COMPOUND PATTERNS

Veerle Ongenae, Wijnand Schepens

Compound Patterns

- Patterns van Patterns
- Samenwerkende patterns
  - Meerdere patterns combineren in één ontwerpoplossing
- Inhoud
  - Voorbeeld: eendenapplicatie met meerdere patterns
  - MVC: Model-View-Controller
- Een compound pattern combineert twee of meer patterns tot een oplossing die een terugkerend of algemeen probleem oplost.

Voorbeeld

- Eendensimulator herbouwen
  - Verschillende patterns naast elkaar gebruiken
- Eerste versie
  - Interface KanKwaken
  - Een aantal Eenden die KanKwaken implementeren
  - Simulator

Eendensimulator: 1e versie


```
public interface KanKwaken
{
     public void kwaak();
}
public class WildeEend implements KanKwaken
{
     public void kwaak() {
         System.out.println("Kwaak");
     }
}
public class BadEend implements KanKwaken
 {
     public void kwaak() {
         System.out.println("Piep");
     }
}
```


Eendensimulator: 1e versie


```
public static void main(String[] args)
{
    KanKwaken wildeEend = new WildeEend();
    KanKwaken badEend = new BadEend();

    simuleer(wildeEend);
    simuleer(badEend);
}

static void simuleer(KanKwaken eend)
{
    eend.kwaak();
}
```

Program to an interface!

Eendensimulator: 2e versie

- Ganzen toevoegen
- Hoe?

```
public class Gans
{
    public void gak()
    {
        System.out.println("Gak");
    }
}
```

Eendensimulator: 2e versie

7

```
public class GansAdapter implements KanKwaken
{
     Gans gans;
     public GansAdapter(Gans gans) {
          this.gans = gans;
     }

     public void kwaak() {
         gans.gak();
     }
}
```


Eendensimulator: 2e versie

8

```
public static void main(String[] args)
{
    KanKwaken wildeEend = new WildeEend();
    KanKwaken badEend = new BadEend();
    KanKwaken gansEend = new GansAdapter(new Gans());

    simuleer(wildeEend);
    simuleer(badEend);
    simuleer(gansEend);
}
```


Eendensimulator: 3e versie

9

- Tellen hoeveel keer eenden kwaken
- Zonder eendenklassen aan te passen
- Hoe?

Eendensimulator: 3e versie

10

```
public class KwaakTeller implements KanKwaken   // Decorator
{
    KanKwaken eend;
    static int aantalKwaken = 0;
    public KwaakTeller(KanKwaken eend)
    {
        this.eend = eend;
    }
    public void kwaak()
    {
        eend.kwaak();
        aantalKwaken++;
    }
    public static int getAantalKwaken()
    {
         return aantalKwaken;
    }
}
```


Eendensimulator: 3e versie

11

```
public static void main(String[] args)
{
   KanKwaken wildeEend = new KwaakTeller(new WildeEend());
   KanKwaken badEend = new KwaakTeller(new BadEend());
   KanKwaken gansEend = new GansAdapter(new Gans());

   simuleer(wildeEend);
   simuleer(badEend);
   simuleer(gansEend);

   System.out.println("De eenden kwaakten " +
              KwaakTeller.getAantalKwaken() + " keer");
   }
}
```


Eendensimulator: 4e versie

12

- Soms worden er eenden aangemaakt die niet gedecoreerd worden met de KwaakTeller
- Controleer (en scherm) het aanmaken en decoreren van eenden (af)
- Hoe?

Eendensimulator: 4e versie

13

```
public abstract class AbstracteEendenFabriek
{
    public abstract KanKwaken maakWildeEend();
    public abstract KanKwaken maakBadEend();
}
```


Eendensimulator: 4e versie

14

```
public class EendenFabriek extends AbstracteEendenFabriek
{
    public KanKwaken maakWildeEend()
    {
        return new WildeEend();
    }

    public KanKwaken maakBadEend()
    {
        return new BadEend();
    }
}
```


Eendensimulator: 4e versie

15

```
public class TelEendenFabriek extends AbstracteEendenFabriek
{
    public KanKwaken maakWildeEend()
    {
        return new KwaakTeller( new WildeEend() );
    }

    public KanKwaken maakBadEend()
    {
        return new KwaakTeller( new BadEend() );
    }
}
```


Eendensimulator: 4e versie

16

```
public static void main(String[] args) {
    AbstracteEendenFabriek fabriek = new TelEendenFabriek();
    simuleer(fabriek);
}

static void simuleer(AbstracteEendenFabriek fabriek)
{
    KanKwaken wildeEend = fabriek.maakWildeEend();
    KanKwaken badEend = fabriek.maakBadEend();

    // ...
}
```


Eendensimulator: 5e versie

17

- Alle eenden apart behandelen is lastig
- Families van eenden bekijken
- Een troep eenden als een geheel behandelen
- Hoe?

Eendensimulator: 5e versie

18

```
public class Troep implements KanKwaken // Composite
{
    ArrayList<KanKwaken> kwakers = new ArrayList<>();

    public void add(KanKwaken kwaker)
    {
        kwakers.add(kwaker);
    }
    public void kwaak()
    {
        for (KanKwaken kwaker : kwakers)
            kwaker.kwaak();
    }
}
```

Eendensimulator: 5e versie

19

```
public static void main(String[] args) { … }
     void simuleer(AbstracteEendenFabriek fabriek) {
          Troep troepEenden = maakTroep(fabriek);
          Troep troepWildeEenden = maakTroepWildeEenden(fabriek);
          troepEenden.add(troepWildeEenden);
          System.out.println("Eendensimulator");
          System.out.println("Volledige troep");
          simuleer(troepEenden);
          System.out.println("Troep wilde eenden");
          simuleer(troepWildeEenden);
          System.out.println("De eenden kwaakten " +
               KwaakTeller.getAantalKwaken() + " keer");
     }
```

```
Troep maakTroep(…){…}
Troep maakTroepWildeEenden(…){…}
void simuleer(KanKwaken eend) {
    eend.kwaak();
}
```


Eendensimulator: 6e versie

20

- Individuele eenden volgen
- Hoe?

Eendensimulator: 6e versie

21

- Individuele eenden volgen
- Hoe?

Eendensimulator: 6e versie

22

```
public interface KwaakObserver
{
    public void update( KwaakObservable bron);
}
public interface KwaakObservable
{
    public void registerObserver(KwaakObserver o);
    public void notifyObservers();
}
public interface KanKwaken extends KwaakObservable
{
    public void kwaak();
}
```


Eendensimulator: 6e versie

23

```
public class StandaardObservable implements KwaakObservable
{
    LinkedHashSet<KwaakObserver> observers = new LinkedHashSet<>();

    public void registerObserver(KwaakObserver o)
    {
        observers.add(o);
    }
    public void notifyObservers()
    {
        for (KwaakObserver o : observers)
            o.update( this );
    }
}
```


Eendensimulator: 6e versie

24

```
public class WildeEend extends StandaardObservable implements KanKwaken
{
     public void kwaak()
     {
         System.out.println("Kwaak");
         notifyObservers();
     }
}
public class Kwakoloog implements KwaakObserver
{
     public void update(KwaakObservable eend)
     {
         System.out.println("Kwakoloog: " + eend + " heeft net gekwaakt");
     }
}
```

Troep observeren?

25

Samenvatting

26

- Start: een hoop KanKwaken
- Er kwam een gans langs …
  - Adapter
- Gesnater tellen
  - Decorator
- Bang om decorator te vergeten
  - Abstract Factory
- Beheren eenden, ganzen, … in een troep
  - Composite en Iterator
- Bericht van een individuele eend
  - Observer

Opmerkingen

27

- Eendenapplicatie
  - Een stel patterns dat samenwerkt
  - Geen compound pattern
- Beetje geforceerd
  - Illustratie
- Overzicht p. 511

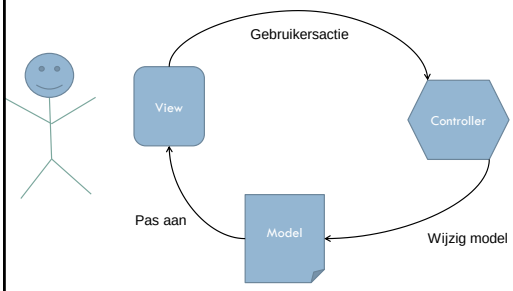
MVC als Compound Pattern

28

- Voorbeeld
  - Mp3-speler
- Interface gebruiken
  - Songs toevoegen
  - Playlist maken
  - …
- Speler
  - Database met gegevens
  - Afspelen song → bijwerken gebruikersinterface (titel, tijd, …)

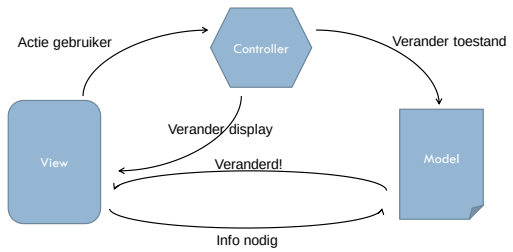
MP3 als MVC

29



MVC grafisch

30



MVC

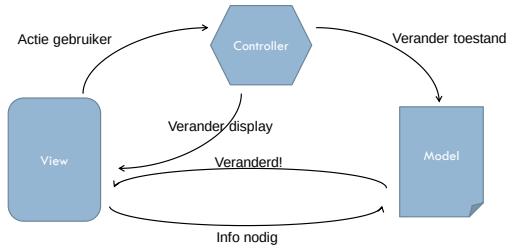
31

- Model
  - Gegevens
  - Toestand
  - Logica
  - Niet bewust van view of controller
- View
  - Presentatie model
- Controller
  - Input gebruiker → actie model

MVC patterns

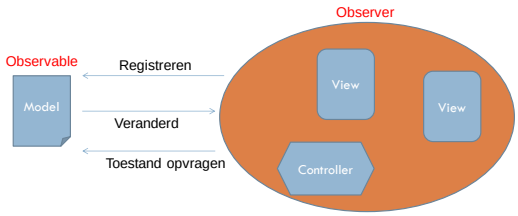
32

- Welke patterns in MVC?



Observer

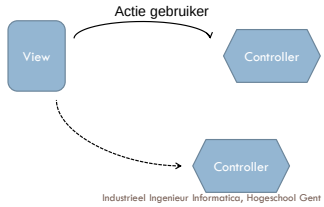
33



Strategy

34

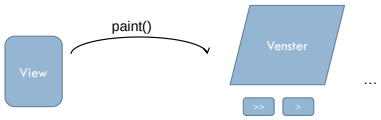
- View delegeert afhandelen gebruikersacties
- Ander gedrag: controller veranderen



Composite

35

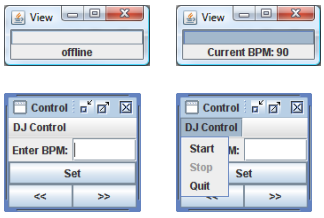
- Compositie van GUI-componenten



MVC - voorbeeld

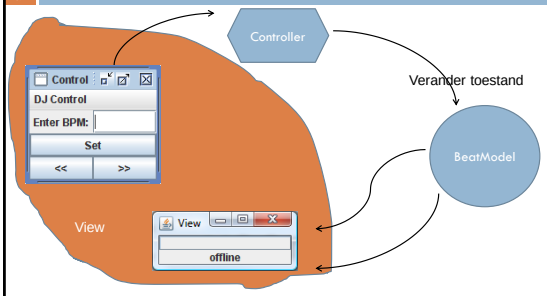
36

- DJ: drumbeat



MVC – structuur voorbeeld

37



Model

38

- BeatModelInterface
- BeatModel

View

39

- DJView
  - 2 frames
  - BeatObserver
  - BMPObserver
- BeatBar

Controller

40

- ControllerInterface
- BeatController

Strategy in MVC

41

- Hartslag tonen in DJView
- HeartModelInterface
- HeartModel

BeatModelInterface
initialize() on() off() setBPM() getBPM() registerObserver(); removeObserver() registerObserver() removeObserver()

HeartModelInterface
getHeartRate() registerObserver() removeObserver() registerObserver() removeObserver()

HeartModel aanpassen aan BeatModel

42

- Adapter
- HeartAdapter

Samenvatting kennis

- OO-principes
  - Isoleer wat verandert
  - Verkies compositie boven overerving
  - Programmeer naar een interface en niet naar een implementatie
  - Streef naar ontwerpen met een zwakke koppeling tussen interagerende objecten
  - Klassen moeten open zijn voor uitbreiding maar gesloten voor verandering
  - Wees afhankelijk van abstracties, niet van concrete klassen
  - Hoe minder je moet weten, hoe beter
  - Bel ons niet, wij bellen jou
  - Een klasse dient maar één reden te hebben om te veranderen
- OO-patterns
  - Strategy, Observer, Decorator, Abstract Factory, Factory Method, Command Pattern, Adapter, Facade, Template Method Pattern, Iterator Pattern, Composite Pattern, State Pattern, Proxy Pattern
  - Het Compound Pattern combineert twee of meerder patterns in een oplossing die een algemeen of terugkerend probleem oplost.