HET STATE PATTERN

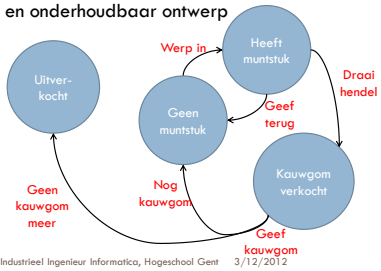
Veerle Ongenae, Wijnand Schepens

Het State Pattern

- Controle gedrag objecten
- Veranderen interne toestand objecten
- Gelinkt met het Strategy Pattern

Kauwgomautomaat

- Toekomst: mogelijk nieuw gedrag
  - Flexibel en onderhoudbaar ontwerp



Kauwgomautomaat

- Toestandsdiagram
  - Toestand
    - Verschillende configuraties
  - Toestandsovergang
    - Actie
    - Niet elke actie kan in elke toestand

Eerste implementatie

- Mogelijke toestanden + huidige toestand

```
final static int UITVERKOCHT = 0;
final static int GEEN_MUNT = 1;
final static int HEEFT_MUNT = 2;
final static int NET_VERKOCHT = 3;
int status = UITVERKOCHT;
```

- Acties
  - Externe interface
  - Interne acties

```
public void werpMuntIn()
{
     if (status == HEEFT_MUNT)
          … // kan niet
     else if (status == UITVERKOCHT)
          … // kan niet
     else if (status == NET_VERKOCHT)
          … // even wachten
     else if (status == GEEN_MUNT) {
          status = HEEFT_MUNT;
          … // berichtje tonen
     }
}
```


Eerste implementatie


```
public class KauwgomMachine
{
     final static int UITVERKOCHT = 0;
     final static int GEEN_MUNT = 1;
     final static int HEEFT_MUNT = 2;
     final static int NET_VERKOCHT = 3;
     int status = UITVERKOCHT;
     int aantal = 0;
     public KauwgomMachine(int aantal) {
          this.aantal = aantal;
          if (aantal > 0)
               status = GEEN_MUNT;
     }
     public void werpMuntIn() {…}
     public void geefMuntTerug() {…}
```

Eerste implementatie (vervolg)

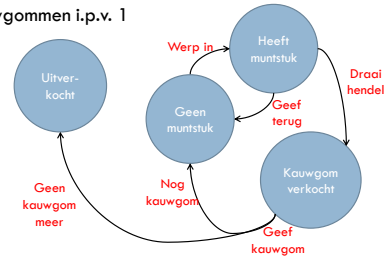
7

```
public void draaiHendel() {
    if (status == HEEFT_MUNT) {
        status = NET_VERKOCHT; geefKauwgom();
    }
    else if (status == UITVERKOCHT)
        … // kan niet
    else if (status == NET_VERKOCHT)
        … // 2 keer → fout
    else if (status == GEEN_MUNT)
        … // kan niet
}
public void geefKauwgom() {
    if (status == NET_VERKOCHT) {
        aantal--;
        if (aantal == 0) { status = UITVERKOCHT; }
        else { status = GEEN_MUNT; }
    }
    else if …
```

Verandering!

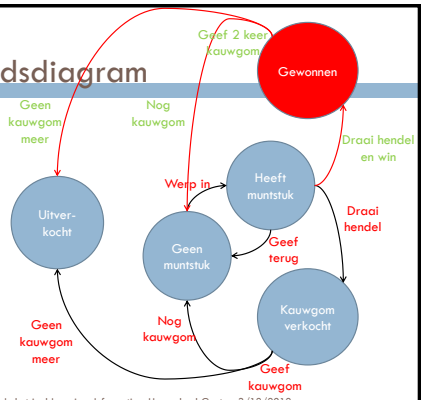
8

- In 10% van de gevallen
  - 2 kauwgommen i.p.v. 1



Nieuw toestandsdiagram

9



Makkelijk aanpasbaar???

10

- Gezondigd tegen welke OO-ontwerpprincipes?

Moeilijk aanpasbaar

11

- Open-gesloten principe?
- Niet erg OO
- Toestandsovergangen
  - Niet expliciet
  - Verborgen in if/else-structuren
- Isoleer wat varieert?
  - Isoleer gedrag van iedere toestand
- Verdere uitbreidingen
  - Fouten in code

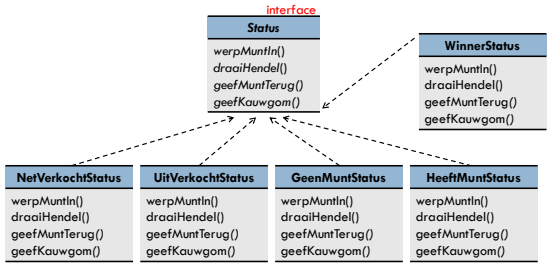
Nieuw ontwerp

12

- Gedrag van een toestand isoleren (in een klasse plaatsen)
- Actie delegeren aan huidige toestand
- Hoe?
  - Interface Status
    - Methode voor elke actie
  - Toestandsklasse voor elke toestand van de machine
    - Gedrag in bepaalde toestand
  - If/else –structuren weg
    - Werk delegeren aan toestand

Interfaces en klassen voor toestand

13



Nieuwe implementatie: status

14

```
public class GeenMuntStatus implements Status
{
    KauwgomMachine kauwgomMachine;
    public GeenMuntStatus(KauwgomMachine kauwgomMachine) {
        this.kauwgomMachine = kauwgomMachine;
    }
    public void werpMuntIn() {
        kauwgomMachine.setStatus(kauwgomMachine.getHeeftMuntStatus());
        … // munt ingeworpen
    }
    public void geefMuntTerug() { … // eerst inwerpen }
    public void draaiHendel() { … // eerst inwerpen }
    public void geefKauwgom() { … // eerst betalen }
}
```


Nieuwe implementatie: machine

15

```
public class KauwgomMachine
{
    Status uitVerkochtStatus, geenMuntStatus,
           heeftMuntStatus, netVerkochtStatus;
    Status status; int aantal = 0;
    public KauwgomMachine(int aantal)
    {
        this.aantal = aantal;
        uitVerkochtStatus = new UitVerkochtStatus(this);
        geenMuntStatus = new GeenMuntStatus(this);
        heeftMuntStatus = new HeeftMuntStatus(this);
        netVerkochtStatus = new NetVerkochtStatus(this);
        if (aantal > 0)
            status = geenMuntStatus;
        else
            status = uitVerkochtStatus;
    }
```

Nieuwe implementatie: machine

16

```
public class KauwgomMachine
{
    Status uitVerkochtStatus, geenMuntStatus, heeftMuntStatus, netVerkochtStatus;
    Status status; int aantal = 0;
    public KauwgomMachine(int aantal) {…}
    public void werpMuntIn() { status.werpMuntIn(); }
    public void geefMuntTerug() { status.geefMuntTerug(); }
    public void draaiHendel() {
        status.draaiHendel();
        status.geefKauwgom();
    }
    void setStatus(Status status) { this.status = status; }
    void neemKauwgom() { // hulpmethode
        if (aantal != 0) {aantal--;}
    }
    …
```


Nieuwe implementatie: andere statussen

17

```
public class HeeftMuntStatus implements Status
{
    KauwgomMachine kauwgomMachine;
    public HeeftMuntStatus(KauwgomMachine kauwgomMachine) {
        this.kauwgomMachine = kauwgomMachine;
    }
    public void werpMuntIn() { … // al ingeworpen }
    public void geefMuntTerug() {
        kauwgomMachine.setStatus(kauwgomMachine.getGeenMuntStatus();
        … // munt terug
    }
    public void draaiHendel() {
        kauwgomMachine.setStatus(kauwgomMachine.getNetVerkochtStatus();
        … // gedraaid }
    public void geefKauwgom() { … // niet van toepassing}
}
```


Nieuwe implementatie: andere statussen

18

```
public class NetVerkochtStatus implements Status {
    KauwgomMachine kauwgomMachine;
    public NetVerkochtStatus(KauwgomMachine kauwgomMachine) {
        this.kauwgomMachine = kauwgomMachine;
    }
    public void werpMuntIn() { … // even wachten }
    public void geefMuntTerug() { … // kan niet, al gedraaid }
    public void draaiHendel() { … // twee keer draaien zinloos}
    public void geefKauwgom() {
        kauwgomMachine.neemKauwgom();
        if (kauwgomMachine.getAantal() > 0) {
            kauwgomMachine.setStatus(kauwgomMachine.getGeenMuntStatus();
        } else {
            kauwgomMachine.setStatus(kauwgomMachine.getUitVerkochtStatus();
            … // leeg }
    } }
```

Nog een status: UitVerkochtStatus

19

- Implementatie?

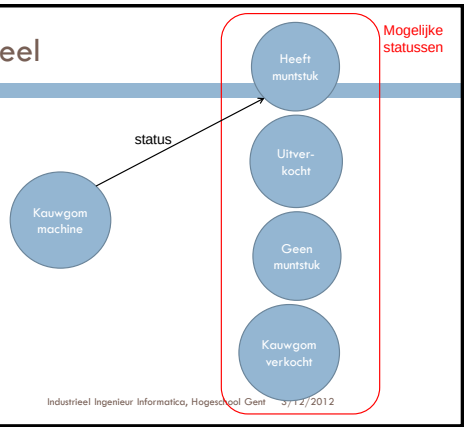
Wat is er veranderd?

20

- Gedrag elke toestand in aparte klasse
- If/else-structuren
  - Foutgevoelig
  - Verwijderd
- Iedere toestand gesloten voor verandering
- Kauwgommachine open voor uitbreiding
  - Nieuw toestanden
- Structuur code en klassen dichter bij diagram

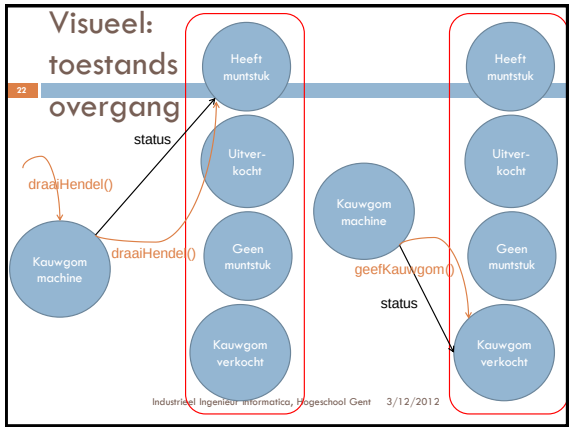
Visueel

21



Visueel: toestands overgang

22



Oefening

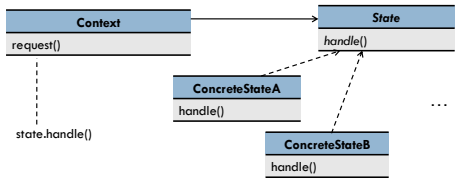
23

- p. 395

State Pattern Definitie

24

- Het State Pattern maakt het voor een object mogelijk zijn gedrag te veranderen wanneer zijn interne toestand verandert. Het object lijkt van klasse te veranderen.



State Pattern ↔ Strategy Pattern

25

- Zelfde klassendiagram
- Andere intenties
  - State
    - Gedrag afgeschermd in toestandsobjecten
    - Huidige toestand doorloopt verzameling toestandsobjecten
    - Client weet weinig of niets over toestandsobjecten
    - Alternatief voor veel conditionele statements in context
  - Strategy
    - Client specifieert samenstelling context + strategy-object
    - Vaak één strategy-object meest geschikt voor een context
    - Flexibel alternatief voor overerving

Opmerkingen

26

- Toestandsovergangen
  - Context beslist
    - Vaste toestandsovergangen
  - Toestandsobjecten beslissen
    - Dynamischere toestandsovergangen
    - Mogelijk veel afhankelijkheden
- Clients communiceren niet met toestandsobjecten
- Toestandsobjecten kunnen gedeeld worden
  - Static attribuut
  - Bij actie ook referentie naar de context meegeven

Kans op extra kauwgom toevoegen

27

```
public class KauwgomMachine {
    Status uitVerkochtStatus, geenMuntStatus, heeftMuntStatus,
  netVerkochtStatus;
    Status winnerStatus;
    Status status; int aantal = 0;
    … // extra get-methode
}
```


Extra status

28

```
public class WinnerStatus implements Status {
    …
    public void werpMuntIn() { … // even wachten }
    public void geefMuntTerug() { … // kan niet, al gedr }
    public void draaiHendel() {  … // twee keer draaien zinloos}
    public void geefKauwgom() {
        kauwgomMachine.neemKauwgom();
        if (kauwgomMachine.getAantal() == 0)
            kauwgomMachine.setStatus(kauwgomMachine.getUitVerkochtStatus();
        else {
            … // gewonnen
            kauwgomMachine.neemKauwgom();
            if (kauwgomMachine.getAantal() > 0)
                kauwgomMachine.setStatus(kauwgomMachine.getGeenMuntStatus();
            else {
                kauwgomMachine.setStatus(kauwgomMachine.getUitVerkochtStatus();
                  …  // leeg }    } }
```

HeeftMuntStatus aanpassen

29

```
public class HeeftMuntStatus implements Status
{
    Random randomWinner = new Random(System.currentTimeMillis());
    …
    public void draaiHendel() {
        int winner = randomWinner.nextInt(10);
        if ((winner == 0) && (kauwgomMachine.getAantal() > 1)
            kauwgomMachine.setStatus(kauwgomMachine.getWinnerStatus();
        else
            kauwgomMachine.setStatus(kauwgomMachine.getNetVerkochtStatus();

        … // gedraaid
    }
}
```

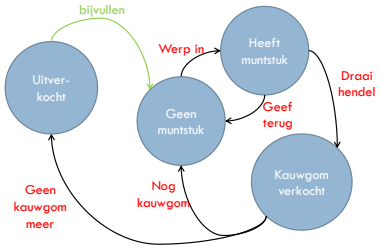

Verbeteringen?

30

- Codeduplicatie in NetVerkochtStatus en WinnerStatus
- Methode geefKauwgom wordt altijd opgeroepen

Bijvullen …

31



Oefening

32

- p. 408

Samenvatting kennis

33

- 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
  - Het State Pattern maakt het mogelijk voor een object om zijn gedrag te veranderen wanneer zijn interne toestand verandert. Het lijkt alsof het object zijn klasse verandert.