**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 het dagelijks leven 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 (6)

## *Observer pattern: 'Don't call us, we'll call you'*

Daar zit je dan. Op zoek naar je volgende baan. Na een half uur wachten wordt je naar een kamer geleid voor je sollicitatiegesprek. Tijdens het gesprek doe je je best om je zo goed mogelijk te profileren en zoveel mogelijk informatie over jezelf te verstrekken. Informatie die de interviewer zal gebruiken bij zijn overwegingen om te bepalen of en zo ja wie zij een aanbieding willen doen. Na 45 minuten sta je weer buiten. Na het bijna obligate 'Don't call us, we'll call you' begint het grote wachten.

Tijdens de periode die dan volgt, kun jij je tijd besteden aan andere activiteiten terwijl je wacht. De interviewer zal de tijd gebruiken om je gegevens te controleren en te vergelijken met andere kandidaten. Als zij eindelijk vijf weken later een beslissing hebben genomen, nemen zij contact op met de gewenste nieuwe werknemer. Uiteraard zullen zij contact opnemen op de wijze, die de kandidaat heeft aangegeven. Dat kan via e-mail, telefoon, of post. Als je het bericht ontvangen hebt, kun je gaan bepalen wat de volgende stappen gaan zijn: andere sollicitaties sturen, aanbieding aanvaarden, verder onderhandelen. Het belangrijkste is dat jij verder kunt met je leven.

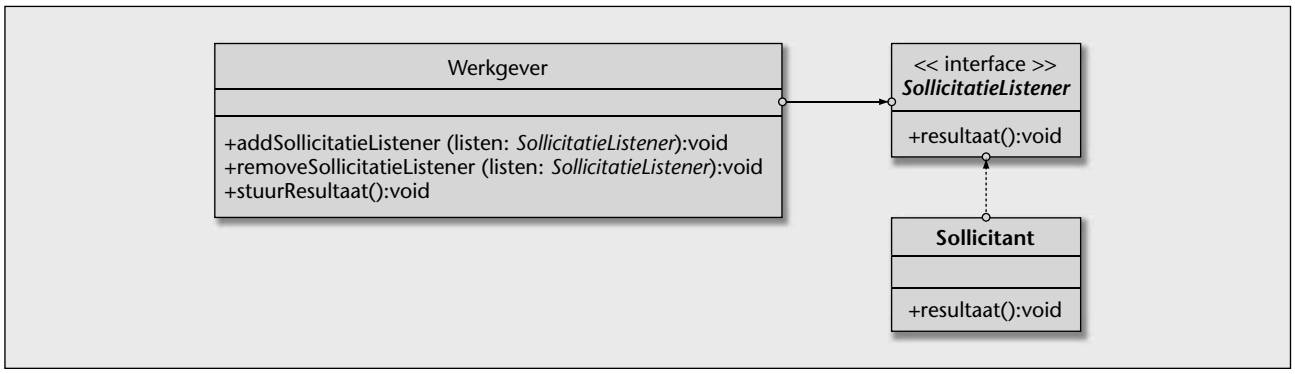
Het hiervoor beschrevene klinkt natuurlijk heel gewoon en dat is het ook. Als je dit in Java programmeert is het weinig zinvol om dit synchroon uit te laten voeren; dat is ongeveer hetzelfde als dat je in de wachtkamer gaat zitten totdat je hoort wat de uitslag is. Dat zou tot zeer vervelende situaties kunnen leiden.

Gelukkig is er in Java het listener principe. De werkgever is de event source en jij als werkzoekende registreert je bij de werkgever als geïnteresseerde listener. Zodra er nieuws is, krijg jij bericht. Deze constructie is op verschillende plaatsen in Java gebruikt en staat ook beschreven in de GoF(1) onder de naam 'Observer'. Omdat in een client-server omgeving er specifieke uitdagingen zijn, kan het 'Callback' pattern (2) daar uitkomst bieden.

**OBSERVER** Het Observer oftewel het Listener pattern is een meer eenvoudige pattern, dat met relatief kleine ingrepen veel flexibiliteit biedt. Het is dan ook niet verwonderlijk dat dit pattern op veel plaatsen in de Java API gebruikt wordt. De meest bekende voorbeelden hiervan zien we natuurlijk in de AWT en Swing API, maar ook de servlet API komen we ze tegen.

Het Observer pattern gebruik je in je eigen code als je wilt bereiken dat er een 'bericht' wordt gestuurd naar geïnteresseerde partijen als er iets gebeurd. Bijvoorbeeld als je een verzameling gegevens bijhoudt in een collection en zodra gegevens worden toegevoegd, moet er opnieuw een berekening uitgevoerd worden. Een ieder die geïnteresseerd is in de nieuwe berekening kan zich registreren als listener en krijgt dan elke keer een bericht als er een nieuwe uitkomst is. Een directe implementatie van het Observer pattern bestaat ook in de Java API in de vorm van de class java.util.Observable en de interface java.util.Observer.

**VOORDELEN** Het grote voordeel van het Observer pattern is de ontkoppeling van verschillende belangen in de code. Aan de ene kant heb je het source object, dat als taak heeft om een bericht te versturen naar de listeners als zich een bepaalde situatie voordoet. Het observable object bepaalt het wanneer van een event. De listeners weten niet wanneer iets gebeurt, enkel dat als er iets gebeurt zij één of meerdere acties moeten uitvoeren. Verdere voordelen vloeien voort uit deze ontkoppeling, zo worden de afzonderlijke onderdelen beter test-




**Het listener principe wordt op verschillende plaatsen in Java gebruikt en wordt ook in de GoF beschreven onder de naam 'Observer'**

baar doordat zij geen kennis van elkaar hebben. Ook biedt deze oplossing meer mogelijkheden voor incrementele ontwikkeling van de code.

**TOEPASSEN EN HERKENNEN** Voor het gebruik van de Observer heb je de volgende zaken nodig:

- Event source (Observable) - het object dat de events gaat versturen en waar de listeners zich kunnen registreren en weer kunnen verwijderen:

```
package sollicitatie;
import java.util.Vector;
public class Werkgever {
  private Vector listeners = new Vector();
  public void addSollicitatieListener(Sollic
  itatieListener listen) {
     listeners.add(listen);
  }
  public void removeSollicitatieListener(Sol
  licitatieListener listen) {
     listeners.remove(listen);
  }
  public void stuurResultaat(String resul-
  taat) {
     final SollicitatieEvent event = new
     SollicitatieEvent(this, resultaat);
     new Thread(
        new Runnable() {
           public void run() {
              Vector listen = null;
              synchronized(listeners) {
              listen = (Vector)listeners.
clone();
              }
              for (int i=0; i<listen.size();
              i++) {
           ((SollicitatieListener)listen.
           get(i)).resultaat(event);
              }
           }
        }
     ).start();
  }
}
```

- Event listener interface - het contract tussen de informatiebron en de luisteraar:

```
package sollicitatie;
import java.util.EventListener;
public interface SollicitatieListener
extends EventListener {
  public void resultaat(SollicitatieEvent
  event);
}
```

- Event listener (Observer) - het object dat geïnteresseerd is in berichten van een specifiek type:

```
package sollicitatie;
public class Sollicitant implements
SollicitatieListener {
  public void resultaat(SollicitatieEvent
  event) {
     //doe iets met de informatie
  }
}
```

- Eventueel een event object als gegevens drager, met de data die van belang is voor deze gebeurtenis.

```
package sollicitatie;
import java.util.EventObject;
public class SollicitatieEvent extends
EventObject {
  private String resultaat;
  public SollicitatieEvent(Object source,
  String resultaat) {
     super(source);
     this.resultaat = resultaat;
  }
}
```

- Om de ontkoppeling compleet te maken kan ook van de Observable een interface gemaakt worden en dan kunnen verschillende Observables eenvoudig verwisseld worden.

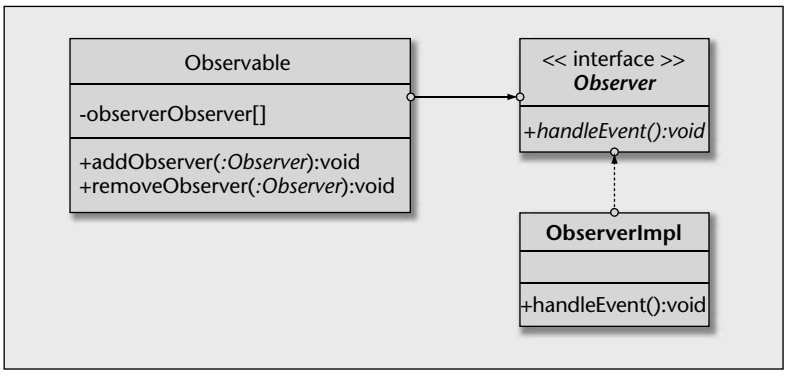
**VARIANT: CALLBACK** In een client-server omgeving kan de situatie dusdanig wijzigen dat het principe

## Literatuur


1) Design Patterns, Erich Gamma, et.al., Addison Wesley, 1995
2) Applied Java Patterns, Stephen Stelting & Olav Maassen, Prentice Hall, 2002


van de Observer nog steeds wenselijk en haalbaar is, maar dat de implementatie enige belangrijke wijzigingen moet ondergaan. Zo kan de server wel in Java geschreven zijn (servlet of JSP), maar de client niet (een browser). Het principe van Java classes die als listener fungeren werkt dan niet meer. Een oplossing hiervoor is vergelijkbaar met het geval van de sollicitanten wanneer een bedrijf niet reageert: regelmatig contact opnemen met de informatiebron om te kijken of er al meer informatie beschikbaar is (dit is ook bekend als 'Client polling' of 'Client pull'). Andere mogelijkheden zijn om te zoeken naar manieren waarop de server wel contact op kan nemen met de client. Als de hoofdreden voor deze constructie asynchrone verwerking is, kan messaging de uitkomst bieden. Op deze wijze kunnen zowel de client als de server asynchroon met elkaar communiceren. De messaging kan dan bovendien fungeren als een brug tussen verschillende technologieën.

**KEUZE** Bij zowel het Observer als bij het Callback patterns zul je moeten kiezen hoeveel informatie je per keer stuurt en hoe specifiek de informatie is die gestuurd wordt. Gespecialiseerde berichten zorgen ervoor dat slechts diegenen die de informatie willen hebben deze ook krijgen. Het nadeel hierbij is echter dat er meer verwacht wordt van de Observable, die moet meer rekenen of meer bijhouden. Algemene berichten zorgen er dan ook voor dat de Observable netjes en kleun blijft, maar dat de listeners meer moeten selecteren en filteren waardoor hun code gaat groeien.




**De Observer ontkoppelt het 'wanneer' en het 'wat' er gebeurd.**

**TENSLOTTE** In de volgende aflevering van deze reeks wordt gekeken hoe een programma zich beter kan gaan gedragen. Dan zijn het Command en Memento pattern aan de beurt.


*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

### HP stimuleert adoptie open source software met JBoss-overeenkomst

HP heeft een overeenkomst gesloten met JBoss, waarmee het voor grote organisaties gemakkelijker wordt om open source software te gebruiken. JBoss is leverancier van Java middleware-oplossingen op basis van open source software, waaronder applicatieservers, mailservers en een ontwikkelraamwerk. HP biedt klanten één contactpunt voor JBoss-ondersteuning en voor een reeks open source consultancy-diensten. HP maakt gebruik van partnerovereenkomsten met leveranciers van open source applicaties en bekende Linux-distributies om innovatieve oplossingen te ontwerpen, bouwen, integreren en te beheren. Daarnaast heeft HP wereldwijd ruim 6.500 in Linux gespecialiseerde service-medewerkers in dienst, om te kunnen voorzien in de behoeften van klanten op het gebied van consultancy en ondersteuning bij oplossingen. HP is de eerste en de enige grote Linux-leverancier met open source software-oplossingen die gecertificeerd zijn voor JBoss. De overeenkomst van vandaag zorgt voor additionele ondersteuning en diensten - architectuur en ontwerp, implementatie, en integratie- en migratie-oplossingen - voor JBoss-producten op de standaardplatforms van HP. Hiertoe behoren onder andere de HP ProLiant en de HP Integrity servers. Daarnaast is HP van plan om begin 2005 HP OpenView Application Management-oplossingen te bieden, die veranderingen in JBoss Application Serveromgevingen controleren en beheren. De overeenkomst met JBoss helpt tevens de risico's te minimaliseren voor klanten die open source-oplossingen willen inzetten in hun rekencentra, dankzij ondersteuning voor HP's Linux Reference Architecture. HP is de eerste en enige grote leverancier die Linux Reference Architectures biedt: een volledig geïntegreerde reeks oplossingen voor Linux, bestaande uit software, diensten en standaardhardware. Met behulp van deze architectuur kunnen klanten snel en effectief open source- en Linux-oplossingen inzetten. Meer informatie over de open source-oplossingen van HP is te vinden op www.hp.com/linux.