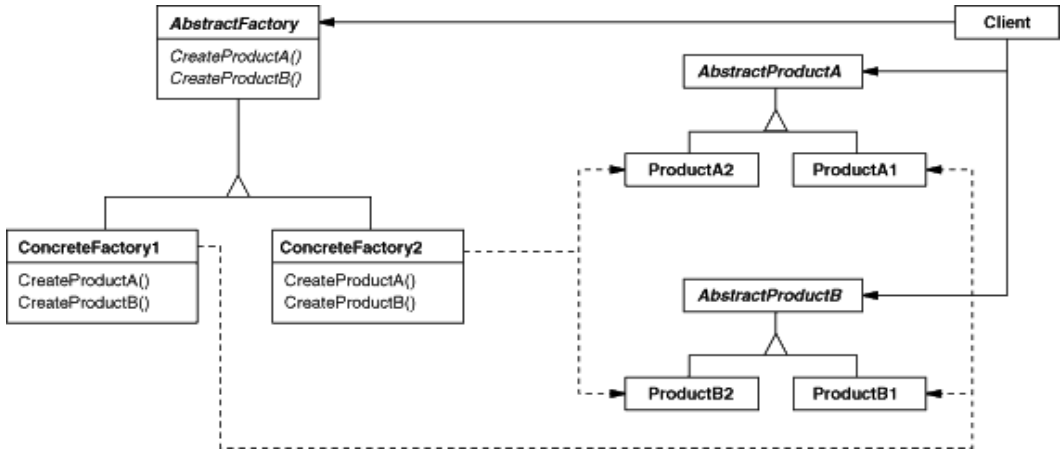
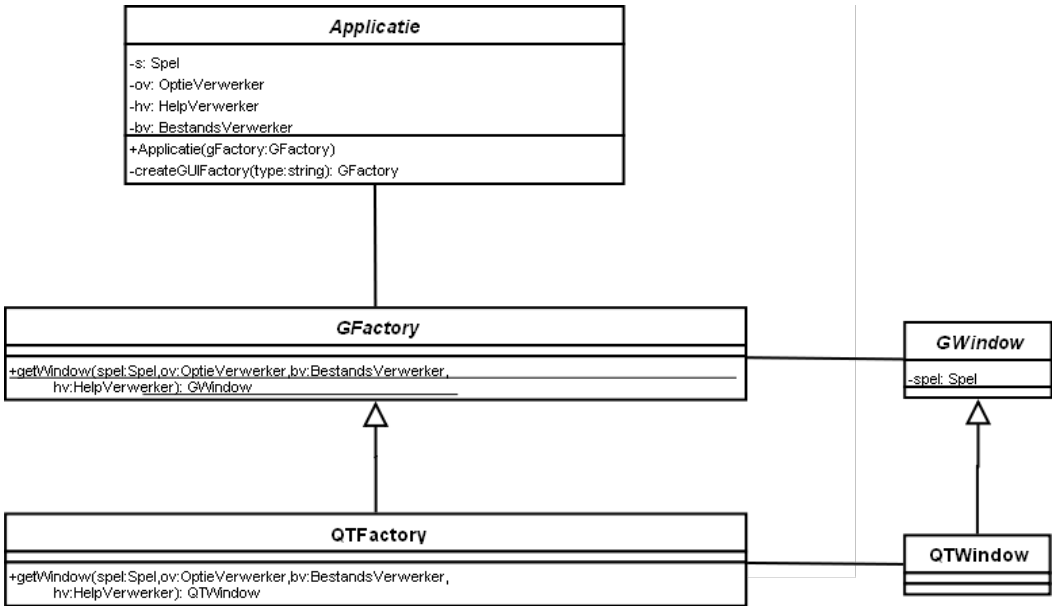
Gebruikte Design Patterns

Abstract Factory (Design Patterns p. 99):

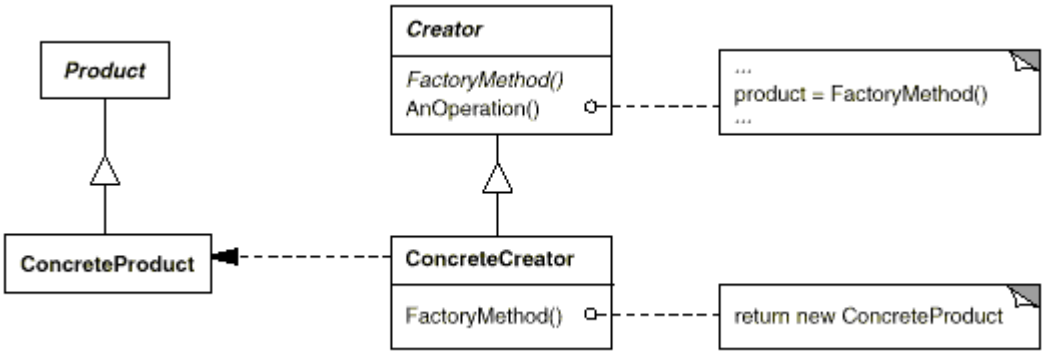


Motivatie:
Dit designpatroon laat ons toe eenvoudig nieuwe API-afhankelijke subklassen te gebruiken. Zo zal *AbstractFactory* een interface zijn, die API-afhankelijke *ConcreteFactories* implementeren. Elk van die concrete factories creëert op hun beurt concrete implementaties van abstracte *Product*-klassen, *Product*-klassen die *AbstractProduct*-interfaces implementeren. De *Client* kan m.b.v. een *ConcreteFactory* (zijnde een referentie naar een *AbstractFactory*) werken met *AbstractProduct*-referenties, gecreëerd doordat de *ConcreteFactory* allerlei concrete *Product*-implementaties bevat.

Betrokken klassen:
Applicatie (cfr. Client), GFactory (cfr. AbstractFactory), QTFactory (cfr. ConcreteFactory), GWindow/GSpeelveld/... (cfr. AbstractProduct), QTWindow/QTSpeelveld/... (cfr. ConcreteProduct).
Applicatie krijgt in zijn constructor een GFactory-referentie mee. Die wordt bekomen d.m.v. het Factory Method pattern. De concrete implementatie van GFactory, in dit geval QTFactory, houdt een GWindow-referentie bij (zijnde in dit geval een QTWindow). Bij de creatie van een QTWindow wordt op zijn beurt allerlei QT-klassen aangemaakt, zijnde concrete implementaties van G-klassen.

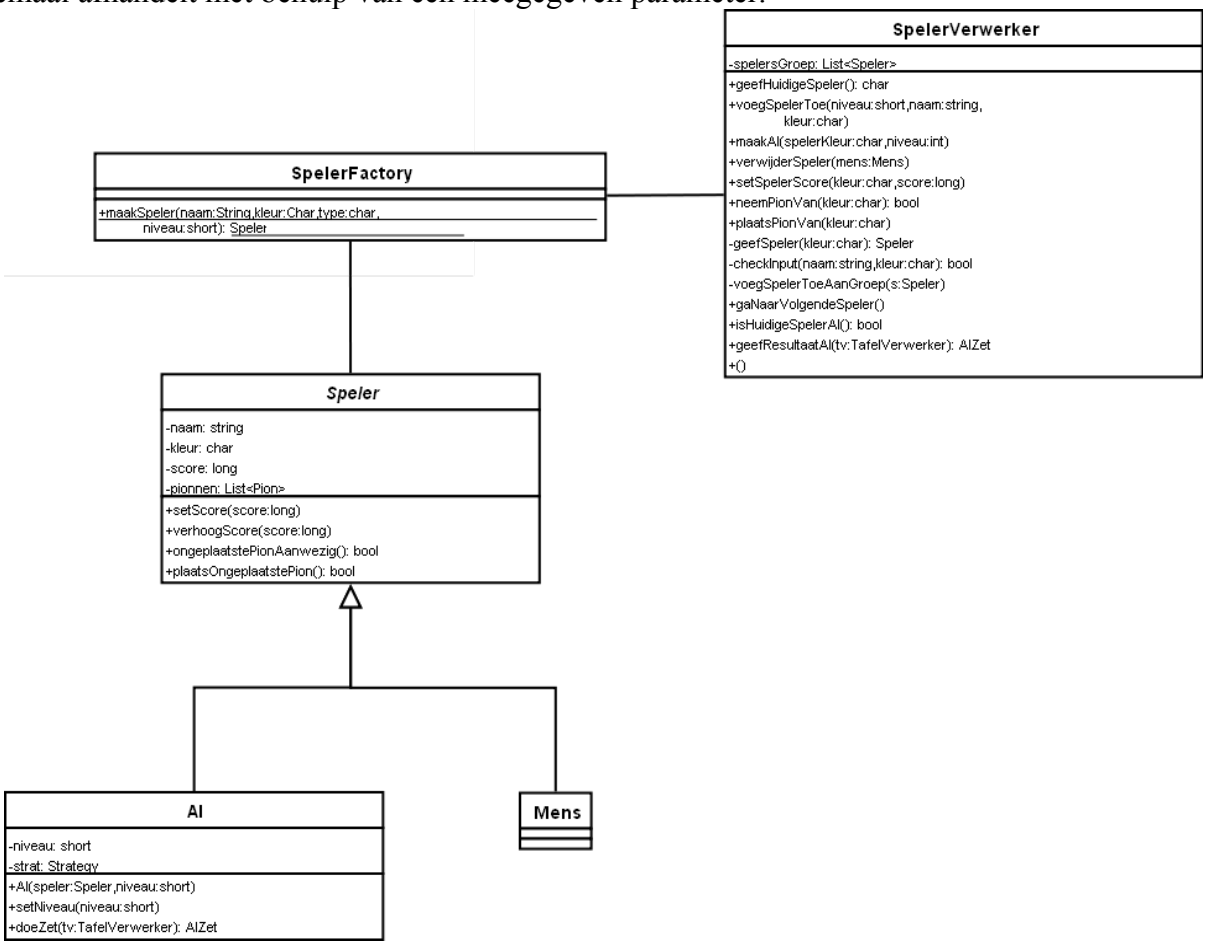


Factory Method (Design Patterns p. 121):

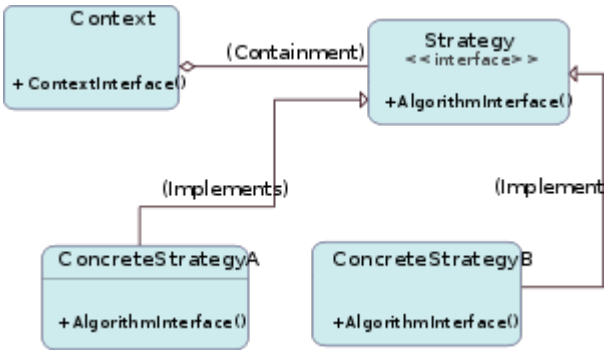


Motivatie:
Dit designpatroon abstraheert constructoren van super- en subklassen tot één methode, die aan de hand van een meegegeven parameter de constructor van de juiste klasse gebruikt om alzo een instantie van de juiste klasse aan te maken. Het resultaat van deze methode is een referentie naar de instantie, bekeken als een superklasse-referentie.

Betrokken klassen:
Speler (cfr. Product), Mens en AI (cfr. ConcreteProduct), SpelerFactory (cfr. ConcreteCreator). FactoryMethod wordt gemapt op maakSpeler(type).
We gebruiken de Speler-interface doorheen de andere klassen, doch moeten we in staat zijn Mens- én AI-instanties (subklassen van Speler) aan te maken. Het gebruik van het Factory Method designpatroon heeft als voordeel dat wanneer we ergens een Speler-instantie moeten aanmaken, we geen gebruik moeten maken van een if-else-structuur, maar van één factorymethode die dat allemaal afhandelt met behulp van een meegegeven parameter.

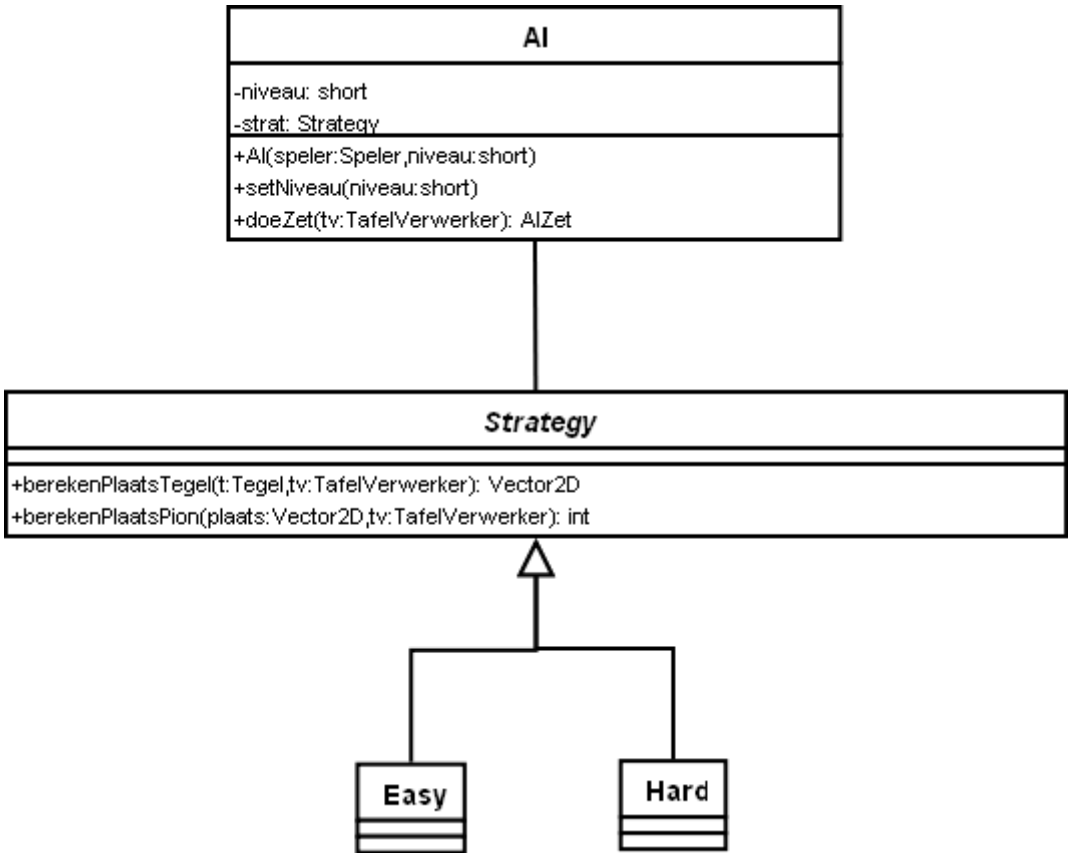


Strategy (Design Patterns p. 349):

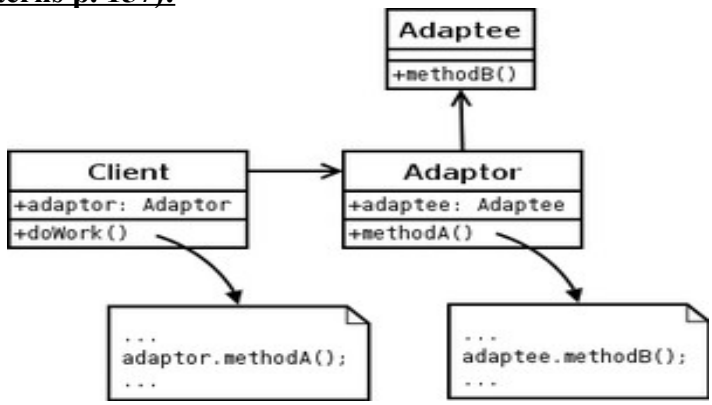


Motivatie:
Context houdt een referentie bij naar een *Strategy*-instantie. Afhankelijk van naar welke concrete instantie deze refereert, verandert het achterliggende algoritme (*AlgorithmInterface*) van een bepaalde methode uit de *Strategy*-interface.

Betrokken klassen:
AI (cfr. Context), Strategy (cfr. Strategy), Easy (cfr. ConcreteStrategy) en Hard (cfr. ConcreteStrategy).
Het spreekt voor zich dat dit designpatroon bijzonder handig is voor de implementatie van verschillende AI-niveaus. In AI maken we afhankelijk van het meegegeven AI-niveau, een referentie naar een Strategy-instantie, zijnde een referentie naar Easy of Hard. Impulsief zouden we dit ook bekomen kunnen hebben d.m.v. if-else-structuren in AI in combinatie met een aantal extra hulpfuncties.

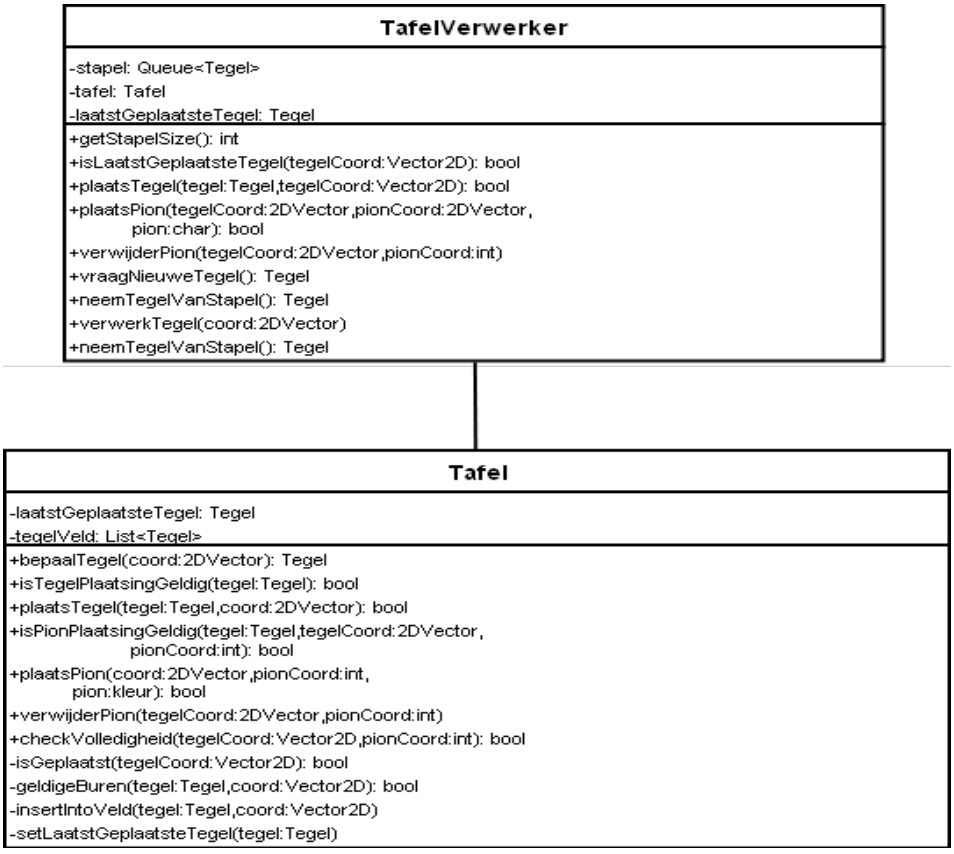


Adapter (Design Patterns p. 157):

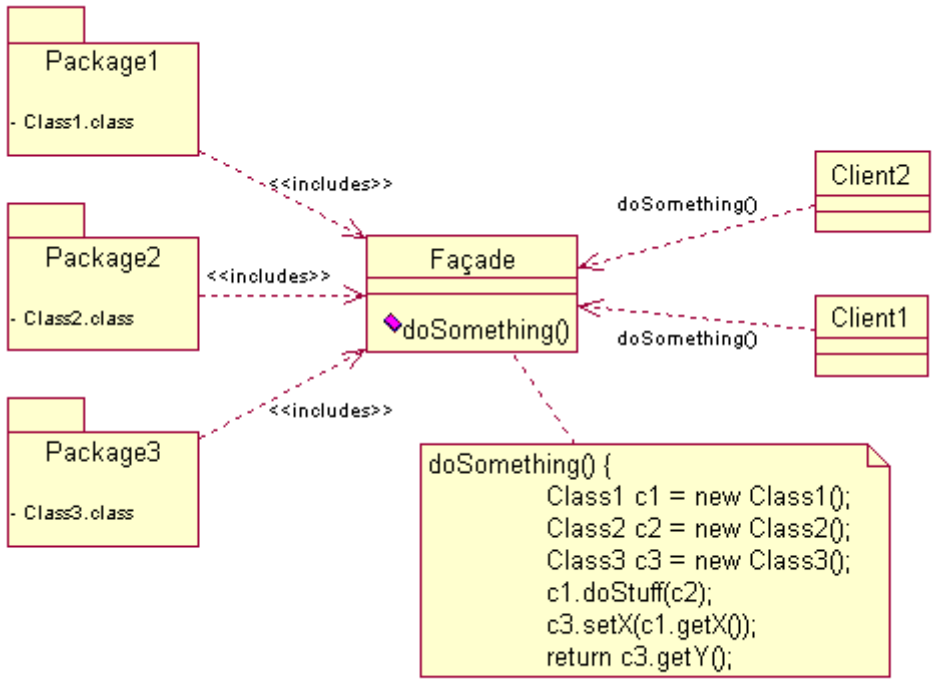


Motivatie:
Dit designpatroon laat ons toe een high-level interface (*Adaptor*) te creëren, een interface die door vele andere klassen (*Client*) kan gebruikt worden, zonder dat die daarvoor de interne structuur/opslag van de *Adaptee*-klassen moeten kennen. Met kan de *Adaptor*-klasse dus beschouwen als een soort van *translation-interface*, die a.h.v. door *Client*s gegeven data de functies van de *Adaptee*-klassen in de juiste volgorde en met de correcte parameters gaat aanroepen.

Betrokken klassen:
Tafel (cfr. Adaptee), TafelVerwerker (cfr. Adaptor), Spel (cfr. Client).
Speler (cfr. Adaptee), SpelerVerwerker (cfr. Adaptor), Spel (cfr. Client).
...
Tafel werkt bijvoorbeeld met een List van Tegels. Hoe die precies gemapt wordt op de visuele voorstelling van het tegelveld moeten andere klassen/clients helemaal niet weten. Daarom is de klasse TafelVerwerker gecreëerd: die wordt als het ware *bovenop* Tafel geplaatst, zodat Clients via TafelVerwerker alle acties op Tegels via tegenover de starttegel relatieve coördinaten kunnen aanspreken.

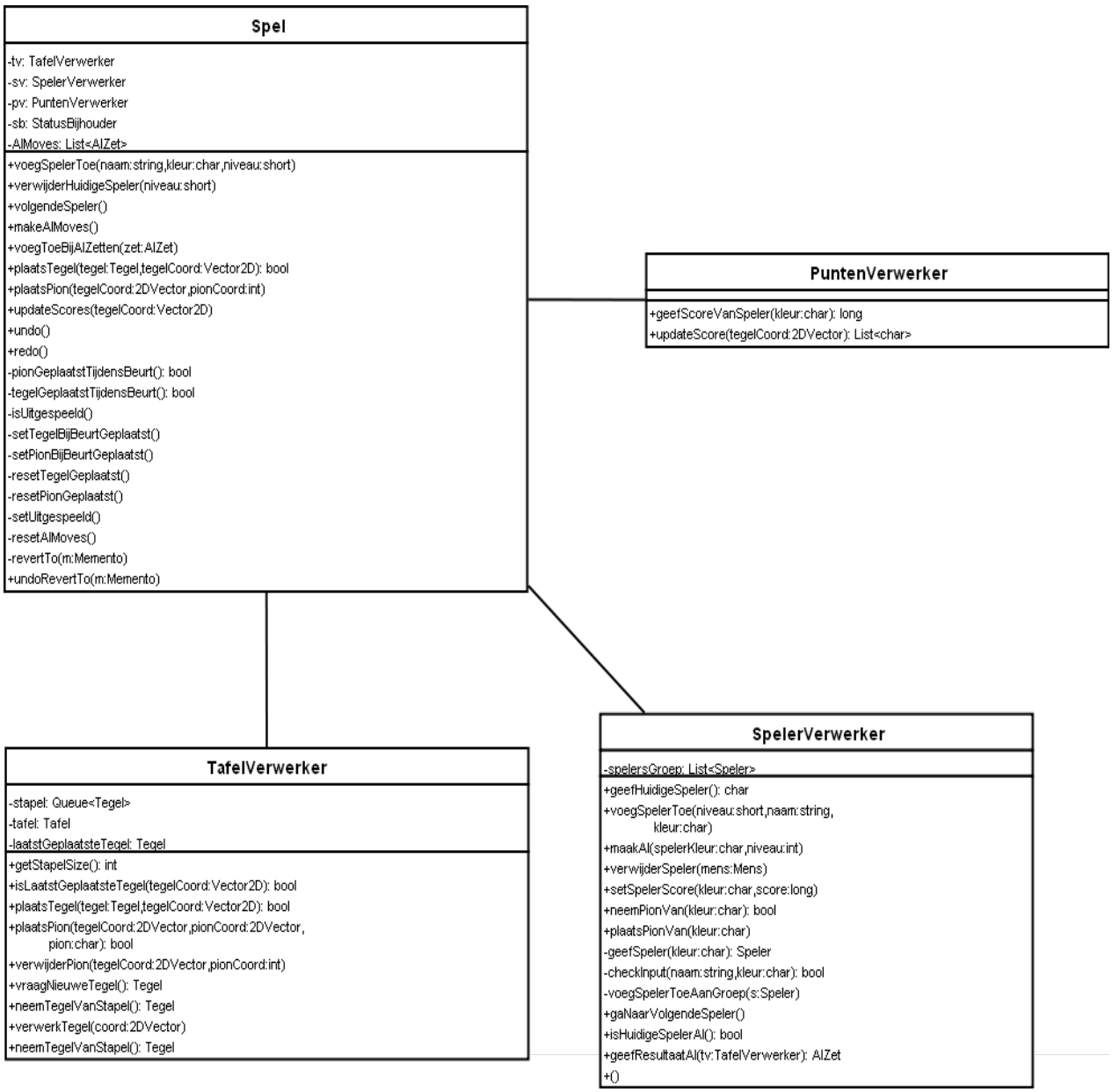


Facade (Design Patterns p. 208):



Motivatie:
Facade bevat een aantal high-levelfuncties, functies die 'buitenstaanders' van de *Facade* kunnen gebruiken. De implementatie van deze functies maakt gebruik van functies uit andere klassen. *Facade* brengt dus samenhorende functionaliteit uit diverse klassen samen in één klasse, één klasse waar 'buitenstaanders' aan genoeg hebben qua functionaliteit.

Betrokken klassen:
Spel (cfr. Facade), SpelerVerwerker (cfr. Class1), PuntenVerwerker (cfr. Class2), TafelVerwerker (cfr. Class), GSpeelVeld (cfr. Client1)
Spel brengt functionaliteit uit SpelerVerwerker, PuntenVerwerker en TafelVerwerker samen in één high-level interface, waarvan GSpeelVeld dan kan gebruik maken. Het automatisch afwerken van de spelbeurt van AI-speler (makeAIMoves()) heeft bvb. nood aan én SpelerVerwerker én TafelVerwerker én PuntenVerwerker.

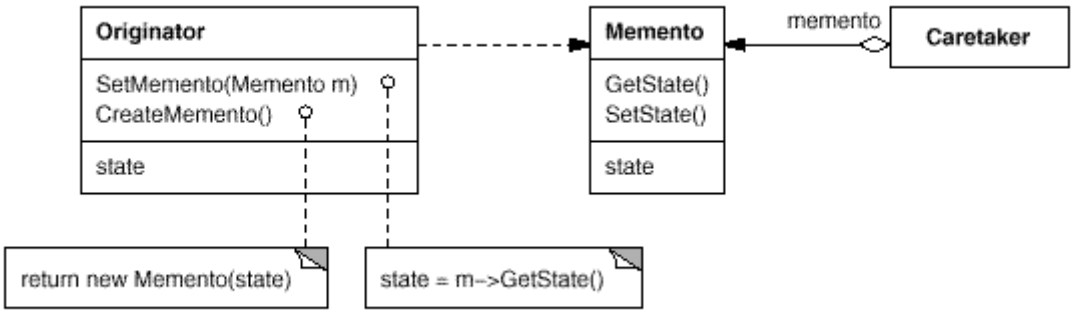


(Facade) Controller (GRASP p. 237):
(zie ook Facade hierboven)

Motivatie:
Facade Controller-klassen zijn klassen die een bepaald subsysteem voorstellen. Deze controllerklasse bevat vooral functies die vanuit de G-klassen moeten worden aangeroepen. Men kan controllerklassen dus bekijken als een communicerende laag tussen de GUI-klassen en de Core-klassen. Gebeurt er bijvoorbeeld een bepaald event in de GUI, dan kan daarop gereageerd worden door een passende functie(s) uit een Facade Controllerklasse aan te roepen. Deze functies zijn dus, net zoals die uit de facade klasse vermeld hierboven, een soort van high-level functies naar de 'buitenwereld' toe.

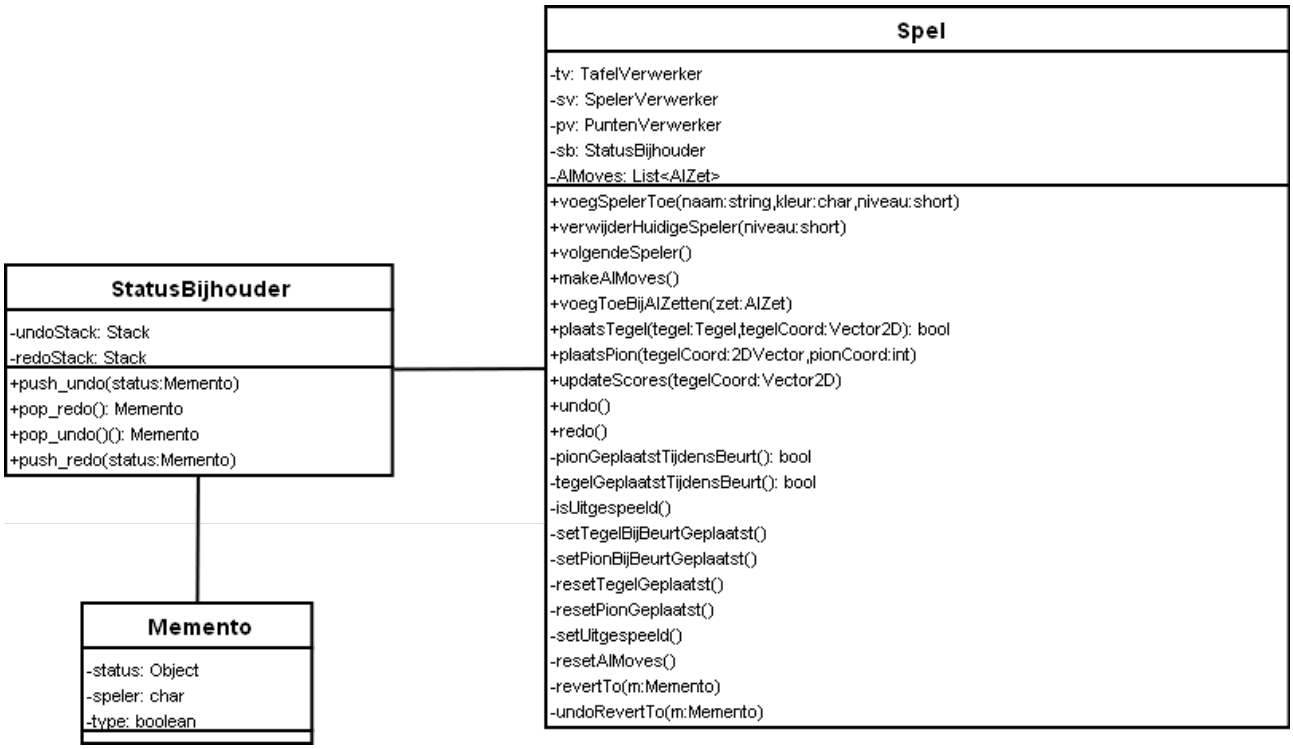
Betrokken klassen:
Spel (de Facede Controllerklasse), SpelerVerwerker (de geabstraheerde Core-klasse), GSpelerInfo (de GUI-klasse). (en gelijkaardige klassen)
GSpelerInfo kan bijvoorbeeld een event opvangen (pion nemen), en daaropvolgend functies uit SpelerVerwerker aanroepen door dat via Spel te doen.

Memento (Design Patterns p. 316):

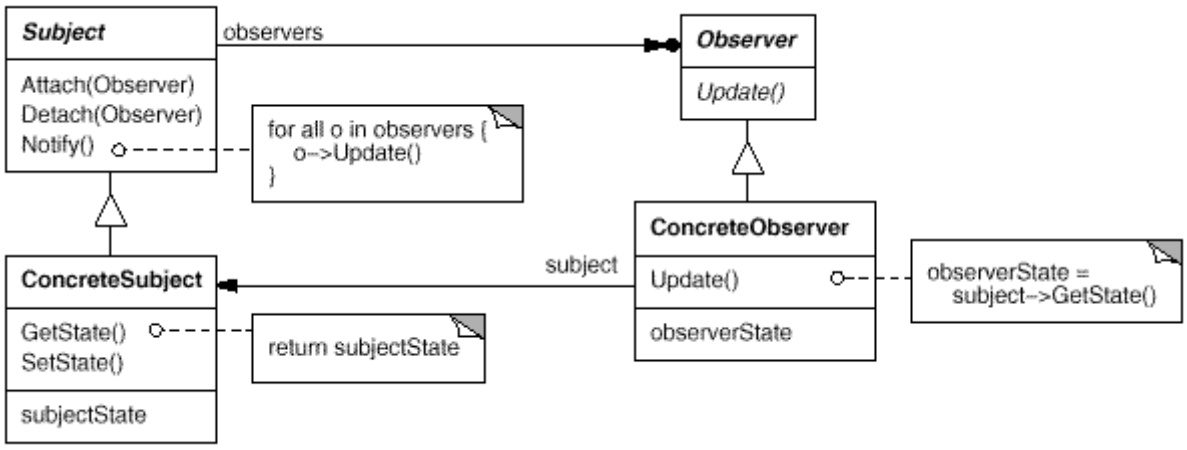


Motivatie:
Dit patroon geeft een eenvoudige structuur om toestanden/states van objecten bij te houden en terug te gebruiken indien gevraagd. *Memento* is een datastructuur die de toestand opslaat. *Originator* is het object waarvan we de toestand willen opslaan of willen terugzetten. *Caretaker* is de containerstructuur die alle *Memento*'s bijhoudt.

Betrokken klassen:
Memento (cfr. Memento), Spel (cfr. Originator), StatusBijhouder (cfr. Caretaker).
Memento is een datastructuur die de toestand van een Spel-instantie uniek kan bepalen. Spel bevat methodes die een status kan terugzetten m.b.v. een gegeven Memento, deze bevat eveneens methodes die zo'n Memento a.h.v. de huidige toestand kan genereren. StatusBijhouder is niet meer dan een container voor Memento-instanties.



Observer (Design Patterns p. 326):



Motivatie:
Dit patroon laat toe een klasse te laten reageren nadat een andere klasse simpelweg kenbaar heeft gemaakt veranderd te zijn. Hiertoe dient *ConcreteSubject* de interface *Subject* te implementeren die functionaliteit aanbiedt zodanig dat andere klassen (bv. *ConcreteObserver*) zich bij *ConcreteSubject* kunnen *aanmelden,* opdat zij op de hoogte zouden kunnen gebracht worden van veranderingen van *ConcreteSubject.* De bij de *ConcreteSubject* aangemelde klassen dienen wel de interface *Observer* te implementeren opdat *ConcreteSubject* voor elk van die klassen een *update*()-functie zou kunnen uitvoeren.

Betrokken klassen:
Observable (= interface cfr. Subject), Observer (cfr. Observer), GSpeelveld (cfr. ConcreteObserver), Spel (cfr. ConcreteSubject). Ook andere GUI-klassen hebben zich als Observer bij Spel geregistreerd.
Het is nu heel eenvoudig veranderingen in de Spel-gegevens (scoreverandering, tegelplaatsing, ...) in de GUI te laten zien. Spel dient enkel maar een gepast bericht naar zijn Observers te sturen, en de GUI-klassen kunnen daarna a.h.v. Spel-functies (functies die gegevens opvragen) de GUI updaten.