Design patterns

W. Oele

1 juli 2014

Deze les

- introductie design patterns

Design patterns

Vragen van begin jaren '90:

- Komen problemen bij het maken van software iedere keer weer terug in een net iets andere vorm?
- Is het handig deze veel voorkomende problemen onder te brengen in een verzameling patronen?
- Kun je bij het oplossen van een probleem de oplossing makkelijker vinden als je een bestaand patroon herkent in het probleem?

Categorisatie

- Design patterns zijn het resultaat van een stelselmatige categorisatie van veel voorkomende, zich iedere keer herhalende, problemen bij het ontwerpen en aanpassen van software.

Categorisatie

- Design patterns zijn het resultaat van een stelselmatige categorisatie van veel voorkomende, zich iedere keer herhalende, problemen bij het ontwerpen en aanpassen van software.
- Het is niet gezegd dat je middels categorisatie tot een beter inzicht in de materie komt of dat dit leidt tot vernieuwingen.

Categorisatie

- Design patterns zijn het resultaat van een stelselmatige categorisatie van veel voorkomende, zich iedere keer herhalende, problemen bij het ontwerpen en aanpassen van software.
- Het is niet gezegd dat je middels categorisatie tot een beter inzicht in de materie komt of dat dit leidt tot vernieuwingen.
- Categorisatie is echter een stelselmatige manier om overzicht te creeëren en *kan* dus zijn nut hebben. . . Beter dan niets.

Categorisatie

- Design patterns zijn het resultaat van een stelselmatige categorisatie van veel voorkomende, zich iedere keer herhalende, problemen bij het ontwerpen en aanpassen van software.
- Het is niet gezegd dat je middels categorisatie tot een beter inzicht in de materie komt of dat dit leidt tot vernieuwingen.
- Categorisatie is echter een stelselmatige manier om overzicht te creeëren en *kan* dus zijn nut hebben. . . Beter dan niets.

Wat is een design pattern?

Een design pattern bestaat uit een aantal elementen:

- de naam van het patroon
- het doel van het patroon: Wat los je ermee op?
- hoe bereik je die oplossing?
- wat zijn de gevolgen en waarmee rekening te houden?

Waarom design patterns te gebruiken?

Twee belangrijke redenen:

- hergebruik van eerder gevonden oplossingen: We gaan niet iedere keer hetzelfde wiel opnieuw uitvinden.
- standaardisatie: Design patterns vormen een mooie kapstok van begrippen waar een team van specialisten zich aan kan vasthouden.

Andere redenenen

- het in perspectief plaatsen van denkwijzes
- bepalen of een gevonden oplossing de *juiste* is i.p.v. slechts *een* oplossing die werkt
- tot code komen die gemakkelijker te veranderen is
- verhoging van flexibiliteit, niets zo erg als specificaties die tijdens het implementeren veranderen. . . komt echter vaak voor
- gemakkelijker alternatieven vinden

Het facade pattern

- naam: het facade patroon
- doel: een complex systeem kunnen gebruiken zonder er alles van af te hoeven weten
- hoe: bouw een interface voor je client applicatie
- gevolgen: beperkte functionaliteit

Voorbeeld

Stel je een een ingewikkelde database applicatie voor, waarin wordt bijgehouden:

- wie waar woont
- wat een huis kost
- huur of koop
- vrijstaand huis/rijtjeshuis/flat
- tuin of niet
- enz.

Voorbeeld

De opdracht:

- bouw een programma, waarmee de klant kan zien of het aantal koopwoningen in een bepaalde stad groeit of slinkt t.o.v. het aantal huurwoningen

Traditionele oplossing:

- bouw een programma dat helemaal integreert in het bestaande programma...
  - zeer ingewikkeld
  - foutgevoelig
  - levert een hele hoop onnodige rompslomp op waar de klant niet op zit te wachten

De facade

- de client applicatie hoeft niet alles van het ingewikkelde systeem te weten
- bouw een een facade klasse die met (onderdelen van) het complexe systeem communiceert
- de facade vormt een interface voor de client applicatie:
  - simpeler
  - gericht op wat de client applicatie nodig heeft zonder overbodige rommel
  - respecteert de compliteit van het systeem zonder deze te willen wijzigen

Het adapter pattern

- naam: het adapter patroon
- doel: de client applicatie laten samenwerken met een ingewikkeld systeem zonder dit systeem te hoeven aanpassen
- hoe: bouw een adapter tussen systeem en client applicatie, de adapter is een soort vertaler
- gevolgen: bestaande systemen kunnen worden uitgebreid zonder rekening te hoeven houden met de interface

Verschillen facade v.s. adapter

Facade:

- bestaan er reeds classes? Ja
- bestaat er reeds een interface waar we ons aan moeten houden? Nee (die maak je zelf)
- is een eenvoudigere interface gewenst? Ja

Adapter:

- bestaan er reeds classes? Ja
- bestaat er reeds een interface waar we ons aan moeten houden? Ja
- is een eenvoudigere interface gewenst? Nee

Voorbeeld

Blender, een 3d modelleringsprogramma. . .
Werking:

- bouw een 3d model
- smeer bitmaps tegen alle vlakken
- render het model

Blender: het probleem

Een groep slimmeriken is bezig met het bouwen van een nieuwe rendering engine...

- de engine bestaat uit een enorme hoeveelheid classes met daarin vele methodes
- men wil blender met deze engine laten werken
- de technici werken vrolijk verder aan hun engine en vragen de heren van Blender niets

Probleem: laat Blender met deze nieuwe engine werken

Blender: oplossing

Bouw een adapter:

- de adapter klasse heeft een pointer naar het rendering object in zich.
- methodes die vanuit Blender worden aangeroepen, worden door de adapter vertaald naar methodes die de engine begrijpt