HET COMPOSITE PATTERN

Veerle Ongenae, Wijnand Schepens

Serveerster

2

```
public class Serveerster
{
    Menu pannenKoekMenu, dinerMenu, cafeMenu;
    // ...

    public void printMenu()
    {
        Iterator ontbijtIterator = pannenKoekMenu.maakIterator();
        Iterator lunchIterator = dinerMenu.maakIterator();
        Iterator cafeIterator = cafeMenu.maakIterator();
        printMenu(ontbijtIterator);
        printMenu(lunchIterator();
        printMenu(cafeIterator);
    }
}
```

Kan beter!

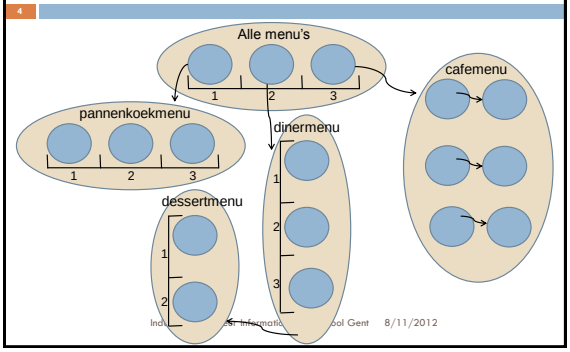
Serveerster

3

```
public class Serveerster {
   ArrayList<Menu> menus;
   // ...
   public void printMenu()
   {
        Iterator menuIterator = menus.iterator();
        while (menuIterator.hasNext()) {
            Menu menu = (Menu) menuIterator.next();
            printMenu(menu.maakIterator());
        }
    }
    private void printMenu(Iterator iterator) {
        while (iterator.hasNext()) {
            MenuItem menuItem = (MenuItem) iterator.next();
        … // print menu-item
    } } }
```


Help! Dessertsubmenu …

4



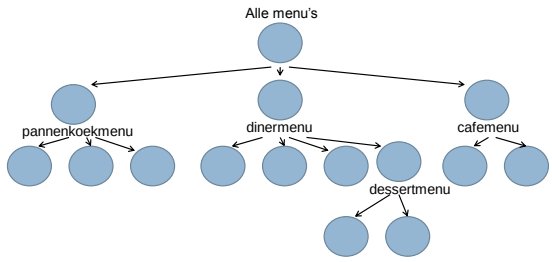
Nodig?

5

- Boomstructuur
  - Menu's
  - Submenu's
  - Menu-items
- Over items in elke menu kunnen lopen
  - Zoals bij Iterator
  - Flexibeler
    - Alleen submenu
    - Alles samen
    - …

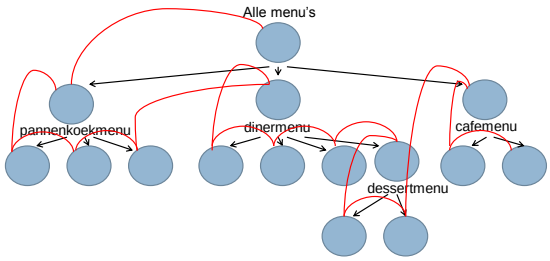
Boomstructuur

6



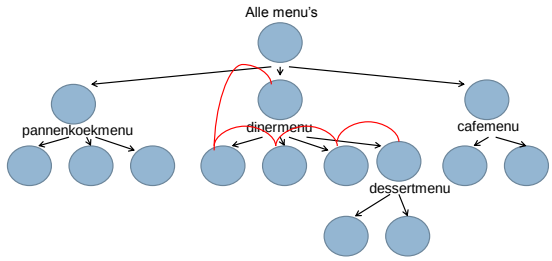
Itereren: volledige boom

7



Itereren: één menu

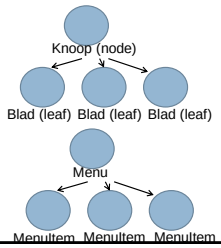
8



Definitie Composite Pattern

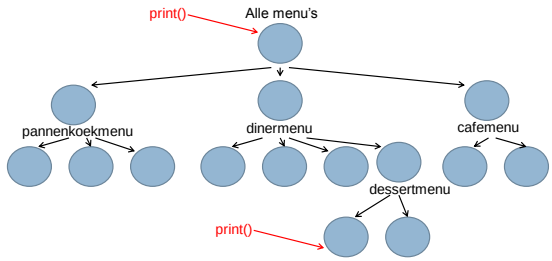
9

- Het Composite Pattern stelt je in staat om objecten in boomstructuren samen te stellen om part-whole hiërarchiën weer te geven. Composite laat clients de afzonderlijke objecten of samengestelde objecten op uniforme wijze behandelen.
  - Boom
    - Geneste menugroepen
    - Menu-items
    - In zelfde structuur
  - Part – whole hiërarchie



Eén geheel ↔ delen (uniform behandelen)

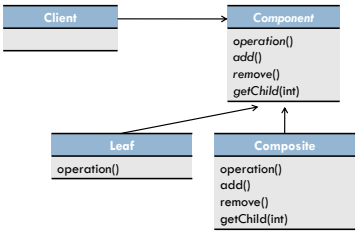
10



Definitie Composite Pattern

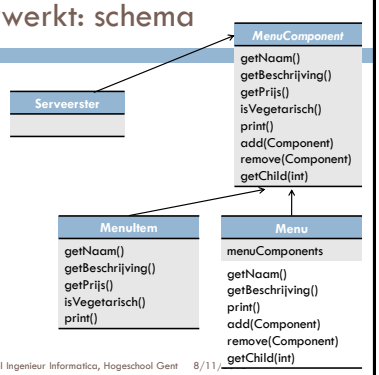
11

- Het Composite Pattern stelt je in staat om objecten in boomstructuren samen te stellen om part-whole hiërarchiën weer te geven. Composite laat clients de afzonderlijke objecten of samengestelde objecten op uniforme wijze behandelen.



Menu's herwerkt: schema

12



MenuComponent

13

```
public abstract class MenuComponent
{
    public void add(MenuComponent menuComponent) {
        throw new UnsupportedOperationException();
    }
    public void remove(MenuComponent menuComponent)
    {
        throw new UnsupportedOperationException();
    }
    public MenuComponent getChild(int i) {
        throw new UnsupportedOperationException();
    }
    public String getNaam() {
        throw new UnsupportedOperationException();
    }
    public String getBeschrijving() {
        throw new UnsupportedOperationException();
    }
    public double getPrijs() {
        throw new UnsupportedOperationException();
    }
    public boolean isVegetarisch() {
        throw new UnsupportedOperationException();
    }
    public void print() {
        throw new UnsupportedOperationException();
    }
}
```


MenuComponent

14

- Methodes
  - Menu
  - Menu-item
- Standaardimplementatie
  - Exceptie gooien

MenuItem

15

```
public class MenuItem extends MenuComponent
{
    String naam, beschrijving;
    boolean vegetarisch;
    double prijs;

    public MenuItem(...) { ... }

    public String getNaam() { return naam; }

    public void print() {…}
}
```

Menu Pannenkoekenhuis

Pannenkoek met bosbessen 3,49
Pannenkoek met verse bosbessen

…

oud

MenuItem
getNaam() getBeschrijving() getPrijs() isVegetarisch()

Menu

16

```
public class Menu extends MenuComponent
{
    ArrayList<MenuComponent> menuComponents = new ArrayList();
    String naam, beschrijving;

    public void add(MenuComponent comp) { menuComponents.add(comp);  }
    public void remove(MenuComponent comp) {menuComponents.remove(comp); }

    public MenuComponent getChild(int i) {
          return menuComponents.get (i);
    }

     public void print() {
        for (MenuComponent comp : menuComponents)
            comp.print();
     }
}
```


Serveerster

17

```
public class Serveerster {
   ArrayList<Menu> menus;
    public Serveerster(ArrayList<Menu> menus) {
         this.menus = menus;
    }
    public void printMenu() {
         Iterator menuIterator = menus.iterator();
         while (menuIterator.hasNext()) {
              Menu menu = (Menu)
menuIterator.next();
              printMenu(menu.maakIterator());
         }
    }
    private void printMenu(Iterator iterator) {
        while (iterator.hasNext()) {
            MenuItem menuItem = (MenuItem)
iterator.next();
        … // print menu-item } } }
```

⇒

```
public class Serveerster {
    MenuComponent alleMenus;
    public Serveerster(MenuComponent
                       alleMenus) {
        this.alleMenus = alleMenus;
    }
    public void printMenu() {
        alleMenus.print();
    }
}
```


Eén klasse, één verantwoordelijkheid?

18

- Composite Pattern
  - Beheert hiërarchie
  - Voert menu-gerelateerd opdrachten uit
  - Combinatie opdrachten menu's en menu-items
- Voorkeur transparantie
  - Uniform behandelen menu's en menu-items
  - Uniform behandelen compositie en bladknoop
- Verlies veiligheid
  - Opdrachten van compositie op blad en omgekeerd
- Compromis

Itereren over compositie

19

- Menu overlopen
  - Vegetarische gerechten selecteren

```
public class Menu extends MenuComponent{
   …
   public Iterator maakIterator() {
       return new CompositeIterator(menuComponents.iterator);
   }
}
```

MenuComponent
getNaam()
getBeschrijving()
getPrijs()
isVegetarisch()
print()
add(Component)
remove(Component)
getChild(int)
maakIterator()

```
public class MenuItem extends MenuComponent{
   …
   public Iterator maakIterator() {
       return new NullIterator();
   }
}
```


Vegetarisch Menu

20

```
public class Serveerster {
    MenuComponent alleMenus;
    public void printMenu() {
        alleMenus.print();
    }
    public void printVegetarischMenu() {
         Iterator iterator = alleMenus.maakIterator();

         while (iterator.hasNext()) {
              MenuComponent comp = (MenuComponent) iterator.next();
              try {
                   if (comp.isVegetarisch()) {
                        comp.print();
                   }
               } catch (UnsupportedOperationException e) { }
         }
     }
```


Opmerkingen

21

- try/catch
  - Afhandelen logica
  - Geen goede manier van programmeren
- Alternatief
  - Type MenuComponent controleren
    - Verlies transparantie
  - Methode isVegetarisch anders implementeren in Menu
    - Geeft niet aan dat het eigenlijk niet ondersteund is

Composite Pattern

22

- Objectenverzameling
  - Whole-partrelaties
  - Uniform behandelen
- Voorbeelden
  - GUI
- Alle objecten zelfde interface
  - Sommige methode-aanroepen zinloos
    - null/false
    - Exceptie gooien
- Boomstructuur
  - Parent ↔ child
- Ordening kinderen kan
- Caching
  - Vaak boom doorlopen
  - Kinderen berekenen

Samenvatting kennis

23

- 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
  - Het Composite Pattern stelt objecten tot boomstructuren samen om een part-wholehiërarchie weer te geven. Het laat clients individuele objecten en objectcomposities op uniforme wijze behandelen.