Inleiding Programmeren
College 5

Universiteit van Amsterdam

Robin de Vries

18 september 2017

Huishoudelijke mededelingen (1)

- Wie mail je wanneer?
- Nabespreking staafdiagram woensdag
- Volgende week woensdag oefentoets!
- Quiz na de pauze

Huishoudelijke mededelingen (2)

Denk je: "Hee ik ben klaar"?!

1. Lees de styleguide nog een keer door
2. Lees de rubric nog een keer door
3. Lees de opdracht nog een keer door
4. Laat een studentassistent naar je code kijken, deze beoordeelt of er nog grove fouten in je code zitten
   - Goedgekeurd[1]? Je kan de laptopcolleges van deze week verder overslaan
   - Afgekeurd? Je moet je code verbeteren en mag nog niet weg

Let op: nakijkpractica mag je nooit overslaan!

Vragen hebben voorrang op laten beoordelen.

[1] Dit is geen garantie voor een 10

Al deze regels zorgen voor blije TAs

Inhoud

Objecten

Java is een object geörienteerde programmeertaal: alle functies (methoden) en variabelen zijn onderdeel van een klasse (`class`) of een `object`.

Een klasse is een definitie van een datatype: een blauwdruk of bouwtekening.

Een object is een instantie van een klasse.

Laten we dit nader bekijken met wat voorbeelden!

De punt klasse

```
class Punt
{
    double x;
    double y;
}
```

Kenmerken van klassen

- Naam begint met een hoofdletter
- Bevat variabelen: `class variables` en `instance variables`
  en functies: `methoden`.
- Deze variabelen krijgen de "standaard" waarden: 0, 0.0, false of null.
- Indien `public`, opgeslagen in een bestand met dezelfde naam.
  `class Punt` dus in `Punt.java` (later hierover meer)

Gebruik van deze Punt klasse

```
class College
{
   public static void main(String args[])
   {
        Punt punt1 = new Punt();

        punt1.x = 1;
        punt1.y = 1;

        System.out.println(punt1.x + ", " + punt1.y);
   }
}
```

Wat gebeurt hier?

We maken een variabele met de naam `punt1` aan van het type `Punt`.

Vervolgens vragen we Java een object te instantiëren met behulp van het **new** keyword.

Daarna kunnen we de variabelen van de **instantie** van de Punt klasse aanpassen en weer uitlezen. Zo'n instantie wordt ook een **object** genoemd.

Methoden

Klassen kunnen ook methoden bevatten, dat ziet er als volgt uit:

```
class Punt
{
    double x;
    double y;

    double afstandTotOorsprong()
    {
        return Math.sqrt(x * x + y * y);
    }
}
```

En dan kunnen we deze methode aanroepen:

```
Punt punt1 = new Punt();
punt1.x = 1;
punt2.x = 2;
System.out.println(punt1.afstandTotOorsprong());
```

Inhoud

Constructoren

Met behulp van een constructor kunnen we gegevens in het object laden direct nadat het is gemaakt.

Hiervoor maken we een methode aan die dezelfde naam het als het object zelf, en geen return value heeft.

Constructor voor de punt klasse

```
class Punt {
    double x;
    double y;

    Punt(int getal1, int getal2)
    {
        x = getal1;
        y = getal2;
    }
}
```

En dan kun je een Punt object ook maken op deze manier:

```
Punt punt1 = new Punt(1, 1);
```

Standaard constructor vervalt

Als we zelf een constructor definieren vervalt de standaard constructor van Java:

```
error: constructor Punt in class Punt cannot be applied to given types;
        Punt punt = new Punt();
                    ^
```

Constructor zonder argumenten

Je kunt ook een constructor zonder argumenten maken:

```
class Punt {
    double x;
    double y;

    Punt()
    {
        x = 13;
        y = 27;
    }
}
```

En dan zal ieder nieuw punt-object standaard 13 en 27 voor de x en y waarde krijgen.

```
Punt punt1 = new Punt();
```

Standaardwaarden voor klasse en instantatievariabelen

Dit zou overigens hetzelfde effect hebben:

```
class Punt {
    double x = 13;
    double y = 27;
}
```

En dan verderop:

```
Punt punt1 = new Punt();
```

Inhoud

Overschaduwing van instantiatievariabelen

Zoals we net al zagen hebben we wel eens dat we parameters dezelfde naam willen geven als de instantiatievariabelen:

```
class Punt {
    double x;
    double y;

    Punt(int x, int y)
    {
        x = x;
        y = y;
    }
}
```

Maar in dit geval overschaduwen de parameters de instantiatievariabelen :-(

Overschaduwing oplossen met behulp van het this keyword

Dit kunnen we oplossen door de speciale variabele **this**. Welke een referentie naar het eigen object bevat.

```
class Punt {
    double x;
    double y;

    Punt(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
```

Inhoud

Method overloading

Wat weten jullie nog van method overloading?

Constructor overloading

Net zoals we meerdere definities van dezelfde methode kunnen hebben, kunnen we ook meerdere definities van een constructor hebben: **constructor overloading**.

```
class Punt {
    double x;
    double y;

    Punt(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    Punt(int x, int y)
    {
        this.x = x + 0.5;
        this.y = y + 0.5;
    }
}
```

Constructor chaining

- Vaak lijken constructoren erg op elkaar en zorgt de strategie van de vorige slide voor veel dubbele code.
- Dit kunnen we oplossen door in een constructor een andere constructor aan te roepen.
- Op deze manier ketenen we constructoren aan elkaar vast: **constructor chaining**.
- Een andere implementatie van de constructor kun je aanroepen met behulp van: `this(parameters)`.

Constructor chaining (2)

```
class Punt {
    double x;
    double y;

    Punt(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    Punt(int x, int y)
    {
        this(x + 0.5, y + 0.5);
    }
}
```

Inhoud

Punten zijn boring

Laten we voor we verder gaan het voorbeeld verder uitbreiden:

- Punt: afstand tussen twee punten
- Lijn(stuk) tussen twee punten

Afstand tussen twee punten

```
/* Bereken de afstand tot het gegeven punt.
 * In geval van punten p1(x1, y1) en p2(x2, y2) is de afstand d:
 * d = sqrt( (x2-x1)^2 + (y2-y1)^2)
 */
double afstandTot(Punt punt)
{
    return Math.sqrt((punt.x - x) * (punt.x - x) +
                     (punt.y - y) * (punt.y - y) );
}
```

Lijn

Een lijnstuk met 2 punten:

```
class Lijn
{
    Punt p1;
    Punt p2;

    Lijn(Punt p1, Punt p2)
    {
        this.p1 = p1;
        this.p2 = p2;
    }
}
```

Hoe kunnen we een constructor maken die 4 punten coördinaten neemt als argumenten?

Met constructor chaining!

```
class Lijn {

    // de rest van de code van net

    Lijn(int x1, int y1, int x2, int y2)
    {
        this(new Punt(x1, y1), new Punt(x2, y2));
    }
}
```

En de lengte van de lijn?

```
class Lijn {

    // de rest van de code van net

    double lengte()
    {
        return p1.afstandTot(p2);
    }
}
```

Middelpunt van een lijn

```
class Lijn {

    // de rest van de code van net

    /* Bereken het punt in het midden van de lijn
     * Gegeven punten p1(x1, y1) en p2(x2, y2)
     * is het middelpunt mp((x1 + x2) / 2, (y1 + y2) / 2)
     */
    Punt middelpunt()
    {
        return new Punt((p1.x + p2.x) / 2.0, (p1.y + p2.y) / 2.0);
    }
}
```

Inhoud

Printen van een object

Als we het middelpunt van net proberen te printen krijgen we iets interessants op het scherm:

```
Punt punt1 = new Punt(1, 1);
Punt punt2 = new Punt(7, 9);
Lijn lijn1 = new Lijn(punt1, punt2);

System.out.println(lijn1.middelpunt());
```

```
Punt@2a139a55
```

toString() zelf definiëren

We kunnen deze uitvoer ook aanpassen door zelf een methode `public String toString()` te definiëren[2]:

```
class Punt
{
    // de rest van de code van net

    public String toString()
    {
        return "Punt (" + x + ", " + y + ")";
    }
}
```

```
Punt (4.0, 5.0)
```

[2] De public modifier is hier verplicht omdat we de standaard toString() overriden, voor de rest mag je het negeren.

Equals

Zoals we eerder bij Strings al hadden geleerd: niet-primitieve datatypen mogen we niet vergelijken met behulp van de == operator!

```
Punt punt1 = new Punt(1, 1);
Punt punt2 = new Punt(1, 1);

System.out.println(punt1 == punt1);
System.out.println(punt1 == punt2);
```

```
true
false
```

Eigen equals implementatie

```
class Punt {

    // de rest van de code van net

    /* Deze vergelijking is wel wat naief aangezien x en y
     * doubles zijn we dus wellicht een epsilon willen gebruiken.
     */
    boolean equals(Punt punt)
    {
        return x == punt.x && punt.y == y;
    }
}
```

Inhoud

Klassevariabelen

- De variabelen (x, y) zoals we die net zagen behoren bij het object, de instantie van de klasse.
- Daarom worden ze instantiatievariabelen genoemd.
- Je kunt ook variabelen definieren die behoren bij de klasse.
- Deze heten: **klassevariabelen**.
- Klassevariabelen zijn dus gelijk voor alle objecten!
- Je definieert een klassevariabelen met de `static` modifier.

Klassevariabelen

Voorbeeld:

```
class Punt
{
    double x;
    double y;
    static int aantalPunten = 0;

    Punt(int x, int y)
    {
        this.x = x;
        this.y = y;
        aantalPunten++;
    }
}
```

Klassevariabelen (2)

Wat is de uitvoer van onderstaande code?

```
Punt punt1 = new Punt(1, 1);
Punt punt2 = new Punt(7, 9);

System.out.println(punt1.aantalPunten);
System.out.println(punt2.aantalPunten);
```

```
2
2
```

Klassevariabelen (3)

Omdat `aantalPunten` een klassevariabele is, hebben we dus geen instantie nodig om hem te lezen of schrijven:

```
System.out.println(Punt.aantalPunten);
```

```
2
```

Static methoden

Zoals we vorige week al deden, kunnen we ook `static` methoden maken.

Omdat deze methoden niet bij een instantie horen, kunnen ze ook niet bij de instantiatievariabelen.