AARHUS UNIVERSITET

Software Engineering and Architecture

Pattern Catalog: Decorator

AARHUS UNIVERSITET

New Requirement

- Alphatown wants to log all coin entries:
  - [time] [value]

- Example:
  - 14:05:12 5 cent
  - 14:05:14 25 cent
  - 14:55:10 25 cent

- ☺

AARHUS UNIVERSITET

The 3-1-2 machinery

- Let us look at the machinery:
- ③ Identify the responsibility whose concrete behaviour may vary
- ① Express responsibility as an interface
- ② Let someone else do the job

- How does this apply?

- What is 3-1-2 here?

AARHUS UNIVERSITET

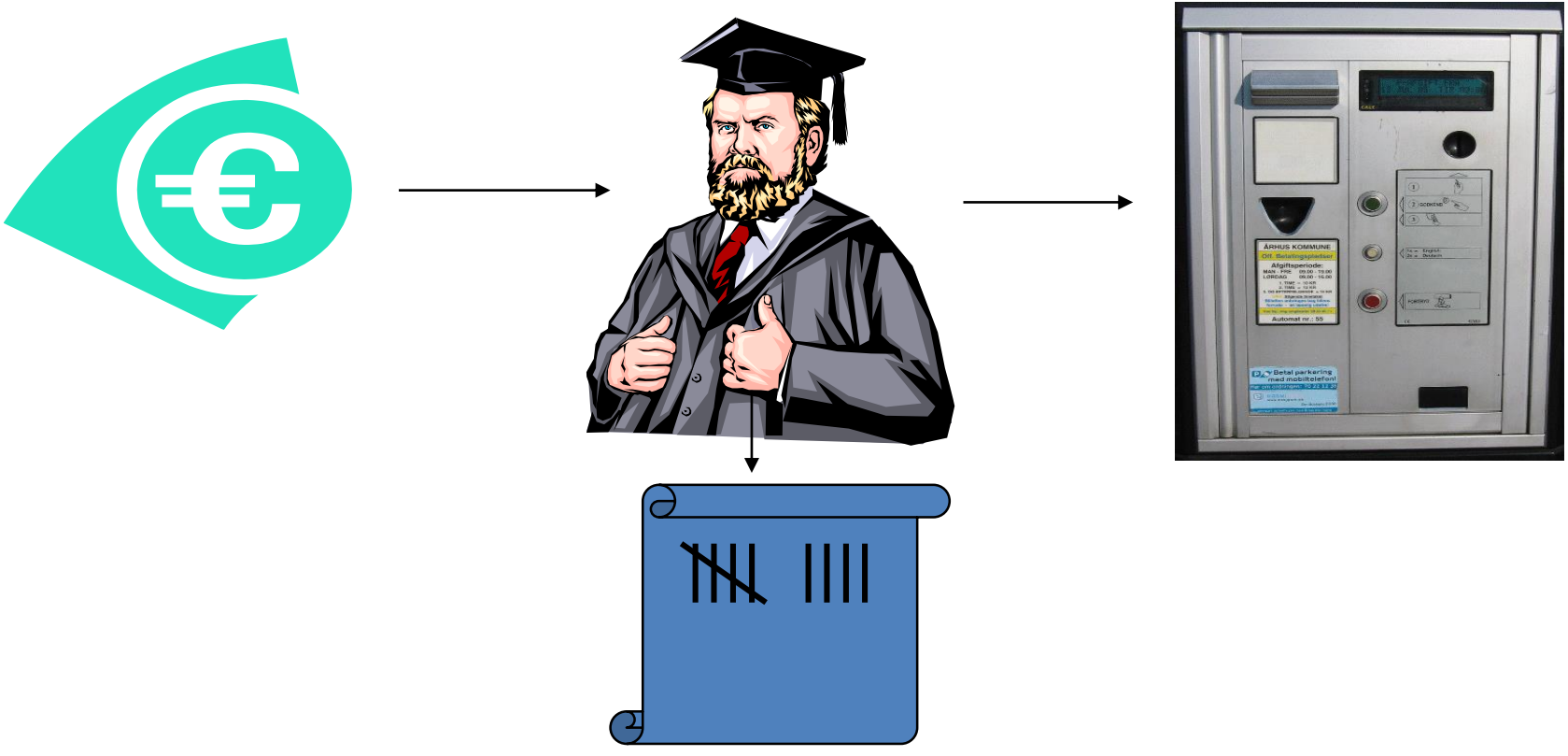
Analysis

- ③ Identify the responsibility whose concrete behaviour may vary
  - It is the “Accept payment” responsibility

- ① Express responsibility as an interface
  - A) PaymentAccepter role?
  - B) PayStation role? Already in place!
- ② Let someone else do the job
  - Maybe let someone handle the coins *before* the parking machine receives them?

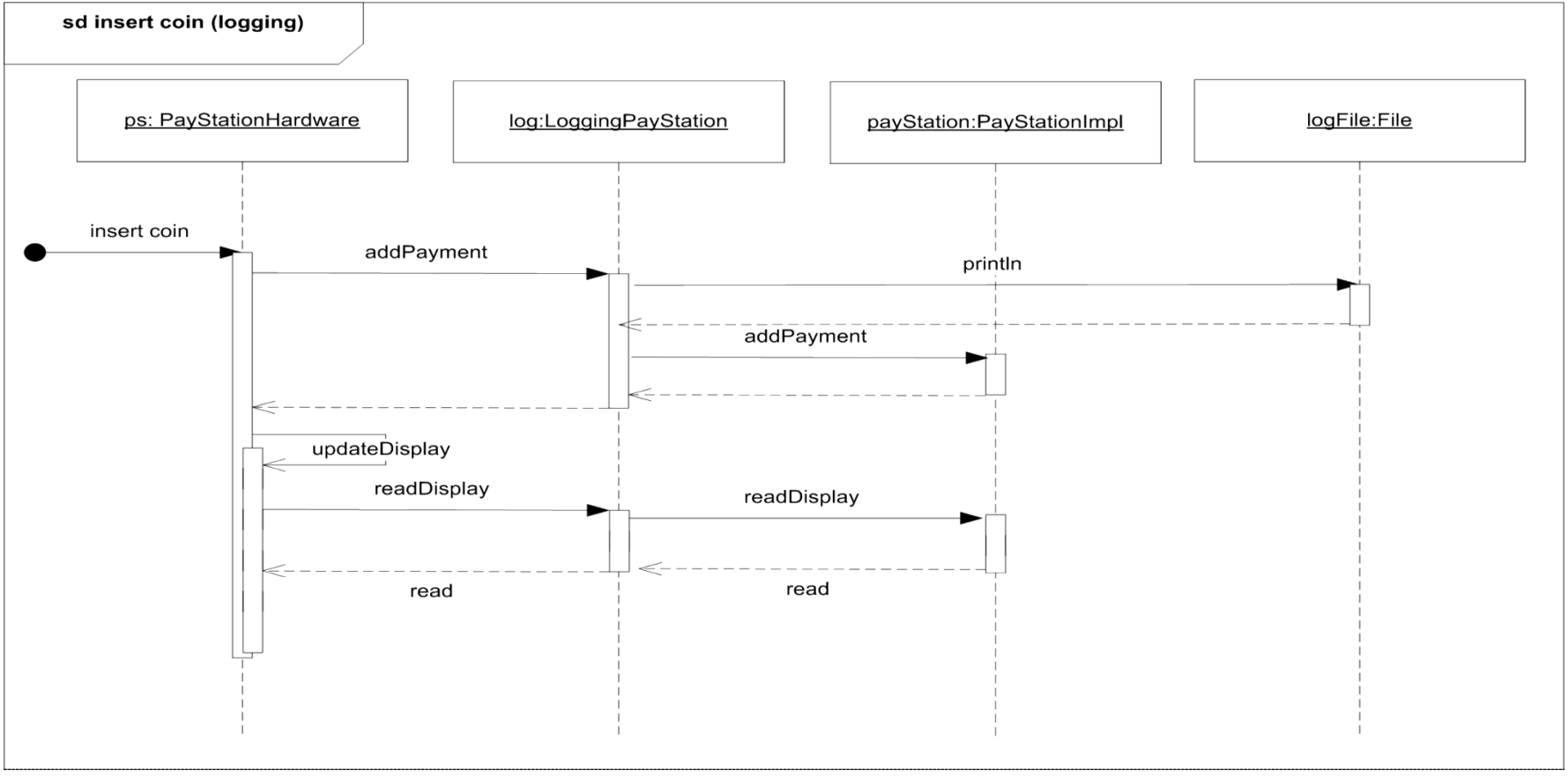
AARHUS UNIVERSITET

Metaphor: Principle 2



AARHUS UNIVERSITET

Dynamics

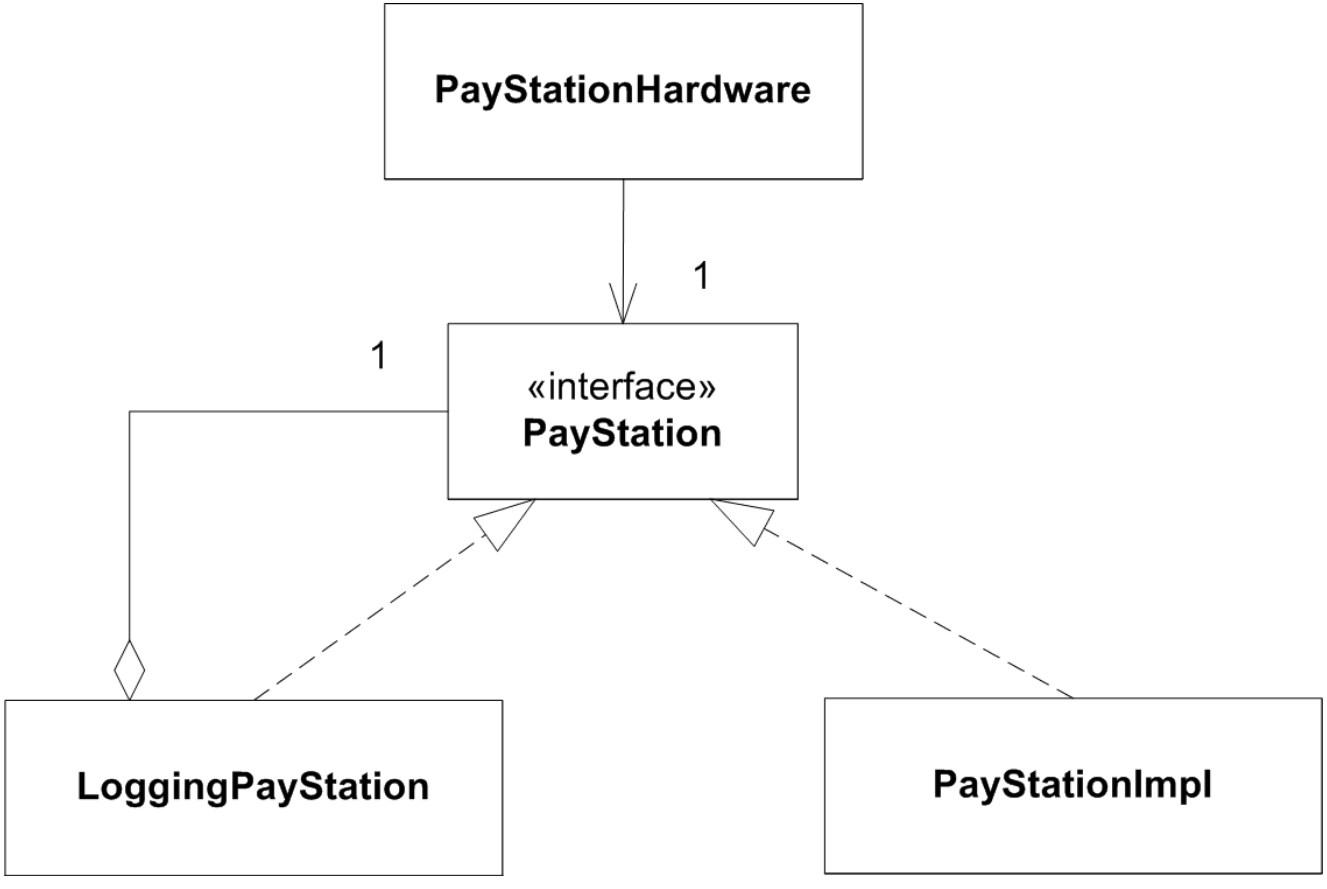


AARHUS UNIVERSITET

[Demo]

- Refactoring process – solution first programming
  - Establish basis: run TestPayStation
  - ps = new LogDecoratedPS(ps);
  - In LogDecoratePS do
    - Intro ‘private PayStation delegate;’
    - Select it, and choose menu ‘Code/Delegate methods…’
  - Rerun tests
  - Introduce the decorating statement in the LogDec…
  - Flip the pointers to enable/disable at runtime

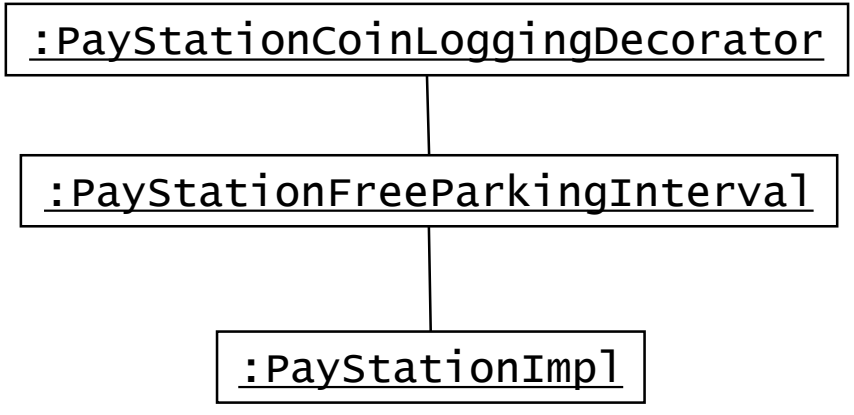
Structure


PayStationHardware
1
1
«interface»
PayStation
LoggingPayStation
PayStationImpl

AARHUS UNIVERSITET

Chaining decorators

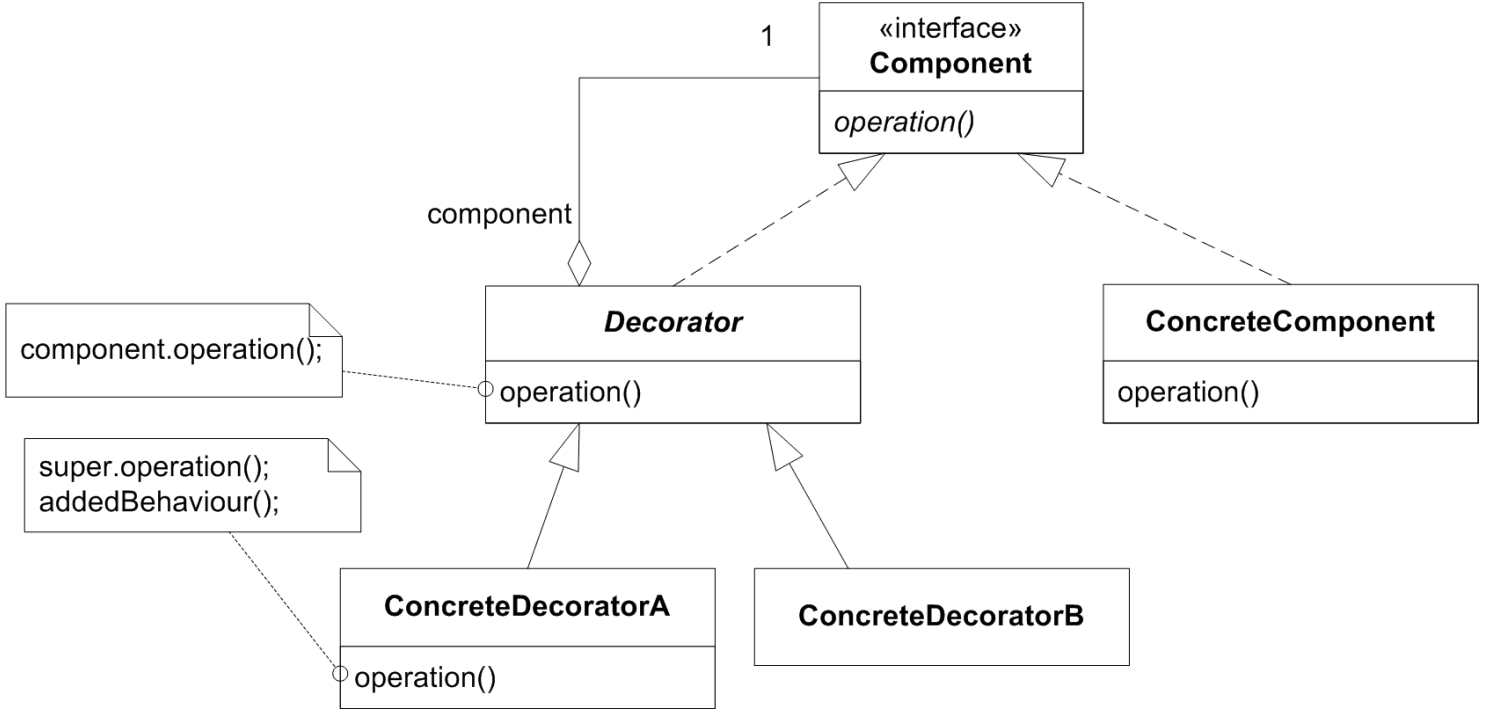
- Decorators can form chains.
- New requirement:
  - no payment possible in 19.00 – 07.00 interval



AARHUS UNIVERSITET

Automagical pattern?

- The decorator is yet another application of 3-1-2 and the principles of flexible design!



AARHUS UNIVERSITET

Consequences

- Benefits
  - Adding and removing behavior at run-time
  - Incrementally add responsibilities
  - Complex behavior by chaining decorators
- Liabilities
  - Analyzability suffers as you end up with lots of little objects
    - Behavior is constructed at run-time instead of being written in the static code
  - Delegation code tedious to write (without IDE ☺)
    - Make a ‘null decorator’ as base class