MUNI
FI

PB007 Week 08

Samuel Sabo

Class diagram types

Analysis class diagram

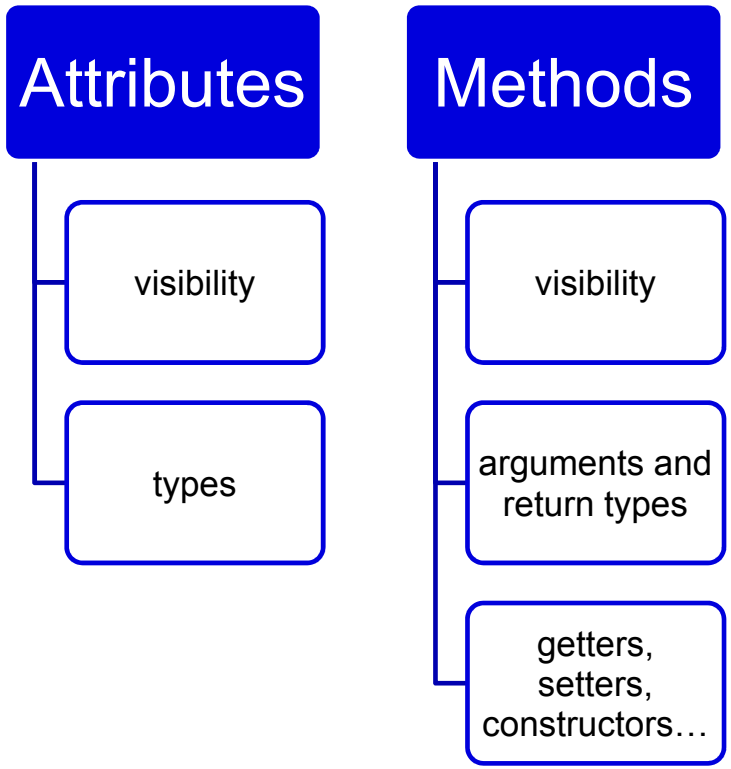
- only **basic classes** representing key system entities, related to requirements
- no implementation details
- helps to **identify key** domain **elements**
- should remain readable and easy to understand
- limited number of classes, attributes, methods…

Design class diagram

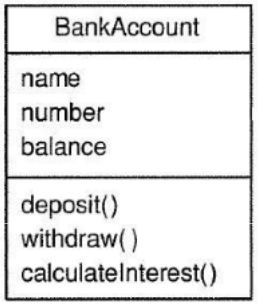
- **all classes**
- adds implementation details
- helps to **program** the system
- language specific constructs
- can have exteremely large number of classes (attributes, methods…)

MUNI FI

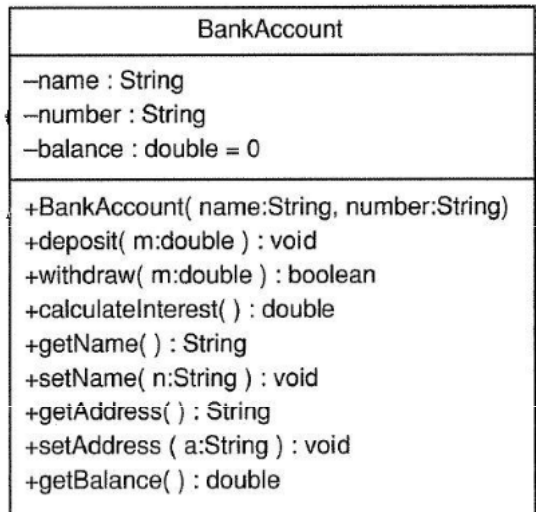
Implementation details



analysis



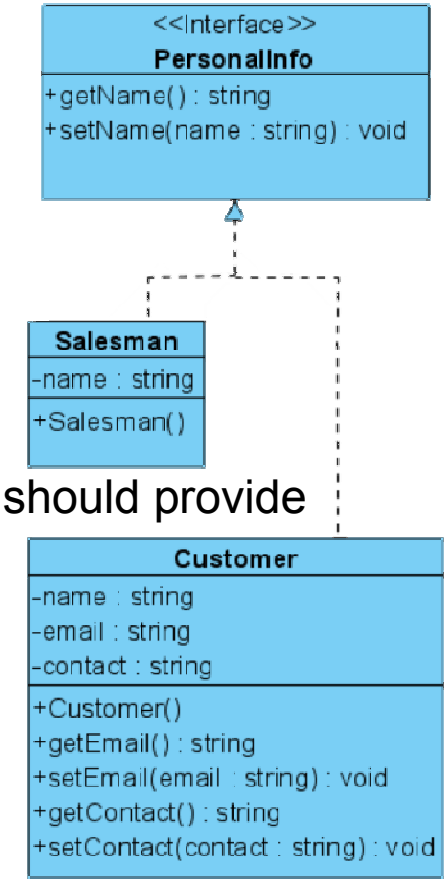
design



MUNI FI

Interfaces

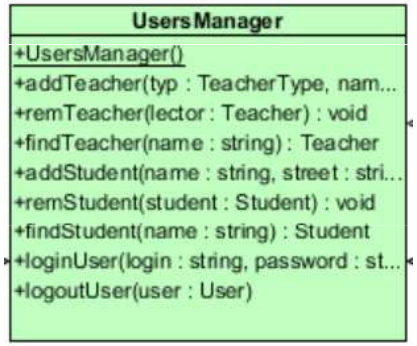
- Define public methods, attributes and relationships
- Without implementation*
  - Class implementing the interface must implement defined methods
- Often used to implement generalization
  - If we do not need to inherit implementation but only define what classes should provide



MUNI FI

Manger classes

- Classes to manage objects
  - to provide basic (CRUD) functionality
  - to provide access to objects for classes that do not have links to them
  - to provide methods users can call from GUI
  - single instance
- Simplification for this course
  - in practice replaced with classes dependent on specific system architecture



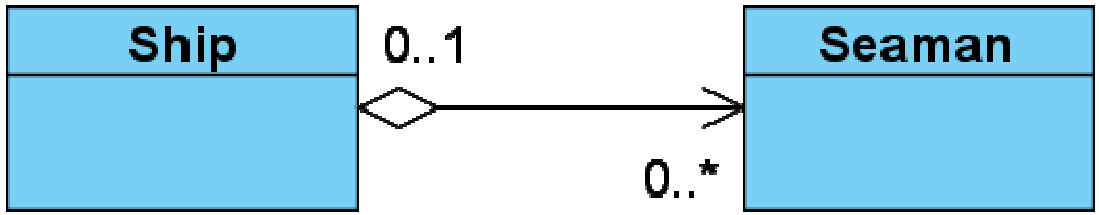
MUNI FI

Association specification

- Specify chosen associations
  - aggregation vs composition
  - only when it makes sense
  - sometimes not easy to decide
- Decompose bidirectional associations to one-way
- Decompose association classes
- Decompose M:N associations

Aggregation

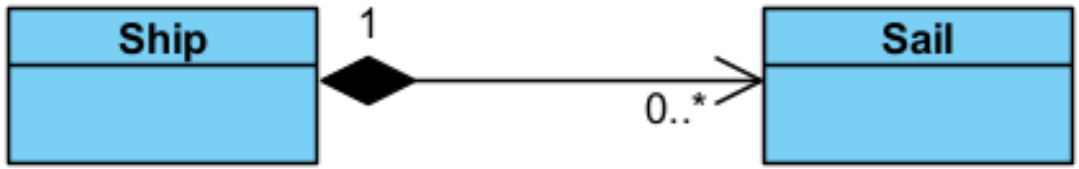
- Aggregation is a whole-part type of relationship.
  - The whole may or may not exist without its parts
  - Parts can usually exist independently from the whole
  - The whole is in a sense incomplete if some parts are missing
  - Part can be shared by multiple whole classes
  - Aggregation is transitive and asymmetrical (without cycles)



MUNI FI

Composition

- Composition is a stronger form of aggregation
  - The part belongs to exactly one whole in the given time
  - The whole is responsible for the lifecycle of its parts
  - The part cannot exist without the whole
  - When deleting, the whole must take care of its parts
    delete or transfer them to another whole
  - Composition is transitive and asymmetrical (without cycles)



MUNI FI

Comparison

Aggregation

```
public class Ship
{
    private Engine _engine;

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

Composition

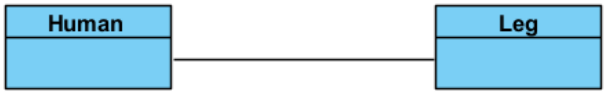
```
public class Ship
{
    private Engine _engine;

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

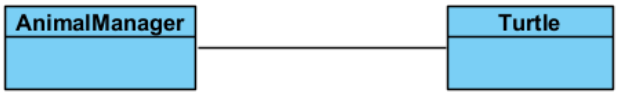
Activity: Association or composition?

Add multiplicity and association type

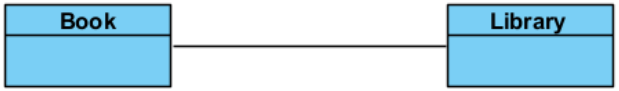
1



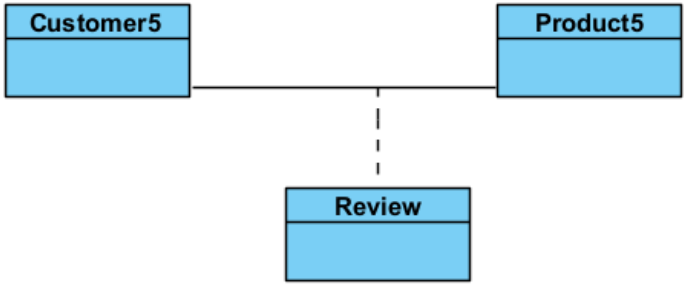
2



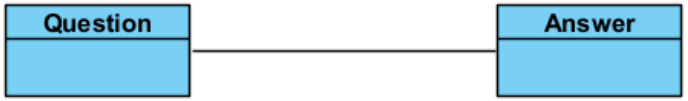
3



4



5

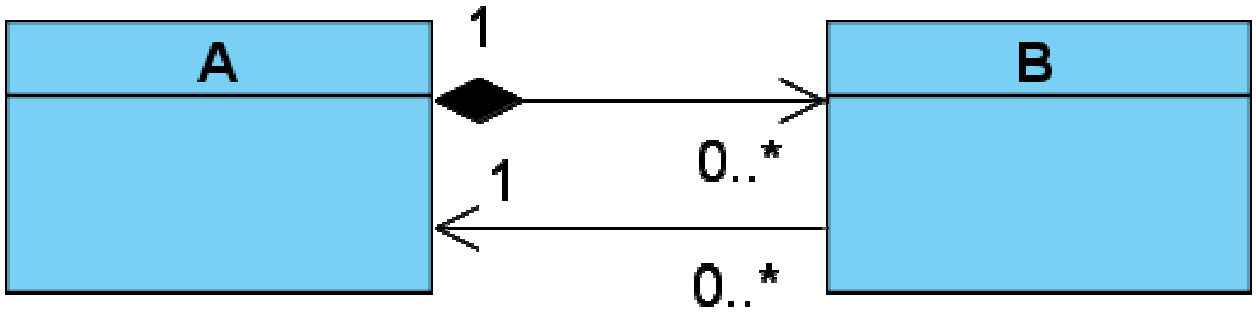


MUNI FI

Association decomposition

Bidirectional

– Someone must "own" it

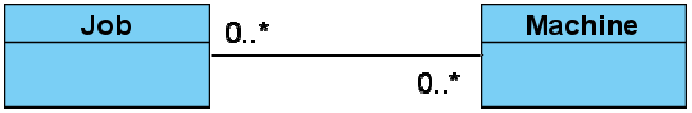


MUNI
FI

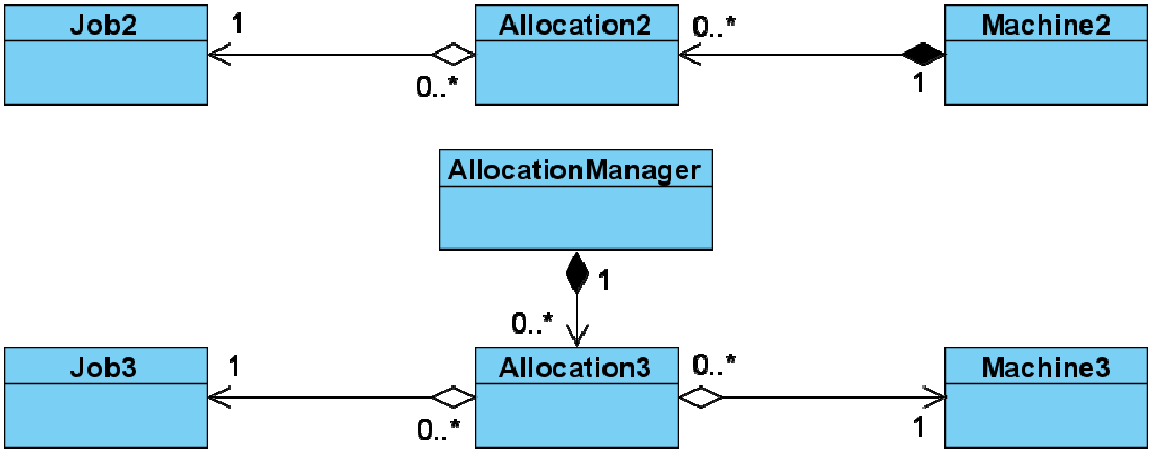
Association decomposition

M:N

– Analysis



– Design
  – decompose if there is
    a need for additional attributes
  – someone must "own" it

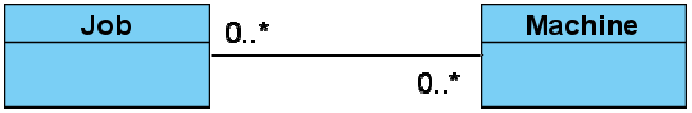


MUNI FI

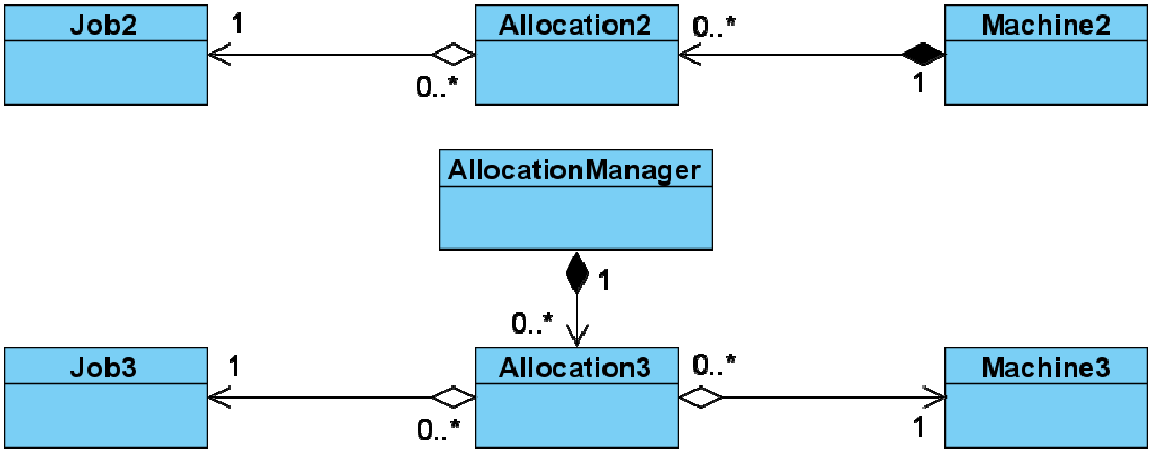
Association decomposition

M:N

- Analysis



- **Design**
  - decompose if there is
    a need for additional attributes
  - someone must "own" it



MUNI
FI

Task for this week

- Review the **entity-relationship diagram** from the previous session. Fix any problem.
- Create Design Class diagram
- Submit this week‘s report in homework vault week08 in format **surname1-surname2-surname3.pdf**