Het Object Pool Pattern

Analysis & Design 2

katholieke hogeschool
associatie KU Leuven
vives

Object Pool Pattern
De context

- Je hebt een **beperkte verzameling** (= een **pool**) van gelijkaardige objecten die je wilt ter beschikking stellen van de rest van het programma.
- De objecten worden door de pool "uitgeleend" aan clients.
- Als een client een object niet meer nodig heeft, dan geeft die het terug aan de pool, zodat het opnieuw gebruikt kan worden.

Waarom een Object Pool

Performancewinst

- Als de kost van het instantiëren van een poolobject hoog is, en poolobjecten dikwijls voor een korte tijd gebruikt worden.
- Als instantiëren van een poolobject mogelijk lang duurt.

Beperking van het aantal objecten

Met het object pool pattern kun je het aantal poolobjecten dat tegelijk bestaat beperken.

Nadelen van het pattern

- De client moet het poolobject expliciet vrijgeven na gebruik (geen garbage collection).

- Poolobjecten moeten na elk gebruik naar een propere begintoestand teruggebracht worden.

- In een multithreaded omgeving is de implementatie niet zo eenvoudig.

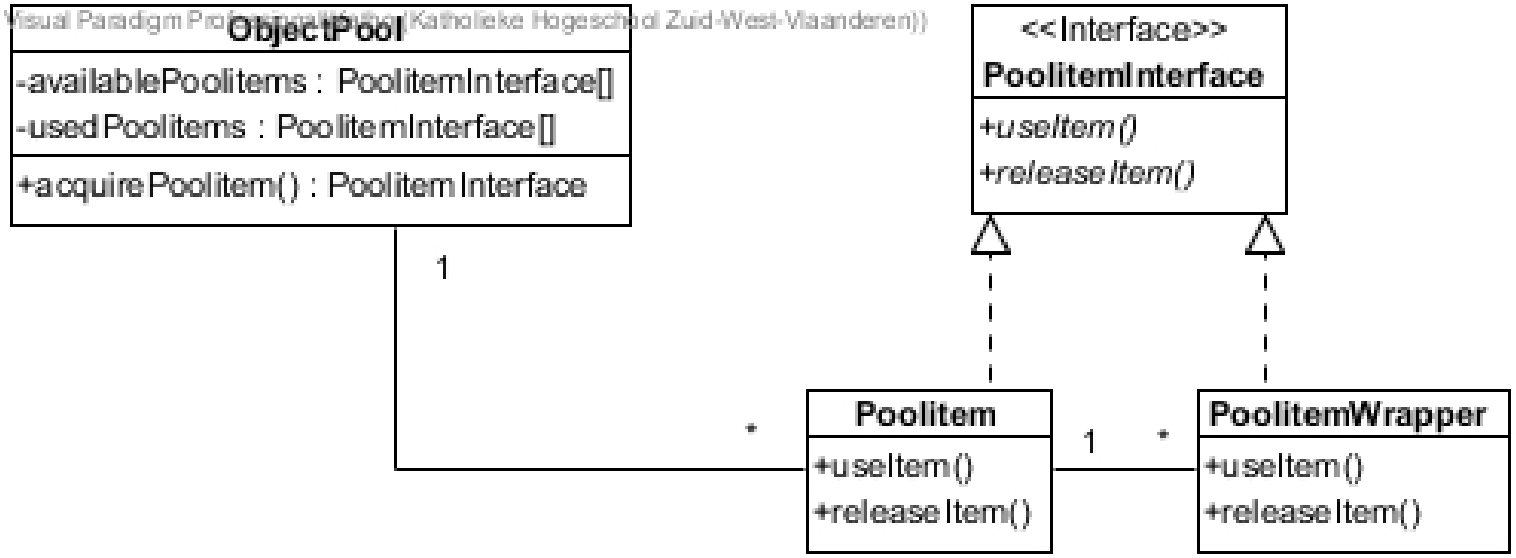
Wat als de pool leeg is?

Wat als een client een object vraagt aan de pool, maar er geen ter beschikking is?

Er zijn drie mogelijkheden:

1. De pool geeft een error.
2. De pool maakt een nieuw poolobject (tot een zeker maximum).
3. In een multithreaded omgeving: de thread wordt in een blokkerende wachttoestand gezet, totdat er een poolobject vrijkomt.

Structuur van het pattern



Implementatie

- De pool heeft een factory-achtige methode waarmee clients een poolobject kunnen opvragen (*acquirePoolitem*).
- Clients krijgen geen rechtstreekse toegang tot poolobjecten (beveiliging). Ze krijgen een wrapperobject met daarin het poolobject.
- De wrapperobjecten hebben een methode waarmee de client ze kan vrijgeven (*close*).

  Die close-methode heeft access nodig tot private methodes van de pool. Dat kan door de wrapper een inner class van de pool te maken.

  (Zie codevoorbeeld.)