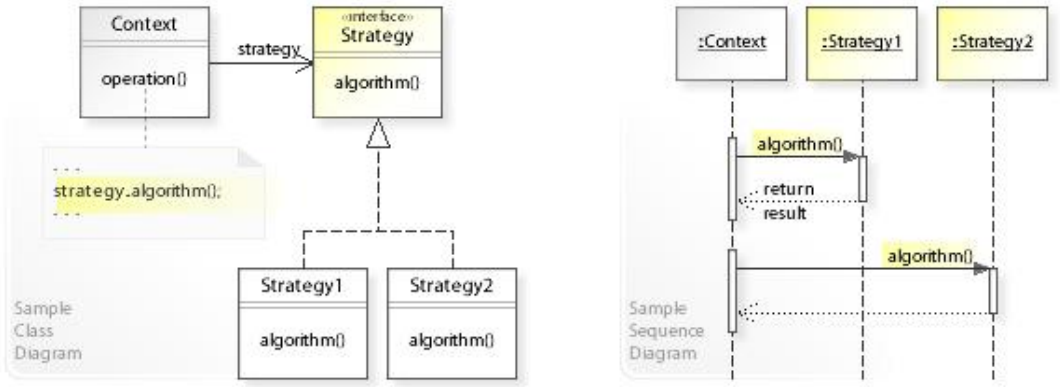
Notities 9 verschillende Game Design Patterns

Wayne Hofstra

Inhoud

Strategy Pattern

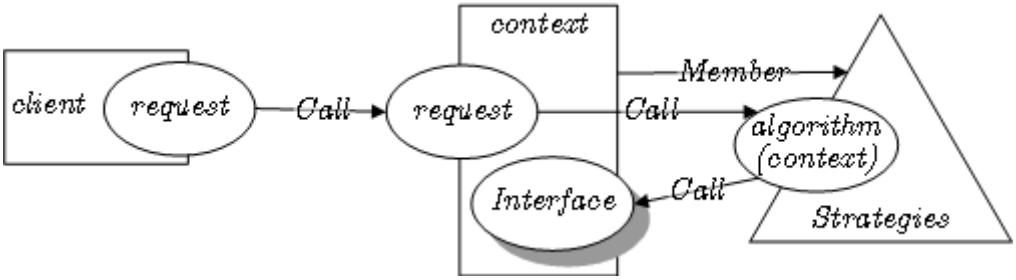
- Het staat ook wel bekend als de ‘Policy Pattern’.
- Het is mogelijk om een algoritme te selecteren gedurende runtime.
- De Strategy Pattern laat het algoritme variëren op basis van de client dat er gebruik van maakt.
- De beslissing om een bepaald algoritme te selecteren wordt door de gebruiker pas in runtime genomen.
- De ‘validation’-algoritmen zijn afgezonderd van het object dat gevalideerd moet worden. Deze algoritmen heten ‘strategies’.
- De ‘strategies’ kunnen worden gebruikt door objecten die gevalideerd moeten worden in andere delen van het systeem en zelfs andere systemen. Dat zonder de code te dupliceren.
- De Strategy Pattern bewaart referenties naar code in een data structuur en kan deze ook weer ophalen doormiddel van functie pointers, ‘first-class’ functies, classes of class-instanties.
- De ‘Context’ implementeert het algoritme niet direct, maar refereert naar de ‘Strategy’-interface om het voor hem te doen.
- De ‘Strategy1’- en ‘Strategy2’-classes implementeren de ‘Strategy’-interface.
- De Strategy Pattern maakt gebruik van ‘Composition over Inheritence’.
- Het gedrag van een class is afgescheiden doormiddel van interfaces, in plaats van overerving.
- De Pattern maakt gebruik van de ‘Open/Closed Principle’ (OCP).

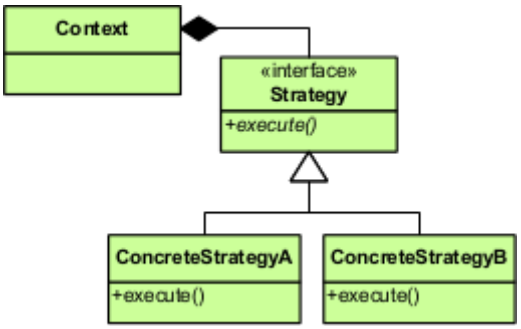


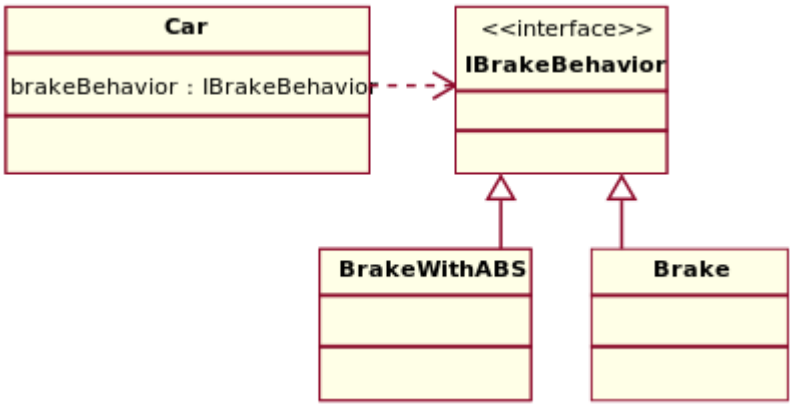
Voorbeelden voor het gebruik ervan:

- Een class dat bepaalde data goedkeurt maakt gebruik van de Strategy Pattern om een algoritme daarvoor te selecteren, afhankelijk van de type data, de herkomst ervan en de keuze van de gebruiker.

https://en.wikipedia.org/wiki/Strategy_pattern

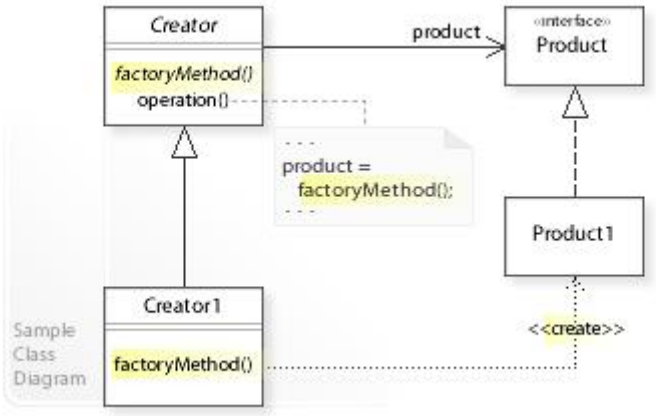

client
request
Call
context
request
Member
Call
algorithm
(context)
Interface
Call
Strategies

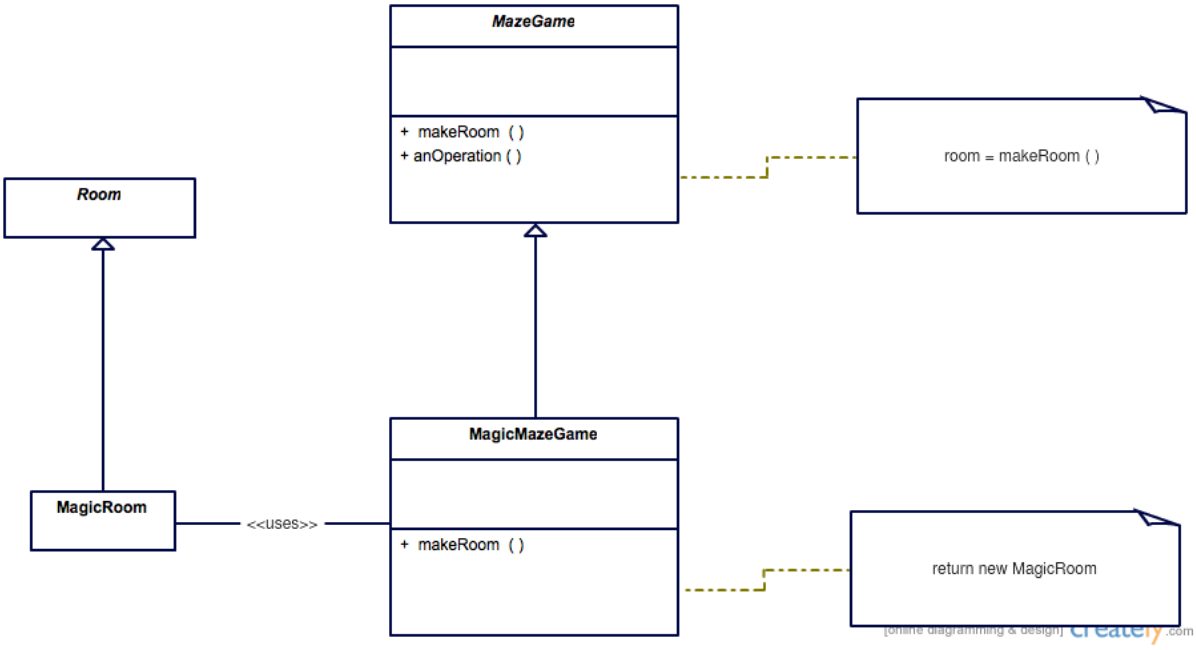

Context
«interface»
Strategy
+execute()
ConcreteStrategyA
+execute()
ConcreteStrategyB
+execute()


Car
brakeBehavior : IBrakeBehavior
<<interface>>
IBrakeBehavior
BrakeWithABS
Brake

Factory Pattern

- De ‘Factory Method Pattern’ is een ‘Creational Pattern’ om objecten op een manier te creëren zonder een exacte class aan te geven voor objecten die gecreëerd worden.
- Deze pattern wordt toegepast door een ‘Factory Method’ aan te roepen. Deze functie is gespecificeerd in de interface, geïmplementeerd in child classes of base classes.
- Het wordt gebruikt in plaats van een gewone class constructor om het construeren van objecten apart te houden van de objecten zelf. Dit valt onder ‘SOLID’ principes van programmeren.
- Classes kunnen geconstrueerd worden met een component van een bepaalde type dat nog niet is vastgesteld, maar wel is aangemaakt in een interface. Dit kan ook een dynamische type zijn.
- Constructie van subclasses waarvan de parent type nog niet vast is gesteld.
- De taak voor het instantiëren van zichzelf kan worden overgedragen van class naar subclass om zo de instantie van een parent class type te voorkomen.
- Dit wordt toegepast om te voorkomen dat de class een object aanmaakt die deze gebruikt. Het veranderen van het aangemaakt object betekent dat er in de parent class iets aangepast moet worden.





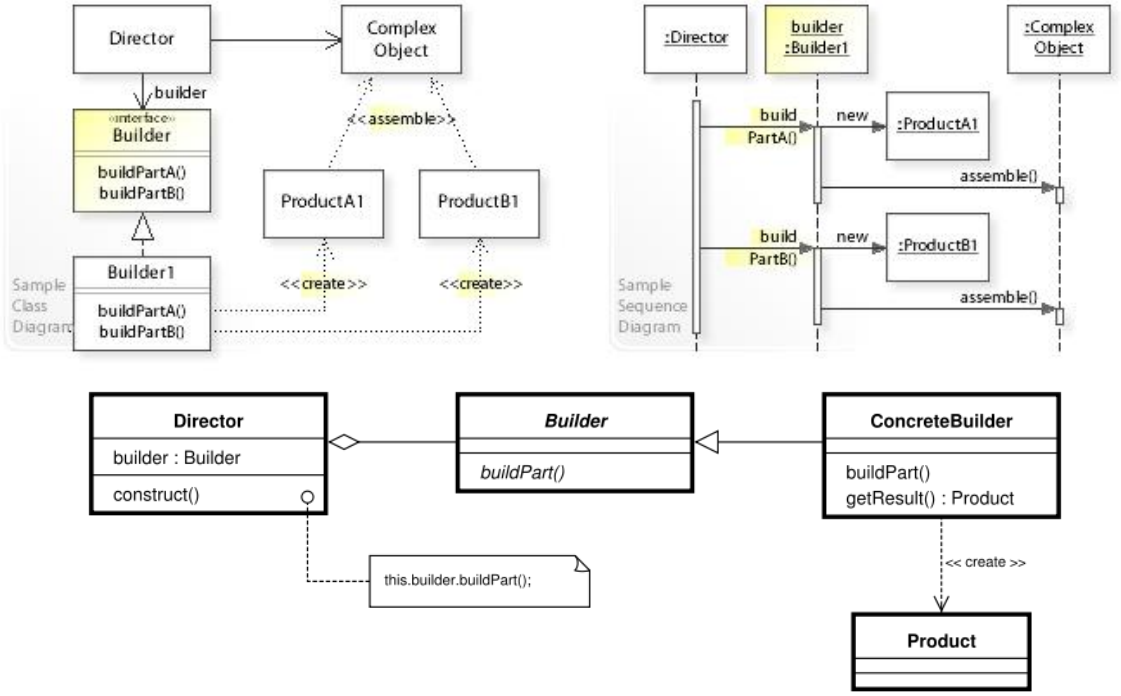
Problemen die de Factory Method Design Pattern kan oplossen:

- Het gebrek aan de beschikbaarheid van informatie voor een aangemaakt object.
- Het veelvoudig dupliceren van code.
- Een matige level van abstractie.

https://en.wikipedia.org/wiki/Factory_method_pattern

Builder Pattern

- De voornaamste doel voor het gebruik van deze Game Design Pattern is het verdelen van de constructie van een complex object en de representatie ervan.
- Hiermee kunnen classes met dezelfde constructie proces verschillende representaties krijgen.
- Het lost het probleem op dat ontstaat wanneer een specifieke representatie gebruikt wordt voor een class en deze dan aangepast moet worden zonder de class zelf te veranderen.
- Het 'Builder'-object scheidt de creatie en opbouw van een complex object af.
- Een class delegeert de creatie van een object naar een of meerdere 'Builder'-objecten voor één of meerdere representaties van een complex object.
- Hetzelfde constructie-proces kan verschillende representaties vormen.
- De 'Builder' is in dit geval een interface.
- De 'ConcreteBuilder' is diegene die de implementatie uitvoert voor de 'Builder'. Hij construeert en monteert om delen van een object te bouwen.



https://en.wikipedia.org/wiki/Builder_pattern

Object Pool Pattern

- Het is een ‘Creation Design Pattern’ dat gebruik maakt van een groep vooraf geïnitialiseerde objecten die klaar staan om gebruikt te worden. Deze worden in een ‘pool’ gehouden.
- De objecten die afkomstig zijn van de ‘pool’ veranderen niet van plaats in het geheugen en worden ook niet verwijderd na gebruik, maar weer terug gezet in de ‘pool’.
- Dit wordt gebruikt voor het verbeteren van performance.
- ‘Object lifetime’ wordt hierdoor gecompliceerder.
- Dit wordt toegepast bij het aanmaken van een groot aantal objecten die veel tijd vergen om aan te maken.
- De objecten uit de ‘pool’ kunnen worden hergebruikt.
- Bestaande referenties naar andere objecten worden ongedaan gemaakt als een object weer in de ‘pool’ terecht komt.
- Sommige object pools zijn qua capaciteit gelimiteerd aan een maximum aantal objecten.
- Bij het toepassen van een connectie naar een server, wordt de link naar de server niet gebroken als de connectie niet meer aanwezig is. De connectie wordt gehouden in de pool totdat er een aanvraag wordt gedaan naar een nieuwe connectie.
- Het aanmaken van objecten in de ‘pool’ neemt alleen veel tijd in beslag bij objecten die de tijd nodig hebben om aangemaakt te worden; zoals database- of socket connecties, threads en grafisch grote objecten, zoals bitmaps.
- In sommige gevallen kan het aanmaken van een object pool averechts werken in de zin dat het de performance achteruit haalt. Dat gebeurt alleen bij objecten die niet veel externe resources bevatten naast het in beslag nemen van geheugen.

Een lege ‘pool’ kan het volgende doen:

- De return is een error naar de client.
- De capaciteit van de pool wordt vergroot door het maximum aantal objecten te vergoten. Nieuwe objecten worden hierin verplaatst.
- In een ‘multithreaded environment’ kan de pool de client blokkeren tot dat er een object terug in de pool gestopt wordt.

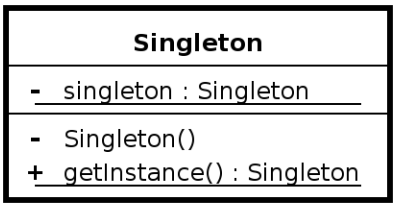
https://en.wikipedia.org/wiki/Object_pool_pattern

Singleton Pattern

- Het is een Software Design Pattern dat de aanmaak van een class beperkt tot één enkel instantie.
- Dit wordt gebruikt voor objecten waar maar één instantie van kan bestaan die bepaalde acties in een systeem kan coördineren.
- Bij gebruik van deze pattern op objecten waarbij het niet voordelig werkt, kan het onnodige beperkingen veroorzaken. Dat veroorzaakt een 'global state' in een applicatie.
- De voornaamste instantie van een class is makkelijker bereikbaar.
- Een class kan zijn eigen instantie beheersen.
- Het aantal instanties van een class wordt beperkt.
- Globale variabelen zijn toegankelijk.
- De Singleton Design Pattern verbergt de constructor van een class. Dit voorkomt dat de class geïnstantieerd wordt buiten de class.
- Het maakt gebruik van een public static operation die de instantie van een class teruggeeft.
- 'Façade'-objecten zijn singletons. Eén Façade verbergt de overige objecten.
- 'State'-objecten zijn singletons.
- De 'Factory Method Pattern' en 'Builder Pattern' kunnen gebruik maken van singletons.
- Van een singleton-class kun je niet overerven.
- Er kan moeilijk achterhaald worden of er bij een nieuwe instantie een bestaande instantie wordt teruggegeven of niet.
- Bij elke test kun je geen nieuwe versie gebruiken van een singleton-class.

Voordelen ten opzichte van globale variabelen:

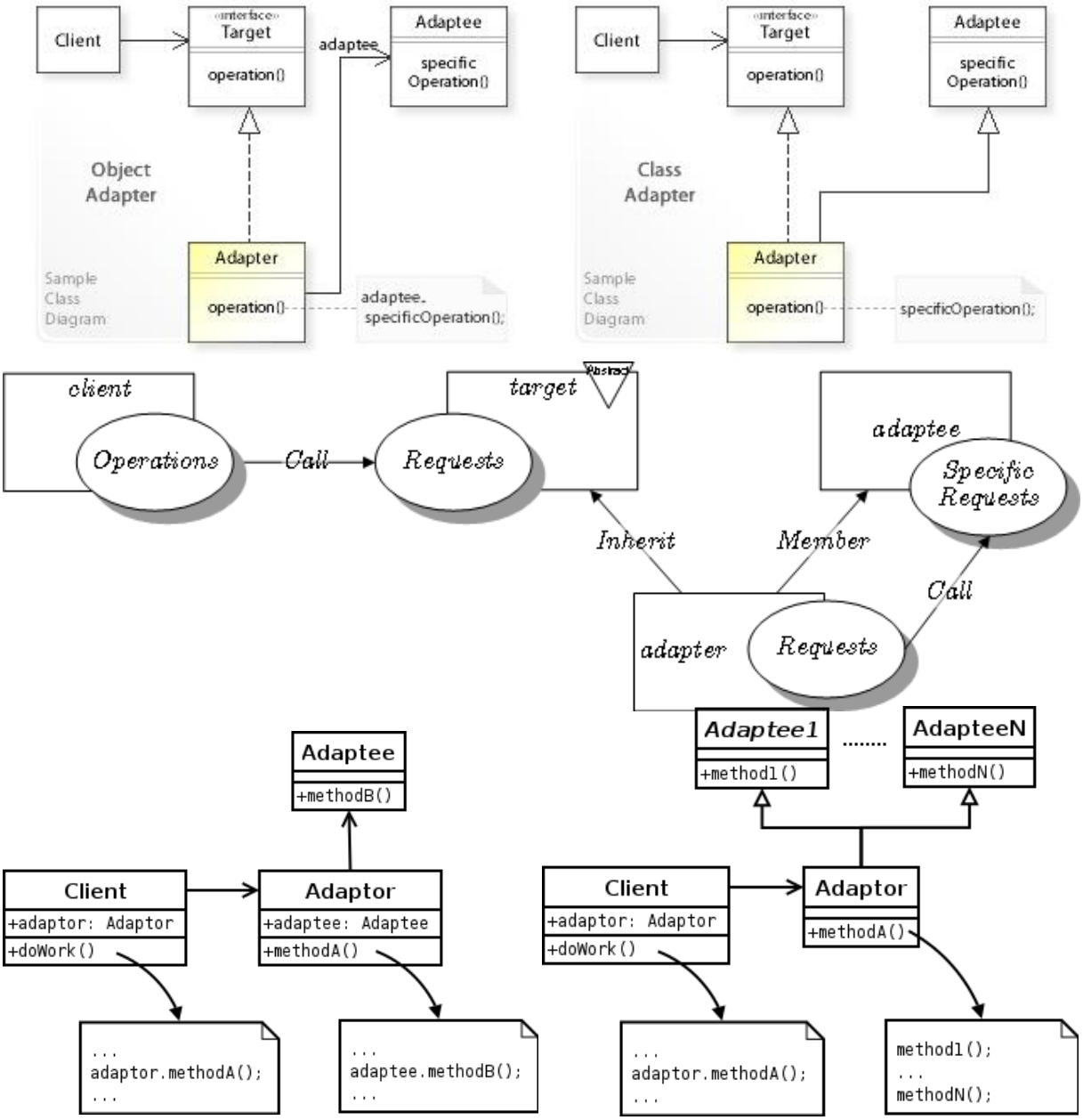
- Het toestaan van 'lazy' allocatie en initialisatie.
- De singletons houden het gebruik van onnodige variabelen tot een minimum om het aantal namen voor globale variabelen klein te houden.



https://en.wikipedia.org/wiki/Singleton_pattern

Adapter Pattern

- Een andere benaming is een ‘wrapper’.
- Het is een ‘Software Design Pattern’ voor het gebruiken van de interface van een bestaande class als andere interface.
- Het wordt gebruikt om bestaande classes samen te laten werken met andere classes zonder de source code aan te passen.
- Het wordt gebruikt wanneer een bepaalde interface gehanteerd moet worden en de huidige interface geconverteerd moet worden naar die bepaalde interface.
- Classes kunnen worden hergebruikt.
- De ‘adaptee’ is in dit geval de interface van de class die niet toe te passen is op de hanteerbare interface.
- Er wordt een aparte adapter gebruikt voor het converteren naar een ander interface.
- Clients weten niet of ze direct werken met de ‘target’-class of dat ze dat doen via een adapter met een class die geen toepasbare interface heeft.



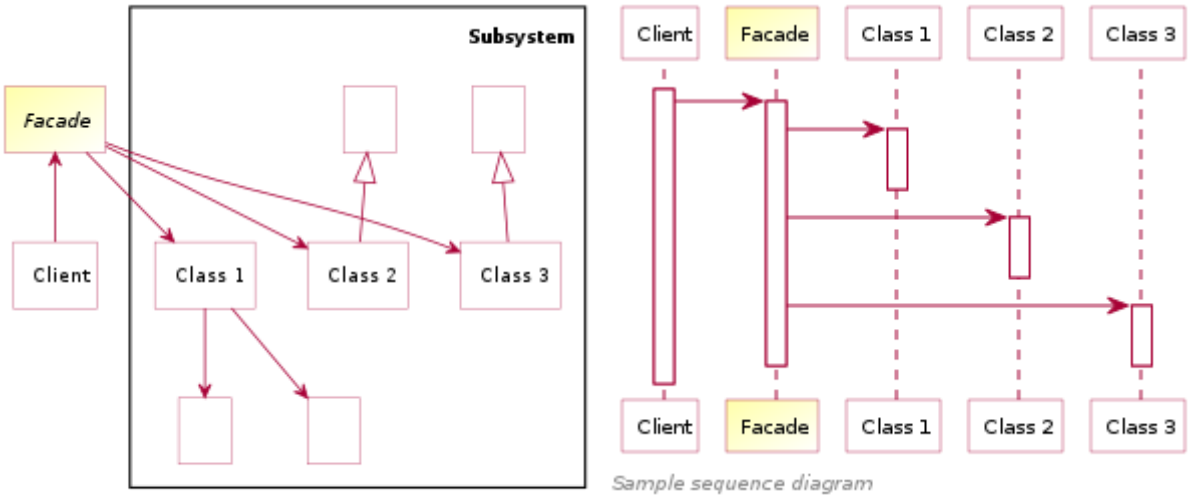
Voorbeelden:

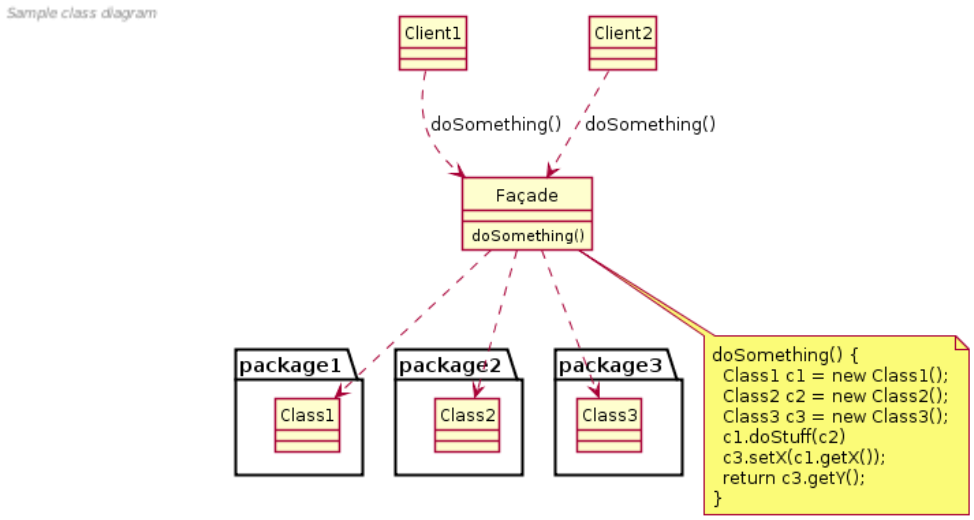
- De interface van een ‘Document Object Model’ van een XML-document die geconverteerd wordt naar een ‘tree structure’ dat op display tentoon kan worden gesteld.

https://en.wikipedia.org/wiki/Adapter_pattern

Facade Pattern

- Een façade is een object dat dient als een ‘front-facing’ interface om complexere onderliggende of structurele code te verbergen.
- Een façade kan zorgen voor een beter gebruik en een betere lesbaarheid van een software library. Door complexe componenten achter een enkele API te verbergen.
- De interface is meer context specifiek.
- De code is niet nauw met elkaar verbonden.
- Wordt vaak gebruikt bij complexe systemen die moeilijk te begrijpen zijn, aangezien het systeem te veel onafhankelijke classes bevat of omdat de source code niet beschikbaar is.
- De Facade Pattern bevat meestal een ‘wrapper’-class dat een set van ‘members’ bevat waar de client bij kan. De ‘members’ hebben toegang tot het systeem en verbergen hun implementatie details.
- Het aantal dependencies van een subsysteem zijn door deze Design Pattern geminimaliseerd.
- Een ‘Façade’-object wordt gedefinieerd door het implementeren van één interface door te delegeren naar de interfaces van de subsystemen. In het geval dat er meerdere interface aan te pas komen.
- Een ‘Façade’-object kan bijkomende functionaliteit bevatten.

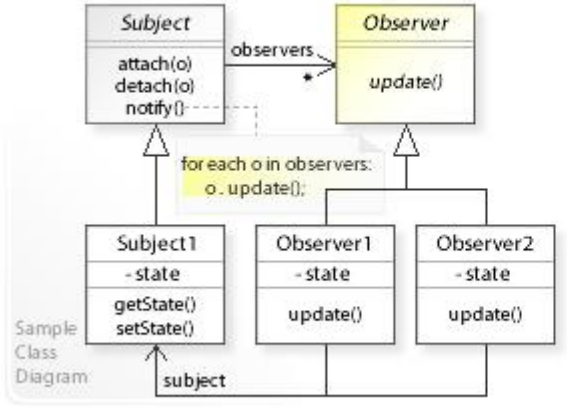


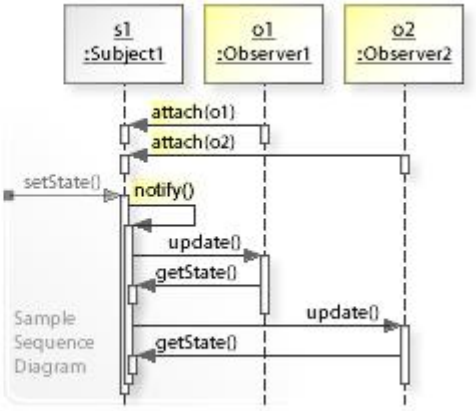


https://en.wikipedia.org/wiki/Facade_pattern

Observer Pattern

- Object heet hier een ‘subject’ en het roept zijn lijst met dependents aan genaamd ‘observers’.
- De dependents worden ingelicht als er een verandering van state plaatsvindt, meestal door het aanroepen van een method/functie.
- Dit wordt voornamelijk toegepast bij ‘event handling’ systemen.
- De observers gebruiken achtergrond threads voor het luisteren naar subject-events.
- Een ‘one-to-many dependency’ tussen objecten kan gedefinieerd worden zonder deze nauw vast te koppelen.
- Wanneer één object van state verandert, dan updaten de dependents automatisch.
- Het is mogelijk voor één object om een openstaand aantal objecten te notificeren.
- De taak van een subject is het onderhouden van een lijst met observers en het aanroepen van hun ‘update’-functies.
- De taak van een observer is het vastmaken of loskoppelen van zichzelf aan een subject en het updaten van hun state wanneer er door de subject geroepen wordt.





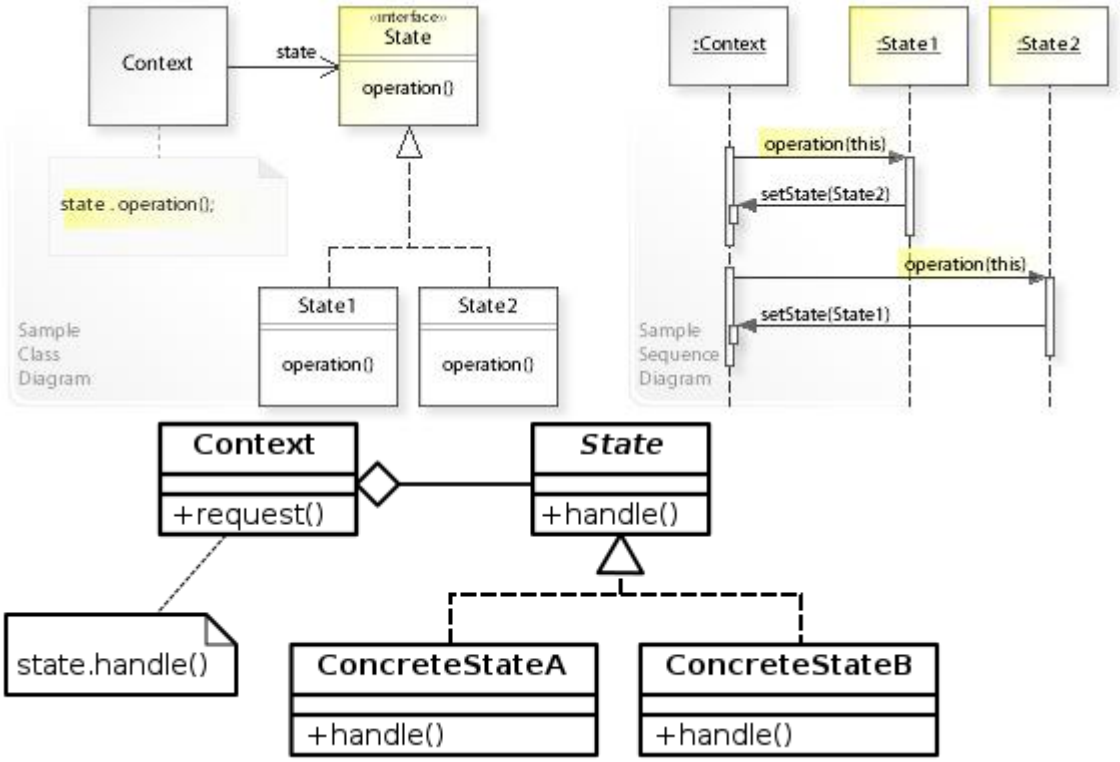
Voorbeelden van toepassingen voor het ontvangen van data:

- HTTP-requests
- GPIO data
- Keyboard- of muis input
- Database distributie
- Blockchains

https://en.wikipedia.org/wiki/Observer_pattern

State Pattern

- De ‘State Pattern’ is een ‘Behavioral Design Pattern’ dat een object zijn gedrag kan aanpassen wanneer zijn interne ‘state’ verandert.
- Deze pattern lijkt op het concept van ‘Finite-State Machines’.
- De State Pattern kan gezien worden als een Strategy Pattern. De strategie van een object kan veranderd worden door het aanroepen van functies in de interface van de patterns.
- Dit wordt gebruikt voor het afzonderen van variërend gedrag in één object, op basis van de ‘state’.
- Het gedrag wordt in ‘runtime’ aangepast.
- Dit wordt gebruikt als alternatief voor conditionele statements. Zo wordt onderhoud veel beter toegepast.
- Gedrag dat ‘state’ specifiek is zou apart van elkaar gedefinieerd moeten worden. Het toevoegen van nieuwe states zal geen effect moeten hebben op de andere states.
- ‘State’ specifiek gedrag zou niet gedefinieerd moeten worden binnen in een class. Het toevoegen van een andere state zou moeten gebeuren zonder het aanpassen van de class.
- States zijn specifieke objecten die afgezonderd zijn van elkaar. Een interface wordt gebruikt voor het uitvoeren van ‘state’ specifiek gedrag.



- Een class delegeert ‘state’ specifiek gedrag naar het state object in plaats van het direct implementeren van gedrag.

https://en.wikipedia.org/wiki/State_pattern