Software Architecture and Design I
SOEN 343

Instructor: Dr. Rodrigo Morales
https://moar82.github.io/
rodrigo.moralesalvarado@concordia.ca

Lecture 2b: Domain Modeling: UML Class Diagrams

Learning objectives (1)

- Introduction to the Unified Process
- Understand the role of the domain model in requirements engineering
- Understand the connections between the domain model, the design class diagram, and the implementation classes
- Learn strategies for obtaining domain models from use case descriptions, including noun phrase
- Identification and conceptual class category lists

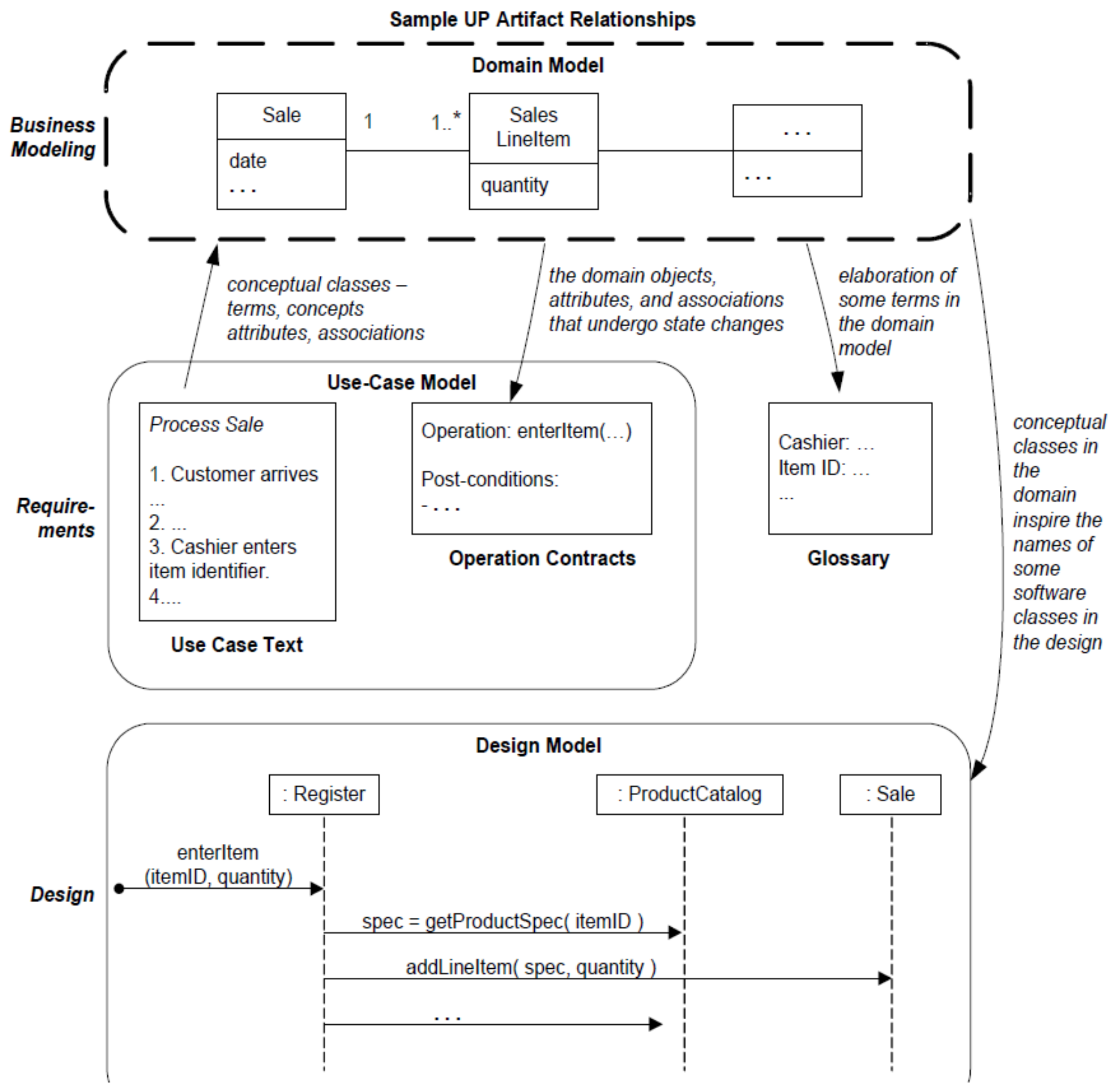
Learning objectives (2)

- Learn how to represent domain models with UML class diagrams
- Learn the use of classes and attributes
- Understand how to model associations between entities
- Understand when and how to use description classes
- Understand how to find associations, name them, and specify their multiplicity
- Learn the difference between aggregation and composition
- Learn how to do quality assurance on a domain model

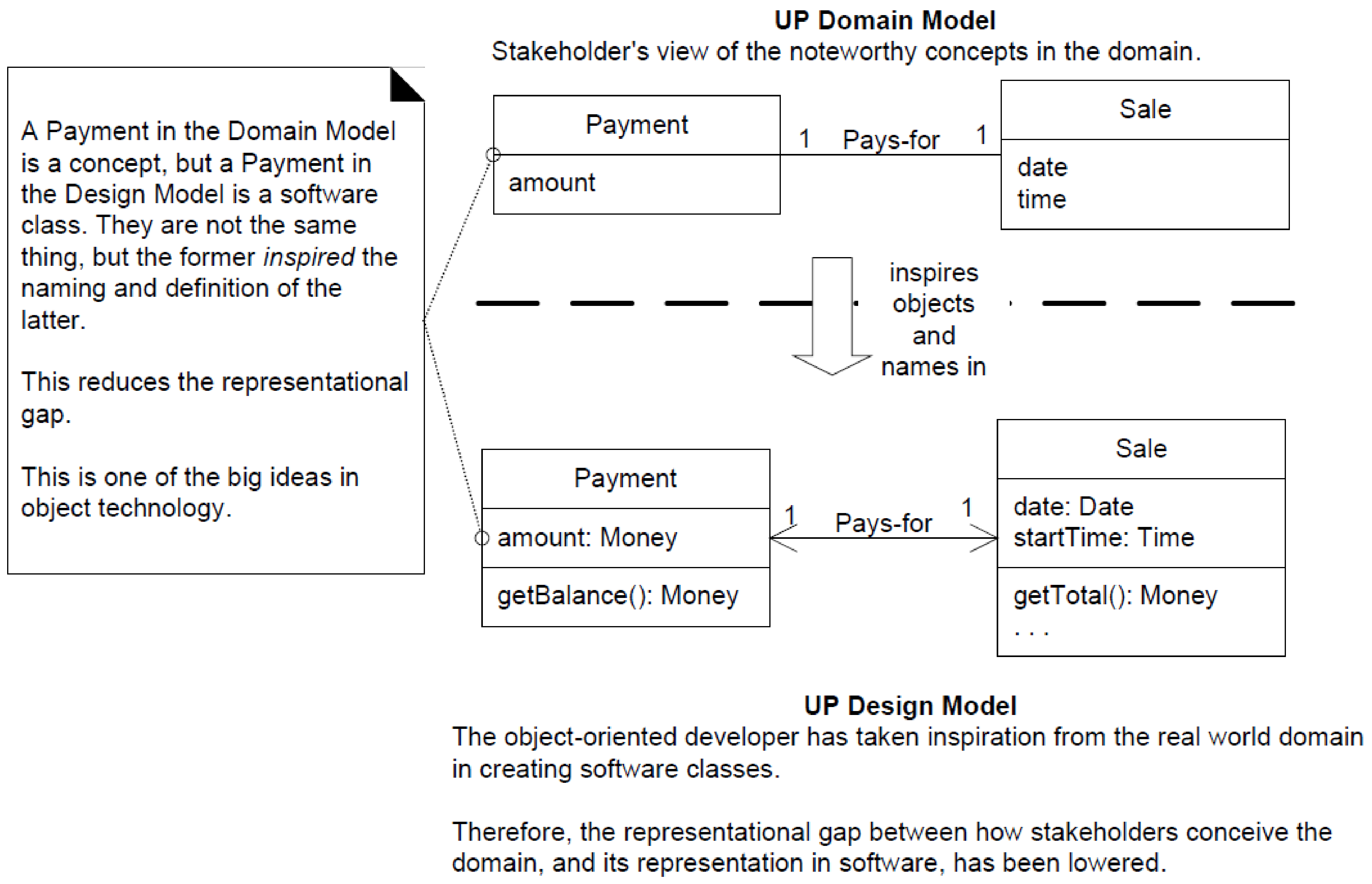
Software development process

- A software development process describes an approach to building, deploying and maintaining software
- The unified process (UP) has emerged as a popular iterative software development process for building OO systems
- UP encourages skillful practices from other iterative methods such as extreme programming (XP), scrum and so forth

Domain Model in the UP


Sample UP Artifact Relationships
Domain Model
Business Modeling
Sale
date
. . .
1
1..*
Sales LineItem
quantity
. . .
. . .
conceptual classes – terms, concepts attributes, associations
the domain objects, attributes, and associations that undergo state changes
elaboration of some terms in the domain model
conceptual classes in the domain inspire the names of some software classes in the design
Use-Case Model
Require-ments
Process Sale
1. Customer arrives ...
2. ...
3. Cashier enters item identifier.
4....
Use Case Text
Operation: enterItem(…)
Post-conditions:
- . . .
Operation Contracts
Cashier: …
Item ID: …
...
Glossary
Design Model
Design
: Register
: ProductCatalog
: Sale
enterItem (itemID, quantity)
spec = getProductSpec(itemID)
addLineItem(spec, quantity)
. . .

Lower representational gap with OO



A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and definition of the latter.

This reduces the representational gap.

This is one of the big ideas in object technology.

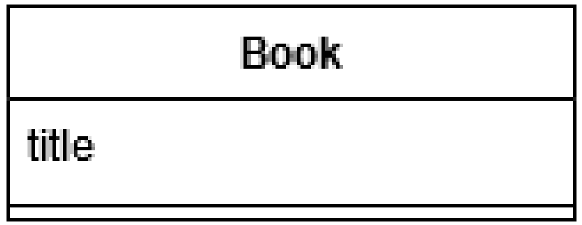
The object-oriented developer has taken inspiration from the real world domain in creating software classes.

Therefore, the representational gap between how stakeholders conceive the domain, and its representation in software, has been lowered.

Domain Model

<u>From Glossary to Domain Model</u>

Domain concept

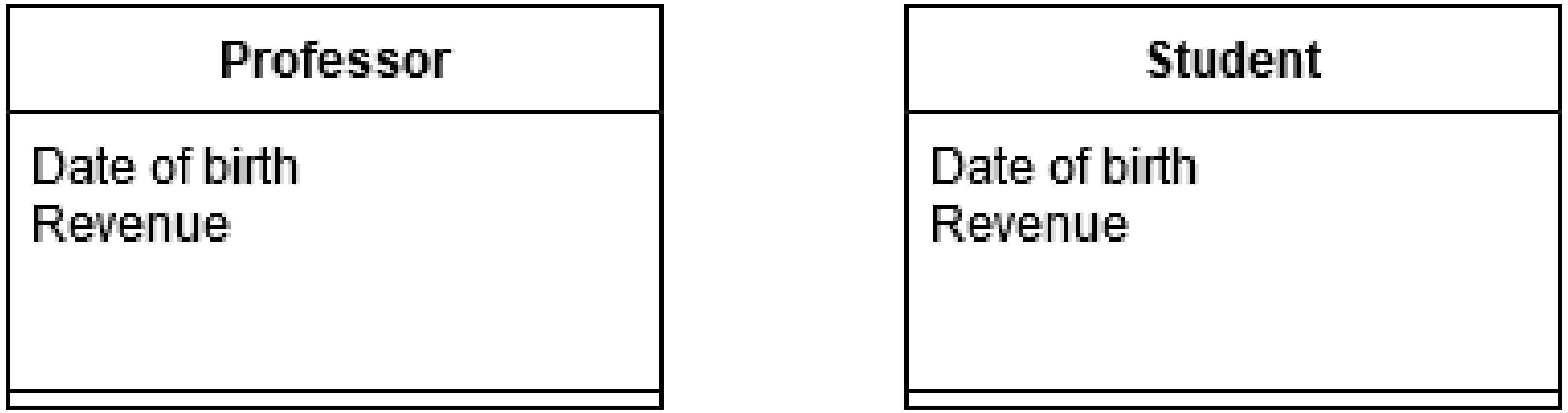


Visualization of domain concept

Domain model

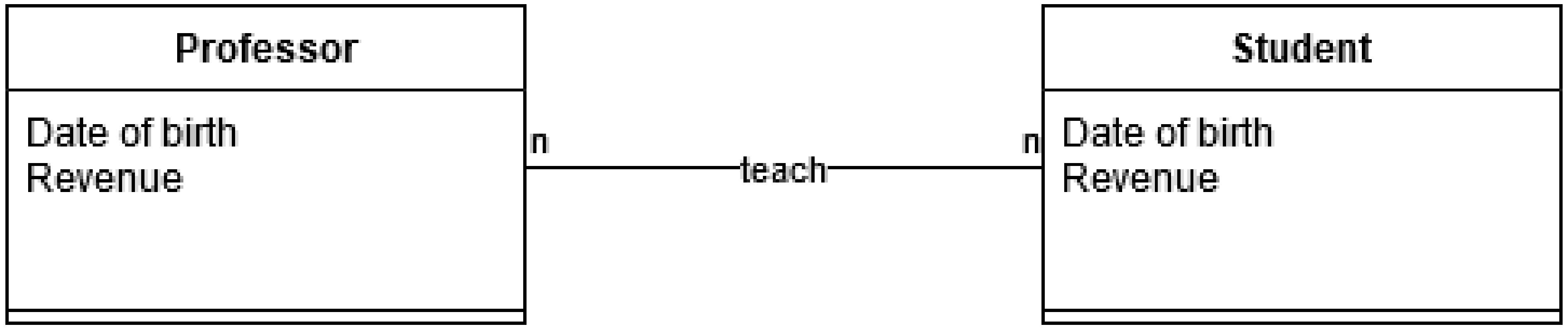
Adding Attributes

- In the domain model, attributes should be simple attributes or data types
- Common types include: boolean, data, number, string, address, color, etc.



Adding associations

- Links and relationships between concepts



Strategies to Identify Conceptual Classes

Two basic approaches

- Use noun phrase (NP) identification.
  - Identify nouns (and noun phrases) in textual descriptions of the problem domain and consider them as concepts or attributes.
  - Use Cases are excellent descriptions to draw for this analysis.
- Use a conceptual class category list
  - Make a list of candidate concepts.

Recognize noun phrases

<u>Example checkout use case</u>

- *Ask the user to input credit card information, send the credit card information to financial department for verification. If the transaction is approved, the credit card is charged; otherwise, prompts the user that the check out process has failed. When the transaction is successful, instructions are sent to shipping department."*

Approach: Recognize noun phrases

Example checkout use case

- *Ask the user to input credit card information, send the credit card information to financial department for verification. If the transaction is approved, the credit card is charged; otherwise, prompts the user that the check out process has failed. When the transaction is successful, instructions are sent to shipping department."*

Approach: Recognize noun phrases

Example checkout use case

- *Ask the user to input credit card information, send the credit card information to financial department for verification. If the transaction is approved, the credit card is charged; otherwise, prompts the user that the check out process has failed. When the transaction is successful, instructions are sent to shipping department."*

Actors do not need to be represented as a class

Approach: Recognize noun phrases

Example checkout use case

- *Ask the user to input credit card information, send the credit card information to financial department for verification. If the transaction is approved, the credit card is charged; otherwise, prompts the user that the check out process has failed. When the transaction is successful, instructions are sent to shipping department."*

It can be modeled as a class

Approach: Recognize noun phrases

Example checkout use case

- *Ask the user to input credit card information, send the credit card information to financial department for verification. If the transaction is approved, the credit card is charged; otherwise, prompts the user that the check out process has failed. When the transaction is successful, instructions are sent to shipping department."*

Actors do not need to be represented as a class

Approach: Recognize noun phrases

Example checkout use case

- *Ask the user to input credit card information, send the credit card information to financial department for verification. If the transaction is approved, the credit card is charged; otherwise, prompts the user that the check out process has failed. When the transaction is successful, instructions are sent to shipping department."*

It can be modeled as a *controller* class

Approach: Recognize noun phrases

Example checkout use case

- *Ask the user to input credit card information, send the credit card information to financial department for verification. If the transaction is approved, the credit card is charged; otherwise, prompts the user that the check out process has failed. When the transaction is successful, instructions are sent to shipping department."*

It can be modeled as an *entity* class

Approach: Recognize noun phrases

Example checkout use case

- *Ask the user to input credit card information, send the credit card information to financial department for verification. If the transaction is approved, the credit card is charged; otherwise, prompts the user that the check out process has failed. When the transaction is successful, instructions are sent to shipping department."*

It can be modeled as a *data entity* class

Approach: Recognize noun phrases

Example checkout use case

- *Ask the user to input credit card information, send the credit card information to financial department for verification. If the transaction is approved, the credit card is charged; otherwise, prompts the user that the check out process has failed. When the transaction is successful, instructions are sent to shipping department."*

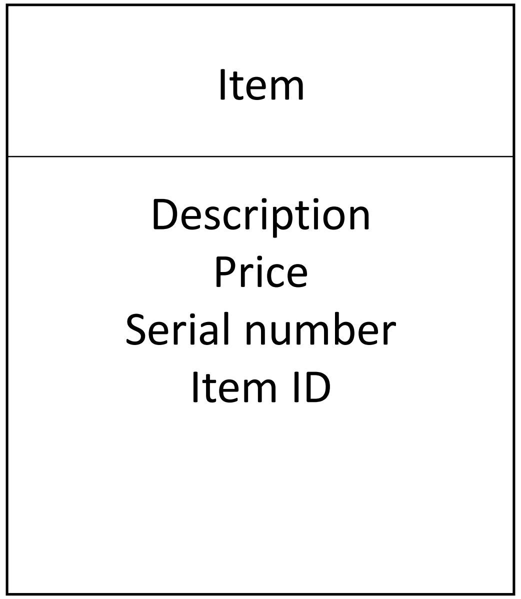
Actors do not need to be represented as a class

Use a Conceptual Class Category List

Concept category	Example
Physical or tangible objects	POS
Specifications, designs or descriptions of things	ProductSpecification
Places	Store
Transactions	Sale, Payment
Transaction line items	SalesLineItem
Roles of people	Cashier
Containers of other things	Store, Bin

Description classes

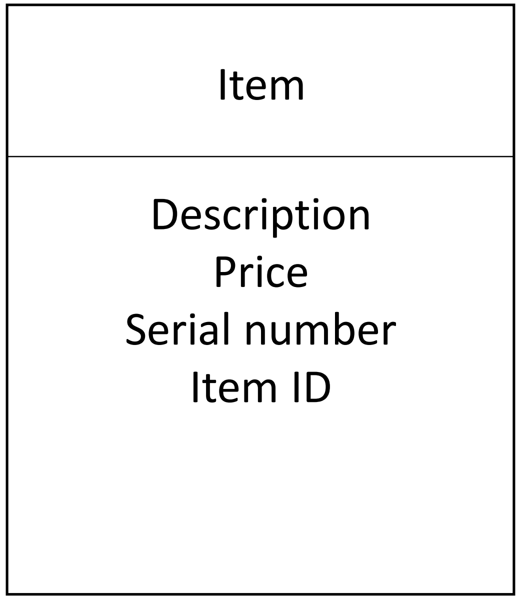
- What is wrong with this class?



Description classes II

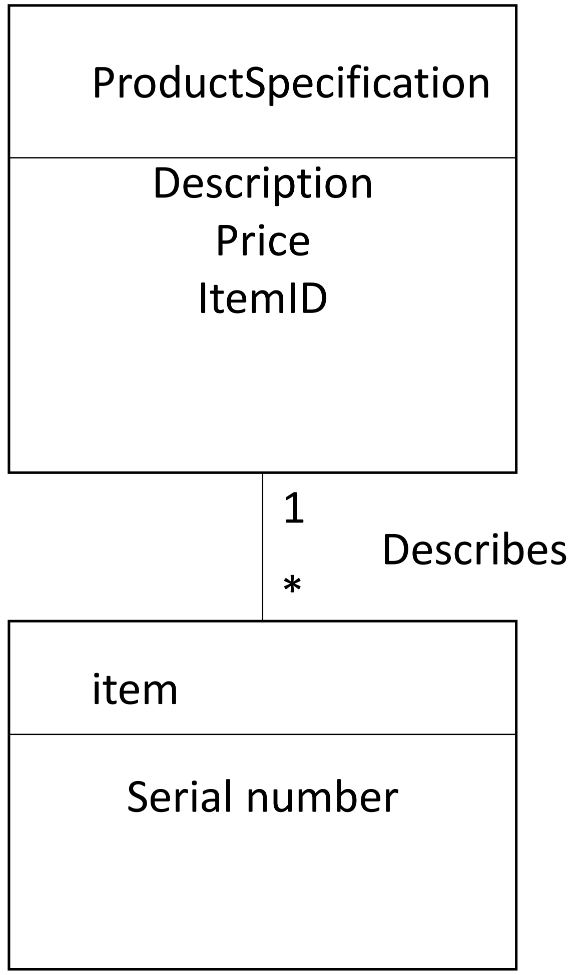
- What is wrong with this class?

- *The memory of the item's price was attached to inventoried instances, which were deleted*
- *Notice also that in this model there is duplicated data (description, price, itemID)*



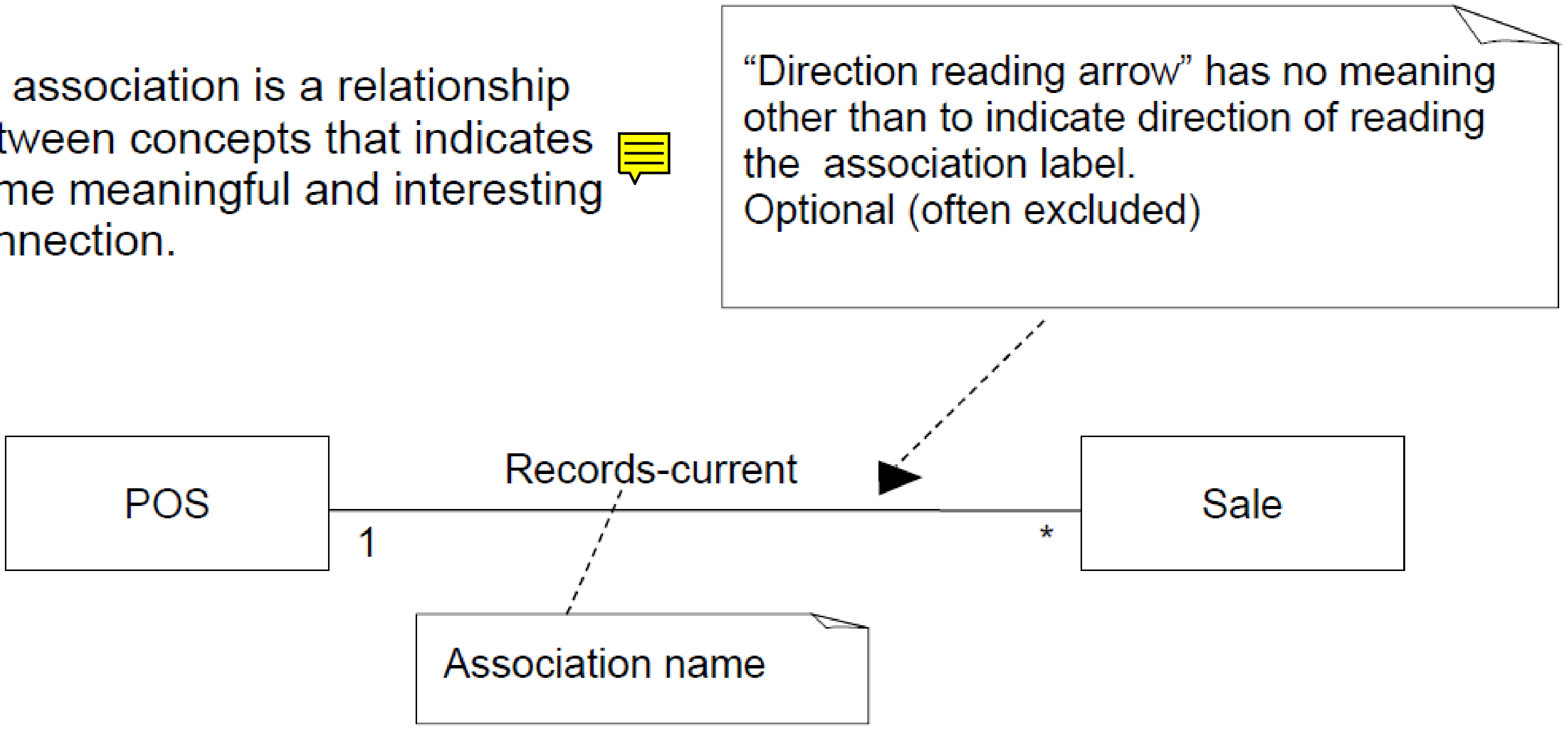
Description classes III

- Add a specification or description concept when:
  - Deleting instances of things they describe results in a loss of information that needs to be maintained, due to the incorrect association of information with the deleted thing
  - It reduces redundant or duplicated information



Adding associations

An association is a relationship between concepts that indicates some meaningful and interesting connection.


"Direction reading arrow" has no meaning other than to indicate direction of reading the association label.
Optional (often excluded)
POS
Records-current
1
*
Sale
Association name

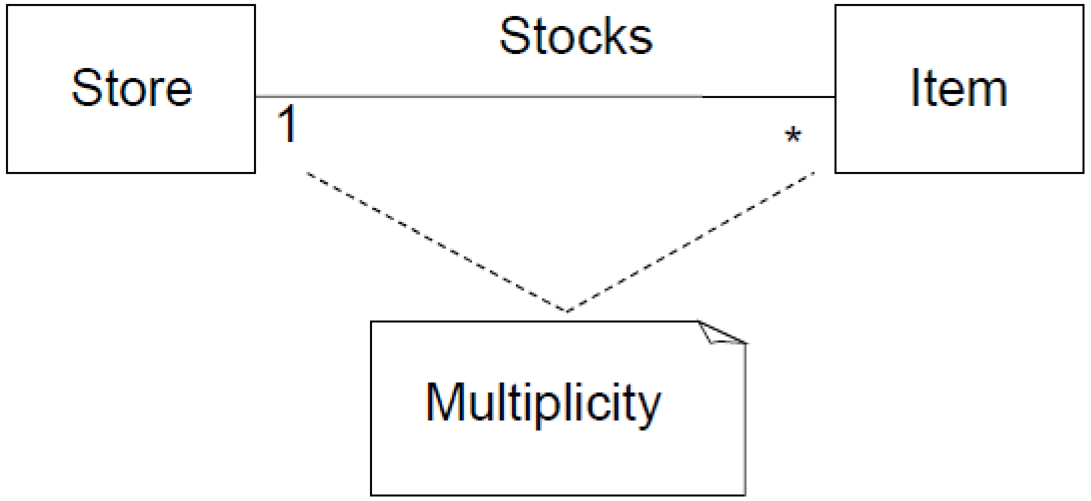
Finding associations

List of common associations

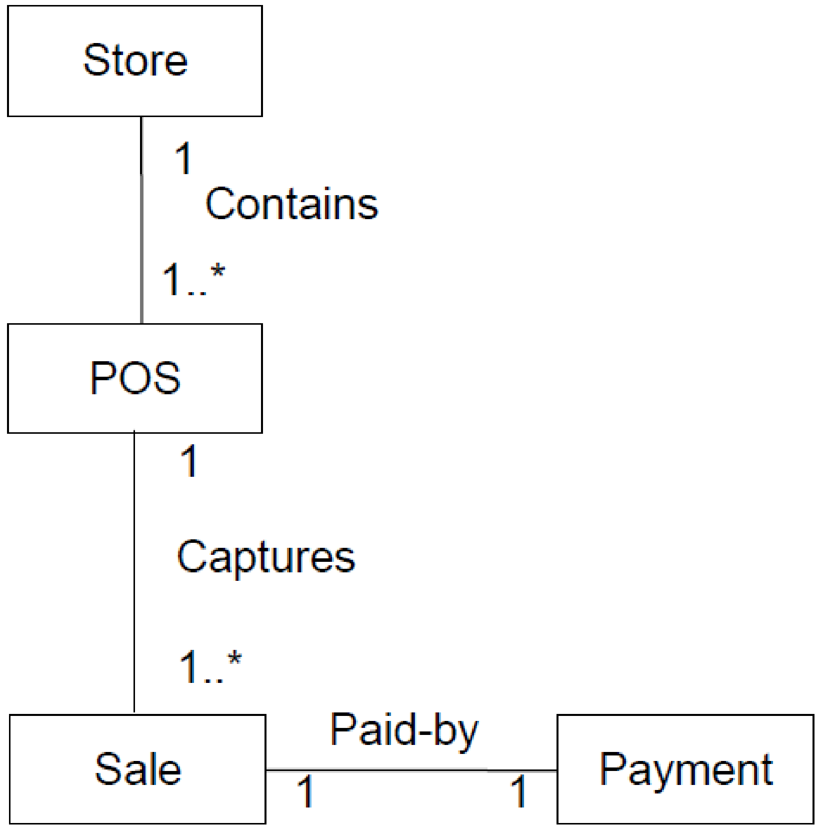
Category	Examples
A is a physical part of B	Drawer - POS
A is a logical part of B	SalesLineItem - Sale
A is physically contained in/on B	POS - Store
A is logically contained in B	ItemDescription - Catalog
A is a description of B	ItemDescription - Item
A is a line item of a transaction or report B	SalesLineItem - Sale
A is known/logged/recorded/ captured in B	Sale – POS
A is a member of B	Cashier - Store

Multiplicity

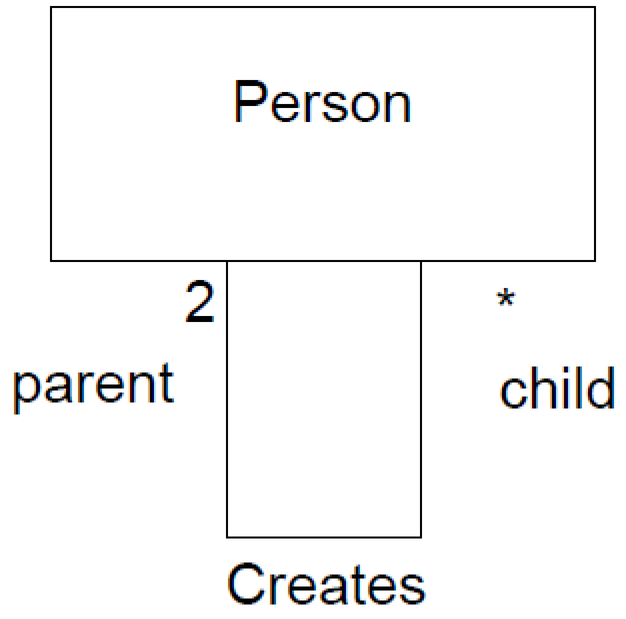
- Multiplicity defines how many instances of a type A can be associated with one instance of a type B, at a moment in time
- For example, a single instance of a Store can be associated with "many" (zero or more) Item instances



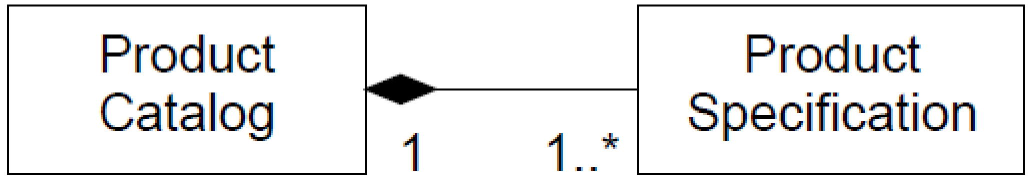
Naming associations



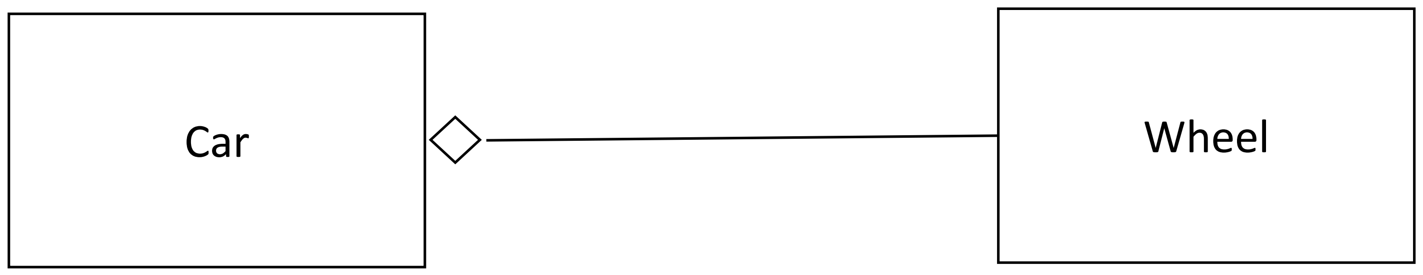
Recursive or reflexive associations



Specialized associations: Composition



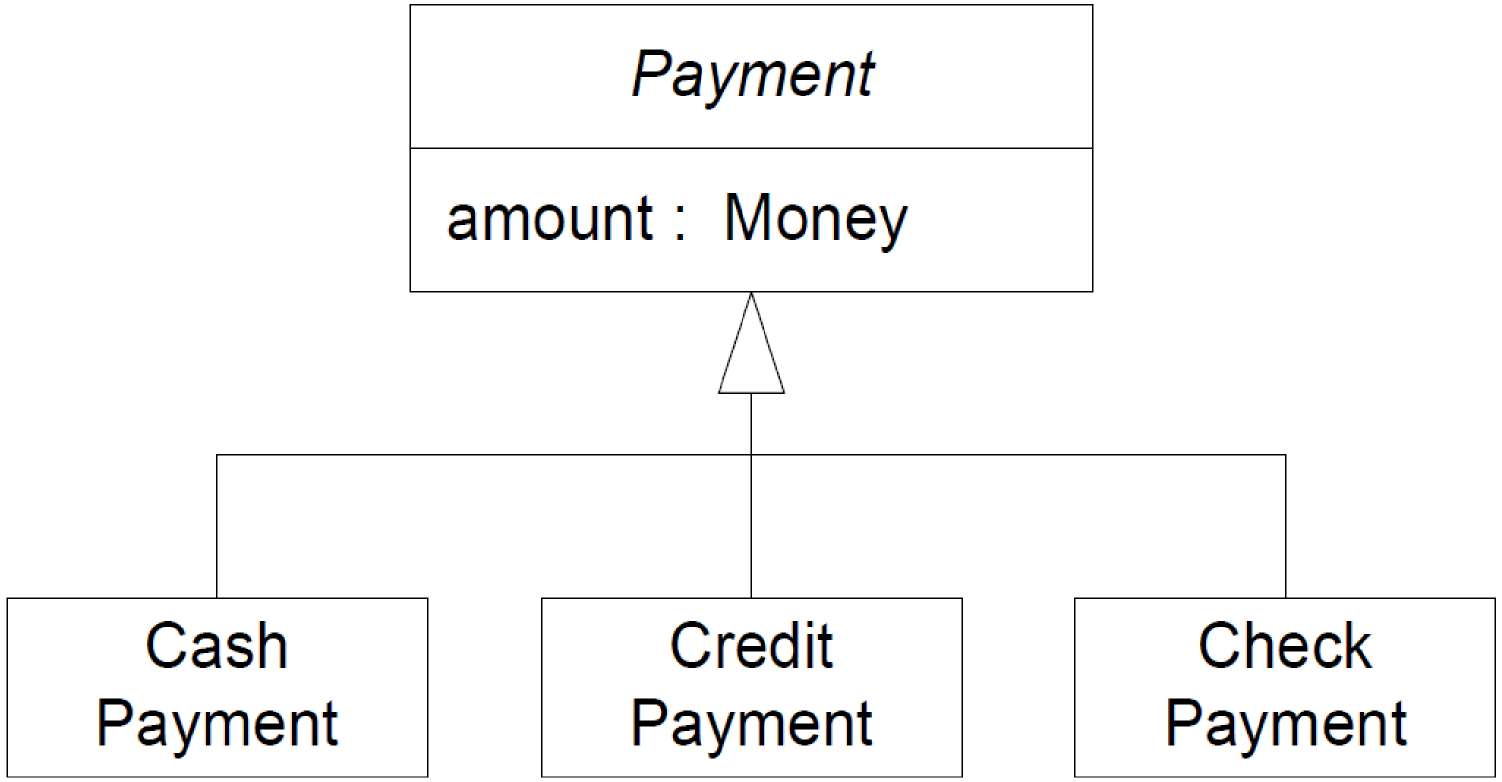
Specialized associations: Aggregation



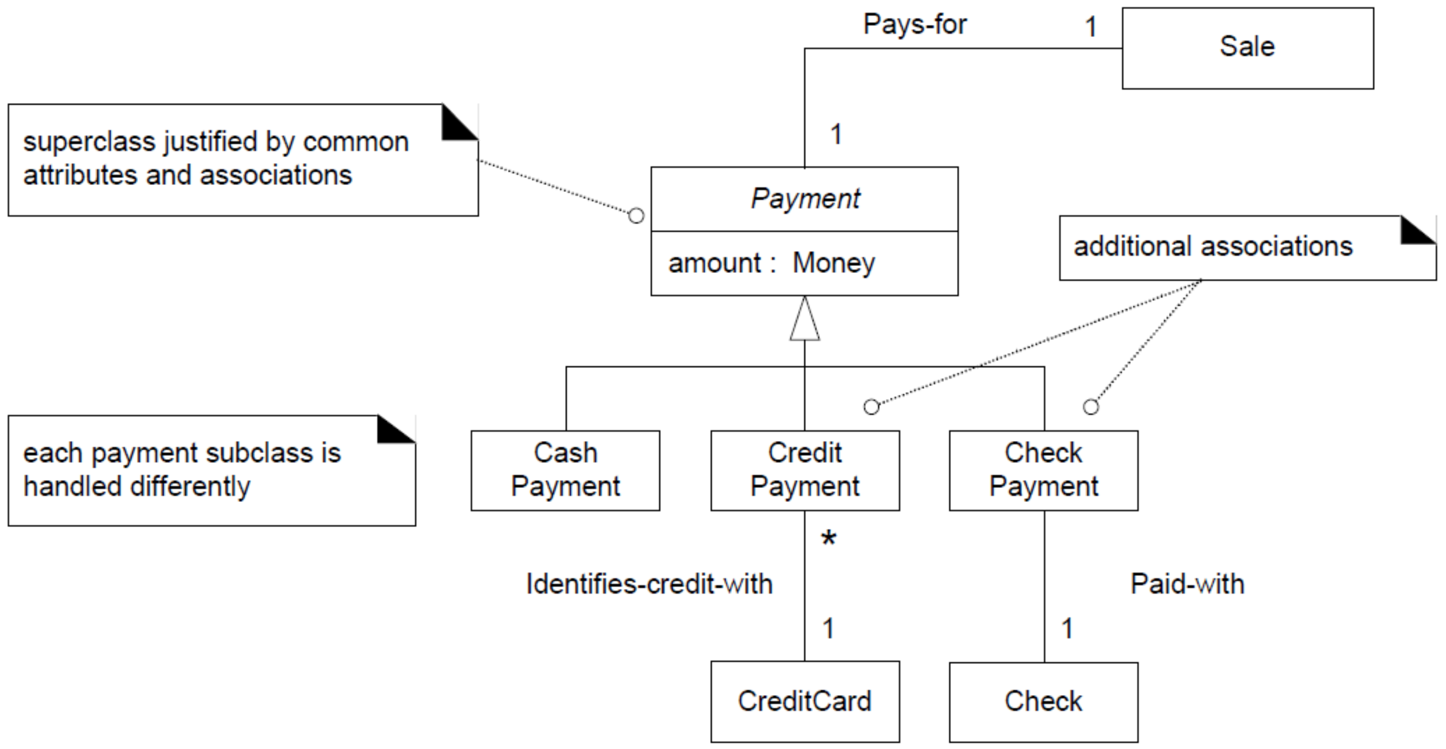
How to Identify Composition

- The lifetime of the part is bound within the lifetime of the composite.
- There is a create-delete dependency of the part on the whole
- There is an obvious whole-part physical or logical assembly
- Some properties of the composite propagate to the parts, such as its location
- Operations applied to the composite propagate to the parts, such as destruction, movement, recording.

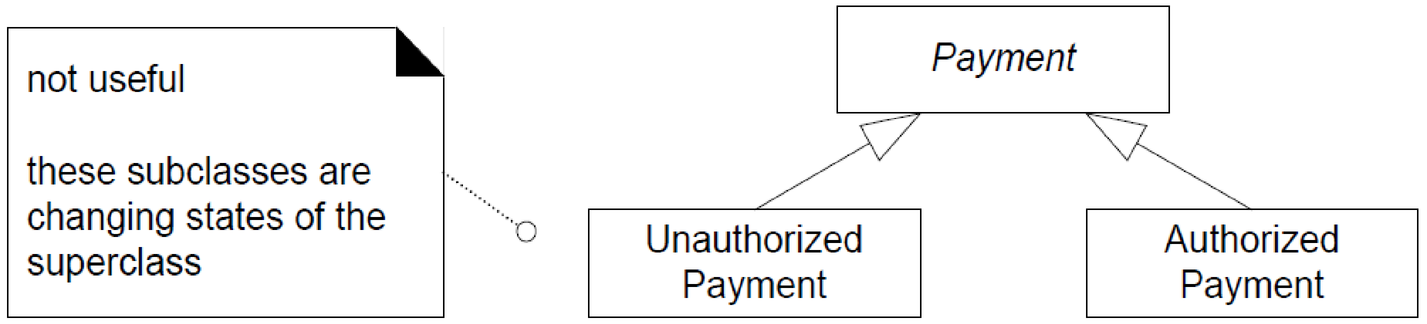
Inheritance



Inheritance II



Inheritance vs States



Building object models best practices

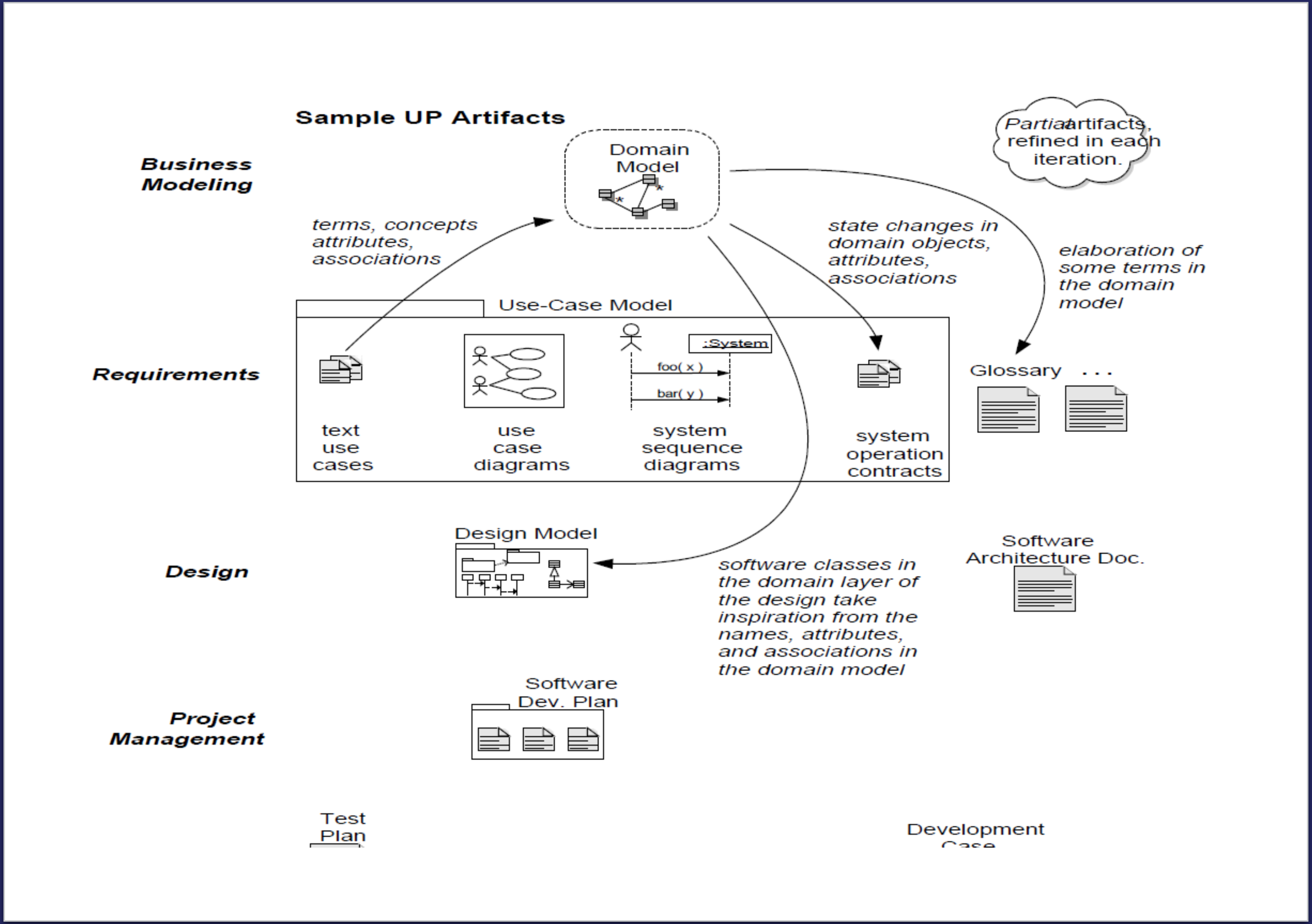
Avoid obscure names for objects & attributes

- Suggestive shortcut of their annotated definition
- From problem world, NOT implementation-oriented
- Specific, NOT vague
- Commonly used, NOT invented

Domain Modeling Summary

- List the candidate real-world concepts
- Draw them in a domain model
- Add the associations necessary to record relationships for which there is a need
- Add the attributes necessary to fulfill the information requirements
- Make sure it is consistent with other artifacts (e.g., use cases)
- Keep traceability in mind when changing/updating artifacts
- Look for existing domain models before developing your own!

Domain Model in Overall Scheme


Sample UP Artifacts
Partial artifacts, refined in each iteration.
Business Modeling
Domain Model
terms, concepts attributes, associations
state changes in domain objects, attributes, associations
elaboration of some terms in the domain model
Use-Case Model
Requirements
:System
foo(x)
bar(y)
text use cases
use case diagrams
system sequence diagrams
system operation contracts
Glossary . . .
Design Model
Design
software classes in the domain layer of the design take inspiration from the names, attributes, and associations in the domain model
Software Architecture Doc.
Software Dev. Plan
Project Management
Test Plan
Development Case

Tool support

There are many UML tools available that can be used for creating a domain model:

- Draw.io
- ArgoUML
- Eclipse-based, e.g., Papyrus
- IBM Rational Architect
- Ms Visio
- Dia
- Lucid chart

Conclusion

- A domain model is a (semi-)formal description of the entities and their relationships within an application domain
- While the use cases and user stories emphasize the dynamic part of the system-to-be, the domain model offers a complementary, static view
- Captured with a suitable representation formalism (e.g., UML class diagrams), it provides for a high-level description of the domain that is both richer and more precise than a simple glossary
- They can be directly used as a basis for creating the design class diagram (DCD), which during implementation is further transformed into the class definitions.

References

1. Chapter 10, Lamsweerde, A V. *Requirements engineering : from system goals to UML models to software specifications*. Chichester, England Hoboken, NJ: John Wiley, 2009. Print. ISBN: 9780470012703
2. Martin Fowler. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley, 3rd edition, 2004.
3. Craig Larman. Applying UML and Patterns. Prentice Hall, third edition, 2005.