Programmering og Problemløsning

14.1: Nedarvning

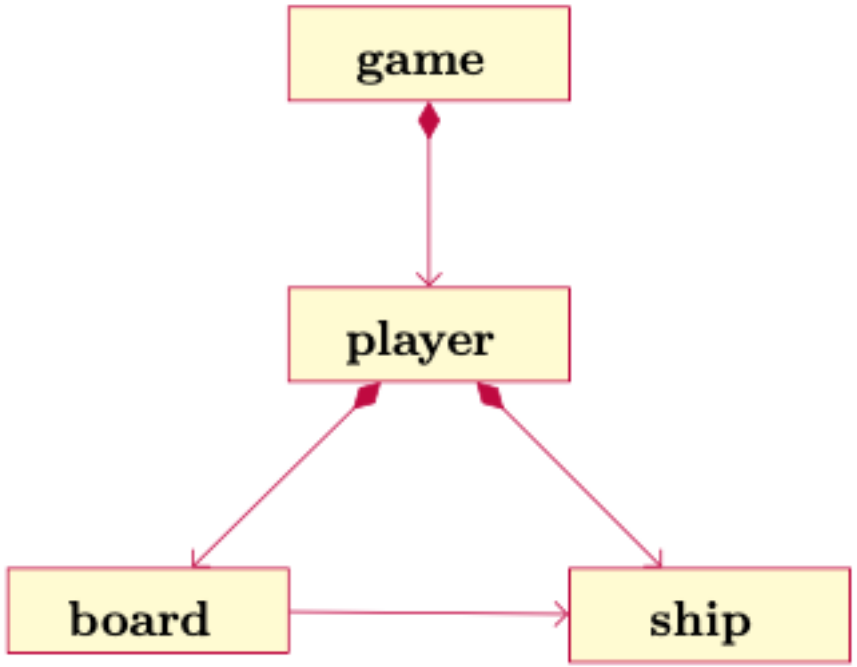
Design efter navne- og udsagnsord

Use-case: Sænke slagskibe

Dette er et spil for to personer, der kan spilles med papir og blyant. Der spilles på fire plader, to for hver spiller, og hver plade er inddelt i 10x10 felter. Hvert felt identificeres vha. dets række- og søjlenummer.

Hver spiller får tildelt et antal skibe, som placeres på spillerens ene plade og markerer, hvor modstanderen har forsøgt at skyde. På den anden plade markerer spilleren tilsvarende, hvor han/hun har forsøgt at ramme modstanderen.

Når skibene er placeret skiftes spillerne til at skyde på modstanderens felt, og modstanderen annoncerer ramt eller plask, alt efter om et skib blev ramt eller ej. Vinderen er den, der først får sænket alle modstanderes skibe.



`battleship.fsx`

Overload

Navne kan genbruges, blot skal parameter antallet og/eller typer være forskelligt.

Overload

Navne kan genbruges, blot skal parameter antallet og/eller typer være forskelligt.

```
type person (name : string) =
  member this.name = name
  member this.greetings () = name+" says hi"
  member this.greetings (str : string) =
    name+" "+str

let p = person ("Jon")
printfn "%s" (p.greetings ())
printfn "%s" (p.greetings "says goodbye")
```

Overload

Navne kan genbruges, blot skal parameter antallet og/eller typer være forskelligt.

```
type person (name : string) =
  member this.name = name
  member this.greetings () = name+" says hi"
  member this.greetings (str : string) =
    name+" "+str

let p = person ("Jon")
printfn "%s" (p.greetings ())
printfn "%s" (p.greetings "says goodbye")
```

```
sporring@Jons-mac src % fsharpi overload.fsx
Jon says hi
Jon says goodbye
```

Overload

Navne kan genbruges, blot skal parameter antallet og/eller typer være forskelligt.

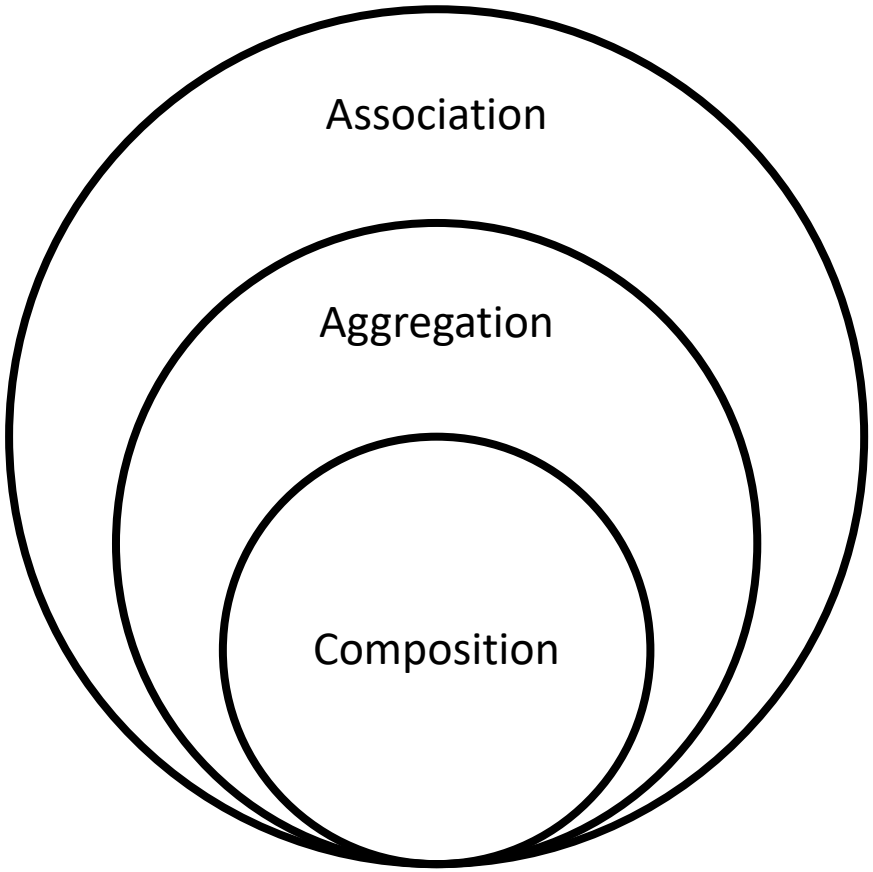
```
type person (name : string) =
  member this.name = name
  member this.greetings () = name+" says hi"
  member this.greetings (str : string) =
    name+" "+str

let p = person ("Jon")
printfn "%s" (p.greetings ())
printfn "%s" (p.greetings "says goodbye")
```

Fordele: Metodenavne kan være mere generelle

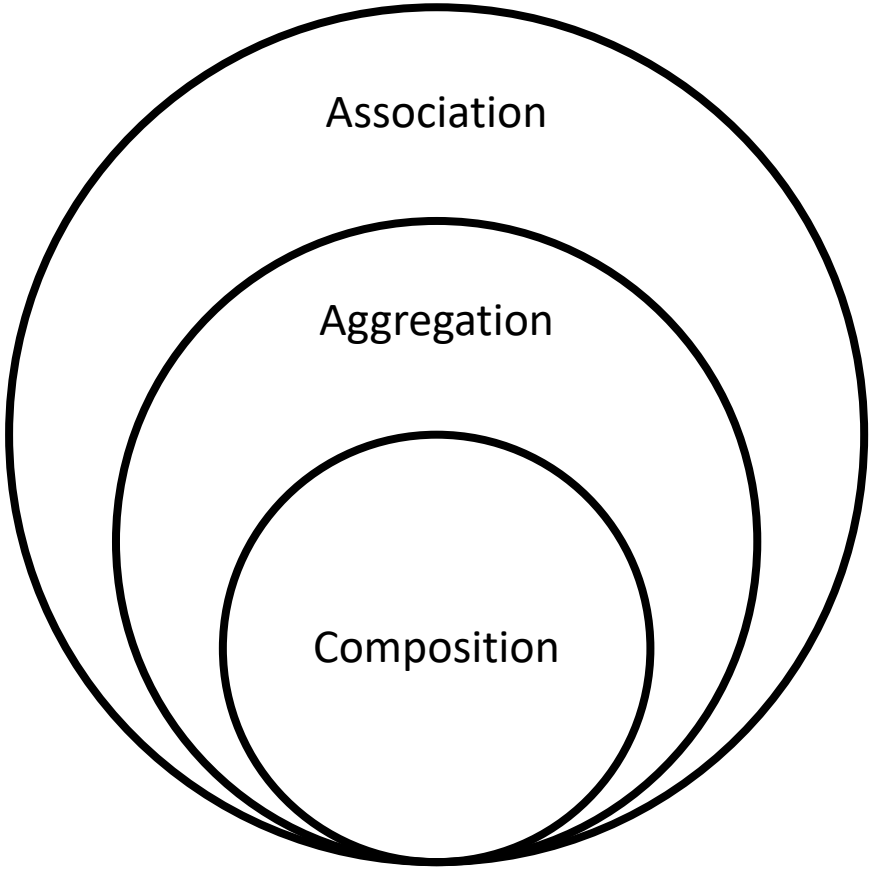
```
sporring@Jons-mac src % fsharpi overload.fsx
Jon says hi
Jon says goodbye
```

Relationer mellem objekter (has-a)

Composition: En hund har 4 ben



Relationer mellem objekter (has-a)

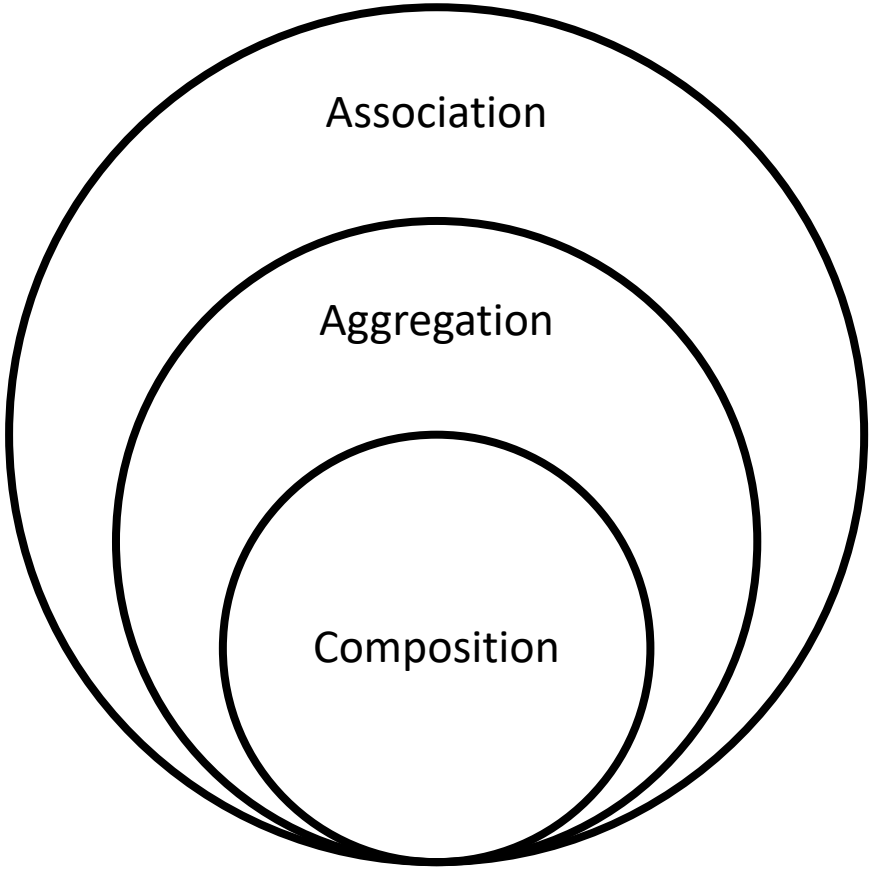


Composition: En hund har 4 ben

Listing 22.3 umlComposition.fsx:
The dog object is a composition of four leg objects.

```
1  type leg () =
2    member this.move = "moving"
3  type dog () =
4    let _leg = List.init 4 (fun e -> leg ())
5
6  let bestFriend = dog ()
```

Relationer mellem objekter (has-a)

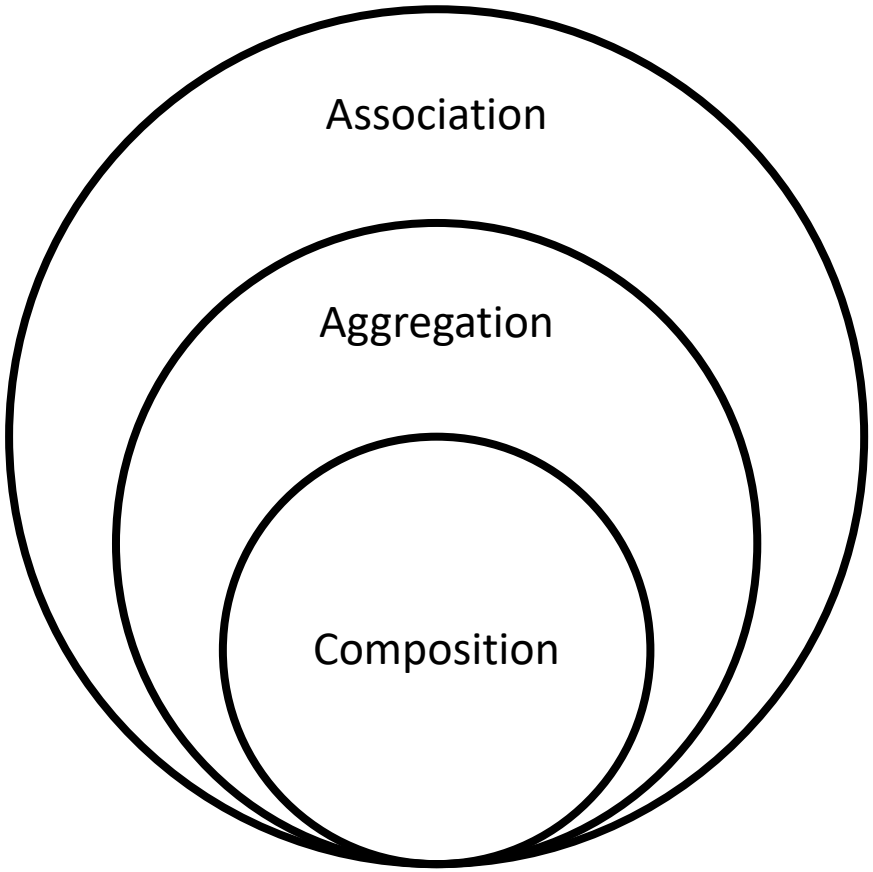


Composition: En hund har 4 ben

Listing 22.3 umlComposition.fsx:
The dog object is a composition of four leg objects.

```
1 type leg () =
2   member this.move = "moving"
3 type dog () =
4   let _leg = List.init 4 (fun e -> leg ())
5
6 let bestFriend = dog ()
```

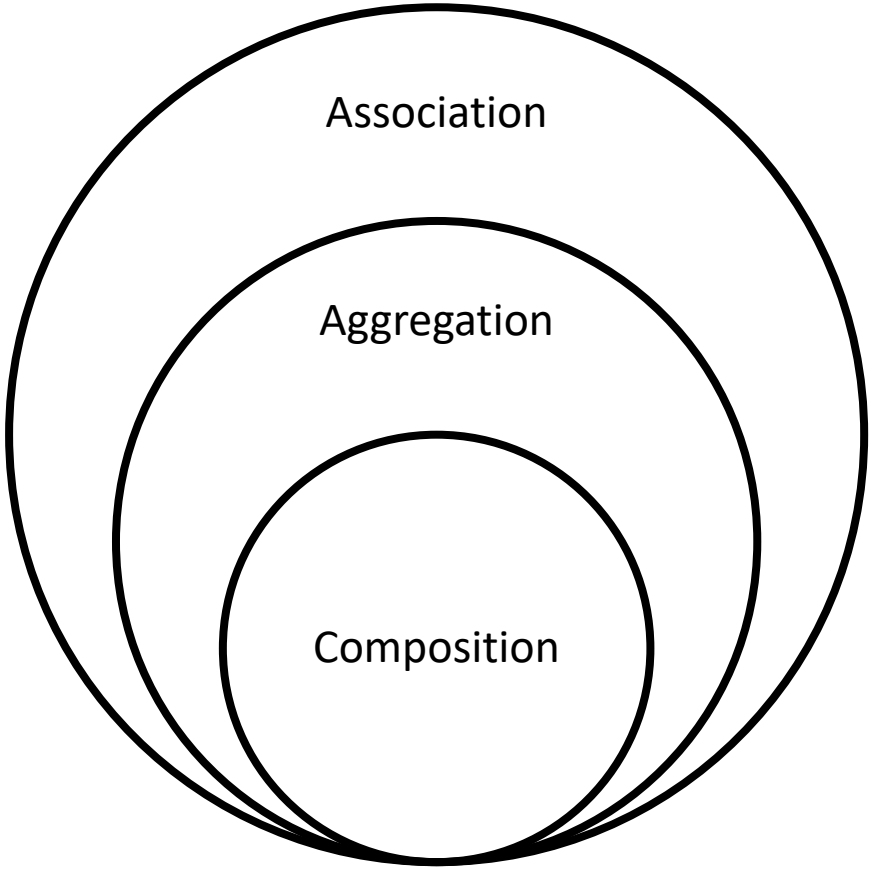
Relationer mellem objekter (has-a)



Composition: En hund har 4 ben

Aggregation: En forfatter skriver og udgiver en bog, en læser køber bogen

Relationer mellem objekter (has-a)



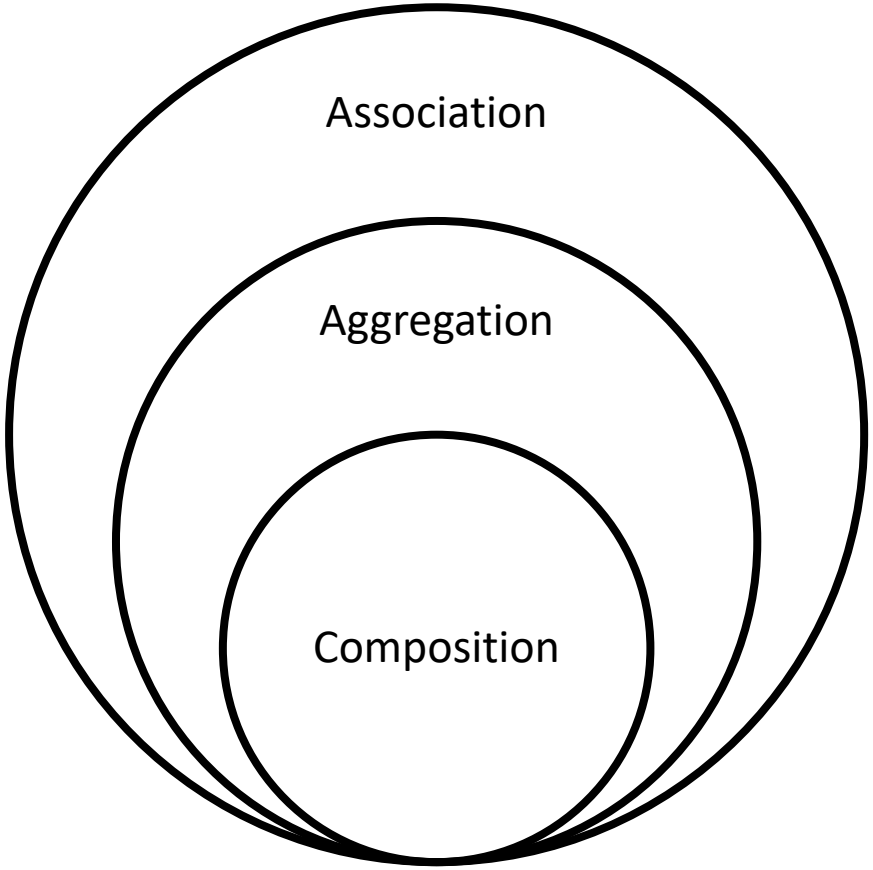
Composition: En hund har 4 ben

Aggregation: En forfatter skriver og udgiver en bog, en læser køber bogen

Listing 22.2 umlAggregation.fsx:
The book has an aggregated relation to author and reader.

```
1  type book (name : string) =
2    let mutable _name = name
3  type author () =
4    let _book = book("Learning to program")
5    member this.publish() = _book
6  type reader () =
7    let mutable _book : book option = None
8    member this.buy (b : book) = _book <- Some b
9
10 let a = author ()
11 let r = reader ()
12 let b = a.publish ()
13 r.buy (b)
```

Relationer mellem objekter (has-a)

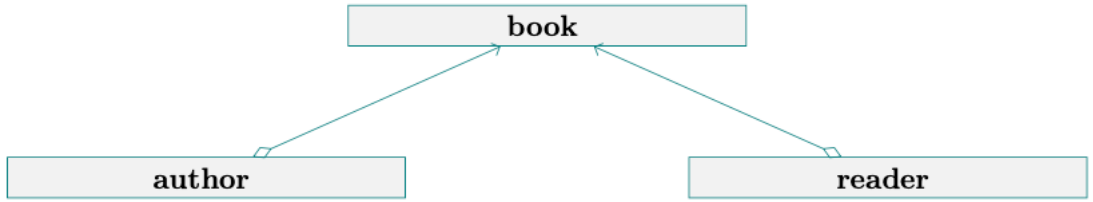


Composition: En hund har 4 ben

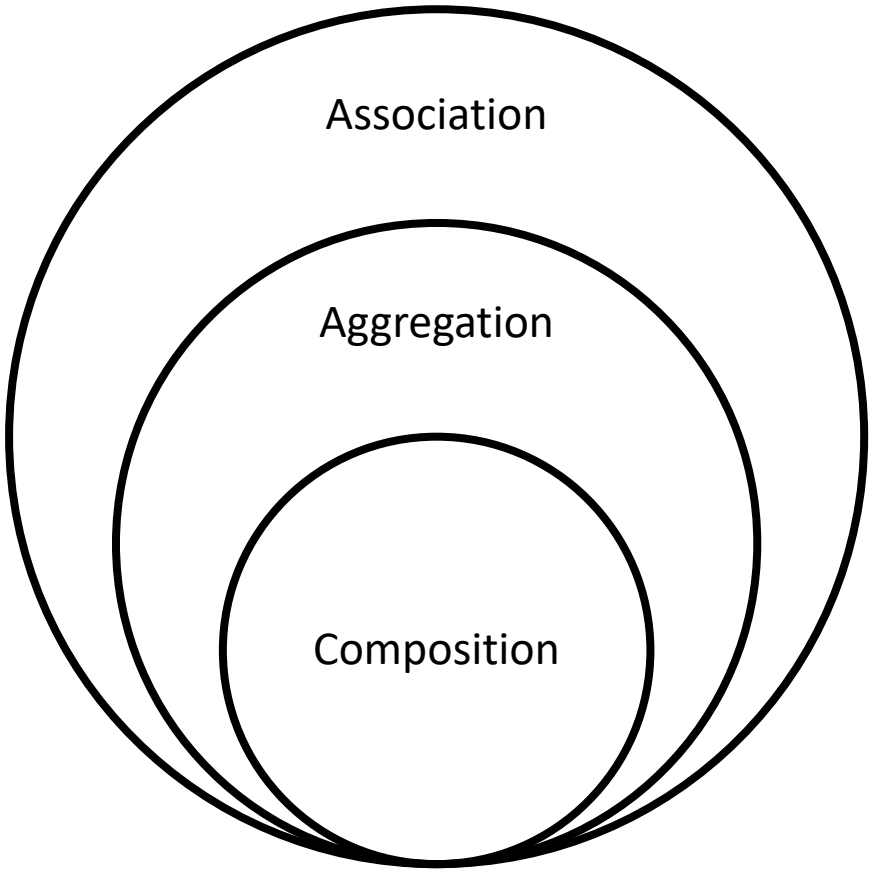
Aggregation: En forfatter skriver og udgiver en bog, en læser køber bogen

Listing 22.2 umlAggregation.fsx:
The book has an aggregated relation to author and reader.

```
1  type book (name : string) =
2    let mutable _name = name
3  type author () =
4    let _book = book("Learning to program")
5    member this.publish() = _book
6  type reader () =
7    let mutable _book : book option = None
8    member this.buy (b : book) = _book <- Some b
9
10 let a = author ()
11 let r = reader ()
12 let b = a.publish ()
13 r.buy (b)
```



Relationer mellem objekter (has-a)

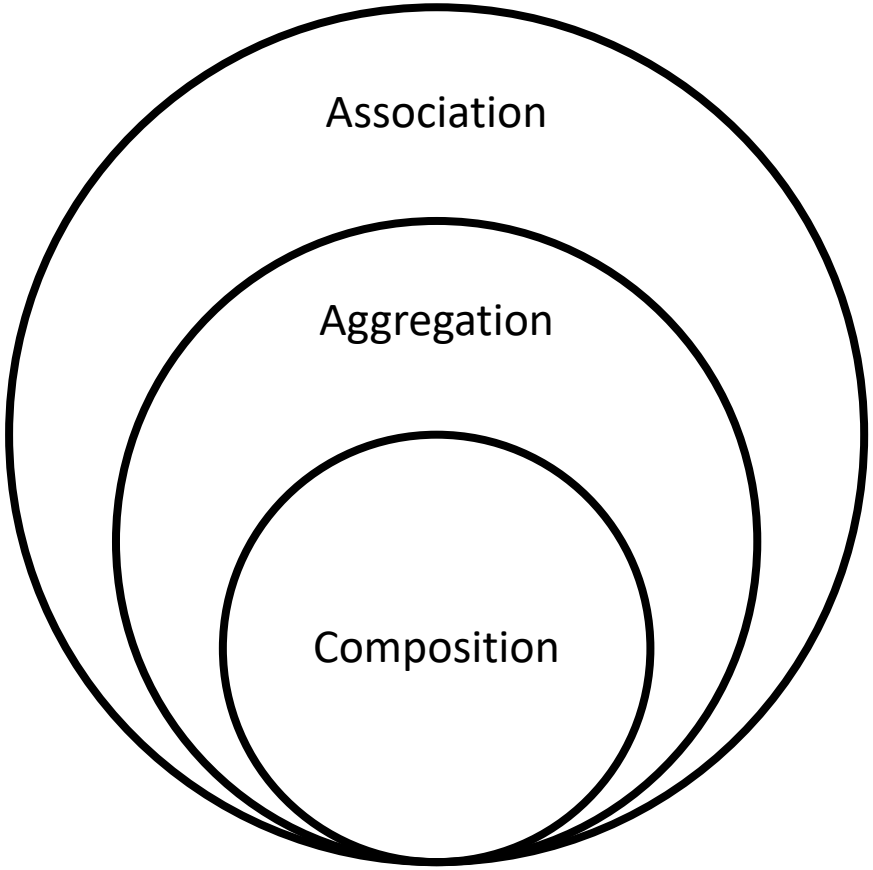


Composition: En hund har 4 ben

Aggregation: En forfatter skriver og udgiver en bog, en læser køber bogen

Association: En studerende kan stille en lærer et spørgsmål

Relationer mellem objekter (has-a)



Composition: En hund har 4 ben

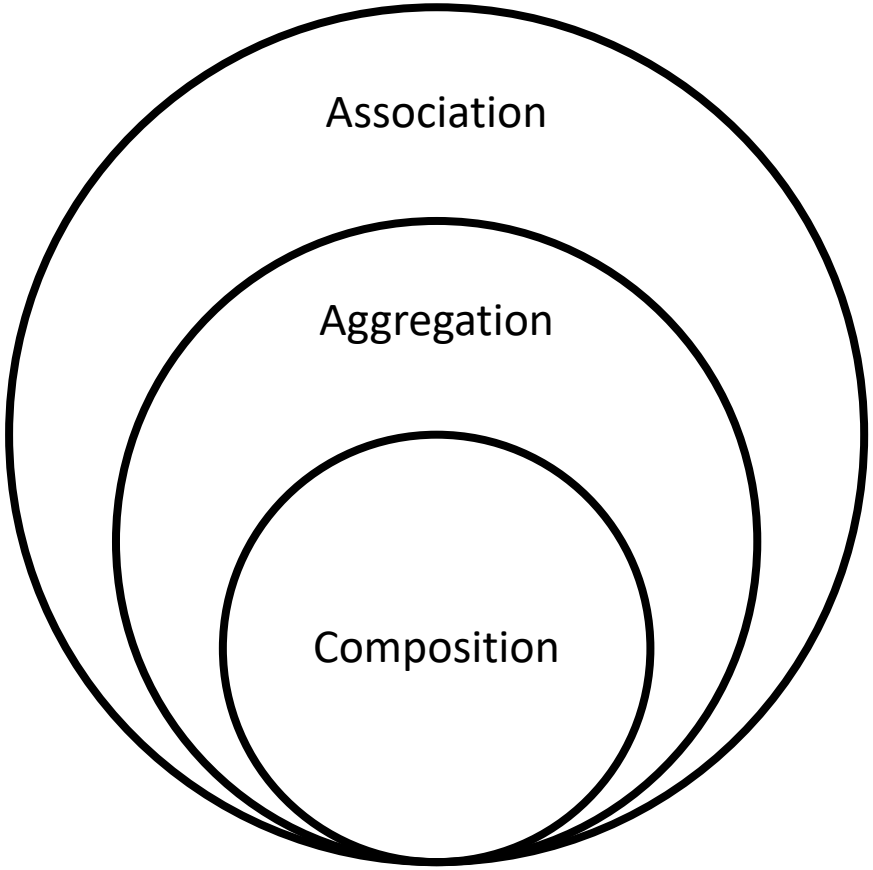
Aggregation: En forfatter skriver og udgiver en bog, en læser køber bogen

Association: En studerende kan stille en lærer et spørgsmål

Listing 22.1 umlAssociation.fsx:
The student is associated with a teacher.

```
1  type teacher () =
2    member this.answer (q : string) = "4"
3  type student (t : teacher) =
4    member this.ask () = t.answer("What is 2+2?")
5
6  let t = teacher ()
7  let s = student (t)
8  s.ask()
```

Relationer mellem objekter (has-a)



Composition: En hund har 4 ben

Aggregation: En forfatter skriver og udgiver en bog, en læser køber bogen

Association: En studerende kan stille en lærer et spørgsmål

Listing 22.1 umlAssociation.fsx:
The student is associated with a teacher.

```
1 type teacher () =
2   member this.answer (q : string) = "4"
3 type student (t : teacher) =
4   member this.ask () = t.answer("What is 2+2?")
5
6 let t = teacher ()
7 let s = student (t)
8 s.ask()
```

Nedarvning (is-a)

En studerende og en lærer har begge et navn. En studerende har en bog, og en lærer har et sæt af powerpoint slides.

Listing 22.4 umlInheritance.fsx:
The student and the teacher class inherits from the person class.

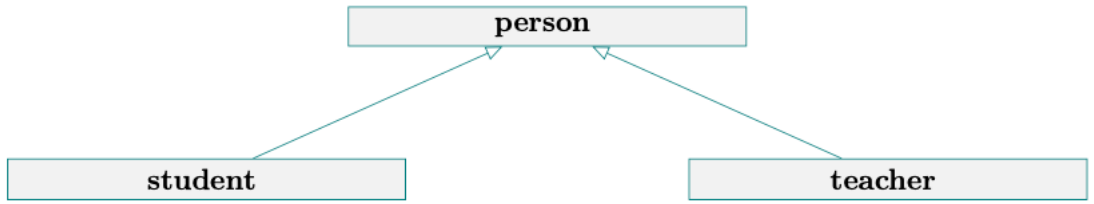
```
1  type person (name : string) =
2    member this.name = name
3  type student (name : string, book : string) =
4    inherit person(name)
5    member this.book = book
6  type teacher (name : string, slides : string) =
7    inherit person(name)
8    member this.slides = slides
9
10 let s = student("Hans", "Learning to Program")
11 let t = teacher("Jon", "Slides of the day")
```

Nedarvning (is-a)

En studerende og en lærer har begge et navn. En studerende har en bog, og en lærer har et sæt af powerpoint slides.

Listing 22.4 umlInheritance.fsx:
The student and the teacher class inherits from the person class.

```
1  type person (name : string) =
2    member this.name = name
3  type student (name : string, book : string) =
4    inherit person(name)
5    member this.book = book
6  type teacher (name : string, slides : string) =
7    inherit person(name)
8    member this.slides = slides
9
10 let s = student("Hans", "Learning to Program")
11 let t = teacher("Jon", "Slides of the day")
```

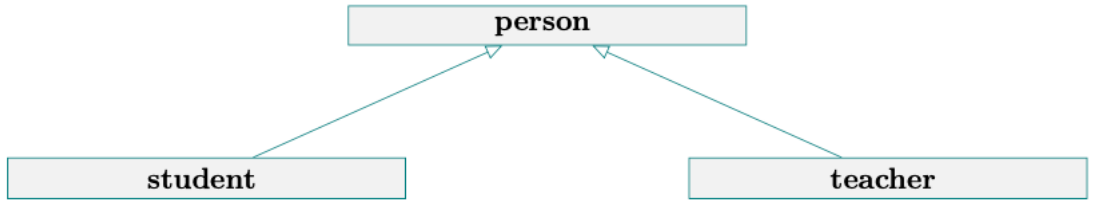


Nedarvning (is-a)

En studerende og en lærer har begge et navn. En studerende har en bog, og en lærer har et sæt af powerpoint slides.

Listing 22.4 umlInheritance.fsx:
The student and the teacher class inherits from the person class.

```
1  type person (name : string) =
2    member this.name = name
3  type student (name : string, book : string) =
4    inherit person(name)
5    member this.book = book
6  type teacher (name : string, slides : string) =
7    inherit person(name)
8    member this.slides = slides
9
10 let s = student("Hans", "Learning to Program")
11 let t = teacher("Jon", "Slides of the day")
```



Fordele: Kodegenbrug, semantisk hierarki
Bagdele: Risiko for spaghettikode

Overshadow, upcasting og downcasting

Genbrug af navne i underklasser overskygger baseklassens navne. Downcasting og upcasting navigerer hiearkiet.

Overshadow, upcasting og downcasting

Genbrug af navne i underklasser overskygger baseklassens navne. Downcasting og upcasting navigerer hiearkiet.

```
type person (name : string) =
  member this.name = name
  member this.introduction = "I'm " + name
type teacher (name : string) =
  inherit person(name)
  member this.introduction = "I'm Prof. " + name

let p = person ("Hans")
printfn "%s" p.introduction
let t = teacher ("Jon")
printfn "%s" t.introduction
let tp = t :> person
printfn "%s" tp.introduction
let tpt = tp :?> teacher
printfn "%s" tpt.introduction
```

Overshadow, upcasting og downcasting

Genbrug af navne i underklasser overskygger baseklassens navne. Downcasting og upcasting navigerer hiearkiet.

```
type person (name : string) =
  member this.name = name
  member this.introduction = "I'm " + name
type teacher (name : string) =
  inherit person(name)
  member this.introduction = "I'm Prof. " + name

let p = person ("Hans")
printfn "%s" p.introduction
let t = teacher ("Jon")
printfn "%s" t.introduction
let tp = t :> person
printfn "%s" tp.introduction
let tpt = tp :?> teacher
printfn "%s" tpt.introduction
```

```
sporring@Jons-mac src % fsharpi overshadow.fsx
I'm Hans
I'm Prof. Jon
I'm Jon
I'm Prof. Jon
```

Overshadow, upcasting og downcasting

Genbrug af navne i underklasser overskygger baseklassens navne. Downcasting og upcasting navigerer hiearkiet.

```
type person (name : string) =
  member this.name = name
  member this.introduction = "I'm " + name
type teacher (name : string) =
  inherit person(name)
  member this.introduction = "I'm Prof. " + name

let p = person ("Hans")
printfn "%s" p.introduction
let t = teacher ("Jon")
printfn "%s" t.introduction
let tp = t :> person
printfn "%s" tp.introduction
let tpt = tp :?> teacher
printfn "%s" tpt.introduction
```

```
sporring@Jons-mac src % fsharpi overshadow.fsx
I'm Hans
I'm Prof. Jon
I'm Jon
I'm Prof. Jon
```

Fordele: ‘reparation’, hierarkiske definitioner, underklasser i samme liste
Bagdele: downcasting kan give run-time fejl

Override, upcasting og downcasting

Genbrug af navne i underklasser overskygger baseklassens navne. Downcasting og upcasting navigerer hiearkiet.

Override, upcasting og downcasting

Genbrug af navne i underklasser overskygger baseklassens navne. Downcasting og upcasting navigerer hiearkiet.

```
[<AbstractClass>]
type person (name : string) =
  member this.name = name
  abstract member introduction : string
type teacher (name : string) =
  inherit person(name)
  override this.introduction = "I'm Prof. " + name

let t = teacher ("Jon")
printfn "%s" t.introduction
let tp = t :> person
printfn "%s" tp.introduction
let tpt = tp :?> teacher
printfn "%s" tpt.introduction
```

Override, upcasting og downcasting

Genbrug af navne i underklasser overskygger baseklassens navne. Downcasting og upcasting navigerer hiearkiet.

```
[<AbstractClass>]
type person (name : string) =
  member this.name = name
  abstract member introduction : string
type teacher (name : string) =
  inherit person(name)
  override this.introduction = "I'm Prof. " + name

let t = teacher ("Jon")
printfn "%s" t.introduction
let tp = t :> person
printfn "%s" tp.introduction
let tpt = tp :?> teacher
printfn "%s" tpt.introduction
```

```
sporring@Jons-mac src % fsharpi override.fsx
I'm Prof. Jon
I'm Prof. Jon
I'm Prof. Jon
```

Override, upcasting og downcasting

Genbrug af navne i underklasser overskygger baseklassens navne. Downcasting og upcasting navigerer hiearkiet.

```
[<AbstractClass>]
type person (name : string) =
  member this.name = name
  abstract member introduction : string
type teacher (name : string) =
  inherit person(name)
  override this.introduction = "I'm Prof. " + name

let t = teacher ("Jon")
printfn "%s" t.introduction
let tp = t :> person
printfn "%s" tp.introduction
let tpt = tp :?> teacher
printfn "%s" tpt.introduction
```

Fordele: kan stille krav til underklasser
Bagdele: baseklassen kan ikke instantieres

```
sporring@Jons-mac src % fsharpi override.fsx
I'm Prof. Jon
I'm Prof. Jon
I'm Prof. Jon
```

Interface (is-a)

Et fjernsyn og en bil har begge en knap. Man kan trykke på knappen, og resultatet er ganske forskelligt.

Interface (is-a)

Et fjernsyn og en bil har begge en knap. Man kan trykke på knappen, og resultatet er ganske forskelligt.

Listing 22.5 umlInterface.fsx:
The `television` and the `car` class both implement the `button` interface.

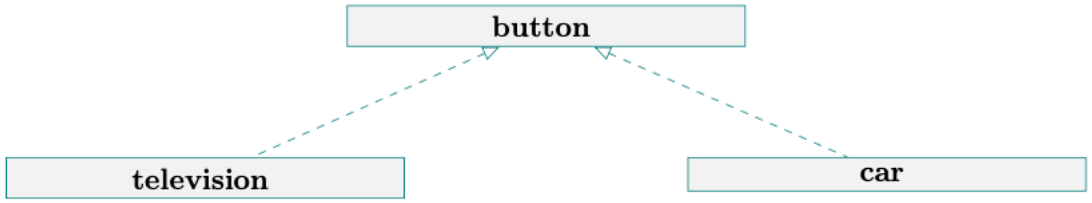
```
1  type button =
2    abstract member press : unit -> string
3  type television () =
4    interface button with
5      member this.press () = "Changing channel"
6  type car () =
7    interface button with
8      member this.press () = "Activating wipers"
9  let pressIt (elm : #button) =
10   elm.press()
11
12 let t = television()
13 let c = car()
14 printfn "%s" (pressIt t)
15 printfn "%s" (pressIt c)
```

Interface (is-a)

Et fjernsyn og en bil har begge en knap. Man kan trykke på knappen, og resultatet er ganske forskelligt.

Listing 22.5 umlInterface.fsx:
The `television` and the `car` class both implement the `button` interface.

```
type button =
  abstract member press : unit -> string
type television () =
  interface button with
    member this.press () = "Changing channel"
type car () =
  interface button with
    member this.press () = "Activating wipers"
let pressIt (elm : #button) =
  elm.press()

let t = television()
let c = car()
printfn "%s" (pressIt t)
printfn "%s" (pressIt c)
```

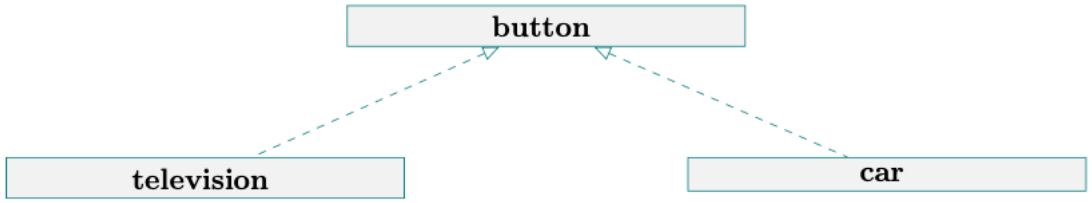


Interface (is-a)

Et fjernsyn og en bil har begge en knap. Man kan trykke på knappen, og resultatet er ganske forskelligt.

Listing 22.5 umlInterface.fsx:
The `television` and the `car` class both implement the `button` interface.

```
type button =
  abstract member press : unit -> string
type television () =
  interface button with
    member this.press () = "Changing channel"
type car () =
  interface button with
    member this.press () = "Activating wipers"
let pressIt (elm : #button) =
  elm.press()

let t = television()
let c = car()
printfn "%s" (pressIt t)
printfn "%s" (pressIt c)
```



Fordele: Angiver egenskaber, semantisk graf
Bagdele: Risiko for megen up- og downcasting

Opsummering

- Med overloading kan vi genbruge navne til små variationer i inputparametre
- Association: “kender-til” - besked relation
- Aggregation: “har-en/flere” – udveksling af ejeskab
- Composition: “har-en/flere” – een ejer
- Overshadow: Navnesammenfald i nedarvning skygger i underklassen
- Abstrakte klasser og override: Abstrakte klasser kan kræve nedarvning og metodedefinitioner.
- Interfaces: Interfaces giver klasser egenskaber, som kan bruges på tværs af det semantiske design.