PATTERNRESTJES

Veerle Ongenae, Wijnand Schepens

Patternrestjes

- GoF-restjes
- Minder gebruikt

Patternrestjes

- Bridge Pattern
- Builder Pattern
- Chain of Responsibility
- Flyweight
- Interpreter
- Mediator
- Memento
- Prototype
- Visitor

Bridge Pattern

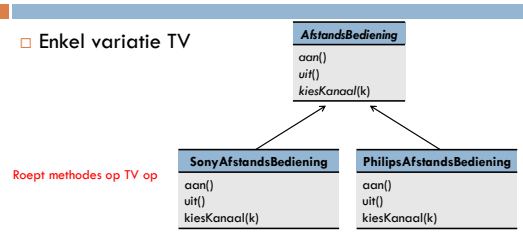
- Variëren
  - Implementaties
  - En abstracties

Bridge Pattern - voorbeeld

- Voorbeeld
  - Compleet nieuw type afstandsbediening voor TV's
    - Verschillende implementaties (voor elk merk een ander)
    - Zelfde abstractie (aan, uit, kanaal kiezen, …)
    - Compleet nieuw
      - Niet onmiddellijk goed
      - Vaak verbeterd worden (bv. op basis van gebruikerservaring)
        - Abstractie aanpassen
  - Variaties
    - TV's
    - Afstandsbediening

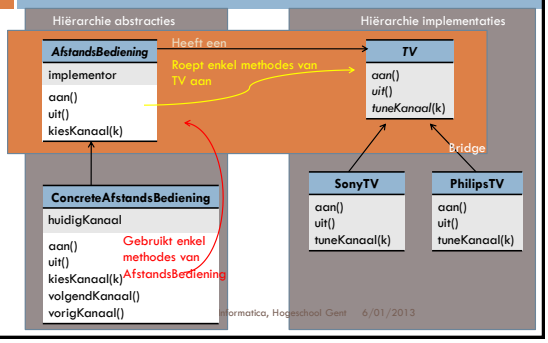
Voorbeeld

- Enkel variatie TV



Voorbeeld: variatie TV en afstandsbediening

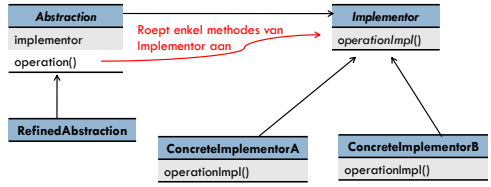
7



Bridge Pattern

8

- Ontkoppelt de implementatie van een abstractie, zodat deze twee onafhankelijk van elkaar kunnen veranderen



Voordelen Bridge

9

- Ontkoppelt implementatie van een bepaalde interface
- Abstractie en implementatie onafhankelijk van elkaar uitbreiden
- Veranderingen in de concrete klassen van de abstractiehiërarchie hebben geen invloed op de implementaties

Wanneer bridge gebruiken?

10

- Twee orthogonale (onafhankelijke) hierarchieën
- Bv. `MemoryMappedFile en DirectReadFile`
  zowel voor Linux als Windows?
  Met bridge vermijd je
  `MemoryMappedLinuxFile, MemoryMappedWindowsFile, DirectReadLinuxFile, DirectReadWindowsFile,`

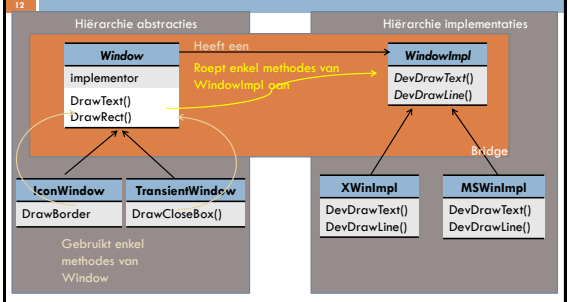
Gebruik en nadelen Bridge

11

- Grafische systemen op meerdere platformen
- Veranderen
  - Interface
  - En implementatie
- Verhoogt complexiteit

Derde voorbeeld Bridge

12



Lijkt op

13

- Strategy
  - Vergelijkbaar klassendiagram maar andere bedoeling
- Adapter
  - "Adapter makes things work after they're designed; Bridge makes them work before they are. [GoF, p219]"

Patternrestjes

14

- Bridge Pattern
- Builder Pattern
- Chain of Responsibility
- Flyweight
- Interpreter
- Mediator
- Memento
- Prototype
- Visitor

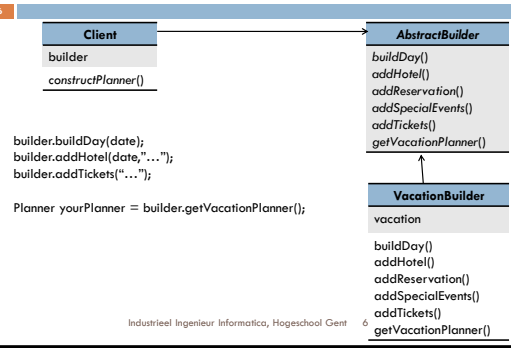
Builder Pattern

15

- Afschermen constructie ingewikkeld object
- Constructie kan in verschillende stappen
- Voorbeeld
  - Vakantieplanner
    - Hotel, tickets, restaurant, attracties, …
    - Meerdere dagen
    - Verschillende activiteiten
    - Flexibele datastructuur
    - Maken in verschillende stappen

Builder - voorbeeld

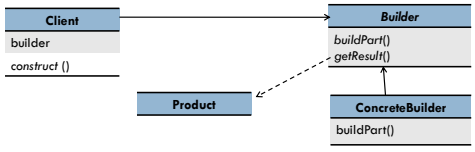
16



Builder Pattern

17

- De constructie van een ingewikkeld object van zijn weergave scheiden zodat hetzelfde constructieproces op verschillende representaties kan worden toegepast.



Voordelen Builder

18

- Schermt aanmaak complex object af
- Biedt mogelijkheid om object in meerdere stappen en wisselende processen aan te maken
- Verbergt interne representatie van het object
- Objectimplementaties kunnen wisselen
  - Client kent enkel abstracte interface
- Client wordt ook soms **Director** genoemd
  - Kan bepaalde *volgorde* van constructie opleggen

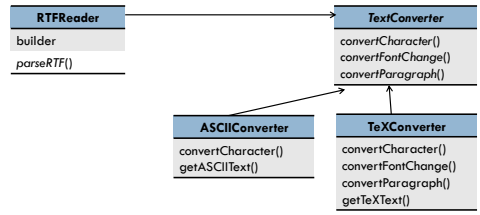
Gebruik en nadelen Builder

19

- Gebruikt voor samengestelde objecten
- Aanmaken object vereist meer domeinkennis dan bij een Factory
- Vergelijk met Abstract Factory, Prototype

Builder – tweede voorbeeld

20



Patternrestjes

21

- Bridge Pattern
- Builder Pattern
- Chain of Responsibility
- Flyweight
- Interpreter
- Mediator
- Memento
- Prototype
- Visitor

Chain of Responsibility

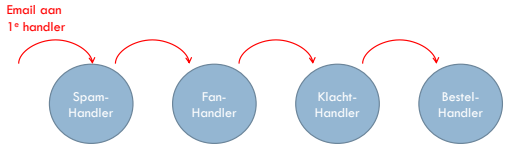
22

- Gebruik de Chain of Responsibility als je meer dan één object de kans wilt geven een verzoek af te handelen
- Voorbeeld
  - Massa's emails
    - Fanmail → CEO
    - Klachten → juridische dienst
    - Plaatsing nieuwe machines →verkoopsafdeling
    - Spam → verwijderd
  - Bestaande AI-detectors

Voorbeeld

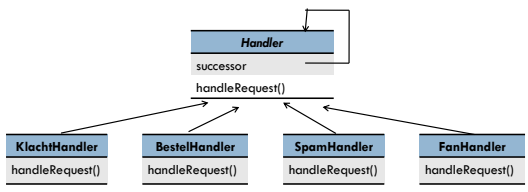
23

- Email → 1e handler → indien niet afgehandeld → 2e handler → …
- Email op einde van de keten → niet afgehandeld



Voorbeeld – klassenstructuur

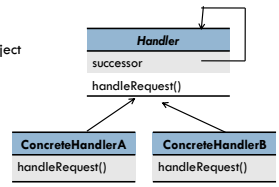
24



Chain of Responsibility

25

- Keten van objecten die een verzoek bekijken
- Elk object
  - Bekijkt verzoek
  - Twee mogelijkheden
    - Handelt af
    - Geeft door aan volgend object



Voordelen Chain of Responsibility

26

- Ontkoppelt zender en ontvanger verzoek
- Ontvangers eenvoudiger
- Dynamisch verantwoordelijkheden toevoegen en verwijderen
  - Onderdelen van de ketting veranderen

Gebruik en nadelen

27

- Muis- en toetsenbordevents af te handelen
- Uitvoering van het verzoek is niet gegarandeerd
  - Ketting verlaten zonder afgehandeld te zijn
- Gedrag at runtime bekijken of debuggen kan lastig zijn

Patternrestjes

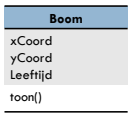
28

- Bridge Pattern
- Builder Pattern
- Chain of Responsibility
- Flyweight
- Interpreter
- Mediator
- Memento
- Prototype
- Visitor

Flyweight

29

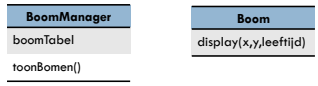
- Gebruik het Flyweight Pattern als één instantie van een klasse gebruikt kan worden om veel 'virtuele instanties' te leveren
- Voorbeeld
  - Applicatie landschapsontwerp
    - Bomen
    - Veel bomen → traag



Voorbeeld

30

- Duizenden Boom-objecten → één instantie van Boom
- Nieuwe situatie
  - BoomManager: toestand alle bomen
  - Één boom-object: hoe boom tekenen



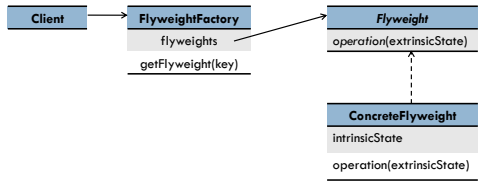
Flyweight

31

- Ondersteunt het gebruik van grote aantallen fijnmazige objecten op een efficiënte manier
- Twee types informatie
  - Toestandsloos: zelfde voor alle objecten (intrinsic state)
    - In Flyweight
  - Specifiek voor één object (extrinsic state)
    - In client-objecten

Flyweight

32



Voordelen, gebruik, nadelen

33

- Verminder aantal objecten at runtime, spaart geheugen
- Concentreert de toestand van veel virtuele objecten op één plaats
- Gebruik Flyweight als je veel instanties van één klasse hebt die allemaal op dezelfde manier beheerd kunnen worden
- Losse instanties van de klasse zijn niet in staat zich onafhankelijk te gedragen van de andere instanties

Patternrestjes

34

- Bridge Pattern
- Builder Pattern
- Chain of Responsibility
- Flyweight
- Interpreter
- Mediator
- Memento
- Prototype
- Visitor

Interpreter

35

- Om een interpreter voor een taal te bouwen
- Eenvoudige taal interpreteren

expression ::= <command> | <sequence> | <repetition>
sequence ::= <expression> ';' <expression>
command ::= right | quack | fly
repetition ::= while '(' <variable> ')' <expression>
variable ::= [A-Z,a-z]+

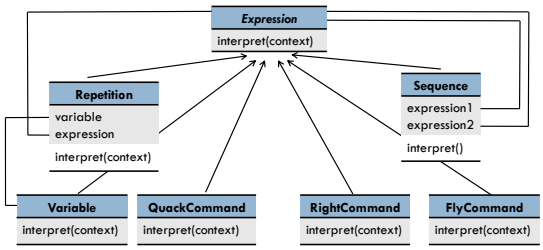
- Mogelijk script

```
right;
while (daylight) fly;
quack;
```


Voorbeeld

36

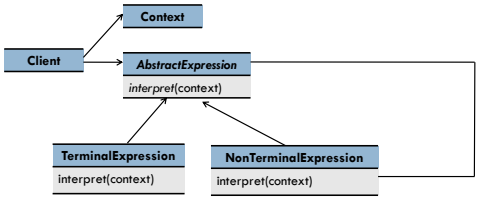
- Voor elke expressie interpret() aanroepen



Interpreter

37

- Om een interpreter van een taal te bouwen



Voordelen interpreter

38

- Elke grammaticale regel voorgesteld in klasse → makkelijk taal implementeren
- Taal makkelijk te wijzigen en uit te breiden
- Extra methodes aan klassen
  - Nieuw gedrag toevoegen
    - Verfijndere validatie
    - Mooier afdrukken

Gebruik en nadelen Interpreter

39

- Gebruik Interpreter voor een eenvoudige taal
- Geschikt voor eenvoudige grammatica
  - Eenvoud belangrijker dan efficiëntie
- Script- en programmeertalen
- Lastig
  - Veel grammaticale regels
  - Beter parser/compiler gebruiken

Patternrestjes

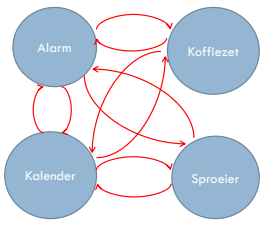
40

- Bridge Pattern
- Builder Pattern
- Chain of Responsibility
- Flyweight
- Interpreter
- Mediator
- Memento
- Prototype
- Visitor

Mediator Pattern

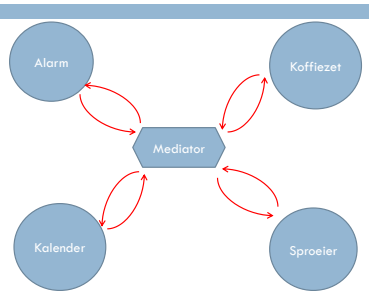
41

- Complexe communicatie tussen objecten op één plaats onderbrengen
- Voorbeeld
  - Wekker start koffiezet
  - Sproeier uit kwartier voor douche
  - Wekker vroeger op dinsdag
  - Geen koffie in weekend
- Probleem
  - Bijhouden relaties tussen objecten
  - Welke regels in welk object



Mediator toevoegen

42



Mediator toevoegen

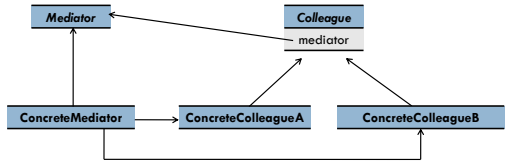
43

- Apparaten vereenvoudigd
  - Vertellen mediator wanneer hun toestand verandert
  - Reageren op verzoeken mediator
- De apparaten zijn ontkoppeld ↔ voorheen sterk gekoppeld
- Mediator bevat besturingslogica van het hele systeem
  - Nieuwe regel of nieuw apparaat
    - Enkel Mediator toevoegen

Mediator Pattern

44

- Een object definiëren dat insluit hoe een reeks objecten met elkaar omgaan. Mediators bevorderen losse koppeling door objecten ervan te weerhouden expliciet naar elkaar te verwijzen en ze maken het mogelijk de interacties tussen de objecten onafhankelijk van elkaar te variëren.



Voordelen Mediator

45

- Verhoogt herbruikbaarheid objecten door ze van het systeem los te koppelen
- Vereenvoudigt het onderhoud van het systeem door de logica op één plek onder te brengen
- Vereenvoudigt en vermindert de verschillende berichten die tussen objecten in het systeem verzonden worden

Gebruik en nadelen

46

- Gewoonlijk gebruikt om GUI-componenten te coördineren
- Zonder goed ontwerp kan het Mediator-object zelf erg ingewikkeld worden

Patternrestjes

47

- Bridge Pattern
- Builder Pattern
- Chain of Responsibility
- Flyweight
- Interpreter
- Mediator
- Memento
- Prototype
- Visitor

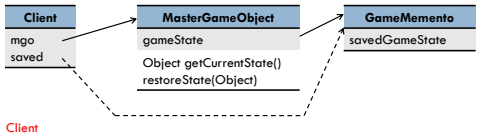
Memento

48

- Om een object terug te brengen naar een van zijn vorige toestanden
  - Bv. bij een undo
- Computerspel met verschillende levels
  - Voortgang in spel opslaan
  - Niet telkens opnieuw beginnen
  - Reeds verworven levels bewaren
- Doel Memento
  - Toestand hoofdobject systeem bewaren
  - Afscherming hoofdobject
    - Scheiden toestand van hoofdobject

Voorbeeld

49



Client

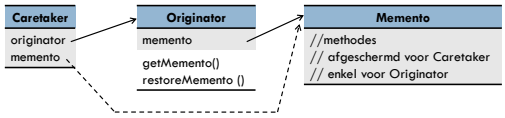
```
Object saved = mgo.getCurrentState();
…
mgo.restoreState(saved);
```

Geen toegang tot gegevens Memento-object

Memento Pattern

50

- De interne status van een object bewaren en extern maken zodat het object later in die status kan hersteld worden en toch de status af te schermen
- Ook Token genoemd



Voordelen Memento

51

- Toestand buiten het hoofdobject bewaren
  - Groter cohesie
- Gegevens hoofdobject afschermen
- Makkelijk vorige toestand herstellen

Gebruik en nadelen

52

- Gebruik om toestand te bewaren
- Toestand bewaren en herstellen kan veel tijd kosten
  - Eventueel toestand incrementeel bewaren (enkel veranderingen)
- In java kan je serialisatie overwegen

Patternrestjes

53

- Bridge Pattern
- Builder Pattern
- Chain of Responsibility
- Flyweight
- Interpreter
- Mediator
- Memento
- Prototype
- Visitor

Prototype

54

- Gebruik Prototype als het maken van een instantie van een gegeven klasse duur of complex is
- Nieuwe objecten worden aangemaakt door te kopiëren
- Voorbeeld computerspel
  - Veel monsters
  - Aangepast aan omgeving
  - Spelers kunnen eigen monsters aanmaken

Voorbeeld

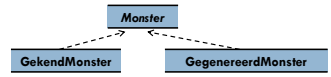
55

- Veel verschillende monsters afhankelijk van omgeving
- Losse koppeling
  - Aanmaken monster
  - Details van het aanmaken
    - Toestand
    - …

Voorbeeld - klassenstructuur

56

- Maker maakt aan, maar kent geen details
- Het eigenlijke aanmaken gebeurt door te kopiëren



MonsterMaker

```
maakRandomMonster() {
    Monster m = monsterRegister.getMonster();
}
```

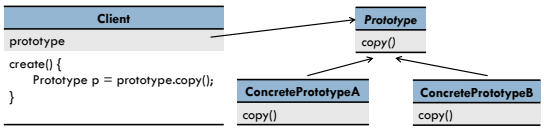
MonsterRegister

```
Monster getMonster() {
    …
    return juisteMonster.clone();
}
```


Prototype Pattern

57

- Een instantie van een prototype specifiëren van de soorten objecten die gemaakt moeten worden. Het creëren van nieuwe objecten gebeurt door het prototype te kopiëren



Kopiëren in Java: clone of serialisatie/deserialisatie (diepe kopie)

Voordelen Prototype

58

- Verbergt voor client ingewikkelde details van het maken van een nieuwe instantie
- Geeft de client de mogelijkheid objecten te maken waarvan het type onbekend is
- Soms is kopiëren efficiënter dan een nieuw object maken

Gebruik en nadelen Prototype

59

- Systeem veel nieuwe objecten aanmaken van veel verschillende typen uit een complexe klassenhiërarchie
  - Editor voor muziekpartituren
- Kopiëren kan ingewikkeld zijn

Patternrestjes

60

- Bridge Pattern
- Builder Pattern
- Chain of Responsibility
- Flyweight
- Interpreter
- Mediator
- Memento
- Prototype
- Visitor

Visitor Pattern

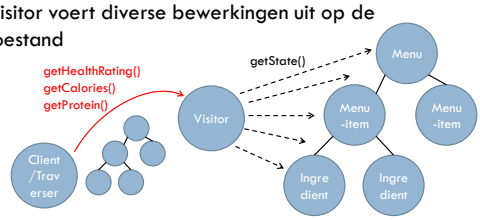
61

- Gebruik het Visitor Pattern als je mogelijkheden wilt toevoegen aan een samenstelling van objecten en afscherming niet belangrijk is
- Voorbeeld
  - Menu's en menu-items
  - Informatie over voedingswaarde toevoegen
    - getHealthRating()
    - getCalories()
    - getProtein()

Visitor - voorbeeld

62

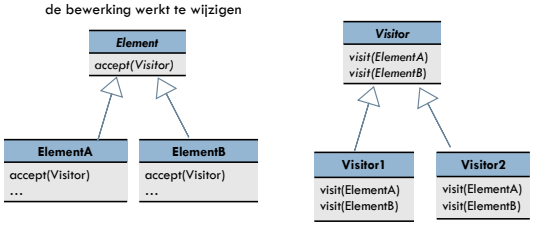
- Visitor bezoekt elk element van de samenstelling
  - Rondgeleid door Traverser (Client)
- Visitor voert diverse bewerkingen uit op de toestand



Visitor Pattern

63

- Voorstelling uit te voeren bewerking op elementen van een objectstructuur
- Nieuwe bewerking definiëren zonder de klassen van de elementen waarop de bewerking werkt te wijzigen



Vb.: zie wikipedia

64

- Double dispatch gesimuleerd door 2 x single dispatch
- Combinatie met Composite Pattern: samengesteld object laat visitor zichzelf bezoeken en geeft visitors door aan kinderen

Voordelen Visitor

65

- Bewerkingen toevoegen aan een samenstelling zonder de structuur van de samenstelling te veranderen
- Nieuwe bewerkingen toevoegen is relatief makkelijk
- De code voor de bewerkingen (die de Visitor) uitvoert is geconcentreerd op één plek

Gebruik en nadelen Visitor

66

- Voorbeeld gebruik
  - Compiler
    - Knopen: opdrachten
    - Visitor: code genereren, type controle, …
- Afscherming klassen uit samenstelling wordt teniet gedaan
- Wijzigingen in structuur samenstelling leveren meer problemen