Design patterns

W. Oele

23 september 2014

Deze les

- Het decorator patroon

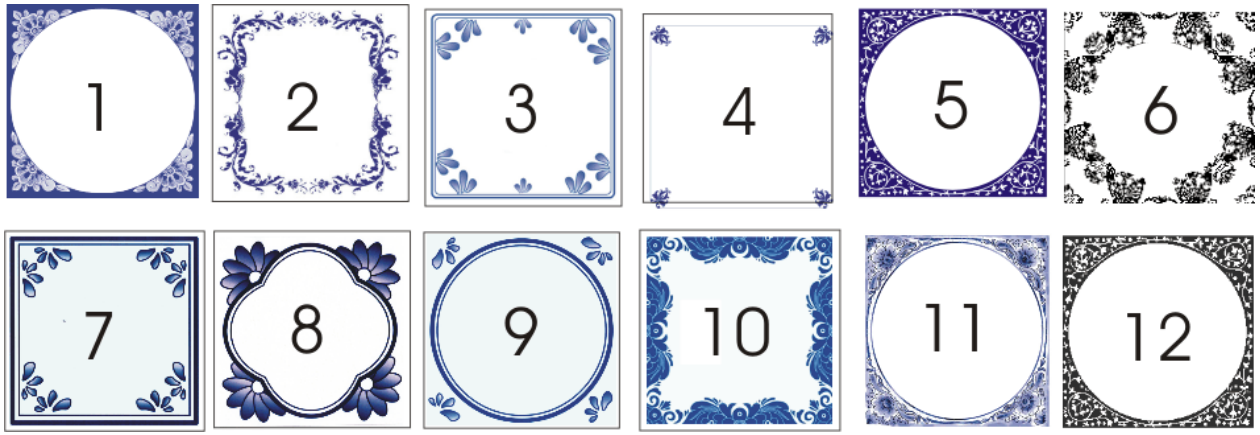
Het decorator pattern

- naam: het decorator pattern
- doel: extra verantwoorelijkheid toevoegen aan bestaande objecten en dit evt. dynamisch kunnen doen.
- hoe: object “inkapselen” in abstracte parentklasse en decorators polymorphistisch afleiden.
- gevolgen: extra verantwoordelijkheden kunnen dynamisch en in willekeurige volgorde worden toegevoegd.

Het probleem: tegels maken

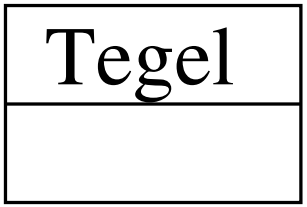
De basis tegel. . . wit

Diverse varianten



- met of zonder tekst
- met of zonder rand
- met of zonder versiersels in de hoeken
- extra glazuur
- etc.

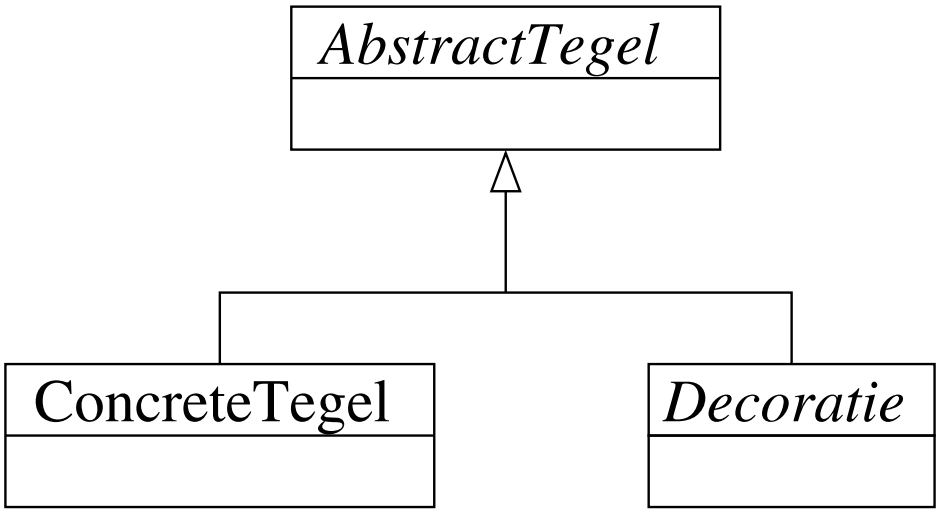
Het model



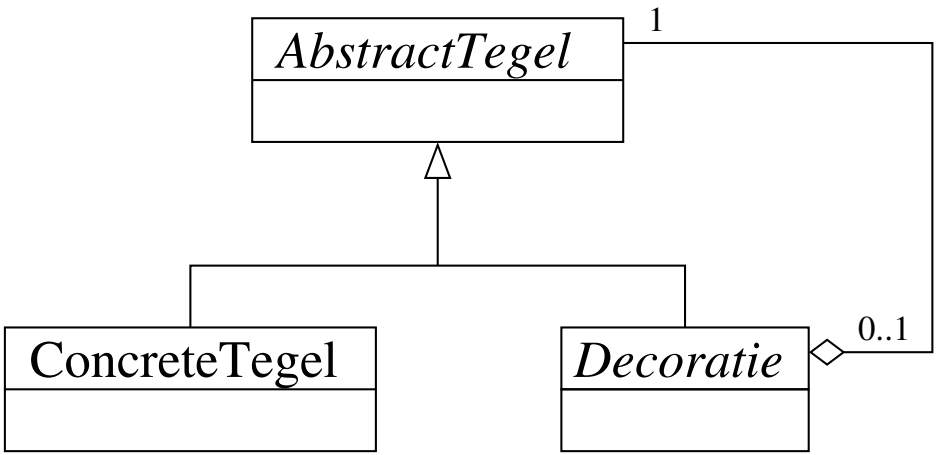
Verschillende “decoraties” kunnen door elkaar gebruikt worden:

- Een ketting van afgeleide klassen maken is dus geen goed idee.
- Polymorphisme is eveneens geen goed idee.

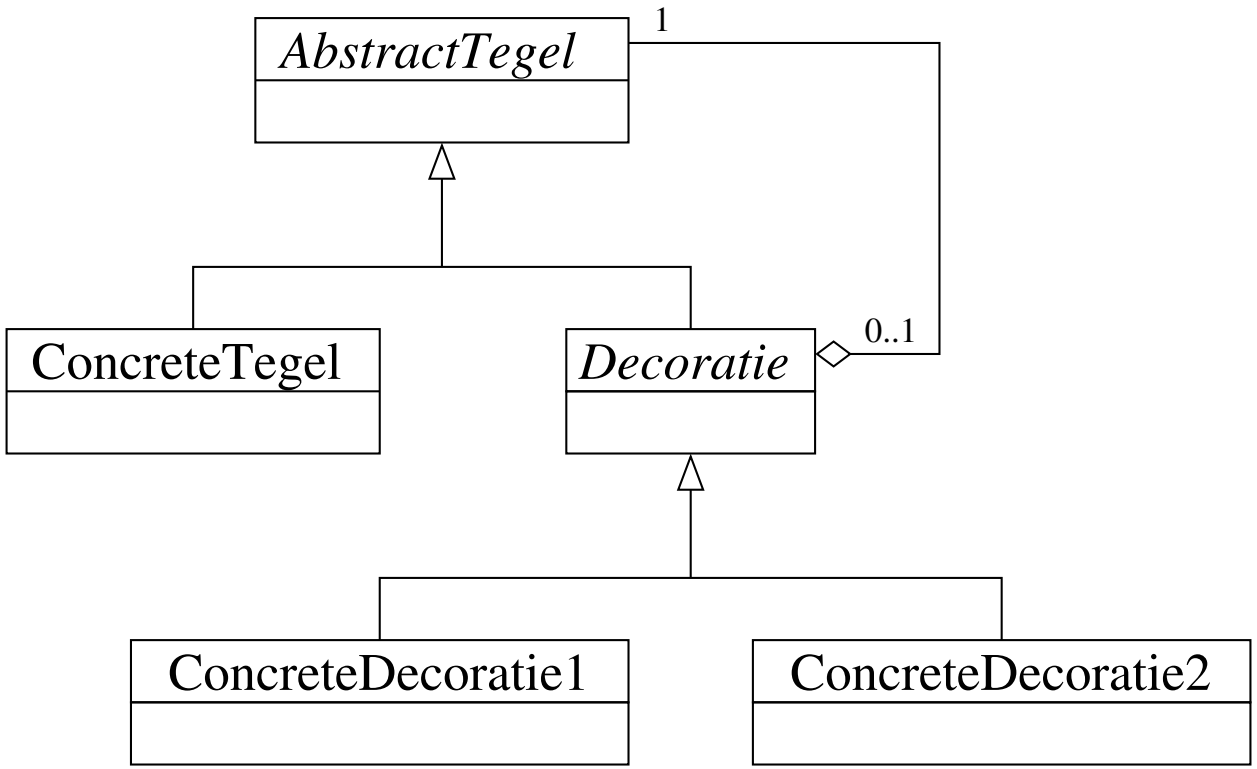
Het model



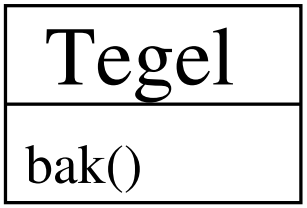
Het model



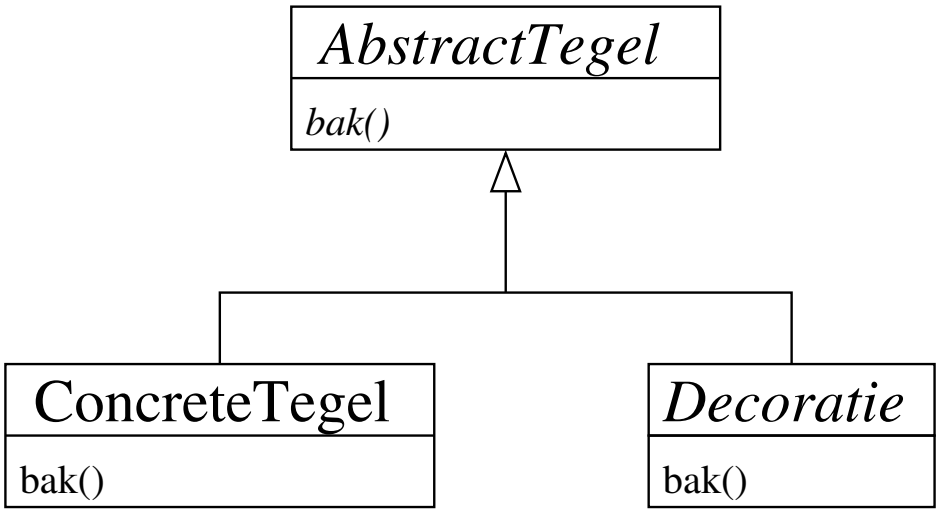
Het model


AbstractTegel
1
ConcreteTegel
Decoratie
0..1
ConcreteDecoratie1
ConcreteDecoratie2

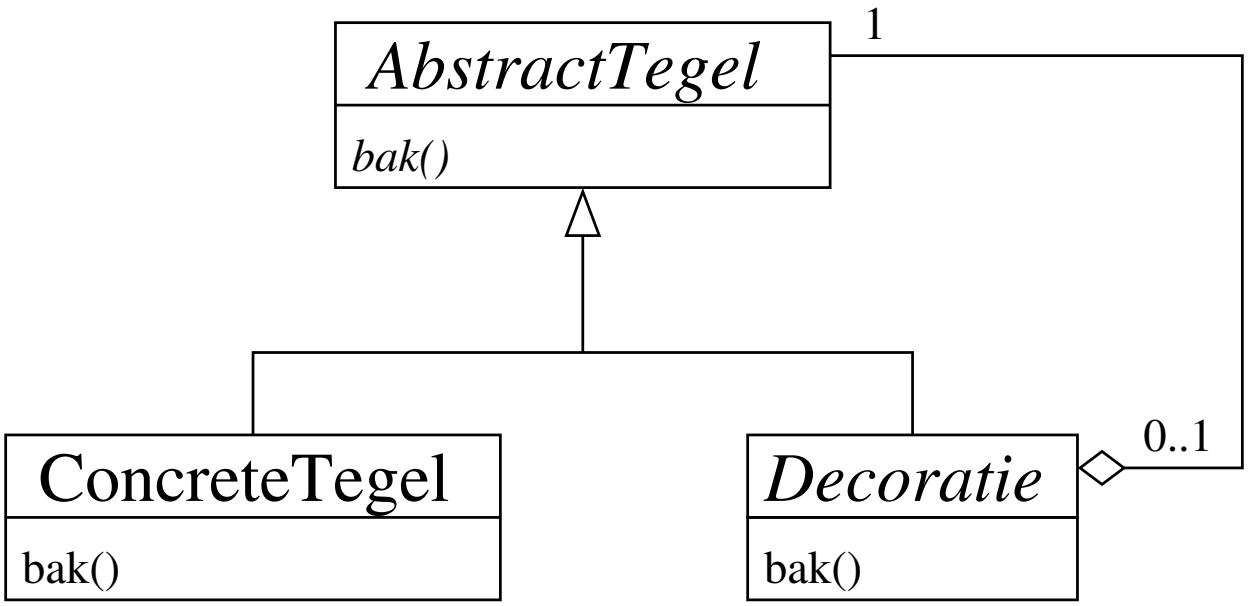
Methodes



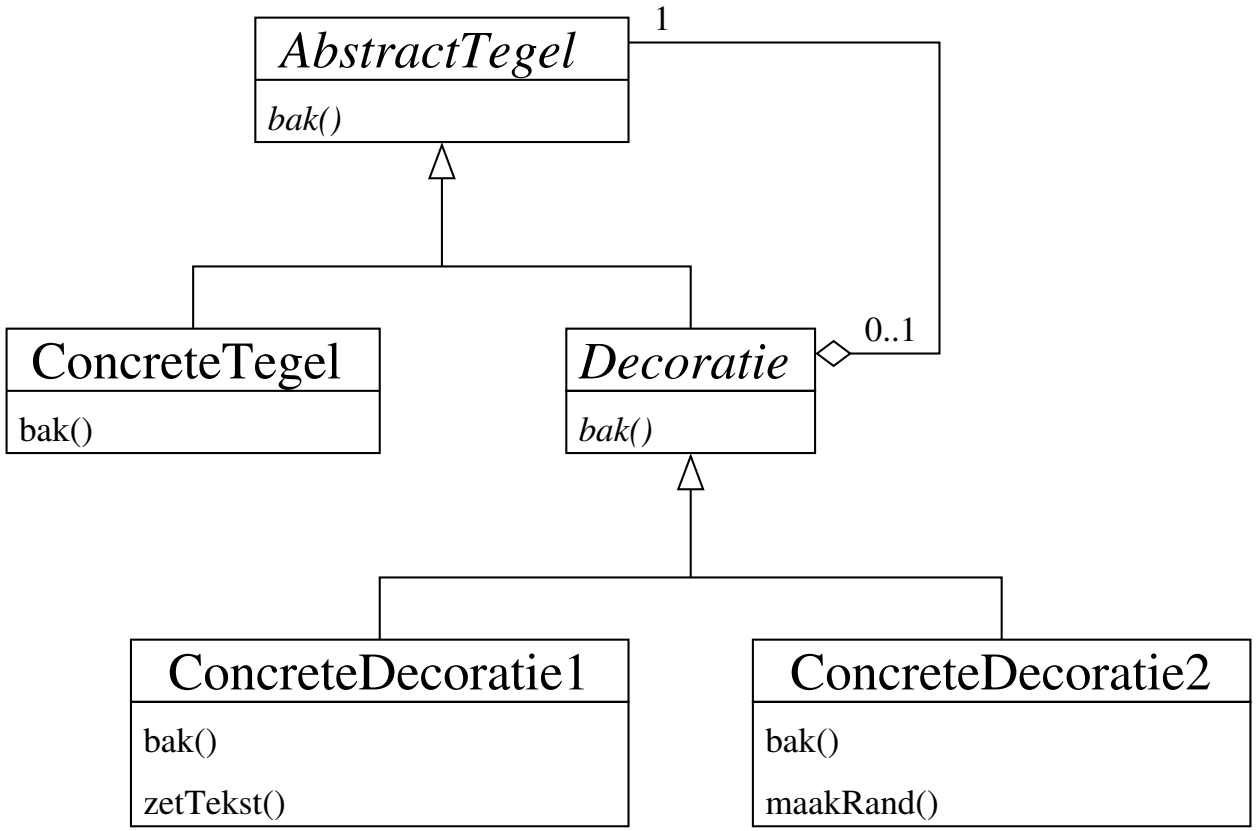
Methodes



Methodes



Methodes


AbstractTegel
bak()
1
ConcreteTegel
bak()
Decoratie
bak()
0..1
ConcreteDecoratie1
bak()
zetTekst()
ConcreteDecoratie2
bak()
maakRand()

Code

```
public abstract class AbstractTegel
{
    public abstract void bak();
}
```

Code

```
public class ConcreteTegel extends AbstractTegel
{
   @override
   public abstract void bak()
   {
      //standaard tegeltje bakken...
   }

}
```

Code

```
public abstract class Decoratie extends AbstractTegel
{
   private Tegel mijntegel;

   public Decoratie(AbstractTegel t)
   {
      mijntegel=t;
   }

   public void bakTegel()
   {
      if(mijntegel!=null)mijntegel.bak();
   }
}
```

Code

```
public class ConcreteDecoratie1 extends Decoratie
{
   public ConcreteDecoratie2(AbstractTegel t)
   {
      super(t);
   }

   public void bak()
   {
      super.bakTegel();
      //extra code voor deze decoratie
   }
}
```