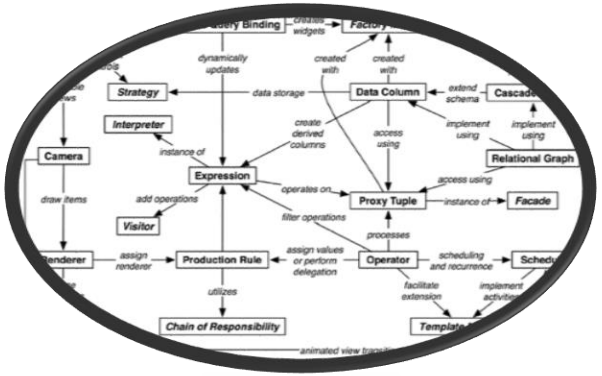
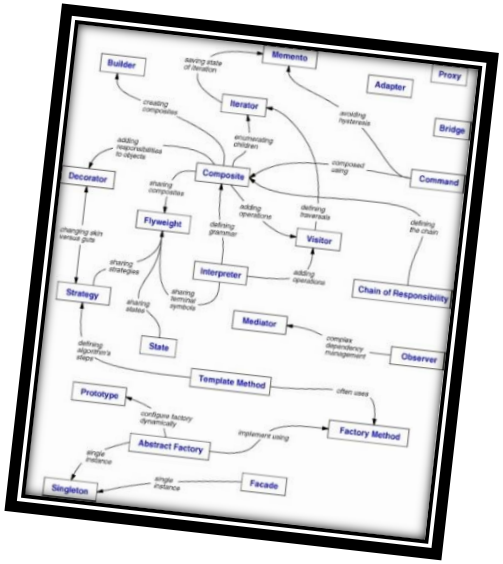
Design Patterns

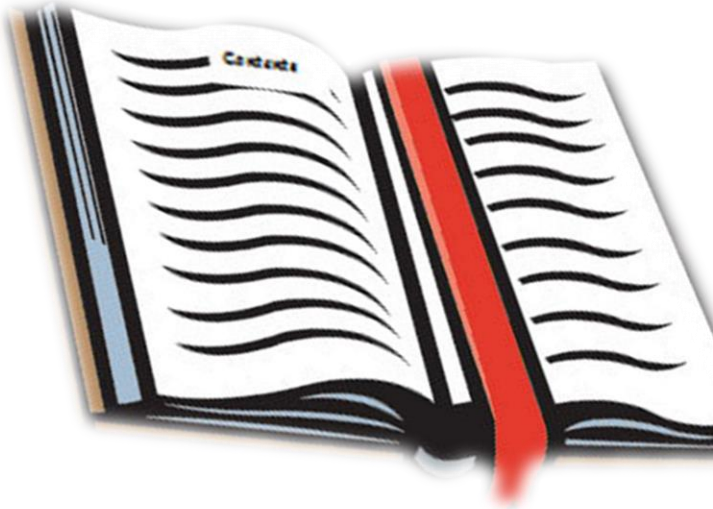


Algemene en herbruikbare oplossingen op veelvoorkomende problemen in software design



Inhoud

- OO-Principes: 10 streefdoelen
- Wat is een Design Pattern?
- Waarom Design Patterns gebruiken?
- Types van Design Patterns
  - Creational patterns
  - Structural patterns
  - Behavioral patterns
- Andere Patterns



OO-Principes: 10 streefdoelen

1. Isoleer wat verandert
2. Prefereer compositie boven overerving
3. Programmeer naar een Interface en niet naar een implementatie
4. Streef naar een ontwerp met een zwakke koppeling tussen inter-agerende objecten
5. Klassen moeten open zijn voor uitbreiding, maar gesloten voor modificaties
6. Wees afhankelijk van abstracties, niet van concrete klassen
7. Spreek alleen met vrienden
8. Bel ons niet, we bellen jou
9. Een klasse dient maar één reden te hebben om te veranderen
10. Delegeer verantwoordelijkheden en streef naar ‘High cohesion’

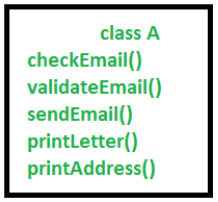


Fig: Low cohesion

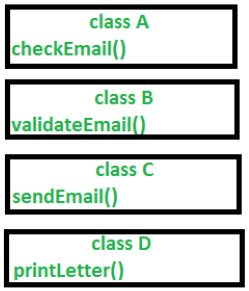


Fig: High cohesion

Enkele Patterns in c#.NET

- Iterator pattern in `foreach` loops in C#
- Observer pattern – events en event handlers
- Adapter pattern wordt gebruikt in ADO.NET
- Decorator: `CryptoStream` decoreert `Stream`
- Command in WPF wordt de aanvraag om een method aan te roepen geëncapsuleerd
- Façade pattern wordt in vele Win32 API gebaseerde classen gebruikt om de Win32 complexiteit te verbergen
- Chain of Responsibility is gelijkaardig aan exceptions
- String.Empty is een Null Object

Wat is een Design Pattern?

- Naam, Probleem, Oplossing en gevolgen

Wat zijn Design Patterns?

- Algemene en herbruikbare oplossingen voor veel voorkomende problemen in software design
  - Problem/solution pairs binnen een gegeven context
- Is geen afgewerkte oplossing
- Een template of recept om bepaalde problemen op te lossen
- Met namen die gebruikt worden om deze te kunnen identificeren

Wat zijn Design Patterns? (2)

- Patterns behandelen
  - Applicatie en systeem design
  - Abstracties bovenop code
  - Relaties tussen classes of andere soort collaboratoren
  - Problemen die reeds al zijn opgelost
- Patterns zijn geen..
  - Algoritmen
  - Frameworks
  - Patterns zijn niet exclusief voor OOPS
  - Specifieke implementaties van classes

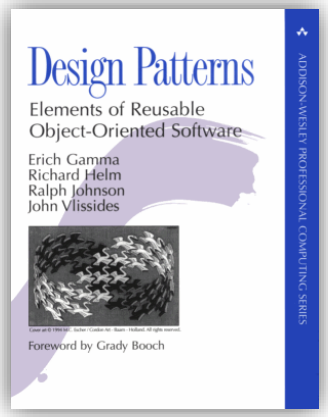
Oorspong van Design Patterns

"Each pattern describes a problem which occurs over and over again in our environment and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it in the same way twice".

- Christopher Alexander
  - architect
  - A Pattern Language: steden, gebouwen, Constructie, 1977
- Context:
  - City Planning en architectuur van gebouwen

Oorspong van Design Patterns (2)

- Terugkerende designs die succesvol waren
  - designs die vanuit de praktijk zijn ontstaan
- Ondersteuning van hogere niveau's van hergebruik van design een tamelijk grote uitdaging
- Beschreven in Gama, Helm, Johnson, Vlissides 1995 (vb, "Gang of Four Book")



- Gebaseerd op werk van Christopher Alexander
  - An Architect on building homes, buildings and towns

Beschrijving van Design Patterns

- Grafische notatie is meestal niet voldoende
- Voor hergebruik van design-beslissingen moeten de alternatieven en trade-offs worden beschouwd
- Concrete voorbeelden zijn ook belangrijk
- De ‘why’, ‘when’, en ‘how’ geven de context voor het gebruik van de geschikte design pattern

Elementen van Design Patterns

- Design patterns bevatten 4 essentiële elementen:
  - Pattern Naam
    - Increases vocabulary of designers
  - Probleem
    - Intentie, context, wanneer toe te passen
  - Oplossing
    - UML-achtige structuur en abstracte code
  - Gevolgen
    - Resultaten en trade-offs

Naam van Pattern

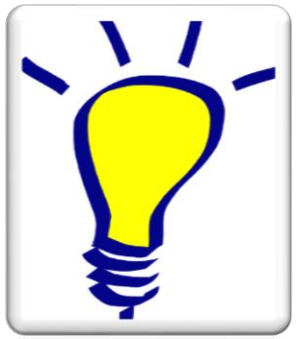
- Gebruikt voor het beschrijven van:
  - Een design probleem
  - De oplossingen
  - De gevolgen
- Design op hoger niveau van abstractie

Probleem

- Beschijft wanneer het pattern kan worden toegepast
- Uitleg over het probleem en zijn context
- Kan specifieke design problemen en/of object structuren beschrijven
- Kan een lijst van precondities die vervuld moeten zijn voordat het zin heeft om het pattern toe te passen

Oplossing

- Beschrijft de elementen die onderdeel uitmaken van
  - Het Design
  - De relaties
  - De verantwoordelijkheden
  - Collaboraties



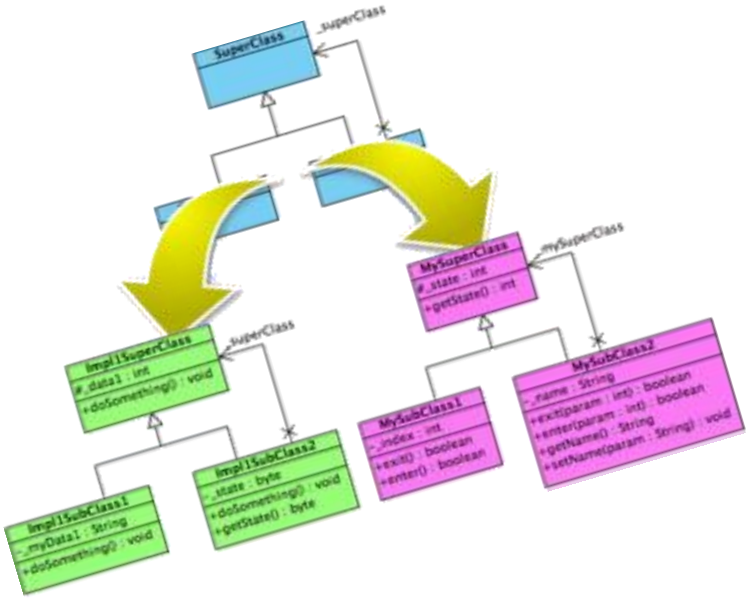
- Beschrijft geen specifieke concrete implementatie
  - Abstracte beschrijving van design problemen en hoe het pattern deze oplost

Gevolgen

- Resultaten en trade-offs bij toepassing van het pattern
- Essentieel voor:
  - Evaluatie van alternatieven voor design
  - Begrip over kosten
  - Begrip over voordelen

- Beschrijft de impact van een pattern op een systeem:
  - Flexibiliteit, Uitbreidbaarheid, Portabiliteit

Waarom Design Patterns toepassen?



Voordelen van Design Patterns

- Design patterns volgen geeft garantie voor hergebruik van software architectuur op grote schaal
- Geeft documentatie over de werking van systemen
- Patterns zijn opgebouwd vanuit expertise en design trade-offs
- Patterns verbeteren de communicatie naar en tussen developers (gedeelde taal)
- Pattern naamgevingen vormen een gedeelde vocabulair
- Patterns vergemakkelijken de overgang naar OO technologie

Wanneer Patterns toepassen?

- Bij oplossingen voor problemen die een variant zijn van een bepaald pattern
  - Indien het probleem zich in één enkele context voordoet, is het geen noodzaak voor gebruik van pattern
- Bij oplossingen die meerdere stappen vereisen:
  - Niet alle problemen hebben alle stappen nodig
  - Patterns kunnen ‘overkill’ zijn indien de oplossing een eenvoudige reeks van instructies is.
- Gebruik geen patterns indien het niet vereist is
  - Overdesign is evil!

Nadelen van Design Patterns

- Patterns leiden niet direct tot code-hergebruik
- Patterns kunnen misleidend eenvoudig lijken
- Teams kunnen lijden onder ‘pattern overload’
- Patterns worden meestal via ervaring en discussies gevalideerd i.p.v. geautomatiseerde testen
- Integratie van patterns in de software development process is intensieve arbeidsactiviteit
- Gebruik patterns enkel indien je ze goed verstaat

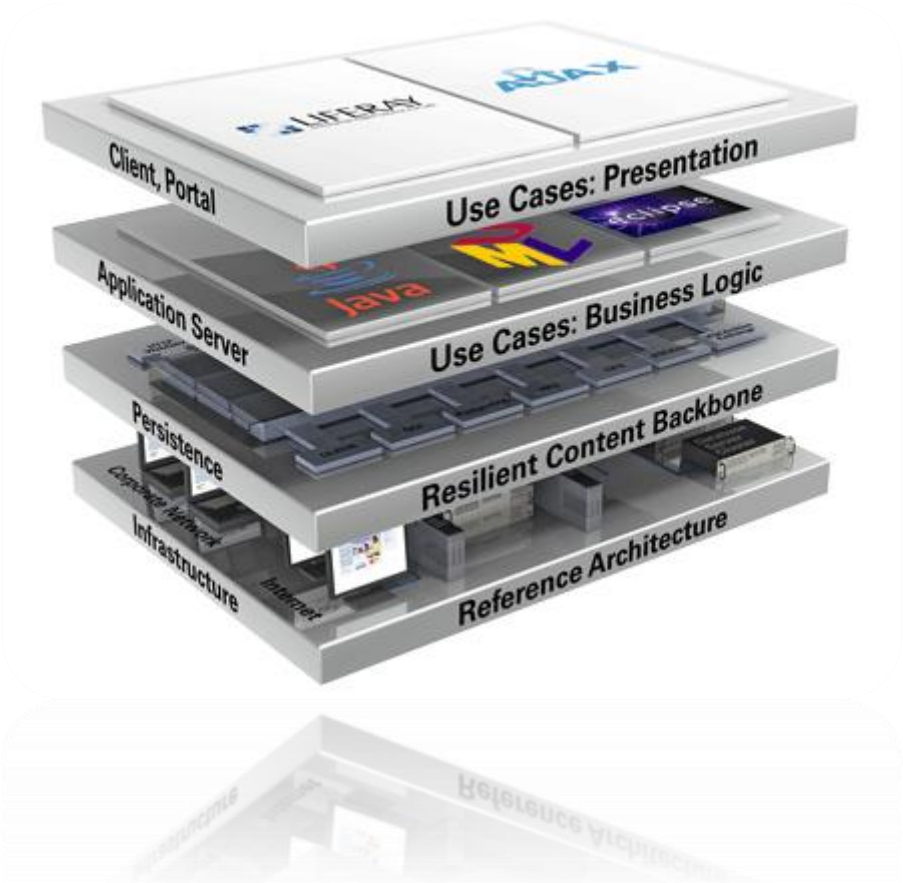
Kritiek op Design Patterns

- Legt focus op verkeerde oorzaak van probleem
  - design patterns kunnen een teken zijn van gebrek aan mogelijkheden van een bepaalde programmeertaal
- Gebrek aan formele basis
  - De studie van design patterns is voornamelijk ‘ad-hoc’ gebeurd
- Kan leiden tot inefficiënte oplossingen
- Geen significant verschil met andere abstracties

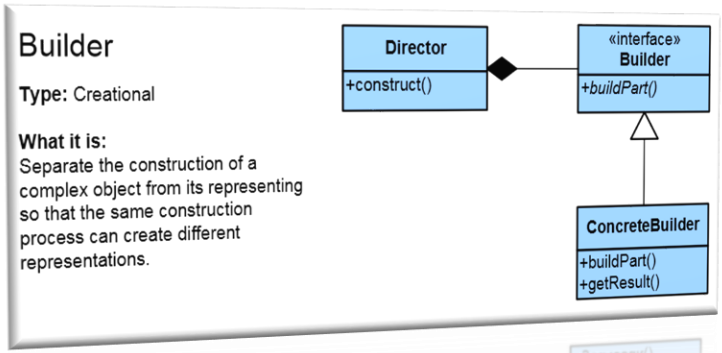
Drie hoofdcategorieën van Patterns

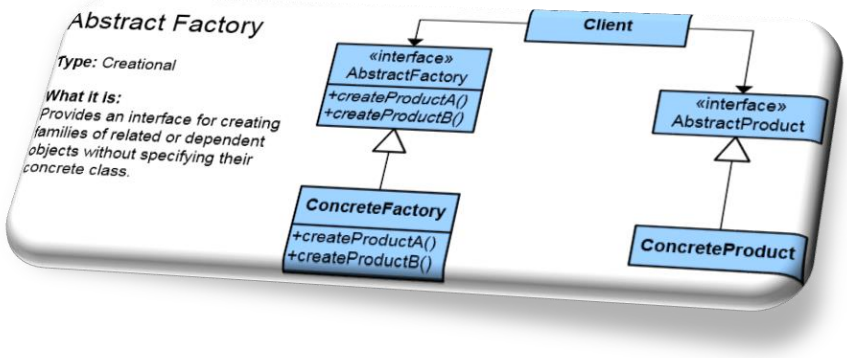
- Creationele patterns
  - Bieden voornamelijk oplossingen voor initializeren en configureren, van classes en objecten
- Structurele patterns
  - Beschrijven manieren om objecten te structureren om nieuwe functionaliteiten te kunnen implementeren
  - Een bepaalde compositie van interfaces, classes en/of objecten
- Gedrags-(Behavioral) patterns
  - Behandelen dynamische interacties tussen een structuur van classen en objecten
  - Hoe de verantwoordelijkheden worden verdeeld

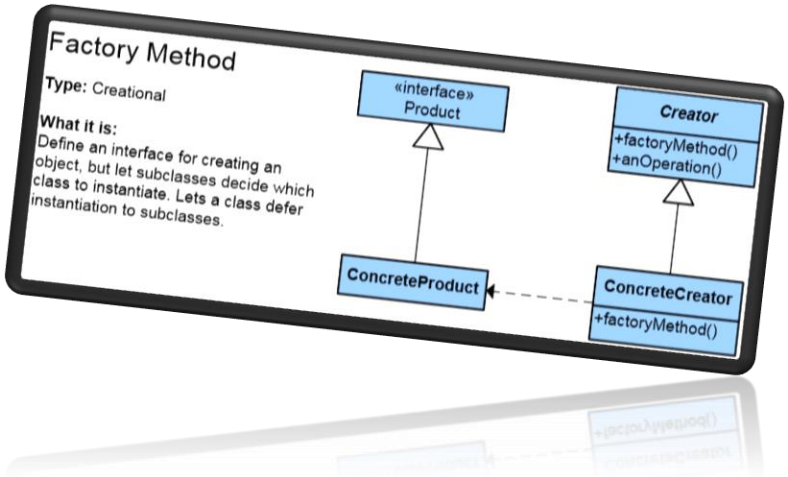
Architectuur (Structurele patterns)

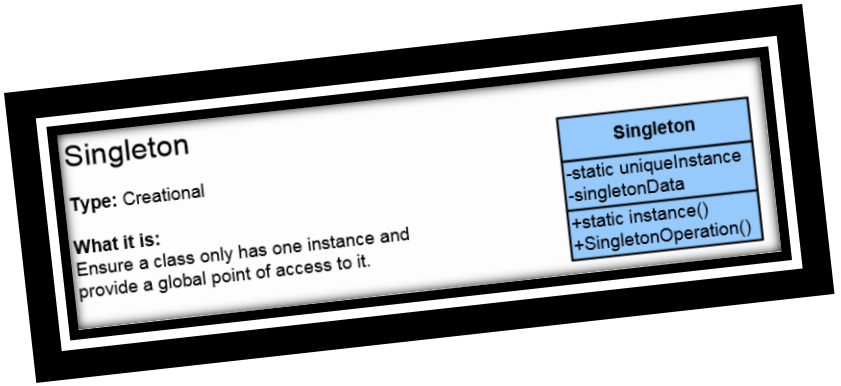


Creationele Patterns

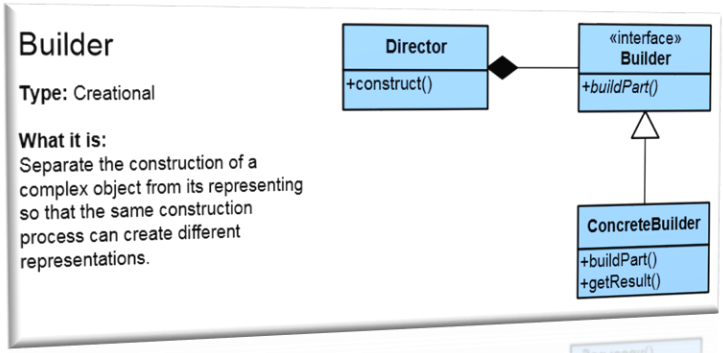


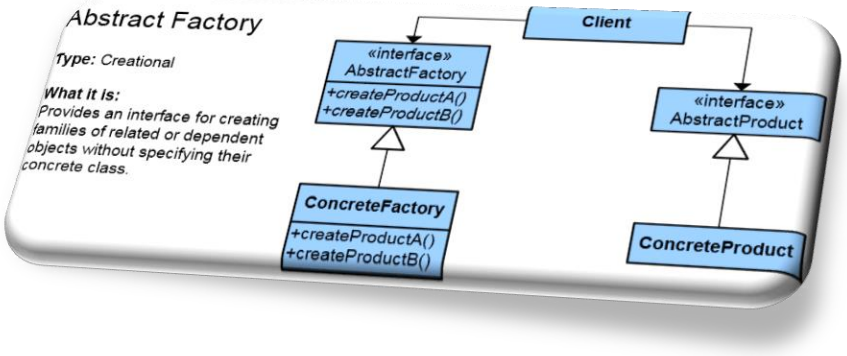


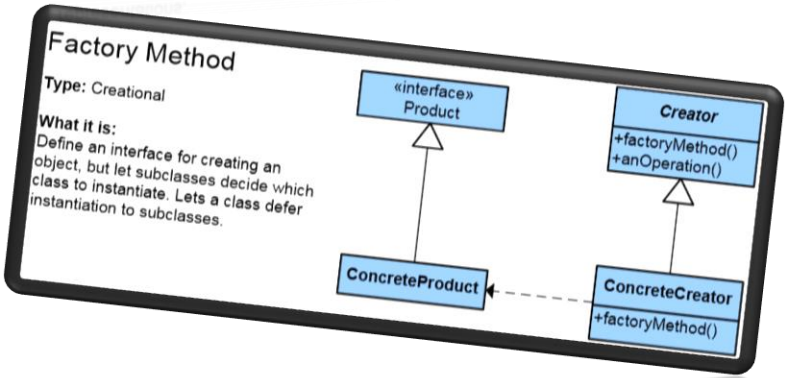


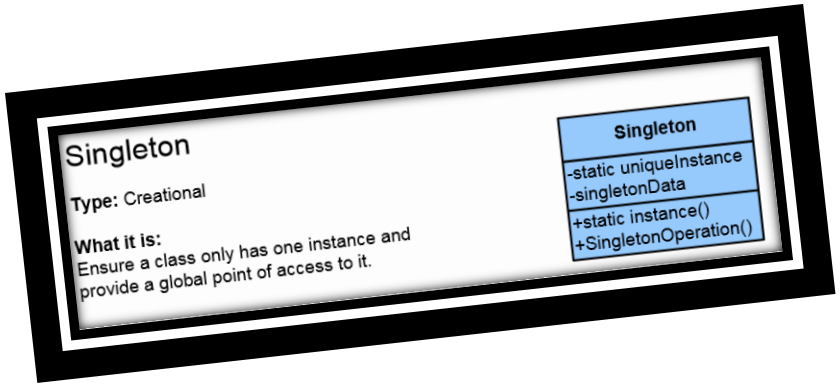


Creational Patterns









Creationele Patterns

- te maken met object creation mechanismen
- Om op bepaalde manier objecten te creëren op een voor de omgeving geschikte manier
- Bestaat uit 2 hoofd-richtlijnen:
  - Encapsulatie van kennis over welke concrete klassen het systeem zal gebruik maken.
  - Verbergen van hoe instanties van deze concrete klassen worden gecreëerd en gecombineerd

Enkele Creational Patterns

Singleton

Simple Factory (is eigenlijk géén pattern)

Factory method

Abstract Factory

The Builder Pattern

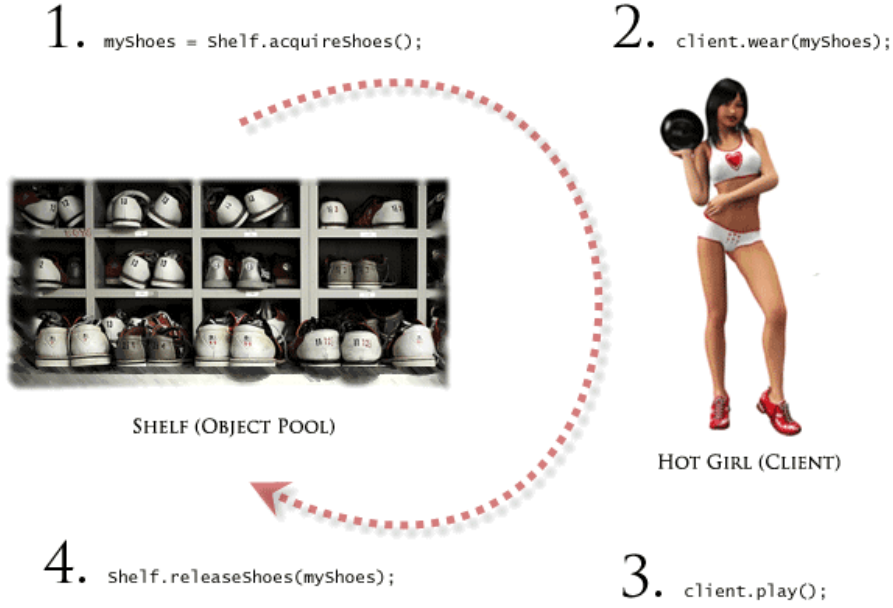
Prototype

Lazy initialization

Object pool

…

Creational Patterns…



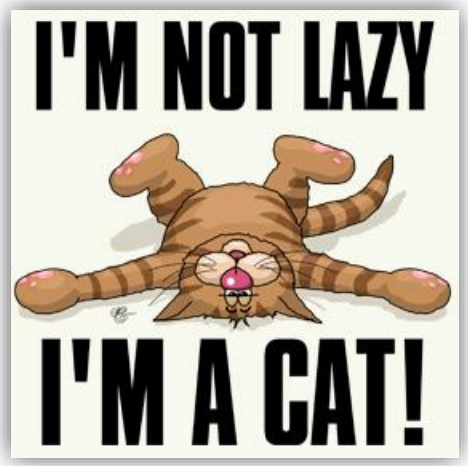
- Object Pool

  Vermijden van extensief verbruik en voorzien van vrijgeven van resources bij recyclage van ongebruikte objecten

- Lazy initialization

  Taktiek om de creatie van een object uit te stellen tot de eerste keer dat het nodig is

  Bv: een complexe berekening van een waarde, of een duur process pas uitvoeren wanneer het voor de eerste keer nodig is.



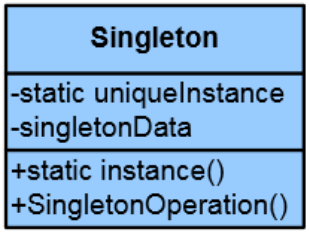
Singleton

- The Singleton class is een class waarvan slechts één enkele instantie kan worden aangemaakt en aangesproken
- Is géén globale variabele
- Mogelijke problemen:
  - Lazy loading
  - Thread-safety

Singleton

Type: Creational

What it is:
Ensure a class only has one instance and provide a global point of access to it.



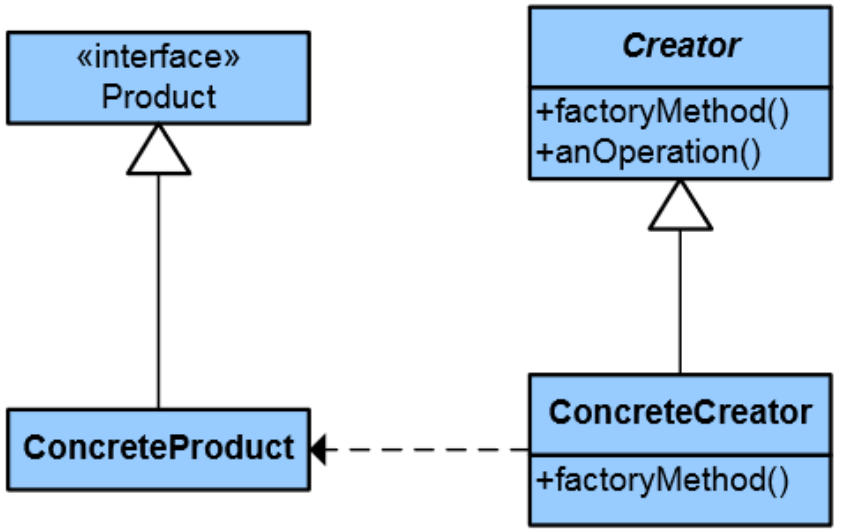
Factory Methode

- Objecten worden gecreëerd met speciale methode
- Produceert dus objecten (objecten-fabriek)
- Zorgt voor grotere flexibiliteit bij wijzigingen

Factory Method

Type: Creational

What it is:
Define an interface for creating an object, but let subclasses decide which class to instantiate. Lets a class defer instantiation to subclasses.



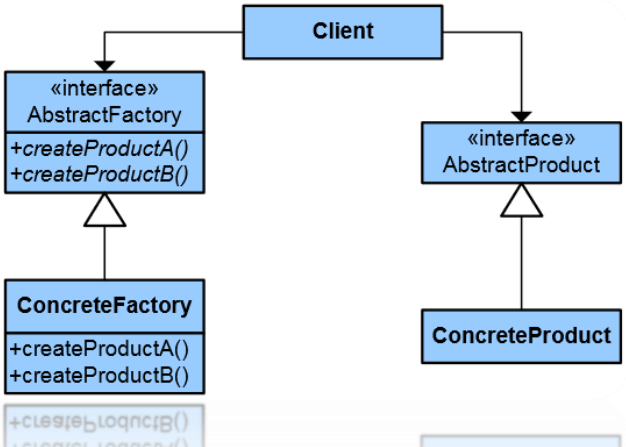
Abstract Factory

- Abstractie in object creatie
  - Creatie van familie van verwante objecten
- De Abstract Factory Pattern definieert een interface voor verwante objecten
  - Zonder kennis van de concrete classen
- Gebruikt in systemen die frequent wijzigen
- Flexibel mechanisme voor vervanging van reeks verwante objecten

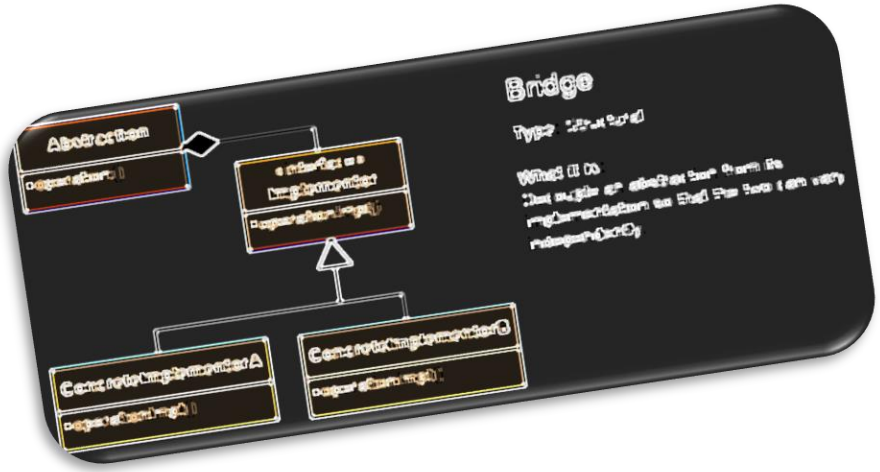
Abstract Factory

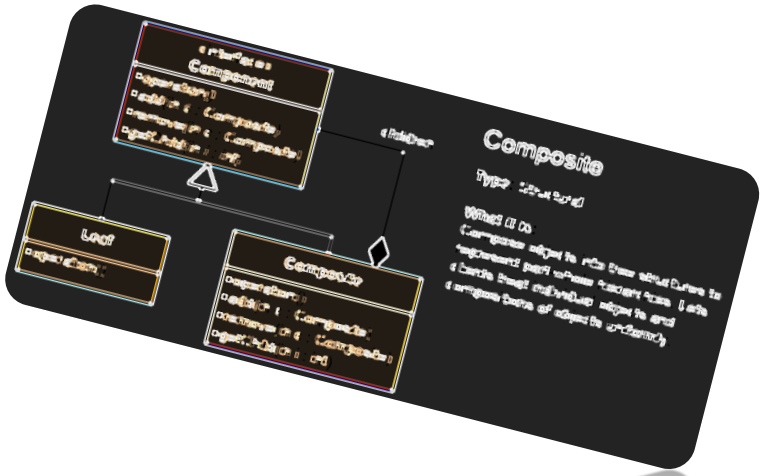
Type: Creational

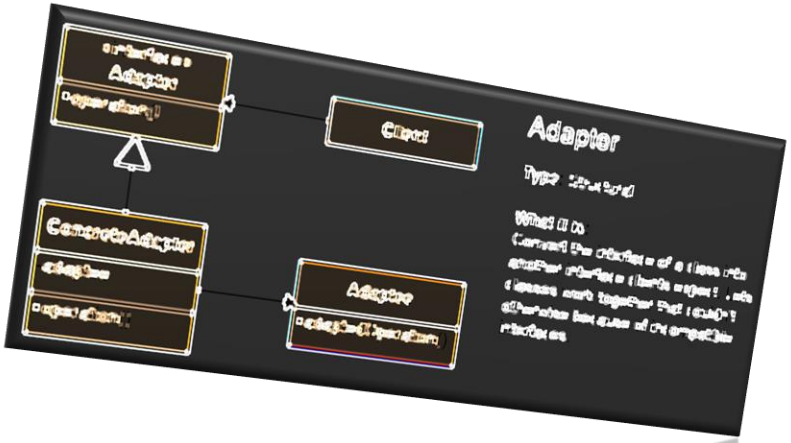
What it is:
Provides an interface for creating families of related or dependent objects without specifying their concrete class.

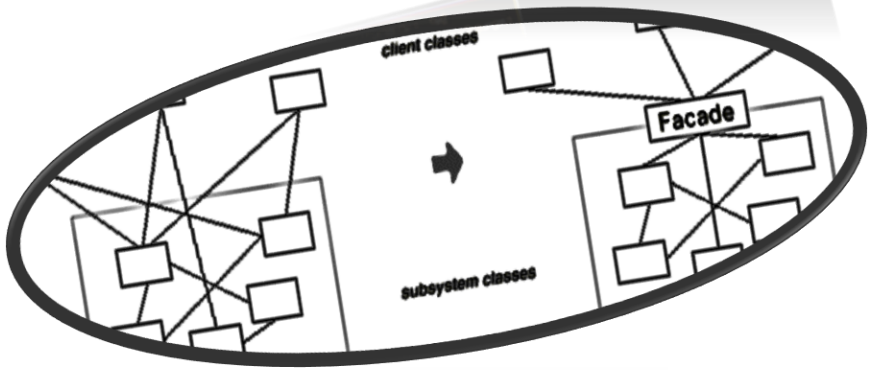


Structurele Patterns







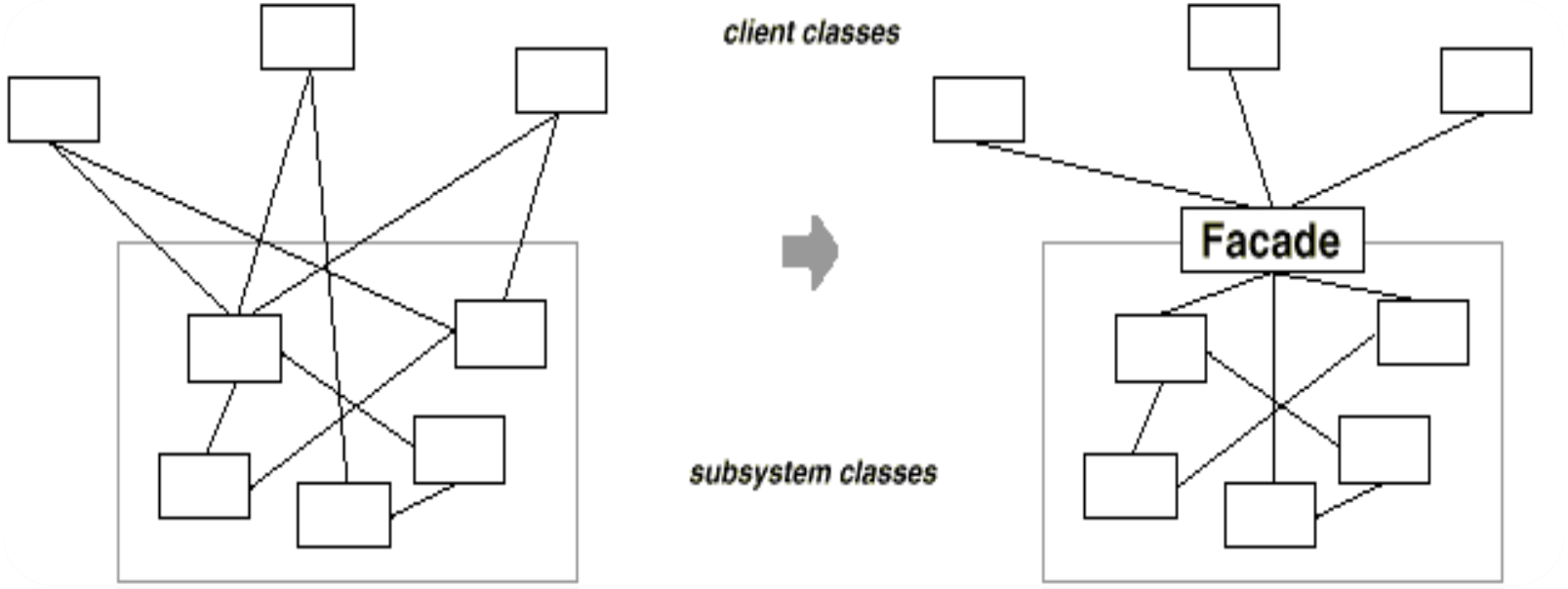


Structurele Patterns

- Beschrijven manieren om objecten te combineren om een nieuwe functionaliteit te implementeren
- Vereenvoudigen van ontwerp door een eenvoudige manier van relaties tussen entiteiten te leggen
- Gaat over Class en Object compositie
  - Structurele class-creatie patterns gebruiken overervingen
  - Structurele object-patterns definiëren manieren van om object- compositie om nieuwe functionaliteit te implementeren

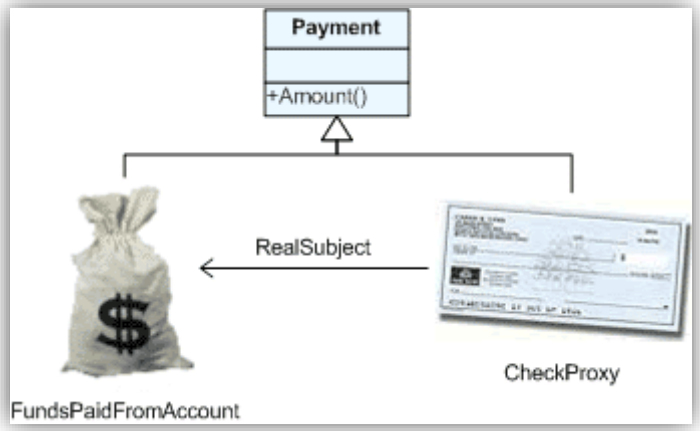
Facade Pattern

- Om een eenvoudiger interface te voorzien voor de klant app om te werken met een ingewikkelde classes door subsystemen te groeperen bv
- Façade pattern wordt bv gebruikt in Win32 API based classes om de Win32 complexiteit te verbergen



The Proxy Pattern

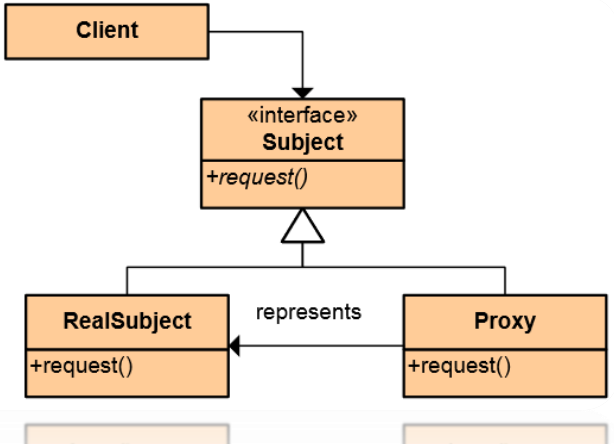
- Een object die een ander object representeert
  - Voorzien van een surrogaat of placeholder voor een ander object om **toegangscontrole** te voorzien naar het oject
  - Een extra niveau voor gedistribueerd of gecontroleerde toegang
  - Toevoegen van een “wrapper” en delegatie om het object te beschermen
- Bv: Web Service



Proxy

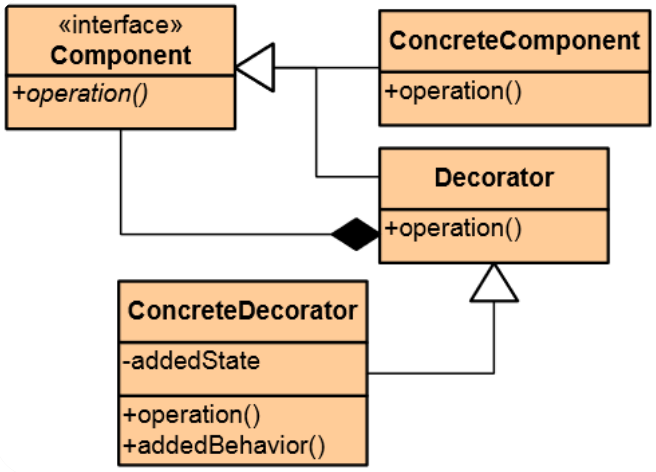
Type: Structural

What it is:
Provide a surrogate or placeholder for another object to control access to it.



Decorator Pattern

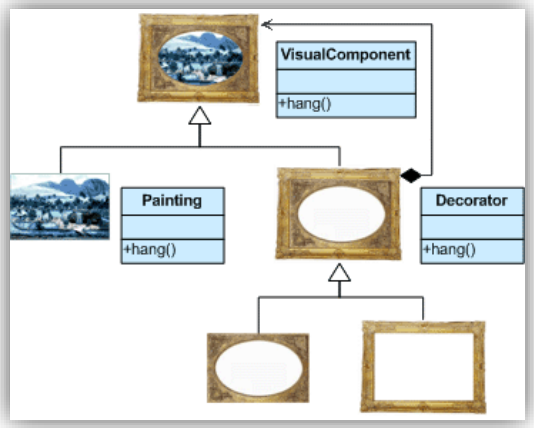
- Dynamisch toevoegen van verantwoordelijkheden aan een object
  - Wrappen van oorspronkelijke component
  - Alternatief voor inheritance (classes explosie vermijden)
- Bv In .NET: `CryptoStream` decoreert `Stream`



Decorator

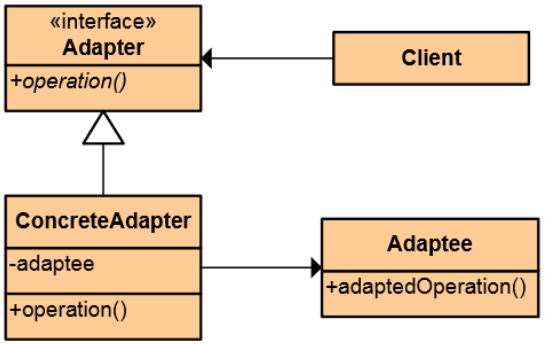
Type: Structural

What it is:
Attach additional responsibilities to an object dynamically. Provide a flexible alternative to sub-classing for extending functionality.



Adapter Pattern

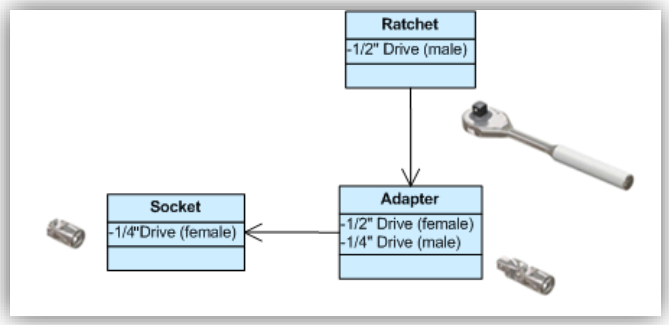
- Converteert de interface van een gegeven class naar een andere class die gebruikt wordt door de klant
  - Wrappen van bestaande class met interface
  - Impedance match an old component to a new system
- Zorgt dat classes met incompatible interfaces kunnen samenwerken incompatible interfaces



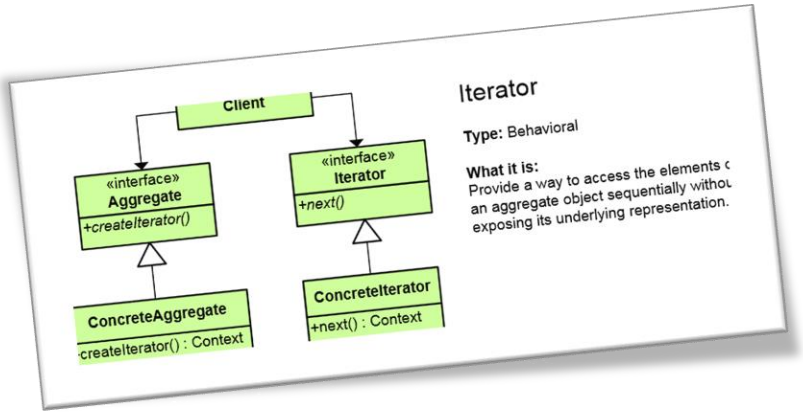
Adapter

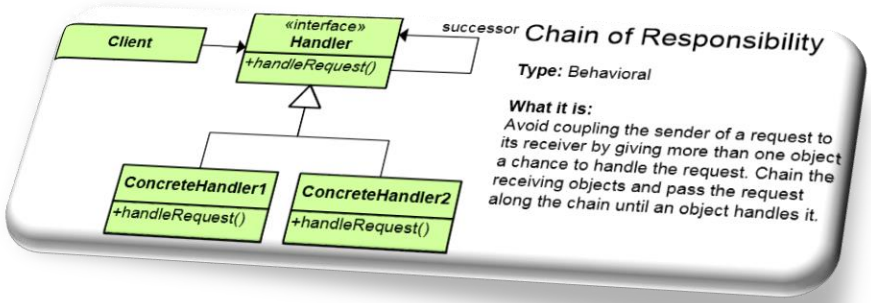
Type: Structural

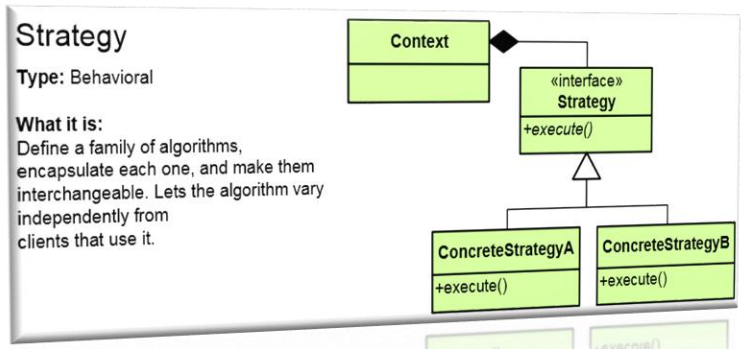
What it is:
Convert the interface of a class into another interface clients expect. Lets classes work together that couldn't otherwise because of incompatible interfaces.

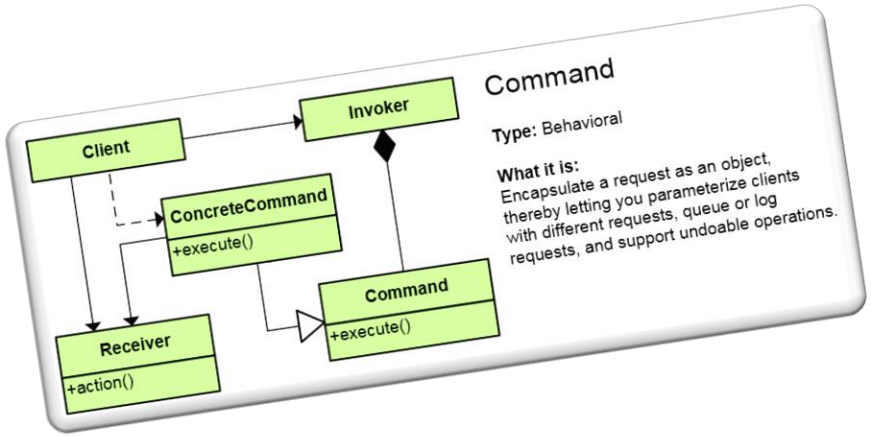


Behavioral(gedrags) Patterns







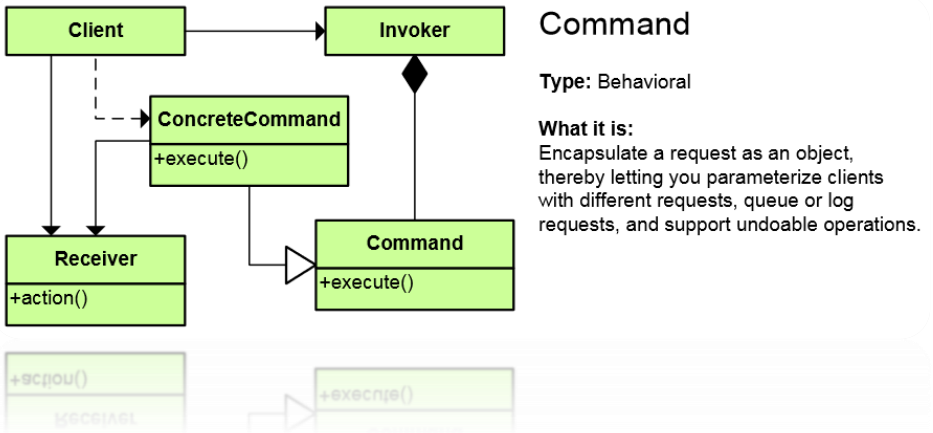


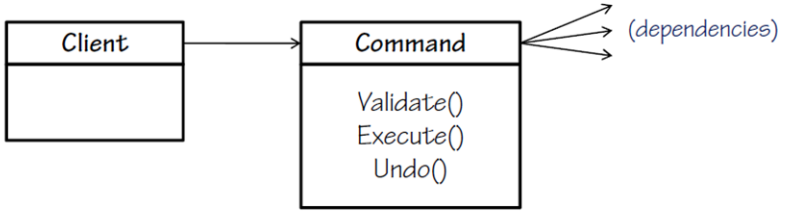
Behavioral Patterns

- Focus op communicatie (interactie) tussen objecten
  - Ofwel door verantwoordelijkheden toe te wijzen
  - Ofwel door gedrag in een object in te kapselen en aanvragen te delegeren
- Voorziet meer flexibiliteit bij communicatie tussen classes

Command Pattern

- Een object kapselt alle information in die nodig zijn om een methode op een later moment aan te roepen
  - Klanten kunnen geparametriseerd worden voor verschillende soorten aanvragen, aanvragen kunnen worden in wachtrij worden gezet of gelogd worden





- Command class in WPF kapselt de aanvragen in voor het aanroepen van methoden met of zonder parameters