

An *Item* Concept Example

- *Item* instance represents a physical item in a store
- *Item* has a description, price and UPC which are not recorded anywhere else
- When an *Item* is sold, corresponding entry is deleted from software dBase

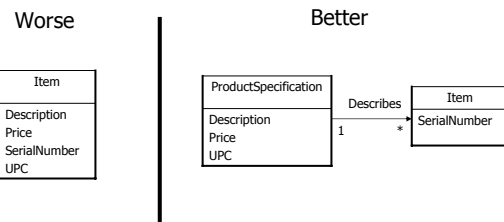
Problems w/ *Item* Concept

- Duplicate data (description, price, UPC)
 - Very space inefficient
- Store sells out *LowCarbBurger*, a high demand *Item*
 - Can we answer a question like "how much *LowCarbBurger* cost"?

Solution: Specification Concept

- Solution to *Item* concept problem is to add a new concept called *ProductSpecification*
 - It represents a description of information about items

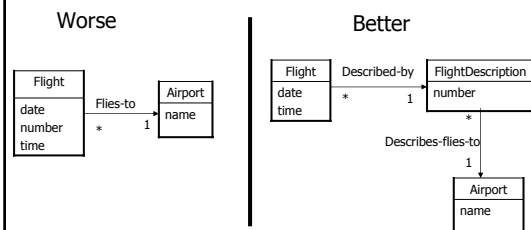
More on *Item* Example



Flight Example

- What airport a flight goes to is in the *Flight* instance
- An airline company suffers a fatal crash
- All their flights are cancelled for 6 months
- Their corresponding *Flight* objects are deleted from software dBase
- How can you get record of flight routes?

More on *Flight* Example



Specification Concept

- In general, add a description or specification concept when
 - Deleting instances of things they describe results in loss of information that needs to be maintained
 - It reduces redundant or duplicate information

Conceptual Model: Associations

Definition

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



Associations to Consider

- Knowledge of the relationship needs to be preserved for some duration
 - *Need-to-know* associations
- Associations derived from *Common Associations List*

Common Associations List

A is physical part of B	<i>Drawer - Register</i>
A is a logical part of B	<i>SalesLineItem - Sale</i>
A is physically contained in/on B	<i>Register - Store Item - Shelf</i>
A is logically contained in B	<i>ItemDescription - Catalog</i>
A is a description for B	<i>ItemDescription - Item</i>
A is a line item of a transaction or report B	<i>SalesLineItem - Sale</i>
A is known/logged/recorded/ reported/captured in B	<i>Sale - Register</i>

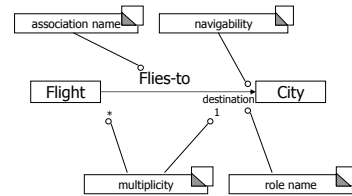
Common Associations List

A is a member of B	<i>Cashier - Store</i>
A is an organizational subunit of B	<i>Department - Store</i>
A uses or manages B	<i>Cashier - Register</i>
A communicates with B	<i>Customer - Cashier</i>
A is related to a transaction B	<i>Customer - Payment</i>
A is a transaction related to another transaction B	<i>Payment - Sale</i>
A is next to B	<i>SalesLineItem - SalesLineItem</i>
A is owned by B	<i>Register - Store</i>
A is an event related to B	<i>Sale - Customer</i>

High Priority Associations

- A is *physical* or *logical* part of B
- A is *physically* or *logically* contained in/on B
- A is *recorded* in B

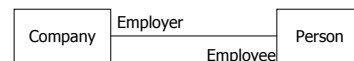
UML Notation



Naming Associations

- Use *TypeName-VerbPhrase-TypeName* format
 - *VerbPhrase* is the name of the *association*
- Example: *Flight – Flies-to – City*

Role Naming



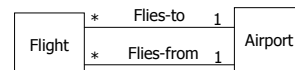
Multiplicity

- Defines how many instances of a type A can be associated with one instance of type B
- Example:
 - One *Store* stocks many *Items*



Multiple Associations

- *Flight – Flies-to – Airport*
- *Flight – Flies-from – Airport*



Multiple Associations (cont)



Association Guidelines

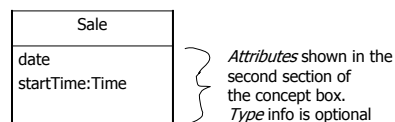
- Focus on “need-to-know” associations
 - High priority associations are strongly “need-to-know”
- It is more important to identify concepts than associations
- Showing too many associations **at this phase** can confuse a conceptual model

Conceptual Model: Attributes

Definition

- An *attribute* is a logical data value of an object
- Include attributes when requirements (e.g., use cases) suggests or imply a need to remember information

UML Notation



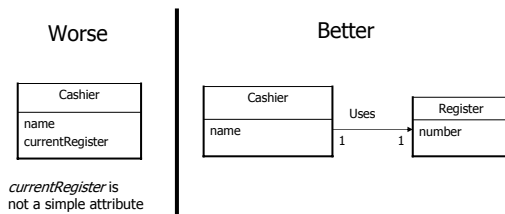
Keep Attributes Simple

- Attributes should preferably be **simple attributes**
- Very common simple attribute types:
 - Boolean, Date, Number, String/Text, Time

Attributes vs. Concepts

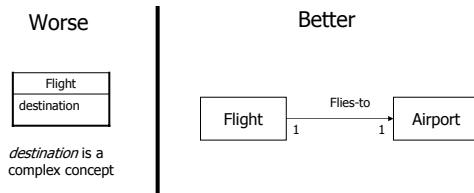
- Does it make sense to distinguish between two separate instances of a *Person* whose names are John Doe?
- In attributes, identity is determined by the value
- **Tip:**
 - Stick to simple types
 - If in doubt, define something as concept, rather than attribute

Cashier-Register Example



Relate with associations, not attributes

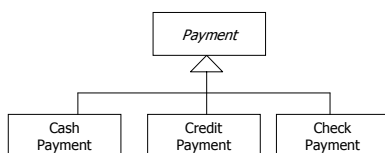
Flight Example



Avoid representing complex domain concepts as attributes; use associations

Conceptual Model: Generalization

Generalization-Specification Class Hierarchy



Superclass represents a more general concept, and *subclass* more specialized ones

Generalization

- Activity of identifying commonality among concepts
- *Superclass* is more general or encompassing than a *subclass*
- All members of a *subclass* set are members of their *superclass* set

Subclass Definition: 100% Rule

- 100% of *superclass*' definition should be applicable to *subclass*
- This includes:
 - Attributes, and
 - Association
- Example:
 - *Payment* has an *amount* and pays for a *sale*
 - This is true for *CreditPayment*, *CashPayment*, *CheckPayment*

Subclass Set: Is-a Rule

- All members of a *subclass* set should be members of their *superclass* set
- **Subclass is a Superclass**
- Example:
 - *CreditPayment* is a *Payment*
 - This is true for *CheckPayment* and *CashPayment*

When to define subclass

- Subclass has additional attributes of interest
 - Library: *Book*, a subclass of *Loanable* resource, has ISBN attribute
- Subclass has additional associations of interest
 - POST: *CreditPayment*, a subclass of *Payment*, is associated with a *CreditCard*

When to define subclass (cont)

- Subclass concept is operated upon, handled, reacted to, or manipulated differently than superclass (or other subclasses) in ways that are of interest
 - POST: *CreditPayment*, a subclass of *Payment*, is handled differently than other kinds of payments

When to define superclass

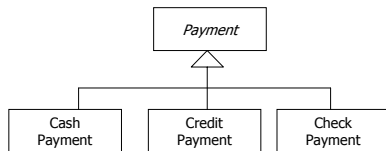
- Potential subclasses represent variations of a similar concept
- Subclasses conform to 100% and Is-a rules
- All subclasses have same attribute that can be factored out and expressed in superclass
- All subclasses have same associations that can be factored out and related to superclass

Abstract Class

- If every member of class *T* is also member of a *subclass*, then class *T* is an abstract class
- Example: If there is no *Payment* that is not *CashPayment*, *CreditPayment* or *CheckPayment*, then *Payment* is an abstract class

Abstract Class: UML Notation

- Abstract super class is indicated by *italics*
- No instance of an abstract object can be created



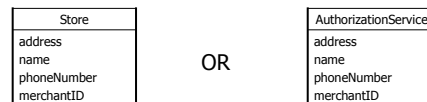
Conceptual Model: More on Associations

Example: Merchant ID

- Authorization services assign a merchant ID to each store
- A store has different merchant ID for each service
- Payment authorization request requires a valid merchant ID

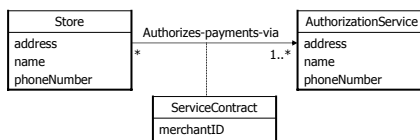
Example: Merchant ID (cont)

- Q: Where does merchant ID attribute belongs to?



Example: Merchant ID (cont)

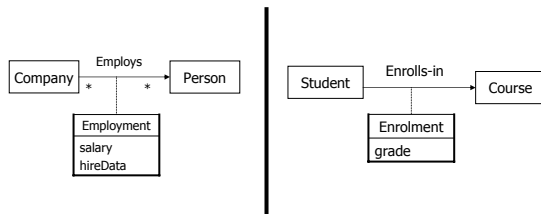
- A: None. Use an associative class



Associative Class Guidelines

- There is many-to-many relationship between two concepts
- An attribute is related to an association
- Instances of associative class have a life-time dependency on association

Associative Classes: Examples



Aggregation and Composition

- A special kind of association
- Models whole-part relationship between things
- Whole is usually referred to as *composite*

Composite aggregation

- Also referred to as composition
- Composite solely owns the part and they are in a tree structure parts hierarchy
- Most common form of aggregation
- In UML, represented by filled diamond



Shared Aggregation

- Part may be in many composite instances
- In UML, represented as hollow diamond

