HET COMMAND PATTERN

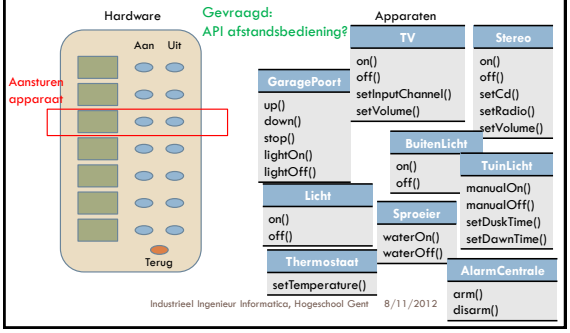
Veerle Ongenae, Wijnand Schepens

Het Command Pattern

- Inkapseling op een nieuw niveau
  - Afschermen methode-aanroepen
- Aanroepende object hoeft niet te weten wat de verschillende uitvoeringsstappen zijn

Voorbeeld

Aan te sturen apparaten
Veel klassen
Geen gemeenschappelijke interface
Nieuwe klassen te verwachten



Ontwerp API afstandsbediening

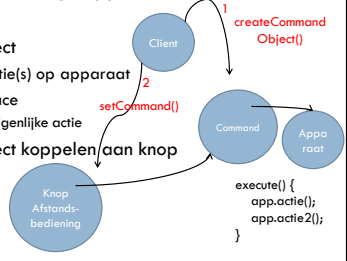
- Afstandsbediening
  - Interpreteren drukken op knop
  - Niets weten over leverancierssklassen (apparaten)
  - Aanvraag actie
- Apparaten
  - Bediening
  - Uitvoeren actie

Principe Command Pattern

- Commando-objecten
  - Voeren actie op apparaat uit
- Knop gekoppeld aan commando-object
- Commando-object
  - Schermt eigenlijke actie af
- Losse koppeling
  - Afstandsbediening
  - Eigenlijke apparaat

Command Pattern en afstandsbediening

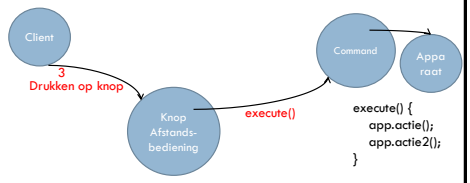
- Knop afstandsbediening koppelen aan functie apparaat
  - Commando-object
    - Aanroepen functie(s) op apparaat
    - Uniforme interface
      - Afschermen eigenlijke actie
  - Commando-object koppelen aan knop
    - setCommand



Command Pattern en afstandsbediening

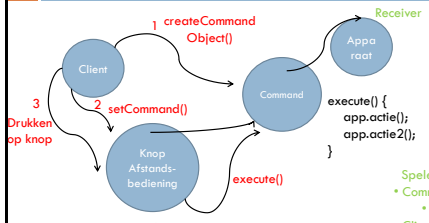
7

- Later … drukken op knop afstandsbediening
  - Aanroepen execute()-methode Commando-object
    - Aanroepen functie op apparaat



Command Pattern en afstandsbediening

8



Spelers Command Pattern
- Command
  - execute()
- Client
- Invoker
  - setCommand()
- Receiver

Code Command-object

9

```
public interface Command {
    public void execute();
}

public class LichtAanCommand implements Command {
    Licht licht;
    public LichtAanCommand(Licht licht) {
        this.licht = licht;
    }
    public void execute() {
        licht.on();
    }
}
```

Interface voor alle Commando-objecten

Licht
on() off()

Code Afstandsbediening en Client

10

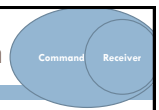
```
public class EenvoudigeAfstandsbediening {
    Command vakje;
    public EenvoudigAfstandsbediening () {}

    public void setCommand (Command commando) {
        vakje = commando;
    }
    public void drukKnop() {
        vakje.execute();
    }
}
```

Eenvoudige afstandsbediening met één knop

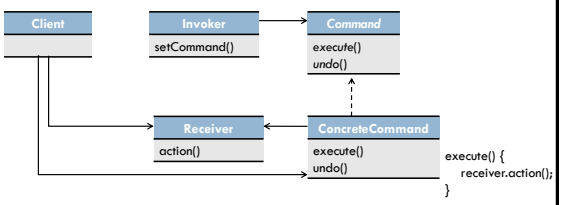
```
public class TestAfstandsbediening {
    public static void main (String[] args) {
        EenvoudigeAfstandsbediening bakje
            = new
            EenvoudigeAfstandsbediening();
        Licht licht = new Licht();
        LichtAanCommand lichtAan
            = new LichtAanCommand(licht);

        bakje.setCommand(lichtAan);
        bakje.drukKnop();
    }
}
```


Definitie Command Pattern

11

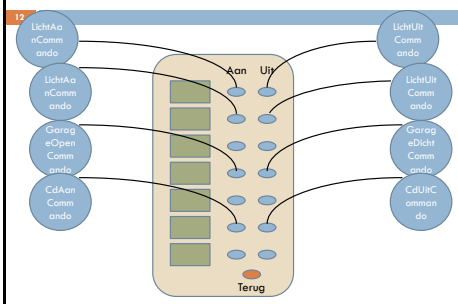


- Het Command Pattern schermt een aanroep af door middel van een object, waarbij je verschillende aanroepen in verschillende objecten kunt opbergen, in een queue kunt zetten of op schijf kunt bewaren; ook undo-operaties kunnen worden ondersteund.



Afstandsbediening = Invoker

12



Implementatie afstandsbediening

13

```
public class Afstandsbediening {
    public static final int AANT = 7;
    Command[] aanCommandos;
    Command[] uitCommandos;
    public Afstandsbediening() {
        aanCommandos  = new Command[AANT];
        uitCommandos = new Command[AANT];
        Command geenOpdracht = new GeenOpdracht();
        for (int i = 0; i < AANT; i++) {
            aanCommandos[i] = geenOpdracht;
            uitCommandos[i] = geenOpdracht;
        }
    }
```

Constructor: Initialisatie opdrachten

Implementatie afstandsbediening

14

```
public class Afstandsbediening {
    …
    public void setCommand(int vakje, Command aanOpdracht, Command uitOpdracht)
    {
        aanCommandos[vakje]  = aanOpdracht;
        uitCommandos[vakje] = uitOpdracht;
    }
    public void drukAanKnop(int vakje) {
        aanCommandos[vakje].execute();
    }
    public void drukUitKnop(int vakje) {
        uitCommandos[vakje].execute();
    }
```

Opdrachten instellen

Aan indrukken

Uit indrukken

Command-objecten

15

```
public class LichtUitCommand
    implements Command {
    Licht licht;
    public LichtUitCommand(Licht
licht) {
        this.licht = licht;
    }
    public void execute() {
        licht.off();
    }
}
```

Licht: on() off()

```
public class CdAanCommand
    implements Command {
    Stereo stereo;
    public CdAanCommand(Stereo
stereo) {
        this. stereo = stereo;
    }
    public void execute() {
        stereo.on();
        stereo.setCd();
        stereo.setVolume(11);
    }
}
```

Stereo: on() off() setCd() setRadio() setVolume()

GeenOpdracht-object

16

```
public class GeenOpdracht implements
    Command {
    public void execute() {    }
}
```

Vermijdt

```
public void drukAanKnop(int vakje) {
    if (aanCommandos[vakje] != null)
        aanCommandos[vakje].execute();
}
```

- Null-object
- Wanneer?
  - Geen zinnig object om naar terug te keren
  - Client niet verantwoordelijk om null af te handelen
- Surrogaat
  - "lege" execute methode

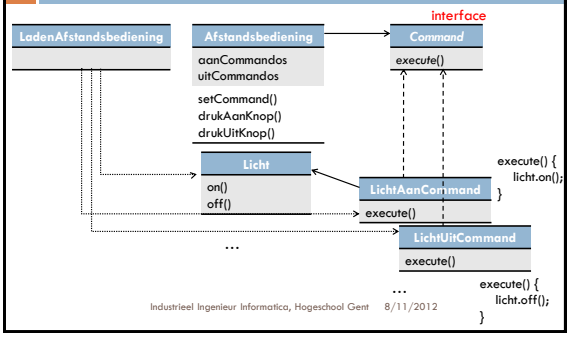
Command Pattern

17

- Afschermen methode aanroep
- Ontkoppeling van Invoker (afstandsbediening) en Receiver (apparaten)

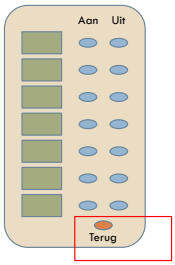
Command Pattern in voorbeeld

18



Terug-knop

19




```
public interface Command {
    public void execute();
    public void undo();
}

public class LichtAanCommand implements Command {
    Licht licht;
    public LichtAanCommand(Licht licht) {
        this.licht = licht;
    }
    public void execute() {
        licht.on();
    }
    public void undo() {
        licht.off();
    }
}
```


Aanpassing afstandsbediening

20

```
public class Afstandsbediening {
    public static final int AANT = 7;
    Command[] aanCommandos;
    Command[] uitCommandos;
    Command terugCommando;
    public Afstandsbediening() {
        aanCommandos  = new Command[AANT];
        uitCommandos = new Command[AANT];
        Command geenOpdracht = new GeenOpdracht();
        for (int i = 0; i < AANT; i++) {
            aanCommandos[i] = geenOpdracht;
            uitCommandos[i] = geenOpdracht;
        }
        terugCommando = geenOpdracht;
    }
```


Implementatie afstandsbediening

21

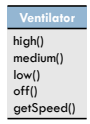
```
public void setCommand(int vakje, Command aanOpdracht, Command uitOpdracht) {
  aanCommandos[vakje]  = aanOpdracht;
  uitCommandos = uitOpdracht;
}
public void drukAanKnop(int vakje) {
  aanCommandos[vakje].execute();
  terugCommando = aanCommands[vakje];
}
public void drukUitKnop(int vakje) {
  uitCommandos[vakje].execute();
  terugCommando = uitCommandos[vakje];
}
```

```
public void drukTerugKnop() {
    terugCommando.undo();
    terugCommando = new
            GeenOpdracht();
}
```


Toestandsinfo bij “Terug”-knop

22

- Ventilator




```
public class Ventilator {
    public static final int HIGH = 3;
    public static final int MEDIUM = 2;
    public static final int LOW = 1;
    public static final int OFF = 0;
    int snelheid;
    public Ventilator() { snelheid = OFF; }
    public void high() { snelheid = HIGH; …}
    public void medium() { snelheid = MEDIUM; …}
    public void low() { snelheid = LOW; …}
    public void off() { snelheid = OFF; …}
    public int getSnelheid {return snelheid; } }
```


Commando-object met “undo”

23

```
public class VentilatorHoogCommand implements Command {
    Ventilator ventilator;
    int vorigeSnelheid;
    public VentilatorHoogCommand(Ventilator ventilator) {
        this.ventilator = ventilator;
    }
    public void execute() {
        vorigeSnelheid = ventilator.getSnelheid();
        ventilator.high();
    }
}
```


Commando-object met “undo”

24

```
public class VentilatorHoogCommand implements Command {
    …
    public void undo() {
        if (vorigeSnelheid == Ventilator.HIGH) {
            ventilator.high();
        } else if (vorigeSnelheid == Ventilator.MEDIUM) {
            ventilator.medium();
        } else if (vorigeSnelheid == Ventilator.LOW) {
            ventilator.low();
        } else if (vorigeSnelheid == Ventilator.OFF) {
            ventilator.off();
        }
    } }
```

Opdrachten combineren

25

- Eén druk
- Meerdere opdrachten
  - Licht dimmen, stereo en TV aan, bubbelbad aan …

```
public class MacroCommand implements Command {
    Command[] opdrachten;
    public MacroCommand(Command[] opdrachten) {
        this.opdrachten = opdrachten;
    }
    public void execute() {
        for (int i=0; i < opdrachten.length; i++) {
            opdrachten[i].execute();
        }   } }
```


Meerdere opdrachten = één knop

26

```
Command[] reeksAan = {lichtAan, stereoAan, tvAan, badAan};
Command[] reeksUit = {lichtUit, stereoUit, tvUit, badUit};
MacroCommand reeksAanMacro = new MacroCommand(reeksAan);
MacroCommand reeksUitMacro = new MacroCommand(reeksUit);

afstandsbediening.setCommand(0,reeksAanMacro, reeksUitMacro);
```


Meerdere opdrachten – "Terug"

27

```
public void undo() {
    for (int i=0; i < opdrachten.length; i++) {
        opdrachten[i].undo();
    }
}
```

- Anders?

Opmerkingen

28

- Altijd Receiver nodig?
- .Net?
- Meerdere undo-operaties?

Andere toepassingen Command Pattern

29

- Wachtrij-aanvragen/in een wachtrij plaatsen
- Logging-aanvragen

Wachtrij aanvragen

30

- Schedulers, thread-pools, job-queues, …
- Receiver + acties → object
  - Overdragen als object
  - Later activeren
  - Eventueel in andere thread
- Job-queues
  - Ene eind: opdrachten toevoegen
  - Andere eind
    - Beperkt aantal thread
    - Per thread: opdracht afhalen, uitvoeren, wachten einde uitvoering, verwijderen opdracht

Logging-aanvragen

31

- Alle acties loggen
- Crash
  - Alles herstellen: acties opnieuw uitvoeren
- Hoe?
  - store- en load-methode toevoegen
- Implementatie?
  - Opdracht uitvoeren + opdracht bewaren op schijf
  - Crash
    - Opdrachten laden + opnieuw uitvoeren (in batch)

Command
execute()
undo()
store()
load()

Samenvatting kennis

32

- 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
- OO-patterns
  - Strategy, Observer, Decorator, Abstract Factory, Factory Method
  - Het Command Pattern schermt een aanroep af door middel van een object, waarbij je verschillende aanroepen in verschillende objecten kunt opbergen, in een queue kunt zetten of op schijf bewaren, en undo-operaties kunt ondersteunen.