HET TEMPLATE METHOD PATTERN

Veerle Ongenae, Wijnand Schepens

Het Template Method Pattern

- Afschermen
  - Objectcreatie
  - Methodeaanroepen
  - Complexe interfaces
  - …
- Algoritmen afschermen

Voorbeeld

- Koffie en thee maken
  - Gelijkaardig
- Koffie
  - Water koken
  - Het water op koffie schenken
  - Koffie inschenken
  - Suiker en melk toevoegen
- Thee
  - Water koken
  - Thee laten trekken
  - Thee inschenken
  - Schijfje citroen toevoegen

Code koffie en thee maken

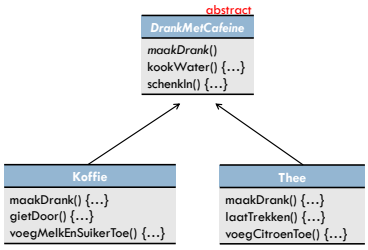

```
public class Koffie {
    void maakDrank() {
        kookWater();
        gietDoor();
        schenkIn();
        voegMelkEnSuikerToe();
    }
    void kookWater() {…}
    void gietDoor() {…}
    void schenkIn() {…}
    void voegMelkEnSuikerToe()
    {…}
}
```

Codeduplicatie !!

abstraheren

```
public class Thee {
    void maakDrank() {
        kookWater();
        laatTrekken();
        schenkIn();
        voegCitroenToe();
    }
    void kookWater() {…}
    void laatTrekken() {…}
    void schenkIn() {…}
    void voegCitroenToe() {…}
}
```


Abstraheren



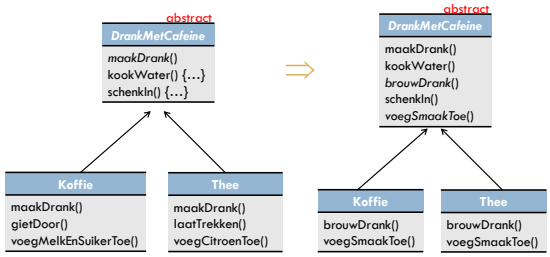
- Nog gemeenschappelijks? Verder abstraheren?

Verder abstraheren

- Beide recepten volgen zelfde algoritme
  - Water koken
  - Heet water gebruiken om drank te maken
  - Drank inschenken
  - Smaakstoffen toevoegen

Verder Abstraheren

7



Code

8

```
abtract class DrankMetCafeine
{
    void maakDrank() {
        kookWater();
        brouwDrank();
        schenkIn();
        voegSmaakToe();
    }
    abstract void brouwDrank();
    abstract void voegSmaakToe();
    void kookWater() {…}
    void schenkIn() {…}
}
```

```
class Koffie extends DrankMetCafeine
{
    void brouwDrank() {…}
    void voegSmaakToe() {…}
}

class Thee extends DrankMetCafeine
{
    void brouwDrank() {…}
    void voegSmaakToe() {…}
}
```


Samenvatting

9

- Recept gelijkaardig
  - Gegeneraliseerd in basisklasse
  - Gelijke stappen in basisklasse
    - Water koken
    - Inschenken
- Aantal stappen verschillend
  - Uitgewerkt door afgeleide klasse
    - Thee laten trekken ↔ koffie doorgieten
    - Citroen toevoegen ↔ suiker en melk toevoegen
- Template Method Pattern gebruikt

Template Method Pattern

10

```
abstract class DrankMetCafeine {
    void maakDrank() {
        void kookWater();
        void brouwDrank();
        void schenkIn();
        void voegSmaakToe();
    }
    abstract void brouwDrank();
    abstract void voegSmaakToe();
    void kookWater() {…}
    void schenkIn() {…}
}
```

Template Method

methodes

Geïmplementeerd in afgeleide klasse

```
Thee mijnThee = new Thee();
mijnThee.maakDrank();
```

- De Template Method definieert de stappen van een algoritme en staat afgeleide klassen toe de implementatie van één of meerdere stappen te leveren.

Voordelen Template Method

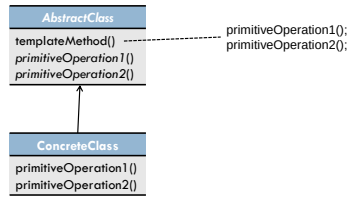
11

- Algoritme is afgeschermd
- Maximaliseren hergebruik
- Algoritme op één plaats
  - Codeveranderingen op één plaats
- Eenvoudig nieuwe cafeïnehoudende dranken toevoegen
- Kennis algoritme geconcentreerd in de klasse DrankMetCafeine

Definitie Template Method Pattern

12

- Het Template Method Pattern definieert het skelet van een algoritme in een methode, waarbij sommige stappen aan subklassen worden overgelaten. De Template Method laat subklassen bepaalde stappen in een algoritme herdefiniëren zonder de structuur van het algoritme te veranderen.



Code Template Method Pattern

13

```
abstract class AbstractClass
{
  final void templateMethod() {
     primitiveOp1();
     primitiveOp2();
     concreteOp();
  }
  abstract void primitiveOp1();
  abstract void primitiveOp2();

  void concreteOp() {…}
}
```


Code Template Method Pattern

14

```
abstract class AbstractClass
{
  final void templateMethod() {
     primitiveOp1();
     primitiveOp2();
     concreteOperation();
     hook();
  }
  abstract void primitiveOp1();
  abstract void primitiveOp2();

  void final concreteOperation() {…}

  void hook() {}
}
```

Kan niet overschreven worden

Concrete methode, doet standaard niets

Inhaken van een Template Method

15

- hook
  - Gedefinieerd in abstracte klasse
  - Lege of standaard implementatie
- Afgeleide klassen
  - Inhaken → methode overschrijven
  - Negeren → standaardimplementatie gebruiken
- Meerdere toepassingen mogelijk
  - Deel algoritme is optioneel
  - Subklasse kan reageren op bepaalde stap in algoritme
  - Subklasse neemt beslissing voor abstracte klasse

Haak in koffie (zonder melk … ?)

16

```
abstract class DrankMetCafeineEnHaak
{
    void maakDrank() {
        void kookWater();
        void brouwDrank();
        void schenkIn();
        if (gebruikerWenstSmaak())
            void voegSmaakToe();
    }
    abstract void brouwDrank();
    abstract void voegSmaakToe();
    void kookWater() {…}
    void schenkIn() {…}
    boolean gebruikerWenstSmaak() {
        return true;
    }
```

```
class KoffieMetHaak
   extends DrankMetCafeineEnHaak
{
    void brouwDrank() {…}
    void voegSmaakToe() {…}

    public boolean gebruikerWenstSmaak() {
        String antwoord
            = vraagGebruiker();
        return antwoord.equals("ja");
    }
}
```


Opmerkingen

17

- Voorwaardelijke besturing algoritme
- Voorbeeld
  - Kon ook zonder hook
- Template Method
  - Abstracte methode
    - Subklasse moet implementeren
  - Hooks
    - Optioneel deel algoritme
- Probeer aantal abstract methodes klein te houden

Hollywoodprincipe

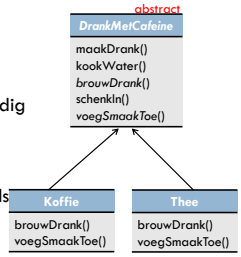
18

- Ontwerpprincipe
  - Bel ons niet, wij bellen jou.
- Voorkomen "afhankelijkheidsrot"
  - High-levelcomponenten afhankelijk van low-levelcomponenten die afhankelijk zijn van high-levelcomponenten …
- Hollywoodprincipe
  - Low-levelcomponenten haken zich aan het systeem
    - Roepen geen methodes van een high-levelcomponent aan
  - High-levelcomponenten bepalen of en hoe ze nodig zijn

Hollywoodprincipe en Template Method

19

- DrankMetCafeine
  - High-levelcomponent
  - Stuurt algoritme
  - Roept subklasse op indien nodig
- Koffie, Thee
  - Low-levelcomponent
  - Bevatten implementatiedetails
  - Roepen nooit rechtstreeks DrankMetCafeine aan



Opmerkingen

20

- Hollywoodprincipe
  - Low-levelcomponenten kunnen inhaken zonder dat er afhankelijkheden worden gecreëerd
  - Vorm van ontkoppeling
  - Minder sterk dan Dependency Inversion Principe
- Verboden methode high-levelcomponent op te roepen?
  - Ja, behalve methodes beschikbaar via overerving

Patterns vergelijken

21

- Template Method
  - Subklassen beslissen hoe de stappen in een algoritme geïmplementeerd moeten worden.
- Strategy
  - Schermt uitwisselbaar gedrag af en delegeert de beslissing over welk gedrag moet worden gebruikt
- Factory Method
  - Subklassen beslissen welke concrete objecten gemaakt moeten worden

Template Method: gebruik

22

- Veel gebruikt
- Opstellen frameworks
  - Bepalen hoe iets gedaan wordt.
  - Concrete implementatie door gebruiker framework
- Bestaande toepassingen
  - Sorteren
  - Swing: JFrame
  - Applets

Sorteren van array's

23

- Arrays-klasse
  - Templatemethode om te sorteren
- Vereenvoudigde versie van de code

```
public static void sort(Object[] objTabel) {
     Object[] kopieTabel = (Object[]) objTabel.clone();
     mergeSort(kopieTabel,  objTabel, 0, objTabel.length, 0);
}
private static void mergeSort(Object[] bron, Object[] doel, int start, int
einde, int off) {
    for (int i = start; i < einde; i++)
        for (int j = i; j > start &&
           ((Comparable)doel[j-1]).compareTo((Comparable)doel[j]) > 0; j--)
             swap(doel, j , j-1);
}
```

Template Method

Abstracte methode

Concrete methode

Eenden sorteren

24

- Sorteren array van eenden
- compareTo() implementeren
  - Arrays heeft geen afgeleide klassen
  - Comparable-interface
    - Methode compareTo()
- De te sorteren objecten moeten Comparable-interface implementeren

Eenden sorteren - code

25

```
public class Eend implements Comparable
{
    String naam;
    double gewicht;

    public int compareTo(Object object)
    {
        Eend andereEend = (Eend) object;
        if (this.gewicht < andereEend.gewicht) {
            return -1;
        else if (this.gewicht == andereEend.gewicht)
            return 0;
        else if (this.gewicht > andereEend.gewicht)
                return 1;
     }
}
```

```
public class TestEendenSorteren
{
    public static void main(String[] args)
    {
        Eend[] eenden = … ;
        …
        Arrays.sort(eenden);
        …
    }
}
```


Eenden sorteren - generic

26

```
public class Eend implements Comparable<Eend>
{
    String naam;
    double gewicht;

    public int compareTo(Eend andereEend)
    {
        Eend andereEend = (Eend) object;
        if (this.gewicht < andereEend.gewicht) {
            return -1;
        else if (this.gewicht == andereEend.gewicht)
            return 0;
        else if (this.gewicht > andereEend.gewicht)
                return 1;
     }
}
```


Eenden sorteren en Template Method

27

- Methode sort()
  - Bestuurt algoritme
  - Niemand kan dit veranderen
  - Rekent op methode compareTo() van Comparable
- Geen overerving
  - Geest Template Method
  - Flexibeler
- Lijkt op Strategy
  - Algoritme wordt volledig geïmplementeerd
  - Sorteren: algoritme incompleet

Swing: JFrame

28

- JFrame
  - Hook-methode: paintComponent() gebruikt in update()

```
public class MijnPanel extends JPanel
{
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawString("Hallo", 100,100);
    }
}
```


Applets

29

- Klein Java-programma in webpagina
- Klasse Applet
  - Verschillende hooks
  - Opgeroepen door browser, …
- Gedrag applet
  - Overschrijven hooks

```
class MijnApplet extends Applet
{
    String bericht;
    public void init() {
        bericht = "Hallo wereld!";
        repaint();
    }
    public void start() {
        bericht = "Start";
        repaint();
    }
    public void stop() {
        bericht = "Stop";
        repaint();
    }
    public void destroy() {}
    public void paint (Graphics g) {
        g.drawString(bericht, 5,15);
    }
}
```


Template Method ↔ Strategy

30

- Definiëren kader algoritme
- Werk
  - Deels door subklassen
- Afhankelijk van afgeleide klassen
- Ideaal voor frameworks
- Familie algoritmen
  - Uitwisselbaar
  - Afgeschermd
- Verschillende implementaties van een algoritme
- Flexibeler wisselen van algoritme

Samenvatting kennis

31

- 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
- OO-patterns
  - Strategy, Observer, Decorator, Abstract Factory, Factory Method, Command Pattern, Adapter, Facade
  - Het Template Method Pattern definieert een skelet voor een algoritme waarbij sommige stappen aan subklassen worden uitbesteed. De Template Method laat subklassen sommige stappen van een algoritme opnieuw definiëren zonder de structuur van het algoritme te veranderen.