Overerving en klassenhiërarchie

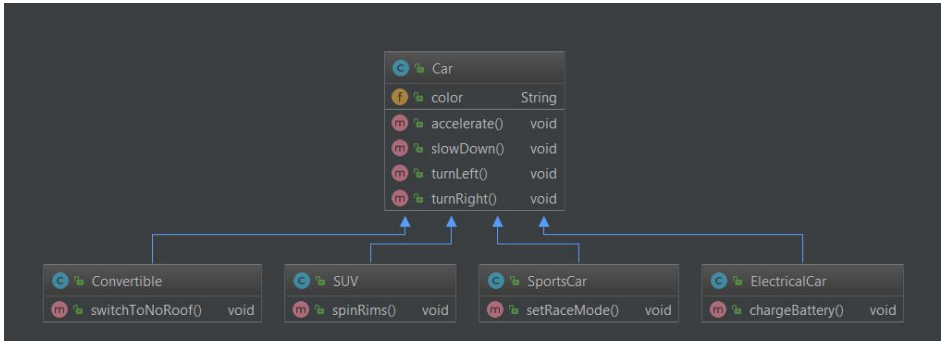
MEGABOOK Hoofdstuk 10

Inleiding

- In het vorige hoofdstuk hebben we gesproken over associaties als manier om code te kunnen hergebruiken (code reuse).
  - ➔ Echter niet de enige manier.
- Een andere manier hiervan is overerving (inheritance)

Subklassen en superklassen

- Klassen kunnen gebaseerd zijn of afgeleid zijn van andere klassen.



- Subklassen die zijn afgeleid van een superklasse erven alle eigenschappen en methoden van die klasse.
- Ze voegen bovendien een aantal specifieke eigenschappen en methoden toe.
- Kunnen zelfs de implementatie van bepaalde methoden @Overide.

Overerving.

- De terreinwagen erft alle eigenschappen en methoden van de klasse auto over.
- Met een terreinwagen kan je rijden zoals je met iedere auto rijdt, omdat alle kenmerken en gedragingen (eigenschappen en methoden) dezelfde zijn als die van een gewone auto.
- De code wordt in een superklasse gemaakt en vervolgens hergebruikt in één of meerdere subklassen.

Subclass methods | properties

toevoegen

- ❏ Tegelijkertijd voegt de klasse terreinwagen ook een aantal eigenschappen en methoden toe die specifiek zijn voor de terreinwagen.
- ➔ vb 4x4 aandrijving
- ➔ kruipversnelling
- ➔ bull bar

Vervangen

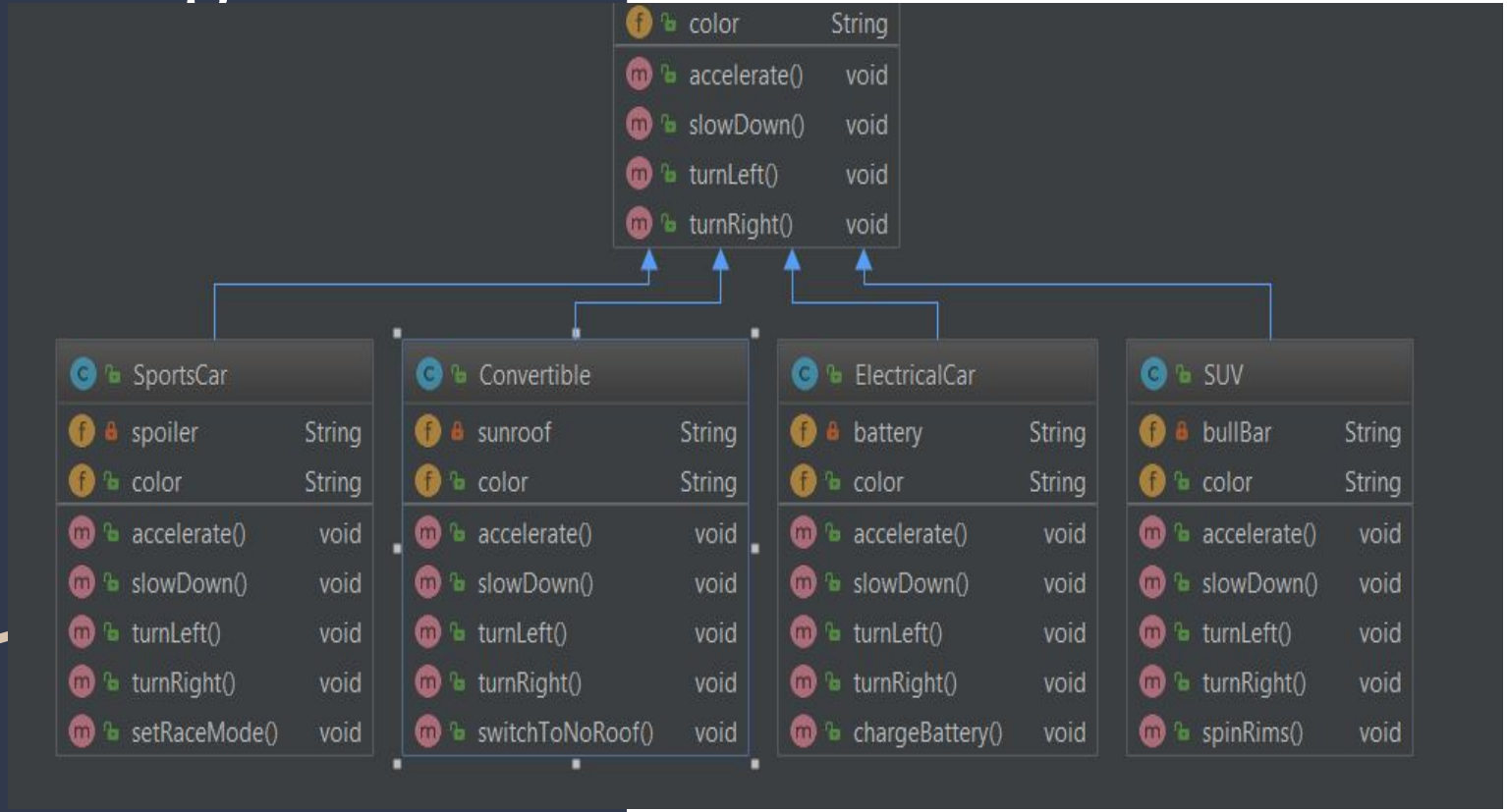
Subklassen kunnen niet enkel methoden toevoegen, ze kunnen ook bepaalde methoden vervangen (override).

- ➔ Bij een gewone auto met verbrandingsmotor kan men de wagen doen sneller rijden door de methode accelerate( ).

- ➔ Bij een verbrandingsmotor zal deze implementatie anders zijn dan bij pakweg een elektrische motor.

    - ◆ We gaan dus bij de elektrische wagen de implementatie moeten wijzigen!

UML - diagram Car



Klassenhiërarchie

- Op de weg vinden weg vinden we ook nog andere voertuigen.
  - ➔ Bus
  - ➔ Vrachtwagen
- Hebben met onze auto gemeen dat ze ook voertuigen zijn.
- Alle wegvoertuigen hebben een aantal gemeenschappelijke kenmerken.
- Wegvoertuig is een superklasse van de klasse Auto op dezelfde wijze als Auto de superklasse is van terreinwagen.
- Zo is de methode accelerate( ) een methode van de klasse Voertuig maar iedere subklasse heeft een andere implementatie van deze klasse.

Abstracte klassen.

- Van de klasse vehicle op zich bestaan er eigenlijk geen concrete objecten (instanties).
- Deze klasse definieert enkel een aantal eigenschappen en methoden die gemeenschappelijk zijn voor de subklassen.
- Daarom noemen we deze klasse een abstracte klasse.
- Abstracte klassen worden dus gebruikt om algemene kenmerken en methoden voor een bepaald soort objecten te definiëren.
- bedoeling is dat van een abstracte klasse andere klassen worden afgeleid.
  - ➔ Hiervan worden dan uiteindelijk klassen gemaakt.

Subklassen definiëren in Java.

- Indien we in Java de superklasse van een klasse willen definiëren maken we gebruik van het woord extends.

- class Subclass **extends** SuperClass{
  ...
  }

- De subklasse **erft** hierbij de eigenschappen en methoden over van de SuperClass.

- Bovendien kan de subklasse eigenschappen en methoden **toevoegen** en eventueel de implementatie van bestaande methodes **vervangen** (override)

Eigenschappen van subklassen.

- De subklasse erven alle eigenschappen van hun superklasse over, zowel **instance - eigenschappen** als **klassen -eigenschappen**.
  - ➔ De toegang tot deze eigenschappen hangt af van het toegangsniveau.
  - ➔ Protected en public eigenschappen zijn toegankelijk vanuit subklasse in ander pakket.

Toevoegen | vervangen of verbergen van eigenschappen.

- Een subklasse kan eigenschappen toevoegen die niet bestaan in de superklasse.

- Men kan in de klassen definitie van de subklasse een variabele declareren met dezelfde naam als een variabele uit de SuperClass.

  - ➔ de variabele van de superklasse wordt in dit geval verborgen.

```
public class Rectangle {
        public String description = “Rectangle”;
}

public class Square extends Rectangle {
        public String description = “Square”;
}
```

Methoden van subklassen.

- De subklasse erven alle methoden van hun superklasse over, zowel **instance - methoden** als **klassen - methoden**.
- De toegang tot deze methoden hangt wederom af van het toegangsniveau.

Vervangen van methoden.

- De klasse Rectangle heeft methoden om de hoogte en de breedte van een rechthoek afzonderlijk in te stellen.
- Voor een vierkant moeten we er steeds voor zorgen dat de hoogte gelijk is aan de breedte.
  - ➔ Indien we deze overgeërfde methoden zomaar zouden gebruik, zouden we dus een onecht vierkant kunnen maken (hoogte != breedte).
- We gaan daarom deze methoden vervangen door een nieuwe implementatie.

Polymorfisme

```
public class Polymorphism {
        public static void main (String[ ] args) {
                ...
                Rectangle rect = new Square( );
                rect.setSide(6) // compilation error
                System.out.println(rect.getHeight( ));
                ...
        }
}
```

- We maken hier een **vierkant** maar definiëren de referentievariabele van het type **Rectangle** ipv **Square**.
  - ➔ Geen probleem vierkant is een rechthoek.
  - ➔ Als we de referentievariabele declareren als een rechthoek kunnen we er echter enkel dingen mee doen die op het niveau van een Rectangle zijn gedefinieerd.

Late binding

- De **concrete koppeling** aan de uit te voeren code gebeurd dus niet tijdens de compilatie maar **tijdens de uitvoering** van het programma.

- Op het moment van de uitvoering wordt er gekeken wat de ware aard is van het object en worden diens vervangen methoden gebruikt.

  ➔ Dit is een beetje het omgekeerde van wat er gebeurd met de eigenschappen.

- Geldt **enkel voor methoden** niet voor eigenschappen

  ➔ **Make Your Stuff Private!**

Constructors van subklassen.

- Constructors worden niet overgeërfd van de superklasse.
- Iedere subklasse moet zijn eigen constructor hebben en daarbij eventueel een constructor van de superklasse aanroepen.
- Indien geen expliciete constructor gedefinieerd wordt, krijgt de klasse een standaard constructor zonder parameters.
  - ➔ Deze werkt enkel als ook de superklasse een constructor zonder parameters heeft.

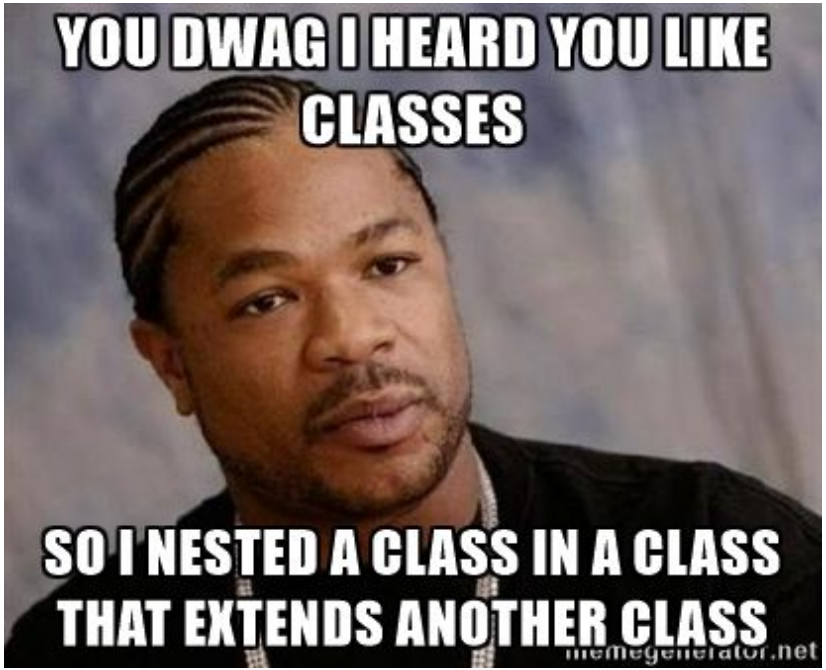
```
public Square(int s) {
        super(s, s);
}
```

De constructor van de superklasse wordt aangeroepen d.m.v super( );

Klassen eigenschappen en klassen methoden.

- ❏ De rechthoek heeft naast een aantal instance members ook klassen-members.
- ➔ Worden ook overgeërfd door de subklasse.
- ❏ Indien we bv willen bijhouden hoeveel **vierkanten** we hebben kunnen we de statische variabele count herdefiniëren.
- ➔ Ook veranderen we best de implementatie van de getCount( ) methode!

Opdracht 6
MEGABOOK P193



Final-klassen en methoden.

- ❏ Final klassen zijn klassen die geen subklassen kunnen hebben.
- ❏ Ze worden gedefinieerd met het woord **final**.
- ➔ Op die manier kan men verhinderen dat men subklassen kan maken van een bepaalde klasse.
- ➔ Indien men wel subklassen wil mogelijk maar wil verhinderen dat men de implementatie van een methode wil verhinderen kan met deze methode **final** maken.

But why you ask?

- **Beveiliging**

➔ Door klassen final te definiëren kunnen er geen subklassen van gemaakt worden.
➔ Kan namelijk een subklasse maken die heel andere dingen doet dan verwacht!

- **Ontwerp**

➔ In het ontwerp van de klassen hiërarchie kan men ertoe komen om bepaalde klassen als af te gaan definiëren.

- **Snelheid**

➔ bij final klassen en/of methoden weet de compiler vaak al bij compilatie welke implementatie gebruikt wordt.

Abstracte klassen

- Kunnen geen concrete instanties hebben.
- Dienen om een abstract kader te scheppen voor de subklassen.
- Dus we kunnen in een abstracte klasse de code onderbrengen die vervolgens wordt overgeërfd door de subklassen.

```
public abstract class AbstractClass {
        …
}
```

abstracte methoden.

Worden als volgt **gedefinieerd**

- **abstract** void abstractMethod( );

➔ Ziet iemand een verschil met een concrete (gewone) methode?

UML: Italics

extra

p 209

In a class hierarchy, if a superclass constructor requires parameters, then all subclasses must pass those parameters “up the line.” This is true whether or not a subclass needs parameters of its own