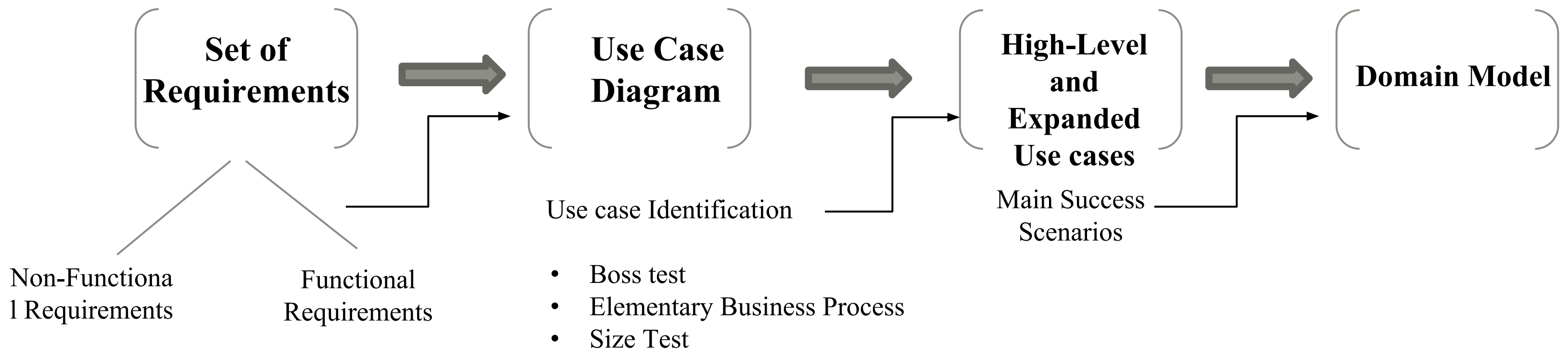
UML MODELLING

Instructor: Mehroze Khan

Revision up till now



- Boss test
- Elementary Business Process
- Size Test

HOW TO CREATE DOMAIN MODEL

How to Create Domain Model

- Find conceptual classes
- Draw them as classes in a UML class diagram
- Add associations and attributes

Finding Conceptual Classes

Three Strategies to Find Conceptual Classes

1. Reuse or modify the existing model if one exists
2. Use a Category List
3. Identify noun phrases in your use-cases

Method 1: Reuse or Modify Existing Models

- There are published, well-‐crafted domain models and data models (which can be modified into domain models) for many common domains, such as inventory, finance, health, and so forth..

- Reusing existing models is excellent, but out of the scope of this course

Method 2: Use a Category List

- We can kick-start the creation of a domain model by making a list of candidate conceptual classes.
- Table contains many common categories that are usually worth considering, with an emphasis on business information system needs.
- The guidelines also suggest some priorities in the analysis. Examples are drawn from the
  1) *POS*
  2) *Monopoly*
  3) *Airline reservation domains*

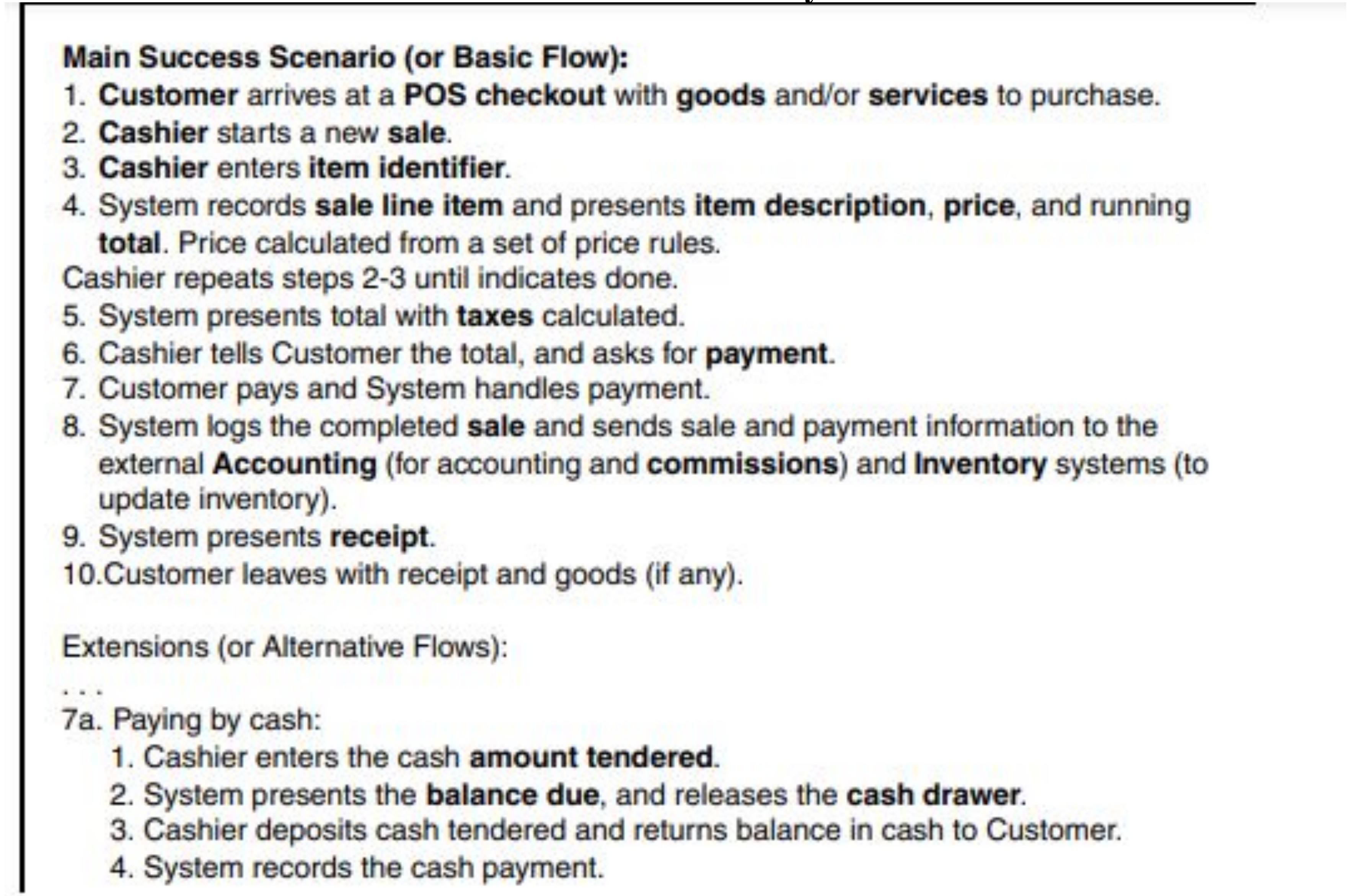
Conceptual Class Category	Examples
business transactions *Guideline*: These are critical (they involve money), so start with transactions.	*Sale, Payment* *Reservation*
transaction line items *Guideline*: Transactions often come with related line items, so consider these next.	*SalesLineItem*
product or service related to a transaction or transaction line item *Guideline*: Transactions are *for* something (a product or service). Consider these next.	*Item* *Flight, Seat, Meal*
where is the transaction recorded? *Guideline*: Important.	*Register, Ledger* *FlightManifest*
roles of people or organizations related to the transaction; actors in the use case *Guideline*: We usually need to know about the parties involved in a transaction.	*Cashier, Customer, Store* *MonopolyPlayer* *Passenger, Airline*
place of transaction; place of service	*Store* *Airport, Plane, Seat*
physical objects *Guideline*: This is especially relevant when creating device-control software, or simulations.	*Item, Register* *Board, Piece, Die* *Airplane*
descriptions of things *Guideline*: See p. 147 for discussion.	*ProductDescription* *FlightDescription*
catalogs *Guideline*: Descriptions are often in a catalog.	*ProductCatalog* *FlightCatalog*
containers of things (physical or information)	*Store, Bin* *Board* *Airplane*
things in a container	*Item* *Square (in a Board)* *Passenger*
other collaborating systems	*CreditAuthorizationSystem* *AirTrafficControl*
records of finance, work, contracts, legal matters	*Receipt, Ledger* *MaintenanceLog*
financial instruments	*Cash, Check, LineOfCredit* *TicketCredit*
schedules, manuals, documents that are regularly referred to in order to perform work	*DailyPriceChangeList* *RepairSchedule*

Method 3: Finding Conceptual Classes with Noun Phrase Identification

- Another useful technique (because of its simplicity) is linguistic analysis: **Identify the nouns and noun phrases** in textual descriptions **(use cases or other documents)** of a domain and consider them as candidate conceptual classes or attributes.

Method 3: Finding Conceptual Classes with Noun Phrase Identification

Basic Flow for a POS System



Main Success Scenario (or Basic Flow):

1. **Customer** arrives at a **POS checkout** with **goods** and/or **services** to purchase.
2. **Cashier** starts a new **sale**.
3. **Cashier** enters **item identifier**.
4. System records **sale line item** and presents **item description**, **price**, and running **total**. Price calculated from a set of price rules.

Cashier repeats steps 2-3 until indicates done.

5. System presents total with **taxes** calculated.
6. Cashier tells Customer the total, and asks for **payment**.
7. Customer pays and System handles payment.
8. System logs the completed **sale** and sends sale and payment information to the external **Accounting** (for accounting and **commissions**) and **Inventory** systems (to update inventory).
9. System presents **receipt**.
10. Customer leaves with receipt and goods (if any).

Extensions (or Alternative Flows):

. . .

7a. Paying by cash:

1. Cashier enters the cash **amount tendered**.
2. System presents the **balance due**, and releases the **cash drawer**.
3. Cashier deposits cash tendered and returns balance in cash to Customer.
4. System records the cash payment.

Example: Find and Draw Conceptual Classes

- Case Study: POS Domain
  - *From the category list and noun phrase analysis, a list is generated of candidate conceptual classes for the domain*

Sale
CashPayment
SalesLineItem
Item Register
Cashier
Customer
Store
ProductDescription
ProductCatalog
.
.
.

Decide which ones are classes and which ones are attributes
Add attributes and relation to the identified domain classes

POS Domain Model

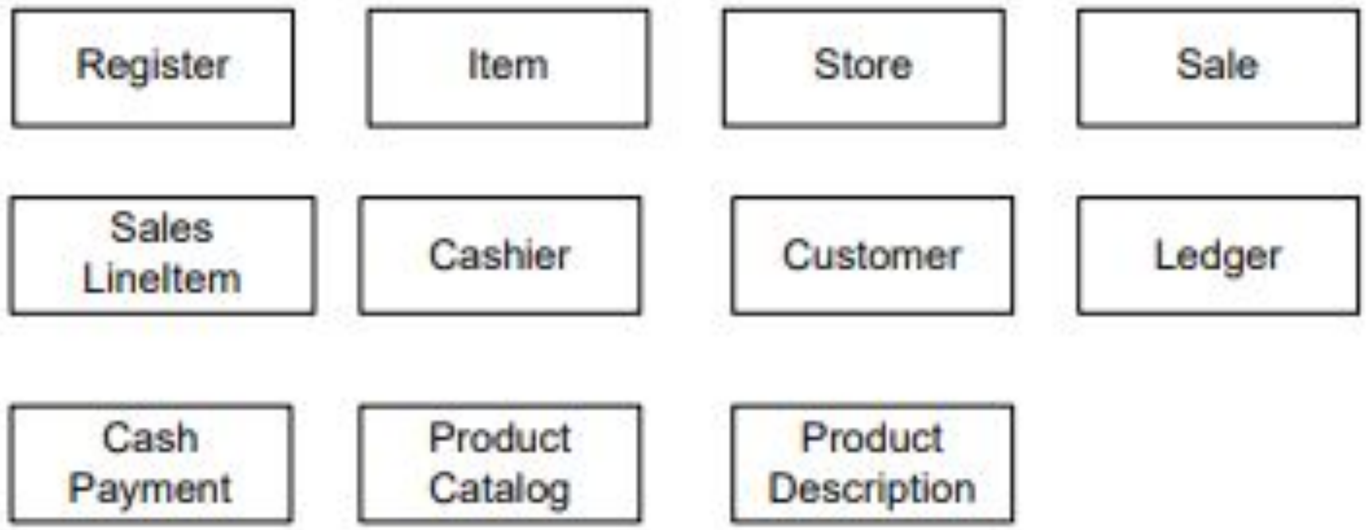
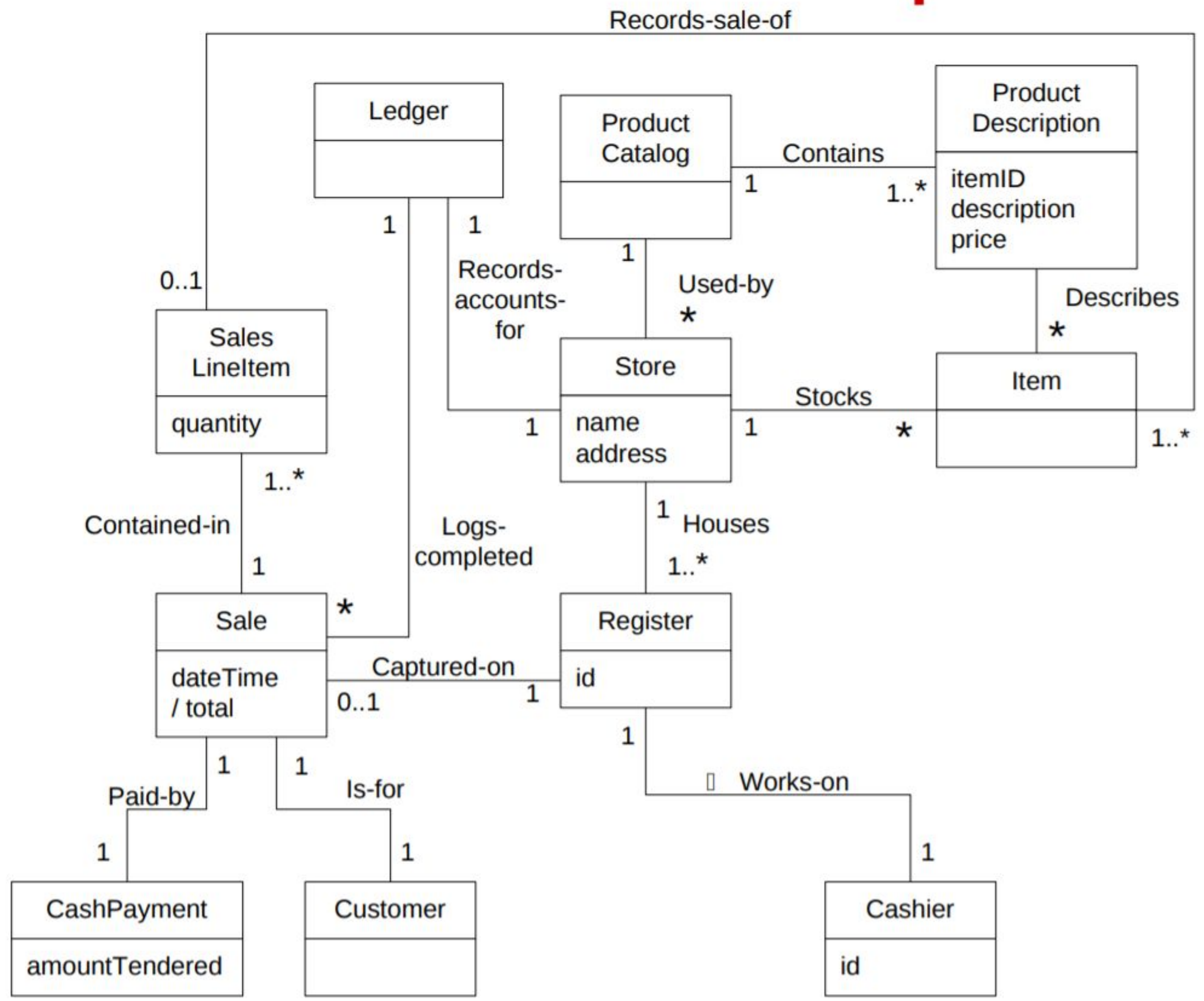


Figure 9.7 Initial POS domain model.


Records-sale-of
Ledger
Product Catalog
Product Description
itemID
description
price
Contains
1
1..*
1
1
1
Records-accounts-for
Used-by
*
Describes
*
0..1
Sales LineItem
quantity
Store
name
address
Stocks
1
1
*
Item
1..*
1..*
Contained-in
1
Logs-completed
1
Houses
1..*
Sale
*
dateTime
/ total
Captured-on
0..1
1
Register
id
1
1
1
Paid-by
Is-for
Works-on
1
1
1
CashPayment
amountTendered
Customer
Cashier
id

Domain Modeling Guidelines

- On Naming and Modeling Things: The Mapmaker

- Mapmakers uses the names of the territory--they do not change the names of cities on a map. For a domain model, this means to use the vocabulary of the domain when naming concepts and attributes. For example, if developing a model for a library, name the customer a "Borrower" --the term used by the library staff.

- A mapmaker deletes things from a map if they are not considered relevant to the purpose of the map; a domain model may exclude concepts in the problems domain not pertinent to the requirements.

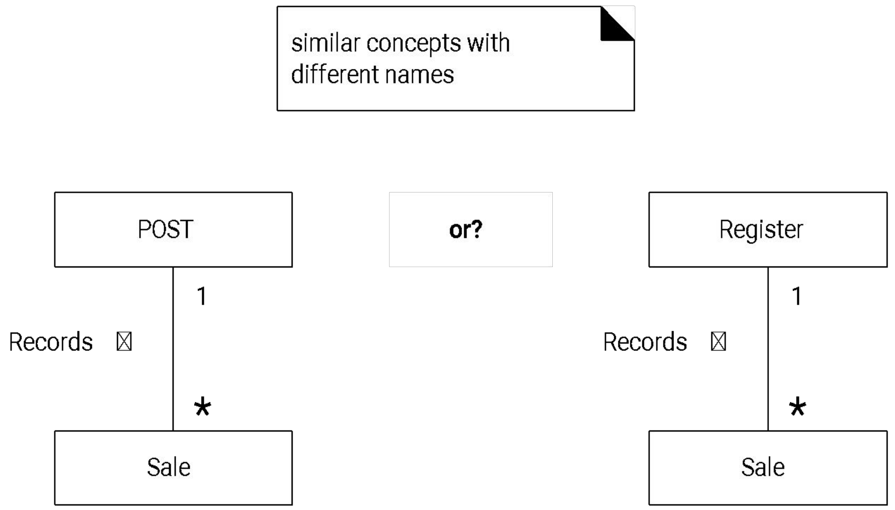
- A mapmaker does not show things that are not there, such as a mountain that does not exist. Similarly, the domain model should exclude things not in the problem domain under consideration.

Domain Modeling Guidelines

- On Naming and Modeling Things: The Mapmaker

- Make a domain model in the spirit of how a mapmaker works:
  - *Use the existing names in the territory.*
  - *Exclude irrelevant features.*
  - *Do not add things that are not there.*

Resolving Similar Concepts - POST versus Register



Rule of thumb, a domain model is not absolutely correct or wrong, but useful tool of communication.

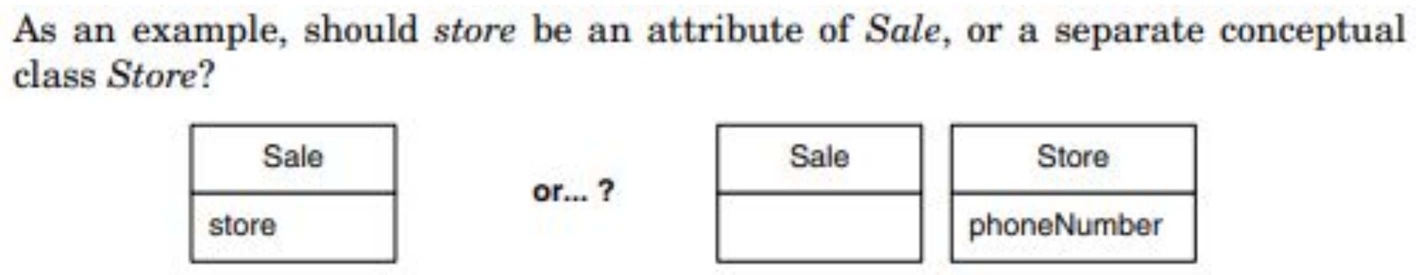
Both POST and Register are equally useful terms, but POST will give a better implementation view.

Concept vs. Attribute

- A Common Mistake in Identifying Concepts:

  Attribute OR Concept?

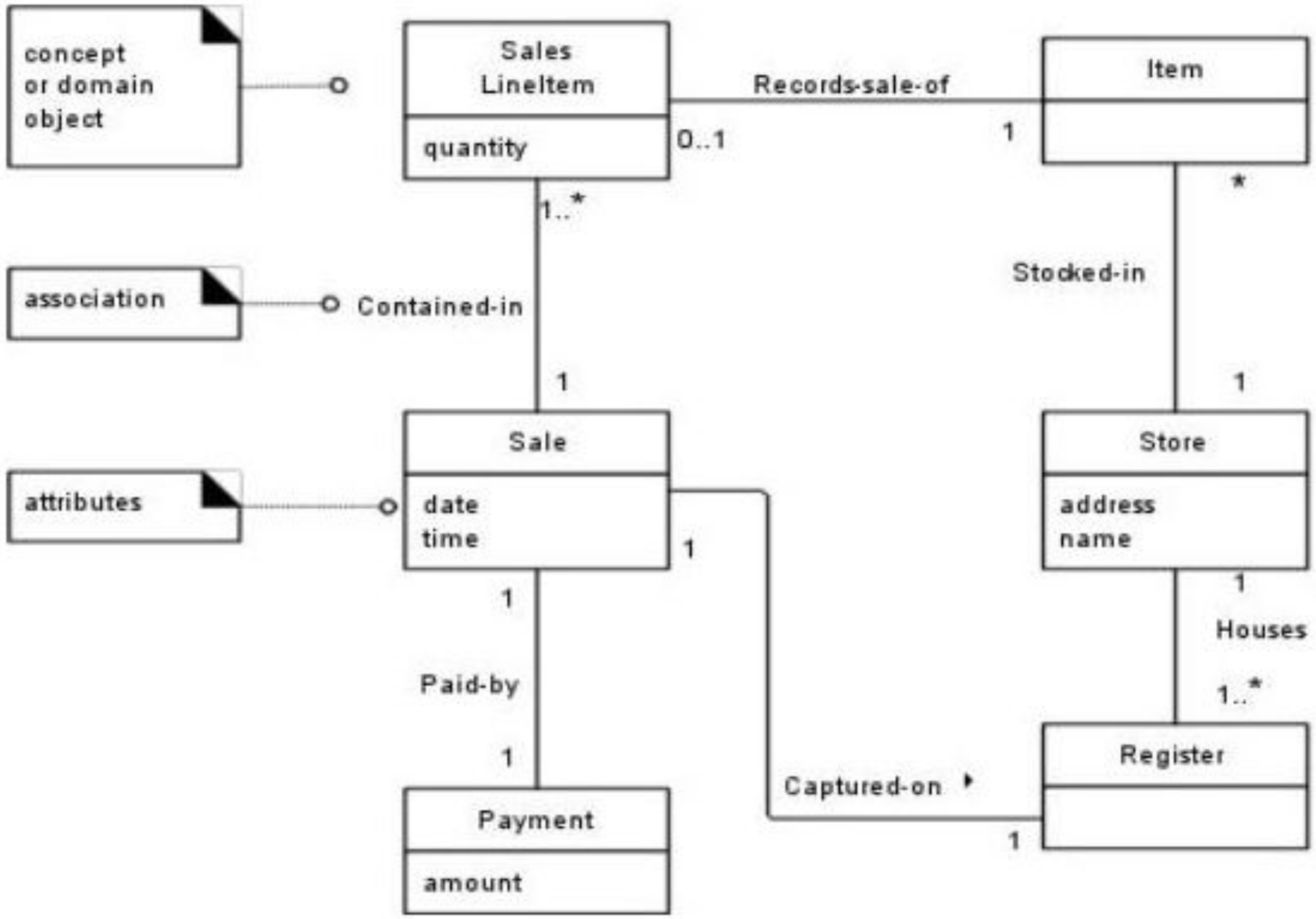
  If we do not think of some concept X as a number or text in the real world,
  X is probably a concept, not an attribute.



As an example, should *store* be an attribute of *Sale*, or a separate conceptual class *Store*?

If in doubt, make it a separate concept.

UML Attribute Notation



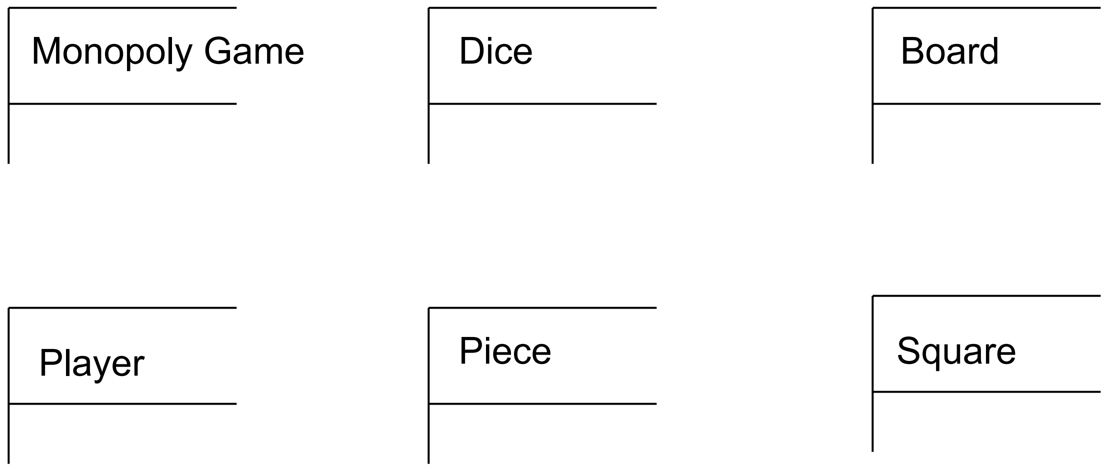
Conclusion

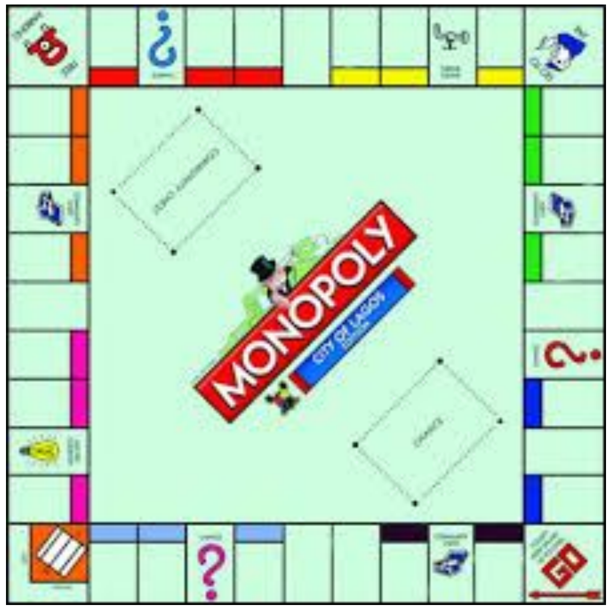
- When in doubt if the concept is required, **keep the concept**.
  - *Why?: During the analysis phase, it's important to **capture all potentially relevant elements** of the system. It's easier to later remove unnecessary classes or merge them with others during design or implementation than to add a missing concept that wasn't initially considered.*
  - *Example: If you're not sure whether "**Address**" should be a separate class, it's safer to keep it as a class in the diagram. You can refine it later.*

Conclusion

- When in doubt if the association is required, **drop it**.
  - *Why?: Associations **add complexity to your model**. Unnecessary relationships can clutter the diagram and make the system harder to understand and implement. It's easier to add an association later if it turns out to be needed, than to remove it if it proves to be unnecessary.*
  - *Example: If you're not sure whether there should be a direct association between "**Customer**" and "**OrderHistory**," it's better to omit it initially. You can always introduce it later if you determine that it's required.*

Monopoly Game domain model (first identify concepts as classes)





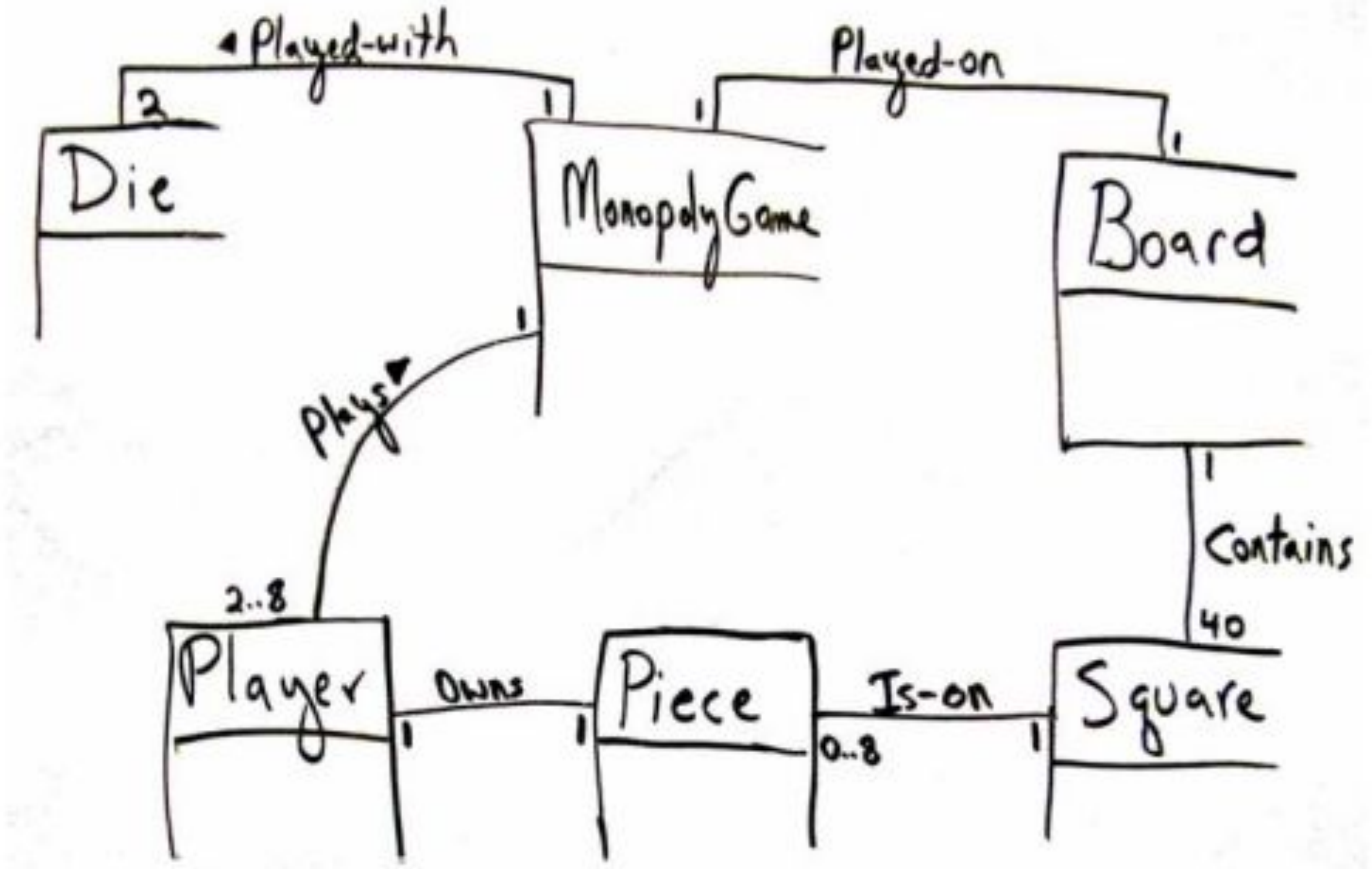


Figure 9.18 Monopoly partial domain model.

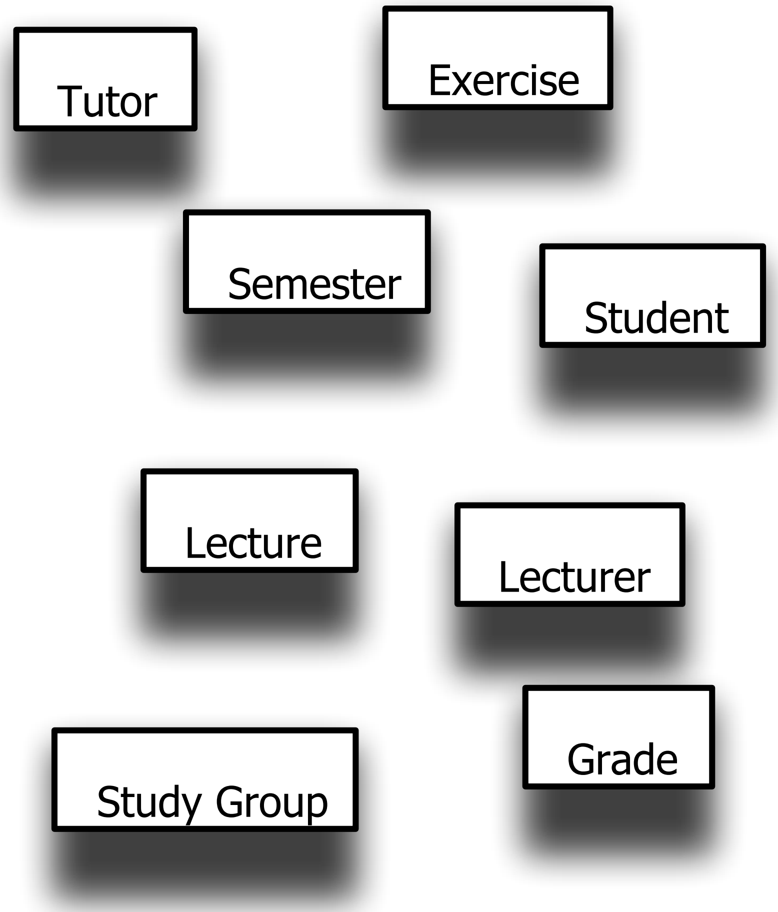
Visualizing Domain Models

Statements about a Course Management System

- During a semester, a lecturer delivers one or more lectures
- Sometimes the lecturer is on leave to focus on doing research, in this case (s)he does not give a lecture
- A student usually attends many, unless (s)he has something better to do
- During the semester there will be several exercises which are meant to be solved by small study groups
- Each student is assigned to one particular study group for the whole semester
- A study group consists of two to three students
- After submission of a solution by a study group it is graded by a tutor

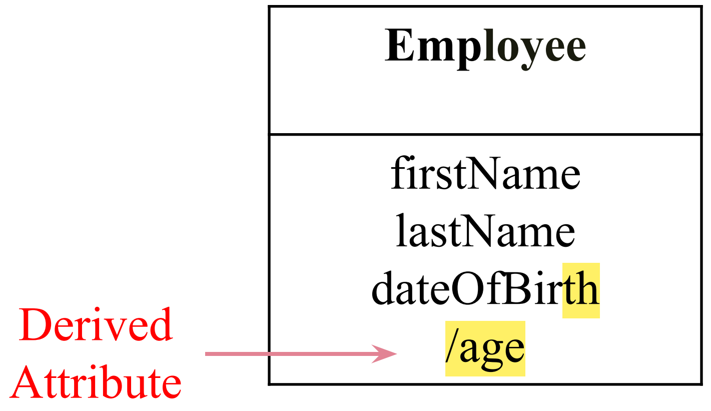
A class describes a set of objects with the same semantics, properties and behavior.

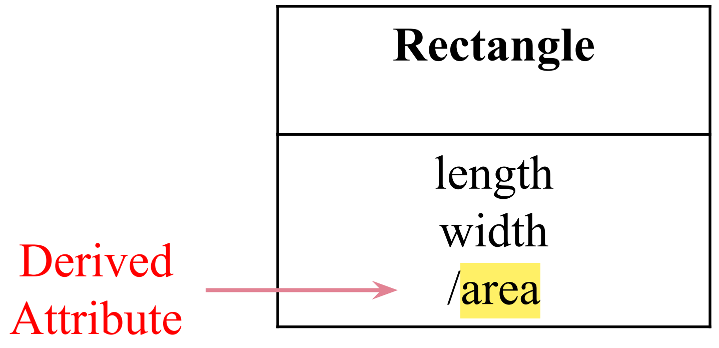
- During a semester a lecturer delivers one or more lectures
- A student usually attends many lectures, ...
- During the semester there will be several exercises...
- Each student is assigned to one particular study group for the whole Semester
- ... it is graded by a tutor



Derived Attribute

A **derived attribute** is an attribute in a class that is not stored directly but is calculated or derived from other attributes or entities in the system.

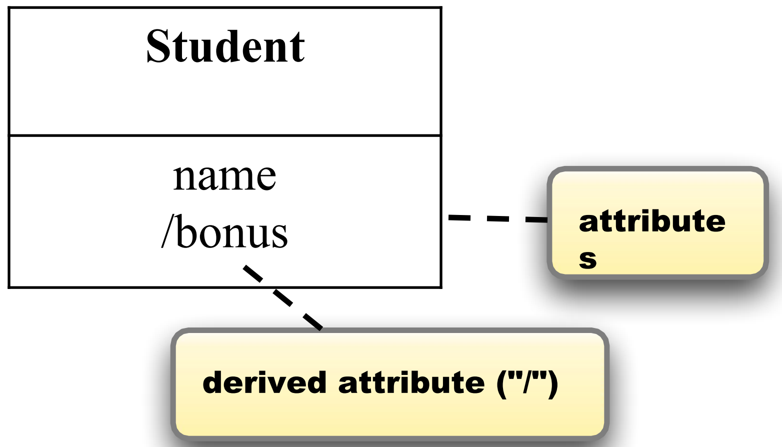




Attributes are logical data values of an object.

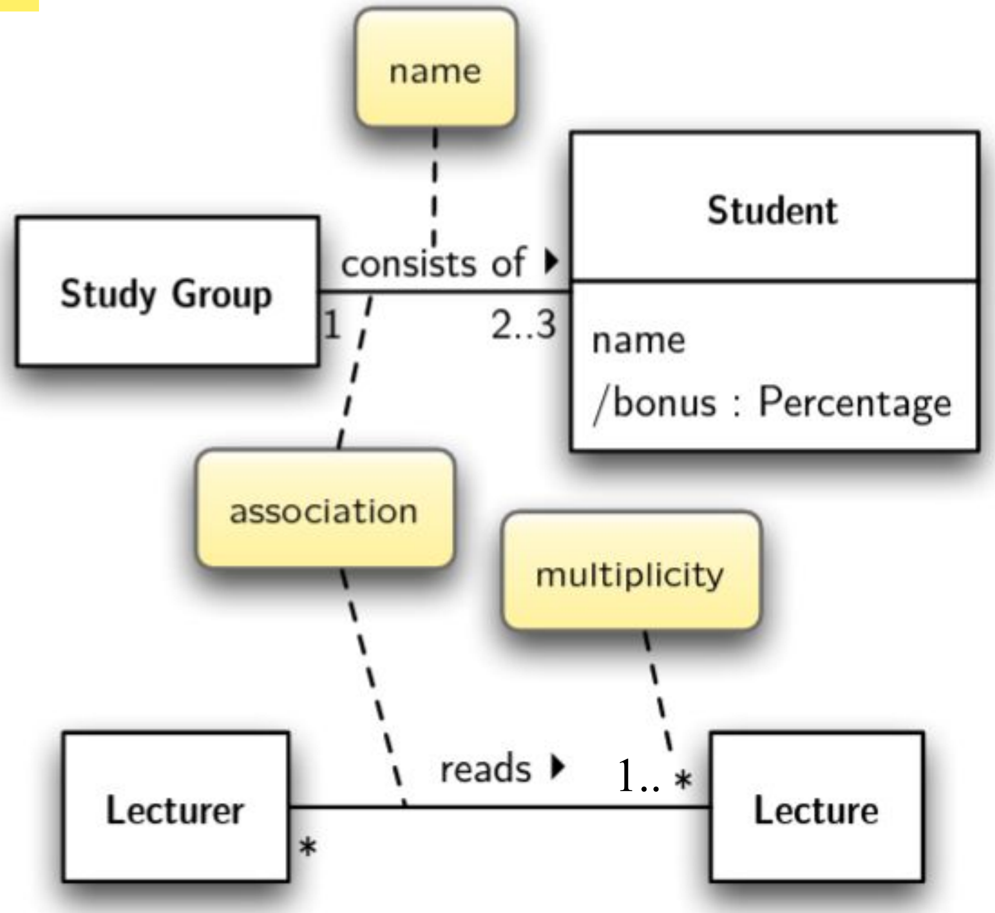
- … after submitting a solution, it is graded by a tutor
- The bonus is a relative bonus that reflects the relative number of exercise points gained during the semester
- …

The bonus is derived



The ends of an association are called roles. Roles optionally have a multiplicity, name and navigabilit

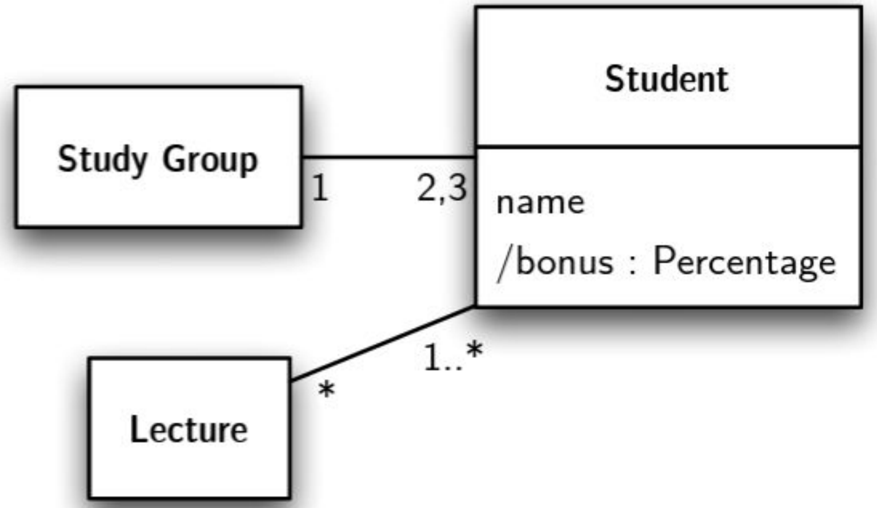
- A lecturer reads one or more lectures...
- A student usually attends many lectures...
- A study group consists of two to three students...
- During the semester there will be several exercises



The multiplicity defines how many instances of a class A can be associated with one instance of a class B at any particular moment.

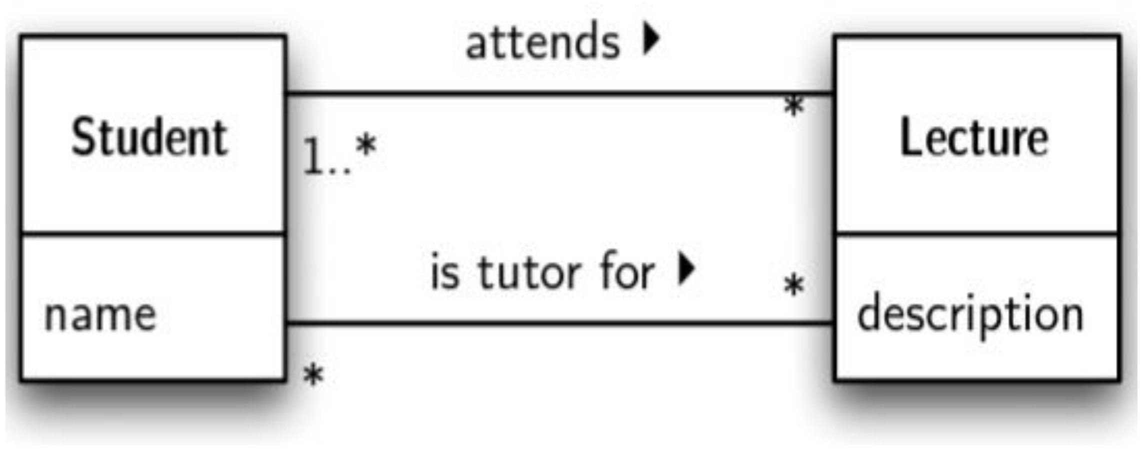
(e.g., * ≙ zero or more; **1..10** between 1 and 10; **1,2** one or two)

- A student usually attends **many** lectures, unless (s)he has something better to do
- A study group consists of **two to three** students...
- ...

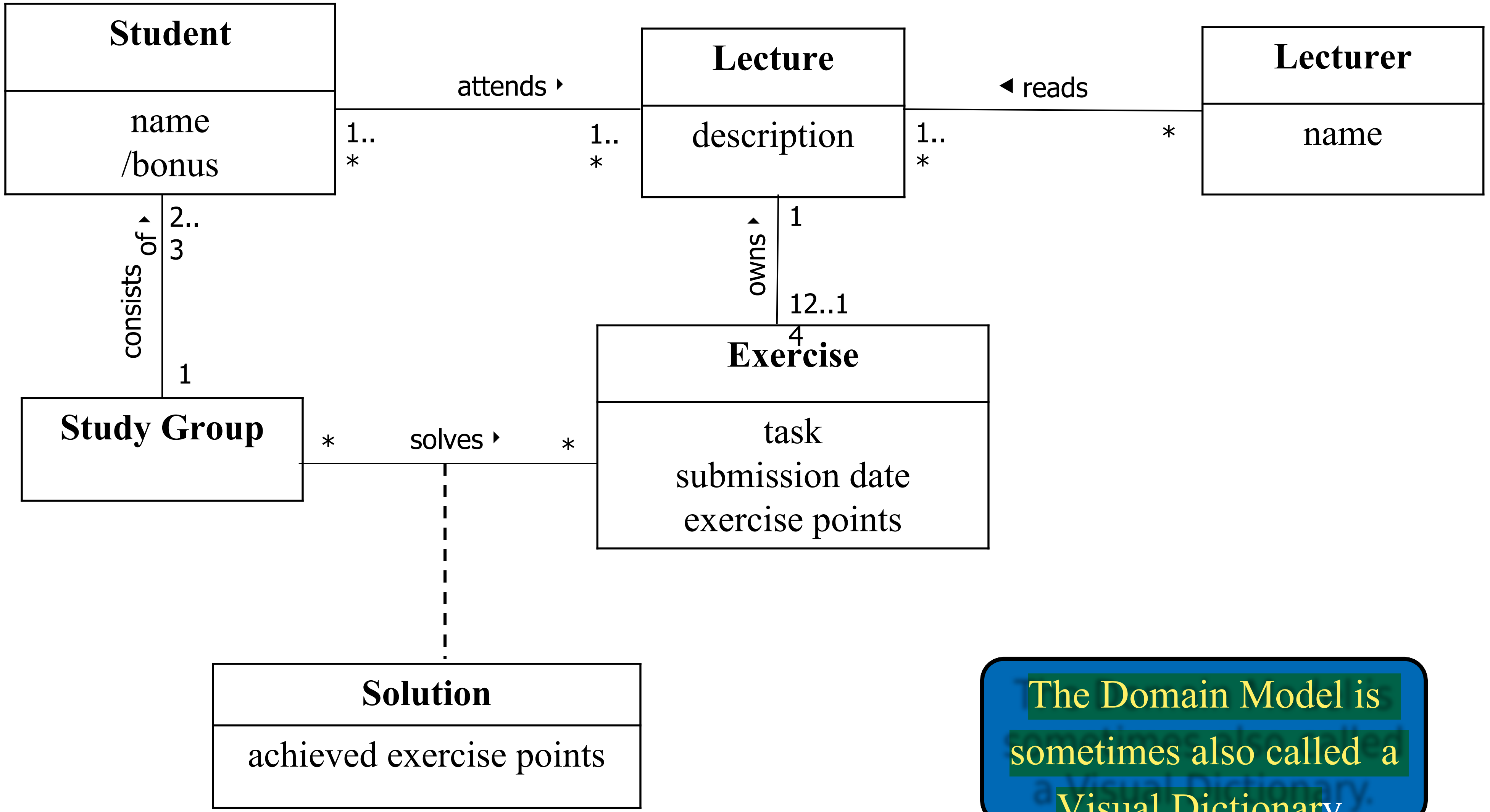


Two Classes can have multiple associations

- A **student** usually **attends many lectures**, unless the student has something better to do
- A study group consists of two to three students; after submitting a solution it is graded by a **tutor who is also a student**



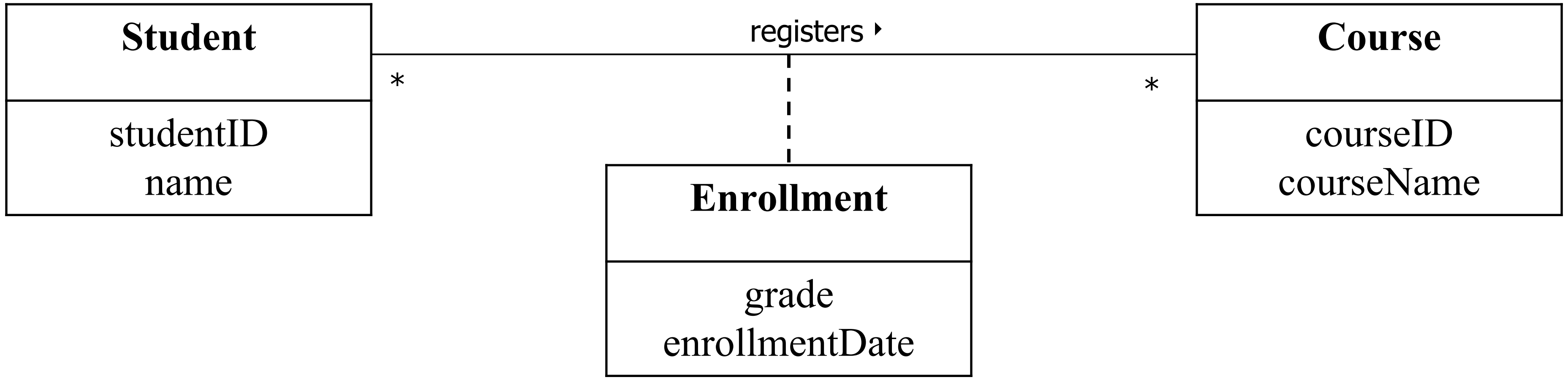
A preliminary domain model for a course management system.



The Domain Model is sometimes also called a Visual Dictionary.

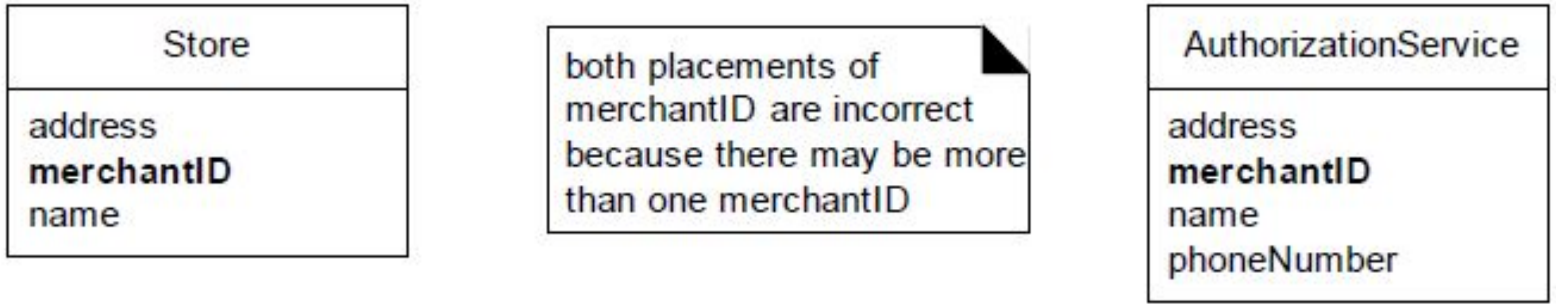
Association Classes

- An association class **represents a relationship between two classes** that also has attributes or operations of its own. It combines both an association and a class into a single concept, allowing you to model a relationship that has its own properties.
- Example: In a course management system, if you have **Student** and **Course** classes, and you want to capture attributes like **grade** or **enrollmentDate** specific to the relationship between a Student and a Course, you might model this with an association class, **Enrollment**.



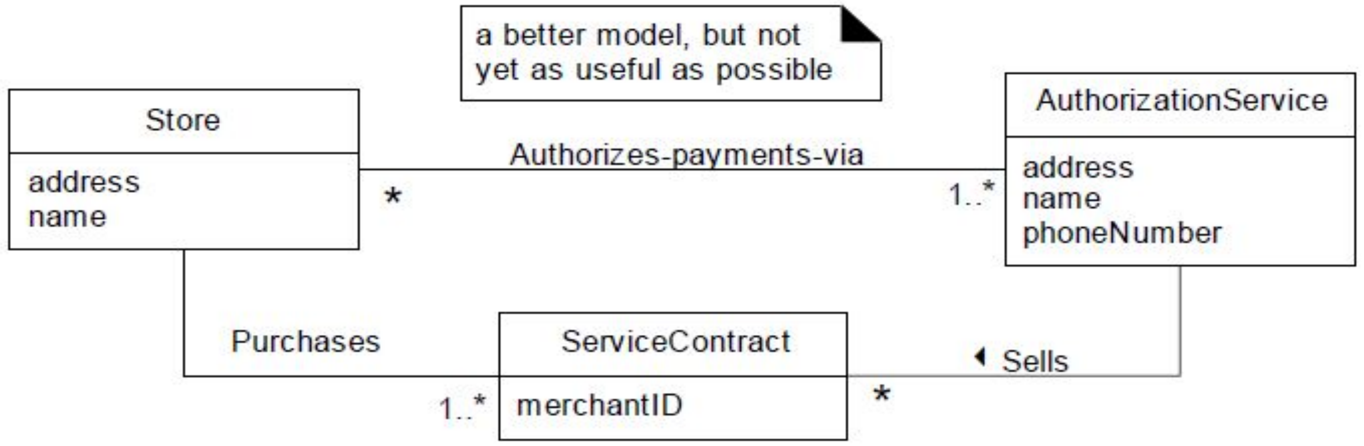
Association Classes

- The following domain requirements set the stage for association classes:
  - *Authorization services assign a merchant ID to each store for identification during communications.*
  - *A payment authorization request from the store to an authorization service needs the merchant ID that identifies the store to the service.*
  - *Furthermore, a store has a different merchant ID for each service.*
- Where in the Domain Model should the merchant ID attribute reside?
- Placing *merchantID* in *Store* is incorrect because a *Store* can have more than one value for *merchantID*. The same is true with placing it in *Authorization-Service*


Store
address
merchantID
name
both placements of merchantID are incorrect because there may be more than one merchantID
AuthorizationService
address
merchantID
name
phoneNumber

Association Classes

- This leads to the following modeling principle:
  - *In a domain model, if a class C can simultaneously have many values for the same kind of attribute A, do not place attribute A in C. Place attribute A in another class that is associated with C.*
- For example:
  - *A Person may have many phone numbers. Place phone number in another class, such as PhoneNumber or ContactInformation, and associate many of these to Person.*

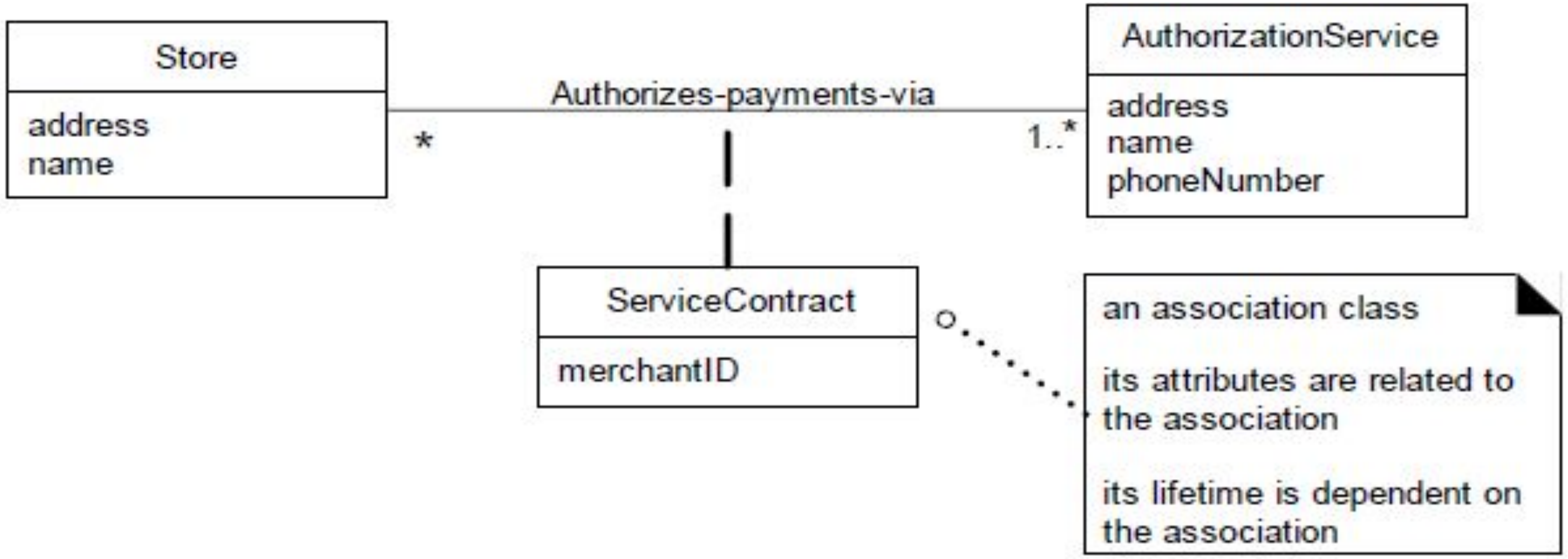


Association Classes

- In the business world, what concept formally records the information related to the services that a service provides to a customer?—a *Contract or Account.*
- The fact that both *Store* and *AuthorizationService* are related to *ServiceContract* is a clue that it is dependent on the relationship between the two.
- The *merchantID* may be thought of as an attribute related to the association between *Store and AuthorizationService.*
- This leads to the notion of an **association class,** in which we can add features to the association itself. *ServiceContract* may be modeled as an association class related to the association between *Store* and *AuthorizationService.*

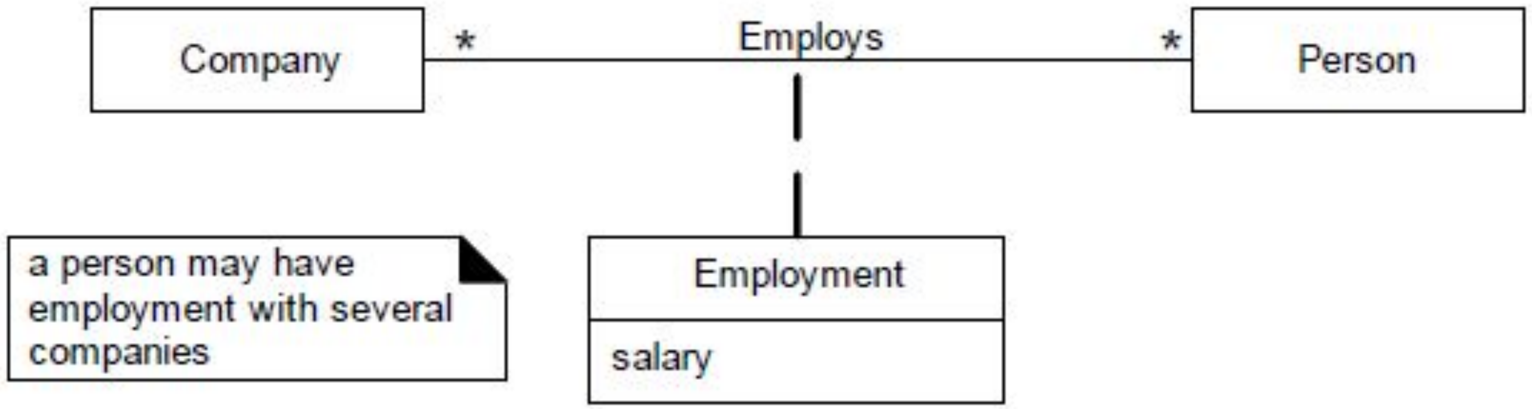
Association Classes

- In the UML, this is illustrated with a dashed line from the association to the association class.
- Figure visually communicates the idea that a *Service-Contract* and its attributes are related to the association between a *Store* and *AuthorizationService,* and that the lifetime of the *ServiceContract* is dependent on the relationship.



Association Classes

- Clues that an association class might be useful in a domain model:
  - *An attribute is related to an association.*
  - *Instances of the association class have a life-time dependency on the association.*
  - *There is a many-to-many association between two concepts, and information associated with the association itself.*

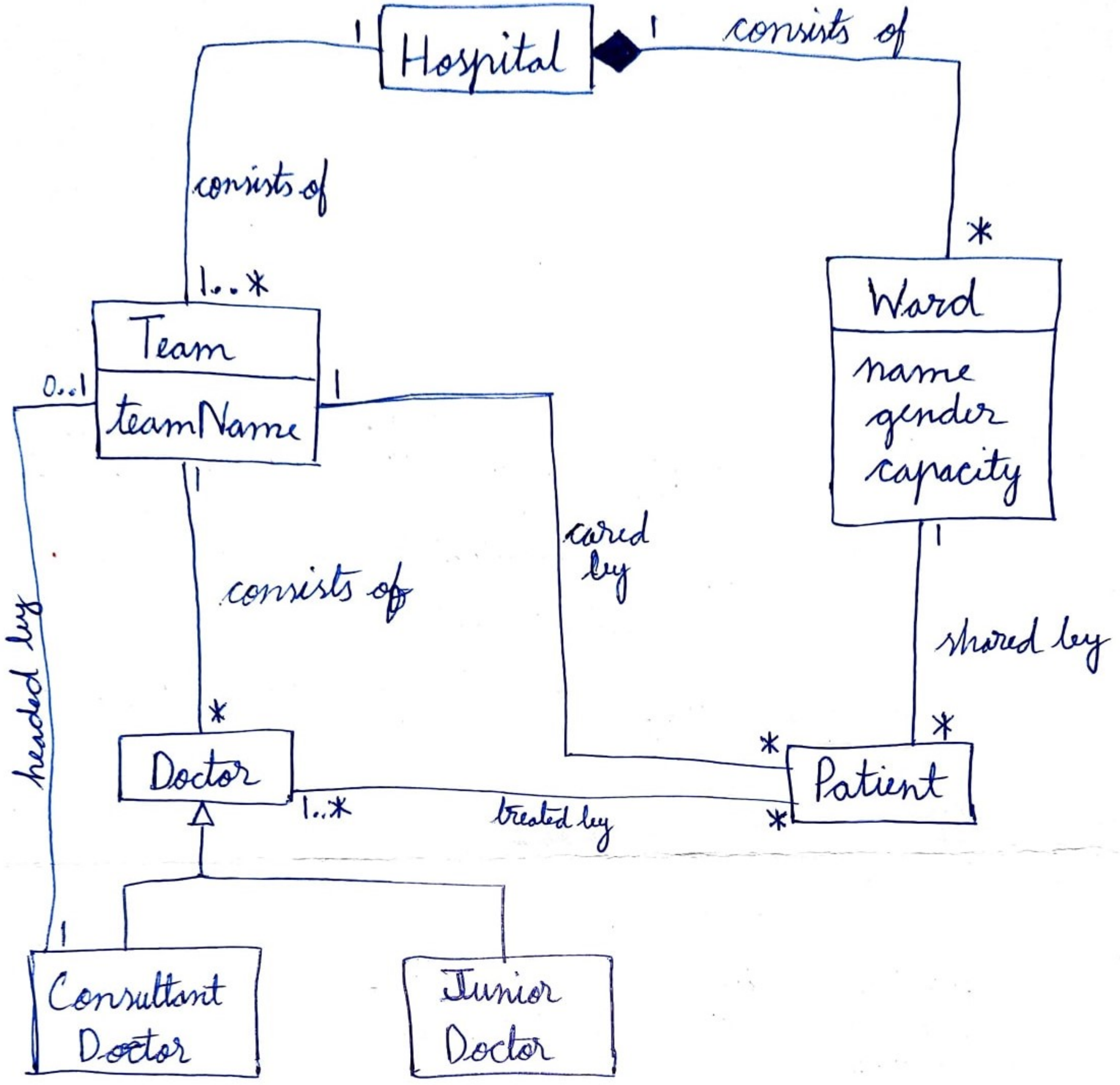


Hospital Management System

- Ward is a division of a hospital shared by patients who need a similar kind of care. In a hospital, there are several wards, each of which may be empty or have in it one or more patients. Each ward has a unique name. Wards are differentiated by gender of its patients, i.e., male wards and female wards. Every ward has a fixed capacity, which is the maximum number of patients that can be on it at one time (i.e., the capacity is the number of beds in the ward).
- Different wards may have different capacities. The doctors in the hospital are organized into teams (also called firms). Each team has a unique name or code (e.g., Orthopedics or Pediatrics) and is headed by a consultant doctor (in the UK, Republic of Ireland, and parts of the Commonwealth)
- Consultant doctor is the senior doctor who has completed all his or her specialist training, residency and practices medicine in a clinic or hospital, in the specialty learned during residency.
- The rest of the team are all junior doctors. Each doctor could be a member of no more than one team. Each patient is in a single ward and is under the care of a single team of doctors. A patient may be treated by any number of doctors, but they must all be in the team that cares for the patient.
- A doctor can treat any number of patients. The team leader accepts ultimate responsibility, legally and otherwise, for the care of all the patients referred to him/her, even with many of the minute-to-minute decisions being made by subordinates.

Hospital Management System

- Ward is a division of a hospital shared by patients who need a similar kind of care. In a hospital, there are several wards, each of which may be empty or have in it one or more patients. Each ward has a unique name. Wards are differentiated by gender of its patients, i.e., male wards and female wards. Every ward has a fixed capacity, which is the maximum number of patients that can be on it at one time (i.e., the capacity is the number of beds in the ward).
- Different wards may have different capacities. The doctors in the hospital are organized into teams (also called firms). Each team has a unique name or code (e.g., Orthopedics or Pediatrics) and is headed by a consultant doctor (in the UK, Republic of Ireland, and parts of the Commonwealth)
- Consultant doctor is the senior doctor who has completed all his or her specialist training, residency and practices medicine in a clinic or hospital, in the specialty learned during residency.
- The rest of the team are all junior doctors. Each doctor could be a member of no more than one team. Each patient is in a single ward and is under the care of a single team of doctors. A patient may be treated by any number of doctors, but they must all be in the team that cares for the patient.
- A doctor can treat any number of patients. The team leader accepts ultimate responsibility, legally and otherwise, for the care of all the patients referred to him/her, even with many of the minute-to-minute decisions being made by subordinates.


Hospital
1
1
consists of
*
Ward
name
gender
capacity
consists of
1..*
Team
teamName
0..1
1
1
cared by
1
shared by
consists of
headed by
*
*
*
Doctor
1..*
treated by
*
Patient
1
Consultant Doctor
Junior Doctor

References

- Satzinger, John W., Robert B. Jackson, and Stephen D. Burd. *Systems analysis and design in a changing world*. 4th Edition.
- Larman, Craig. Applying UML and patterns: an introduction to object oriented analysis and design and interative development. Pearson Education India, 2012.
- Object-Oriented Modeling and Design with UML, Michael R. Blaha and James R. Rumbaugh, 2nd Edition, Pearson, 2005.