Hoofdstuk 8: Klassen definiëren

All you do in Java is define classes, make objects of those classes, and send messages to those objects.

Bruce Eckel

Inleiding

In het hoofdstuk over **object georiënteerd programmeren** hebben we het reeds gehad over objecten.

- ➔ We hebben kennis gemaakt met eenvoudige objecten
- ➔ Eigenschappen en methoden

In **dit hoofdstuk** gaan we dieper in op het zelf maken van klassen.

Rectangle

Deze klasse Rectangle heeft enkele properties:

- Height
- width
- x
- y

Alsook enkele methoden:

- **set**Height(int height)
- **set**Width(int width)
- **set**X(int x)
- **set**Y(int y)
- **get**Height( )
- **get**Width( )
- **get**X( )
- **get**Y( )

De declaratie van een klasse

De **declaratie** van de **klasse** gebeurt in de 1ste regel

Syntax

- **class Rectangle {**
  Minimum syntax
  }

Algemene klasse definitie

public abstract final **class ClassName** extends SuperClassName implements InterfaceName

De klasse omschrijving : body

We hebben reeds gezien dat **objecten** een **verzameling** zijn van

- ➔ **eigenschappen** (variabelen)
- ➔ **methoden** (codeblokken)

We noemen deze ook **member-variables** & **member-methods.**

In de **body van** de **klasse** worden deze members **gedefinieerd**.

Daarnaast worden er **eventueel** een **aantal constructors gedefinieerd.**

Herhalen

De **BODY** bevat dus:

- ➔ Eigenschappen
- ➔ Methoden
- ➔ Constructors

Eigenschappen

Onze Rectangle heeft **vier** eigenschappen

1. x
2. y
3. height
4. width

Het is convention de variabelen eerst te definiëren als we een klasse maken.

```
public class Rectangle{
        public int x;
        public int y;
        public int height;
        public int width;
}
```

Methoden

- ➔ Methoden zijn codeblokken verbonden aan een object.
- ➔ d.m.v. Methoden kunnen andere objecten boodschappen sturen naar dit object en eventueel een resultaat terugkrijgen.
- ➔ De methode heeft net als een eigenschap verschillende toegangsniveaus.
- ➔ Indien niet gespecificeerd bij de creatie is het package-toegangsniveau.
- ➔ Welke waren weer de andere toegangsniveau’s?

Body van de methode

Na de declaratie van de methode volgt er het codeblok van de methode.

- ➔ codeblok wordt uitgevoerd bij het aanroepen van de methode.

Binnen dit codeblok kunnen we variabelen declareren.

- ➔ Men noemt dit lokale variabelen.
- ➔ Lokale variabelen hebben een bereik (scope) dat zich uitstrekt over het codeblok waarin ze gedefinieerd zijn en elk ander codeblok dat daar deel van uitmaakt.

In tegenstelling tot member variabelen moeten lokale variabelen expliciet geïnitialiseerd worden alvorens ze men kan gebruiken

Parameters

- Voor iedere parameter wordt het **type** en de **naam** van de variabele gedefinieerd.
- Meerdere parameters worden gescheiden door komma's
- Parameters kunnen beschouwd worden als lokale variabelen, die gedeclareerd en geïnitialiseerd worden tijdens het aanroepen van de methode

```
public class Rectangle{

        public int x;
        public int y;
…

        public void setPosition(int x, int y) {
                this.x = x;
                this.y = y;
        }
}
```

Parameter

- ➔ Zoals bij lokale variabelen kan een parameter ook dezelfde naam hebben van een member-variabele
- ➔ Member variabele wordt verborgen
- ➔ toch bereikbaar via this.

Waarden teruggeven

- ➔ Iedere methode heeft de mogelijkheid om een bepaalde waarde terug te geven aan het object dat de methode aanroept.
- ➔ Het datatype van deze waarde wordt gedefinieerd in de declaratie van de methode.
- ➔ Indien **geen waarde** wordt teruggegeven wordt **void** als returntype gebruikt

code more they said, it will be fun they said.

Maak onderstaande methode in de klasse Rectangle

- ```
  public double getArea(){
      return height * width;
  }
  ```

Daarna geef volgende code in een App Klasse.

```
public static void main (String[ ]args) {
    Rectangle rect = new Rectangle ( );
    rect.setWidth(14);
    rect.setHeight(55);
    double area = rect.getArea( );
    System.out.println(area);
}
```
```

Setters and Getters

```
public setHeight(int height){
        this.height = (height < 0) ? -height:height;
}

public class RectangleApp{
…
        public static void main(String[ ] ...args){
                …
                rect.height = -10;
                rect.setHeight(-10);

        }
}
```

Method name overloading

- Java biedt de mogelijkheid om verschillende methoden dezelfde naam te geven.
- De methoden met dezelfde naam worden onderscheiden door het aantal parameters en het datatype van de parameters.
- De compiler kiest de juiste methode op basis van de parameters die worden meegegeven.

Constructors

Iedere klasse heeft een constructor.

Een constructor bevat de code die wordt uitgevoerd als een object gecreëerd wordt met de new operator.

- ➔ Deze code wordt gebruikt om het object te initialiseren.

- ➔ Deze code kan maar 1 keer opgeroepen worden, dit is tijdens de initialisatie van een instantie.

```
[public | private | protected] ClassName(parameters){
        //initialisatie code
}
```

- ➔ Ziet eruit als een methode
  - Geen returntype
  - Naam is gelijk aan de klasseNaam
  - Hoofdletter (CASE SENSITIVE)

Constructors

Een klasse kan meerdere constructors hebben.

- ➔ constructor **overloading**.
  - ◆ Verschil in aantal en type parameters.

Als er geen expliciete constructor gedefinieerd wordt, krijgt de klasse een impliciete standaard constructor.

- ➔ Zodra er een constructor expliciet gespecificeerd wordt, vervalt de impliciete standaard constructor.
  - ◆ Als je deze toch wil zal je hem expliciet moeten maken.

Sorry still constructors.

Het is mogelijk om een constructor een andere constructor te laten aanroepen met **this( )**

➔ Constructor call: gebeurd steeds op eerste regel.

➔ Deze referentie variabele verwijst steeds naar het eigen object.

Voordeel dat de eigenlijke initialisatiecode slechts eenmaal geschreven wordt.

➔ onderhoudsvriendelijk

Instance en Class members

Inleiding

Instance variabelen en **methoden**

➔ Noemen we instance members

Klasse variabelen en **methoden**

➔ Noemen we class members

Instance variabelen

Voor ieder object dat we creëren met de new operator wordt een unieke kopie van de instance variabelen gemaakt

- ➔ Deze variabelen bestaan niet zonder een object.
- ➔ Om de variabele te manipuleren moeten we steeds aangeven om welk object het gaat.

Dat is OOP: we manipuleren de data via objecten

Instance variabele

Indien we de instance variabele **intern** in het object willen gebruiken volstaat het om **enkel** de **naam** van de **variabele** te gebruiken.

We kunnen de naam laten voorafgaan door de this-referentie variabele.

➔ De compiler veronderstelt automatisch **this**

➔ **Tenzij** we een lokale variabele hebben die de instance variabele verbergt

Initialisatie van instance variabelen.

Instance variabelen kunnen op 3 manieren geïnitialiseerd worden

1. Tijdens de **declaratie**.
2. In een **initialisatie blok**.
3. In een **constructor**.

Klasse variabelen

Zijn gemeenschappelijk voor alle objecten van dezelfde klasse.

Er bestaat maar een 1 variabele die gedeeld wordt.

➔ Worden gedefinieerd door de component **static** tijdens de declaratie.

public static final int **ANGLES** = 4;

Klasse variabelen worden gebruikt door de klassenaam te laten volgen door de “.” separator en de naam van de variabele

System.out.println(Rectangle.ANGLES);

Klasse variabelen

Klasse variabelen kunnen op twee manieren geïnitialiseerd worden:

1. Tijdens de declaratie
2. In een static initialisatie blok

static codeblok wordt eenmalig uitgevoerd als de klasse voor het eerst gebruikt wordt.

Klasse variabelen

public static int counter = 0;

Telkens we een object creëren wordt deze count verhoogt tijdens het uitvoeren van het static initialisatie blok.

➔ We kunnen dit ook verhogen via de **constructor** met de **meeste parameters**. Dit omdat we in de constructors met minder parameters telkens naar die met de meeste doorverwijzen.

Klasse methoden

➔ Kunnen enkel gebruik maken van klasse variabelen en lokale variabelen die binnen de methode gedeclareerd worden.

- public static int getCount( ){
  return counter;
  }

➔ Worden aangeroepen op basis van een klasseNaam.

System.out.println(Rectangle.getCount( ));

Instance methoden

- Bepalen welke boodschappen een object kan ontvangen.
- Kunnen zowel gebruik maken van klasse variabelen als van instance variabelen.
- Moeten worden aangeroepen aan de hand van een concreet object.
- Zij kunnen enkel gebruik maken van de instance variabelen van het object waartoe ze behoren
  - rect.setPosition(40,50);

Utility class

- ➔ Sommige klassen hebben enkel statische methodes!
- ➔ We noemen deze utility classes.