FACTORY PATTERNS

Veerle Ongenae, Wijnand Schepens

Het Factory Pattern

- Bakken op OO-niveau
- Maken objecten
  - new
  - Publiek?
    - koppelingsproblemen

OO-ontwerpprincipes ↔ new

- Programmeer naar een interface:

```
Eend eend = new WildeEend();
```

- new
  - Instantiëren concrete klasse
  - Implementatie
  - Zwakkere, minder flexibele code

OO-ontwerpprincipes ↔ new

- Verzameling samenhangende concrete klassen

```
Eend eend;
if (picnic)
    eend = new WildeEend();
else if (jagen)
    eend = new LokEend();
else if (inBad)
    eend = new BadEend();
```

- Afhankelijk van context andere klasse instantiëren
  - At runtime
- Verandering
  - Code openbreken …
  - Meerdere plaatsen?

Verandering ↔ new

- Coderen naar interfaces
  - Afschermen van verandering
- Code met veel concrete klassen
  - Moet aangepast indien nieuwe concrete klassen
  - Zondigt tegen welk OO-ontwerpprincipe?
- Oplossing?
  - OO-ontwerpprincipe?

Voorbeeld

- Pizzeria
  - Pizza bestellen

```
Pizza bestelPizza()
{
    Pizza pizza = new Pizza();
    pizza.maak();
    pizza.bak();
    pizza.snij();
    pizza.inDoos();
    return pizza;
}
```

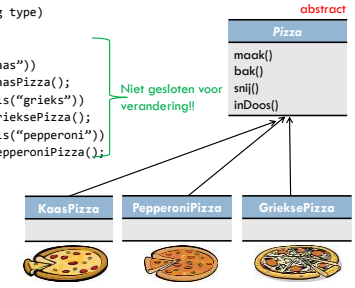
Flexibel? Hoe?
Interface?
Abstracte klasse? …

Pizza bestellen

7

```
Pizza bestelPizza(String type)
{
     Pizza pizza;
     if (type.equals("kaas"))
          pizza = new KaasPizza();
     else if (type.equals("grieks"))
          pizza = new GrieksePizza();
     else if (type.equals("pepperoni"))
          pizza = new PepperoniPizza();

     pizza.maak();
     pizza.bak();
     pizza.snij();
     pizza.inDoos();
     return pizza;
}
```

Niet gesloten voor verandering!!



Pizza-aanbod verandert

8

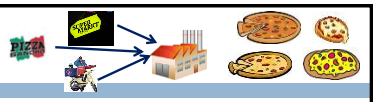
```
Pizza bestelPizza(String type)
{
     Pizza pizza;
     if (type.equals("kaas"))
          pizza = new KaasPizza();
     else if (type.equals("grieks"))
          pizza = new GrieksePizza();
     else if (type.equals("pepperoni"))
          pizza = new PepperoniPizza();

     pizza.maak();
     pizza.bak();
     pizza.snij();
     pizza.inDoos();
     return pizza;
}
```

⇒

```
Pizza bestelPizza(String type)
{
     Pizza pizza;
     if (type.equals("kaas"))
          pizza = new KaasPizza();
     else if (type.equals("pepperoni"))
          pizza = new PepperoniPizza();
     else if (type.equals("vegetarisch"))
          pizza = new VeggiePizza();
     else if (type.equals("zeevruchten"))
          pizza = new ZeeVruchtenPizza();

     pizza.maak();
     pizza.bak();
     pizza.snij();
     pizza.inDoos();
     return pizza;
}
```

Oplossing

9



```
public class SimplePizzaFabriek
{
    public Pizza maakPizza(String type)
    {
        if (type.equals("kaas"))
            return new KaasPizza();
        else if (type.equals("pepperoni"))
            return new PepperoniPizza();
        else if (type.equals("vegetarisch"))
            return new VeggiePizza();
        else if (type.equals("zeevruchten"))
            return new ZeeVruchtenPizza();
        else
            throw ...;
    }
}
```

- Isoleer wat verandert
  - Aanmaken pizza-objecten
  - Fabriek, Factory

Voordelen

10

- Probleem doorgeschoven?
  - Meerdere klanten van SimplePizzaFabriek mogelijk
    - Wijzigingen op één plaats
  - Bestellen pizza's moet niet aangepast bij wijziging

Pizzeria

11

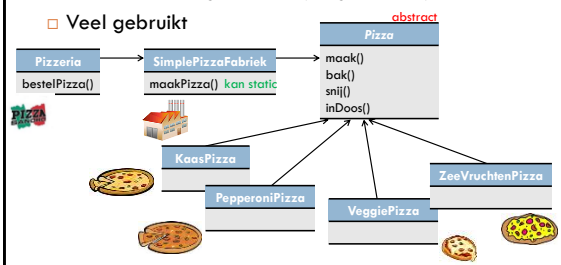
```
public class Pizzeria
{
    SimplePizzaFabriek fabriek = new SimplePizzaFabriek();

    public Pizza bestelPizza(String type) {
        Pizza pizza = fabriek.maakPizza(type);
        pizza.maak();
        pizza.bak();
        pizza.snij();
        pizza.inDoos();
        return pizza;
    }
    …
}
```

Hoe veranderen van fabriek?

Simple Factory

12

- Geen echt Design Pattern (volgens Go4)
- Veel gebruikt



Static Factory

13

- maakPizza() statische methode?
  - Static Factory genoemd
  - Nadeel
    - Geen subklasse → gedrag methode kan niet veranderd worden

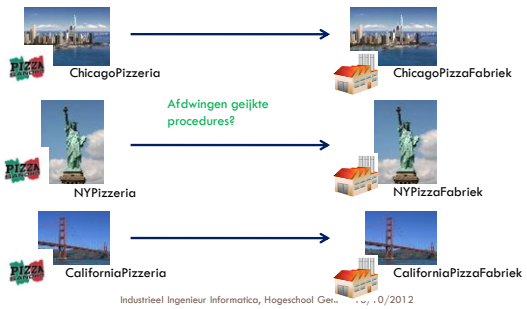
Verschillende pizzeria's

14

- Kwaliteit
  - Geteste code hergebruiken
  - Bakken, snijden, in dozen steken moet volgens de "geijkte procedures"
- Regionale verschillen toegelaten
  - Verschillende pizza's afhankelijk van locatie
    - Dunne ↔ dikke bodems
    - …
  - Verschillende fabrieken per locatie
    - NYPizzaFabriek
    - ChicagoPizzaFabriek
    - CaliforniaPizzaFabriek

Gebruik Simple Factory

15



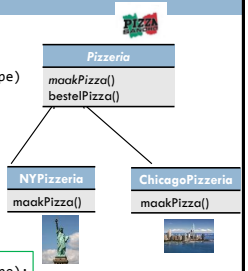
Framework voor de pizzeria's

16

```
public abstract class Pizzeria
{
    public Pizza bestelPizza(String type)
    {
        Pizza pizza = maakPizza(type);
        pizza.maak();
        pizza.bak();
        pizza.snij();
        pizza.inDoos();
        return pizza;
    }
    abstract Pizza maakPizza(String type);
}
```

blijft gelijk

verandering



Concrete pizzeria

17

```
public class NYPizzeria extends Pizzeria
{
    public Pizza maakPizza(String type)
    {
        if (type.equals("kaas"))
            return new NYKaasPizza();
        else if (type.equals("pepperoni"))
            return new NYPepperoniPizza();
        else if (type.equals("vegetarisch"))
            return new NYVeggiePizza();
        else if (type.equals("zeevruchten"))
            return new NYZeevruchtenPizza();
        else
            throw ...;
    }
}
```

```
public class ChicagoPizzeria extends Pizzeria
{
    public Pizza maakPizza(String type)
    {
        if (type.equals("kaas"))
            return new ChicagoKaasPizza();
        else if (type.equals("pepperoni"))
            return new ChicagoPepperoniPizza();
        else if (type.equals("vegetarisch"))
            return new ChicagoVeggiePizza();
        else if (type.equals("zeevruchten"))
            return new ChicagoZeevruchtenPizza();
        else
            throw ...;
    }
}
```


Fabrieksmethode

18

```
public abstract class Pizzeria
{
    public Pizza bestelPizza(String type)
    {
        Pizza pizza = maakPizza(type);
        pizza.maak();
        pizza.bak();
        pizza.snij();
        pizza.inDoos();
        return pizza;
    }

    abstract Pizza maakPizza(String type);
}
```

- Afgeleide klassen verantwoordelijk voor objectcreatie
- Fabrieksmethode
  - Factory method
  - Afhandelen objectcreatie
  - Isoleren in afgeleide klasse

Pizza's bestellen

19

- Pizzeria kiezen

```
Pizzeria nyPizzeria = new NYPizzeria();
```

- Pizza bestellen

```
nyPizzeria.bestelPizza("kaas");
```

- Pizza maken

```
Pizza pizza = maakPizza("kaas");
```

Pizza bepaald door lokale stijl

- Pizza bereiden

```
pizza.maak();
pizza.bak();
pizza.snij();
pizza.inDoos();
```


Pizza's

20

```
public abstract class Pizza
{
    String naam, deeg, saus;
    ArrayList<String> beleg
        = new ArrayList<>();

    void maak() { … }
    void bak() { … }
    void snij() { … }
    void inDoos() {…}

    public String getNaam() {
        return naam;
    }
}
```

```
public class NYKaasPizza
                extends Pizza
{
    public NYKaasPizza()
    {
        naam = "NY Style Saus&Kaas Pizza";
        deeg = "Dunne bodem";
        saus = "Kruidige tomatensaus";
        beleg.add("Geraspte kaas");
    }
}
```


Testen

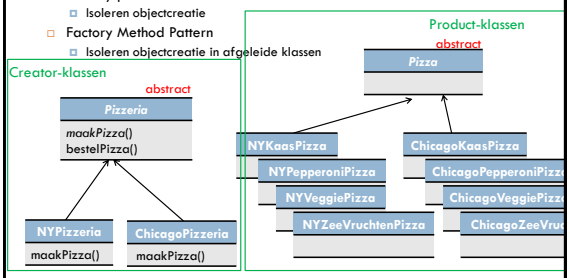
21

```
public class TestPizzerias
{
    public static void main(String[] args)
    {
        Pizzeria nyPizzeria = new NYPizzeria();
        Pizzeria chicagoPizzeria = new ChicagoPizzeria();

        Pizza pizza = nyPizzeria.bestelPizza("kaas");
        System.out.println("Eerste pizza: " + pizza.getNaam());

        pizza = chicagoPizzeria.bestelPizza("kaas");
        System.out.println("Tweede pizza: " + pizza.getNaam());
    }
}
```


Factory Method Pattern

22

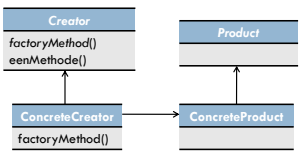
- Factory patterns
  - Isoleren objectcreatie
- Factory Method Pattern
  - Isoleren objectcreatie in afgeleide klassen



Factory Method Pattern

23

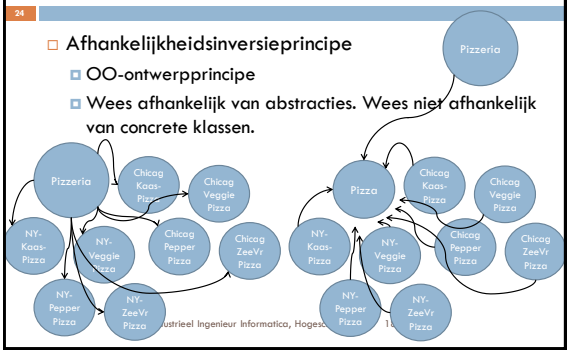
- Het Factory Method Pattern definieert een interface voor het creëren van een object, maar laat de afgeleide klassen beslissen welke klasse geïnstantieerd wordt. De Factory Method draagt de instantiatie over aan de afgeleide klassen.



Dependency Inversion-principe

24

- Afhankelijkheidsinversieprincipe
  - OO-ontwerpprincipe
  - Wees afhankelijk van abstracties. Wees niet afhankelijk van concrete klassen.



Dependency Inversion-principe

25

A. High-levelcomponenten (bv. Pizzeria) mogen niet afhankelijk zijn van low-levelcomponenten (bv. Pizza-implementaties). Beide moeten afhankelijk zijn van abstracties.
B. Abstracties mogen niet afhankelijk zijn van details. Details moeten afhankelijk zijn van abstractions.

- Sterker dan "Programmeer naar een interface"

Dependency Inversion-principe

26

- Inversie?
  - Low-levelcomponenten afhankelijk van een abstractie op hoger niveau (omdraaien pijlen)
- Richtlijnen voor realisatie
  - Geen enkele variabele bevat een referentie naar een concrete klasse.
  - Van een concrete klasse wordt geen klasse afgeleid.
  - Geen enkel methode zou een geïmplementeerde methode van een basisklasse mogen overschrijven.
  - "Streven", weten waar je ze niet volgt

Terug naar de pizzeria

27

- De juiste procedures worden gevolgd (bakken, snijden, …)
- Sommige pizzeria's gebruiken echter minderwaardige ingrediënten
- Oplossing
  - Eigen fabriek
    - Aanmaken en leveren ingrediënten
- Ingrediënten afhankelijk van de regio

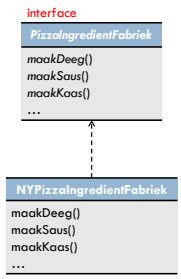
Ingrediëntenfabriek

28

- Eén fabriek per regio
- Maakt ingrediënten
  - Deeg, saus, kaas, zeevruchten, groenten, …
  - Voor elk type ingrediënt een klasse
- Interface
  - Methode per ingrediënt
  - Bepaalt wat de fabrieken kunnen maken

Ingrediëntenfabriek

29



Ingrediëntenfabriek

30

```
interface PizzaIngredientFabriek
{
    Deeg maakDeeg();
    Saus maakSaus();
    Kaas maakKaas();
    Groenten[] maakGroeten();
    Pepers maakPepers();
    ZeeVruchten maakZeeVruchten();
}
```

```
class NYPizzaIngredientFabriek
            implements PizzaIngredientFabriek
{
    public Deeg maakDeeg()
    {
        return new DunKrokantDeeg();
    }
    public Saus maakSaus()
    {
        return new KruidigeSaus();
    }
    public Groenten[] maakGroenten() {
        return new Groenten[]
          { new Look(), new Ajuin() };
    }
    // ...
}
```

Pizza's aanpassen

31

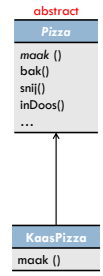
```
public abstract class Pizza
{
    String naam;
    Deeg deeg;
    Saus saus;
    Kaas kaas;
    Groenten[] groenten;
    // …

    abstract void maak() ;

    void bak() { … }
    void snij() { … }
    void inDoos() {…}
}
```

Geen pizzaklasse
per soort
per regio
meer nodig !!

Pizza's aanpassen

32

```
public class KaasPizza extends Pizza
{
    PizzaIngredientFabriek ingredientFabriek;

    public KaasPizza(PizzaIngredientFabriek
                     ingredientFabriek)
    {
        this.ingredientFabriek = ingredientFabriek;
    }

    void maak()
    {
        deeg = ingredientFabriek.maakDeeg();
        saus  = ingredientFabriek.maakSaus();
        kaas = ingredientFabriek.maakKaas();
    }
}
```



Pizzeria's aanpassen

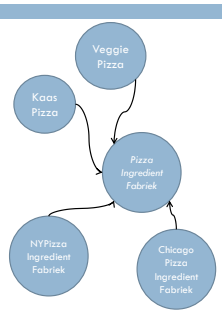
33

```
public class NYPizzeria extends Pizzeria
{
    protected Pizza maakPizza(String type)
    {
        Pizza pizza = null;
        PizzaIngredientFabriek ingredientFabriek = new NYPizzaIngredientFabriek();
        if (type.equals("kaas"))
        {
            pizza = new KaasPizza(ingredientFabriek);
            pizza.setNaam("NY Style Kaaspizza");
        }
        else if (type.equals("pepperoni"))
        {
            pizza = new  PepperoniPizza(ingredientFabriek);
            pizza.setNaam(("NY Style Pepperonipizza");
        }
        else …

        return pizza;
    }
}
```


Overzicht aanpassingen

34

- Lijst beschikbare ingredienten
  - PizzaIngredientFabriek
- Ingrediënten verschillend per regio
  - NYPizzaIngredientFabriek
- Abstract Factory
  - Interface
  - Maken reeks producten
  - Implementatie
    - Verschillende fabrieken mogelijk
- Ontkoppeling
  - Maken pizza's
  - Feitelijke fabriek
    - Actuele producten



Pizza's bestellen

35

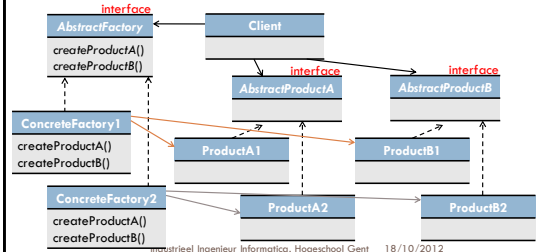
- Pizzeria kiezen
  Pizzeria nyPizzeria = new NYPizzeria();
- Pizza bestellen
  nyPizzeria.bestelPizza("kaas");
- Pizza maken
  Pizza pizza = maakPizza("kaas");
- Methode maakPizza van NYPizzeria
  return new NYKaasPizza();
- Pizza bereiden
  pizza.maak(); Ingrediënten overlopen
  pizza.bak();
  pizza.snij();
  pizza.inDoos();

- Pizzeria kiezen
  Pizzeria nyPizzeria = new NYPizzeria();
- Pizza bestellen
  nyPizzeria.bestelPizza("kaas");
- Pizza maken
  Pizza pizza = maakPizza("kaas");
- Methode maakPizza van NYPizzeria
  return new KaasPizza(ingredientFabriek);
- Pizza maken
  void maak() {
  deeg = ingredientFabriek.maakDeeg();
  saus = ingredientFabriek.maakSaus();
  kaas = ingredientFabriek.maakKaas();
  }
- Pizza verderbereiden
  pizza.bak();
  pizza.snij();
  pizza.inDoos();

Abstract Factory Pattern

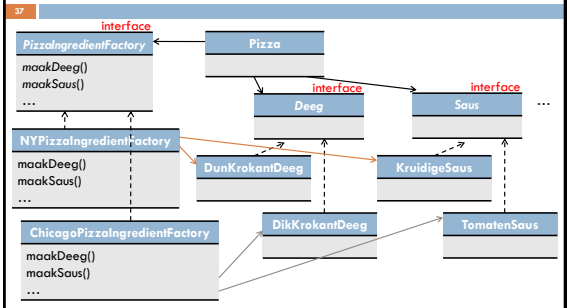
36

- Het Abstract Factory Pattern levert een interface voor de vervaardiging van reeksen gerelateerde of afhankelijke objecten zonder hun concrete klassen te specifiëren.



Abstract Factory Pattern – pizza's

37



Factory Method ↔ Abstract Factory

38

- Methodes van een Abstract Factory worden vaak geïmplementeerd als een Factory Method
- Ontkoppelen applicaties van specifieke implementaties (isolatie objectcreatie)

- Factory Method: gebruikt overerving
- Abstract Factory: gebruikt objectcompositie

- Factory Method: maakt één product
- Abstract Factory: maakt meerdere producten (familie)

Samenvatting kennis

39

- 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 levert een interface voor het maken van reeksen gerelateerde of afhankelijke objecten zonder hun concrete klassen te specificeren.
  - Factory Method definieert een interface voor objectcreatie, maar laat de subklassen beslissen over welke klasse wordt geïnstantieerd. De Factory Method zorgt ervoor dat een klasse de instantiatie overlaat aan de subklasse.