



Mobile Technologies & Smart Devices

2020-2021

overtref jezelf



Aan het eind van deze module begrijp je de principes van een (mobile) framework, en kan je een mobiele app maken.

Weekoverzicht

- ▶ Week 1-2
Verdieping en herhaling object orientatie
- ▶ Week 3-8
Mobile development
- ▶ Week 4
Geen les i.v.m. Pinksteren

Module

Lessen

- ▶ **Theorielessen**
- ▶ **Werkcolleges**
- ▶ **Online materialen (video's en docs)**

Toetsing

- ▶ **Praktijkopdracht**
(maken)
- ▶ **Tentamen (online)**
(theorie / open boek)
- ▶ Lees cursushandleiding op Cumlaude

Classes

- ▶ Wat is een class?
- ▶ Wat weten we van classes?

Classes

- ▶ Blauwdruk voor object
- ▶ Bevat declaratie fields/attributen (variabelen)
- ▶ Bevat methoden (functies)

Classes en objecten

- ▶ Class:
Blauwdruk voor object

```
class ...
```

- ▶ Object:
Instantie van class in geheugen

```
new ...
```

Inkapseling / Encapsulation

- ▶ Binnen een class maken we onderscheid tussen fields en methods voor intern en extern gebruik
- ▶ Intern: werking van de class
- ▶ Extern: handvatten voor de “buitenwereld” (die niet hoeft te weten hoe een class werkt, als hij hem maar kan “bedienen”)

Protection Levels

- ▶ private
(alleen class zelf heeft toegang)
- ▶ public
(iedereen heeft toegang)

Java

- ▶ Java lijkt qua syntax op Javascript
- ▶ Maar is Strongly typed
- ▶ Puur object georiënteerde taal
- ▶ Gebruikt in Android framework...
maar eerst gaan we console apps maken om met
objectoriëntatie te oefenen

Hello Java

```
package nl.hr.cmtprg03_7;

public class Main {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Namespace / Package

- ▶ Groepeert classes die bij elkaar horen
- ▶ Is onderdeel van de class-name, waardoor het makkelijker is om deze uniek te houden
- ▶ Hiërarchisch opgebouwd
- ▶ Moet gekoppeld worden als je classes uit een andere package wilt aanroepen met enkel hun class name (in Java m.b.v. import)

Hiërarchie?

Classes en geheugen

Class / Static memory
(declaratie met variabelen en functies)

Object / Instance memory
(type informatie, en waarden van instantievariabelen)

```
▶ public class Car {  
  
    private int speed;  
    private int position;  
  
    public Car() {  
        this.position = 10;  
        this.speed = 3;  
    }  
  
    void drive() {  
        this.position = this.getPosition() + this.speed;  
    }  
  
    public int getPosition() {  
        return position;  
    }  
}
```

```
▶ public static void main(String[] args) {  
  
    Car car = new Car();  
}
```

Static

- ▶ Directe toegang tot de class
- ▶ Truc om in een object georiënteerde taal zonder objecten te werken
- ▶ O.a. gebruikt wanneer je een method maakt die geen objectgegevens gebruikt (vgl. losse functies in php of js)

- ▶ Vb. Math.sin()

- ▶ In Console app gebruikt om 'kip-ei' probleem op te lossen

Static

- ▶ Static method net gezien in Console app
- ▶ Als methoden geen instantie-variabelen nodig hebben, is het overbodig daar een object voor aan te maken.
Declareer zulke methoden static om de compiler te vertellen dat deze methode direct via de class-definitie gebruikt wordt.
- ▶ Als een class alleen maar static methods bevat noemen we de class zelf static
- ▶ Geheugenmodel >>

Static

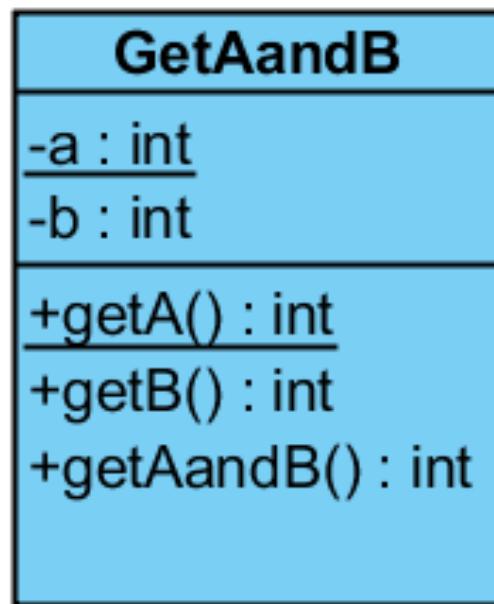
- ▶ Fields kunnen ook static zijn
 - ▶ Static geeft aan dat het field niet bij de instantie hoort, maar bij de class-definitie
 - ▶ Static methoden kunnen daardoor wel bij static fields (bij instantie variabelen kunnen ze niet, waarom?)
 - ▶ Objecten kunnen ook bij het static field, maar omdat er maar 1 is, wordt deze door alle objecten van dit type gebruikt
-
- ▶ Geheugenmodel >>

Static: toepassingen

- ▶ Vervanger van “ouderwetse” functie, bijv. formules (method)
- ▶ Gegevens bijhouden die alle objecten willen weten, bijv. een teller (field)
- ▶ Geheugenruimte besparen als alle objecten dezelfde informatie nodig hebben, bijv. een texture (field)

Class-diagram

GetAandB



Overerving / Inheritance

- ▶ Soms maak je een class die heel erg op een andere class lijkt, maar net iets meer moet kunnen, of die een dingetje (lees:methode) anders moet doen
- ▶ In dat geval gebruik je overerving: een subclass erft alle eigenschappen van zijn parent
- ▶ Subclass kan functionaliteit uitbreiden, of veranderen

Protection Levels

- ▶ **private**
(alleen class zelf heeft toegang)
- ▶ **public**
(iedereen heeft toegang)
- ▶ **protected**
(subclasses hebben toegang)
- ▶ <http://stackoverflow.com/questions/215497/in-java-difference-between-default-public-protected-and-private>

Verborgen methoden

- ▶ Een methode uit de parent wordt verborgen (hide) als de subclass deze ook heeft
- ▶ Het is dan toch mogelijk om deze te gebruiken door expliciet aan te geven dat je de methode uit de parent wil hebben (super in java)

Overloading van methoden

- ▶ Als methodes andere parameters (verschillend aantal of type) verwachten (signature) dan mogen ze dezelfde naam hebben
- ▶ De compiler kiest op basis van de signature de juiste methode

NB. In PHP werkt overloading uiteraard niet

Werkcollege

- ▶ Video's
- ▶ Installatie
- ▶ Opdrachten

Afbeeldingsbronnen

→ <https://mobilebcitcomputing.wordpress.com/2015/05/23/hello-world/>