SE463
Software Requirements Specification & Analysis

Domain Models

Requirements Models

Model: a simplified version of something complex used in analyzing and solving problems or making predictions

Uses of Models in Requirements

- can guide elicitation
- can provide a measure of progress
- can help to uncover problems, especially omissions
- can help us check our understanding

Unified Modeling Language (UML)

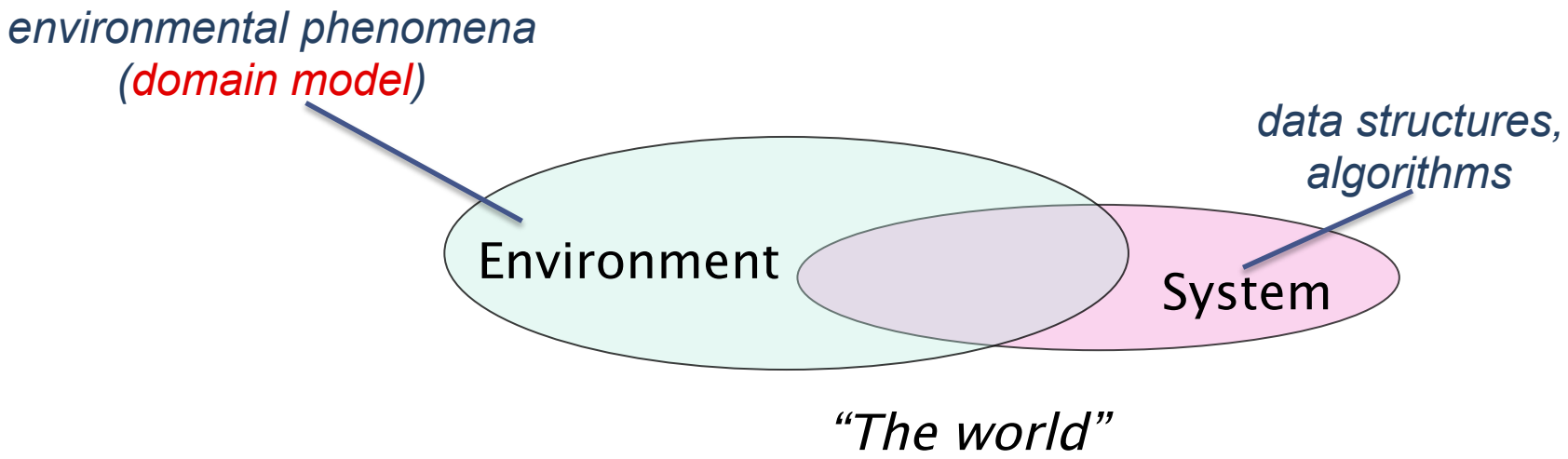
Structural

- Class Diagrams
- Object Diagrams
- Composite Structure Diagrams
- Component Diagrams
- Package Diagrams
- Deployment Diagrams

Behavioural

- Use Case Diagrams
- Activity Diagrams
- Interaction Overview Diagrams
- Sequence Diagrams
- Communication Diagrams
- Timing Diagrams
- State Machine Diagrams

Environmental Phenomena

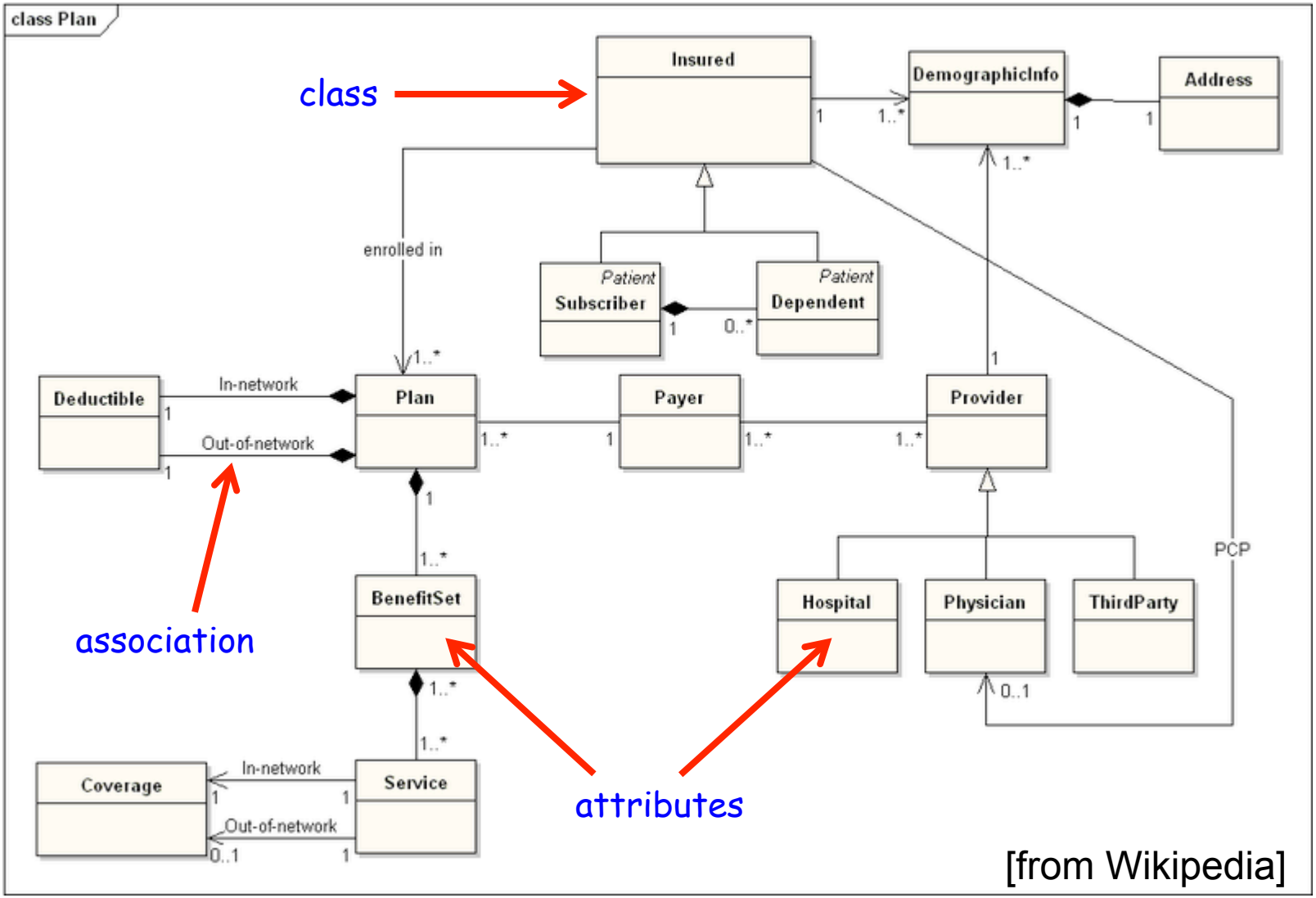


Requirements are expressed in terms of environmental phenomena.

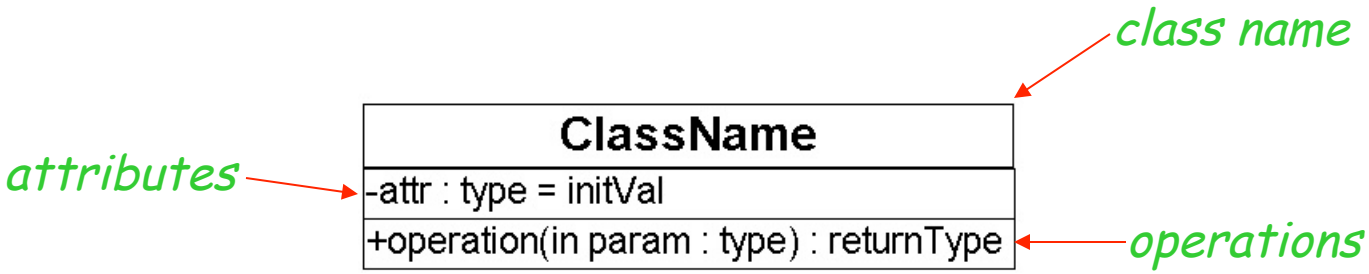
A domain model visualizes the elements of the environment that are relevant to the Work or SUD.

- a graphical data dictionary

Example of (early) domain model



Domain Model Classes



Often leave out details early on:

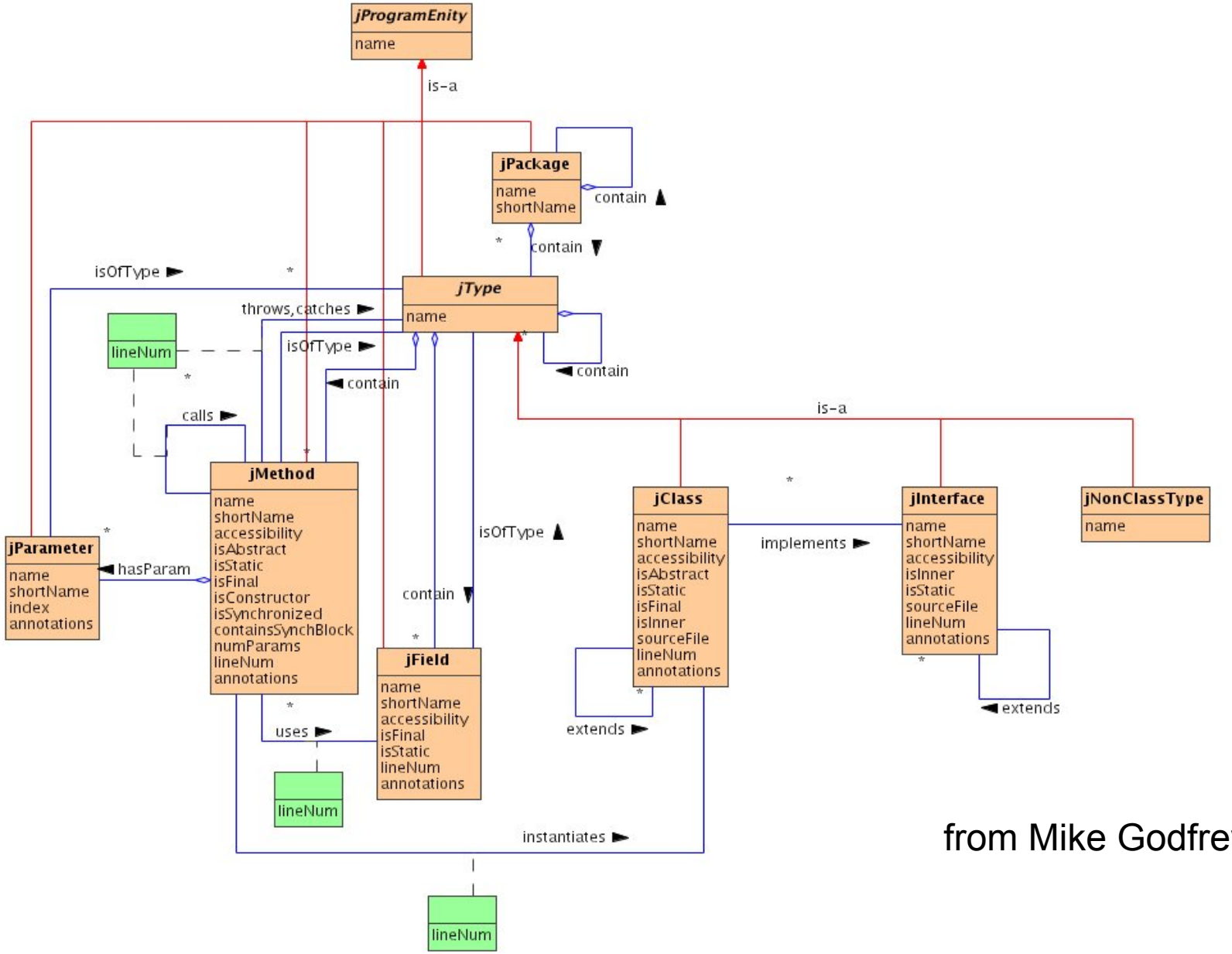
- start with just a class name
- add attribute names
- (maybe) add data types, params, initVals, *etc.*

} reqs

- NO operation names
- NO prog. lang. datatypes, access permissions, *etc.:*
  - + `public`, # `protected`, - `private`
  - $ static, / derived

} design

Partial domain model for a Java code analyzer



from Mike Godfrey

Larman's canonical conceptual classes

Conceptual class category	*Example (airline reservation)*
Business transactions	*FlightReservation*
Transaction line items	*SalesLineItem*
Product/service related to a trans. or TLI	*Flight, seat, meal*
Where is the trans recorded?	*FlightManifest*
Roles of people/orgs related to trans; actors in use cases	*Passenger, Airline*
Place of transaction / service	*Airplane, Plane, Seat*
Events (that have noteworthy time/place)	*FlightDepature*
Physical objects	*Airplane*

[Larman Table 9.1]

Larman's canonical conceptual classes

Conceptual class category	*Example (airline reservation)*
Descriptions of things	*FlightDescription*
Catalogs	*FlightCatalog*
Containers of things (physical or info)	*Airplane*
Things in a container	*Passengers*
Other collaborating systems	*AirTrafficControl*
Records of finance, work, contracts	*MaintenanceLog*
Financial instruments	*TicketCredit*
Schedules, documents, manuals that are regularly referred to when performing work	*RepairSchedule*

[Larman Table 9.1]

Example: UW Parking Permits

- All students, employees, and visitors to the university must pay for parking. Students and employees may buy permits that allow them to park in particular lots. Or they may park as visitors, and pay a fee to park in visitors' parking.
  - **Reserved Employee Parking Lots (Lots A, B, H, K, L, O, R)** Each permit is associated with a particular parking lot and allows the holder of the permit to park in that lot, anytime between 6:00AM and 3:00AM. Employees buy a long-term permit for which they are charged a monthly fee. The monthly fee is the same for all reserved parking lots. A permit has a start date and an open-ended end date; the permit ends when the permit is cancelled. Reserved Employee Parking Lots can be used by visitors during the evenings; the starting time for visitor parking is different for each lot.
  - **Unreserved Student Parking Lots (Lots C, N, W, X)** Permits for these lots are sold to students on a per term basis. Holders of such permits may park in any of the unreserved student parking lots anytime between 6:00AM and 3:00AM. There is no visitor parking in the unreserved student parking lots.
  - **Visitor Parking Lots (Y, Z)** There are two parking lots that are dedicated to visitor parking. Anyone can pay a daily fee to park in the visitor lots anytime between 6:00AM and 3:00AM.

How to find domain entities

Larman recommends three approaches:

1) *Refine an existing domain model*
   - Created for a previous system, or a well understood domain

2) *Use a conceptual class category list*
   - Brainstorm about all the *entities* and *entity kinds* that are relevant to the system

3) *Identify noun phrases*
   - … in use cases (and other docs, if available)

Attributes

An *attribute* is a *simple* sub-part of an object.

- Characteristics or properties of object instances
- Initially, an attribute is modelled as just a name
- Or, an attribute may be a more complex type whose structure is unimportant to the problem
- There are some default UML primitive (simple) types.
  - Feel free to invent meaningful primitive types for your specification, such as “money” or “date”.

Associations

- Represent *important* relationships between (instances of) two conceptual classes
  - Physical relationships are good candidates, but need not be named ("has" or "part-of" can be inferred).
  - Transient relationships are bad candidates
    - e.g., "communicates with"
  - Avoid vague relations like "knows about". Be specific or be quiet!
  - Larman: "Reduce visual noise!"
    - Focus on *"We need to remember this fact"* rather than *"Here's the set of all possible things that are true about the domain model"*
- An association name (if exists) is typically a short verb
- The same two classes can be related by multiple associations

UW Parking Permits (Revisited)

- All students, employees, and visitors to the university must pay for parking. Students and employees may buy permits that allow them to park in particular lots. Or they may park as visitors, and pay a fee to park in visitors' parking.
  - **Reserved Employee Parking Lots (Lots A, B, H, K, L, O, R)** Each permit is associated with a particular parking lot and allows the holder of the permit to park in that lot, anytime between 6:00AM and 3:00AM. Employees buy a long-term permit for which they are charged a monthly fee. The monthly fee is the same for all reserved parking lots. A permit has a start date and an open-ended end date; the permit ends when the permit is cancelled. Reserved Employee Parking Lots can be used by visitors during the evenings; the starting time for visitor parking is different for each lot.
  - **Unreserved Student Parking Lots (Lots C, N, W, X)** Permits for these lots are sold to students on a per term basis. Holders of such permits may park in any of the unreserved student parking lots anytime between 6:00AM and 3:00AM. There is no visitor parking in the unreserved student parking lots.
  - **Visitor Parking Lots (Y, Z)** There are two parking lots that are dedicated to visitor parking. Anyone can pay a daily fee to park in the visitor lots anytime between 6:00AM and 3:00AM.

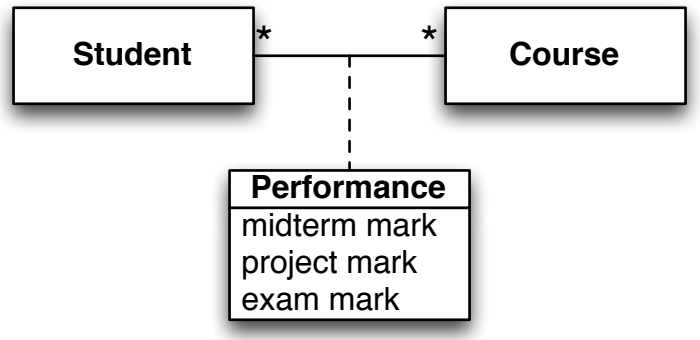
Roles and multiplicities

- A *role* is one end of an association
  - It describes how the “local” class is viewed by the “remote” class within the context of the relationship
- Can specify *multiplicities* of associations
  - *e.g.,* 7, 1..52, 0..1, *
  - If not stated, assume that multiplicity is “unstated”
- Roles and multiplicities are optional
- NO *navigability* arrow (this is a design concept)

Association classes

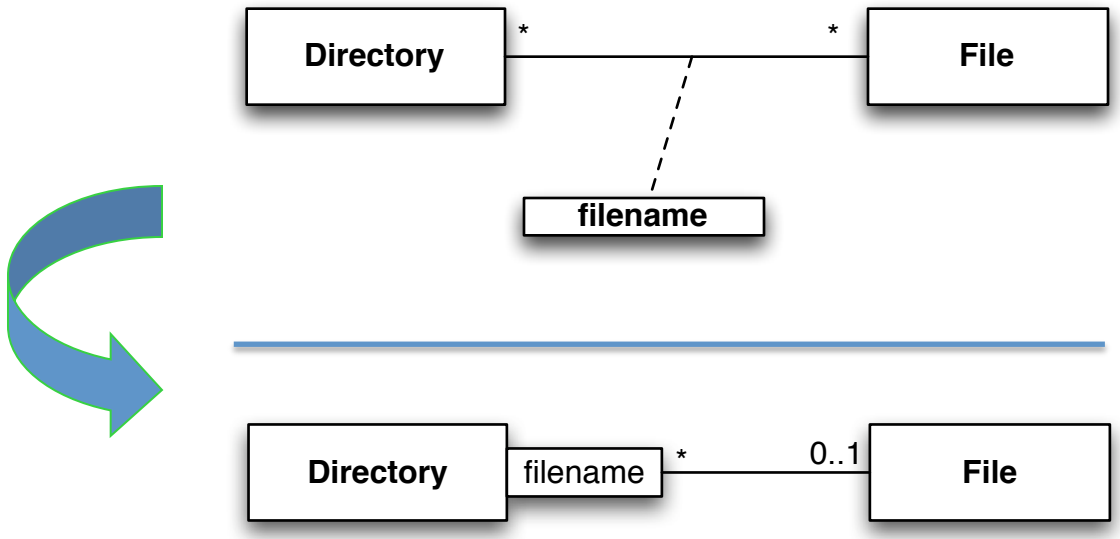
If a relationship has interesting attributes of its own, it should be an *association class*.

- The state of an association class is knowable only by instances of the two classes
  - If others need to know this info too, just make it into a normal class
- Any two objects can have at most one instance of a given association class

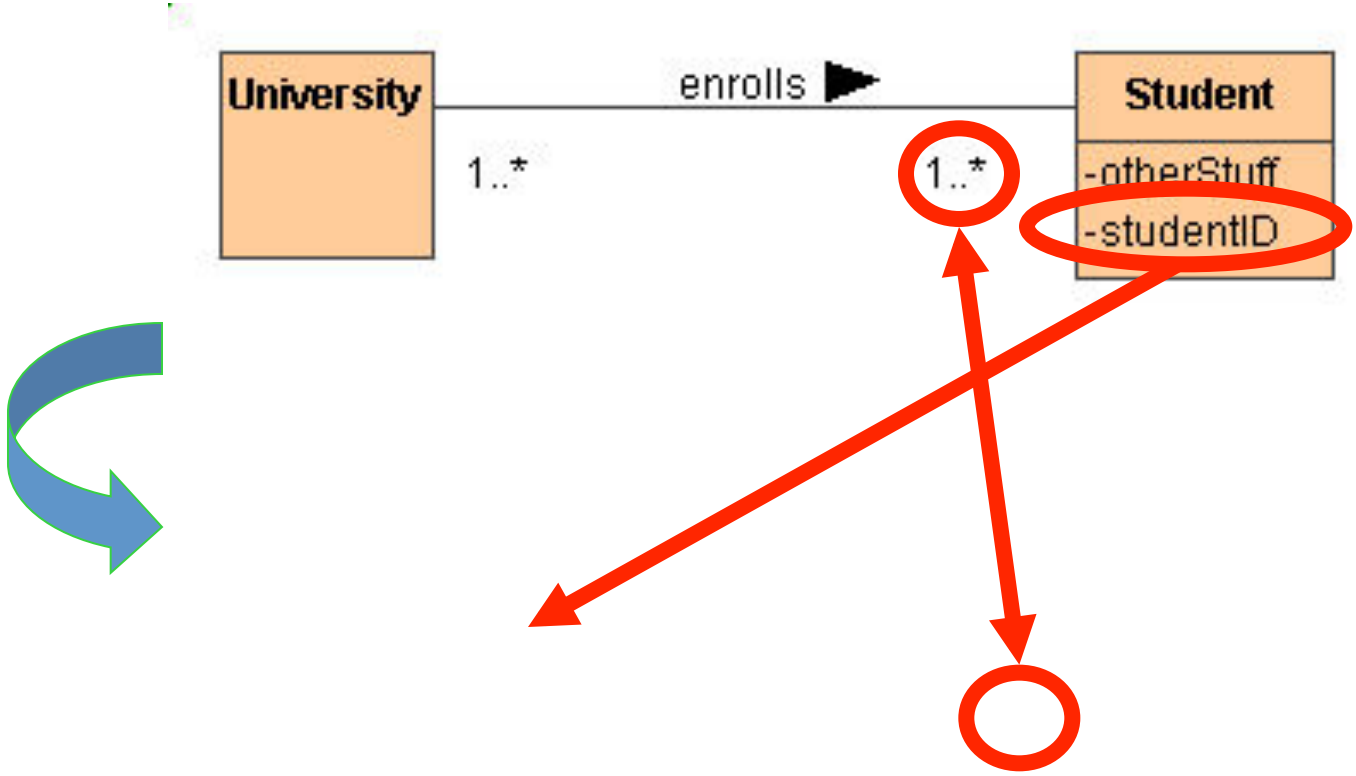


Association qualifiers

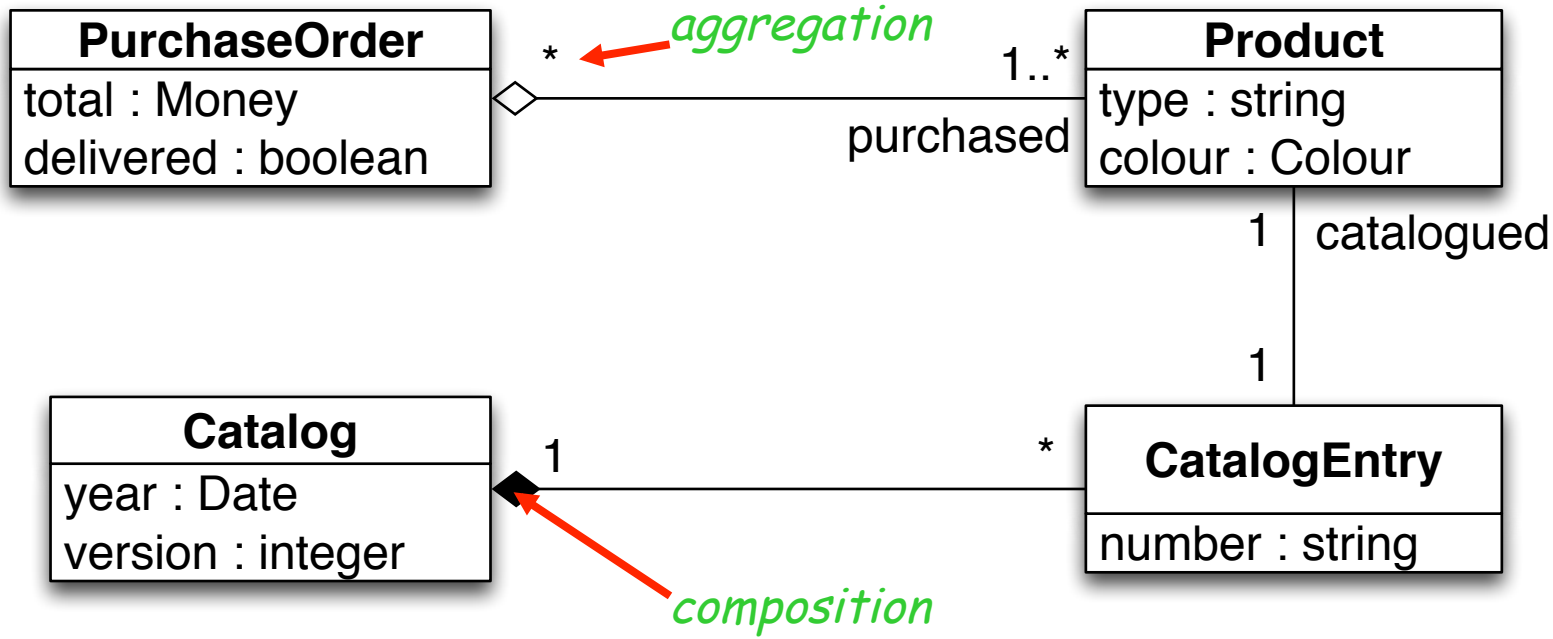
- A *qualifier* is a special association attribute used at one end of an association to distinguish among the set of objects at the other end of the assoc
  - *"uniquely identifies"*



Association qualifiers



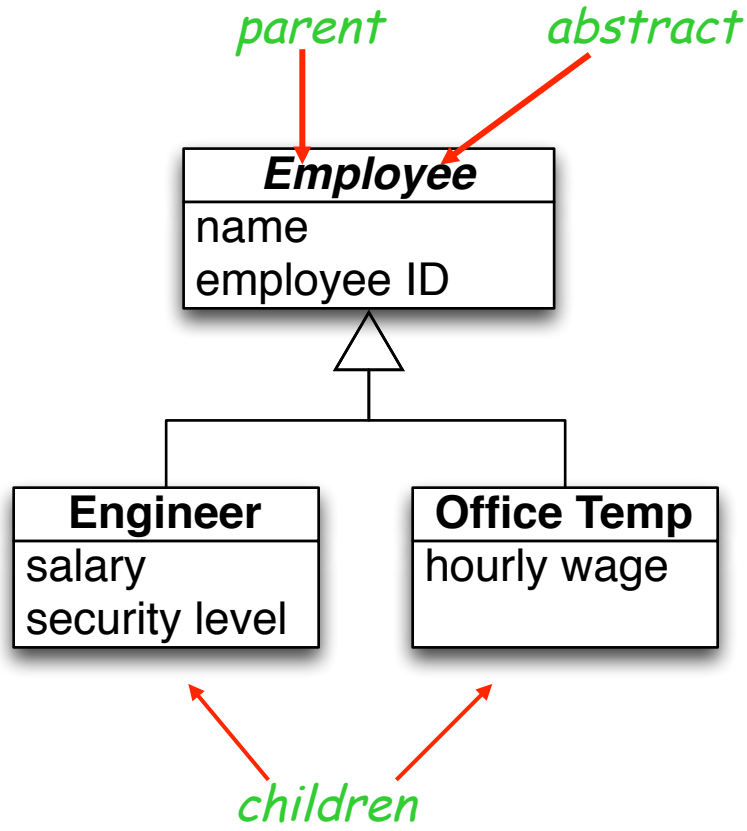
Composition / aggregation



- Composition ◆ an object may *strongly* be a part of at most one container
  - When container dies, subparts die too.
- Aggregation ◇ an object may belong *weakly* to several containers

Inheritance / Generalization

- The *is-a* (is-a-kind-of) relationship
  - *Engineer* is a kind of *Employee*
- All attributes / associations / other properties of parent are inherited by children
- Children can
  - Fill in abstract details of parent's features
  - Add new features
- Non-leaf nodes can be *abstract*
  - *i.e.,* represent abstractions, not concrete entities
  - Indicated by *italicized* name



General advice on domain models

- Class diagrams vs. domain models
  - Making class diagrams is easy: follow the rules
  - Domain modelling is much harder
    - Modelling experience, heuristics
    - Domain knowledge

- Simplicity is your friend
  - Start with just classes and associations
  - Limit the amount of detail you add
    - "I need this info to handle the problem I'm working on"

    c.f., laying out a use case in full detail

SE1 domain models *should* have …

- Attributes and their types
- Multiplicities on all associations (including “1” multiplicities)
- Association names, or role names, for all non-trivial associations
- Qualifiers, to simplify multiplicities on associations
- Actors shown as stick figures or as classes with «actor» stereotype
  - This requires you to show multiplicities between actors and classes, which can be a valuable requirements detail.
- For specifications: Interface devices as classes with «interface» stereotype

SE1 domain models should *not* have

- Class-level operations or methods
- Visibility annotations (i.e., private, protected, public)
- Navigability arrows
- Initial attribute values (unless you need them for model correctness)
- Object construction and destruction functions

Watch your language!

- The goal is to create a *conceptual* model:
  - Models of real-world entities (customers, accounts, bills) and not of system entities (databases, sw components)
- Focus on the information/artifacts that the system will input, transform, analyze, display, etc; physical and conceptual

Modelling Conundrums

Reference:
J. Rumbaugh, *OMT Insights, SIGS* Books, Inc., 1996.

Tightly-coupled classes

When there is a one-to-one association between two classes, are the classes distinct or should they be combined into a single class?

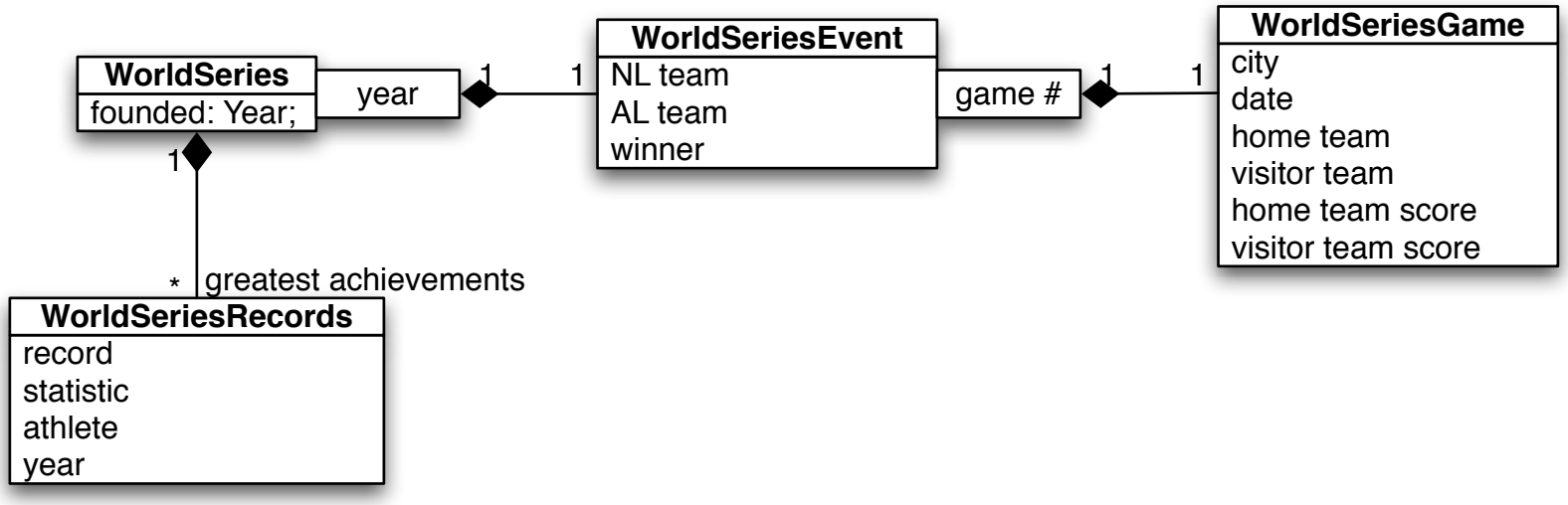
Distinct classes:

Single class:

If the links are dynamic, then the classes should be distinct

Similar but distinct classes

A common problem is mixing together two distinct concepts in a single class – especially when the two concepts have the same name.



Similar but distinct classes

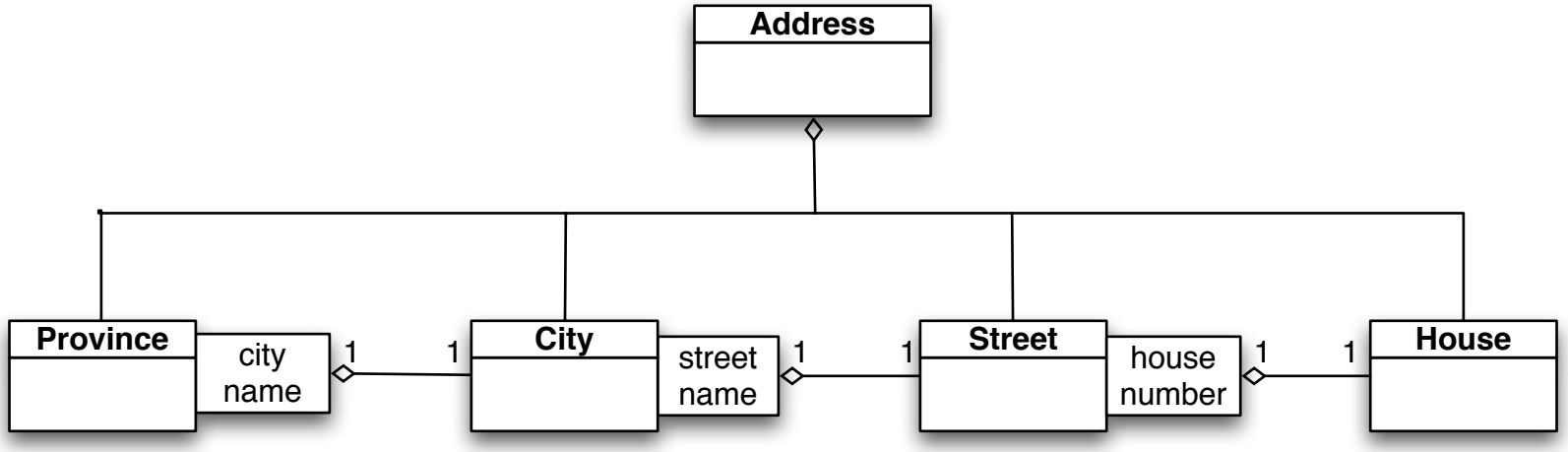
Suppose we want to keep track of cell-phone services, such as text messaging and roaming, which must be subscribed to in advance and that have a charge for each use.

- The phone company offers a set of services to which customers can subscribe.
- A customer may subscribe to one or more services. Each subscription may have properties.
- The use of subscribed services needs to be tracked, so that the customer can be charged for each usage.

Names are not objects

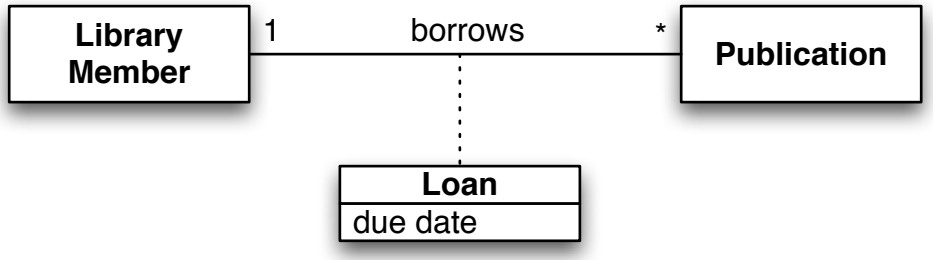
An object has an identity; it can be distinguished from other objects with the same value. A name is simply a string indistinguishable from other strings with the same value.

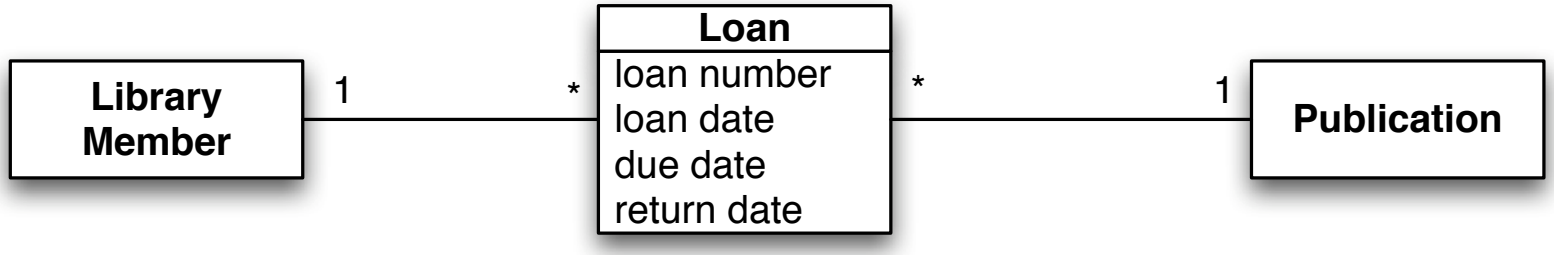
Bad model of address:



Temporal Objects

Sometimes we want to model only the current value of an object, and sometimes we want to record a history of values.



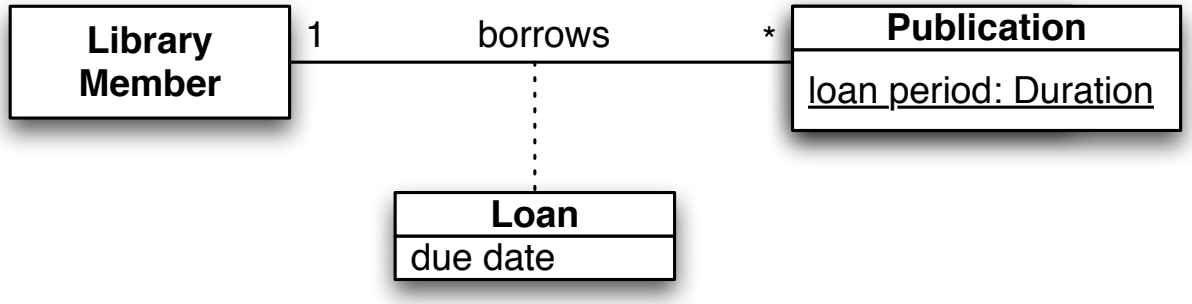


Class Scope Attributes

By default, every object has its own copy of attributes, and may have its own attribute values.

A class-scope attribute is an attribute whose value is changeable, but is shared by all of the class's object instances. Thus, all objects will have the same class-scope attribute value.

Syntax: underlined attribute declaration



Class Scope Attributes

What if we wanted to model the case where a publication's loan period depends on the type of the publication (e.g., book, periodical, CD)?

Class Scope Attributes

What if we wanted to model the case where a publication's loan period depends on a combination of the publication's type and the status of the library member (e.g., faculty, student)?

Example: Job Mine

Create a domain model for Job Mine. Include

- students
- employers
- jobs
- applications (and their components)
- interview schedules
- rankings
- matches

Summary

Domain models offer a graphical visualization of the environmental phenomena that are relevant to the requirements problem.

- visual data dictionary
- big-picture view of the problem space