Object-Oriented Programming

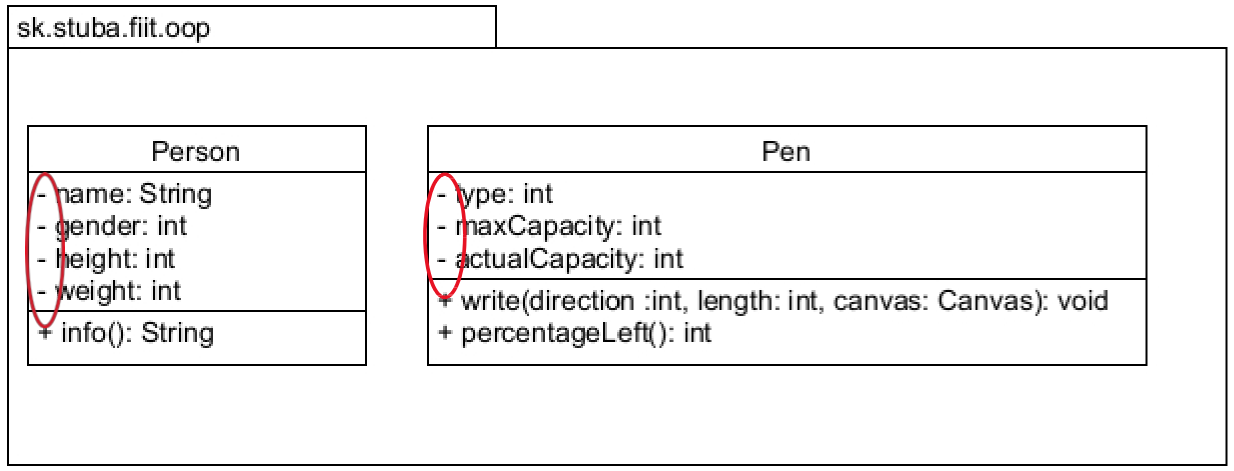
2. Lecture
LS 2025/2026
Author: Juraj Petrík

Back to the roots

- Let's return to the basic principles of OOP

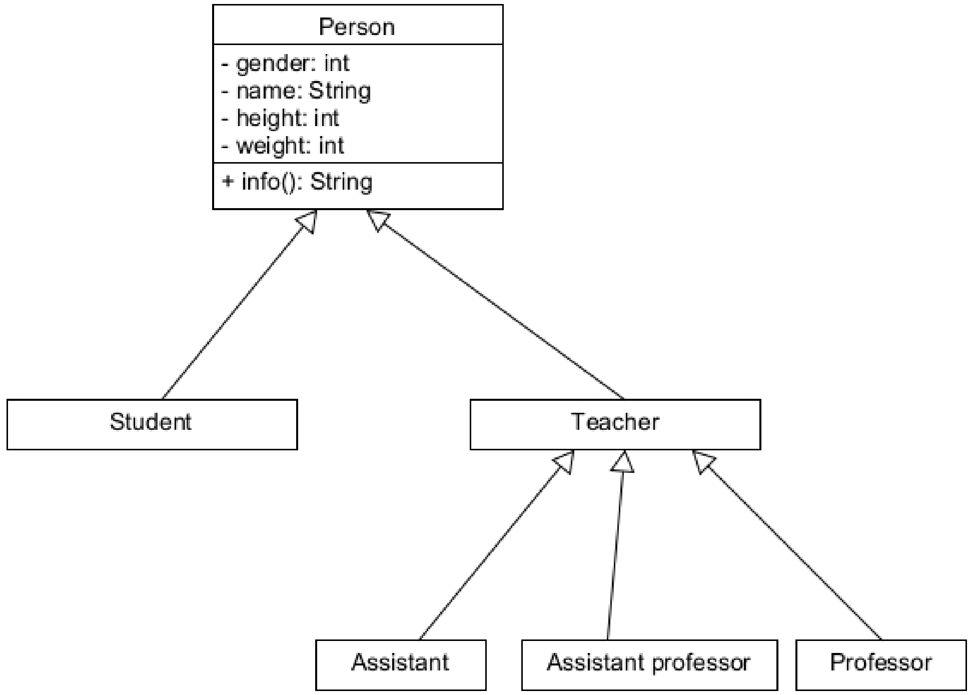
Encapsulation

- We do not manipulate data directly, but using methods
- In Java, for example, the keyword private (helps to achieve this)
- Mutators, but not only that – it is the overall concept that I do not work directly with data outside the class



Inheritance

- Allows classes to be "derived" from other classes
- A tree is created
- In Java, the keyword extends



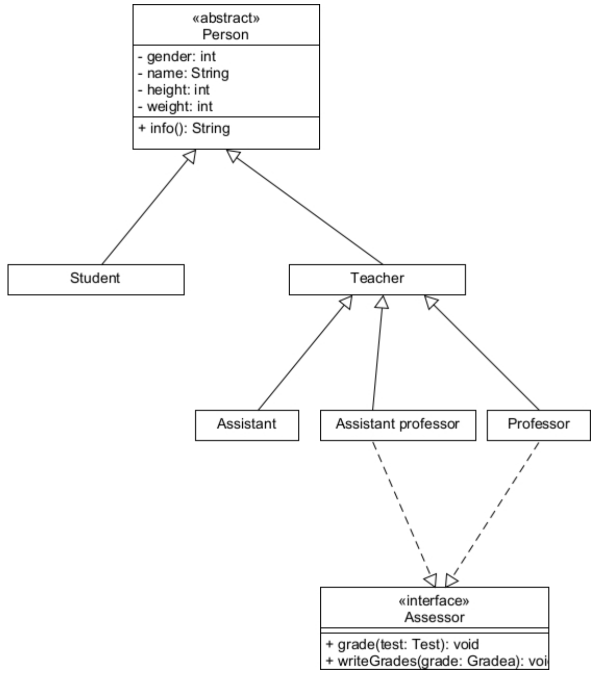
Polymorphism

- Compile time
  - Method overloading – same method name, different parameters
- Runtime
  - Method overriding – inheritance, specificity of behavior
  - *Virtual functions*

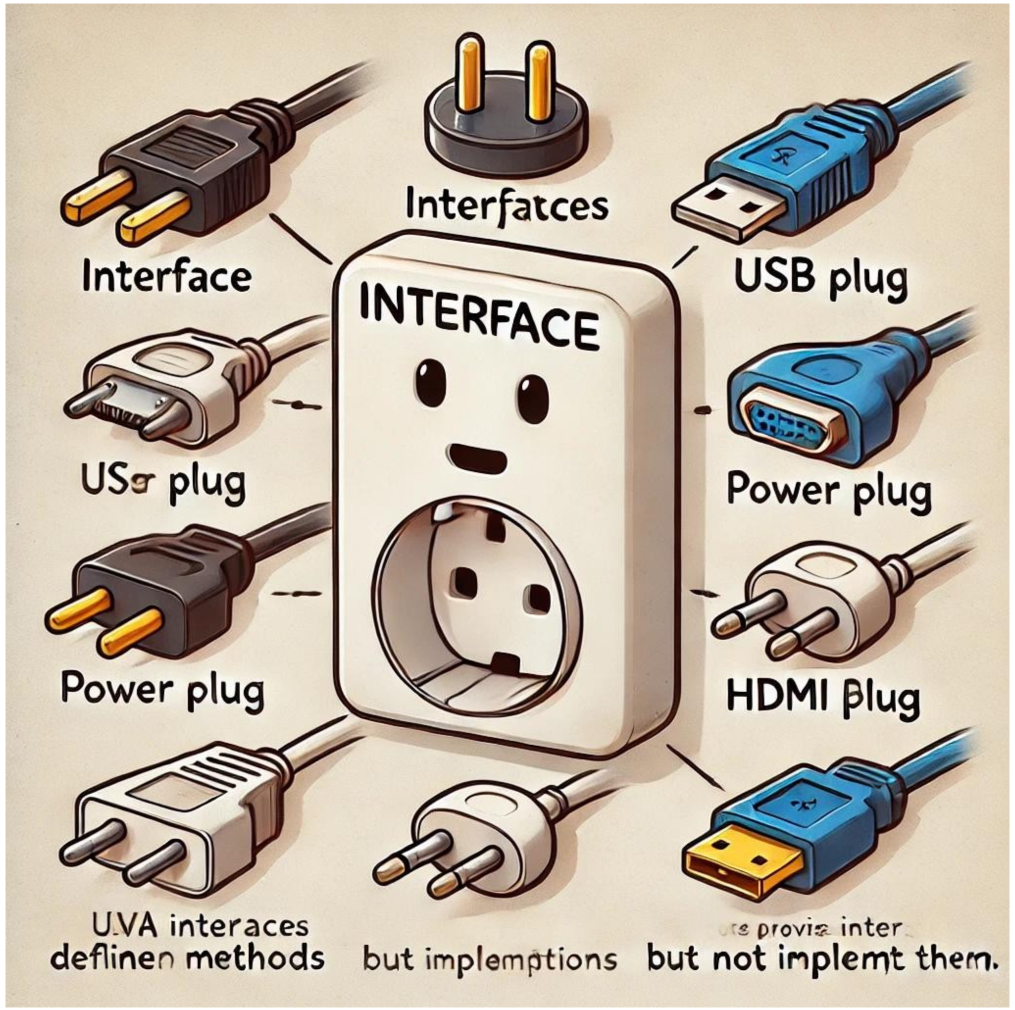
Abstraction

- “Hiding” internal implementation
- Abstraction is at the design level, encapsulation is at the implementation level
- In Java, for example, Abstract class and

    Interface (also help)



Interface


Interfatces
Interface
INTERFACE
USB plug
USɠ plug
Power plug
Power plug
HDMI Plug
U.VA interaces
deflinen methods
but implemptions
provis inter
but not implemt them.

Interface II.

INTERFACE
ABSTRACT CLASS
Corporate needs you to find the differences between this picture and this picture.
They're the same picture.
imgflip.com

Classes II.

- Constructors:
  - Object initialization
  - Not inherited
  - If there is none, an empty one is automatically created
- this vs super

Methods

- Java is **pass by value (copy)**
- Any number and type of parameters
- We return max one "object"

Class attributes vs. local variables

- Class attributes (instance variables) are declared in the class, not in the method
- Local variables are declared in the method

- Class attributes always have a default value
- Local variables must be initialized before use

- == vs equals()

Access modifiers

- Classes
  - Public – everywhere
  - *Default* – only from the same package
- Attributes, methods, constructors
  - Public – accessible to all classes
  - Private – only in own class
  - *Default* – only from the same package
  - Protected – only from the same package and subclasses

Class design and implementation

- What should the class do?
- What attributes?
- What methods?
- *Tests*
- Pseudocode for methods
- Implementation
- Testing
- Debugging and correction

Project

- Create a game in Java with a graphical user interface (GUI). The goal is not to focus on graphics (2D or 3D) or multiplayer networking, but on the use of (proper) object-oriented programming and the features of the Java language. As an aid to thinking, use, for example, a list of game genres (https://en.wikipedia.org/wiki/List_of_video_game_genres).
- NOT Minecraft-like games

Project

- Develop a project focus (maximum length 250 words), a more specific description of what you are going to do, but NOT a technical description of what you are going to do. Imagine that you are going to present the game to the manager of a game studio (i.e., someone who knows basically nothing about programming).
- Create a simplified specification (requirements) of what the game should "do," assign them a priority (at least 10 functional and 3 non-functional), and then implement your project according to this priority.
- Identify the most important classes and their most important methods, and draw them in a simplified UML class diagram.
- Remember that you are studying OOP, so when designing your project, keep in mind that you will need to use an object-oriented paradigm in the implementation itself.
- Consult the focus with the instructors

Quiz time