MUNI
FI

lasaris LAB OF SOFTWARE ARCHITECTURES AND INFORMATION SYSTEMS

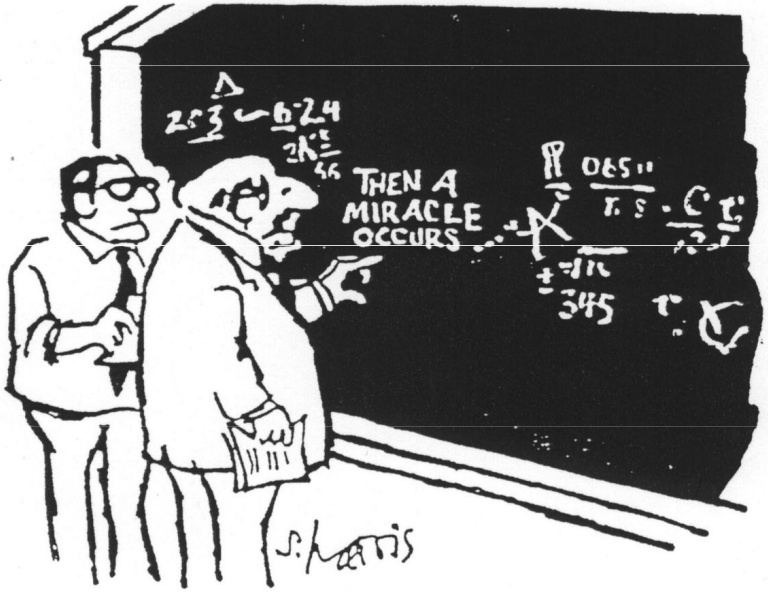
Design Class Diagram

PB007 Software Engineering I

Lukáš Daubner
daubner@mail.muni.cz

Design Class Diagram

- Focus on implementation details
- It goes further than capturing domain
  - Solutions to more technical problems
- Extends and enrich the analytical class diagram



I think you should be a little more specific, here in Step 2

MUNI FI

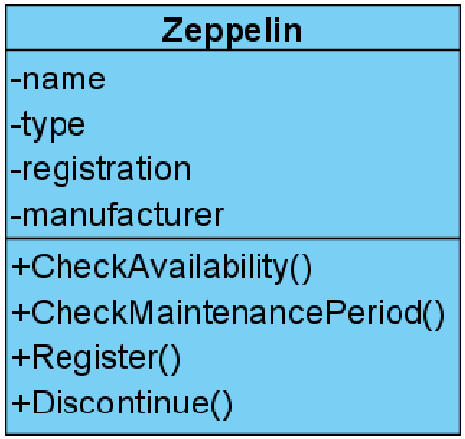
Design Class Diagram

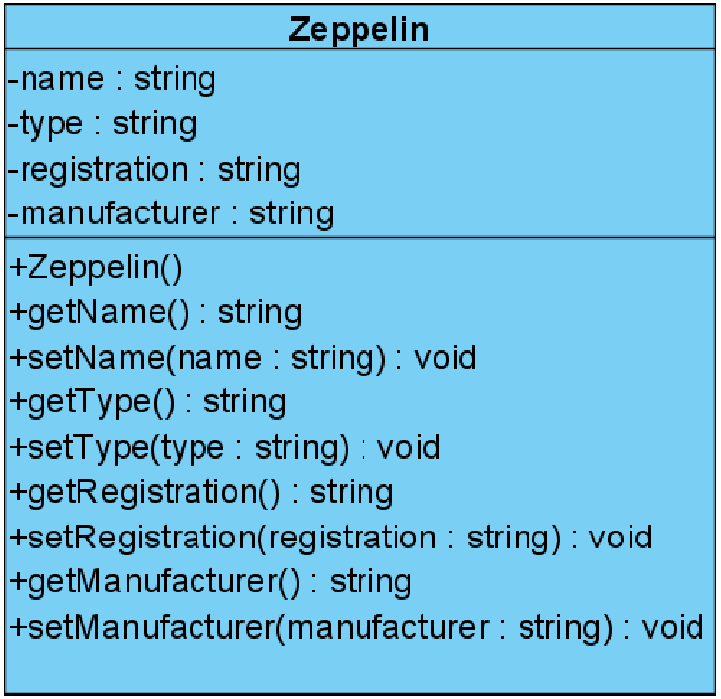
How should it look like?

- All that you need for implementation
  - Except method body (we will get to that)
- Detailed specification of analytical class
- Technology-related class (Service, Controller, DBContext, etc.)
- Visibility and types are specified
  - Attributes
  - Method arguments
  - Return values
- Constructor
- Properties (Getters, Setters)
- Methods needed for implementation

Design Class Diagram

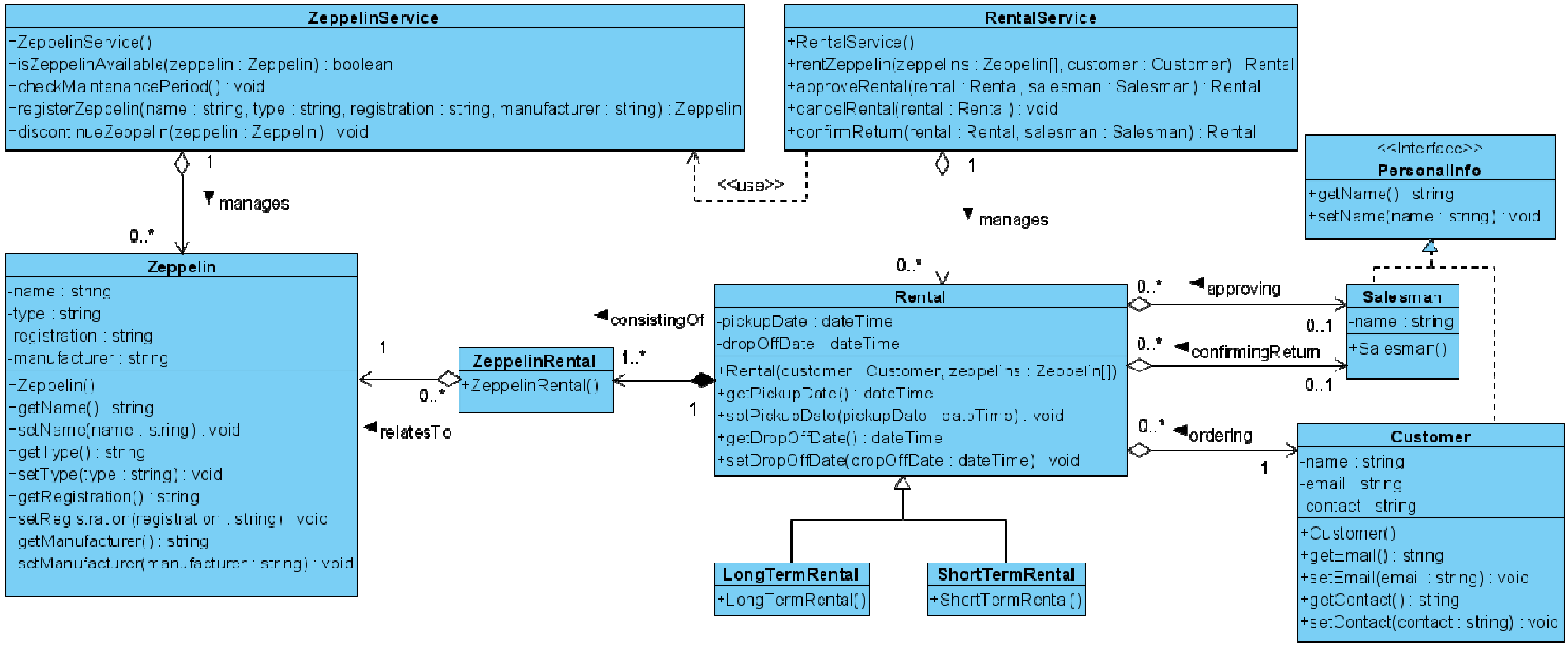
Analytical VS. Design class





MUNI FI

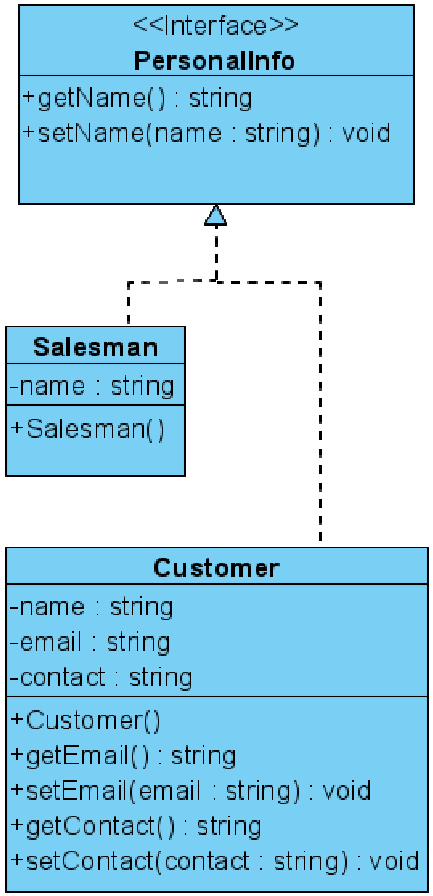
Design Class Diagram – Example



MUNI FI

Interface

- Defines set of public services
  - Methods
  - Attributes
  - Relationships
- Does not contain implementation
- Defines so called “Contract”



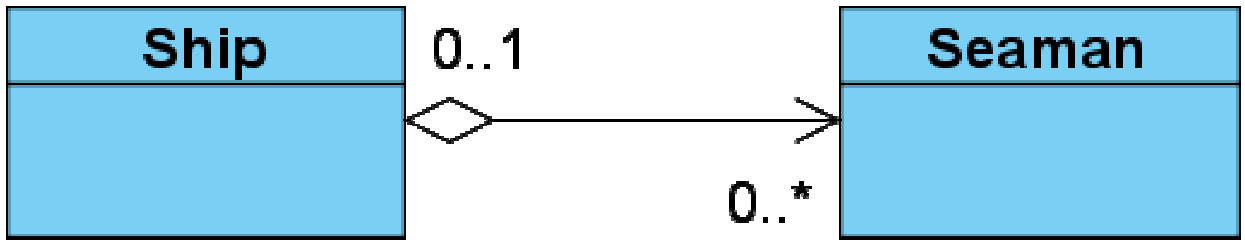
MUNI FI

Specialized Associations

- Specification of aggregation and composition (see following slides)
  - There is a lot of confusion regarding this topic.
    See https://bellekens.com/2010/12/20/uml-composition-vs-aggregation-vs-association/
- Names, navigability, and multiplicities
  - Afterall, it is an important part of the specification
- Decomposition of bidirectional associations
- Decomposition of M:N associations and association classes

Specialized Associations – Aggregation

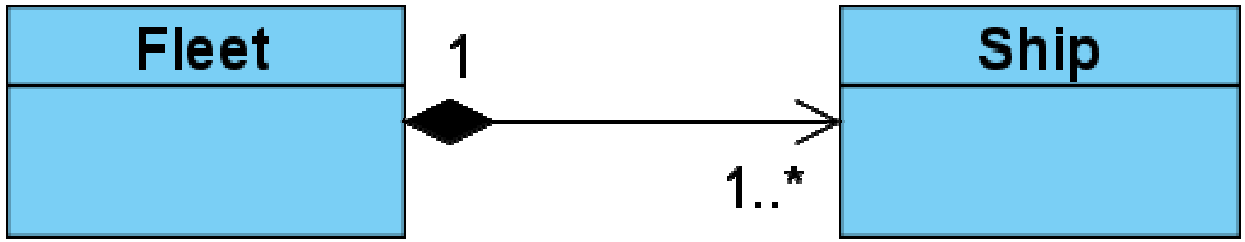
- Whole-part relationship
  - The whole may and may not exit without its parts
  - Parts can exist independently from the whole
  - The whole is in some sense incomplete if some parts are missing (but still valid)
  - Parts can be shared by multiple wholes
- Transitive and asymmetrical (without cycles)



MUNI FI

Specialized Associations – Composition

- "Stronger" form of aggregation
  - The part belong to **exactly one** whole in the given time
  - The part is not valid without the whole
  - The whole is responsible for lifecycle of its parts
  - When deleting, the whole must take care of its parts (delete or transfer them)
- Transitive and asymmetrical (without cycles)



MUNI FI

Aggregation vs. Composition

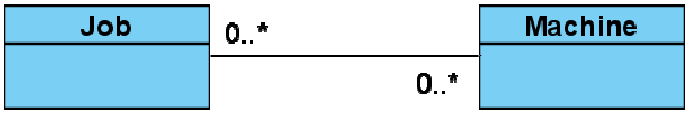
Code comparison

```
public class Ship
{
    private Engine _engine;

    public Ship(Engine engine)
    {
        _engine = engine;
    }
}
```

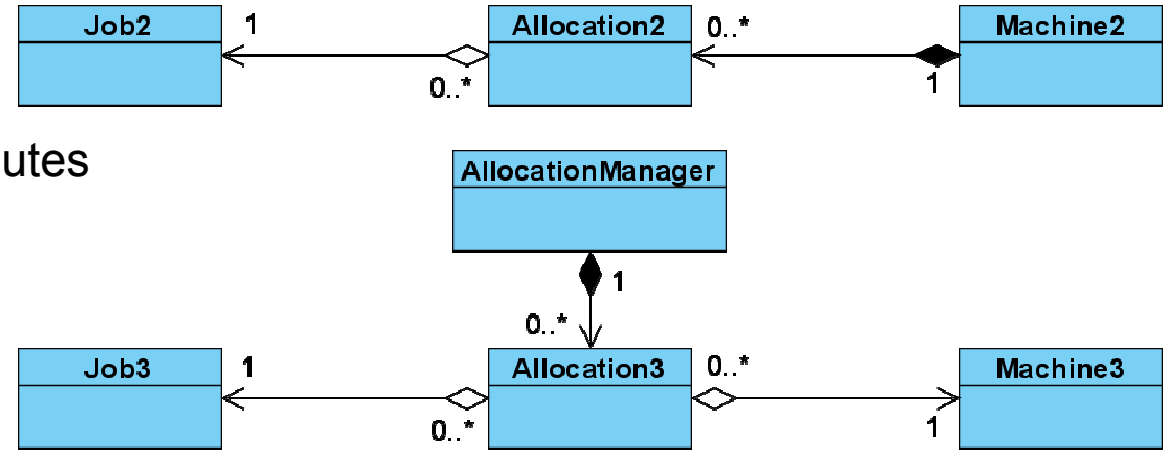
```
public class Ship
{
    private Engine _engine;

    public Ship()
    {
        _engine = new Engine();
    }
}
```

Association Decomposition – M:N

- Analytical



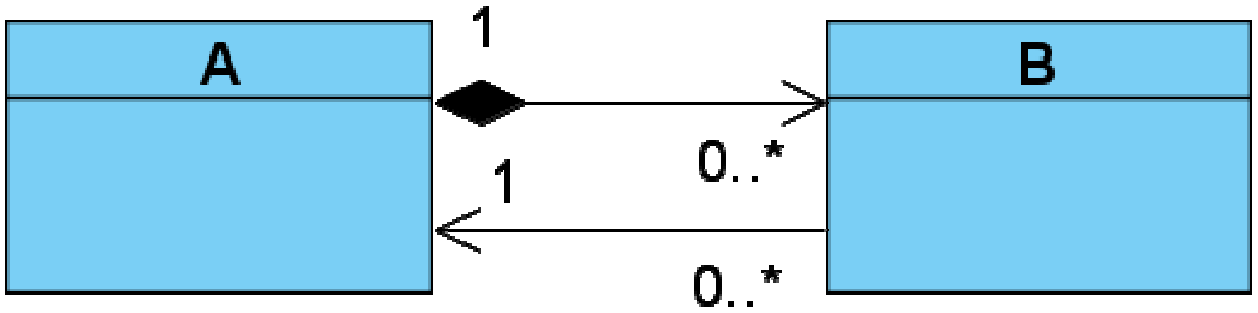
- Design
  - Decompose if there is a need for additional attributes
  - Someone must “own” it



MUNI FI

Association Decomposition – Bidirectional

- Someone must “own” it
  - To ensure consistency



MUNI FI

Task for this week

You gotta do what you gotta do

- Process the feedback
- Copy and extend analytical class diagram to design class diagram
  - Add all required methods, properties, constructors, etc.
  - Specialize and decompose associations when suitable
  - Type everything
  - Add technology-related and service classes
- NB! Do not delete the original analytical class diagram