Design Patterns

Gerwin van Dijken (gerwin.vandijken@inholland.nl)

Lesmateriaal

Moodle-course:

- 2021 Inf1.4 Ontwikkeling

Boek:

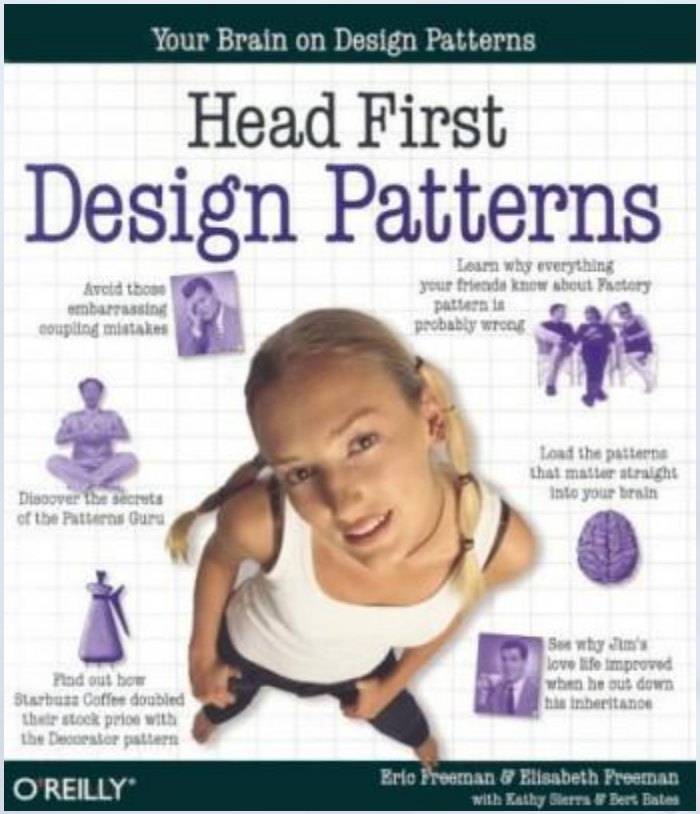
- *geen fysiek boek (op Moodle staan verwijzingen)*

Opdrachten:

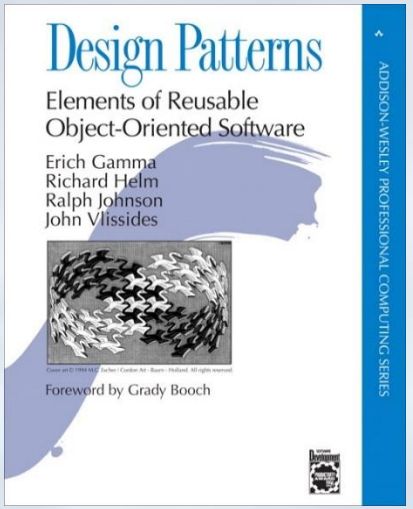
- wekelijks (6x), individueel, verplicht
- inleveren op Moodle (CodeGrade), deadline volgende week
- alle AutoTests en handmatige controle moeten 10/10 zijn
- hulp/feedback in de consultancy meetings *(MS Teams)*

Aan te raden boek…

- ‘Head First Design Patterns’ (Java…)



GoF



Programma periode 1.4

01 (wk-15)	abstracte classes en interfaces
02 (wk-16)	Template Method pattern / Observer pattern
03 (wk-17)	MVC pattern
04 (wk-18)	*geen lessen (meivakantie)*
05 (wk-19)	Strategy pattern / Adapter pattern
06 (wk-20)	Singleton pattern / State pattern
07 (wk-21)	Factory patterns
08 (wk-22)	herhaling / proeftentamen

---

09 (wk-23)	tentamen (praktijk)
10 (wk-24)	*hertentamens (vakken periode 1.3)*
11 (wk-25)	*hertentamens (vakken periode 1.4)*

Samenvatting Programmeren 3

- We hebben het (in periode 1.3) gehad over:
  - classes: bevatten zowel data als code
  - afgeleide classes: erven van een base class
  - members: data en methoden van een object
  - properties (set en get): member fields met toegangsregeling, eventueel read-only
  - access modifiers: toegankelijkheid van members (public, protected, private)

Member fields maken we standaard private of protected; eventueel kunnen we deze members met public properties beschikbaar maken.

Wat zijn design patterns?

- Een design pattern is een ontwerp/patroon (bedacht door anderen) voor ‘standaard’ problemen

- Bedoeld om software:
  - onderhoudbaarder te maken;
  - geschikt te maken voor uitbreidingen;
  - flexibeler te maken;

Design patterns zijn niet geschikt voor kleine applicaties; ze zijn alleen van toepassing/nuttig bij grote/complexe software applicaties.

Interfaces

- Basis van de meeste Design Patterns zijn ‘interfaces’…

Abstracte classes

- Zijn altijd de basis voor andere classes (er kunnen geen objecten van gemaakt worden)
- Bevatten een deel-implementatie (data en/of methoden)
- Kunnen één of meerdere methoden zonder body bevatten; afgeleide classes moeten deze (abstracte) methoden implementeren

Voorbeeld abstracte class

```
public abstract class UitleenObject
{
  private UitleenStatus status;
  public int ExemplaarNummer { get; set; }
  public string Titel { get; set; }

  // constructor
  public UitleenObject(int exemplaarNr, string titel)
  {
    this.ExemplaarNummer = exemplaarNr;
    this.Titel = titel;
    this.status = UitleenStatus.Aanwezig;
  }

  public void Uitlenen()
  {
    if (status == UitleenStatus.Uitgeleend)
      throw new Exception("Boek is reeds uitgeleend!");
    this.status = UitleenStatus.Uitgeleend;
  }

  // meer methoden hier...
}
```

Class 'UitleenObject' is abstract; er kunnen dus geen instanties van gemaakt worden met: new UitleenObject(...)

Deze abstracte class bevat een aantal properties en methoden; de afgeleide classes hoeven deze dus niet te implementeren (ze 'erven' deze over).

Een afgeleide class

```
class Boek : UitleenObject
{
  public string Auteur { get; set; }

  // constructor
  public Boek(int exemplaarNr, string titel, string auteur)
    : base(exemplaarNr, titel)
  {
    this.Auteur = auteur;
  }

  public override string ToString()
  {
    return String.Format("[Boek] '{0}' ({1})", Titel, Auteur);
  }
}
```

Overerven gaat op dezelfde manier als bij 'normale' (niet abstracte) classes, dus via <...> : <...>.

We geven alle informatie voor de base class direct door in de constructor via : base (...).

Abstracte methoden

- Abstracte methoden moeten geimplementeerd worden door een afgeleide class

```
public abstract class Figure
{
    protected int x, y;

    public int X { get { return x; } }
    public int Y { get { return y; } }

    // constructor
    public Figure(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public abstract void Draw(Graphic

    // other methods...
}
```

Class 'Figuur' is abstract; er kunnen dus geen instanties van gemaakt worden met: new Figuur(...)

Een afgeleide class moet methode 'Teken' implementeren, omdat deze methode abstract is.

Als een abstracte methode niet geimplementeerd wordt, dan volgt de melding:
'<class-naam> does not implement inherited abstract member <methode-naam>'.

Abstracte methoden

Class 'Vierkant' erf van abstracte class 'Figuur'.

```
public abstract class Figuur
{
  protected int x, y;

  public int X { get { return x; } }
  public int Y { get { return y; } }

  // constructor
  public Figuur(int x, int y)
  {
    this.x = x;
    this.y = y;
  }

  public abstract void Teken(Graphics g);

  // andere methoden...
}
```

```
public class Vierkant : Figuur
{
  protected int width;

  public int Width {
    get { return width; }
  }

  // constructor
  public Vierkant(int x, int y, int width)
    : base(x, y)
  {
    this.width = width;
  }

  public override void Teken(Graphics g)
  {
    g.DrawRectangle(new Pen(Color.Red),
                    x, y, width, width);
  }
}
```

Class 'Vierkant' implementeert abstracte methode 'Teken'. Keyword 'override' is hierbij nodig.

Interfaces

Een interface heeft geen implementatie (geen data and geen ingevulde methoden).

- Een nadeel van abstracte classes is dat een class maar van één base class kan afleiden
- Een class kan wel meerdere interfaces "implementeren"
- Interface is een contract waar 'sub-classes' aan moeten voldoen
- Een interface bevat uitsluitend abstracte methoden/properties *(geen implementatie!)*

Voorbeeld interface

- Een interface beschrijft vaak een gedrag, dat andere classes moeten implementeren (Bestuurbaar, Opblaasbaar, Sorteerbaar, Vergelijkbaar, …) engels: '…able'

De methoden in een interface zijn leeg, dus geen code tussen accolades { … }, maar een ';' als afsluiter. Ze zijn 'by default' public.

```
interface IBestuurbaar
{
  void NaarLinks();
  void NaarRechts();
  void NaarVoren();
  void NaarAchteren();
}
```

'Bestuurbare' dingen kunnen 4 kanten op.

Goede gewoonte: gebruik een hoofdletter 'I' voor elke interface-naam.

Classes die een interface implementeren moeten alle methoden v/d interface implementeren (in dit geval dus 4 methoden), met exact dezelfde signatuur (naam en parameters).

Voorbeeld interface

Een interface 'overerven' gaat op dezelfde manier als het overerven van een class, dus via <...> : <...>.

```
interface IBestuurbaar
{
  void NaarLlinks();
  void NaarRechts();
  void NaarVoren();
  void NaarAchteren();
}
```

```
public class SpeelgoedAuto : IBestuurbaar
{
  private int x, y;

  public int X { get { return x; } }
  public int Y { get { return y; } }

  public SpeelgoedAuto(int x, int y)
  {
    this.x = x;
    this.y = y;
  }

  // interface methods
  public void NaarLinks() { x--; }
  public void NaarRechts() { x++; }
  public void NaarVoren() { y--; }
  public void NaarAchteren() { y++; }
}
```

Class 'SpeelgoedAuto' implementeert interface 'IBestuurbaar' door de 4 methoden te implementeren.

Als een interface methode niet geimplementeerd wordt, dan volgt de melding:
'<class-naam> does not implement interface member <methode-naam>'.

Voorbeeld interface

- We kunnen nu 'tegen een interface aan' programmeren, zonder de implementatie te weten!

We maken een 'SpeelgoedAuto' aan, maar de referentie is van type 'IBestuurbaar'.

```
static void Main(string[] args)
{
  IBestuurbaar auto = new SpeelgoedAuto(10, 10);
  RondjeRijden(auto);
}
```

```
interface IBestuurbaar
{
  void NaarLinks();
  void NaarRechts();
  void NaarVoren();
  void NaarAchteren();
}
```

```
static void RondjeRijden(IBestuurbaar voertuig)
{
  voertuig.NaarLinks();
  voertuig.NaarRechts();
  voertuig.NaarLinks();
  voertuig.NaarVoren();
}
```

Deze methode 'RondjeRijden' heeft geen idee wat voor ding bestuurd wordt, maar dat maakt hem niet uit! (zolang het maar 'bestuurbaar' is...)

Samenvattend

- Van een abstracte class kun je geen objecten aanmaken *(via new)*
- Een abstracte class bevat meestal één of meerdere abstracte methoden, die de afgeleide classes moeten implementeren

- Een interface bevat uitsluitend 'abstracte methoden'
- Een class kan maar van één basis class afleiden
- Een class kan meerdere interfaces implementeren

- Het is een goede gewoonte om 'tegen een interface' te programmeren

Opdrachten

- Zie Moodle: ‘Week 1 opdrachten’