PROGRAMMEREN 2

TOPIC 9: OVERERVING

Download PDF-versie

UCLL HOGESCHOOL

OVERZICHT

- Probleemstelling
- Overerving in Python
- Method overriding en Polymorfie
- `object`
- Meervoudige overerving, abstracte klassen en interfaces

WAAROM OVERERVING?

PROBLEEMSTELLING

Bekijk volgende twee klassen. Wat valt je op?

Python

```python
class Dog:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def celebrate_birthday(self):
    self.age += 1

  def is_adult(self):
    return self.age >= 1
```

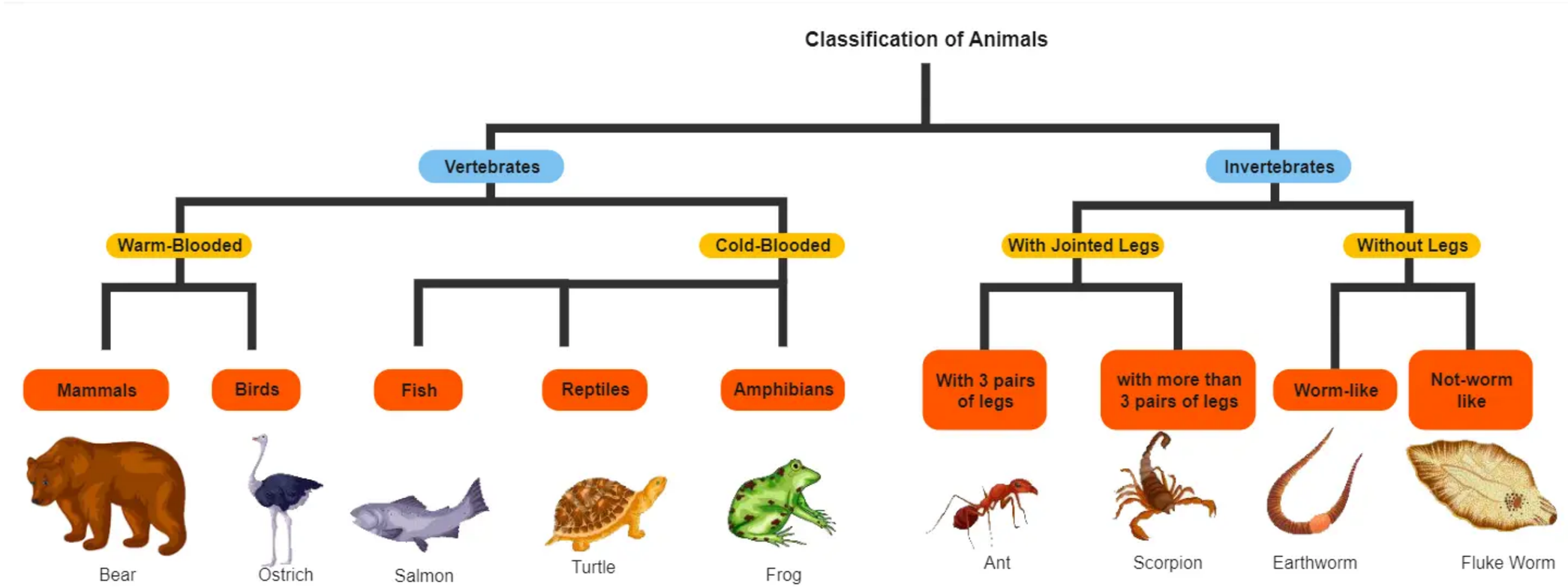
Python

```python
class Cat:
def __init__(self, name, age):
  self.name = name
  self.age = age

def celebrate_birthday(self):
  self.age += 1

def is_adult(self):
  return self.age >= 2
```

PROBLEEMSTELLING

Het gebeurt heel vaak dat klassen op elkaar lijken. In zo'n gevallen willen we een manier hebben om **code te hergebruiken** tussen de klassen. Meer algemeen willen we een **hiërarchie van klassen** kunnen maken.

Zeker als we bepaalde concepten willen modelleren uit de werkelijkheid, komt dit goed van pas. Denk bevoorbeeld aan stambomen uit de biologie.



VOORBEELD

Stel je voor dat je een persoon op de campus Hertogstraat wilt voorstellen door middel van klassen:

- Welke splitsingen maak je op het hoogste niveau?

VOORBEELD

Stel je voor dat je een persoon op de campus Hertogstraat wilt voorstellen door middel van klassen:

- Welke splitsingen maak je op het hoogste niveau?
- Hoe zou je studenten verder opdelen?

VOORBEELD

Stel je voor dat je een persoon op de campus Hertogstraat wilt voorstellen door middel van klassen:

- Welke splitsingen maak je op het hoogste niveau?
- Hoe zou je studenten verder opdelen?
- Hoe zou je personeel verder opdelen?

VOORBEELD

Stel je voor dat je een persoon op de campus Hertogstraat wilt voorstellen door middel van klassen:

- Welke splitsingen maak je op het hoogste niveau?
- Hoe zou je studenten verder opdelen?
- Hoe zou je personeel verder opdelen?
- Kan je behalve studenten en personeel nog andere (sub)groepen onderscheiden?

OVERERVING IN PYTHON

TERMINOLOGIE

Doel: we herschrijven het voorbeeld van `Dog` en `Cat` door gebruik te maken van **overerving** (Engels: *inheritance*).

`Dog` en `Cat` zullen een **superklasse** `Pet` delen. Deze klasse bevat de code die op beide **subklassen** van toepassing is.

VOORBEELD MET OVERERVING

```python
class Pet:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def celebrate_birthday(self):
    self.age += 1
```

```python
class Dog(Pet):
  def is_adult(self):
    return self.age >= 1

  def fetch_ball(self):
    pass

class Cat(Pet):
  def is_adult(self):
    return self.age >= 2
```

Algemeen kan een klasse `B` overerven van een klasse `A` door het volgende te schrijven:

```python
class B(A):
```

Elk object van type `B` zal automatisch de attributen en methodes van `A` overnemen.

KLASSEN OPROEPEN

Voorbeeld:

```python
tom = Cat("Tom", 1)
neighbour_dog = Dog("Basil", 1)

assert tom.age == 1
assert neighbour_dog.age == 1

assert tom.is_adult() == False
assert neighbour_dog.is_adult() == True
```

METHOD OVERRIDING EN POLYMORFIE

METHOD OVERRIDING EN POLYMORFIE

In het codevoorbeeld hebben we twee klassen `Dog` en `Cat` die elk de methode `is_adult()` definiëren.

We kunnen deze methode ook in de superklasse `Pet` definiëren:

```python
class Pet:
  def is_adult(self):
    self.age > 18
```

Er bestaan nu 3 versies van `is_adult()`. De subklassen `Dog` en `Cat` overschrijven de versie van de superklasse `Pet`. Men noemt dit meer algemeen het **overschrijven** van methodes (Engels: *method overriding*).

Wanneer twee of meerdere klassen een verschillende versie hebben van dezelfde methode, noemt men dit **polymorfie** (Engels: *Polymorphism*).

KEUZE VAN VERSIE?

Welke versie wordt nu gebruikt voor een object? Dit hangt af van het type:

1. Als de methode bestaat in de klasse die overeenkomt met het type, roep dan die versie op
2. Anders als er een superklasse is, herhaal (1) met de superklasse
3. Anders: stop en gooi een error

Dit proces werkt recursief tot er geen superklasse meer is (je hebt de top van de boomstructuur bereikt).

Toegepast op het voorbeeld:

- `tom` gebruikt `is_adult()` van `Cat`
- `neighbour_dog` gebruikt `is_adult()` van `Dog`
- stel dat `is_adult()` niet bestond in `Dog`, dan had `neighbour_dog` de versie van `Pet` gebruikt

VERWIJZEN NAAR SUPERKLASSE

Stel je voor dat we aan `Cat` een nieuw attribuut `is_tired` willen toevoegen. We zouden de volledige `__init__()` van `Cat` kunnen kopiëren en uitbreiden:

Python

```
class Cat(Pet):
  def __init__(self, name, age, is_tired):
    self.name = name
    self.age = age
    self.is_tired = is_tired
```

Dit wordt echter snel vervelend. Het gaat ook in tegen het doel van OGP om code te hergebruiken.

`super()`

Programmeertalen hebben vaak een gereserveerd woord om naar de superklasse te verwijzen. In Python is dit `super()`. Met `super()` kan je gebruik maken van attributen en methodes van de superklasse in je code om kopiëren tot een minimum te beperken

Python

```
class Cat(Pet):
  def __init__(self, name, age, is_tired):
    super().__init__(name, age)
    self.is_tired = is_tired
```

`super()` heeft enkel betekenis binnen een `class`. Buiten een klasse zal het gebruik ervan een error geven.

OPDRACHT

Schrijf in Python code om personen op de campus Hertogstraat te modelleren:

- Definieer de klassen in een hiërarchie met overerving
- Voeg attributen toe op de juiste levels
- Definieer een methode `print_card()` die de informatie van een persoon afprint, net alsof ze op een studentenkaart/personeelskaart/bezoekerskaart zou staan
  - Definieer ze sowieso op het hoogste niveau
  - Overschrijf de methode binnen elke subklasse indien nodig
  - Gebruik `super()` om code te hergebruiken

DE `object` CLASS

DE `object` CLASS

Klassen gebruiken overerving om hun superklasse te overschrijven of uit te breiden. Ook ingebouwde methodes zoals `__init__()` werken op deze manier. Maar van waar komen die ingebouwde methodes dan?

Elke klasse in Python erft - direct of indirect - van de speciale `object` klasse. Deze klasse bevat de ingebouwde methodes die elke klasse moet hebben. De implementaties van de methodes zijn simpel. Het is aan subklasses om ze te overschijven of uit te breiden.

Merk op dat `object` met een kleine 'o' wordt geschreven. Dit is een uitzondering op de algemene regel in Python.

DE `object` CLASS

`object` is de 'bron' van alle klassen. Er is geen superklasse boven. Je kan expliciet erven van `object`, maar dit is niet nodig. De volgende twee klassen zijn equivalent:

```python
class Pet(object):
```

Python

```python
class Pet:
```

Python

Je kan verwijzen naar methodes van `object` vanuit `Pet` via `super()`. Je gaat dit echter niet vaak nodig hebben.

MEERVOUDIGE OVERERVING, ABSTRACTE KLASSEN EN INTERFACES

MEERVOUDIGE OVERERVING

Tot nu toe hebben we enkel voorbeelden gezien van enkelvoudige overerving (Engels: *Single inheritance*). Wat als een subklasse concepten van meerdere superklassen wil overnemen?

Voorbeeld: stel dat we een vogelbekdier willen modelleren in een spel. Vogelbekdieren delen kenmerken van zoogdieren en vogels. Bovendien produceert het dier een gif, net zoals bepaalde reptielen en sommige andere diersoorten. De klasse `Platypus` zou code over moeten nemen van 3 klassen: `Mammal`, `Bird` en `PoisonousAnimal`.

STARTCODE

Merk op dat `get_average_body_temperature()` in zowel `Mammal` als `Bird` is gedefinieerd.

Python

```
class Mammal:
  def can_hibernate(self):
    pass
  def get_average_body_temperature(self):
    return 37

class Bird:
  def get_egg_hatching_time(self):
    pass
  def get_average_body_temperature(self):
    return 42

class PoisonousAnimal:
  def get_antidote(self):
    pass
```

OPLOSSING 1: MEERVOUDIGE OVERERVING

Meervoudige overerving (Engels: *multiple inheritance*) houdt in dat `Platypus` de 3 klassen een voor een overerft. In Python ziet dit er als volgt uit:

Python

```
class Platypus(Mammal, Bird, PoisonousAnimal):
  pass

perry = Platypus()
print(perry.get_average_temperature())
```

Wat zal de code hierboven afprinten?

37

42

OPLOSSING 1: MEERVOUDIGE OVERERVING

Meervoudige overerving (Engels: *multiple inheritance*) houdt in dat `Platypus` de 3 klassen een voor een overerft. In Python ziet dit er als volgt uit:

Python

```
class Platypus(Mammal, Bird, PoisonousAnimal):
  pass

perry = Platypus()
print(perry.get_average_temperature())
```

Wat zal de code hierboven afprinten?

37

42

Het resultaat is 37 omdat `Mammal` voor `Bird` staat in de klassedeclaratie van `Platypus`. Draai je de volgorde om, dan is het resultaat 42.

Meervoudige overerving klinkt simpel, maar bevat verschillende valkuilen en wordt snel complex om te begrijpen. Slechts een klein aantal talen ondersteunt meervoudige overerving om die redenen. Python en C++ zijn de bekendste.

In dit vak is het gebruik van meervoudige overerving verboden, tenzij anders aangegeven.

Meervoudige overerving wordt algemeen afgeraden als programmeertechniek. Het wordt zelden gegeven in een vak op een middelbare school.

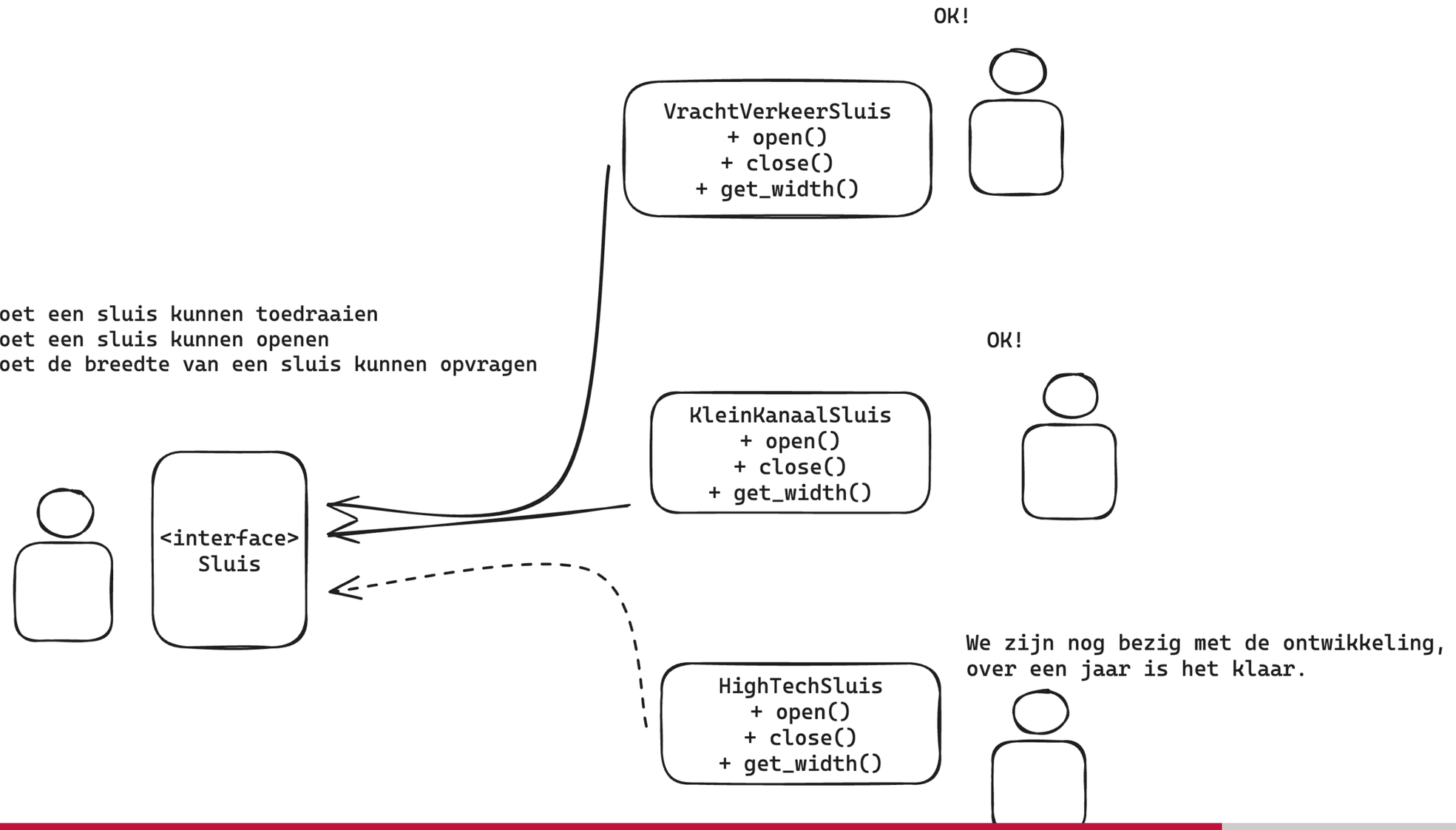
OPLOSSING 2: INTERFACES

Een interface is een 'lichte' versie van een klasse. Het bevat enkel methodes, geen attributen. De methodes hebben vaak geen implementatie. Een klasse kan meerdere interfaces implementeren.

Interfaces worden vaak gebruikt als **brug** tussen verschillende delen van de code van een project. Een interface definieert een groep van afspraken (de methodes), maar laat de details over aan de klassen die de interface implementeren.

INTERFACES: VOORBEELD

Stel je voor dat je een systeem moet ontwerpen om sluizen in kanalen te besturen vanop afstand. Er zijn verschillende soorten sluizen, en in de toekomst kunnen er nieuwe soorten bijkomen. De interface `Sluis` legt vast welke methodes elke soort sluis moet ondersteunen, maar laat de details over aan de sluizen zelf.



INTERFACES IN PYTHON

In veel talen (Java, TypeScript, C#, Go, ...) is `interface` een gereserveerd woord zoals `class`. Je kan het gebruiken om een interface te definiëren.

Python ondersteunt interfaces niet expliciet, er is geen `interface` woord. Je kan wel een klasse maken die lijkt op een interface:

```python
class Sluis:
  def open(self):
    raise NotImplementedError

  def close(self):
    raise NotImplementedError

  def get_width(self):
    raise NotImplementedError
```

- De klasse heeft geen attributen en geen `__init__()`
- `NotImplementedError` is een speciale error in Python die aangeeft dat de methode overschreven moet worden door klassen die de interface implementeren. Als je dit toch vergeet, krijg je een error bij uitvoering

INTERFACES TOEGEPAST

We creëeren een aantal interfaces die `Platypus` nadien kan implementeren. De interfaces zijn herbruikbaar voor andere dieren.

Python

```
class Poisonous:
  def get_antidote(self):
    raise NotImplementedError

class EggLayer:
  def get_egg_hatching_time(self):
    raise NotImplementedError

class HotBlooded:
  def get_average_body_temperature(self):
    raise NotImplementedError
```

Python

```
class Platypus(Poisonous, EggLayer, HotBlooded):
  def get_average_body_temperature(self):
    return 32


  # Implementatie van andere methodes...
```

Het handboek suggereert om `NotImplemented` terug te geven i.p.v. `NotImplementedError` te gooien. **Doe dit niet**. Er zijn subtiele verschillen tussen beiden. Wil je meer details, bekijk dan dit antwoord op StackOverflow.

OPLOSSING 3: ABSTRACTE KLASSE EN INTERFACES

Een abstracte klasse is een klasse waar je geen instantie van kan maken. De constructor van de klasse blokkeert dit. In alle andere opzichten gedraagt het zich als een klasse.

Abstracte klassen mogen methodes bevatten zonder implementatie, net zoals interfaces. Het is dan aan de subklassen om ze te implementeren. Ze mogen ook methodes bevatten mét implementatie.

Een klasse kan maar maximaal van één abstracte klasse overerven (net zoals bij niet-abstracte klassen). Daarom zal een abstracte klasse vaak gecombineerd worden met interfaces.

De meeste talen ondersteunen abstracte klassen met het gereserveerde woord `abstract`. **Python heeft dit echter niet**. Er is geen(*) manier om abstracte klassen af te dwingen in Python.

(*) Python heeft in recent versies abstract base classestoegevoegd, maar dit zou ons te ver leiden.

ABSTRACTE KLASSE TOEGEPAST

We maken van `Mammal` een abstracte klasse (zonder dit af te dwingen). We laten `Platypus` overerven van `Mammal`, aangezien een vogelbekdier daar het meeste mee overeenkomt. De andere methodes worden geïmplementeerd via interfaces.

Python

```
class Mammal:
  def __init__(self):
    # Kenmerken van zoogdieren...

  def get_average_body_temperature(self):
    raise NotImplementedError

class Platypus(Mammal, EggLaying, Poisonous):
  def __init__(self):
    super().__init__()
    # Kenmerken van vogelbekdieren...

  def get_average_body_temperature(self):
    return 32
```

WELKE TECHNIEK IS HET BESTE?

Dit hangt af van meerdere factoren:

- Meervoudige overerving heeft nadelen en wordt afgeraden, maar sommige talen ondersteunen de andere technieken niet zo goed
- Enkel interfaces is heel erg populair in modernere talen zoals Rust. Het schaalt over het algemeen beter in grotere projecten. Soms leidt het echter toch tot wat kopieer- en plakwerk
- Abstracte klasse + interfaces wordt het meest gebruikt in oudere talen zoals Java en C#

Zorg dat je goed weet welke technieken een taal (niet) ondersteunt, voordat je lesgeeft over OGP en overerving in die taal. Sommige talen laten veel toe, andere talen dwingen één bepaalde techniek af.

OPDRACHTEN

OPDRACHTEN

Maak opdracht 22.1 en opdracht 22.2 in het handboek. Deze opdrachten bouwen voort op de code van het vorige topic (uitbreiding OGP). Het helpt als je die oefeningen reeds hebt gemaakt.

Maak opdracht 22.3 in het handboek. Deze opdracht heeft te maken met speltheorie, een domein dat ook aan bod is gekomen tijdens de OD-dag over AI - meer specifiek het minimax-algoritme. Je krijgt startcode aangeboden en moet deze verder uitbreiden met een aantal spelstrategieën.