De laatste jaren zijn patterns, met name design patterns, sterk in populariteit gegroeid. Vaak wordt er vanuit gegaan dat iedereen volledig vertrouwd is met de concepten hierachter. In deze reeks artikelen behandelt Olav Maassen design patterns aan de hand van voorbeelden uit de dagelijkse praktijk die in Java geprogrammeerd zouden kunnen worden. De doelstelling is om design patterns inzichtelijk en begrijpelijk te maken en zo de mystiek weg te nemen en er zelf mee aan de slag te kunnen.

Het zijn maar patronen (5)

Factory-patronen binnen Java

Inmiddels zijn er al verschillende soorten patterns ontstaan: architectural patterns, system patterns, anti-patterns. In artikelen en boeken wordt verwezen naar design patterns zoals beschreven in het boek van Erich Gamma (et al.) (1)

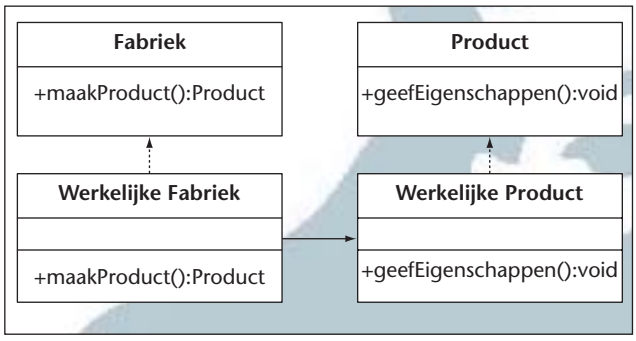
Sinds de industriële revolutie is de opmars van fabrieken schijnbaar onstuitbaar geweest. Geen stad is meer volwaardig zonder een industrie gebied vol fabrieken. Sinds de informatie revolutie zijn de fabrieken zelfs aanwezig in programmatuur, gelukkig zonder bijkomende milieu effecten. Binnen Java programma's wordt veelvuldig gebruik gemaakt van fabrieken, oftewel factories. Er zijn echter twee verschillende patterns die allebei de naam factory in de naam hebben: 'Abstract Factory' en 'Factory Method' (allebei uit het boek van Erich Gamma). Beide patterns zijn belast met het maken van objecten, maar de doelstelling is verschillend. Om een eind te maken aan de verwarring, worden beide patterns besproken en vergeleken.

Factory Method Op het moment dat je een applicatie gaat ontwikkelen, is het vaak nog niet duidelijk welke objecten je gaat gebruiken. Later tijdens het ontwikkelen zal dat verder uitgewerkt worden, maar om toch gelijk door te kunnen gaan, wordt er gebruik gemaakt van interfaces. Echter, het nadeel van interfaces is dat zij geen constructor kunnen hebben. Dit nadeel kun je met behulp van een Factory Method omzetten naar een voordeel.

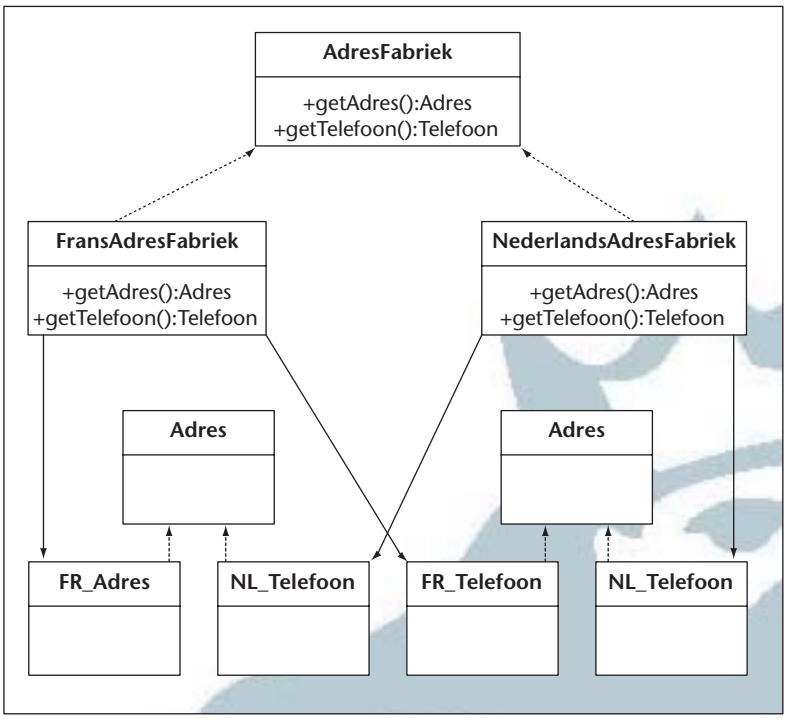
Uiteraard is het mogelijk om gelijk een class te programmeren die verantwoordelijk is voor het instantiëren van de classes die de interface implementeren, maar dat was nu net wat we niet wilden doen. Doel was uit te gaan van interfaces. Oplossing is ook het creëren van objecten in een interface te stoppen, maar dan zonder een constructor. Entree voor de Factory Method: een interface met een methode die als return type een andere interface heeft. Doordat het maken van de instanties in een interface zit, is het bepalen van de implementaties uitgesteld tot een later tijdstip, maar het tijdstip van instantiëren kan al wel gebruikt worden. Tevens biedt het de mogelijkheid later andere implementaties toe te voegen.

> Doel van de Factory Method is het kunnen maken van objecten anders dan via de constructor

Een bijkomend voordeel van deze aanpak is dat als je de interface distribueert en anderen gaan er gebruik van maken, je enkel en alleen de interface prijs hoeft te geven. Via interfaces is de volledige functionaliteit



Figuur 1. Factory Method is een interface met een methode die als return type een andere interface heeft



FIGUUR 2. De Abstract Factory is een productie platform voor het maken van objecten die hetzelfde thema delen

beschikbaar zonder dat de implementerende subclasses bekend worden. Op deze manier is het mogelijk geworden subclasses te 'verbergen'.

Een andere reden voor het gebruik van de Factory Method is een beperking van de Java taal. Een constructor moet altijd dezelfde naam hebben als de class waar deze in zit. Als er meerdere constructors in een class staan, moeten deze allemaal een parameterlijst hebben die voldoende verschilt van de andere constructors (overloading). Echter als je twee constructors met dezelfde parameters wil hebben, heeft Java daar problemen mee.

De oplossing voor dit probleem is de Factory Method. De Factory Method geeft de mogelijkheid meerdere methoden te hebben die werken als een constructor. Door de namen te laten verschillen van deze Factory Methods is het wel mogelijk dat op basis van dezelfde parameterlijsten verschillende objecten gemaakt worden zonder dat Java gaat klagen.

> Doel van Abstract Factory is het creëren van een set van gerelateerde of afhankelijke objecten

Voordelen Factory Method

- Factory Method is uitermate geschikt voor het opzetten van een framework. De implementatie is gescheiden van de interface.
- Beslissingen kunnen uitgesteld/verschoven worden naar de subclasses.
- Je kan een interface geven voor wanneer een object gemaakt moet worden, zonder de controle te verliezen over welk specifiek type geïnstantieerd wordt.
- Meerdere constructors met dezelfde argumentenlijst

Abstract Factory Stel, je applicatie kent een aantal context afhankelijke typen. Te denken valt aan adres en telefoon gegevens en hoe deze weergegeven moeten worden in verschillende landen of een platform onafhankelijke GUI. In beide gevallen zijn er duidelijke relaties tussen verschillende componenten, een persoon heeft altijd één of meer adressen en één of meer telefoonnummers, een button moet op het ene platform anders getekend worden dan op een ander.

De typen delen een gezamenlijk thema waardoor ze sterk aan elkaar gerelateerd zijn. Dit heet ook wel een familie. Het is uiteraard verre van handig in elk van deze typen af te vragen in welke context of thema het programma nu zit. Beter is het om deze check één keer uit te voeren en vanaf dat moment alleen de objecten van dat thema te maken.

Dit is wat de Abstract Factory biedt. De Abstract Factory is een productie platform voor het maken van objecten die hetzelfde thema delen. In Frankrijk zal het programma de adres- en telefoonregels van Frankrijk gebruiken en in Nederland de Nederlandse. Een GUI zal een keer controleren op welk platform het draait en vervolgens alleen grafische componenten voor dat platform maken.

Ook bij de Abstract Factory is het mogelijk de implementerende classes te verbergen van de client. Alle interacties kunnen verlopen via abstract classes en interfaces. Deze constructie komt de onderhoudbaarheid van de code zeer ten goede. Wijzigingen kunnen nu beperkt worden tot de classes of het thema waar het betrekking op heeft.

Voordelen Abstract Factory

- Client van de Abstract Factory wordt onafhankelijk van de implementatie van de onderliggende producten
- De client kan eenvoudiger omgaan met context-afhankelijke groepen van objecten
- De implementerende classes zijn verborgen voor de client terwijl het toch mogelijk blijft families van objecten te creëren.
- Gerelateerde hiërarchieën van objecten zijn eenvoudiger te managen.

Conclusie Met zowel Factory Method als Abstract Factory is het mogelijk objecten te creëren (vandaar dat zij in de categorie 'creational patterns' zitten). Beide gebruiken interfaces of abstract classes om het maken van objecten los te trekken van de specifieke implementerende classes. Op deze manier verhogen zij het abstractieniveau van de code. Tevens bieden zij allebei de eigenschap implementerende classes te kunnen verbergen voor de rest van de code.

De Abstract Factory maakt vaak gebruik van de Factory Method voor het maken van objecten. De doelstelling van beide is echter verschillend. De Factory Method wil de toegang tot een specifieke interface (of abstract class) abstract maken, met de hierboven vernoemde voordelen. De Abstract Factory dient als productieplatform voor objecten die gerelateerd zijn, of die hetzelfde thema hebben.

Literatuur

Design Patterns, Erich Gamma et al. Addison Wesley, 1995

Applied Java Patterns, Stephen Stelting & Olav Maassen, Prentice Hall, 2002

Effective Java, Stephen Bloch, Addison Wesley, 2003

Refactoring, Martin Fowler et al., Addison Wesley, 1999

Olav Maassen (o.maassen@delion.com) is senior software ontwikkelaar bij Delion B.V. te Gorinchem en co-auteur van 'Applied Java Patterns' met Stephen Stelting.

PATCHES Patches PATCHES Patches PATCHES Patches PATCHES

Vervolg van pagina 17.

planning en estimation tool voor softwareprojecten van Software Productivity Center (SPC), een in Canada gevestigd bedrijf voor softwareprocesverbetering. Deze nieuw aangekochte tool zal opgenomen worden in de volgende versie van Borland's requirements managementoplossing, Borland CaliberRM.

Met de op requirements gebaseerde planningsmogelijkheden van ESTIMATE Professional kunnen gebruikers betrouwbare voorspellingen doen over de vereiste investering, time-to-market en de risico's van softwareontwikkelingsprojecten voor de aanvang ervan. De tool biedt mogelijkheden om de juiste projecten te selecteren, de juiste middelen toe te passen op deze projecten en om betrouwbaardere voorspellingen te doen over het resultaat van deze investeringsbeslissingen. De nieuwe mogelijkheden in CaliberRM helpen gebruikers bij het voorspellen van personeels-, plannings- en budgetparameters die worden afgezet tegen specifieke business requirements. De oplossing markeert tevens problemen en spoort afwijkingen op in de ontwikkellevenscyclus. Hierdoor kunnen gebruikers problemen in een vroeger stadium van de cyclus aanpakken, wanneer de kosten lager zijn. Zij kunnen dan het project weer op de juiste weg brengen, of het beëindigen voordat extra middelen worden aangewend voor een project dat geen adequate terugverdientijd kan bieden. Als onderdeel van de Borland Application Lifecycle Management (ALM) portfolio van oplossingen is CaliberRM een crossplatform requirements managementoplossing, die een basis vormt voor succesvolle software. Deze oplossing helpt softwareontwikkelteams de veranderende business behoeften beter te begrijpen en deze bij te houden, wat de noodzaak en de kosten van softwarewijzigingen reduceert. Borland CaliberRM 6.5 is gratis te downloaden van de Borland-website: http://www.borland.com/caliber/index.html.

De Zwarte Madonna's van de IT

Zoals al geconstateerd in het hoofdredactionele commentaar van het vorige nummer van Software Release Magazine, slaagt Ron Tolido er uitstekend in pakkende columns te schrijven. Blijkbaar is hij daar zelf ook van overtuigd, want onlangs verscheen bij bergboek.nl een aantal van zijn columns in boekvorm, onder de titel "De Zwarte Madonna's van de IT".

Ron Tolido is Chief Technology Officer voor Noord Europa en Asia Pacific bij Capgemini. Hij publiceerde boeken en artikelen over technologietrends en innovatieve systeemontwikkeling. Als informaticus ontwikkelde hij methoden en technieken voor systeemontwikkeling, was hij betrokken in architectuur- en strategieprojecten en stond hij aan de wieg van SDW, jarenlang de marktleider in Nederland op het gebied van software voor analyse en ontwerp. Gezien zijn achtergrond - een afgebroken studie Nederlandse Taal- en Letterkunde en een afgeronde ingenieursopleiding Hogere Informatica - is het niet verbazingwekkend dat hij zich altijd in het schemergebied tussen technologie en journalistiek heeft begeven. Zijn neigingen tot polemiek geeft hij de vrije teugel op het moment dat de journalistieke bril opgaat. Vanaf 1999 heeft hij columns geschreven voor diverse IT-bladen (Business Consultant, CM Corporate en Datanews Technical Series), momenteel voor Software Release Magazine. De gebundelde columns van Ron Tolido zijn te bestellen via het volgende Internet-adres: http://www.bergboek.nl/shop.htm.