Design class diagram

PB007 Software engineering I

Marián Macik
originally by Stanislav Chren

Week 08

Design class diagram

Class diagram represents a static view of classes, their attributes, operations and relationships.

Analytical class diagram

- Models the business domain of the system - focus on main concpets and relationships
- Attempts to maintain clarity and simplicity without the implementation details

Design class diagram

- Extends the analytical class diagram with implementation classes and details

Design classes

Design class provides such a level of abstraction so that it can be easily implemented

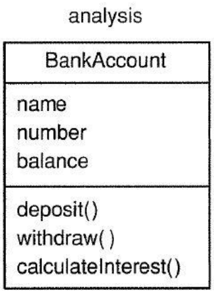
Design classes can originate from:

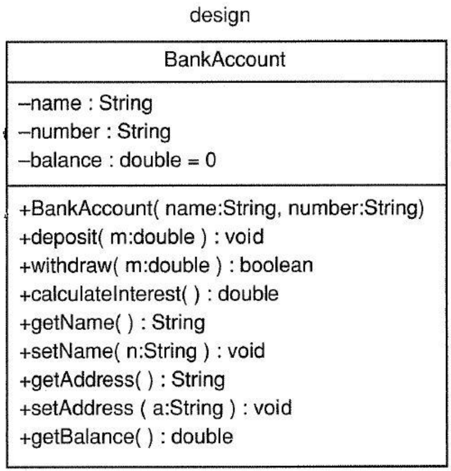
- Business domain - more detailed specification of analytical classes (decomposition, inclusion of implementation details) .
- Solution domain - technology-related classes (classes for working with GUI, DB, ...)

Implementation details include:

- Visibility and types of attributes.
- Visibility, arguments and return types of methods.
- Methods decomposed from analytical operations, constructors (destructors), getter/setter methods, implementation methods.

Design class - Example




design
BankAccount
–name : String
–number : String
–balance : double = 0
+BankAccount(name:String, number:String)
+deposit(m:double) : void
+withdraw(m:double) : boolean
+calculateInterest( ) : double
+getName( ) : String
+setName(n:String) : void
+getAddress( ) : String
+setAddress (a:String) : void
+getBalance( ) : double

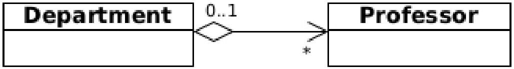
Revision of analytical associations

- Specification of aggregation/composition association types.
- Definition of names, navigability and multiplicities.
- Decomposition of bidirectional associations.
- Revision of 1:1, 1:M and M:1 associations.
- Decomposition of M:N associations.
- Decomposition of association classes.

Aggregation

Aggregation is a *whole-part* type of relationship.

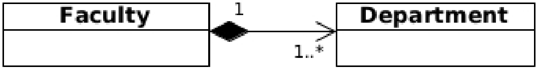
- The *whole* usually may or may not exist without its parts.
- *Parts* can usually exist independently from the *whole*.
- The *whole* is in a sense incomplete if some parts are missing.
- *Part* can be in theory shared by multiple *whole* classes.
- Aggregation is transitive and asymmetrical (without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



Composition

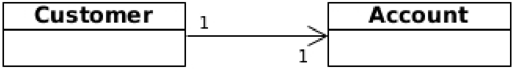
Composition is a stronger form of aggregation

- At any given time, *parts* can belong to exactly one *whole*.
- The *whole* is usually responsible for managment of its *parts*.
- If the *whole* is deleted, it has to either delete its *parts* or the *parts* have to be associated with another *whole*.
- Composition is transitive and asymmetrical (without cycles).

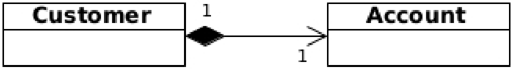


Revision of 1:1 associations

Analysis:

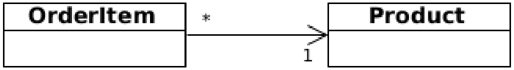


Design:

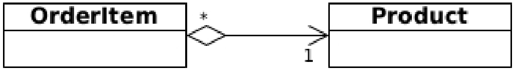


Revision of M:1 associations

Analysis:

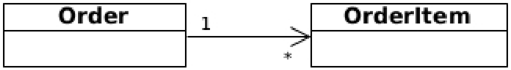


Design:

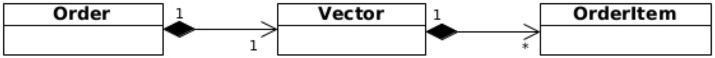


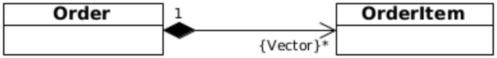
Revision of 1:M associations

Analysis:



Design:

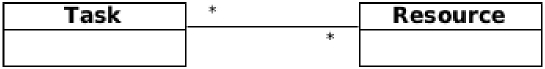




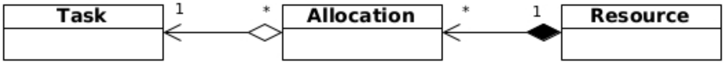
{unique, nonunique, ordered, unordered}
{Vector},...

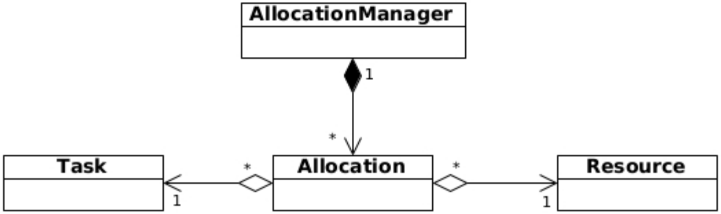
Decomposition of M:N associations

Analysis:



Design:

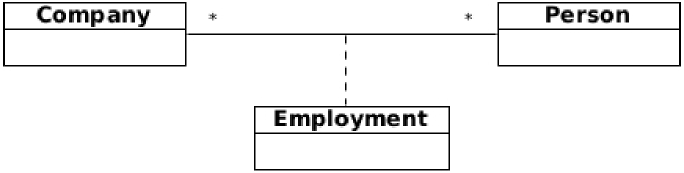




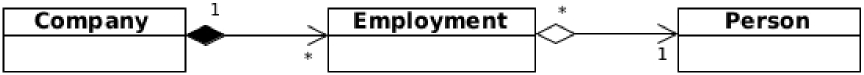
Note.: This decomposition is suitable only in cases when the allocation class has additional attributes. Otherwise, the M:N association does not have to be decomposed.

Decomposition of association classes

Analysis:

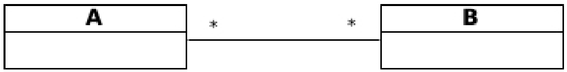


Design:

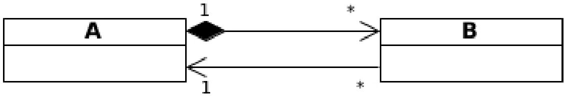


Decomposition of bidirectional associations

Analysis:

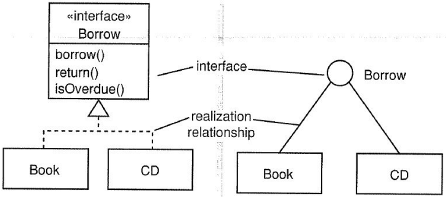


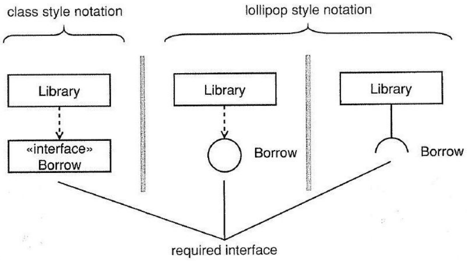
Design:



Interfaces

Interface is a special element that defines a set of public services, attributes and relationships but it does not implement them. They are used to define the contract for implementing classes.





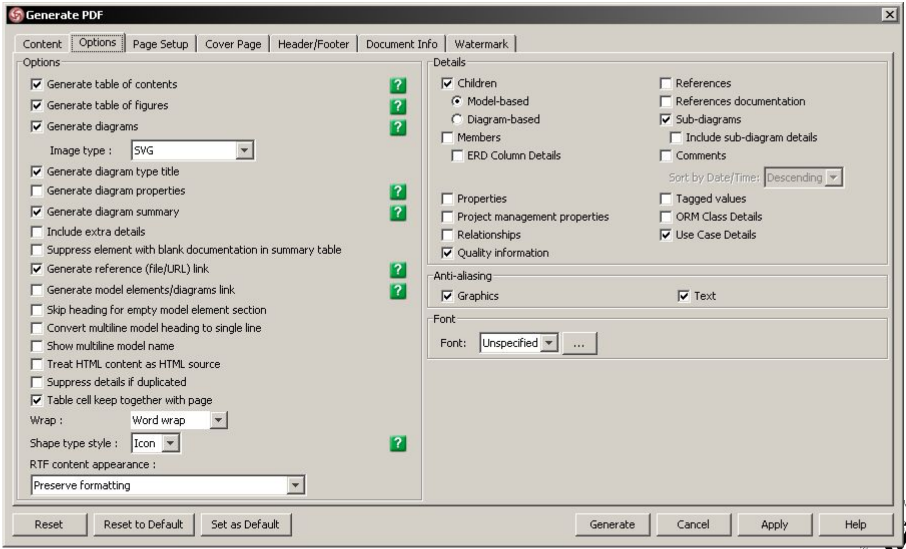
Tasks

- Extend a copy of analytical class diagram into design class diagram.
- Specify visibility and type of all attributes.
- Add methods that originated from decomposition of analytical operations, implementation and helper methods (constructors, getters/setters, ...) and determine their visibility, arguments and return types.
  The getters/setters should be added only if it is necessary.
- Further specify the associations (names, multipicity navigability, modifiesrs, determine the aggregation/composition and decompose the association classes).
- Add dependency relationships.
- If necessary, add other implmentation classes, enums and interfaces.
- Generate a **PDF report** and upload it to the homework vault (**Week 08**).

Rules for report submission

1. Submit the PDF report, not the VP source file and not an exported image.
2. PDF report must be created using the procedure shown on the seminars including the report settings.
3. The name of the PDF report file should be *lastname1-lastname2-lastname3* of the team members.
4. PDF report must contain all diagrams modelled until now.
5. PDF report must be uploaded to the homework vault by the specified deadline.
6. PDF report must be uploaded to the correct homework vault. The name of the homework vault is always specified on the slides.
7. Each team uploads only a single PDF report for the whole team.
8. Submitted diagrams must be clear and readable.
9. Submitted diagrams should not contain serious mistakes. At least, they should not contain mistakes mentioned in the *Catalogue of common mistakes*.

VP report settings



Generate PDF

Content | Options | Page Setup | Cover Page | Header/Footer | Document Info | Watermark

Options

- [✓] Generate table of contents
- [✓] Generate table of figures
- [✓] Generate diagrams
- Image type : SVG
- [✓] Generate diagram type title
- [ ] Generate diagram properties
- [✓] Generate diagram summary
- [ ] Include extra details
- [ ] Suppress element with blank documentation in summary table
- [✓] Generate reference (file/URL) link
- [ ] Generate model elements/diagrams link
- [ ] Skip heading for empty model element section
- [ ] Convert multiline model heading to single line
- [ ] Show multiline model name
- [ ] Treat HTML content as HTML source
- [ ] Suppress details if duplicated
- [✓] Table cell keep together with page
- Wrap : Word wrap
- Shape type style : Icon
- RTF content appearance : Preserve formatting

Details

- [✓] Children
  - ● Model-based
  - ○ Diagram-based
- [ ] Members
  - [ ] ERD Column Details
- [ ] Properties
- [ ] Project management properties
- [ ] Relationships
- [✓] Quality information
- [ ] References
- [ ] References documentation
- [✓] Sub-diagrams
  - [ ] Include sub-diagram details
- [ ] Comments
  - Sort by Date/Time: Descending
- [ ] Tagged values
- [ ] ORM Class Details
- [✓] Use Case Details

Anti-aliasing

- [✓] Graphics
- [✓] Text

Font

Font: Unspecified ...