



Mobile Technologies & Smart Devices

2020~2021

overtref jezelf



Huiswerk

- ▶ Lesvoorbeelden en huiswerkopdrachten:

<https://github.com/HR-CMGT/PRG07-2020-2021>

- ▶ Wat is polymorfisme?

Huiswerk

Overerving

- ▶ Voorbeelden
 - ▶ Bike, uit de lesbrief
 - ▶ Activities in het Android framework:

```
public class PokemonLocatorActivity extends Activity {
```

Overerving: hiding

- ▶ Als bij overerving een method met dezelfde naam gebruikt wordt is de method in de parent niet meer zichtbaar. We noemen dit 'hiding' (vorige week)
- ▶ Omdat de aanpassing alleen in de uitbreiding (extends) zit wordt deze method als we gebruik maken van polymorfisme (in principe een cast van child naar parent) wel weer zichtbaar.
- ▶ Dit is meestal onwenselijk -> overriding

Overerving: overriding

- ▶ Bij overerving is het mogelijk om een method aan te passen, ook als we gebruik maken van polymorfisme: dit heet overridden.
- ▶ We geven dit in Java aan met de annotatie “@Override”, maar is in de nieuwere versies van Java niet meer verplicht
- ▶ In Android framework gebruikt bij ‘hook’-methods

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_pokemon_locator);  
}
```

Case

- ▶ Overerving is handig als je iets uit kunt breiden, maar soms is er helemaal niks om uit te breiden maar wil je wel polymorfisme gebruiken zodat een framework begrijpt waar je class voor is.
- ▶ Stel we hebben een heel eenvoudig framework dat teksten kan tonen, en ik heb twee classes: HelloWorld en RandomWord.
- ▶ Wat is nu de basis?

Case

Interfaces

- ▶ Interfaces bieden de oplossing. Je kunt dan een basis zonder functionaliteit definieren, die wel gebruikt kan worden voor polymorfisme.

Interface

- ▶ Soort functioneel ontwerp voor een class
- ▶ Beschrijft enkel welke methoden er in een class **moeten** staan, maar bevat GEEN code
- ▶ Geen protection level bij methoden
- ▶ Het is uiteraard NIET mogelijk een instantie van een Interface te maken, je moet hem eerst implementeren (**waarom?**)

Interface

- ▶ Waarschijnlijk implementeer je vaker een interface, dan dat je er één schrijft
- ▶ Hierdoor maak je je class dan geschikt voor alle functies die deze interface kennen
- ▶ Bijv.
 - ▶ Comparable<> en ArrayList<>.sort()
 - ▶ LocationListener in Android Framework

```
public class PokemonLocatorActivity extends Activity  
    implements LocationListener {
```

OO Polymorfisme op een rijtje (tot nu toe)

- ▶ Extenden/overerving is nuttig bij het uitbreiden of aanpassen van default functionaliteit (override)
- ▶ Interfaces zijn handig als classes gemeenschappelijk wil kunnen behandelen, maar geen defaultfunctionaliteit hebt

Abstracte class

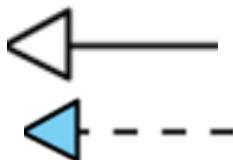
- ▶ Type interface
- ▶ Kan wel code bevatten
- ▶ Kan geen instantie van gemaakt worden
- ▶ Altijd uitbreiden om een “echte” class te maken
- ▶ **Kan dus alles wat met overerving en met interfaces kan!**
(je kan er alleen geen instantie van maken)

OO Polymorfisme op een rijtje

- ▶ Extenden/overerving is nuttig bij het uitbreiden of aanpassen van default functionaliteit (hide/override)
- ▶ Interfaces zijn handig als classes gemeenschappelijk wilt kunnen behandelen, maar geen defaultfunctionaliteit hebt
- ▶ Abstracte classes zijn handig als je een combinatie van overerving en interfaces wilt maken,
OF
Als je een basis-class wilt gebruiken voor overerving, maar niet wilt dat hij geinstantieerd kan worden

UML Klassediagram

- ▶ Overerving, IS A
- ▶ Implementatie, IS A
- ▶ Interface
- ▶ Static
- ▶ Abstracte class
- ▶ Abstract field
- ▶ Public
- ▶ Private
- ▶ Protected
- ▶ Datatype field
- ▶ Parameter
- ▶ Return type



<<interface>>
staticField of staticMethod
AbstractClassName
abstractField
+
-

field:datatype
method(param:datatype)
method():returntype

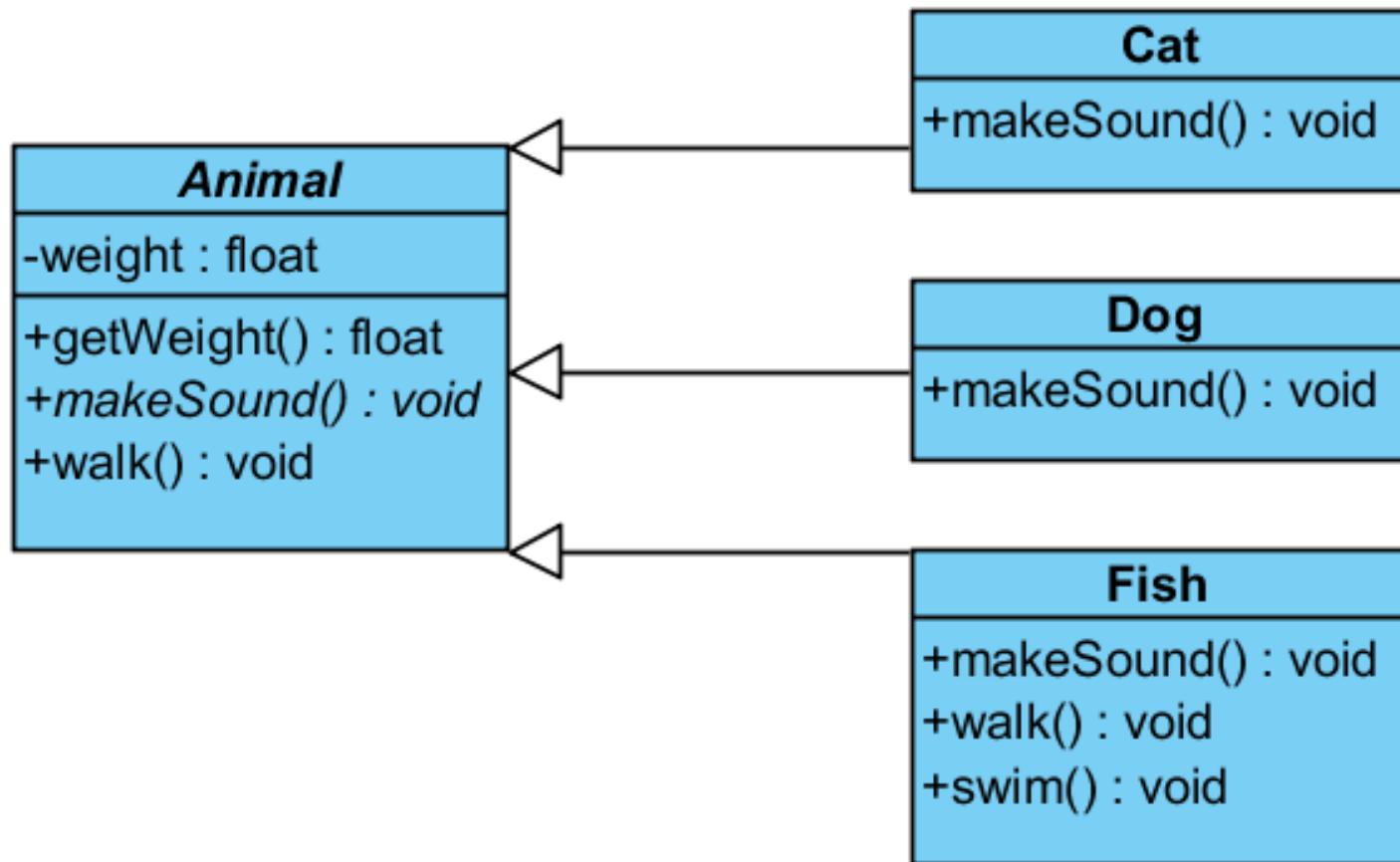
Voorbeeld met Animals

- ▶ Elk dier heeft een gewicht, en kan dit vertellen
- ▶ Elk dier maakt geluid, maar natuurlijk geen default geluid
- ▶ Bijna elk dier kan lopen alleen bij vissen kan dat natuurlijk niet
- ▶ Vissen kunnen zwemmen, maar verder niemand

>> class diagram

Class Diagram

Class Diagram



Werkcollege

- ▶ Huiswerk bespreken
- ▶ Interface voorbeeld
- ▶ Animal voorbeeld uitwerken

Afbeeldingsbronnen

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