Geavanceerd Object-georiënteerd Programmeren

Prof. dr. Kris Luyten
dr. Bram van Deurzen
Dries Cardinaels
Gilles Eerlings

Agenda

1. Java
2. Java
3. Java
4. Lezen over Java
5. Programmeren in Java

Object-georiënteerd Programmeren
+
Basisprincipes in Java

Leesmateriaal (verplicht)

- Thinking in Java 3rd ed., Bruce Eckell
  - Hoofdstuk 1, p. 36 - 61 **(*kennen* & kunnen)**
  - Hoofdstuk 2, p. 85 - 114 **(enkel kunnen)**
  - Hoofdstuk 3 doornemen en wanneer nodig nadien raadplegen
  - Hoofdstuk 4, "Guaranteed initialization with the constructor" en "Method overloading" **(enkel kunnen)**

http://mindview.net/Books/TIJ/#ElectronicBookFormats

Basisprincipes in Java

Een Java programma is een collectie van (geïnstantiëerde) klassen

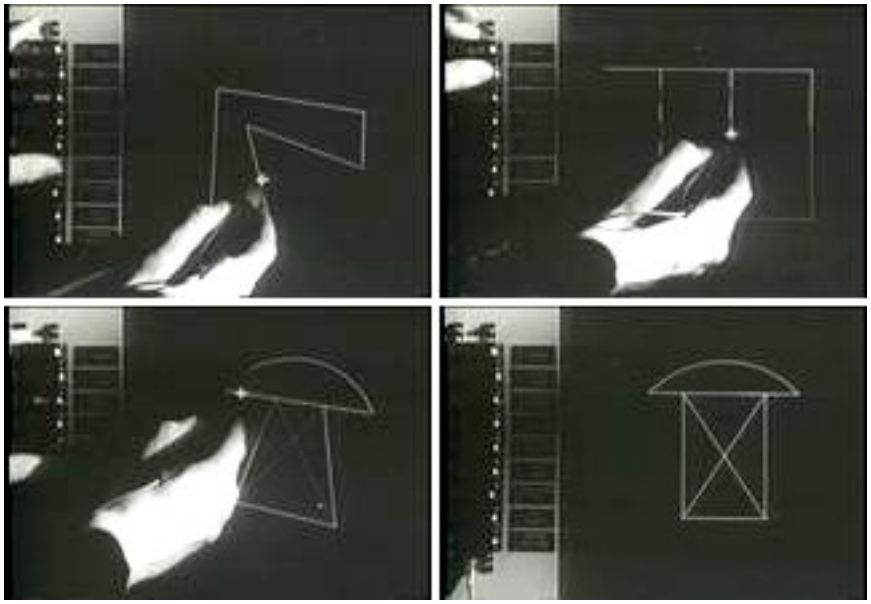
Klassen en objecten

Wat zijn dat nu juist?

Het ontstaan van het Object

Oorspronkelijk idee komt uit
Sketchpad (1960-1961)

van *Ivan Sutherland*



[Opdracht_02_01] Bekijk https://www.youtube.com/watch?v=57wj8diYpgY

Het ontstaan van het Object

Oorspronkelijk idee komt uit

Sketchpad (1960-1961)

van *Ivan Sutherland*

“Records met gerelateerde procedures”

Het ontstaan van het Object

Oorspronkelijk idee komt uit

Sketchpad (1960-1961)

van *Ivan Sutherland*

Wat nog uit Sketchpad kwam: GUIs, pen interactie, vector graphics, constraint-based geometry,…

Het ontstaan van het Object

Simula 67

Allereerste programmeertaal met classes en objects

Het ontstaan van het Object

Smalltalk

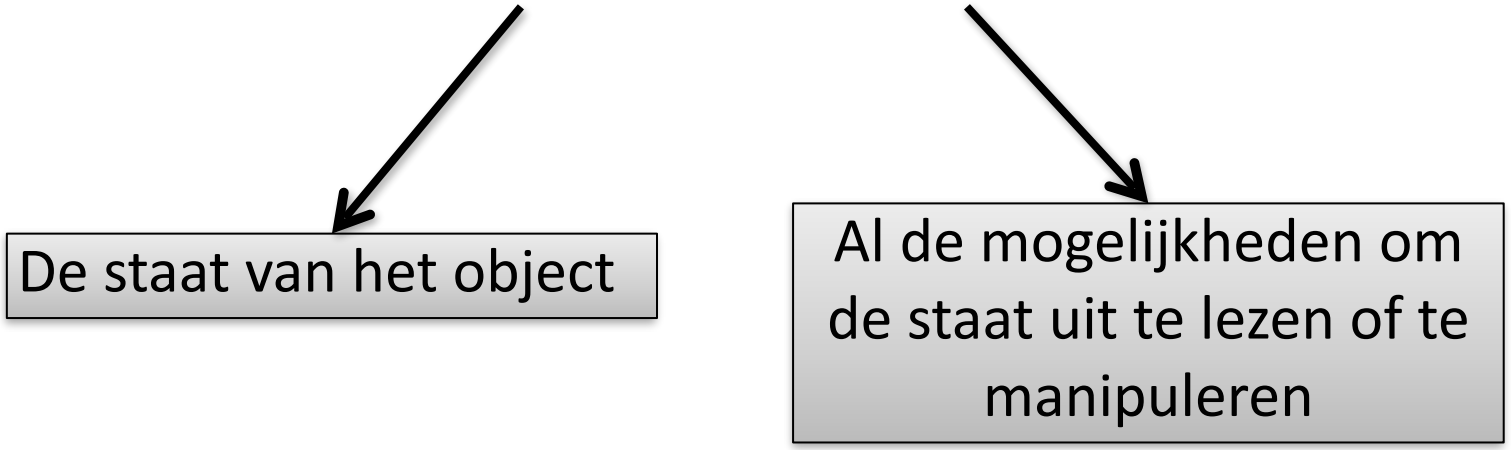
De eerste dynamische OO taal

Klassen en objecten worden “first class citizens” van de programmeertaal.

Het ontstaan van het Object

Smalltalk

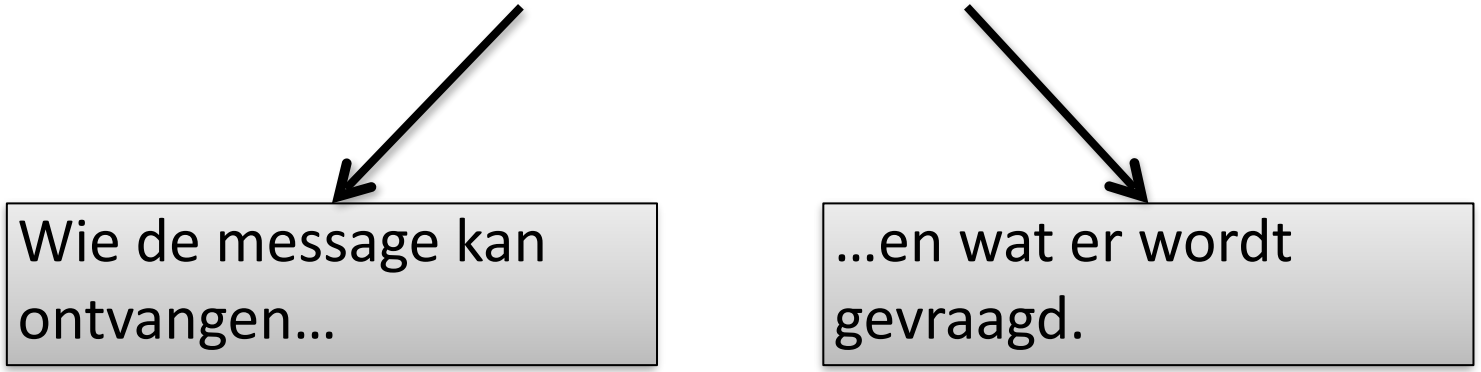
Object = Data + Operations (+identiteit)



Het ontstaan van het Object

Smalltalk

Message = Receiver + Requested Operation



Messages vs Methods

Dynamisch aan een doel gebonden (run-time)

Wordt verstuurd

Bijv. in Objective-C:
[calc bereken:x]

Statisch aan doel gebonden (compile time)

Wordt opgeroepen

Bijv. in Java:

calc.bereken(x)

equivalent

Java

It should be "simple, object-oriented and familiar"

It should be "robust and secure"

It should be "architecture-neutral and portable"

It should execute with "high performance"

It should be "interpreted, threaded, and dynamic"

Early Java white paper

Java

Object-Oriented

Class-based

Methods

General Purpose

Garbage Collection

Virtual Machine

Platform Independent

Syntax gebaseerd op C/C++

Java

No multiple inheritance

No operator overloading

Libraries, libraries, libraries

- Verzameling van voorgedefinieerde packages standaard beschikbaar
- Voor datastructuren: java.util.*
  http://docs.oracle.com/javase/6/docs/api/java/util/package-summary.html
- Voor GUIs: javax.swing.*
  http://docs.oracle.com/javase/6/docs/api/javax/swing/package-summary.html
- Voor netwerking: java.net
  http://docs.oracle.com/javase/6/docs/api/java/net/package-summary.html
- Math, Collections, XML, SQL, Graphics, Security, I/O…

```
import java.util.*;

public class Stuff{

        public static void main(String args[]){
                Vector v = new Vector();
                v.addElement(“blaai”);
                v.addElement(“vlaai”);
                v.addElement(“kers”);
                System.out.println(“size == “ + v.size() );
                System.out.println(v.elementAt(2));
        }

}
```

addElement

```
public void addElement(Object obj)
```

Adds the specified component to the end of this vector, increasing its size by one. The capacity of this vector is increased if its size becomes greater than its capacity.

This method is identical in functionality to the add(Object) method (which is part of the List interface).

Parameters:

`obj` - the component to be added.

See Also:

`add(Object)`, `List`

Vragen?

Een sprong in het diepe

Een sprong in het diepe

Vier op een rij in Java…

```
package vieropeenrij;

/**
 * This class starts and manages the game logic. It uses the {@VorMatrix} as a model for the
 * game board and {@VierOpEenRijVenster} as the user interface for the model.
 * @author Kris Luyten
 */
public class VierOpEenRij{

   private VorMatrix $vorMatrix;
   public static int ROWS = 6;
   public static int COLS = 7;
   public static enum FILL { RED , YELLOW , EMPTY };
   private FILL $turn;


   /**
   * The main method – the application starts here
   * @param args the command line arguments
   */
   public static void main(String[] args) {
     try{
        VierOpEenRij.getInstance().start();
      }catch(Exception e){
        //code om alle onverwachte exceptions op te vangen en elegant af te handelen
      }
```

```
 */
public static void main(String[] args) {
  try{
      VierOpEenRij.getInstance().start();
   }catch(Exception e){
      //code om alle onverwachte exceptions op te vangen en elegant af te handelen
   }
}

/**
* Starts the main thread of this application. Standard Java stuff.
*/
public void start(){
    java.awt.EventQueue.invokeLater(new RunnableImpl());
}

private class RunnableImpl implements Runnable {

   public RunnableImpl() {
   }

   public void run() {
       $vorMatrix = new VorMatrix();
   }
```

```
package vieropeenrij;

/**
 * VorMatrix is the model of the game and contains the current state of the game board
 * @author Kris Luyten
 */
public class VorMatrix extends java.util.Observable{
    private VierOpEenRij.FILL $vorMatrix[][] =
                new VierOpEenRij.FILL[VierOpEenRij.COLS][VierOpEenRij.ROWS];

    /**
     * Creates an empty model for the board
     */
    public VorMatrix(){
        initMatrix();
    }

    /**
     * Initializes game board with empty buckets
     */
    private void initMatrix(){
        for(int i=0; i<VierOpEenRij.COLS;i++)
            for(int j=0; j<VierOpEenRij.ROWS;j++){
                $vorMatrix[i][j]=VierOpEenRij.FILL.EMPTY;
            }
```

Vragen?

Nog enkele belangrijke ~~tips~~ ~~richtlijnen~~ ***regels***

1. Overerving is misschien wel de moeilijkste OO techniek om juist toe te passen

- Als je **twijfelt**: verkies ***associaties*** **boven overerving**
- Overerving *vermindert de flexibiliteit* van je code en creëert een *sterke afhankelijkheid* tussen klassen (“high coupling” ipv “low coupling”)

2. Methods zijn kort

niet meer dan 15 regels code

(er zijn uitzonderingen, maar die zijn *schaars*)

3. Inspectie en Mutatie

- Methodes duidelijk herkenbaar maken
  - **Inspector**: geeft een resultaat terug waarvoor de data van het object nodig is en verandert niets aan de staat van het object.
    “get” methods of getters
  - **Mutator**: verandert de staat van het object (wijzigt bijvoorbeeld een member variabele) en geeft geen resultaat terug.
    “set” methods of setters

3. Inspectie en Mutatie

- Simpele Opdeling
- Code wordt veel leesbaarder en duidelijker
- Minder bugs

  Methodes die zowel de staat van een object aanpassen als een waarde terug geven zijn een bron van fouten

Ondersteuning in sommige programmeertalen

- C# Accessors

```
public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
    }
}
```

- C++ **const** keyword

Vragen?

Oefeningen

- Vind je de opgaves niet duidelijk genoeg: *maak zelf ook keuzes*

Oefeningen

- **[opdracht_02_01]** TIJ3 Hoofdstuk 2: oefeningen 1 & 7
- **[opdracht_02_02]** Schrijf een klasse `Matrix`, waarmee je een 3x3 matrix kan voorstellen. Voorzie de nodige methods om twee matrices op te tellen en te vermenigvuldigen.
- **[opdracht_02_03]** TIJ3 Hoofdstuk 3: oefeningen 7 & 9

Oefeningen

- **[opdracht_02_04]**
  1. download Vier op een rij
  2. bestudeer de code
  3. gebruik je favoriete java-programmeer-omgeving en compileer de code
  4. voer de applicatie uit.