Mere om Nedarvning
(Virtual dispatching*)

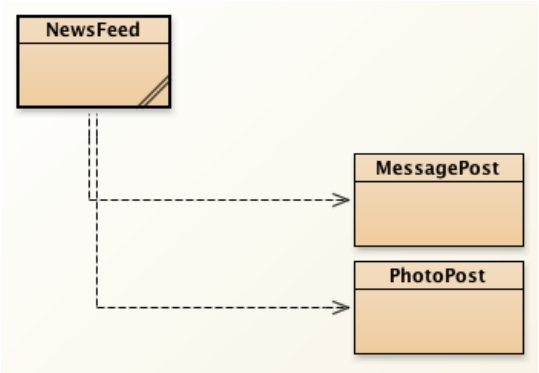
GRPRO: "Grundlæggende Programmering"

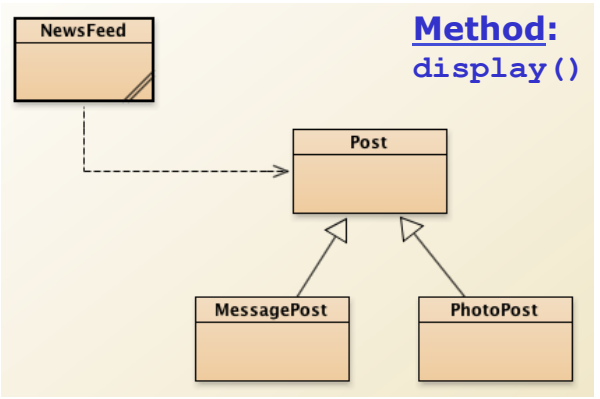
*) aka, *"dynamic dispatching"*
*) aka, *"polymorphism"* (which is something else) **:-(**

A G E N D A

- **Recap**
- **Virtual dispatching!**
- **Statisk type** -vs- **dynamic type**
- **Visibility modifiers** (public, private, ...)

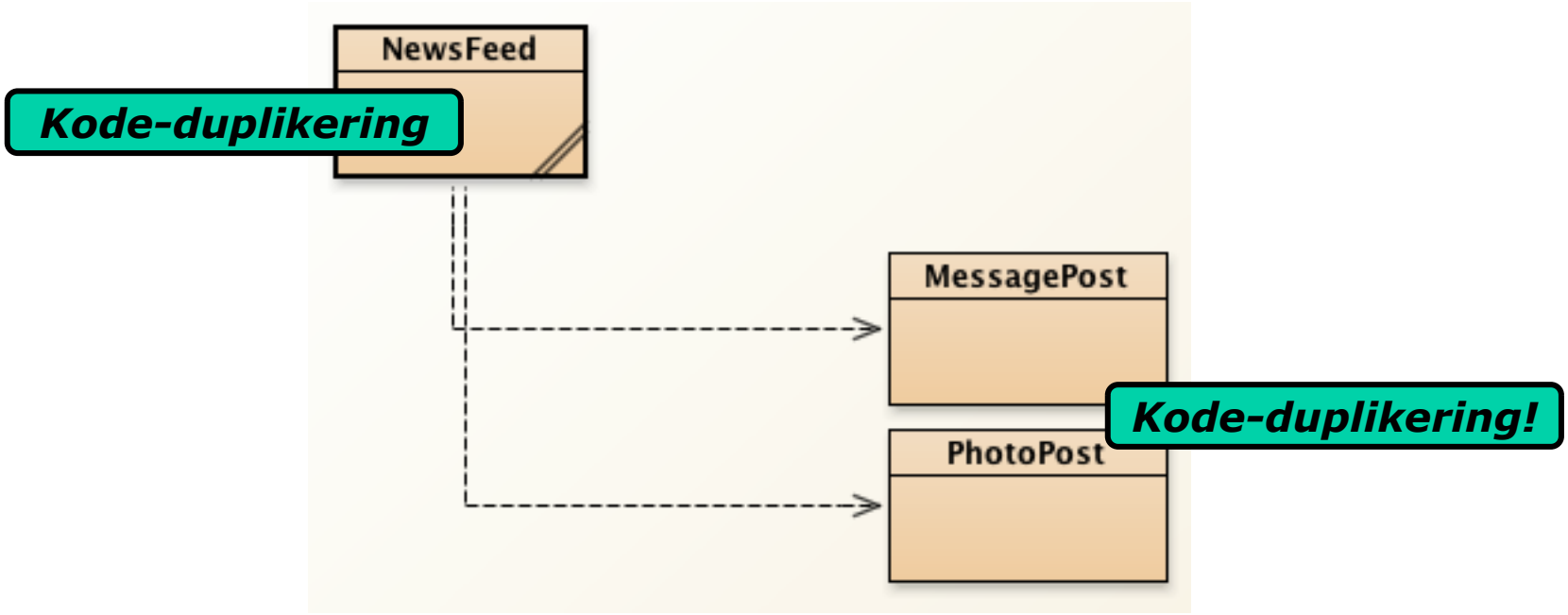
- Example: **`NewsFeed`** (à la Facebook):



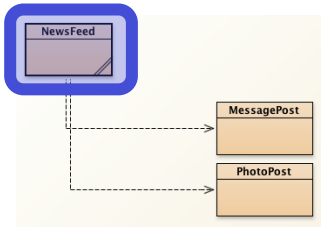


NewsFeed uden Arv!

- **NewsFeed uden Arv:** `network-v1`:



NewsFeed uden Arv




```
public class NewsFeed {
   private ArrayList<MessagePost> messages;
   private ArrayList<PhotoPost> photos;

   public NewsFeed() {
      messages = new ArrayList<MessagePost>();
      photos = new ArrayList<PhotoPost>();
   }

   public void addMessagePost(MessagePost message) {
      messages.add(message);
   }

   public void addPhotoPost(PhotoPost photo) {
        photos.add(photo);
   }

   public void show() {
      // display all text posts
      for(MessagePost message : messages) {
         message.display();
         System.out.println(); // line between posts
      }
      // display all photos
      for(PhotoPost photo : photos) {
         photo.display();
         System.out.println(); // line between posts
      }
  }
}
```

MessagePost

PhotoPost

MessagePost

PhotoPost

MessagePost

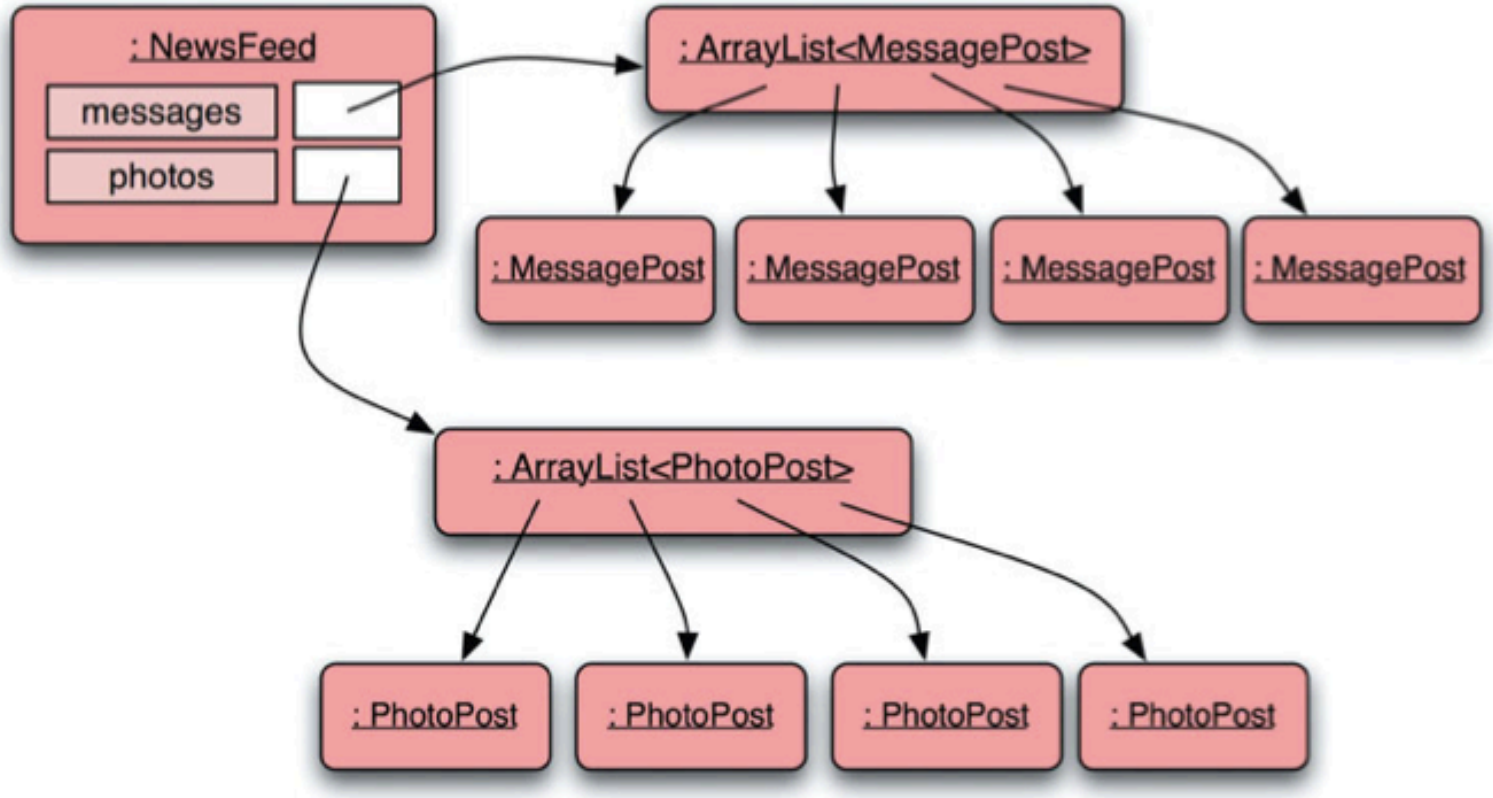
PhotoPost

MessagePost

PhotoPost

NewsFeed uden Arv!

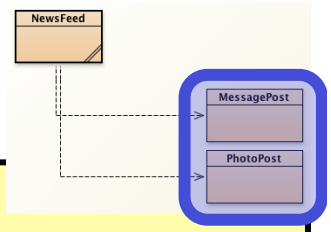
- **`NewsFeed`** har to lister (og kender til både):
  - **`ArrayList<MessagePost> messages;`**
  - **`ArrayList<PhotoPost> photos;`**



MessagePost.java

```
public class MessagePost {
  private String username;
  private String message;
  private long ts;
  private int likes;
  private ArrayList<String> comments;


  public MessagePost(String author,
                     String text) {

    username = author;
    message = text;

    ts = System.currentTimeMillis();
    likes = 0;
    comments = new ArrayList<String>();
  }

  ...
```

PhotoPost.java



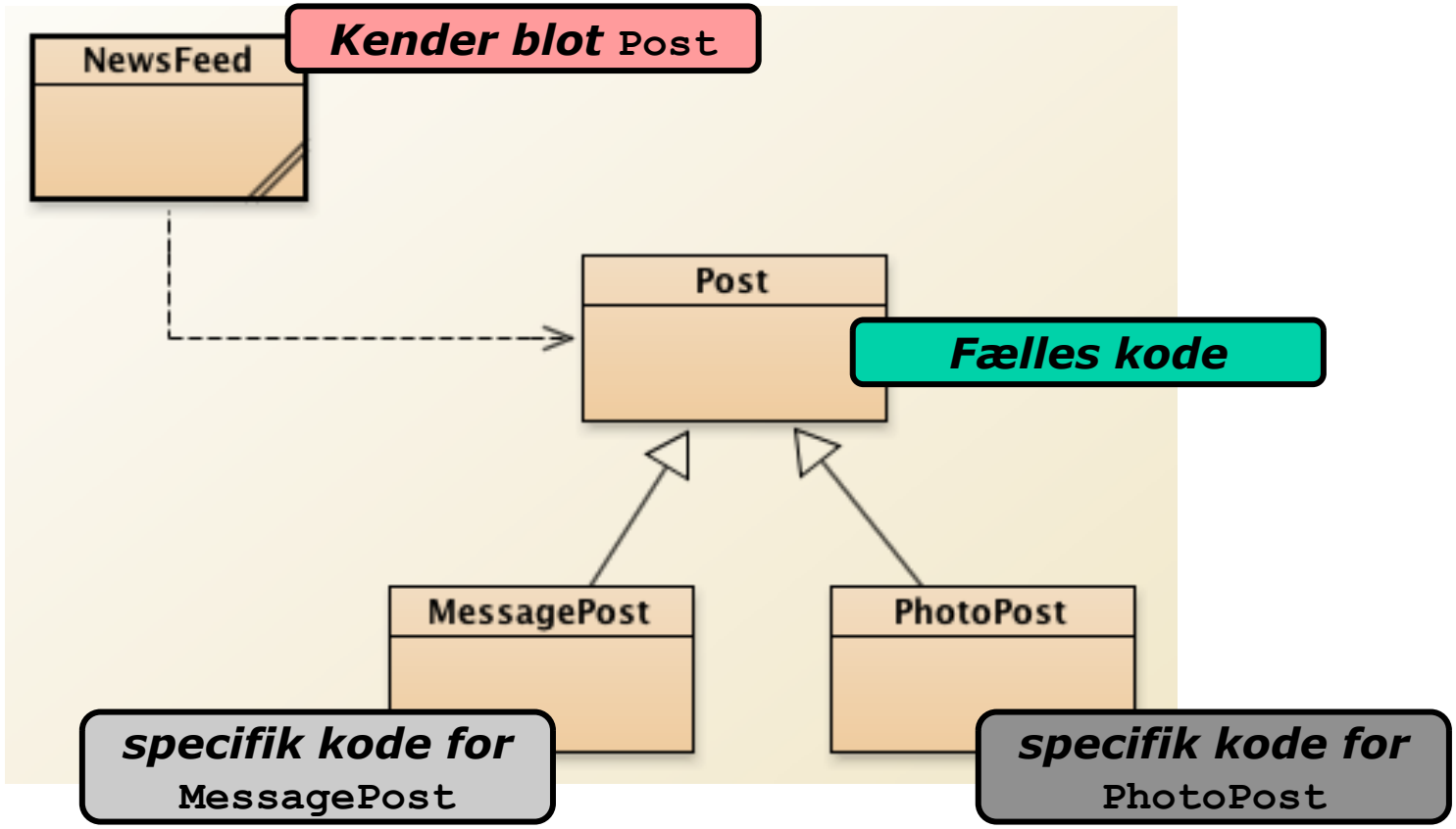

```
public class PhotoPost {
  private String username;
  private String filename;
  private String caption;
  private long ts;
  private int likes;
  private ArrayList<String> comments;

  public PhotoPost(String author,
                   String filename,
                   String caption) {
    username = author;
    this.filename = filename;
    this.caption = caption;
    ts = System.currentTimeMillis();
    likes = 0;
    comments = new ArrayList<String>();
  }

  ...
```

[+3 sider mere]

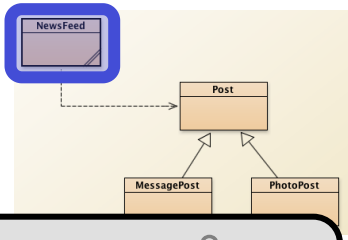
NB: Rigtigt meget ***kode-duplikering*** (redundans)

NewsFeed med Arv!

- **NewsFeed med Arv: `network-v2`:**



Den nye NewsFeed-klasse




```
public class NewsFeed {
    private ArrayList<Post> posts;

    public NewsFeed() {
        posts = new ArrayList<Post>();
    }

    public void addPost(Post post) {
        posts.add(post);
    }

    public void show() {
        // display all posts
        for (Post post : posts) {
            post.display();
            System.out.println();
        }
    }
}
```

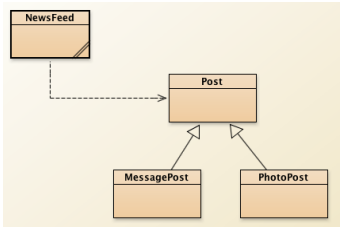
Håndterer **Post**; dvs både **MessagePost** og **PhotoPost**

Håndterer **Post**; dvs både **MessagePost** og **PhotoPost**

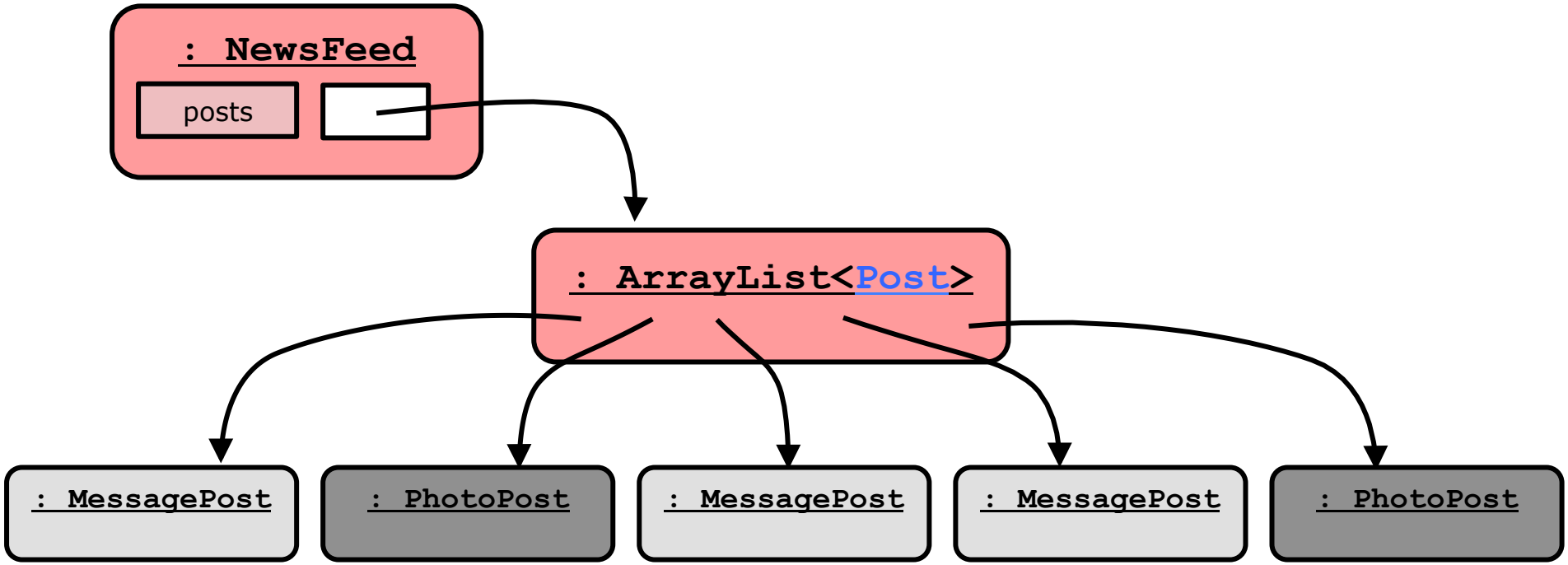
Håndterer **Post**; dvs både **MessagePost** og **PhotoPost**

- **NB:** Vi kan nu blot bruge **Post** i stedet for **MessagePost** og **P.P.**
- Nemt at udvide med nye slags; **EventPost**, **VideoPost**, ...

NewsFeed med Arv!

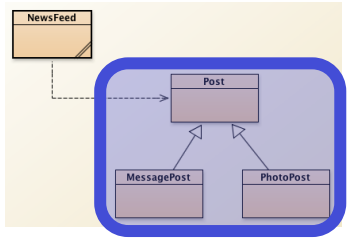


- **NewsFeed** har nu blot én liste:
  - **ArrayList<Post> posts;**



- **NB:** Højere abstraktionsniveau!
  - **NewsFeed** kender superklassen **Post** (ikke subklasserne)
  - I.e., stabil overfor tilføjelser af nye slags **Posts**!

Arv i Java




```
public class Post {
    private String username;
    private long ts;
    private int likes;
    private ArrayList<String> comments;

    ... // methods
}
```

Fælles kode

```
public class MessagePost extends Post {
    private String message;

    ... // methods
}
```

specifik kode for MessagePost

```
public class PhotoPost extends Post {
    private String filename;
    private String caption;

    ... // methods
}
```

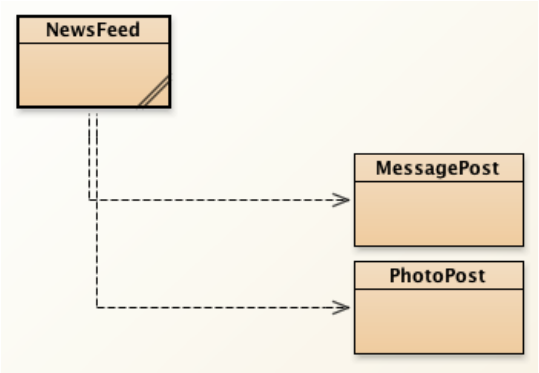
specifik kode for PhotoPost

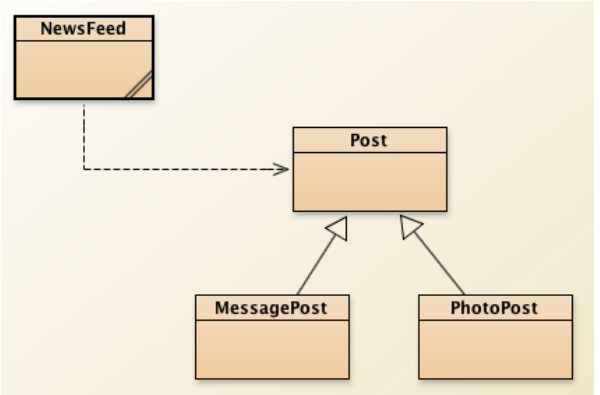
- Klasse `MessagePost` ***er-en*** *(subklasse af)* `Post`,
  hvis man kan sige: ***"a*** *`MessagePost`* ***is-a*** *`Post`*"
- Klasse `MessagePost` ***har-en*** *(instansvariabel)* `message`,
  hvis man kan sige: ***"a*** *`MessagePost`* ***has-a*** *`message`*"

A G E N D A

- **Recap**
- **Virtual dispatching!**
- **Statisk type** -vs- **dynamic type**
- **Visibility modifiers** (public, private, ...)

- Example: **`NewsFeed`** (à la Facebook):

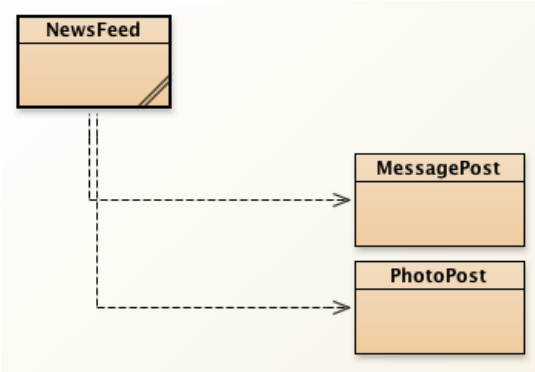




Problem med `display()`!

```
Leonardo da Vinci
Had a great idea this morning.
Something about flying...
40 seconds ago   -   2 people like this.
   No comments.

Alexander Graham Bell
[experiment.jpg]
I think I might call this thing 'telephone'.
12 minutes ago   -   4 people like this.
   No comments.
```

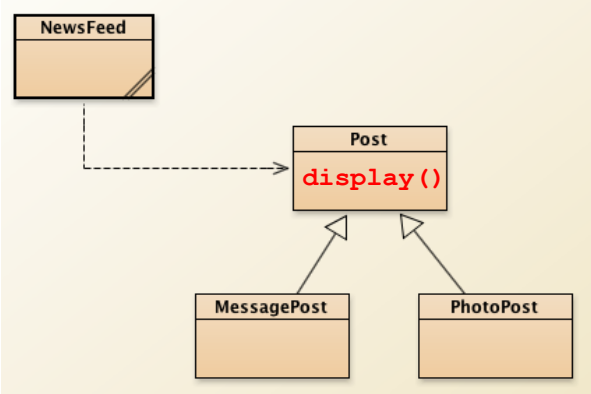
v1



- **`display()`** udskriver kun ***fælles felter:***

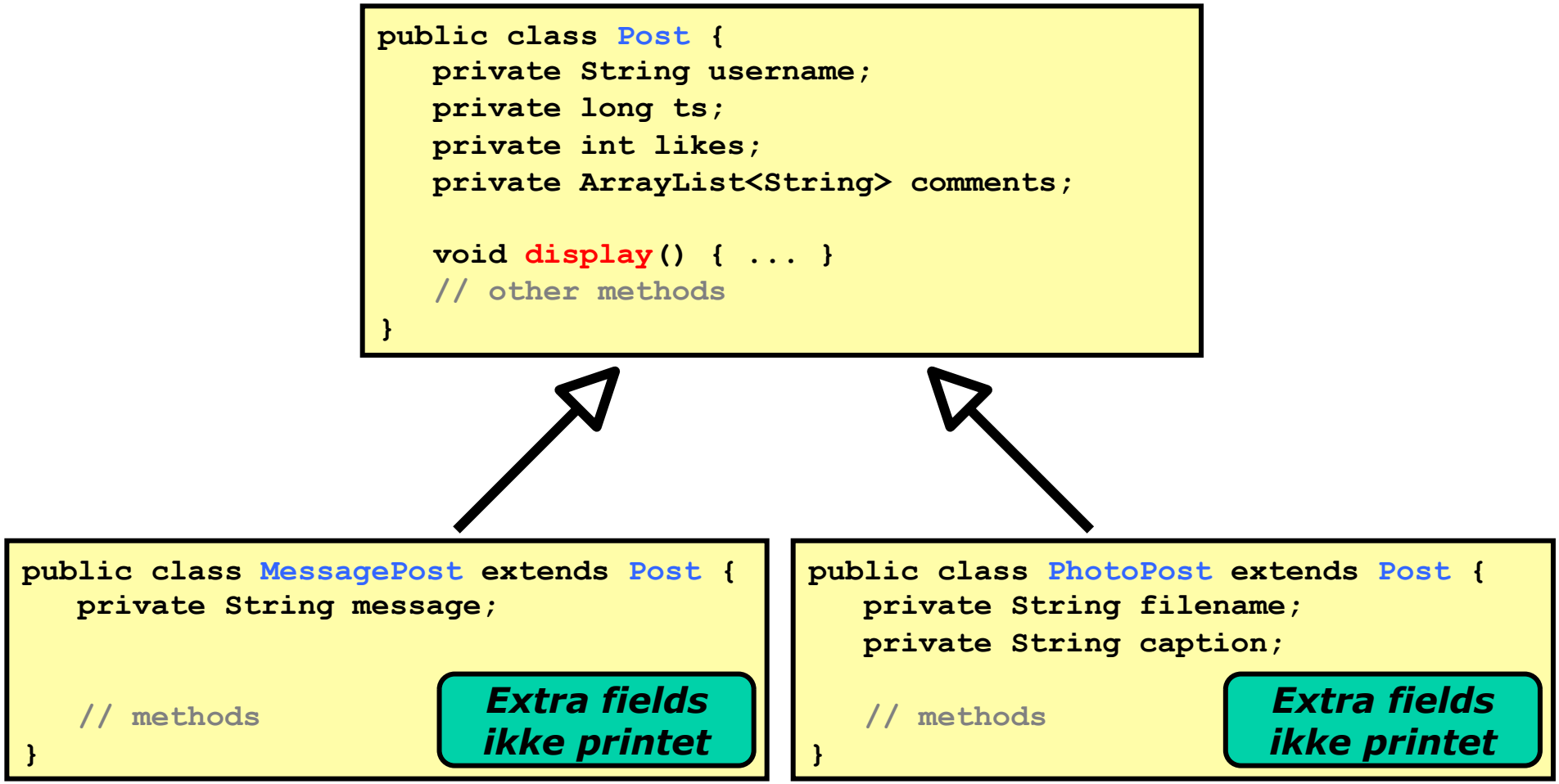
```
Leonardo da Vinci
40 seconds ago   -   2 people like this.
   No comments.

Alexander Graham Bell
12 minutes ago   -   4 people like this.
   No comments.
```

v2



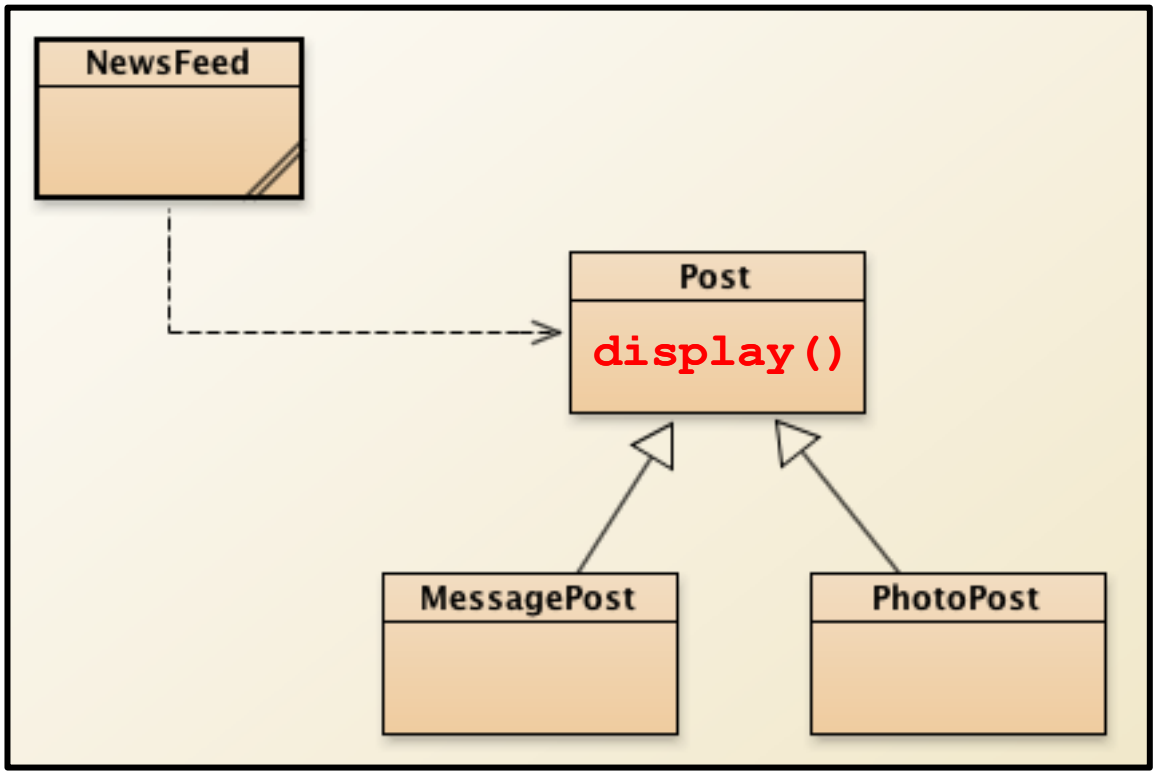
ÅRSAG: `display()` er i superklassen

- Metode **`display()`** er defineret i **`Post`** og kan derfor ikke benytte felterne i subklasserne:

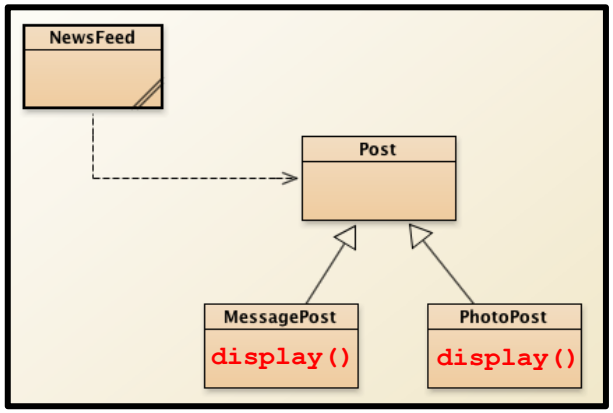


Flytte `display()`

- Flytte **`display()`** metoden:
  - Fra super-klassen til sub-klasserne !



Flytte `display()`




```
public class Post {
   private String username;
   private long ts;
   private int likes;
   private ArrayList<String> comments;
   ...
}
```

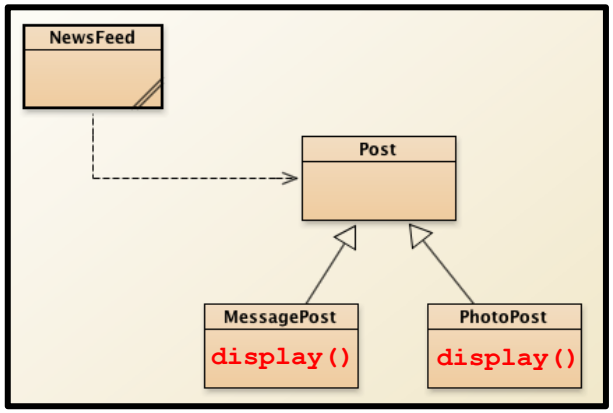
```
public class MessagePost extends Post {
   private String message;


   void display() { ... }
   // methods
}
```

```
public class PhotoPost extends Post {
   private String filename;
   private String caption;

   void display() { ... }
   // methods
}
```

Problemer:

- **1) Redundans:** MessagePost.display() -vs- PhotoPost.display()
- **2)** Vi skal bruge **accessors** til private felter i super-klassen Post
- **3)** Men også ...

NewsFeed



- **NewsFeed:**

```
public class NewsFeed {
    ...
    public void show() {
        // display all posts
        for (Post post : posts) {
            post.display();
            System.out.println();
        }
    }
}
```

*** No such method:
'Post.display()'

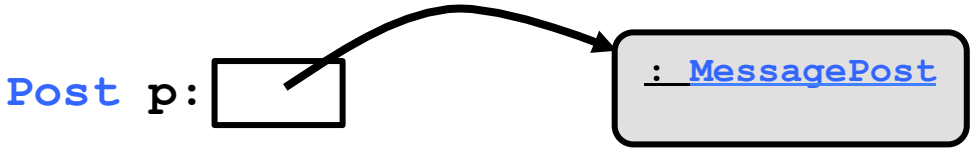
- I.e.: **compile-error i NewsFeed ! :-(**
  - `Post` **har jo ikke en** `display()` **metode!**

Static Type -vs- Dynamic Type

- Consider:

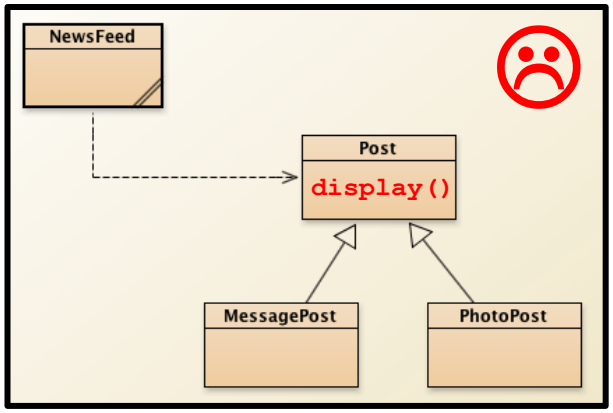
```
Post p = new MessagePost("Paris Hilton", "I just bought a purse");
...
p.like();
```

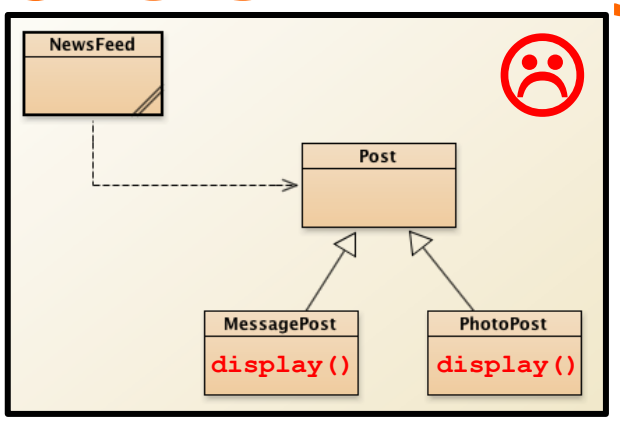
- **Q:** *"What is the **type of 'p'?"***
- **A:** *"What **kind of type** do you mean**?**" :-)*



- **Static Type = `Post`**
- **Dynamic Type = `MessagePost`**

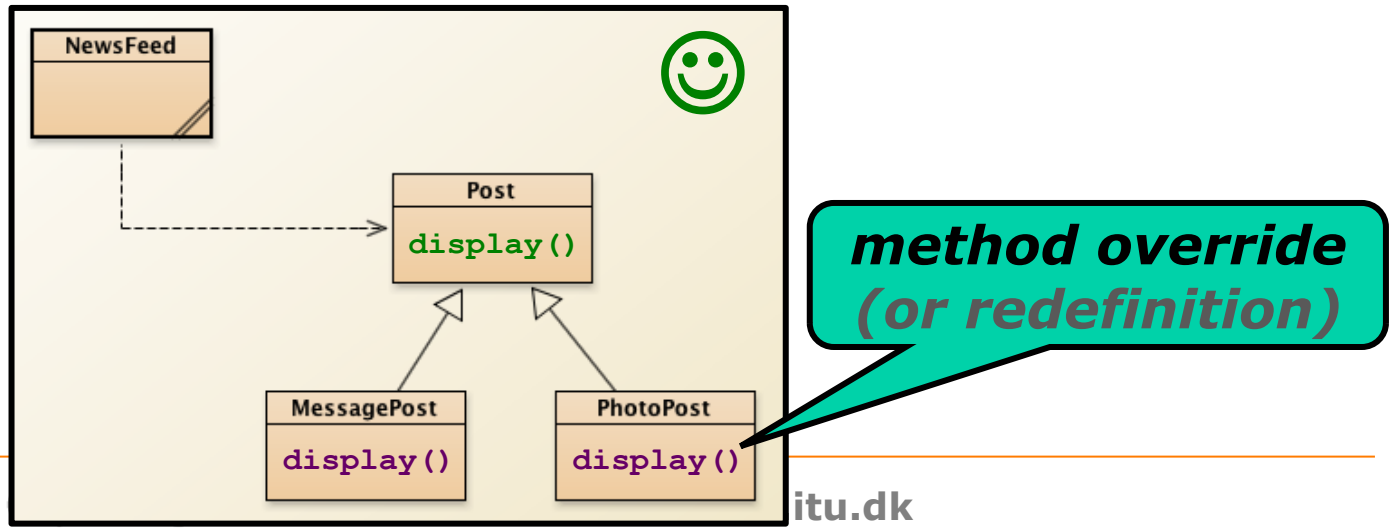
LØSNING: Metodeoverskrivning



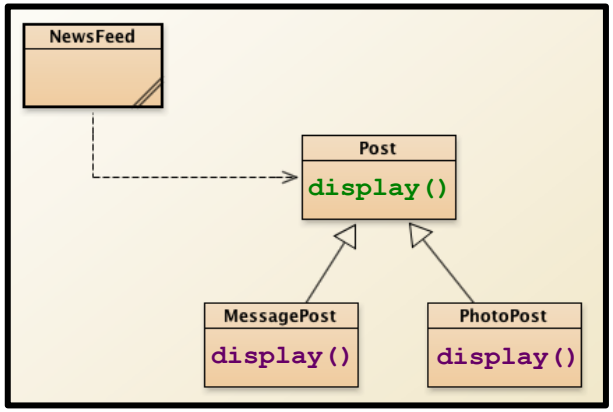


Løsning:

- **1)** ***Definer*** `display()` i superklassen (`Post`)
- **2)** og ***Overskriv*** `display()` i subklasserne **!**



Java kode: NewsFeed v3




```
public class Post {
  private String username;
  private int likes;
  ...
  public void display() {
    System.out.println(username);
    ...timestamp...
    ...likes...
    ...comments...
  }
}
```

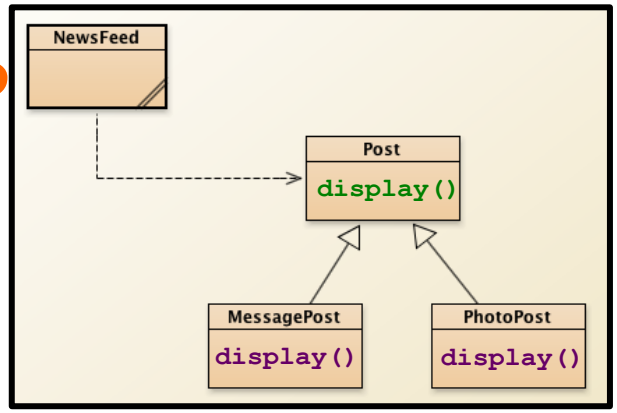
```
public class MessagePost extends Post {
  private String message;
  ...
  public void display() {
    System.out.println(message);
  }
}
```

NB: same signature

```
public class PhotoPost extends Post {
  private String filename;
  private String caption;
  ...
  public void display() {
    System.out.println("[" + filename + "]");
    System.out.println(caption);
  }
}
```

NB: same signature

Hvilken `display()` kaldes?

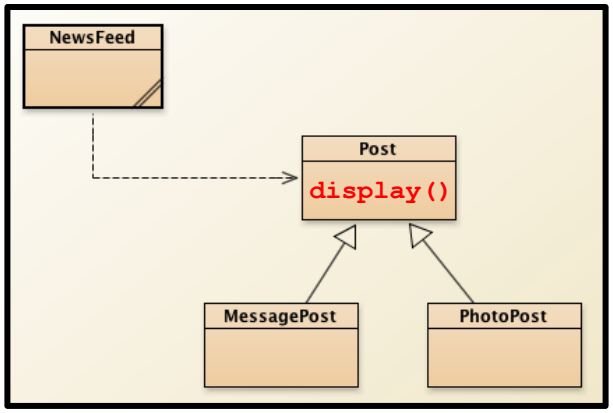


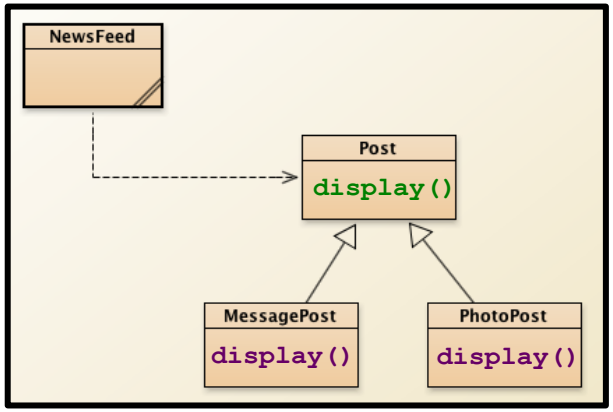
- **NewsFeed:**

```
public class NewsFeed {
    ...
    public void show() {
        for (Post post : posts) {
            post.display();
            System.out.println();
        }
    }
}
```

- **Q: Hvilken `display()` metode kaldes?!?**
  - `display()` i `Post`?
  - `display()` i `MessagePost` henholdsvis `PhotoPost`?
- **A: *Dynamic dispatching:* `display()` !!!**
  - Den ***statiske type*** tillader at vi kalder `display()`, mens
  - Den ***dynamisk type*** bestemmer hvilken kode der køres!

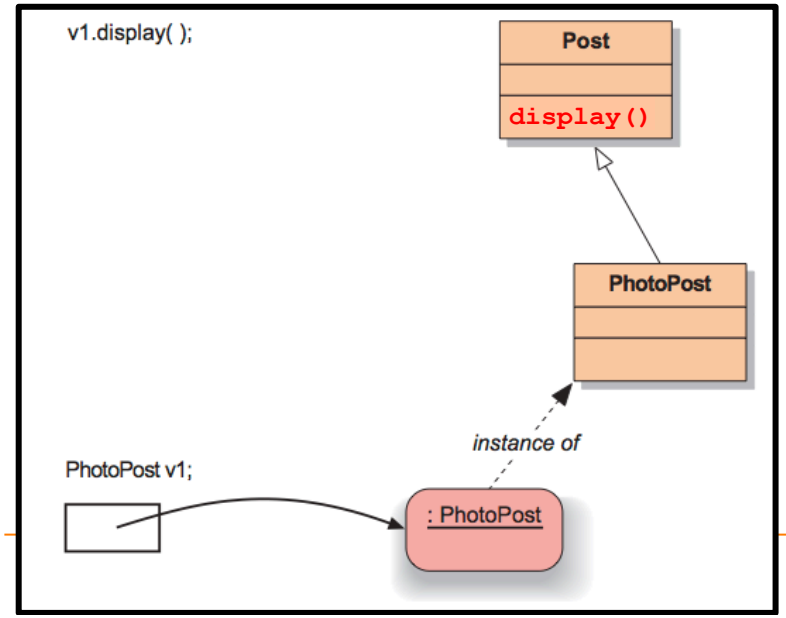
Virtual Dispatching
(= dynamic method lookup)

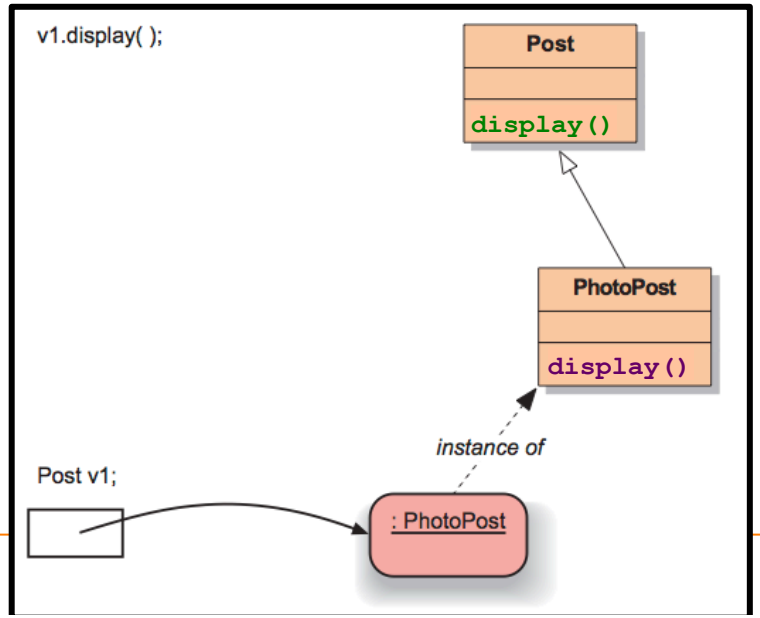





```
Post p;
p = new MessagePost(...);
p.display();
```

```
Post p;
p = new MessagePost(...);
p.display();
```

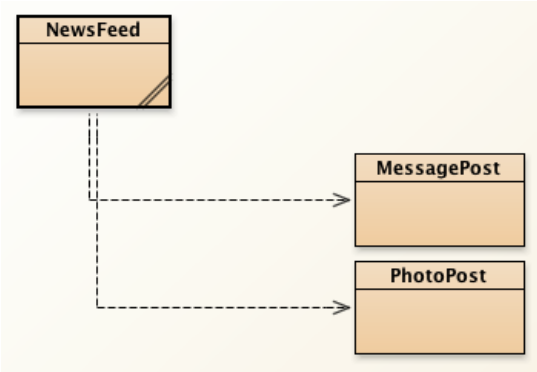




Stadig problem med `display()`!

```
Leonardo da Vinci
Had a great idea this morning.
Something about flying...
40 seconds ago  -  2 people like this.
   No comments.

Alexander Graham Bell
[experiment.jpg]
I think I might call this thing 'telephone'.
12 minutes ago  -  4 people like this.
   No comments.
```

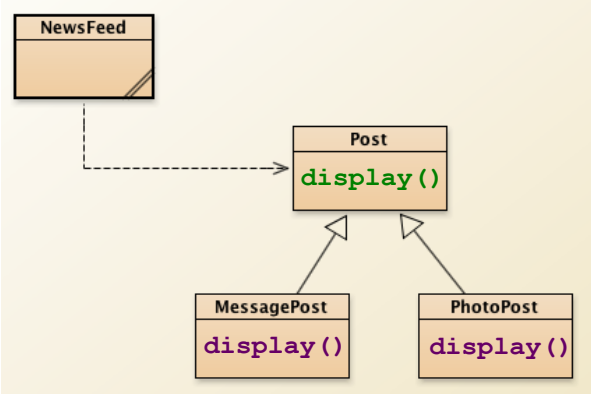
v1



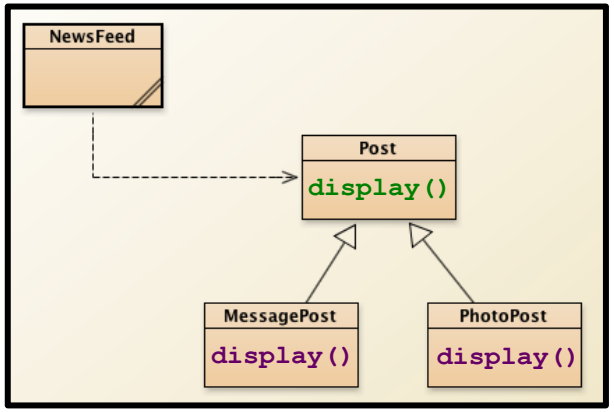
- **`display()`** udskriver ***ikke fælles felter:***

```
Had a great idea this morning.
Something about flying...

[experiment.jpg]
I think I might call this thing 'telephone'.
```

v3



Kald `super.display()` !




```
public class Post {
  private String username;
  private int likes;
  ...
  public void display() {
    System.out.println(username);
    ...timestamp...
    ...likes...
    ...comments...
  }
}
```

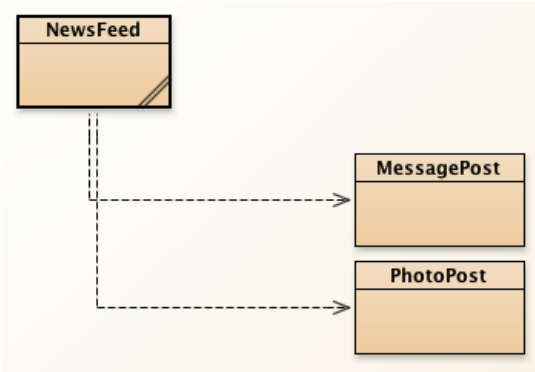
```
public class MessagePost extends Post {
  private String message;
  ...
  public void display() {
    super.display();
    System.out.println(message);
  }
}
```

```
public class PhotoPost extends Post {
  private String filename;
  private String caption;
  ...
  public void display() {
    super.display();
    System.out.println("[" + filename + "]");
    System.out.println(caption);
  }
}
```

Yeeeeeeeehaaa !!!

```
Leonardo da Vinci
Had a great idea this morning.
Something about flying...
40 seconds ago   -   2 people like this.
   No comments.

Alexander Graham Bell
[experiment.jpg]
I think I might call this thing 'telephone'.
12 minutes ago   -   4 people like this.
   No comments.
```

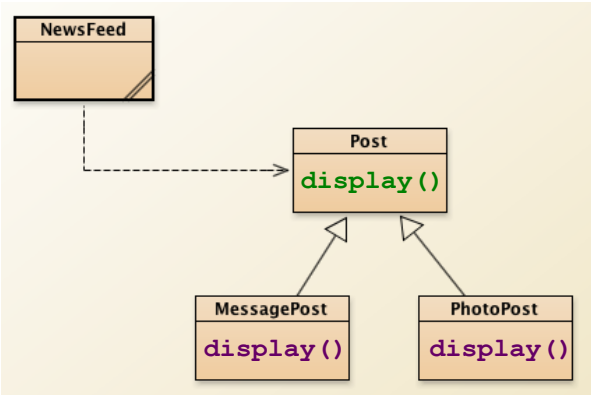
v1



- `display()` med kald til `super.display()`:

```
Leonardo da Vinci
40 seconds ago   -   2 people like this.
   No comments.
Had a great idea this morning.
Something about flying...

Alexander Graham Bell
12 minutes ago   -   4 people like this.
   No comments.
[experiment.jpg]
I think I might call this thing 'telephone'.
```

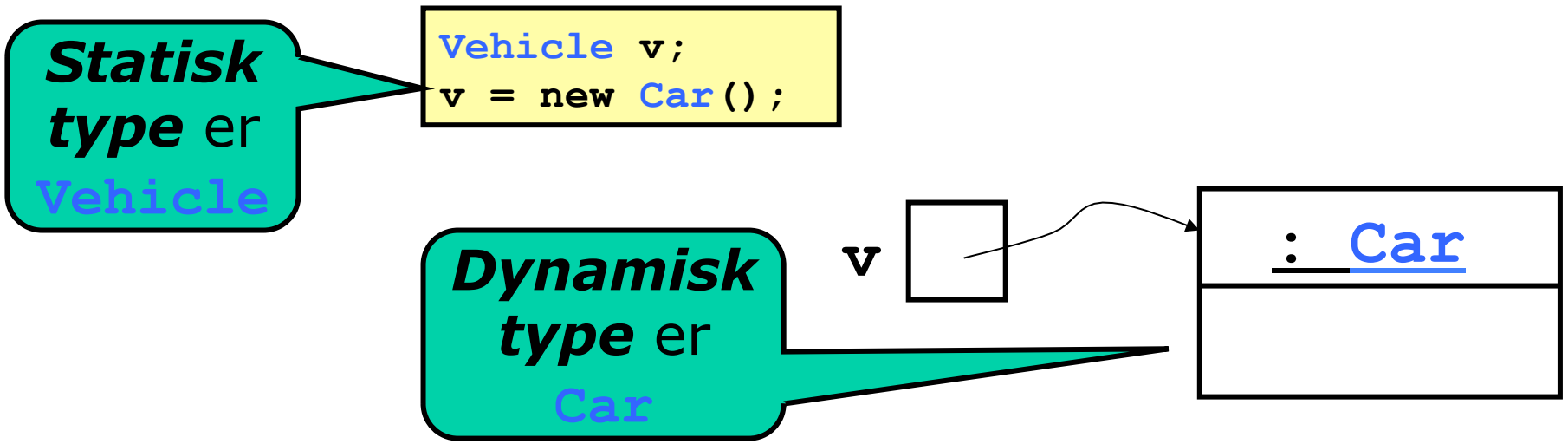
v4



Ingen kode-duplikering og "trivielt" at udvide !

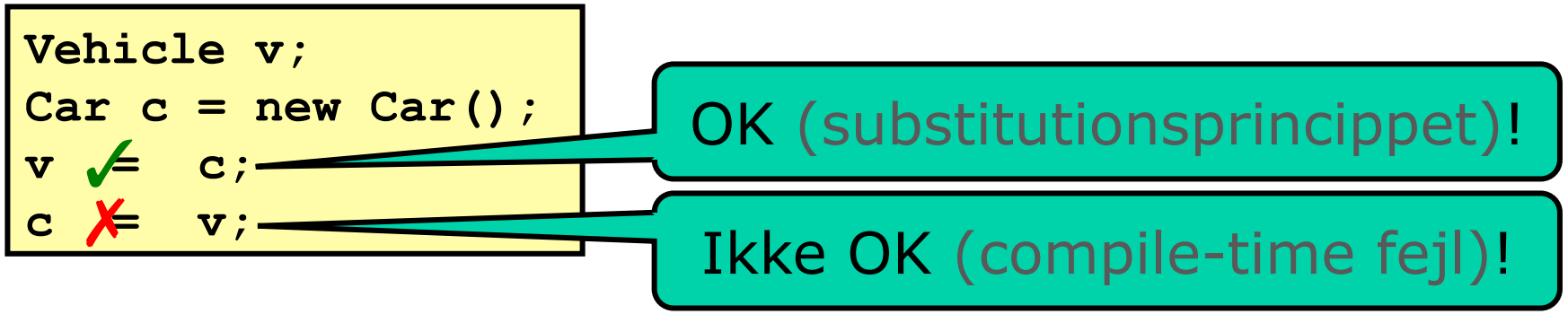
Statisk type -vs- Dynamisk type

- ***Statisk type*** (aka, *compile-time type*)
  (aka, oversættelsestidstype)
  - Den erklærede type for variabel eller parameter
- ***Dynamisk type*** (aka, *runtime type*)
  (aka, køretidstype)
  - Den faktiske klasse af objektet på køretid



Statisk type -vs- Dynamisk type

- Java-oversætteren tjekker de statiske typer
- Det er derfor oversætteren forkaster dette:

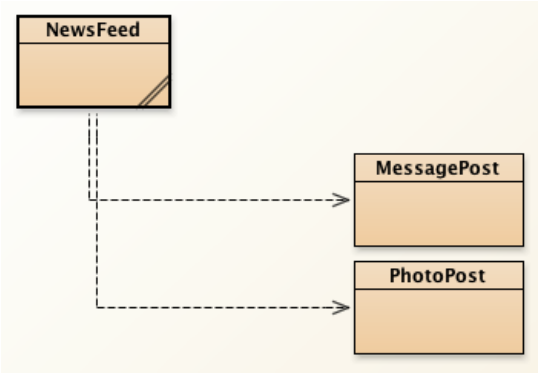


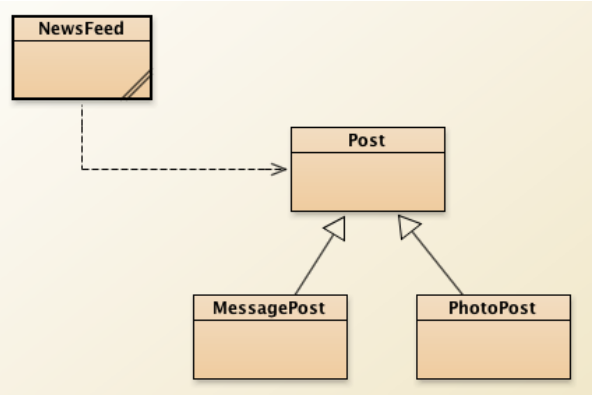
- En var med ***statisk type*** klasse **x** indeholder:
  - **null**; eller
  - en reference til et objekt hvis dynamiske type er
    - **x**; eller
    - en subklasse af **x**
- Dette gør Java til et (type-) "sikkert sprog"; i.e.:
  - Vi får aldrig missing field-or-method på runtime

A G E N D A

- **Recap**
- **Virtual dispatching!**
- **Statisk type** -vs- **dynamic type**
- **Visibility modifiers** (public, private, ...)

- Example: **`NewsFeed`** (à la Facebook):





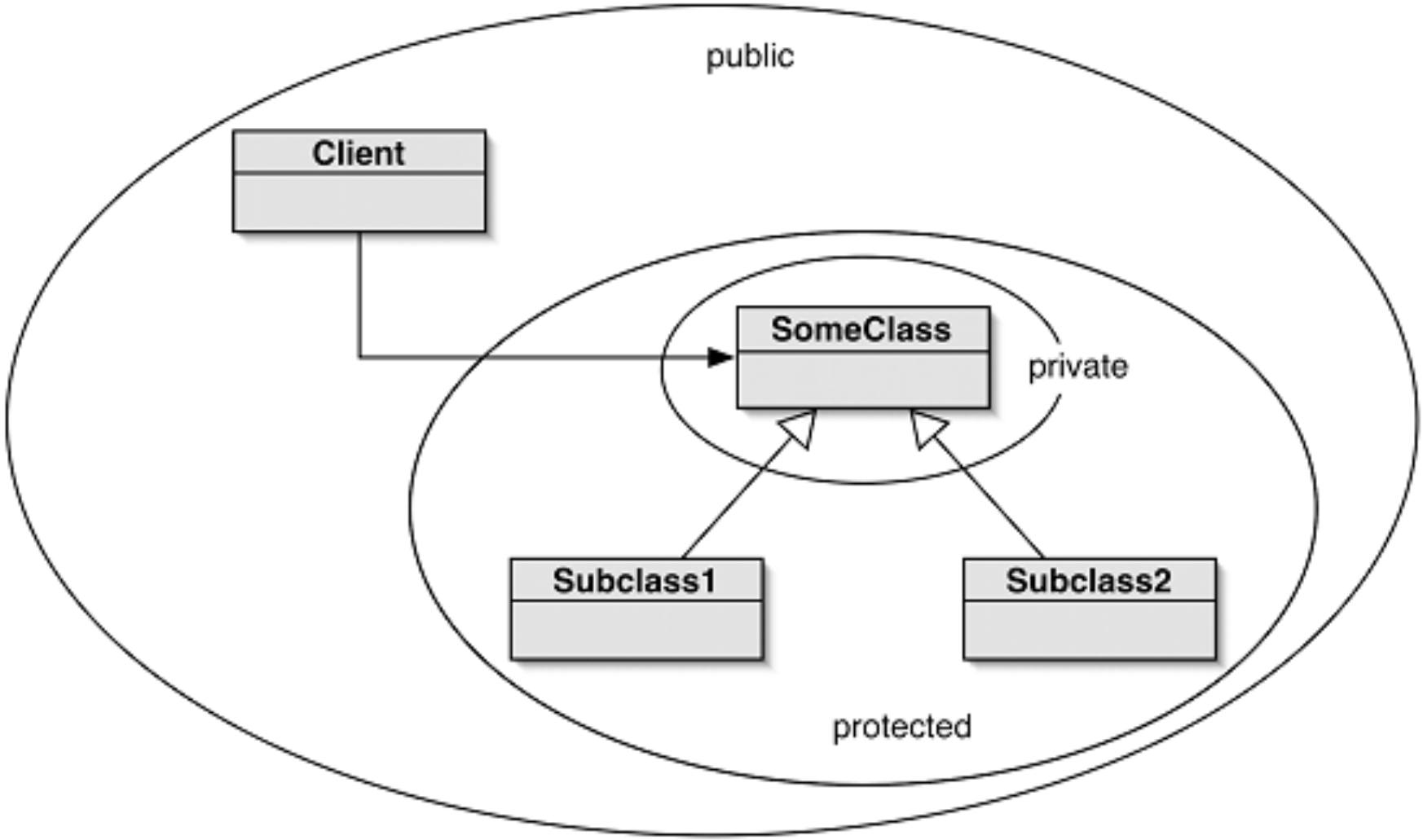
`Protected`: synlig også i subklasser

- Felterne fra `Post` er `private` og derfor ikke synlige i `MessagePost` og `PhotoPost`

- Vi kan gøre dem `protected` i stedet, så er de synlige i subklasser af Post:
  - i.e., `MessagePost` og `PhotoPost`

- Generelt skal felter være `private`
- (Metoder og constructors er ofte `protected` i stedet for `private`)

Visibility modifiers:
(public, protected, private)



Bedre at overskrive `toString()`

```
class Object {
   public String toString() { ... } // udskriver objektets #hashcode
   public boolean equals(Object other) { ... }
   ... // nogle få andre metoder
}
```

- De arves af alle klasser og kan derfor ***overskrives:***

```
public class Post { // equivalent to: "extends Object"!
   private String username;
   ...
   public String toString() { // method overwrite!
      return "post by '" + username + "'" ;
   }
}
```

```
public class MessagePost extends Post {
   private String message;
   ...
   public String toString() { // method overwrite (using super)!
      return super.toString() + " with messsage '" + message + "'" ;
   }
}
```

Smart fordi ...

- Man kan lave `toString()` og undvære `print()`:

```
public class NewsFeed {
    ...
    public void show() {
        for (Post post : posts) {
            System.out.println(post.toString()); // toString()!
        }
    }
}
```

- Så behøver man ikke skrive kaldet `toString()`:

```
public class NewsFeed {
    ...
    public void show() {
        for (Post post : posts) {
            System.out.println(post);
        }
    }
}
```

`toString()` kaldes automatisk af `System.out.println`

OPGAVE

[8.12]: Assume that we have four classes:

- **`Person`**, **`Teacher`**, **`Student`**, and **`PhDStudent`**.

(**`Teacher`** and **`Student`** are both subclasses of **`Person`**. **`PhDStudent`** is a subclass of **`Student`**.)

- **a)** Which of the following assignments are legal, and why or why not?

```
Person p = new Student();
Person p = new PhDStudent();
PhDStudent x = new Student();
```

```
Teacher t = new Person();
Teacher t = new Student();
Student s = new PhDStudent();
```

- Suppose we have the following (legal) declarations and assignments:

```
Person p = new Person();
PhDStudent x = new PhDStudent();
```

```
Teacher t = new Teacher();
Student s = new Student();
```

- **b)** Based on those just mentioned, which of the following assignments are legal, and why or why not?

```
s = p;
```

```
p = s;
```

```
t = s;
```

```
s = x;
```

```
x = s;
```

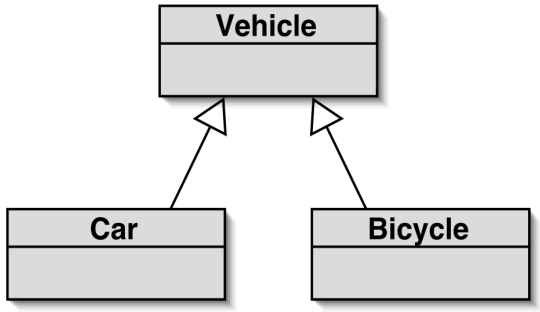
[8.13]: Test your answers to the previous question by creating bare-bones versions of the classes mentioned in that exercise and trying it out.

Dynamisk Casting & "instanceof"

- Vi kan spørge om et objekt er af en given klasse:

  <EXP> instanceof <classname>

  (...gives us a boolean answer)




```
Vehicle v;
...
v = new Car(); // ok (substitutionsprincippet)!
...
v.start();
if (v instanceof Car) {
    Car c = (Car) v;
    c.drive();
    ...
}
```

true hvis og kun hvis:
v *peger på* et objekt afklasse **Car**
(eller en subklasse af **Car**)

Cast fejler ***ikke*** da:
v instanceof Car

Equality

- ***Reference equality:*** '**==**' (same object)

```
MessagePost p = new MessagePost("Obama", "The State of the union is good!");
MessagePost q = new MessagePost("Obama", "The State of the union is good!");
if (p == q) {
   System.out.println("same!");
}
```

nothing is printed!

- -vs- ***Equality*****:** '**x.equals(y)**' (equivalence)

```
public class MessagePost extends Post {
   ...
   public boolean equals(Object obj) { // override Object::equals()
      if (this == obj) return true;
      if (!(obj instanceof MessagePost)) return false;
      MessagePost other = (MessagePost) obj;
      return username.equals(other.username) && message.equals(other.message);
   }
}
```

Warning: must effectively define an ***"equivalence relation"!***

```
MessagePost p = new MessagePost("Obama", "State of union is good");
MessagePost q = new MessagePost("Obama", "State of union is good");
if (p.equals(q)) {
   System.out.println("equal!");
}
```

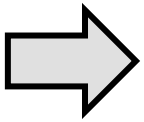
equal!

Warning: when overriding `equals()`, you should also override `hashcode()`

Klassisk begynderfejl

- Klassisk begynderfejl:

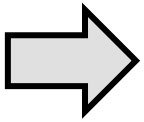
```
boolean b;
b = ...;
if (b == true) {
   ...
} else {
   ...
}
```



```
boolean b;
b = ...;
if (b) { // allerede boolean!
   ...
} else {
   ...
}
```

- Tilsvarende:

```
boolean b;
b = ...;
if (b == false) {
   ...
} else {
   ...
}
```

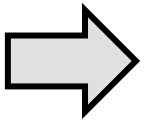


```
boolean b;
b = ...;
if (!b) {
   ...
} else {
   ...
}
```

The `switch` Statement

- Mange nestede **`if`**'er kan i stedet skrives:

```
if (a == 1) {
    ...
} else if (a == 2) {
    ...
} else if (a == 3) {
    ...
} else {
    ...
}
```



```
switch (a) {
  case 1:
    ...
    break;
  case 2:
    ...
    break;
  case 3:
    ...
    break;
  default:
    ...
    break;
}
```

Thx!

Spg?