Technische Universität Braunschweig

Institut für rechnergestützte Modellierung im Bauingenieurwesen

Algorithms & Programming

Lecture 4

Dr. Jan Linxweiler

Pull out your calender

Assignment 1

Available on StudIP:	30.11.2022
Submission Deadline:	16.12.2022

Assignment 2

Available on StudIP:	11.01.2023
Submission Deadline:	27.01.2023

Oral assessement: 06.02. - 10.02.2023

Succesfull completion of both assignments & the oral assessment is necessary to pass the module Algorithms & Progamming and to get the 8 ECTS!

Question?

What do you think should be the most important aspect of software development?

Software should …

A. ... do what it is supposed to do.
B. ... be easy to extend with new functionality.
C. ... be cost-effective.
D. ... be available on as many different platforms as possible.

“ The art of programming is the art of organizing complexity. ”

Edsger W. Dijkstra

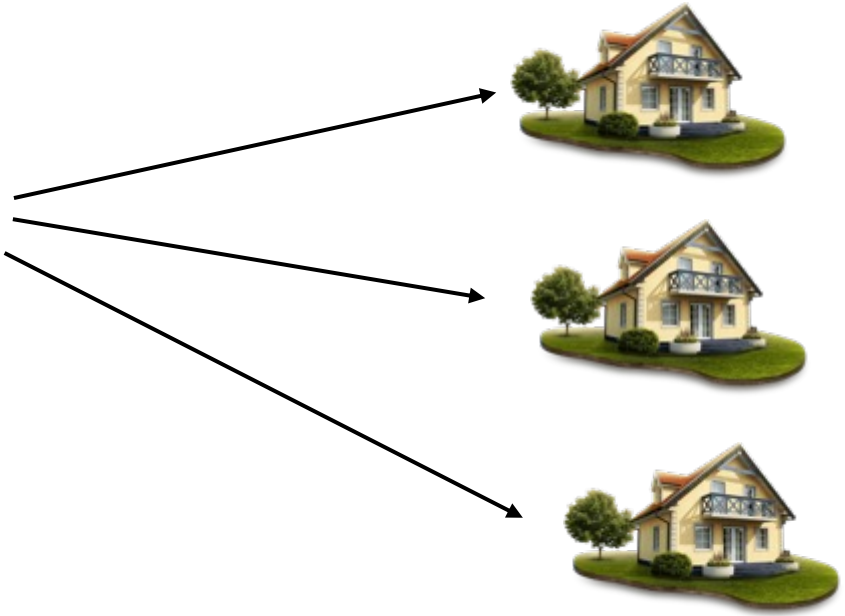
Basics of object oriented programming

- **Standard method** of software development since beginning of 90s
- OOP is based on the **classification** of data related to their properties and behavior
- **mapping of the real world to software**
  → objects with properties/ attributes and behavior / operations
- the concept of **classes** groups together objects with similar properties: a class describes the structure of an object - the attributes (properties) and methods (behavior) – a building plan for an object
- objects are reduced to properties in the context of the described model context -> **we only consider aspects relevant for a given use case**.

Class & object relationship

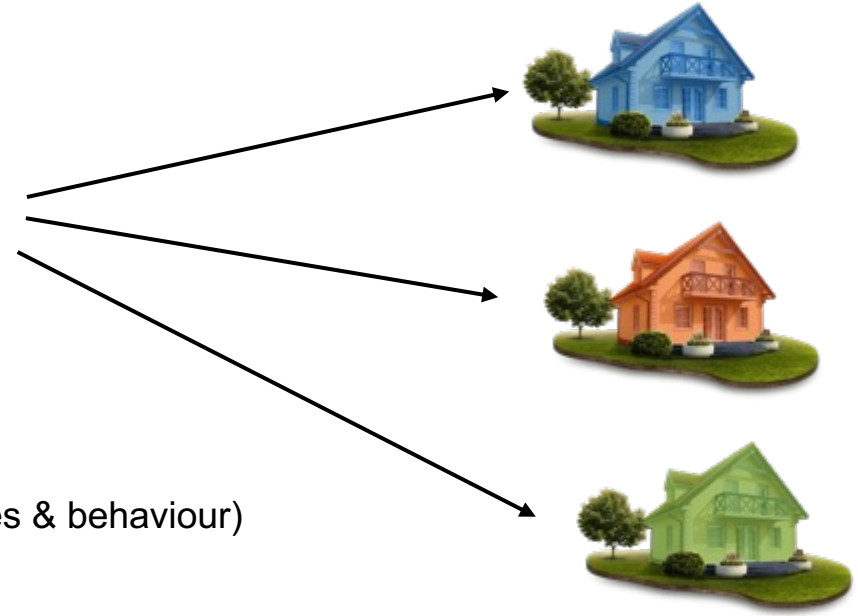
Class = Building Plan

Multiple objects (instances) of the same class (type)



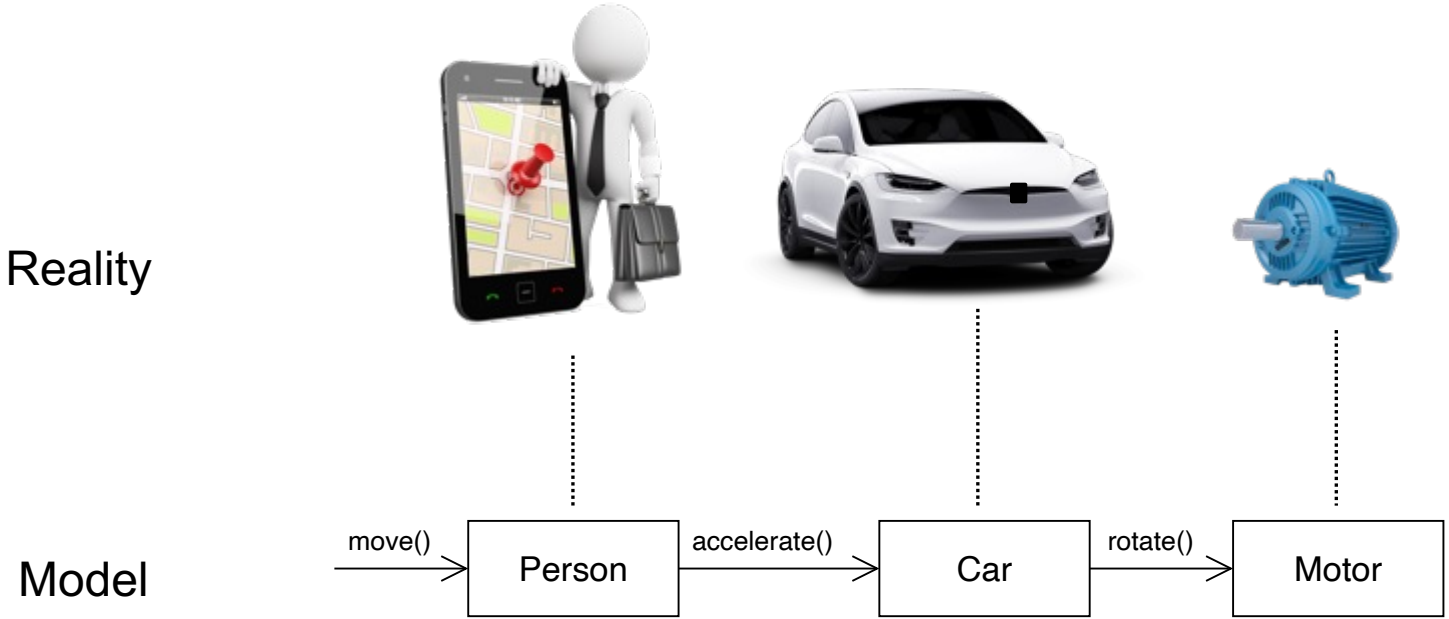
Class & object relationship

The properties (e.g. color) can have different values (e.g. blue, red, etc.)



The class specifies the structure (properties & behaviour) of the objects / instances

„Message“ oriented programming



Context sensitive properties / attributes

bakery

Bread

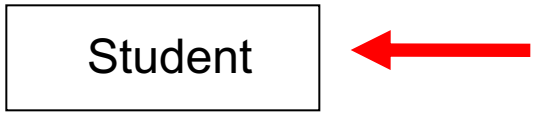
flour
yeast
salt
…

super market

Bread

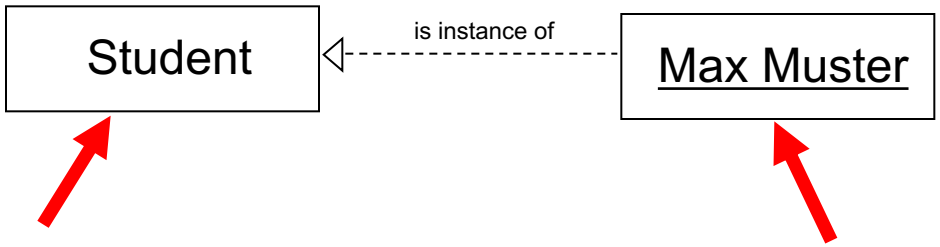
price
weight
durability date
…

UML diagram for design



„simplest“ UML class diagram

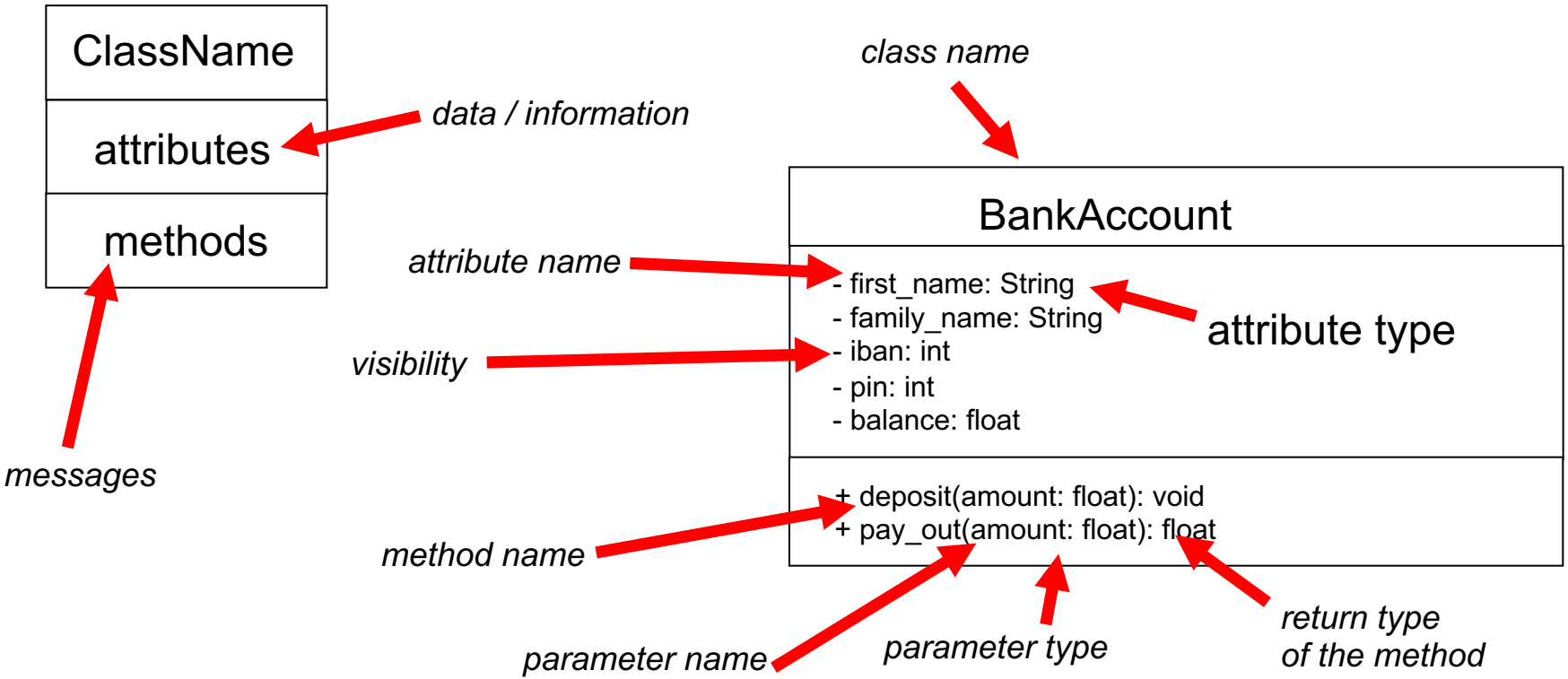
class name: noun, singular



class
(building plan)

concrete instance / object

UML class diagram - components

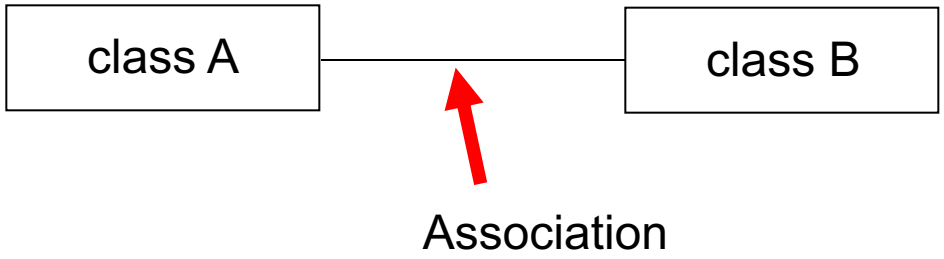


class example

BankAccount
- first_name: String - family_name: String - iban: int - pin: int - balance: float
+ deposit(amount: float): void + pay_out(amount: float): float

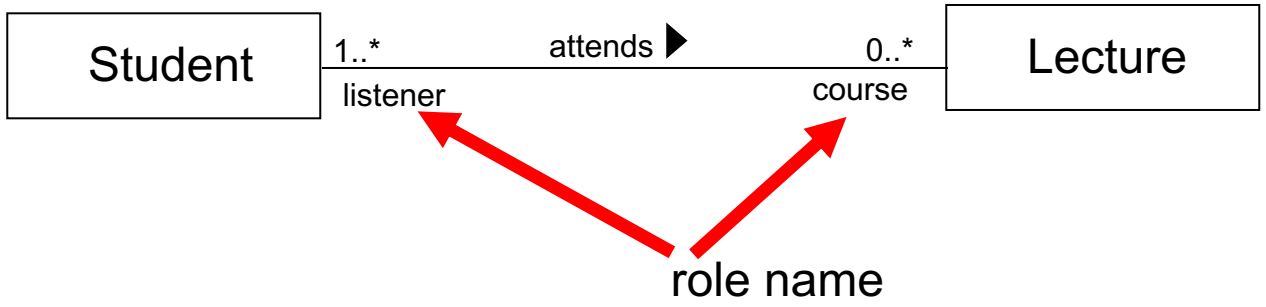
```python
class BankAccount:
    def __init__(
        self,
        first_name: str,
        last_name: str,
        iban: int,
        pin: int,
    ):
        self._first_name = first_name
        self._last_name = last_name
        self._iban = iban
        self._pin = pin
        self._balance = 0.0

    def deposit(self, amount: float) -> None:
        self._balance = self._balance + amount
        print(f"Deposited {amount}€")

    def pay_out(self, amount: float) -> float:
        self._balance = self._balance - amount
        print(f"Paid out {amount}€")
        return amount

    def __str__(self) -> str:
        return "{} {} IBAN:{} PIN:{} Balance:{}".format(
            self._first_name,
            self._last_name,
            self._iban,
            self._pin,
            self._balance,
        )


if __name__ == "__main__":
    account = BankAccount("John", "Doe", 123456789, 1234)
    print(account)
    account.deposit(100)
```

UML class diagram - object relations



An Association represents a family of links which describe relations between different objects.

UML - object relations – detailed view



Association

- relation of the same grade

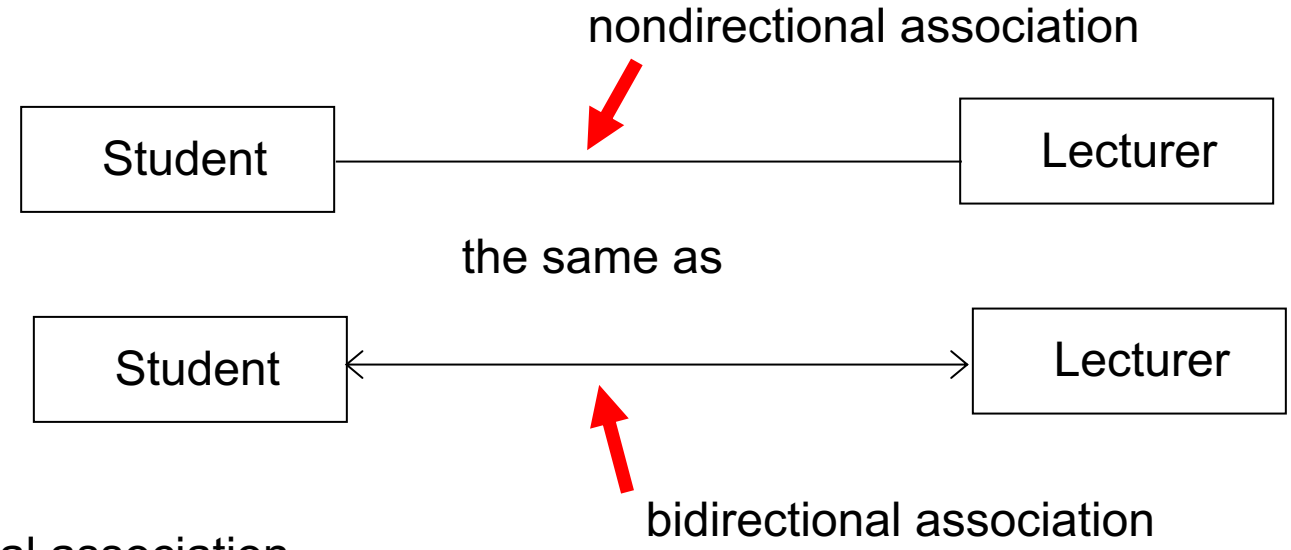
Multiplicity / Cardinality

* = any number of

1 = 1

1..* = at least 1 up to any number of

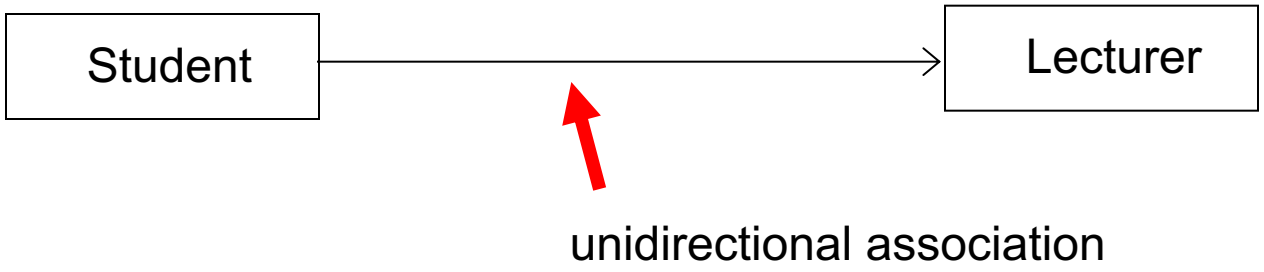
UML - object relations – navigation



bidirectional association

- student „knows“ his lecturer
- lecturer „knows“ his / her students

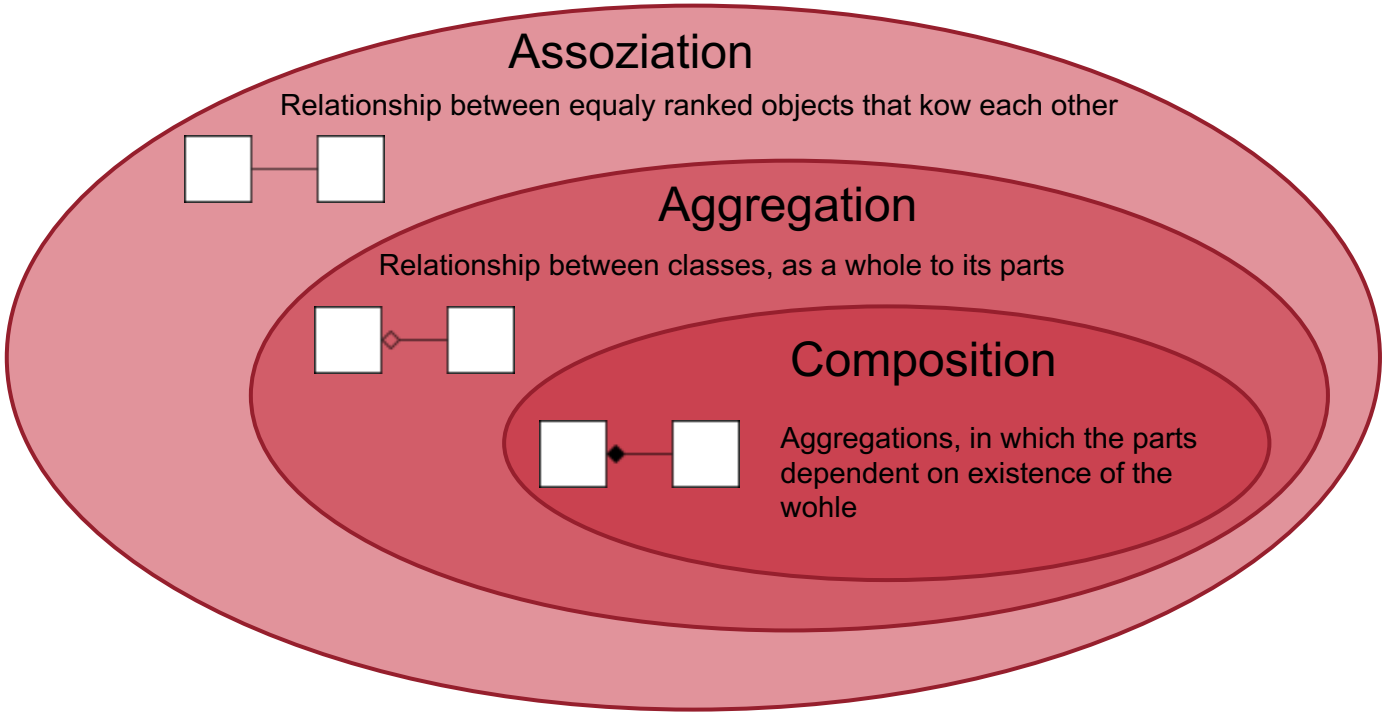
UML - object relations – navigation



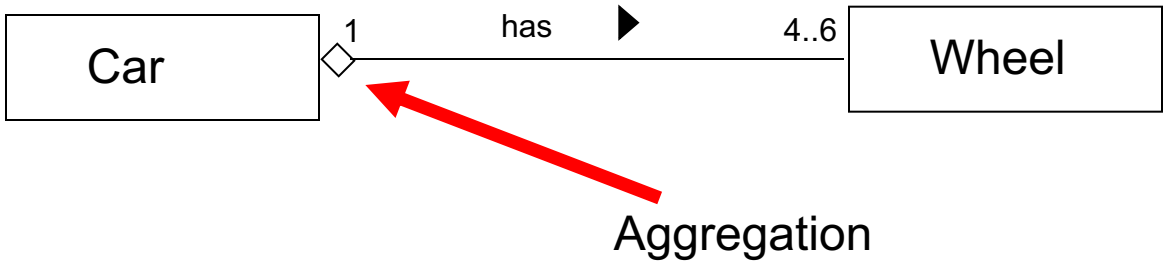
unidirectional association

- student „knows“ her/his lecture
- lecturer **does not** know her/his students ☹

UML - Object relations



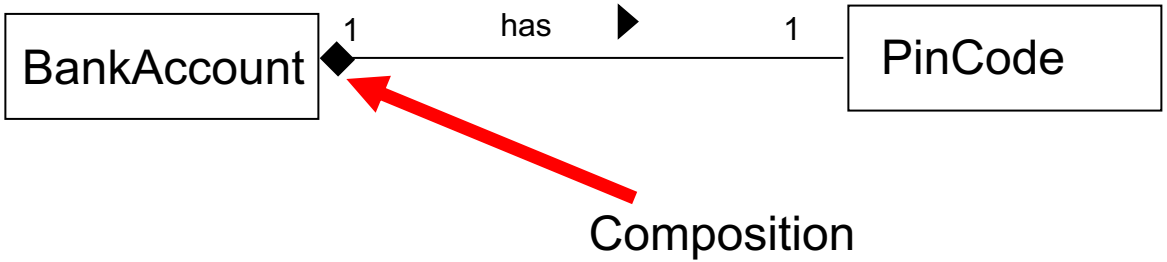
UML - object relations



Aggregation

- is a variant of the "has a" or association relationship
- aggregation is more specific than association
- an association that represents a part-whole or part-of relationship

UML - object relations



Composition

- is a stronger variant of the “has a" or association relationship
- is more specific than aggregation.
- the parts of the composed class exist only if the composed class exist

example: by deleting Invoice, all LineItems will be deleted

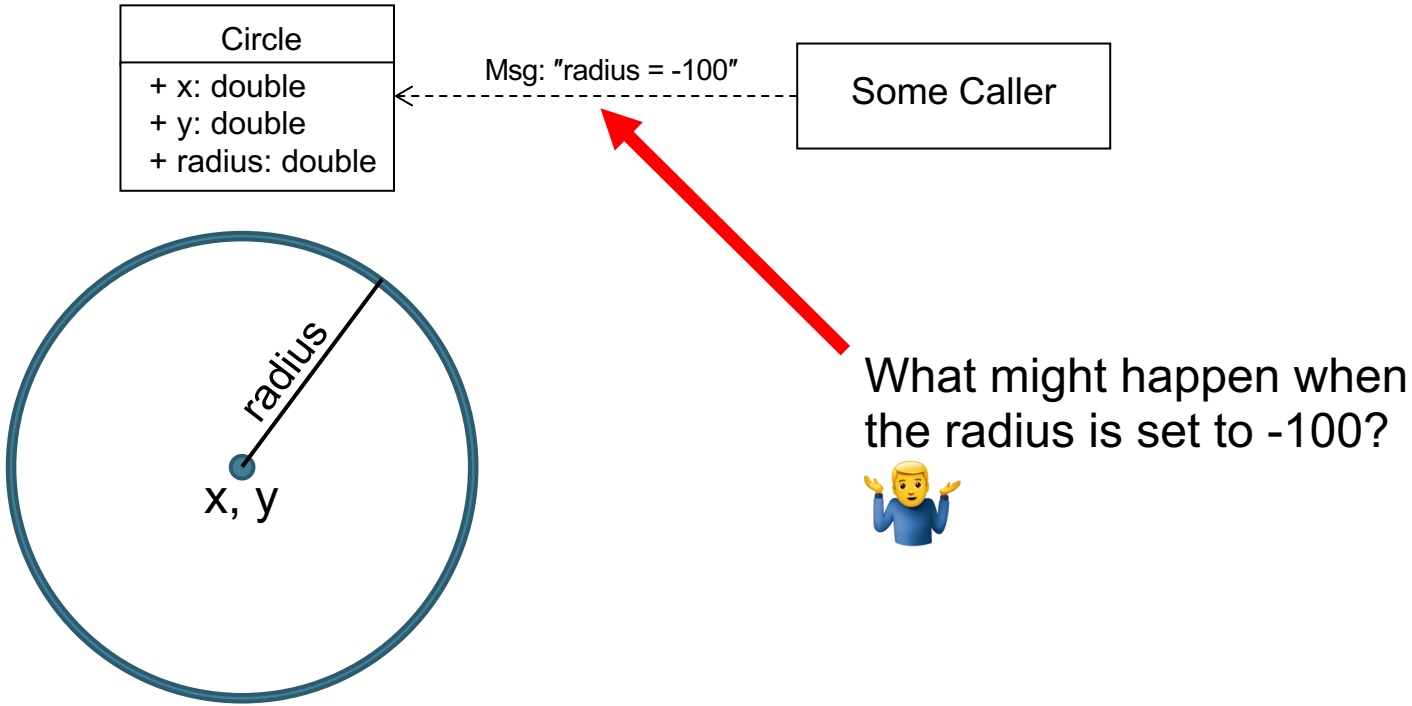
Object Oriented Programming - Principles

In the 1980s Alan Kay introduced the term „Object Oriented Programming“.

An OO Language should at least support the following three Principles:

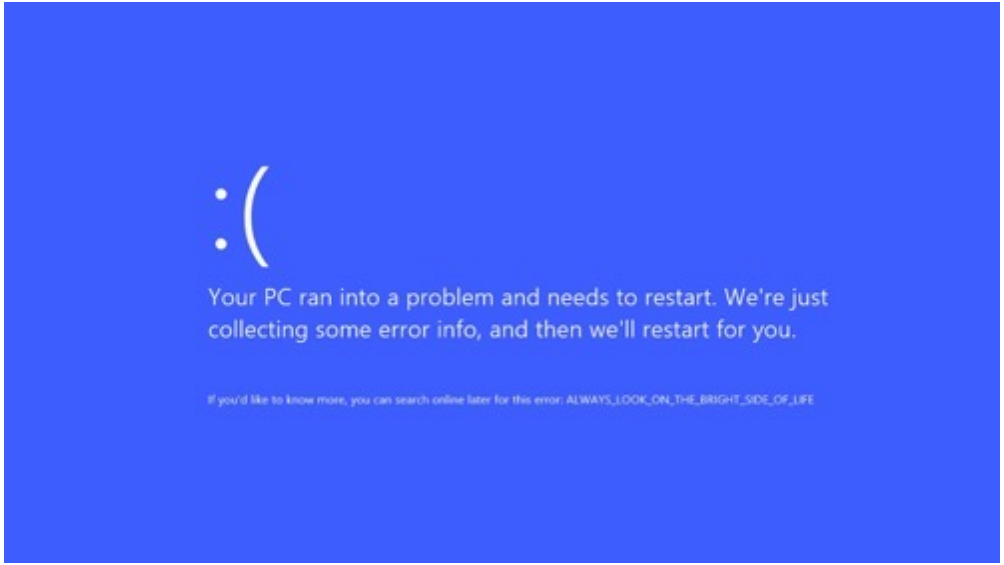
- **Encapsulation**
- **Inheritance**
- **Polymorphism**

Object Oriented Programming - Encapsulation



Who‘s to blame?

a) The circle
b) The caller
c) Both
d) None of them



Keep in mind!

An object is responsible to guarantee it's state is valid!

Object Oriented Programming - Encapsulation

In a procedural programming approach data and functions are separated resulting in potential access to arbitrary data from any point of the program.

Encapsulation means that the internal representation of an object is generally hidden from outside the object's definition.

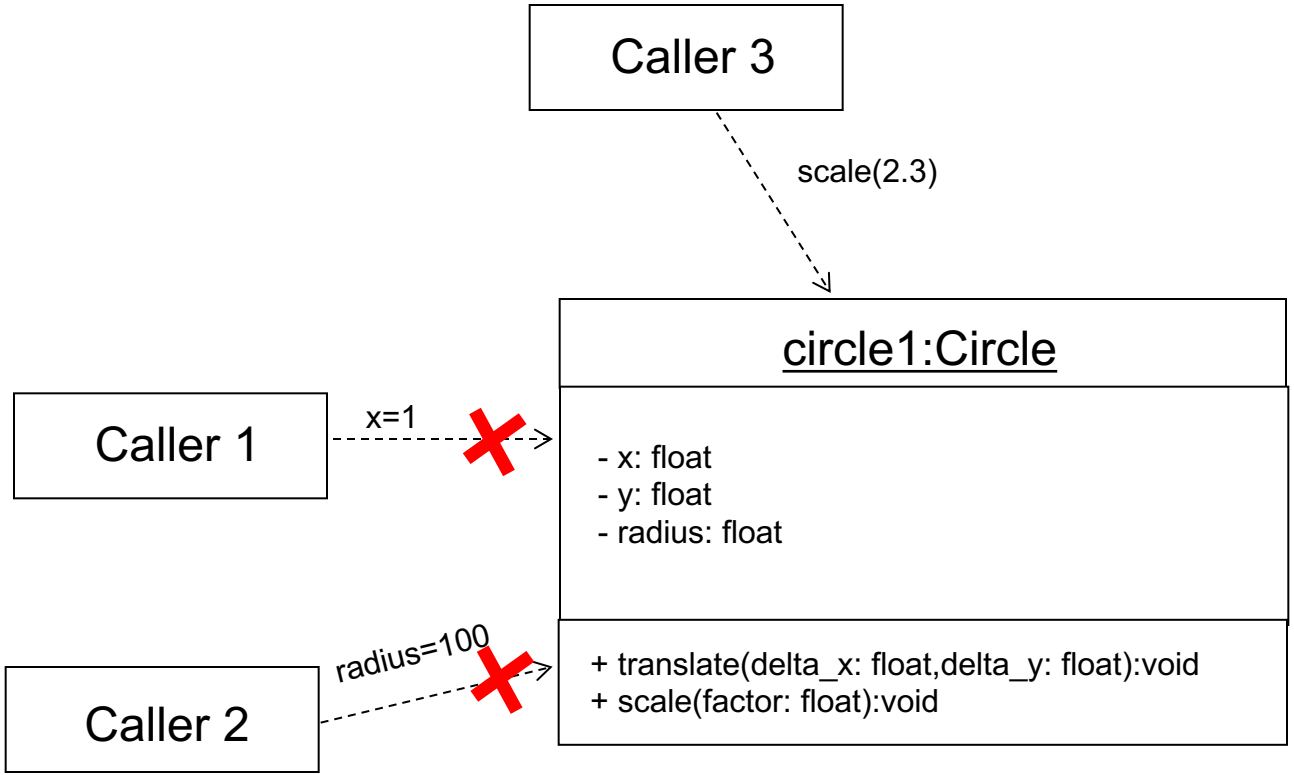
Typically, only the object's own methods can directly read or manipulate its attributes.

Data and functions are combined in classes.

Direct acces to data is possible only inside this class.

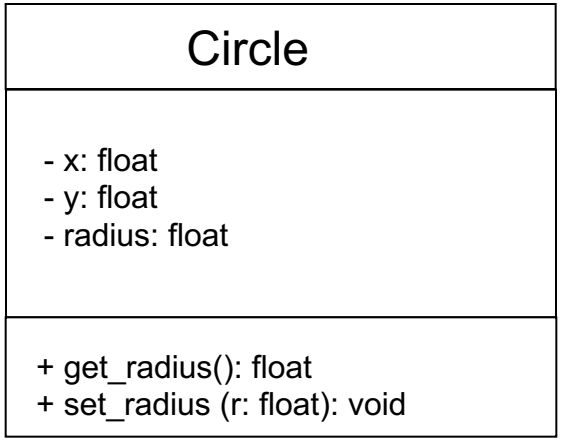
Access and modification to data only via methods of the object.

Object Oriented Programming - Encapsulation

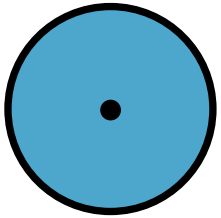


Object Oriented Programming - Encapsulation

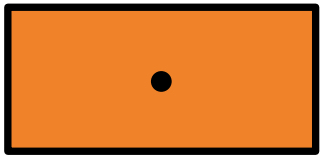
In order to read and write attributes, **getter and setter methods** are used.
Using (internal) methods to access data allows class-specific control and filtering.
Example: Circle – The radius must be larger than 0.



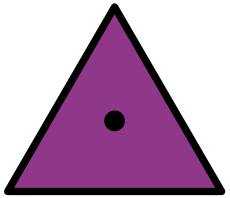
Object Oriented Programming - Inheritance



circle

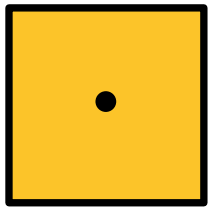


rectangle

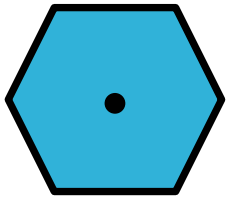


triangle

miniCAD App

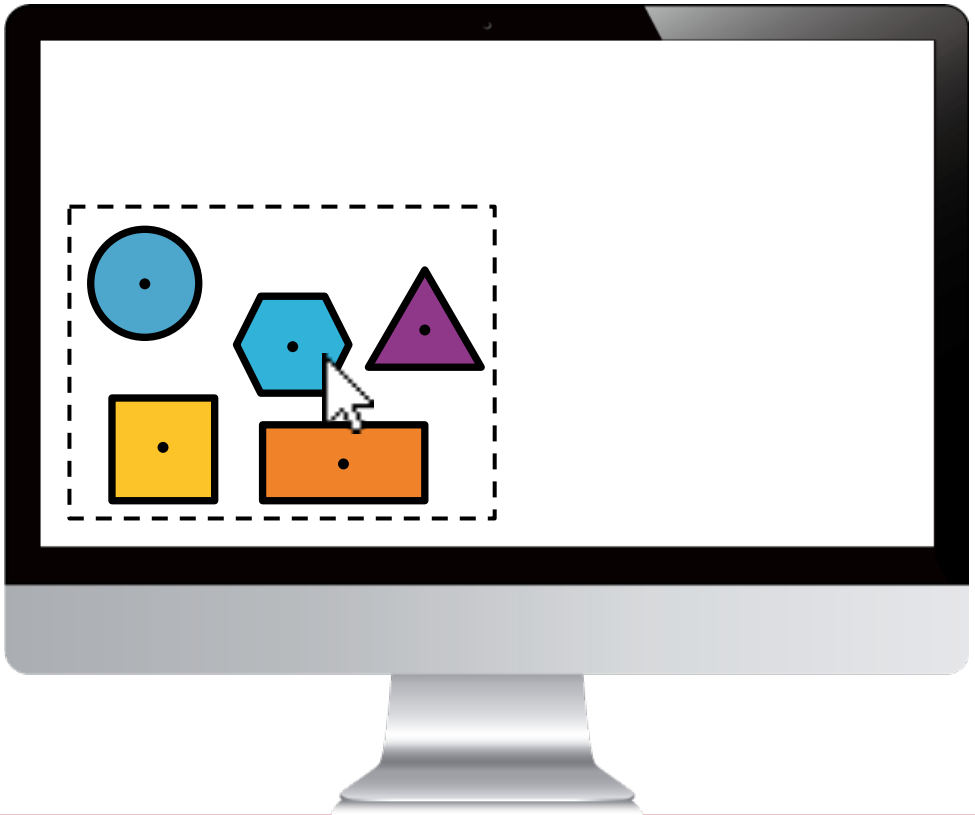


square

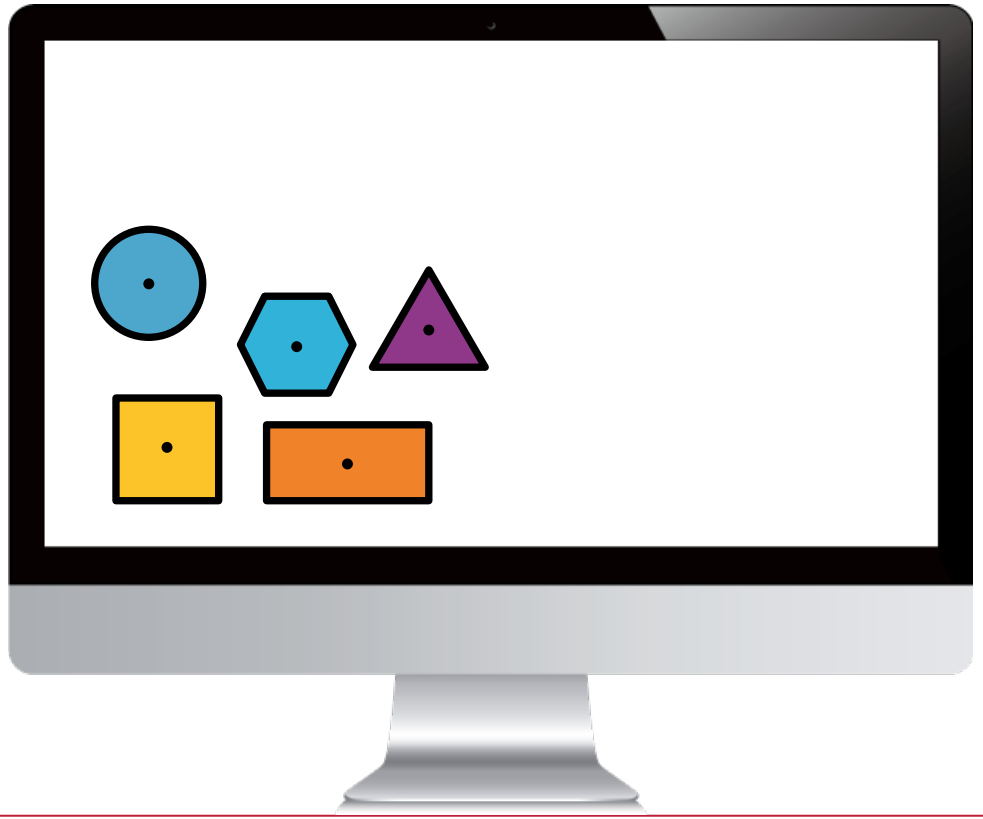


hexagon

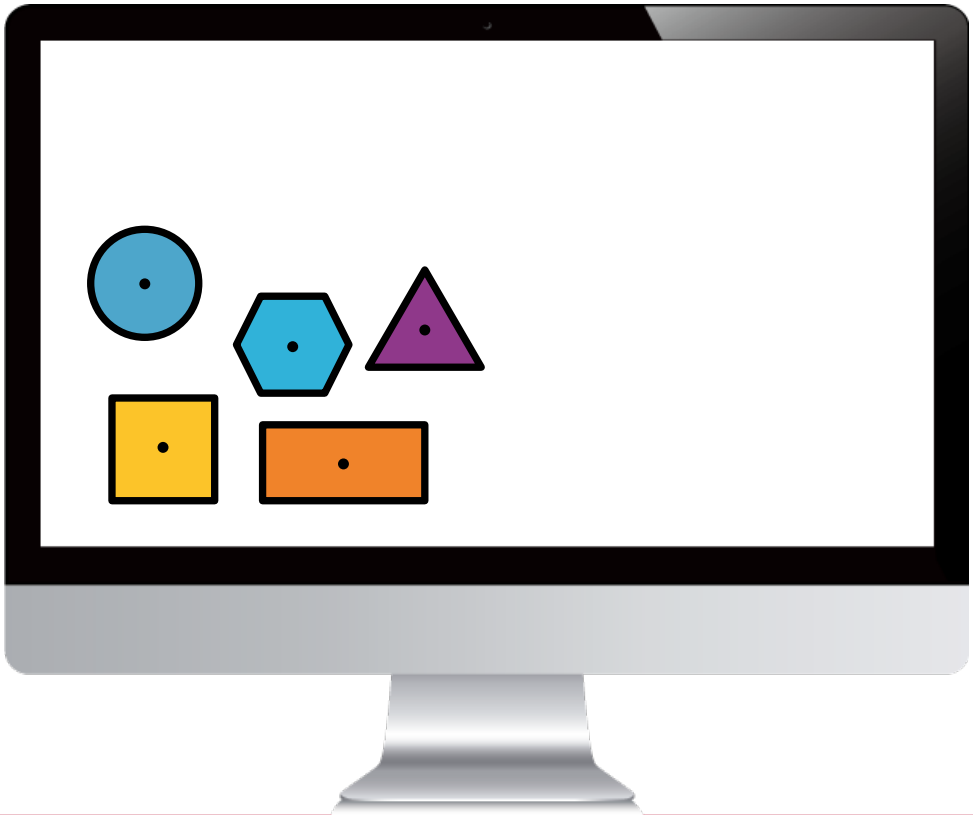
Move objects on the screen



What actually happens …



After you fixed it …



Object Oriented Programming - Inheritance

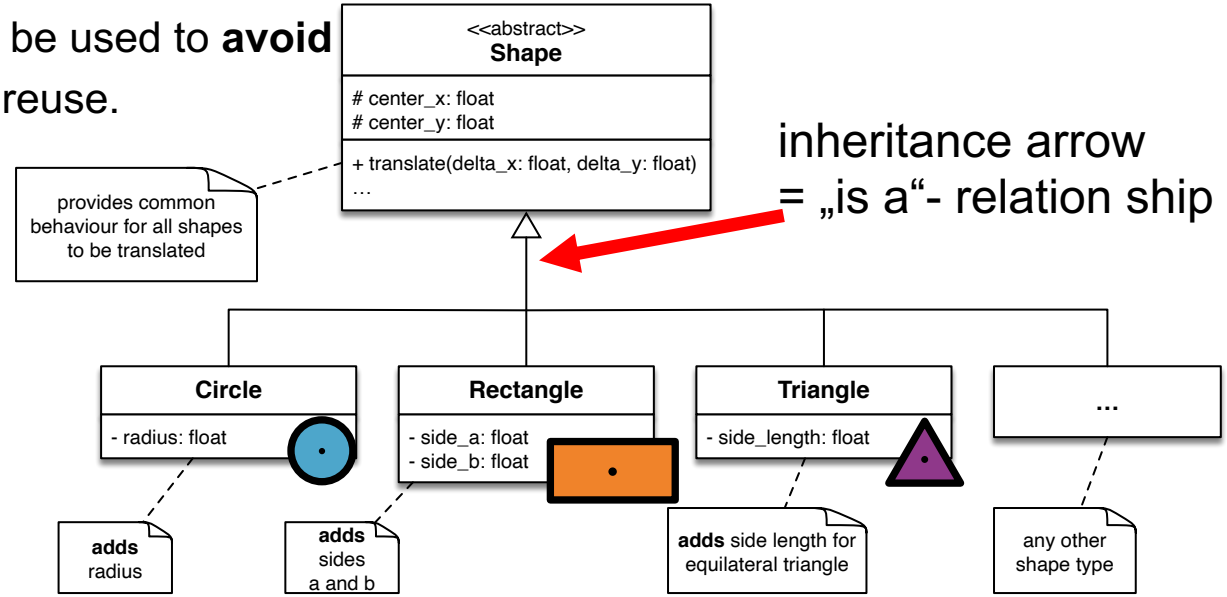
Classes can be specializations of other classes. That means classes can be in hierarchical order by inheriting properties and behavior of higher ranked classes.

Lower ranked classes can specialize (override) or extend higher classes.

On code level Inheritance may be used to **avoid redundancy** by allowing code reuse.

Supports
Don't Repeat Yourself (DRY)
principle
[A. Hunt and D. Thomas, The Pragmatic Programmer, 1999]

Inheritance enables Polymorphism!



Object Oriented Programming - Polymorphism

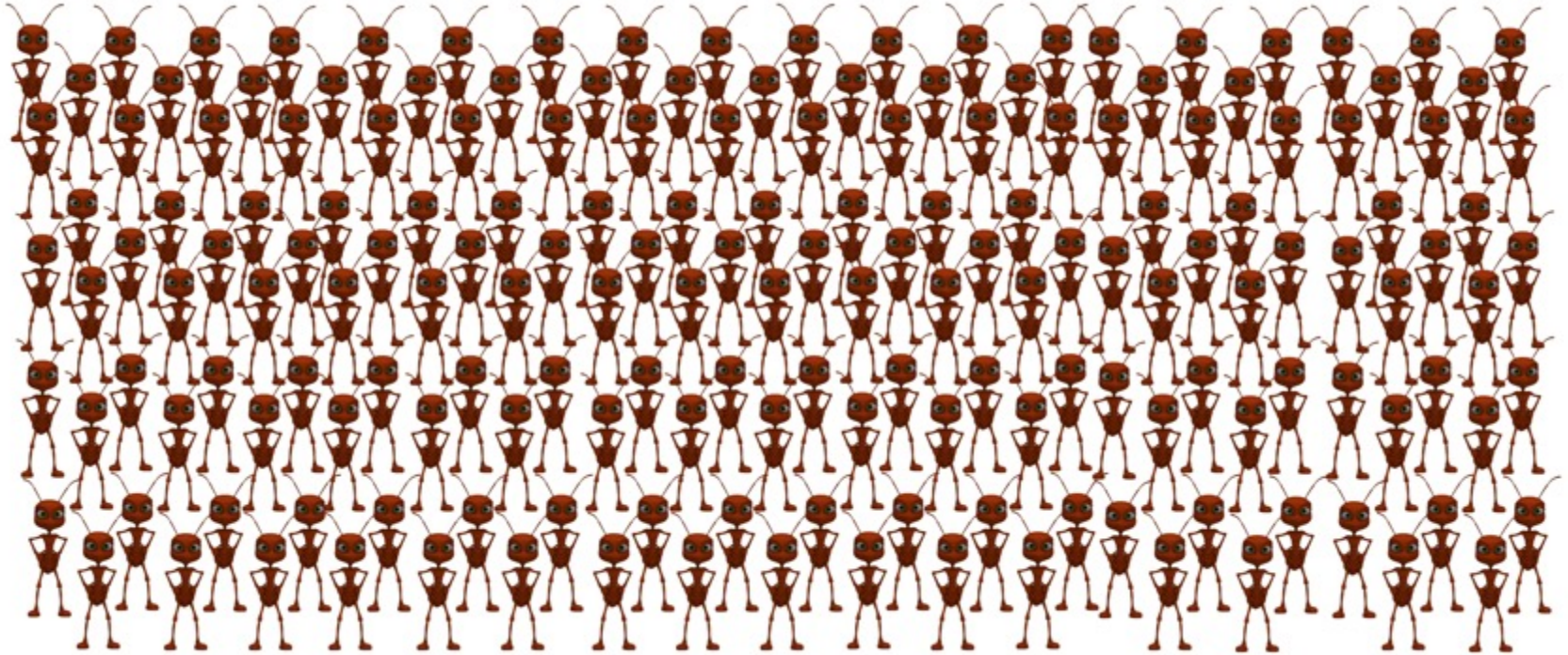
Generally, the ability to have different individuals of a species.
(…also in Biology, Chemistry)

In object-oriented programming, polymorphism refers to a programming language's ability of objects to react differently to one and the same message depending on their class.
(Technically, this is achieved by redefining methods in derived classes - Inheritance)

One may also speak of the autonomy or independence of objects.

Polymorphism - Ant Hill

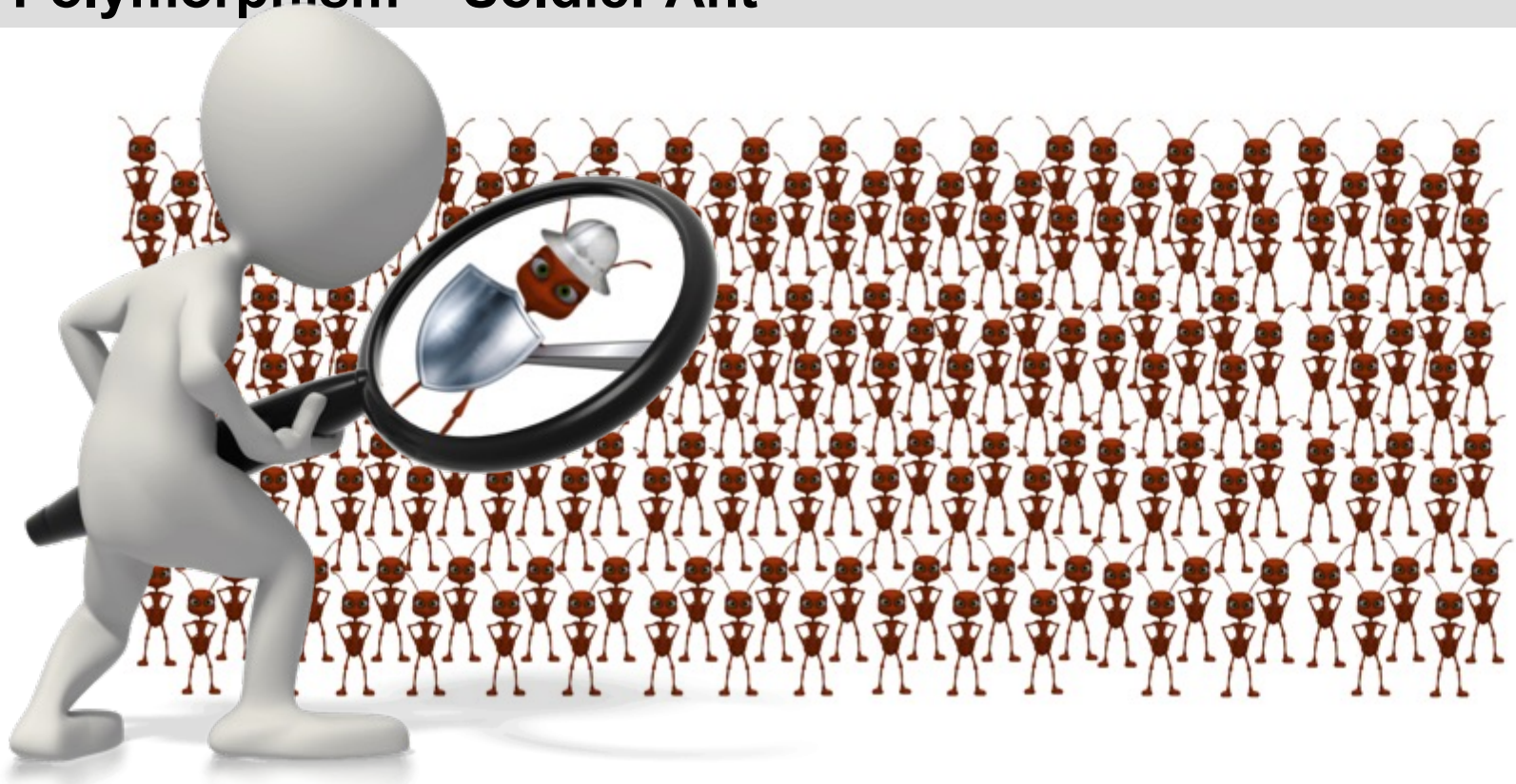
Polymorphism – All are ants



Polymorphism – Queen is managing the tribe (Core Unit)

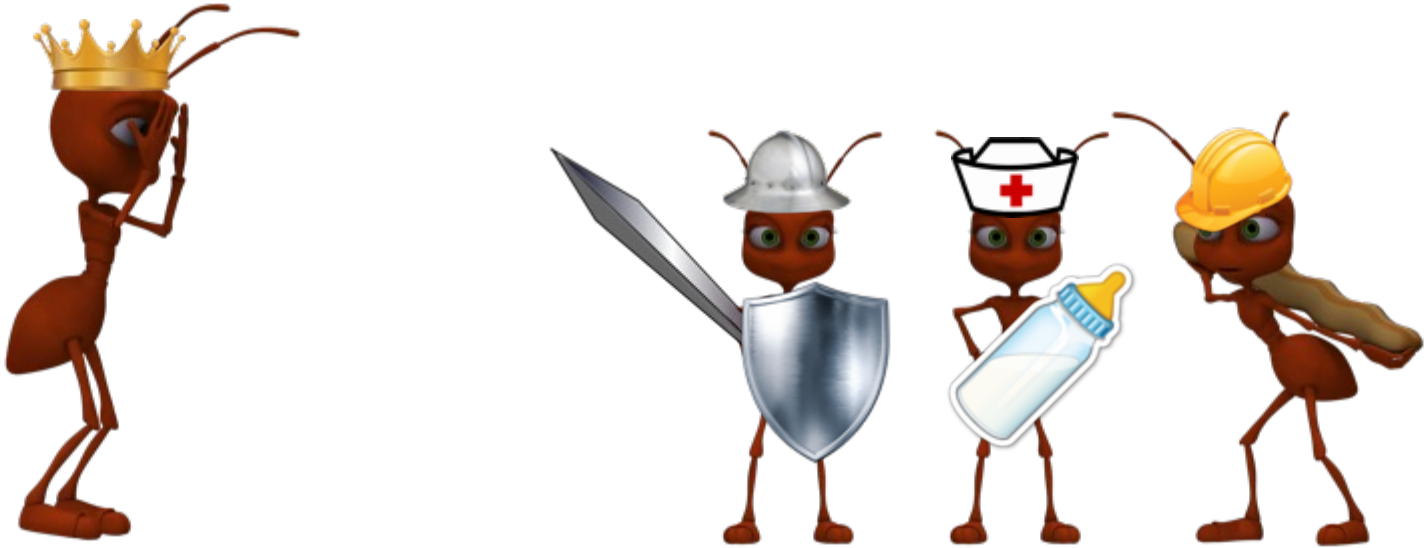
Polymorphism - Worker Ant

Polymorphism – Soldier Ant

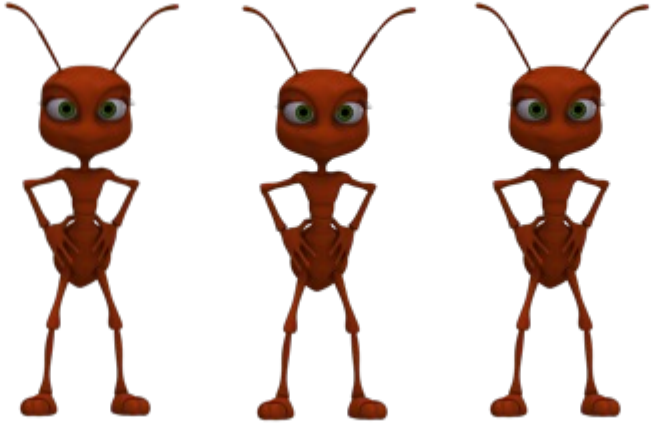


Polymorphism – Nurse Ant

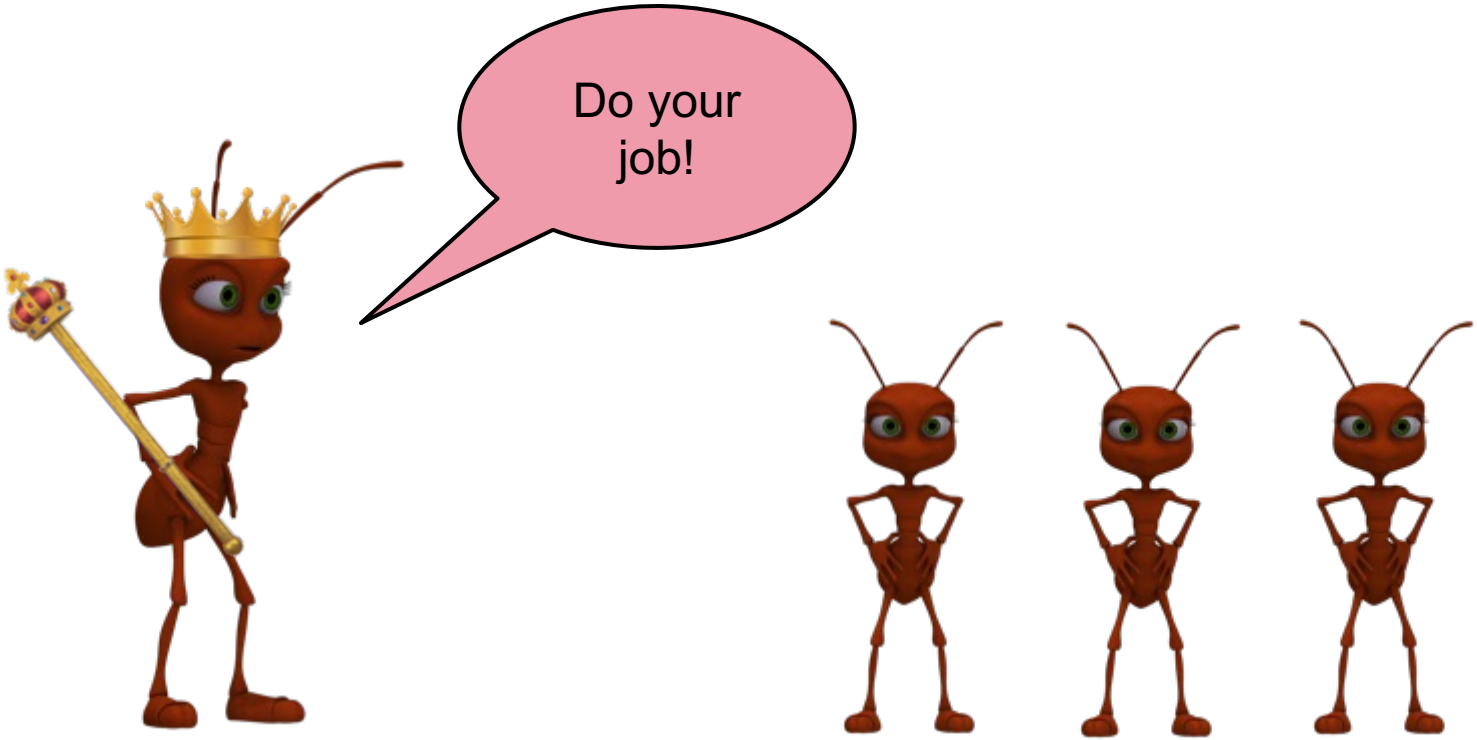
Polymorphism – Queen doesn‘t know specific ants



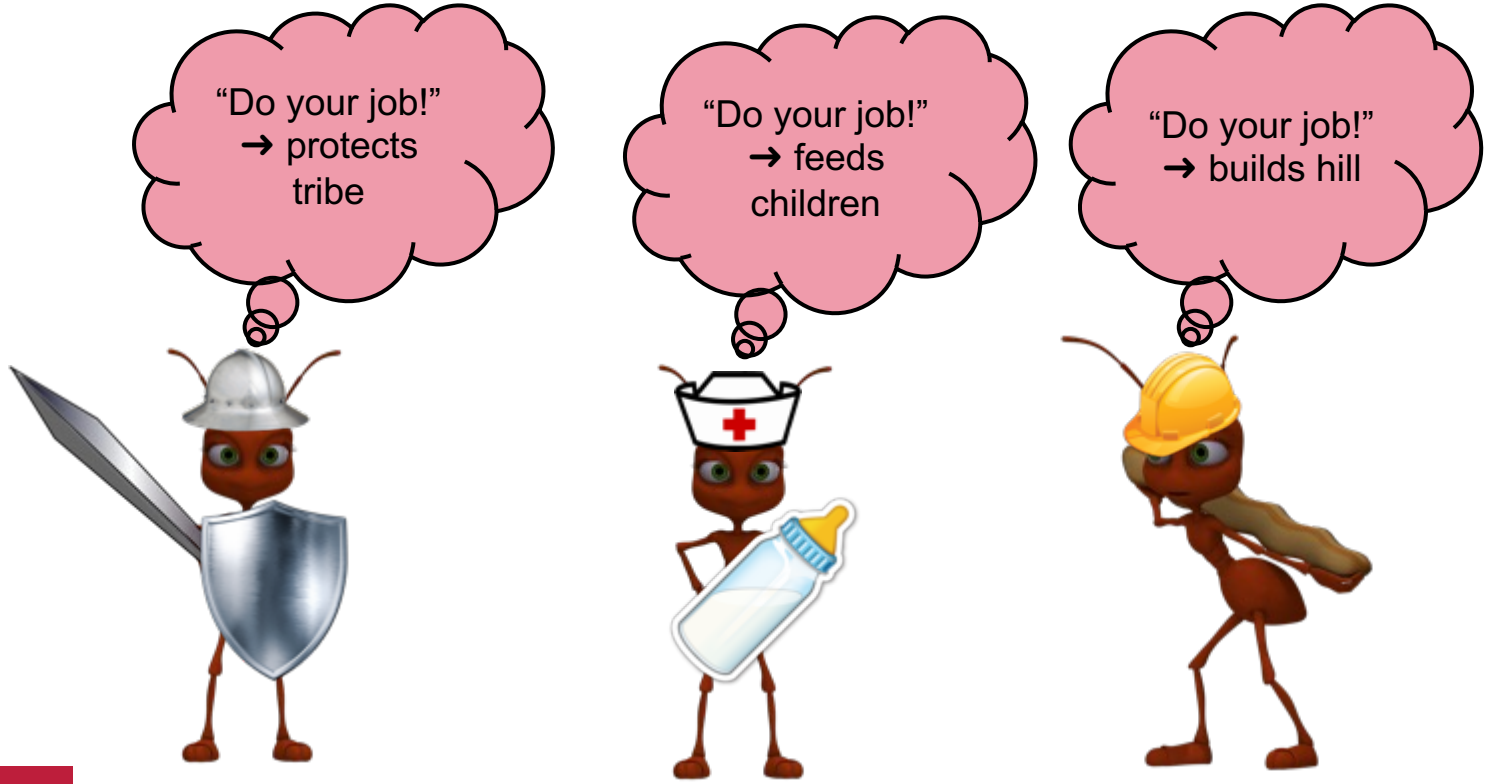
Polymorphism – Queen only knows ants



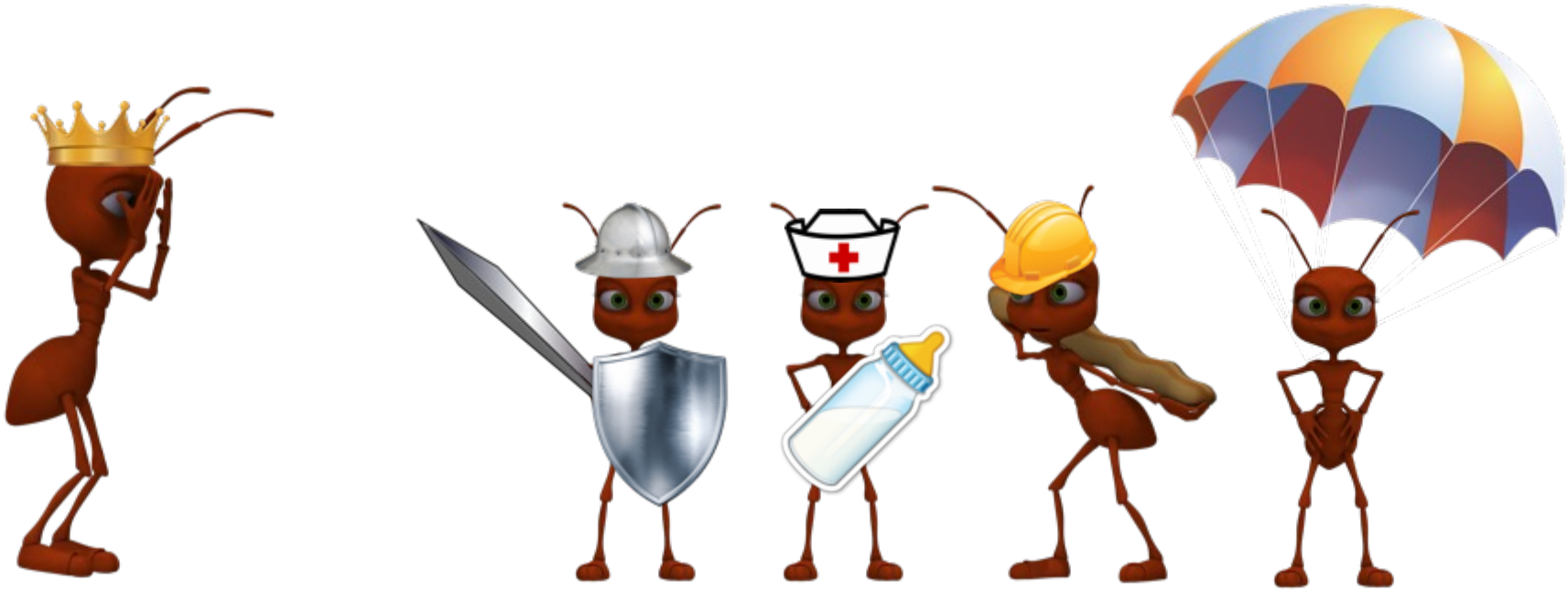
Queen knows that ants understand the message: „DO YOUR JOB!“



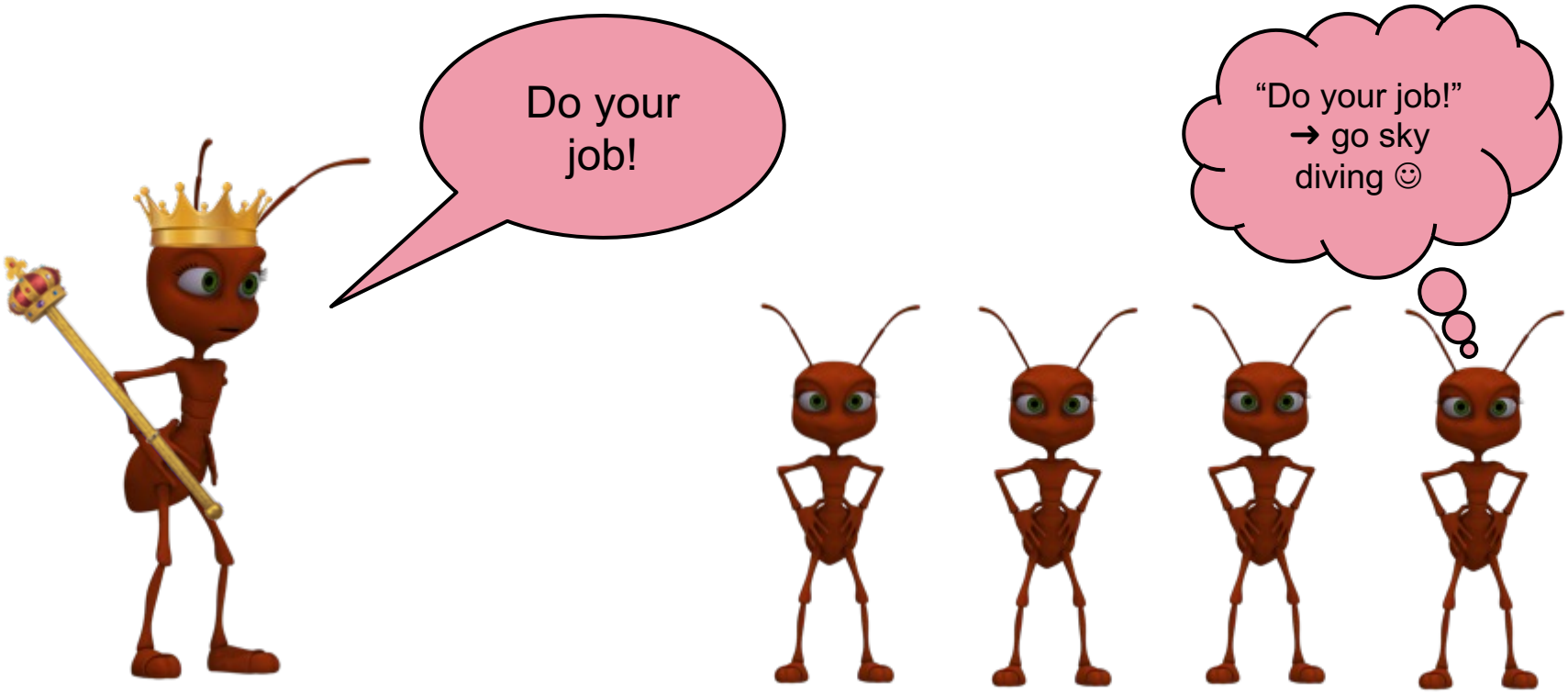
All react differently to the message according to their subtype



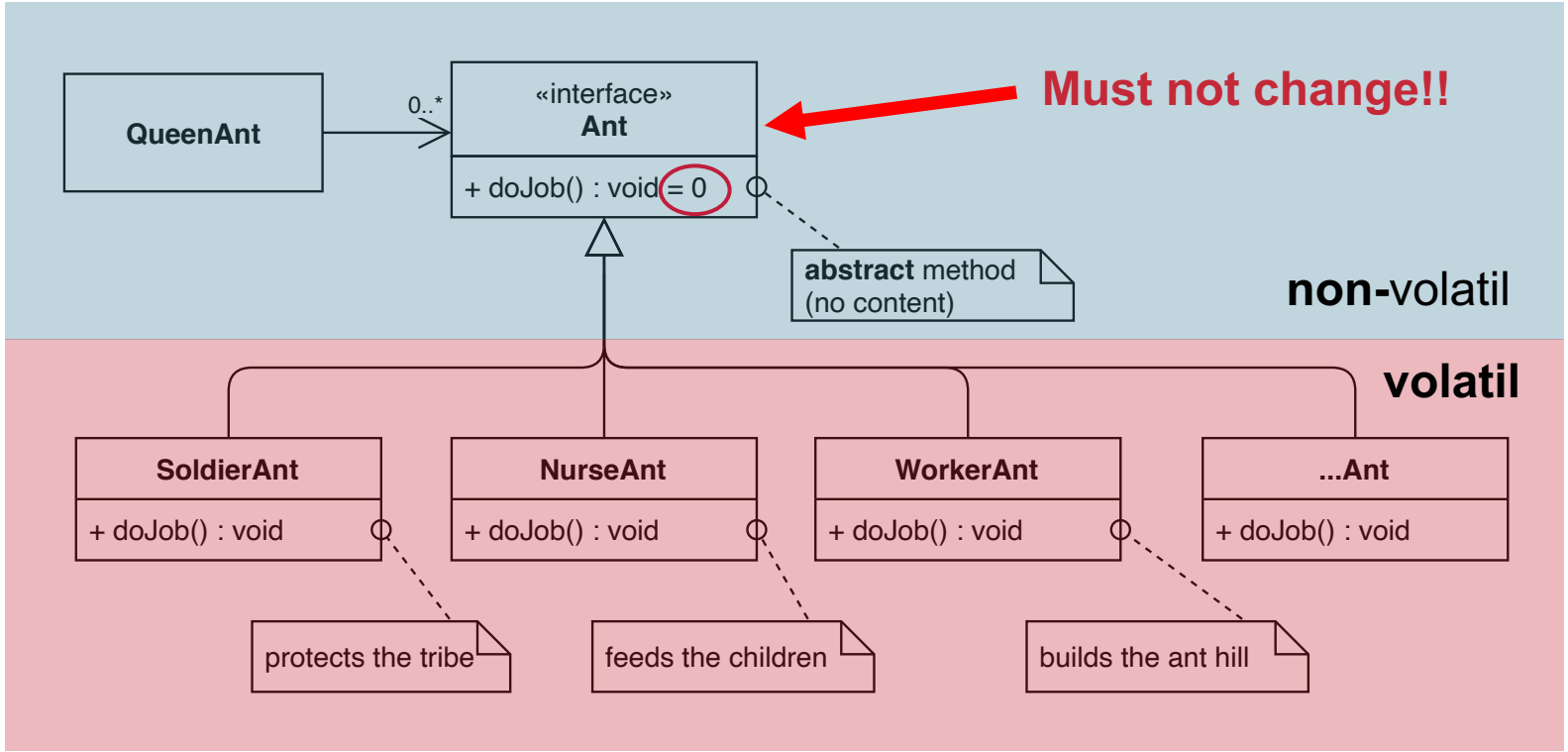
Polymorphism – Ready for Evolution - SkyDriver Ant



Polymorphism – Ready for Evolution - SkyDriver Ant

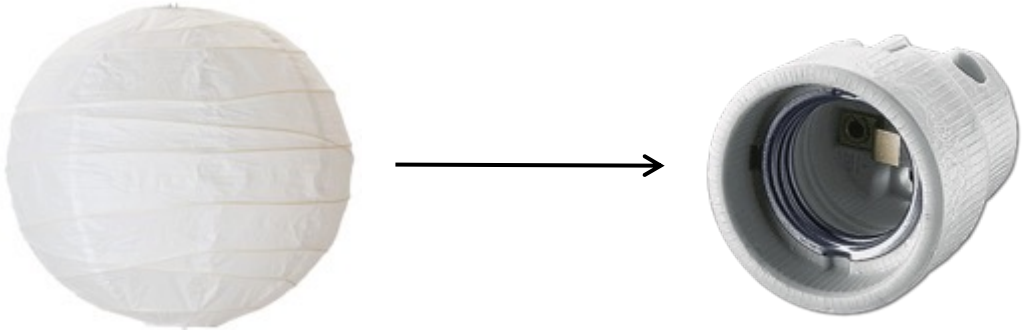


Polymorphism – Queen Ant is not affected by change



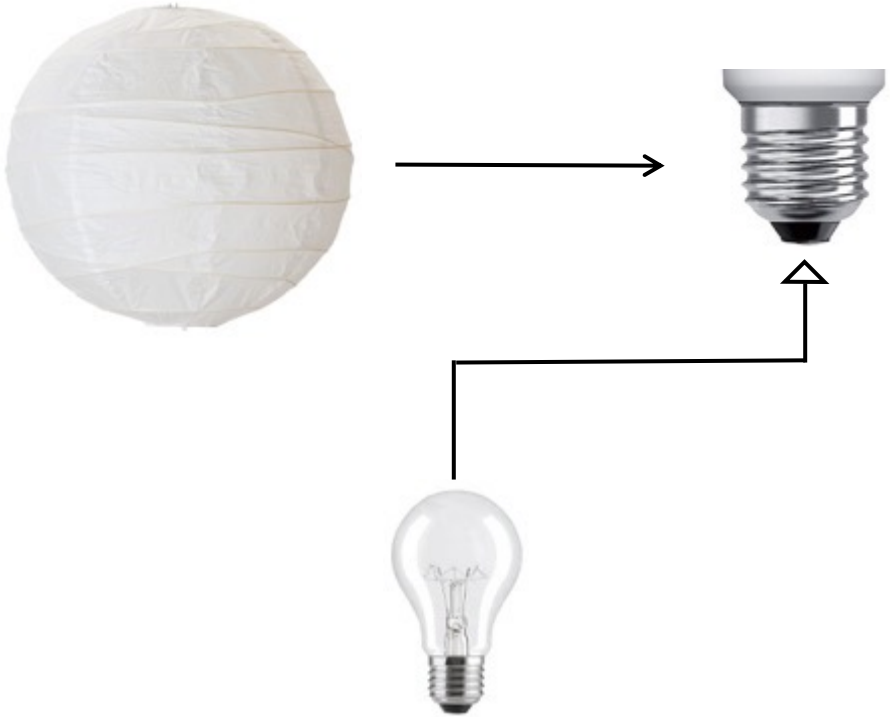
Object Oriented Programming - Polymorphism

Object Oriented Programming - Polymorphism



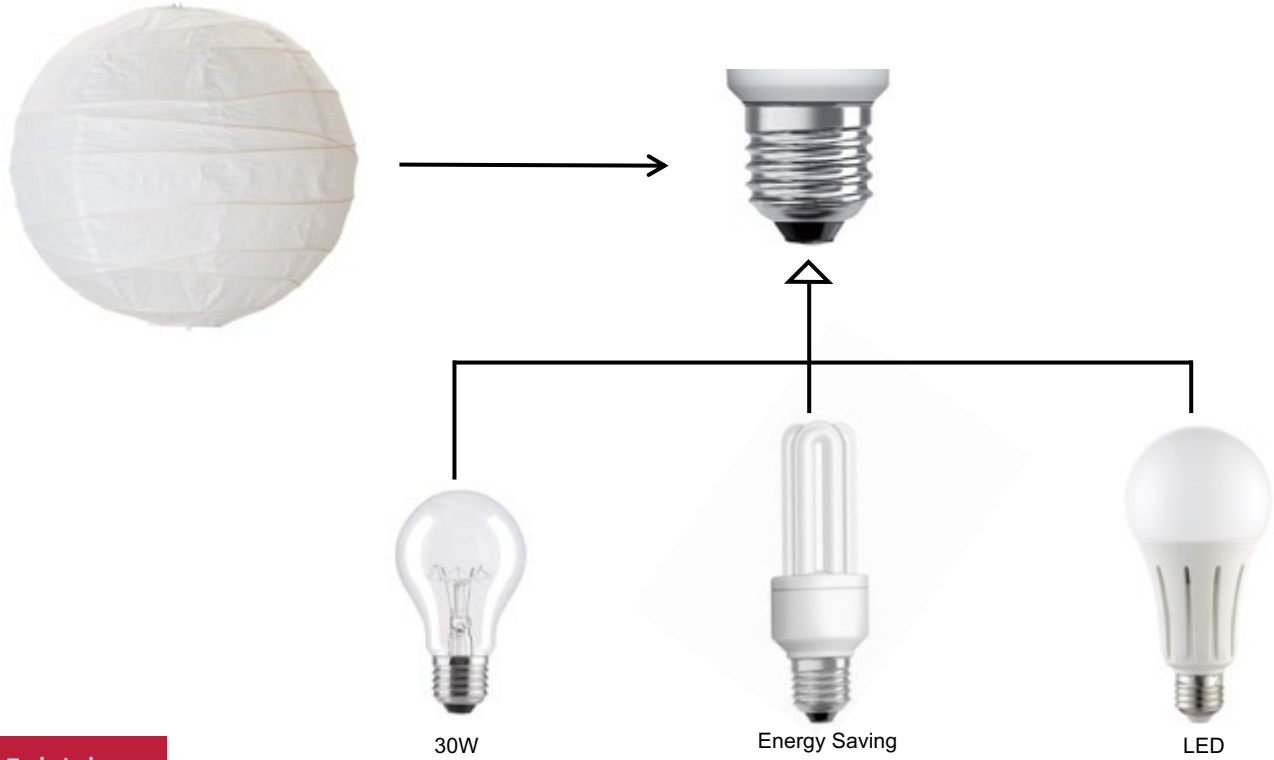
Object Oriented Programming - Polymorphism

Object Oriented Programming - Polymorphism



Object Oriented Programming - Polymorphism

Object Oriented Programming - Polymorphism



Object Oriented Programming - Polymorphism

Must not change!!

30W

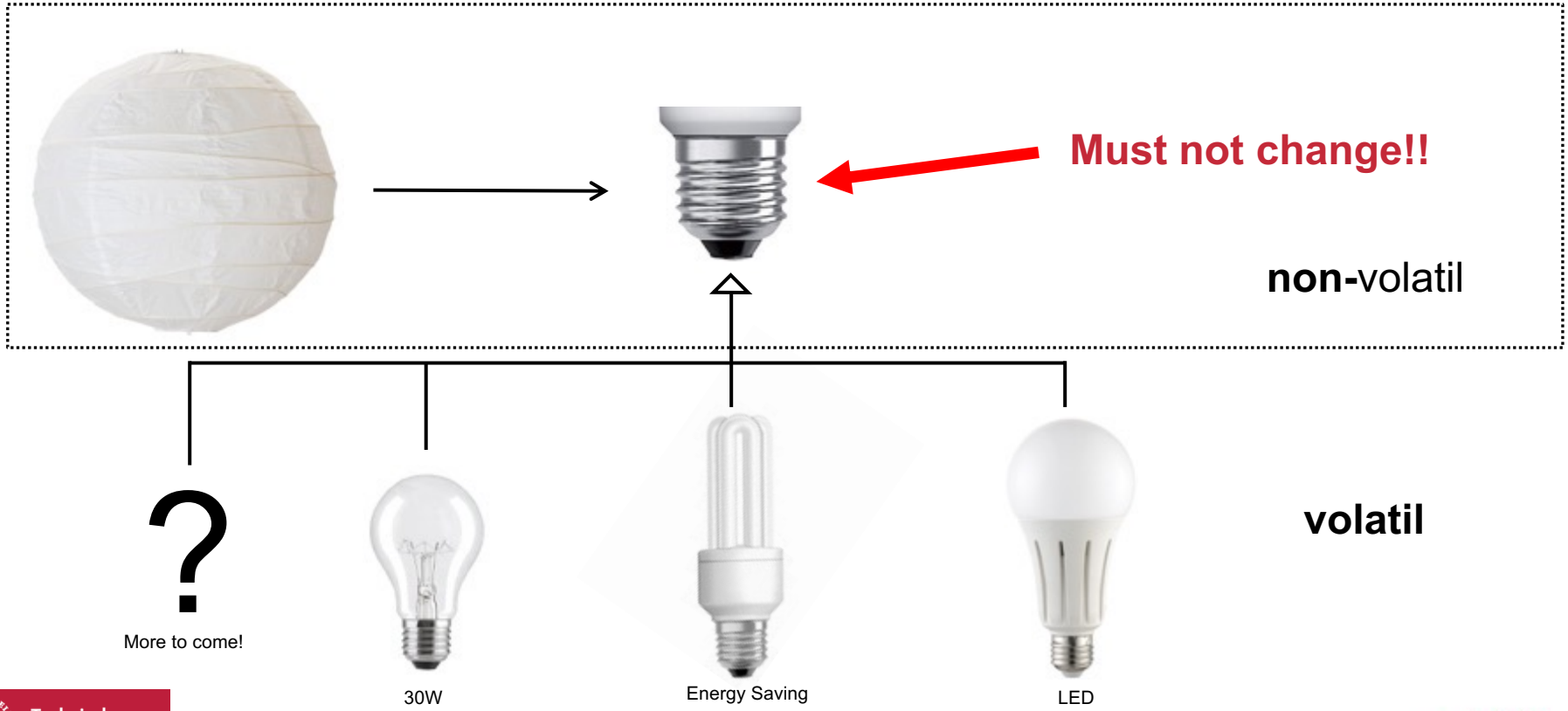
Energy Saving

LED

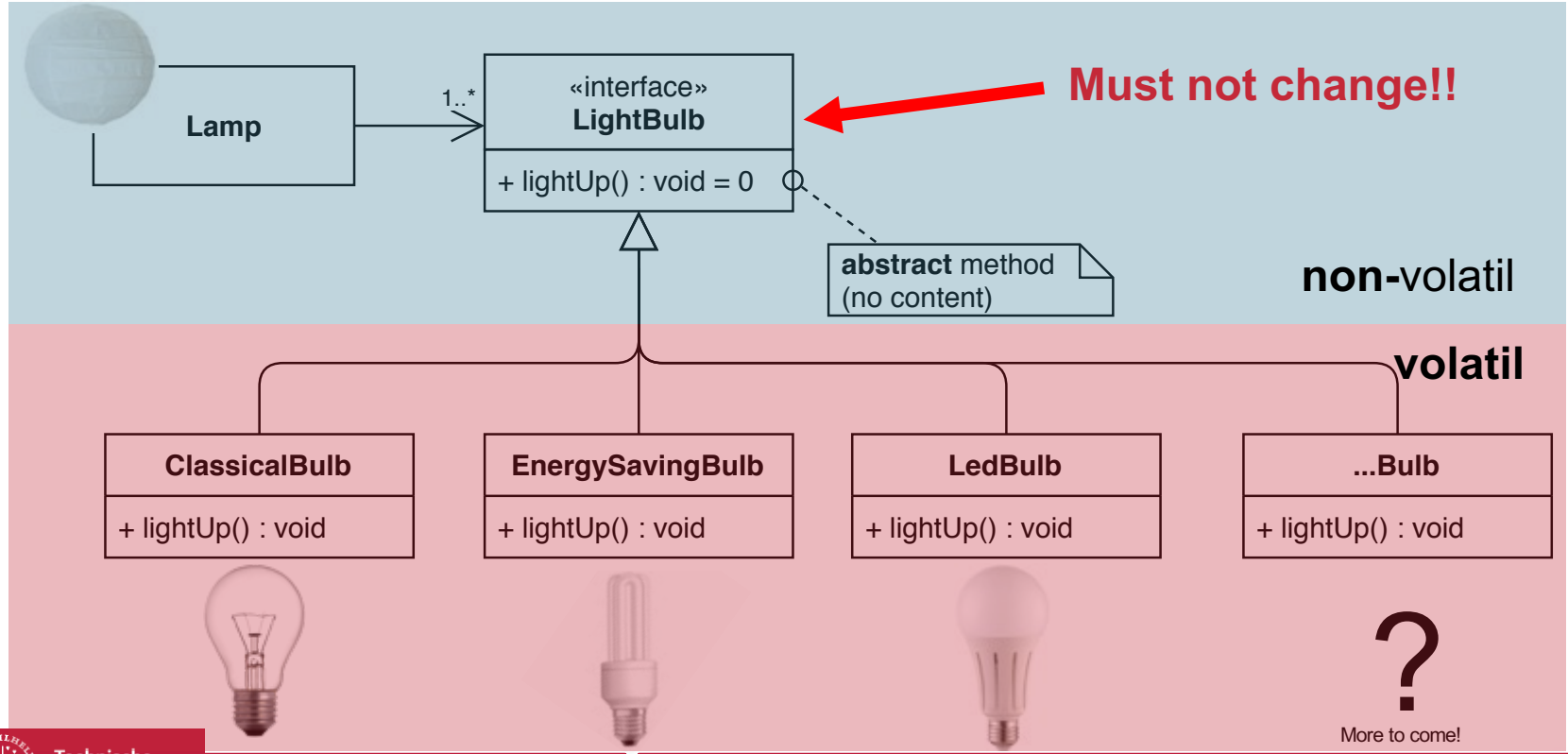
?

More to come!

Object Oriented Programming - Polymorphism



Object Oriented Programming - Polymorphism



References

Objektorientierte Programmierung - Das umfassende Handbuch
von Bernhard Lahres, Gregor Rayman, http://openbook.rheinwerk-verlag.de/oop/

Frage

A. Ja
B. Nein
C. Auf jeden Fall!
D. Ich hab die Frage nicht verstanden!