Opleiding Applicatie Ontwikkelaar

Leerlijn Programmeren

Object Oriented Programming in Java

Domein A Level 3

Auteur: Mast, Erik

Datum: 14-6-2018

Inhoudsopgave

Overzicht

Level: Domein A Level 3
Duur: 4 weken
Methode: Weekplanning

Voorkennis

Module A Level 2, Module B Level 2

Kerntaak en werkprocessen

In deze leerlijn zullen we werken aan kerntaak 1 en kerntaak 2 .

Leerdoelen

Na het bestuderen zal je een simpel programma kunnen ontwerpen en bouwen met objecten.

Materialen

Er wordt gebruik gemaakt van de volgende studiematerialen:

- www.w3schools.com
- Jouw laptop
- Eclipse of Netbeans (of een andere Java ontwikkelomgeving)

Bronnen

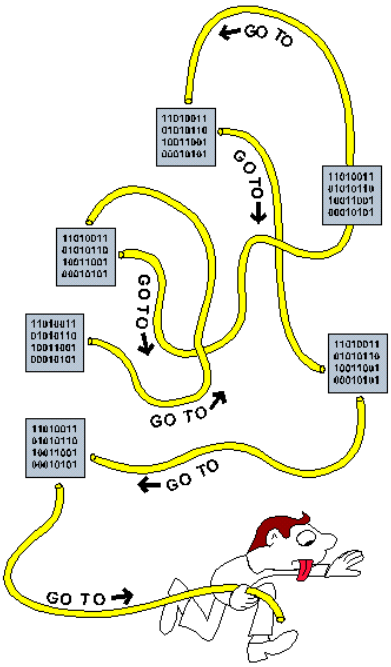
Zie bijlage Bronnen

Toetsing

Deze periode wordt afgesloten met een eindopdracht.

Theorie

Wat is een object?



In programma's heb je variabelen en regels code. Doorgaans doet de code iets met de variabelen (data) maar gaat het al heel snel door elkaar lopen en gaat de code op spaghetti lijken(->).

Om dit te voorkomen kun je zorgen dat de variabelen en code gegroepeerd worden. Een veelgebruikte groepering is OOP, Object Oriented Programming.

Binnen OOP onderscheiden we twee zaken wat dit betreft:

- Klassen
- Objecten (ook wel instanties genoemd)

Een klasse

Stel je een voorwerp voor, bijvoorbeeld een **Auto.** Een auto heeft een aantal eigenschappen, zoals merk, type en kleur. Je zou dus elke auto kunnen beschrijven aan de hand van die kenmerken. Elk kenmerk wordt opgeslagen in een variabele speciaal daarvoor gemaakt. Maar als je in je code die variabelen los neer zou zetten dan zou elke auto van het zelfde merk en type zijn en dezelfde kleur zijn.

Henry Ford said:
"A customer can have a car painted any color he wants as long as it's black"

Dat werkte in 1910, maar nu niet meer.

Je bedenkt dus een soort prototype, een definitie. Je wilt dat elke Auto een eigen kleur, merk en type heeft. En dat geheel van code en gegevens groepeer je in een **class**.

```
public class Auto {
    private String merk;
    private String type;
    private String kleur;
}
```

Je hebt nu een groepering van code en kenmerken, maar je kunt er niets mee want de velden zijn private, wat zoveel betekent dat een andere klasse er niet aan mag komen. En dat is eigenlijk wel logisch, want een van de voorwaarden voor Object georiënteerd programmeren is dat je de klasse zelf verantwoordelijk maakt voor zijn eigenschappen. Om toch van buiten eigenschappen te kunnen

veranderen gebruik je getters en setters, en dat zijn gewoon functies (we noemen ze hier methodes) waarmee je eigenschappen kun bekijken en aanpassen.

De klasse zou er dus bijvoorbeeld ongeveer zo uit kunnen zien, alleen voor de kleur zijn de methodes uitgewerkt.

```
public class Auto {
    private String merk;
    private String type;
    private String kleur;

    public String getKleur() {
        return kleur;
    }
    public void setKleur(String kleur) {
        this.kleur = kleur;
    }
}
```

De eigenschappen van een object worden over het algemeen aangepast door middel van setters. In het bovenstaande geval kun je alleen de kleur van buitenaf aanpassen door middel van het gebruik van de methode SetKleur.

En als de auto een Ford zou zijn uit 1910, dan zou de klasse er zo uit zien:

```
 public class Ford1910 {
    private String merk;
    private String type;
    private String kleur;

    public String getKleur() {
        return kleur;
    }

    public void setKleur(String kleur) {
        //de kleur die we meegeven negeren we gewoon
        //de auto is altijd zwart
        this.kleur = "zwart";
    }
}
```

Als je dit voorbeeld vergelijkt met het voorbeeld wat verder omhoog, kun je zien: je kunt de methode aanroepen met elke kleur die je wilt (bv setKleur("rood")) maar het object zal dit negeren. Want volgens het citaat kun je elke kleur vragen, maar de auto zal zwart worden.

Hier is heel duidelijk te zien: de klasse is verantwoordelijk voor de eigenschappen!

Instantie

Stel je voor dat je een auto koopt, op dat moment is duidelijk welke eigenschappen deze bezit. Voor jou is die auto nieuw, dus zo behandel je hem ook. Je maakt een object (of instantie) van het type Auto:

```
Auto mijnKar = new Auto();
```

Je gebruikt een variabele mijnKar, en die is van het type Auto (net als int, String etc) en daar maak je een nieuwe van.

Vervolgens heb je een mijnKar variabele (je eigen auto!) die je wel elke kleur zou kunnen geven.

```
mijnKar.setKleur("groen");
```

Hetzelfde of toch anders…

Nu heb je een **Auto** klasse bedacht, maar de eigenschappen die we tot nu aan dit object hebben gegeven zouden ook van toepassing kunnen zijn op een fiets. Of een Hoverboard. Of…

Dit is eigenlijk van toepassing op elk vervoermiddel, dus de klasse zou eigenlijk **Vervoermiddel** moeten heten. Een **Auto** klasse zou dan een **specialisatie** van een Vervoermiddel zijn, het is een Vervoermiddel met een beetje code of data meer. Denk bijvoorbeeld aan het gewicht, of de brandstof. Die twee zijn ook weer bepalend voor de wegenbelasting die je zou betalen. Voor een fiets of een hoverboard betaal je geen wegenbelasting, dus dat is geen eigenschap van een Vervoermiddel. De Vervoermiddel klasse noemen we dan ook **generalisatie**, en dat betekent dat het alleen de eigenschappen beschrijft die alle vervoermiddelen gemeenschappelijk hebben. Vervoermiddel is dan ook een **generieke** klasse. Hier onder zie je hoe die uitgewerkt is:

```
public class Vervoermiddel {
	private String merk;
	private String type;
	private String kleur;

	public String getMerk() { return merk; }
	public void setMerk(String merk) { this.merk = merk; }

	public String getType() { return type;	}
	public void setType(String type) { this.type = type; }

	public String getKleur() { return kleur; }
	public void setKleur(String kleur) { this.kleur = kleur; }

}
```

Dat een **Auto** klasse een specialisatie, of **afgeleide klasse**, is van Vervoermiddel, en alle eigenschappen erft, wordt in programmeerkringen **inheritance** (of overerving) genoemd. Hieronder een voorbeeld in code:

```
public class Auto extends Vervoermiddel{

	private int weight;
	private double tax;

	public double getTax() { return tax; }
	public void setTax(double tax) { this.tax = tax; }

	public int getWeight(){ return weight; }
	public void setWeight(int weight){ this.weight = weight; }
}
```

Alles wat je verder definieert in de klasse Auto is alleen van toepassing op een Auto en niet op een Vervoermiddel.

Een ander voordeel hiervan is dat je methodes of functies die je in de generieke klasse definieert, ook anders kunt definiëren in een afgeleide klassen:

```
public class Vervoermiddel {

	private int wheelCount;

	public int getWheelCount() { return wheelCount; }
	public void setWheelCount(int wheelCount) { this.wheelCount = wheelCount; }

	/* Eerdere code weggelaten */
}
```

```
public class Auto extends Vervoermiddel{

	@Override
	public int getWheelCount() { return 4; }

	/* Eerdere code weggelaten */

}
```

Hier staat in code: een vervoermiddel heeft een aantal wielen, hoeveel weet ik nog niet. Dat weet ik wel als ik hem ga gebruiken. Een auto is ook een vervoermiddel, maar die heeft altijd 4 wielen (dit is een aanname voor het gemak van de uitleg)

Alle regels in de code die beginnen met @ zijn compiler aanwijzingen. In dit geval geeft de regel @Override aan de compiler aan dat de onderstaande methode een overschrijving is van een bestaande methode. Je gebruikt in plaats van de methode in de generieke klasse (Vervoermiddel) dan de methode in de afgeleide klasse (Auto).

Als de compiler de originele overschreven methode niet kan vinden geeft hij bij het compiler een foutmelding. Als je de regel met @Override weglaat verandert de werking van het programma niet. Het gebruik van @Override kan moeilijk opspoorbare logische fouten voorkomen, zoals een typefout in de naam van de overschreven methode.

Een speciale klasse

Java kent ook een heleboel speciale klassen.

De belangrijkste is wel de Main klasse. Deze heeft een methode die er als volgt uit ziet:

```
public static void main(String[] args) {
}
```

Deze methode wordt aangeroepen zodra je je programma start. Dit heb je waarschijnlijk eerder gezien. Het woord **static** is wel opvallend: het betekent eigenlijk dat je de methode aan kunt roepen zonder dat je een object hoeft te maken. Als je er goed over nadenkt is dit ook wel logisch, anders zou je een object moeten maken voor je methode aan kunt roepen, en een object kun je niet zo maar aanmaken, dat moet weer in een methode in een object gebeuren. Dat is eigenlijk hetzelfde als de vraag “wat was er eerder, de kip of het ei”.

Hoe werkt het nu in het echt? Zo dus:

```
public class Main{

	public static void main(String[] args) {
		Auto auto = new Auto();

		//setters overgenomen van Vervoermiddel-klasse
		auto.setMerk("Mercedes");
		auto.setType("S350d");
		auto.setKleur("selenitgrau metallic");

		//setters van Auto-klasse
		auto.setWeight(1875);
		auto.setTax(570.00);
		System.out.println("De wegenbelasting bedraagt "+
			mijnKar.getTax()+" euro per maand");
	}

}
```

Regels voor OOP

Voor het programmeren in OOP zijn er een aantal regels:

Een klasse heeft één taak.

Dit betekent eigenlijk dat je niet teveel verschillende dingen door één klasse moet laten doen. Dit levert vaak verwarrende code op en is vaak een teken dat je je code niet goed overdacht hebt, of direct bent begonnen met programmeren.

Een klasse is zelf verantwoordelijk voor zijn data.

Wijzigen van data in een klasse gaat via een methode. Een klasse kan zelf dan bepalen of de aangeboden eigenschappen wel passend zijn, en ze eventueel aanpassen of een foutmelding teruggeven. Voorbeeld: een auto kan best 4 wielen hebben of 3 of misschien wel 6 maar 20 is echt teveel. Dat zou meer een aanhanger voor een grote truck zijn.

In Java: één Mainobject

Een Java programma heeft altijd één mainobject waarin het programma gestart wordt, en alleen de noodzakelijke objecten worden aangemaakt (maar meer ook niet!)

Opdrachten

Opdracht 1

a. Maak een simpele klasse genaamd Persoon en zorg dat je daarin voornaam en achternaam kunt bewaren.
b. Voeg de geboortedatum toe
c. Maak een methode die teruggeeft hoe oud de persoon is in dagen.

Opdracht 2

a. Werk Auto en Vervoermiddel klassen uit Lesbrief 1 uit, gebruik echter wel eigenschappen en methodes in het Nederlands.
b. Voeg aan het Auto een functie toe waaraan je kunt vragen wat een bepaalde instantie van een Auto kost aan wegenbelasting. Daarvoor zul je misschien velden moeten toevoegen en je zult moeten gaan kijken op de website van de Belastingdienst hoe zij de wegenbelasting berekenen voor een benzine auto. Je hoeft alleen maar te kijken voor Drenthe.

Opdracht 3

a. Als je er goed over nadenkt: de wegenbelasting berekening met bijbehorende getallen zouden in een aparte klasse moeten. Werk deze klasse ook uit.

Inleveren

Laat de werking van opdracht 2 en 3 zien aan de docent. Ga pas verder als het goed werkt, en lever aan het eind de code in samen met de eindopdrachten.

Een userinterface

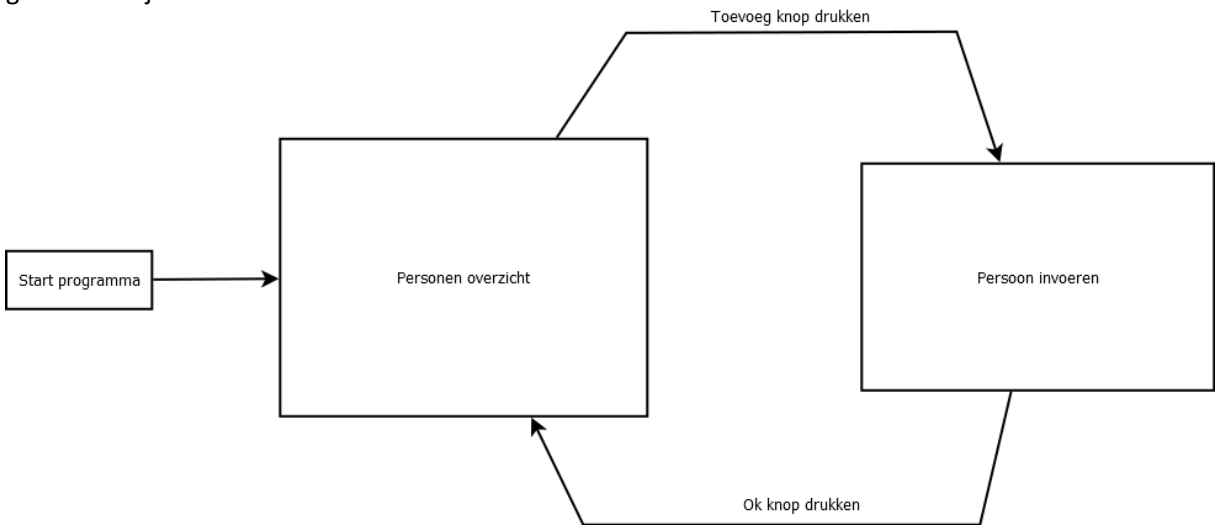
Bestuderen

Zie Bronnen: Swing tutorials. Volg één (kies zelf welke) van de genoemde tutorials.

Opdrachten

Opdracht 4

a. Maak een lijst waarin je objecten van het type Persoon kunt opslaan, zodat je ze gemakkelijk kunt raadplegen.
b. Maak in het JFrame een JList zodat je alle personen die je hebt ingevoerd onder elkaar kunt zien.
c. Maak in Swing een JFrame met daarop twee invoer velden (voornaam en achternaam) en een knop, zodat je personen kunt toevoegen met die knop.
   Als hulp staat hieronder een "flowchart", dat is een tekening waarin je kunt zien wat er gebeurt als je iets doet.



Inleveren

Laat de werking van opdracht 4 zien aan de docent. Ga pas verder als het goed werkt, en lever aan het eind de code in samen met de eindopdrachten.

Eindopdrachten

Eindopdracht 1

Maak een memory spel (van 4x4 “kaartjes”) waarbij je alle opgedane kennis verwerkt. Probeer waar nodig klassen te maken zodat de code overzichtelijk wordt. Denk goed na over welke functionaliteit bij welke data hoort en maak daar een klasse van. Denk in dingen (objecten en klasses) en niet in programmaregels.

Hints & Tips voor opdracht 1

- Gebruik een JButton om een plaatje op te zetten, dan heb je in één keer de functionaliteit die je nodig hebt.
- Voor het willekeurig genereren van welk plaatje op welke button komt, kun je het beste een aparte klasse maken, je kunt extra punten verdienen als je daarvoor een testklasse maakt (Google op JUnitTest)

Eindopdracht 2

a. Kopieer de code uit opdracht 4 naar een nieuwe map.
b. Ga uitzoeken hoe je een verbinding maakt met een MySQL server in Java. In de Bronnen vind je een link naar een tutorial. Deze informatie moet je in een class verwerken zodat je methodes krijgt om een Persoon te bewaren en op te halen.
c. Maak in deze klasse een methode die alle Personen uit de database zou kunnen halen.
d. Gebruik deze methodes om de GUI uit opdracht 4 Personen in de database op te slaan en op te halen.

Inleveren

Lever de code van alle opdrachten in, keurig in verschillende mapjes, in één zip file.
Vergeet niet

- de gebruikte plaatjes, en let er ook op dat je plaatjes niet een pad hebben naar je eigen computer.
- om de SQL dump vanuit PhpMyAdmin toe te voegen(Eindopdracht 2).

Bronnen

MySQL en Java

- ✓ **MySQL and Java JDBC - Tutorial**
  http://www.vogella.com/tutorials/MySQLJava/article.html

Swing tutorials

- ✓ **Creating a GUI With JFC/Swing (The Java© Tutorials)**
  https://docs.oracle.com/javase/tutorial/uiswing/index.html

- ✓ **Swing Tutorial**
  https://www.tutorialspoint.com/swing/

Over OOP

- ✓ **S.O.L.I.D: The First 5 Principles of Object Oriented Design**
  https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design