Design patterns

W. Oele

23 juli 2014

Deze les

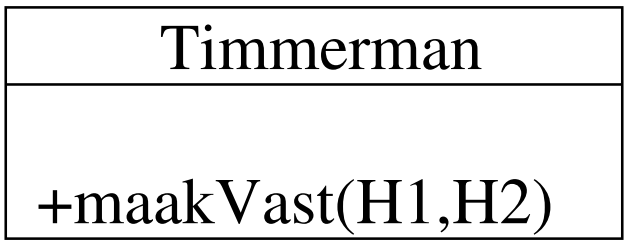
- Het strategy patroon
- Het bridge patroon

Het strategy pattern

- naam: het strategy pattern
- doel: algoritmen runtime kunnen inzetten
- hoe: implementatie van algoritmen scheiden van selectie
- gevolgen: alle algoritmen moeten dezelfde interface hebben

Het strategy pattern

- Een timmerman moet twee stukken hout aan elkaar vastmaken.
- Hij doet dat met een hamer en spijkers.



Het strategy pattern

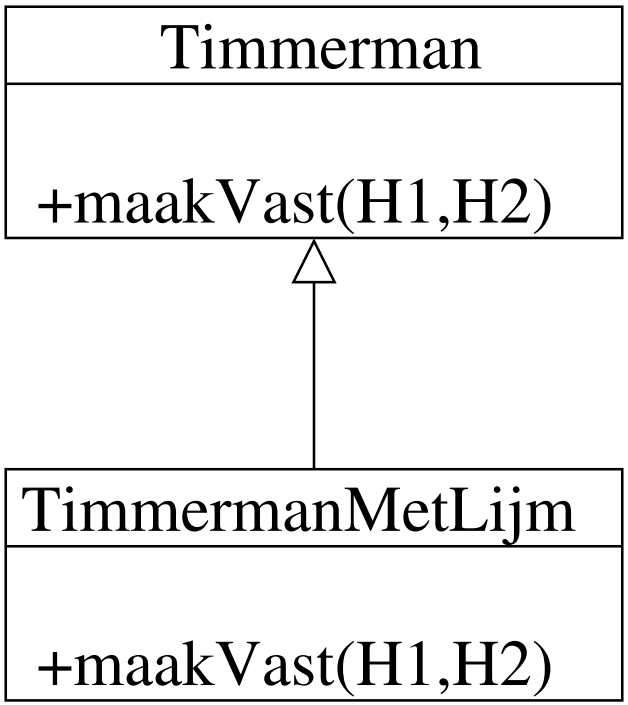
```
public class Timmerman
{
public Constructie maakVast(H1, H2)
   {
      ...
      timmeren met hamer en spijkers;
      ...
      return constructie;
   }
}
```

Het strategy pattern

Sommige constructies worden *niet* met hamer en spijkers gemaakt:

- houtlijm

Een poging



Een poging

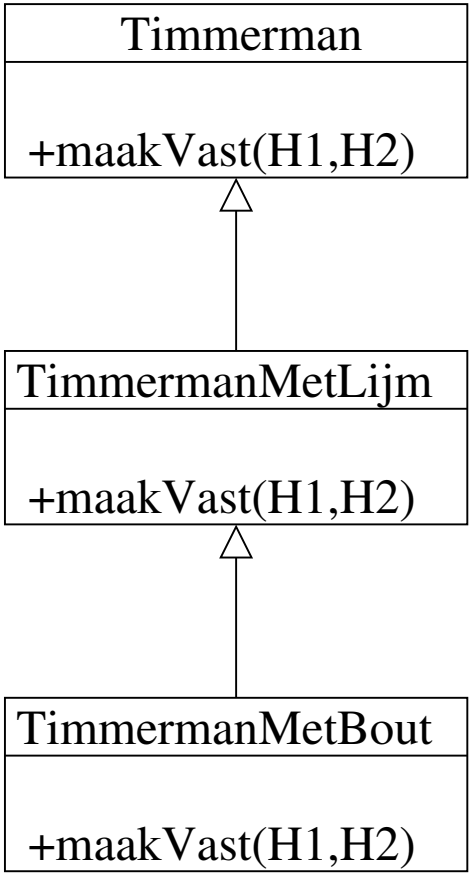
```
public class TimmermanMetLijm extends Timmerman
{
   public Constructie maakVast(H1, H2)
   {
      ...
      stukken hout vastmaken met lijm ;
      ...
      return constructie;
   }
}
```

Het strategy pattern

Sommige constructies worden *niet* met hamer en spijkers gemaakt:

- bouten en moeren

Een poging


Timmerman
+maakVast(H1,H2)
TimmermanMetLijm
+maakVast(H1,H2)
TimmermanMetBout
+maakVast(H1,H2)

Het strategy pattern

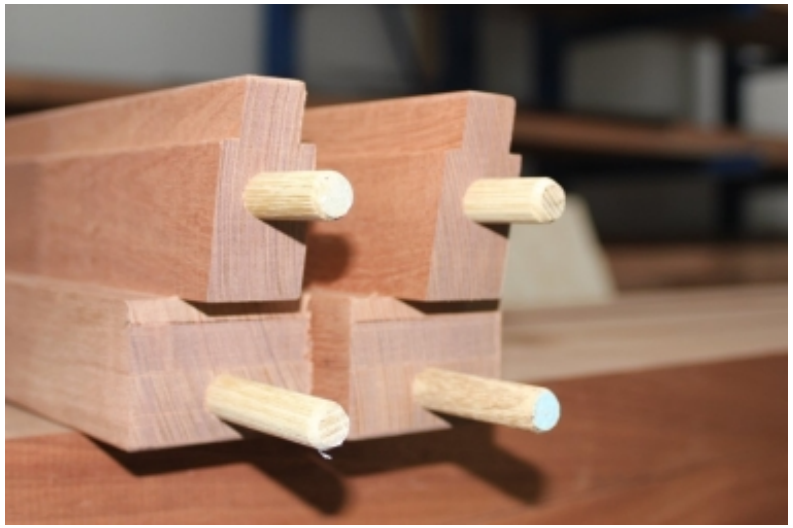
Sommige constructies worden *niet* met hamer en spijkers gemaakt:

- schroeven

Het strategy pattern

Sommige constructies worden *niet* met hamer en spijkers gemaakt:

- deuvels



Het strategy pattern

Mogelijke manieren om twee stukken hout te verbinden:

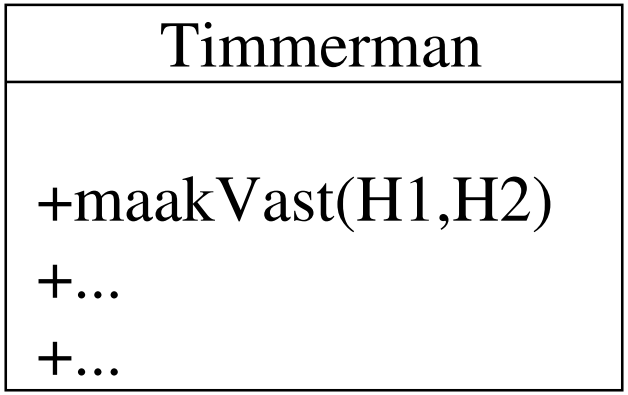
- spijkers
- houtlijm
- bouten en moeren
- schroeven
- deuvels

Het strategy pattern

- Voorgaande manier van ontwerpen leidt tot een lange keten van afgeleide klassen.
- Niet fraai
- Foutgevoelig

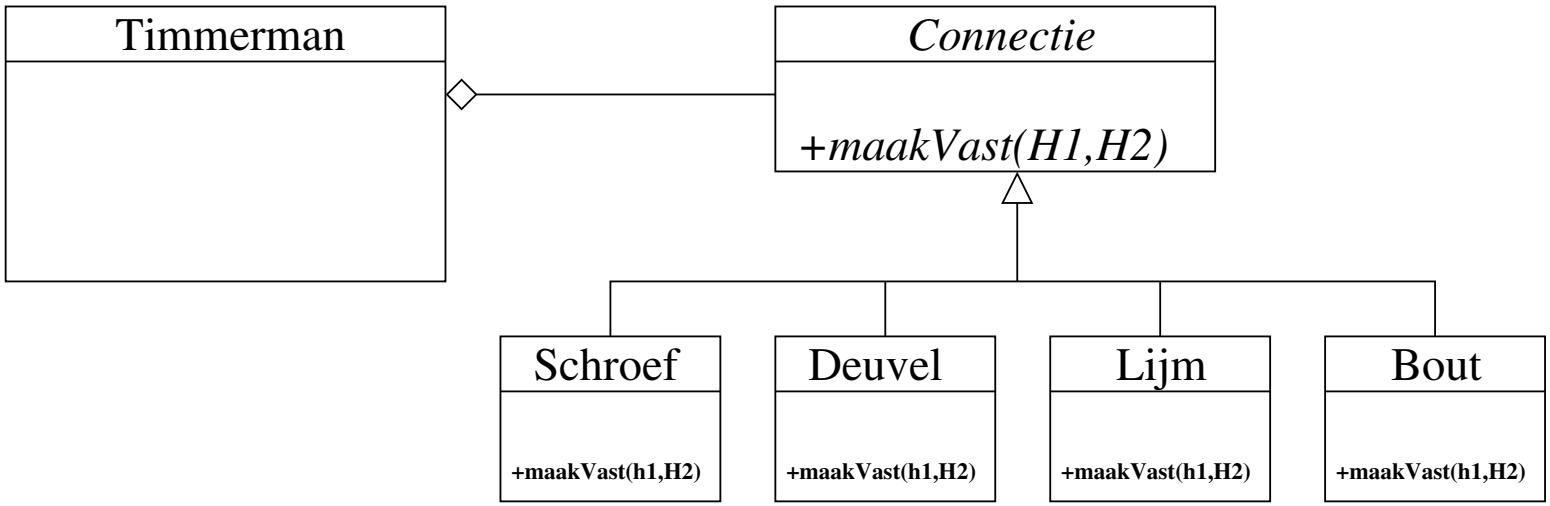
Het strategy pattern

- Een timmerman kan, naast 2 stukken hout aan elkaar vast maken, veel meer doen.
- Het afleiden van allerlei timmermannen is derhalve niet handig.



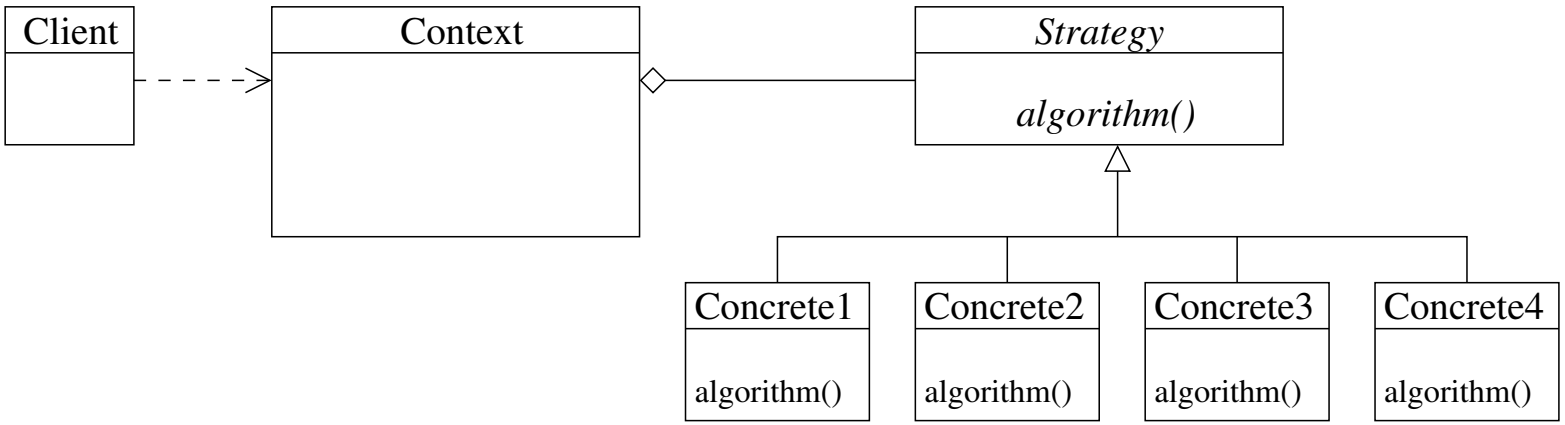
Het strategy pattern

- Wat varieert is iedere keer het aan elkaar vast maken van twee stukken hout.
- We kapselen dit in in een aparte klasse:



"Find what varies and encapsulate it. . . "

Het strategy pattern: algemeen



Richtlijnen

- Find what varies and encapsulate it.
- Denk over een klasse in termen van verantwoordelijkheden:
  - Wie is verantwoordelijk voor wat?
  - Teveel verantwoordelijkheid? $\rightarrow$ opsplitsen

Het bridge pattern

- naam: het bridge patroon
- doel: decoupling: implementaties scheiden van abstractie
- hoe: interface ontwerpen voor alle afgeleide klassen
- gevolgen:
  - clients weten niets van implementatie
  - verbeterde uitbreidbaarheid
  - geen combinatorische explosies

Het probleem

Een machine produceert verschillende soorten glaswerk:

- erlenmeyer
- maatcylinder
- kolf
- fles
- maatbeker
- etc.

Het probleem

Een machine produceert verschillende soorten glaswerk:

- erlenmeyer
- maatcylinder
- kolf
- fles
- maatbeker
- etc.

Elk van deze soorten dient voorzien te worden van:

- opdruk
- graveerwerk
- stickers

Het probleem

Een machine produceert verschillende soorten glaswerk:

- erlenmeyer
- maatcylinder
- kolf
- fles
- maatbeker
- etc.

Elk van deze soorten dient voorzien te worden van:

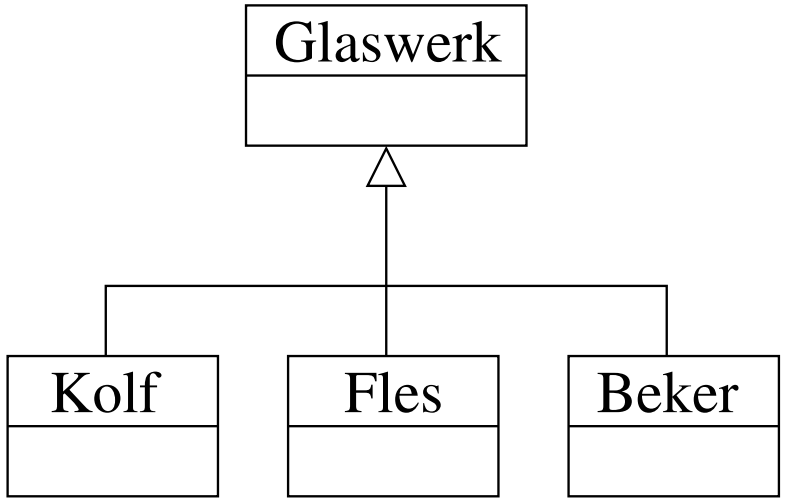
- opdruk
- graveerwerk
- stickers

Ook voor het graveerwerk hebben we een machine. . .

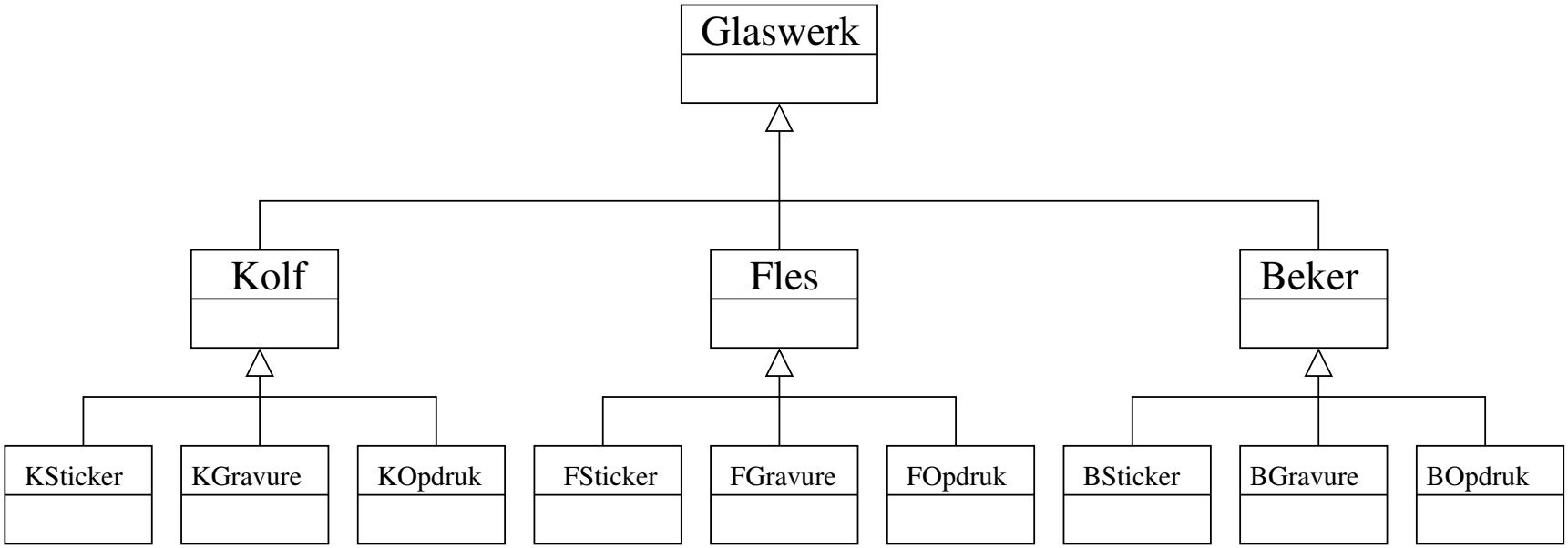
Glaswerk(1)

Glaswerk(2)

Het model: poging 1

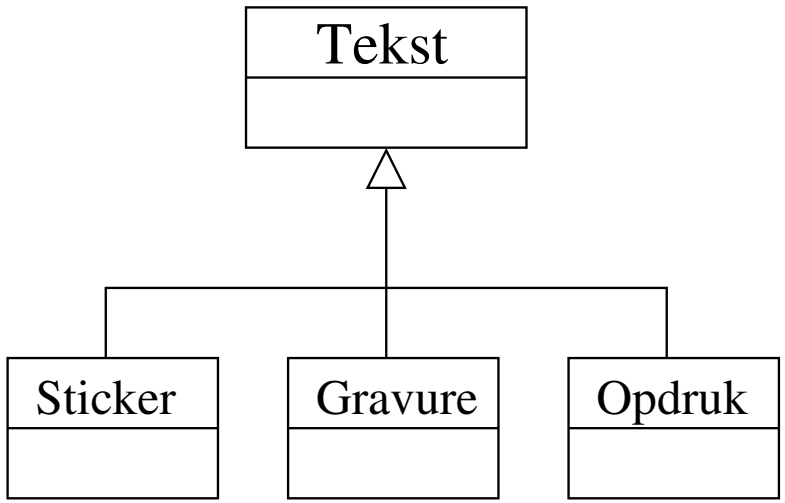


Het model: poging 1

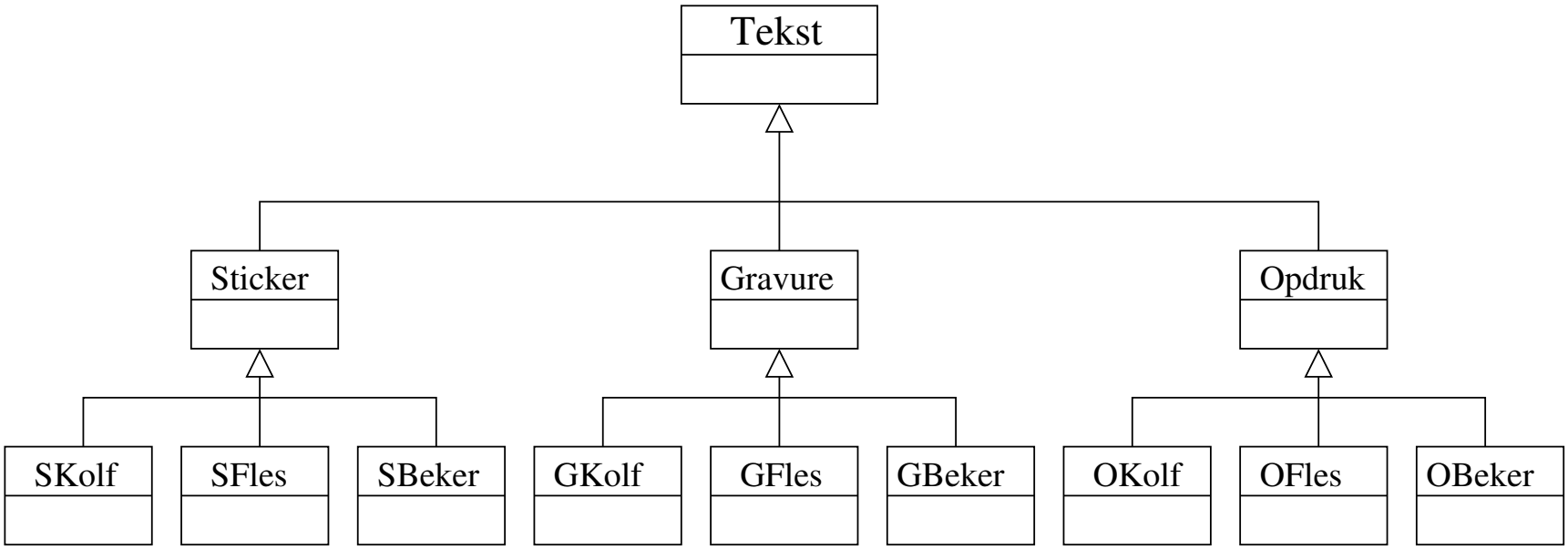


Slechts drie soorten glaswerk en drie soorten tekst: 13 klassen!

Het model: poging 2



Het model: poging 2



Slechts drie soorten tekst en drie soorten glaswerk: opnieuw 13 klassen!

Het bridge pattern

Welke abstracties zitten er in het probleem?

Het bridge pattern

Welke abstracties zitten er in het probleem?

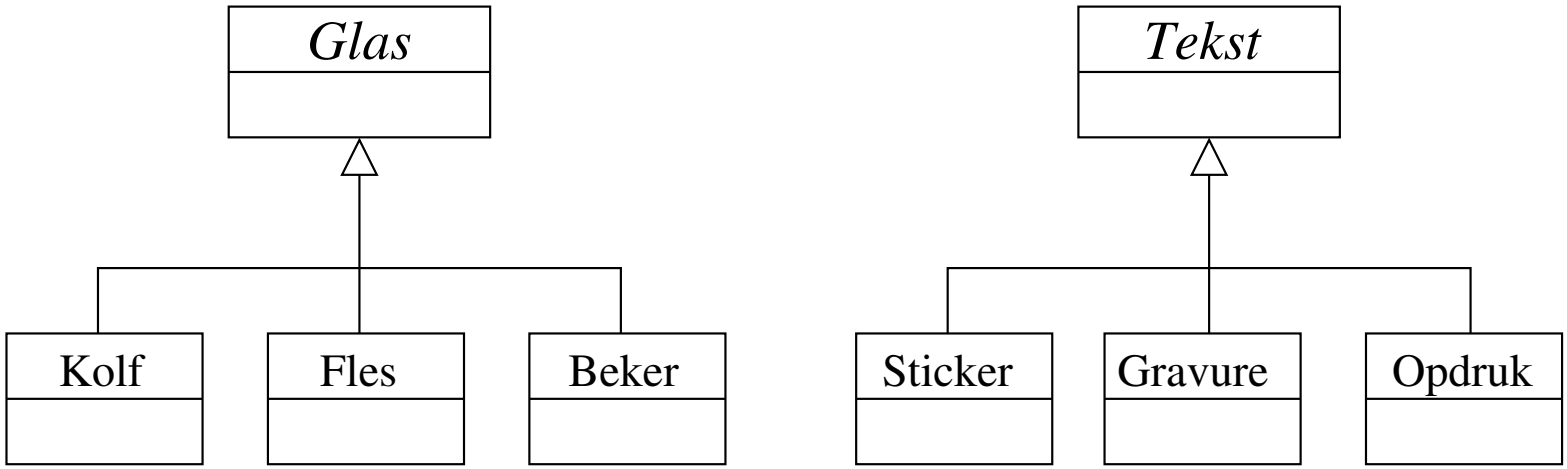
- verschillende soorten glaswerk:
  - kolf, fles, beker,. . .

Het bridge pattern

Welke abstracties zitten er in het probleem?

- verschillende soorten glaswerk:
  - kolf, fles, beker,...
- verschillende manieren om "iets" op het glas te krijgen:
  - sticker, gravure, opdruk,...

Het bridge pattern



Het bridge pattern

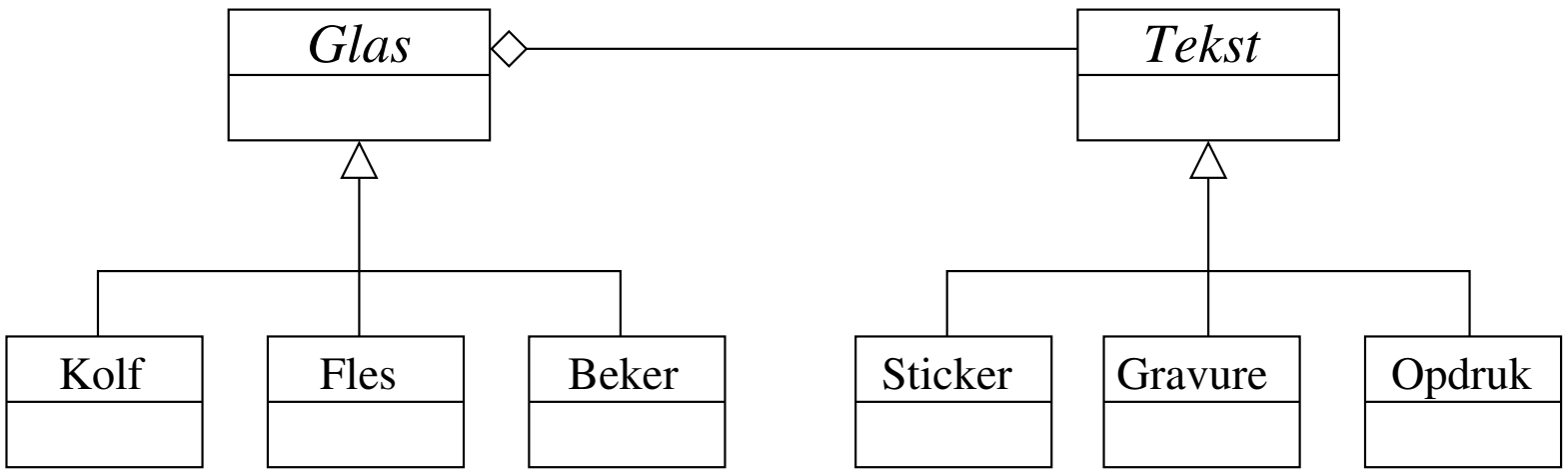
Wat hebben beide abstracties met elkaar te maken?

Het bridge pattern

Wat hebben beide abstracties met elkaar te maken?

- Een tekst heeft een glas om op te staan?
- Een glas heeft een tekst die er op één of andere manier op gezet is.

Het bridge pattern



Het bridge pattern

```
public abstract class Glas
{
   protected Tekst tekst;
   public abstract void plaatsTekst(String s);

   public Glas(Tekst t)
   {
      tekst=t;
   }

}
```

Het bridge pattern algemeen