DE ADAPTER EN FACADE PATTERNS

Veerle Ongenae, Wijnand Schepens

De Adapter en Facade Patterns

- Wees adaptief
  - Vierkante plug in een rond gat
  - Objecten inpakken
  - Andere interfaces

Adapter in het gewone leven

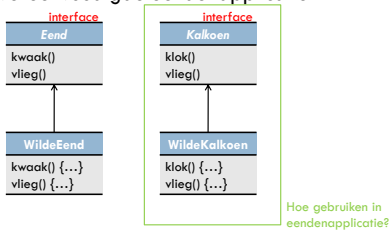
- Belgisch stopcontact in Britse contactdoos?
  - Gebruik een adapter
- Adapter
  - Verandert de interface van de contactdoos
  - Verandering van vorm
  - Verandering van spanning

Adapter OO

- Bestaand systeem
  - Gebruikt een interface
- Koppelen aan
- Nieuwe bibliotheek
  - Heeft een interface
- Beide interfaces zijn niet dezelfde
- Hoe koppelen?
  - Zonder bestaande code te wijzigen
  - Maak een adapter
    - Aanpassen interface bibliotheek aan verwachte interface systeem

Voorbeeld

- Vereenvoudigde eendenapplicatie



Adapter: kalkoen → eend


```
public class KalkoenAdapter implements Eend {
    Kalkoen kalkoen;
    public KalkoenAdapter(Kalkoen kalkoen) {
        this.kalkoen = kalkoen;
    }
    public void kwaak() {
        kalkoen.klok();
    }
    public void vlieg() {
        for (int i = 0; i < 5; i++) {
            kalkoen.vlieg();
        }
    }}
```

Gebruik adapter

7

```
public class TestEendenApplicatie{
    public static void main(String[] args) {
        Eend eend = new WildeEend();
        Kalkoen kalkoen = new WildeKalkoen();
        Eend kalkoenAdapter = new KalkoenAdapter(kalkoen);
        testEend(kalkoenAdapter);
        testEend(Eend);
    }
    static void testEend(Eend eend) {
        eend.kwaak();
        eend.vlieg();
    }
}
```


Adapter: eend → kalkoen

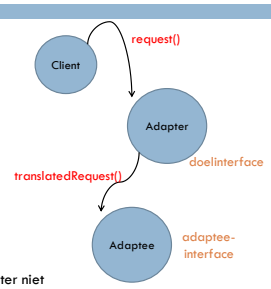
8

- Hoe?

Het Adapter Pattern

9

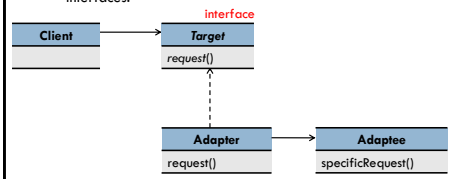
- Client
  - Aanvraag aan adapter
  - Methode doelinterface
- Adapter
  - Vertaalt aanvraag
  - Één of meerdere aanroepen
    - Aan adaptee
    - Adaptee-interface
- Client
  - Ontvangt resultaten
  - Kent het bestaan van de adapter niet



Definitie Adapter Pattern

10

- Het Adapter Pattern converteert de interface van een klasse naar een andere interface die de client verwacht. Adapters zorgen ervoor dat klassen samenwerken. Zonder adapters lukt dit niet vanwege incompatibele interfaces.



Opmerkingen

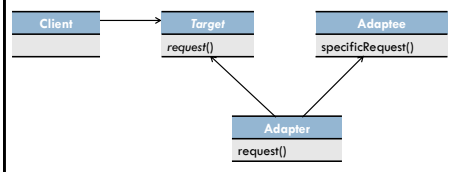
11

- Ontkoppeling
  - Client
  - Adaptee-interface
    - Ingekapseld door adapter
    - Kan veranderen zonder de client te beïnvloeden
- Gebruikte OO-ontwerpprincipes
  - Verkies compositie boven overerving
  - Programmeer naar interface, niet naar implementaties
- Adapter kan ook rond meerdere adaptee's
- Client gebruikt oude en nieuwe interface
  - Adapter ondersteunt beide

Object- en klassenadapters

12

- Twee soorten adapters
  - Objectadapters
  - Klassenadapters
    - Maakt gebruik van meervoudige overerving



Objectadapter ↔ Klassenadapter

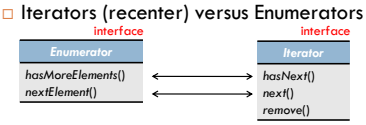
13

- Compositie
  - Ook bruikbaar voor afgeleide klassen van Adaptee
- Flexibel
- Overerving
  - Minder implementatie
- Efficiënt
  - Minder objecten

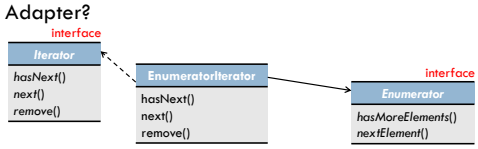
Adapters in de praktijk

14

- Iterators (recenter) versus Enumerators (ouder)



- Adapter?



remove() ?

15

- Enumerator
  - Niet ondersteund
- Interator
  - remove() kan een UnsupportedOperationException gooien
  - Niet perfect, maar aanvaardbaar

Objecten omwikkelen

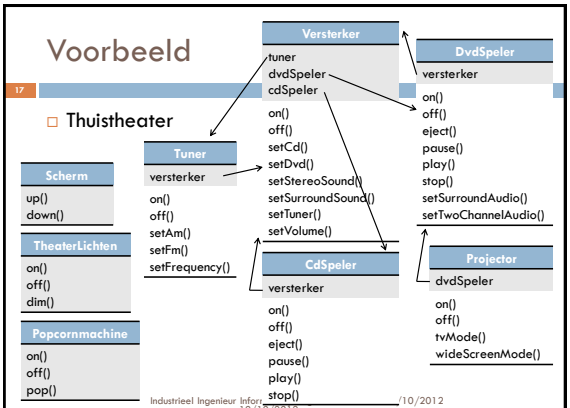
16

- Adapter
  - Interfaces converteren
- Facade
  - Interface vereenvoudigen
  - Complexiteit verbergen

Voorbeeld

17

- Thuistheater



Film bekijken

18

- Popcornmachine aanzetten
- Popcornmachine opstarten
- Verlichting dimmen
- Scherm neerlaten
- Beamer aanzetten
- Beamer op dvd instellen
- Beamer op breedbeeld instellen
- Versterker aanzetten
- Versterker op dvd input instellen
- Versterker op surround sound instellen
- Het volume van de versterker instellen
- Dvd-speler aanzetten
- Dvd-speler starten

Code film bekijken

19

```
popcornMachine.on();
popcornMachine.pop();

lichten.dim(10);

scherm.down();

projector.on();
projector.wideScreenMode();

versterker.on();
versterker.setDvd(dvd);
versterker.setSurroundSound();
versterker.setVolume(5);

dvd.on();
dvd.play(film);
```


Nog meer te doen

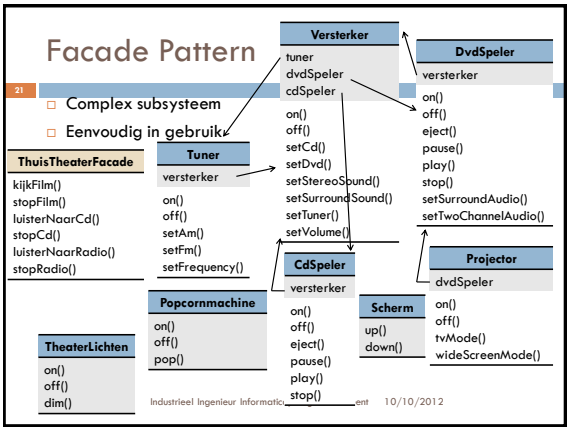
20

- Na film
  - Alles uitzetten?
- Cd luisteren
- Radio luisteren
- Systeem upgraden, afwijkende procedures?

Facade Pattern

21

- Complex subsysteem
- Eenvoudig in gebruik



Werkwijze facade

22

- Facade klasse ThuisTheaterFacade
  - Eenvoudige methodes
- Componenten thuistheater
  - Subsysteem
  - Methodes gebruikt door facade
- Client
  - Methodes facade
- Subsysteem blijft toegankelijk

Opmerkingen

23

- Facade kan ook nieuwe functionaliteit toevoegen
- Meerder facades mogelijk voor één subsysteem
- Ontkoppeling
  - Client
  - Subsysteem

ThuisTheaterFacade - code

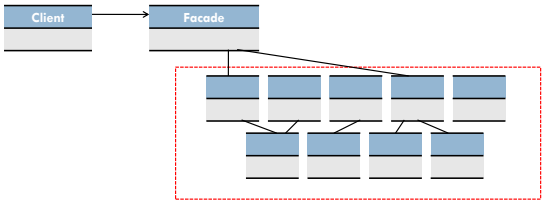
24

```
public class ThuisTheaterFacade {
    Versterker versterker;
    Tuner tuner;
    DvdSpeler dvd;
    CdSpeler cd;
    Projector projector;
    TheaterLichten lichten;
    Scherm scherm;
    PopcornMachine popcornMachine;

    public ThuisTheaterFacade(…) {
        this.versterker = versterker;
        … }

public void kijkFilm(String film) {
    popcornMachine.on();
    popcornMachine.pop();
    lichten.dim(10);
    scherm.down();
    projector.on();
    projector.wideScreenMode();
    versterker.on();
    versterker.setDvd(dvd);
    versterker.setSurroundSound();
    versterker.setVolume(5);
    dvd.on();   dvd.play(film); }
```

Definitie Facade Pattern

25

- Het Facade Pattern zorgt voor een vereenvoudigde interface naar een verzameling interfaces in een subsysteem. De facade definieert een interface op een hoger niveau zodat het gebruik van het subsysteem vereenvoudigt.



Facade Pattern in Java API

26

- Voorbeelden?

Principe van de kennisabstractie (Principle of Least Knowledge)

27

- Ontwerpprincipe
  - Praat alleen met je directe vrienden. Hoe minder je weet hoe beter.
- Bij ontwerp
  - Met welke klassen interageert de klasse? Waarom?
  - Vermijd ontwerpen met veel gekoppelde klassen
    - Invloed veranderingen!
  - Veel koppelingen
    - Veel afhankelijkheden
    - Moeilijk onderhoudbaar
    - Moeilijk te begrijpen

Voorbeeld

28

- Veel koppeling

```
public float getTemp() {
    return station.getThermometer().getTemperature();
}
```

- Minder koppeling

```
public float getTemp() {
    return station.getTemperature();
}
```


Principe kennisabstractie: hoe?

29

- Richtlijnen
  - Methodes oproepen in een object
    - Van het object zelf
    - Van parameters van de methode
    - Van het object dat de methode creëert of instantieert
    - Van een component van het object
- Andere naam
  - Wet van Demeter
- Voor- en nadelen
  - Minder afhankelijkheden
  - Beter softwareonderhoud
  - Meer wrapperklassen
    - Complexer
    - Minder performant

Voorbeeld

30

```
public class Auto {
   Motor motor; …

   public Auto() {…}

   public void pasDashboardSchermAan() {
      …
   }
```

```
   public void start(Sleutel sleutel) {
      Deuren deuren = new Deuren();
      boolean toegang = sleutel.draait();

      if (toegang) {
         motor.start();
         pasDashboardSchermAan();
         deuren.sluit();
      }
   }
```

Samenvatting kennis

31

- OO-basis
- 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
- OO-patterns
  - Strategy, Observer, Decorator, Abstract Factory, Factory Method, Command Pattern
  - Het Adapter Pattern converteert de interface van een klasse in een andere interface die door clients verwacht wordt. Laat klassen samenwerken die dat anders niet konden vanwege incompatibele interfaces.
  - Het Facade Pattern zorgt voor een uniforme interface naar een verzameling interfaces in een subsysteem. De facade definieert een interface op een hoger niveau waardoor je een subsysteem eenvoudiger kunt gebruiken.