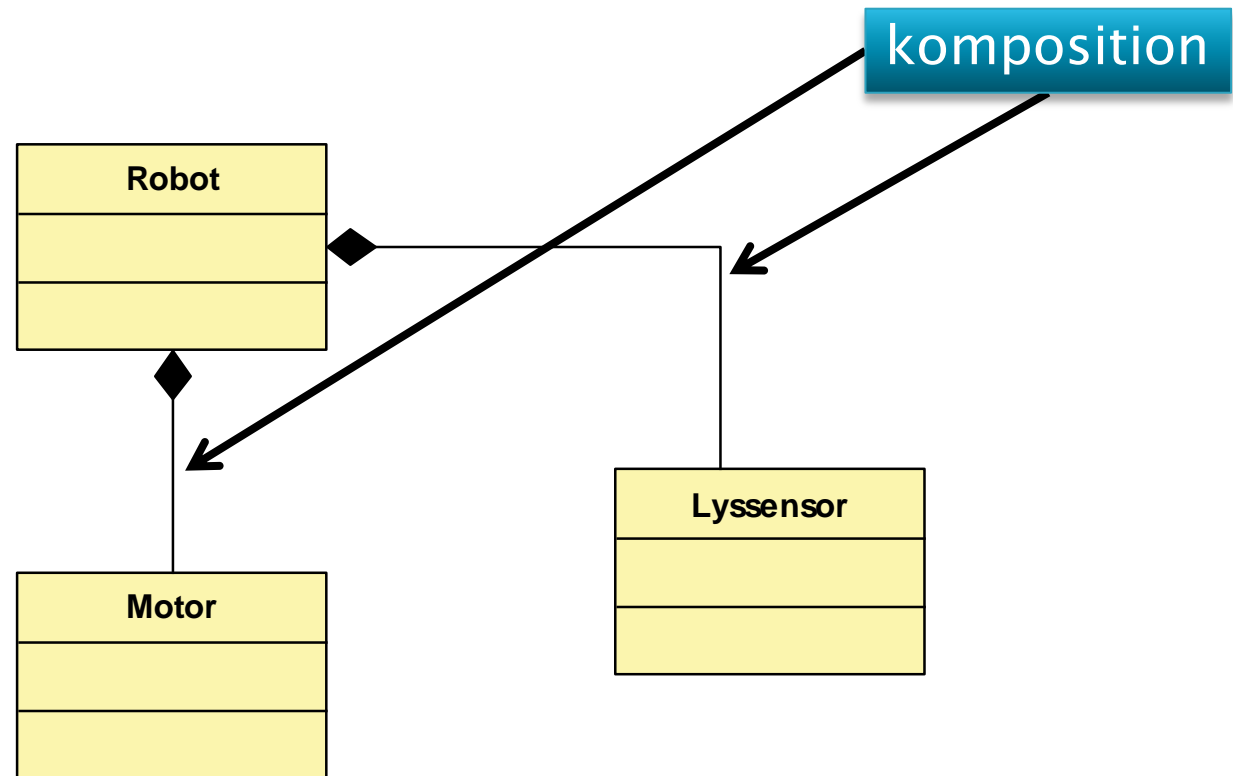
► Eksempel:

```
class Appartment
{
public:
    Appartment(int nbOfRooms);
    .
    .
private:
    int numberOfRooms_;
    Room rooms_[numberOfRooms_];
};
```



komposition

Komposition – 6



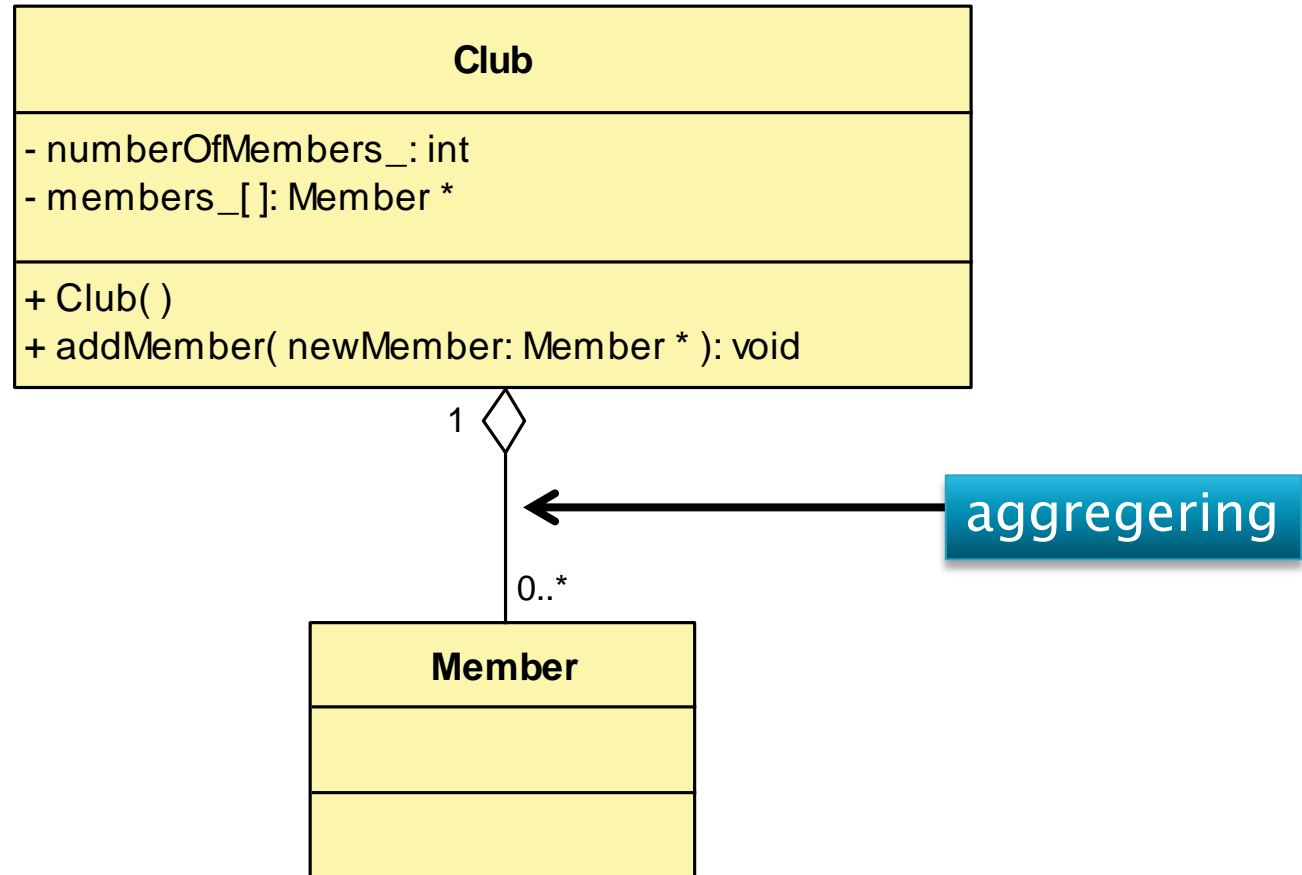
Aggregering– 1

- ▶ Aggregering er også en "har en/et"–relation
 - En Klub *har* et Medlem (eller flere)
- ▶ Man kan også sige "består af"
 - En Klub *består af* Medlemmer
 - Et Medlem er *en del* af en Klub
- ▶ Men...der er *ikke* tale om "ejerskab"
 - Et Medlem kan godt være medlem af mere end en Klub
- ▶ Levertiderne for objekter i en aggregering er *forskellige*
 - Hvis man nedlægger en Klub lever et Medlem jo videre – enten som "fri" eller som Medlem af en anden Klub.

Aggregering– 2

- ▶ Aggregering er derfor en svagere relation end komposition
- ▶ Komposition implementeres via pointere
 - Klassen Klub *har* en (eller flere) pointere (array af pointere) til Medlem objekter som privat attribut
 - Klassen Lejlighed *har* et (eller flere) Værelse objekter som privat attribut(er)

Aggregering- 3



Association og arv

- ▶ Dem lærer du om senere i dette kursus 😊