Comp435 Object-Oriented Design

Week 5

Computer Science

PSU HBG

1

Overview

- Introduction to Elaboration Phase
- Introduction to Domain Modeling
- Refining the Domain Model
  - Modeling Generalizations
  - Association Classes

2

Inception Phase

- A short requirements workshop
- Most actors, goals, and use cases named
- Most use cases written in brief format
- 10-20% use cases in fully dressed format
- Most influential and risky requirement identified
- ...
- Plan for the first iteration

3

Elaboration Phase

- Initial series of iterations
  - Core, risky software architecture is programmed and tested
  - Majority of requirements are discovered and stabilized
  - Major risks are mitigated and retired

4

Elaboration Phase

- Build core architecture
- Resolve the high-risk elements
- Define most requirements
- Estimate the overall schedule and resources

5

Elaboration Phase

- Artifacts
  - Domain model
  - Design model
  - Software architecture document
  - Data model
  - Use-Case Storyboards, UI Prototypes

6

Domain Model

- Identify important concepts
  - In the problem domain
  - Using object-oriented techniques
- Domain Modeling (Domain analysis)
  - Object-oriented domain modeling in the context of the UP

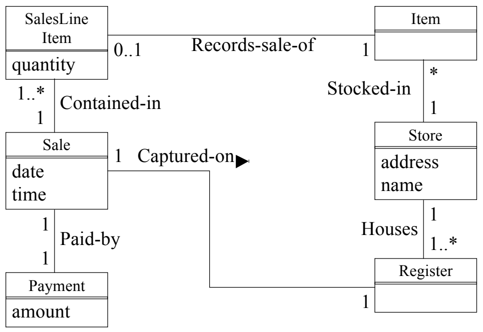
7

Domain Model

- Representation of real world conceptual classes
  - In problem domain
  - Not a representation of software classes
- Represented by UML class diagram
  - Class attributes
  - Associations relationships
  - Generalization relationships
- Identify a rich set of conceptual classes

8

Domain Model



9

UML Diagrams

- UML Diagrams mean different things in different contexts
  - Conceptual perspective
    - Description of things in the problem domain.
  - Specification perspective
    - Description of software abstractions or components
  - Implementation perspective

10

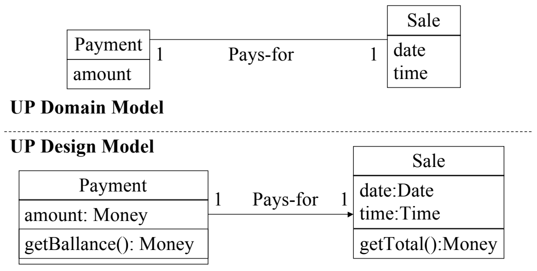
UML Class Diagrams

- For domain analysis
  - Essential perspective
  - Elements: conceptual classes
- For design
  - Specification or implementation perspective
  - Elements
    - design classes
    - implementation classes

11

Focus on the Problem

- Do not represent software artifacts



12

Building the Domain Model

- Over several iterations
  - Driven by the use cases
  - Common to miss conceptual classes in the beginning and add them later
  - Iterative development / refinement

Identifying Conceptual Classes

- Analysis patterns
  - Reuse / modify existing models
- Linguistic analysis
  - Noun phrase identification
- Category list
  - List of candidate conceptual classes

Common Categories of Classes

• Category	• Examples
Physical Objects	Register, Airplane
Places	Store, Airport
Transactions	Sale, Payment
Transaction Line Items	SaleLineItem
Roles of people	Cashier, Manager
Events	Sale, Meeting, Flight
Record	Receipt, Ledger
Specifications	ProductSpecification
Catalogs	ProductCatalog

Example: Process Sale

1. Customer arrives with items
2. Cashier starts a new sale

Possible conceptual classes:
Customer, Cashier, Item, **Sale**

Example (cont)

3. Cashier enters item ID
4. System records sales line item and presents item description, price, and running total
5. Cashier tells Customer the total and asks for payment

Possible conceptual classes:
SalesLineItem, Payment
ProductSpecification
(contains description, price, and itemID)

Example (cont)

6. Cashier enters amount tendered (cash)
7. System presents change due, and releases cash drawer
8. Cashier deposits cash and returns change
9. System presents receipt

Possible conceptual classes:
Register (implied by cash drawer), **Receipt**

Example (cont)

- For completely integrated system
  - May have to define more conceptual classes
  - **Example:**
    - **Store, ProductCatalog, Manager**

19

No "Correct List"

- A collection of concepts that the modeler chooses
- Example:
  - Should **Receipt** be included as a conceptual class?

20

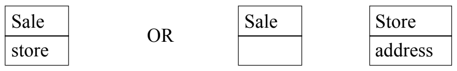
Possible Initial Model

- Just the conceptual classes
- Use existing names in the territory
  - Use vocabulary from the problem domain
- Exclude irrelevant features
  - Ignore conceptual classes irrelevant to the requirements

21

A common mistake

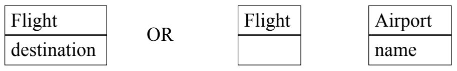
- Often things represented as attributes should be represented as conceptual classes



22

A common mistake

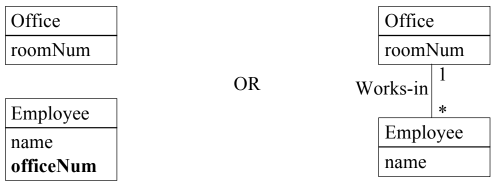
- Another example



23

A Common Mistake

- No foreign keys



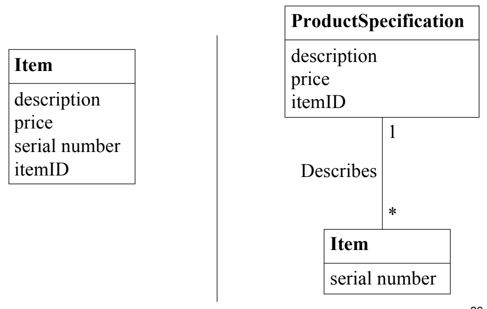
24

Specification Conceptual Classes

- Example
  - class **Item** represents a physical item in the store
    - Each item has a unique serial number
    - All items of the same kind (e.g., XV-S400 DVD player) have the same itemID and price
- We could represent itemID and price as attributes of **Item**. Why not?

25

The two alternatives



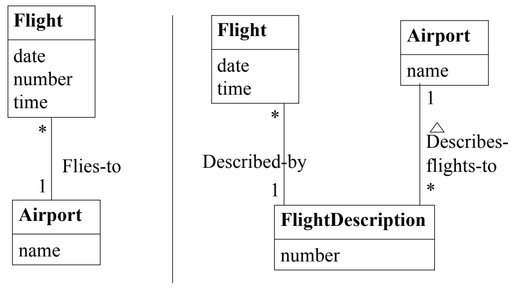
26

When Do We Need This

- When there is need of description of an item, regardless of existence of those items
- When specifications would reduce redundant or duplicate information

27

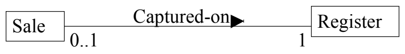
Another Example



28

Domain Model: Adding Associations

- Association
  - Relationship between instances of conceptual classes
  - Relatively permanent relationship



30

Typical Associations

- **A** is a physical/logical part of **B**
  - Wing-Airplane, Finger-Hand SalesLineItem-Sale, FlightLeg-FlightRoute
- **A** is physically/logically contained in **B**
  - Item-Shelf, Flight-FlightSchedule
- **A** is recorded/reported/captured in **B**
  - Sale-Register
- **A** is a description of **B**
  - ProductSpecification-Item

31

Typical Associations

- **A** is a member of **B**
  - Cashier-Store
- **A** uses or manages **B**
  - Cashier-Register, Pilot-Airplane, Manager-Cashier
- **A** is related to a transaction **B**
  - Payment-Sale, Reservation-Cancellation
- **A** is owned by **B**
  - Airplane-Airline

32

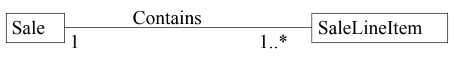
Finding Associations

- Consider the typical categories
- Focus on associations that are relevant with respect to the use cases

33

Examples

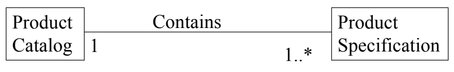
- SaleLineItem-Sale
  - A SaleLineItem is a logical part of the Sale
  - Needed in the context of **ProcessSale** use case



34

Examples

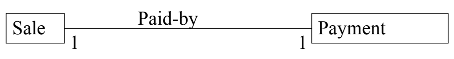
- ProductSpecification-ProductCatalog
  - "Contained-in" relationship



35

Examples

- Payment-Sale
  - Two related transactions



36

Associations

- Roles – each end of association
  - name, multiplicity, and navigability
- Level of detail
  - Emphasize "need-to-know" but add "comprehension-only" associations as well.
- Association names
  - *TypeName---VerbPhrase---TypeName*
  - Readable sequence: Sale---Paid-by---Payment

37

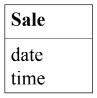
Association and Implementation

- In design and coding:
  - Standard mechanisms to implement associations
- In the domain model
  - An association is conceptual and does not imply that a particular implementation will be used

38

Domain Model: Adding Attributes

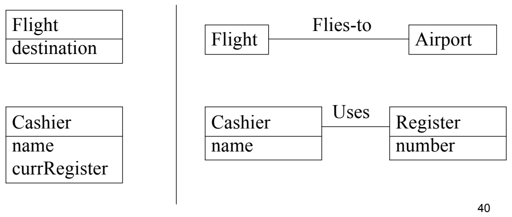
- Attributes that are relevant for the scenarios under consideration
- Example: Process Sale use case
  - Need to remember the date/time of a sale in order to print a receipt and to log the sale
  - Conceptual class **Sale** needs **date** and **time**



39

Attributes vs. Classes

- Attributes should be simple, not complex domain concepts



40

Common Attribute Types

- Primitive types
  - Number, String, Boolean
- Other simple types
  - Date, Time, Name, Address, Color, PhoneNumber, SSN, ZIP, enumeration types, etc…
- Attributes should only be
  - **value objects**,
  - not **reference objects**

41

Attribute Types as Classes

- Simple attribute type may have to be represented as a separate class.

Guidelines:
- Has operations
- Has other attributes
- Quantity with a unit

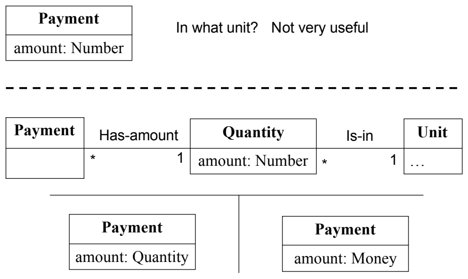
42

Example: Quantities

- Different quantities have units
- According to the guideline, should be represented as conceptual classes with associations
- But since instances are not important, attributes acceptable too.

43

Example: Quantities



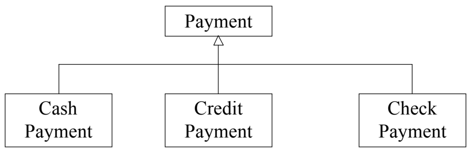
Quantity: Pure data values, so suitable as an attribute
Money: specialized quantity whose unit is a currency

Overview

- Introduction to Elaboration Phase
- Introduction to Domain Modeling
- Refining the Domain Model
  - Modeling Generalizations
  - Association Classes

Generalization

- Superclass-subclass relationships
- Used in the **domain model** and in the **design model**

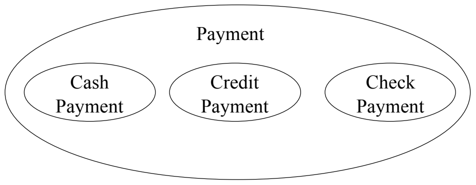


Basic Idea

- Domain model
  - a superclass represents a general concept
  - a subclass represents some specialization
- Design
  - the subclass interface conforms to the interface of the superclass
  - Software components with interfaces
  - The subclass can be used at any place where the superclass is allowed

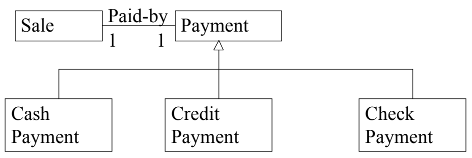
Meaning of Generalization

- **is-a-kind-of**



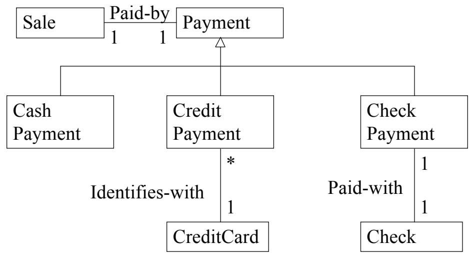
Meaning of Generalization

- All associations and attributes of the superclass apply to the subclass



Additions

- Subclass could add associations/attributes



50

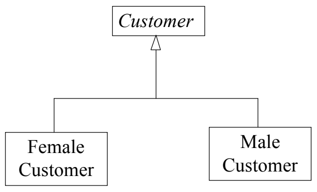
Motivation for subclasses

- The subclass has additional attributes
- The subclass has additional associations
- The subclass is handled/reacted to differently

51

An Example

- Is this a good idea for the POS system?



52

Motivation for Superclasses

- When does it make sense to create a superclass for a set of classes?

53

Creating Superclasses

- All superclass attributes/associations apply to all subclasses
- If all subclasses have the same attribute, it should be moved to the superclass
- If all subclasses have the same association, it should be moved to the superclass

54

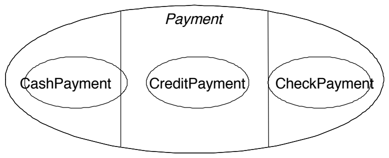
Example

- POS system uses external authorization services for credit payment
- Three different kinds of transactions: requests, approvals, denials.
- Each has date and time
- Approvals and denials have processing time

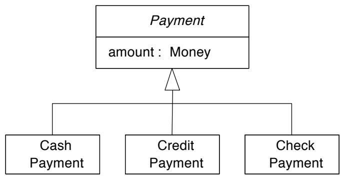
55

Abstract Classes

- If every member of class **C** must be a member of a subclass, then class **C** is an **abstract conceptual class**.

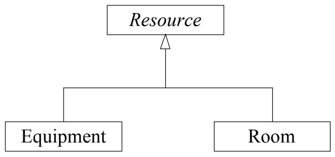


An example



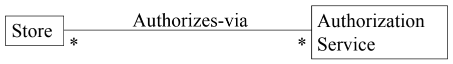
Example

- How about this?



Association Classes: An Example

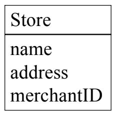
- A store uses a set of external authorization services for payments



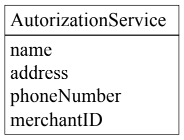
- Each service associates a merchantID with the store
  - The ID is provided by the store as part of the authorization request
  - Store has different IDs for different services

Example

- Option 1:

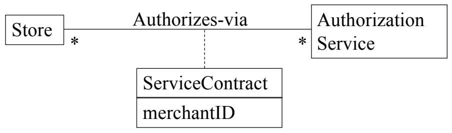


- Option 2:



Example

- Attribute merchantID is conceptually related to the association, not to the class
- Therefore: use **association class** to represent attributes of the association

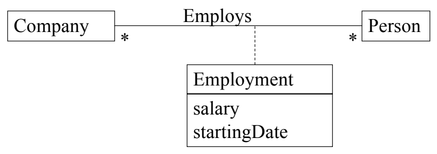


Association Classes

- An association class contains attributes of an association
- An association class may be useful:
  - When an attribute “does not fit” in the classes participating in an association
  - When the lifetime of the attribute depends on the lifetime of the association
  - With many-to-many associations

Another Example

- A company may employ several persons
- A person may be employed by several companies
- Attributes: salary, starting date...



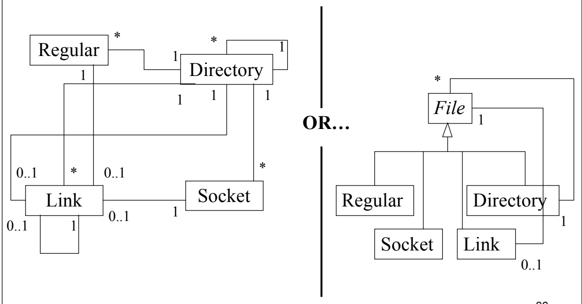
Summary of Domain Modeling

- Central focus: conceptual classes
  - Associations, attributes, and generalizations
  - Represented by UML class diagram
- No single correct model
  - All are approximations of the problem domain
  - Should capture essential domain aspects
    - In the context of current requirements
  - Should aid the understanding of the domain

Example

- You have <u>regular files</u>, <u>sockets</u>, <u>directories</u>, and <u>links</u>.
  - A directory stores files
  - A link is a special “pointer” to a target file
  - A file can be arbitrarily complex
- Can you draw a domain model?

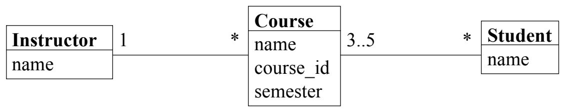
Example (cont.)



Example

- Consider a software system for processing course information. Students register for <u>courses</u>, and <u>instructors</u> teach <u>courses</u>. It is important that course info is destroyed at the end of the semester.

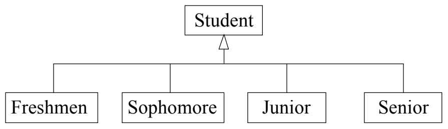
Example (cont.)



- Better: We can use CourseDescription

Example

- Consider a software system that has to store info about students and their current status in a university (i.e., freshman, sophomore, etc.). Is this a good idea? Why?


Student
Freshmen
Sophomore
Junior
Senior