Samenvatting Ontwerpen III

HoGent

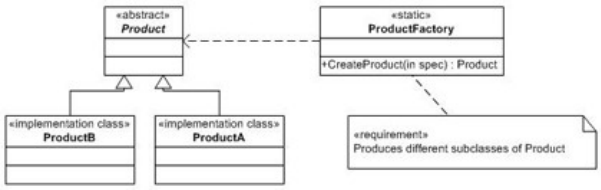
Lorenz Verschingel

8 april 2015

1 Factory Pattern

We pakken de code voor de creatie op en verplaatsen deze naar een ander object dat alleen maar het maken van producten als taak zal hebben. Dit object noemen we *Factory*.

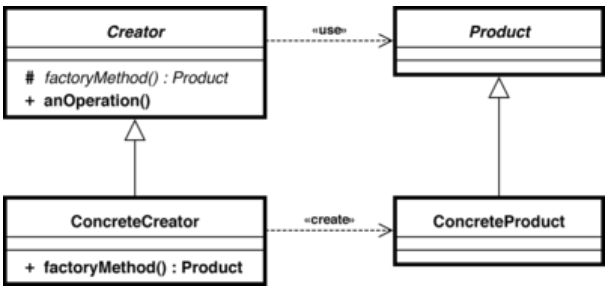
1.1 Simple Factory



Figuur 1: UML Simple Factory

Volgens de UML in figuur 1 kan de ProductFactory producten van het type Product afleveren aan zijn cliënten.

1.2 Factory Methode



Figuur 2: UML Factory Methode

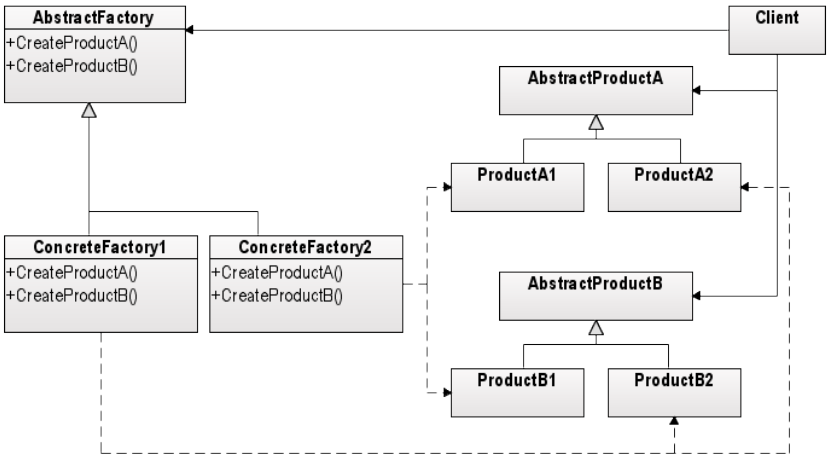
Ten opzichte van figuur 1 is er niet zoveel veranderd. In figuur 2 is de Factory klasse abstract geworden en is de create methode ook abstract. Deze wordt dan later door een concrete factory geïmplementeerd.

Het Factory Method Pattern definieert een interface voor het creëren van een object, maar laat de subklassen beslissen welke klasse er geïnstantieerd wordt. De Factory Method draagt de instanties over aan de subklassen.

1.3 Dependency Inversion-principe

Wees afhankelijk van abstracties, niet afhankelijk van concrete klassen.

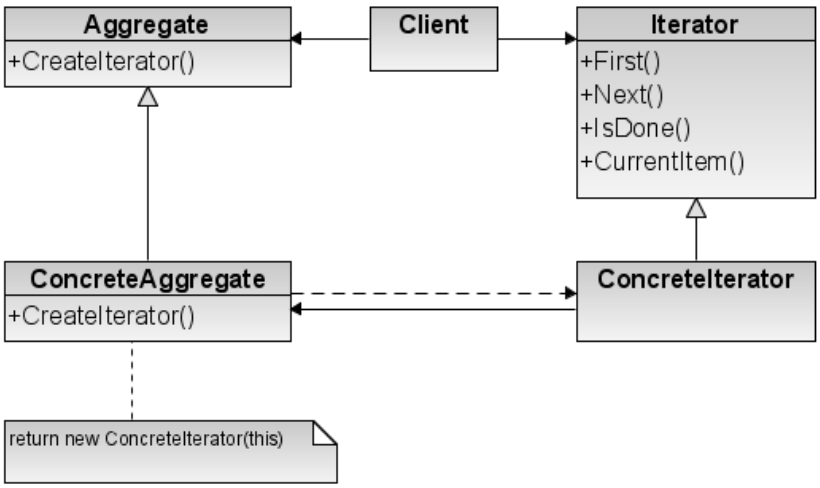
1.4 Abstract Factory Pattern



Figuur 3: UML Abstract Factory

Het Abstract Factory Pattern levert een interface voor de vervaardiging van reeksen gerelateerde of afhankelijke objecten zonder hun concrete klassen te specificeren.

2 Iterator Pattern



Figuur 4: UML Iterator Pattern

Het Iterator Pattern voorziet ons van een manier voor sequentiële toegang tot de elementen van een aggregaatobject zonder de onderliggende representatie weer te geven.

Figuur 4 heeft een redelijk uitgebreide verantwoordelijkheid aan de iterator. In de meeste gevallen volstaat het om een methode hasNext() en Next() in te voeren.

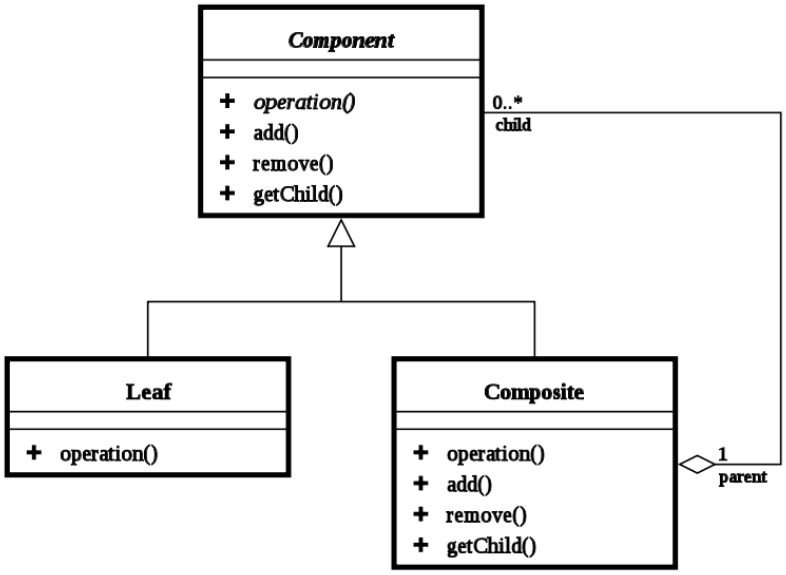
Java voorziet het iterator pattern met de klasse Iterator. Hiervoor moet men zorgen dat de iterator-klasse de klasse Iterator van Java gebruikt en dat de methode createIterator() het type Iterator<teItererenType> terug geeft.

3 Composite Pattern

Het Composite Pattern stelt je in staat om objecten in boomstructuren samen te stellen om partwhole hiërarchiën weer te geven. Composite laat clients de afzonderlijke objecten of samengestelde objecten op uniforme wijze behandelen.

Beschouw figuur 5. De klasse Component is een interface voor alle objecten in de compositie. Een leaf definiëert het gedrag voor de elementen in de compositie. Dit gebeurt via implementatie van de operaties die de klasse Composite ondersteunt. De klasse Composite definiëert het gedrag van de component met kinderen. Deze klasse moet ook alle kinderen kunnen bijhouden.

Zowel Composite als Leaf overriden enkel de methoden die zin hebben, en gebruiken de standaardimplementatie uit Component voor de methoden die niet zinnig zijn.



Figuur 5: UML Composite Pattern

3.1 De compositie-iterator

Om een Composite-iterator te implementeren voegen we de methode createIterator() toe aan iedere component. Voor de Composite klasse geeft deze methode een iterator over zijn kinderen terug, voor de klasse Leaf een NullIterator. Als over de hele boomstructuur geïtereerd moet worden dan moet de iterator van de root in een CompositeIterator klasse verpakt worden.

De klasse CompositeIterator implementeert de interface Iterator. Verder bevat deze klasse een stack waarop alle afzonderlijke Iterators geplaatst kunnen worden.