Class Diagram

Unified Modeling Language, Version 2.0

- 2 types of diagrams
  - Structure Diagrams
    - Provide a way for representing the data and static relationships that are in an information system
    - Class diagram
  - Behavior Diagrams

Structure Diagrams

- Class Diagram
  - Describe the structure of the system in terms of classes and objects
  - Primary purpose during analysis workflow: to create a vocabulary that is used by both the analyst and users

What is a Class?

- A general template that we use to create specific instances or objects in the application domain
- Represents a kind of person, place, or thing about which the system will need to capture and store information
- Abstractions that specify the attributes and behaviors of a set of objects

What is an Object?

- Entities that encapsulate state and behavior
- Each object has an identity
  - It can be referred individually
  - It is distinguishable from other objects

Types of Classes

- Ones found during analysis:
  - people, places, events, and things about which the system will capture information
  - ones found in application domain
- Ones found during design
  - specific objects like windows and forms that are used to build the system

2 Kinds of Classes during Analysis

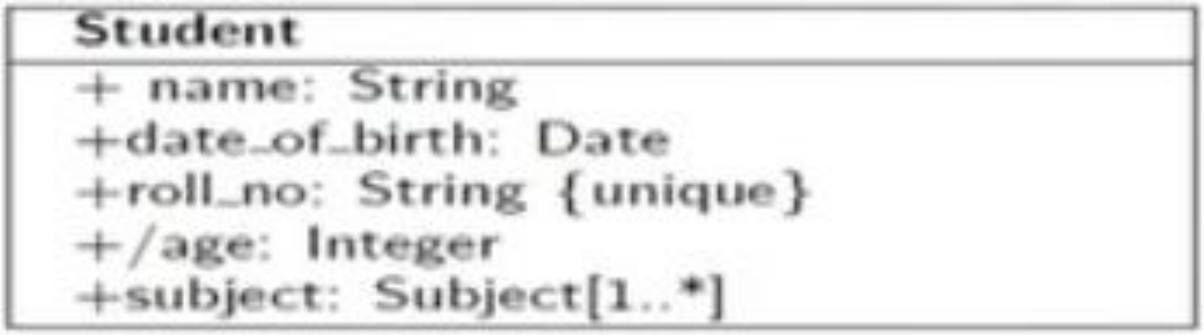
- Concrete
  - Class from application domain
  - Example: Customer class and Employee class
- Abstract
  - Useful abstractions
  - Example: Person class

Attributes in a Class

- Properties of the class about which we want to capture information
- Represents a piece of information that is relevant to the description of the class within the application domain

Attributes in a Class

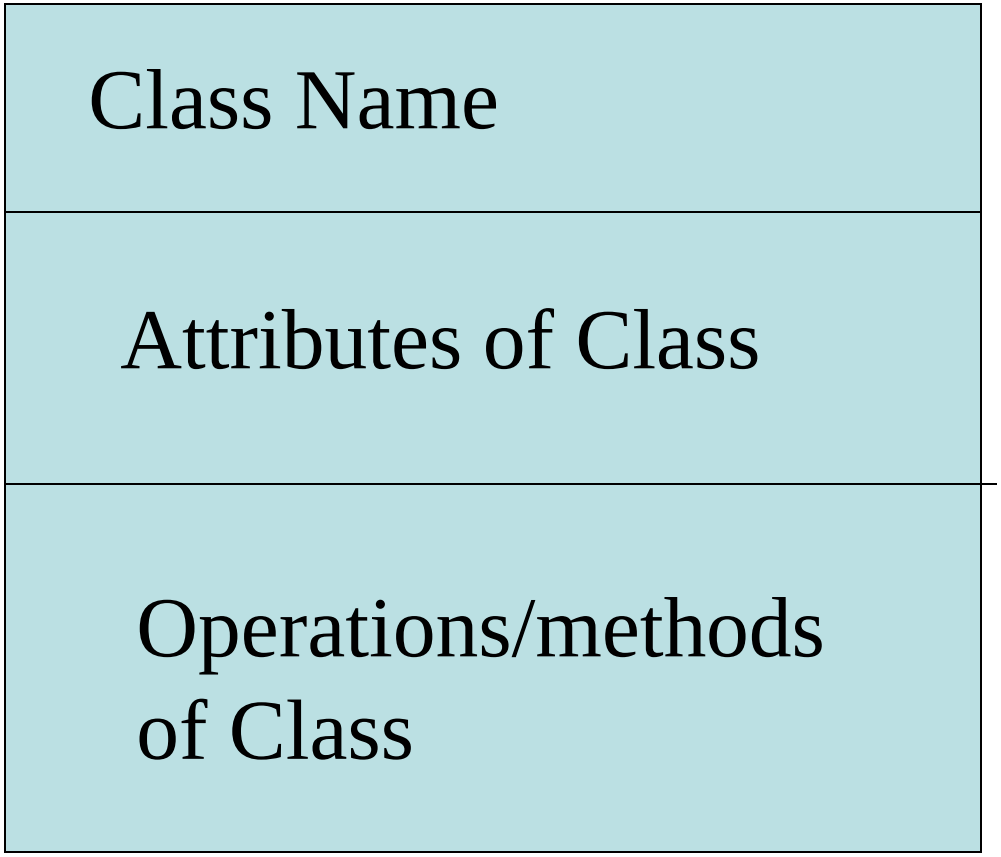
- Only add attributes that are primitive or atomic types
- Derived attribute
  - attributes that are calculated or derived from other attributes
  - denoted by placing slash (/) before name



Operations in a Class

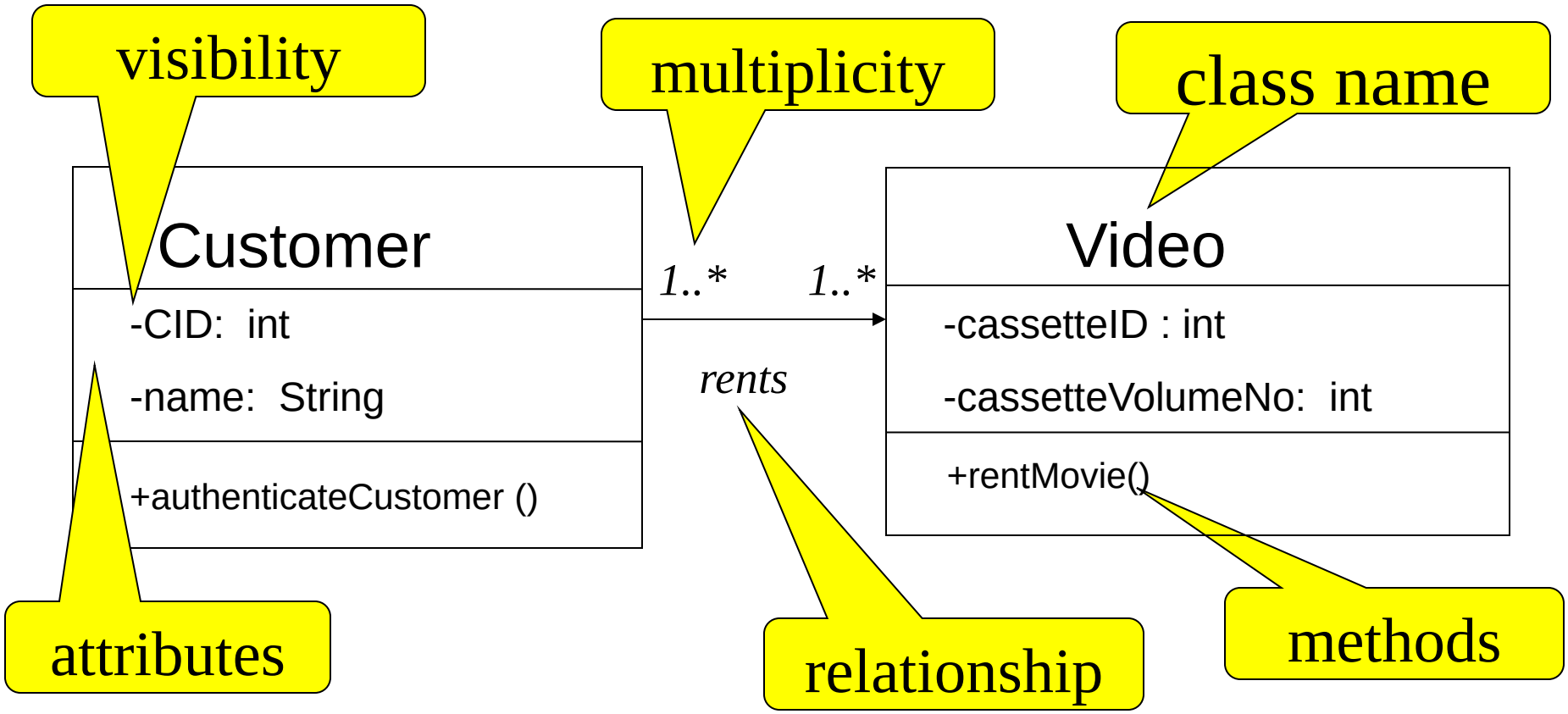
- Represents the actions or functions that a class can perform
- Describes the actions to which the instances of the class will be capable of responding
- Can be classified as a constructor, query, or update operation

UML Representation of Class



Example of a Class Diagram

Video Rental System



Visibility of Attributes and Operations

- Relates to the level of information hiding to be enforced

Visibility	**Symbol**	**Accessible To**
Public	+	All objects within your system.
Protected	#	Instances of the implementing class and its subclasses.
Private	-	Instances of the implementing class.

Relationships among Classes

- Represents a connection between multiple classes or a class and itself
- 3 basic categories:
  - association relationships
  - generalization relationships
  - aggregation relationships

Association Relationship

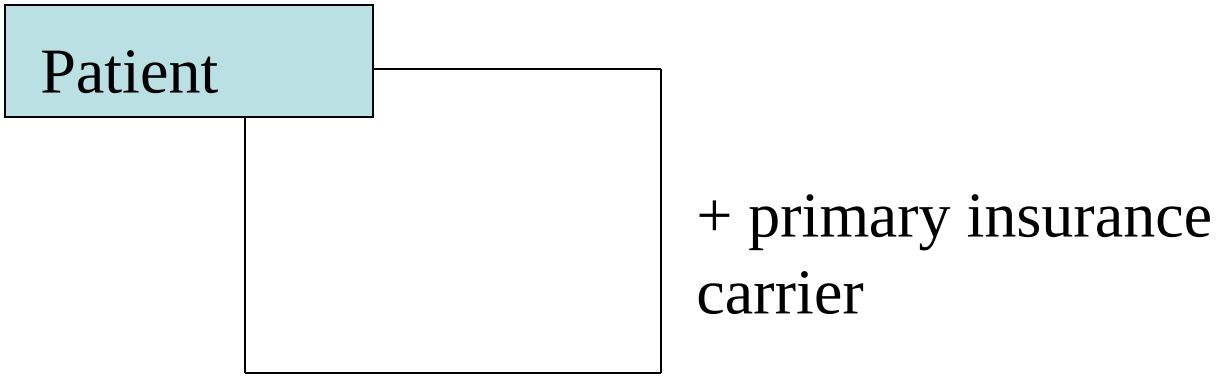
- A bidirectional semantic connection between classes
- Type:
  - name of relationship
  - role that classes play in the relationship

Association Relationship

- Name of relationship type shown by:
  - drawing line between classes
  - labeling with the name of the relationship
  - indicating with a small solid triangle beside the name of the relationship the direction of the association

Association Relationship

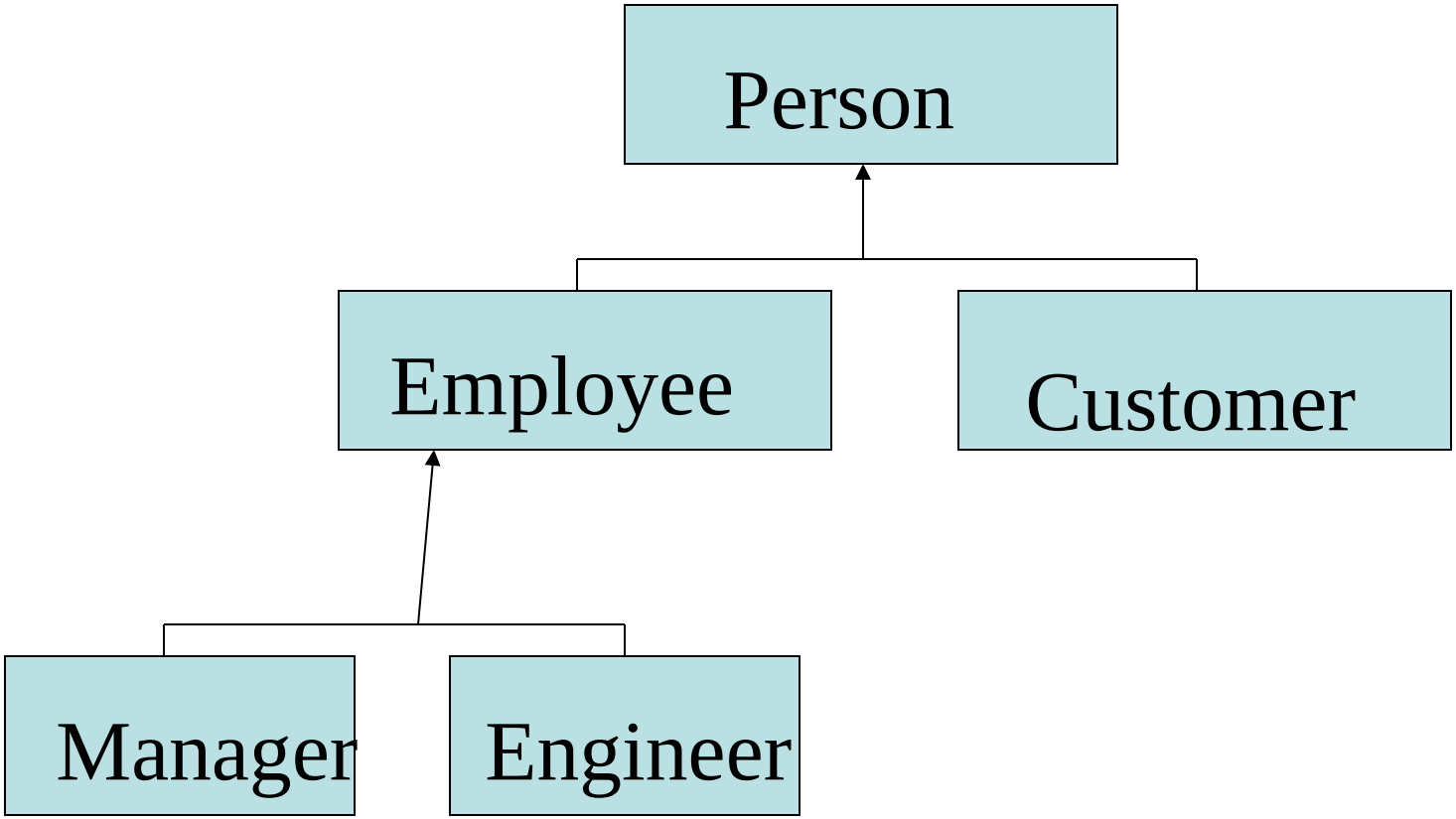
- Role type shown by:
  - drawing line between classes
  - indicating with a plus sign before the role name



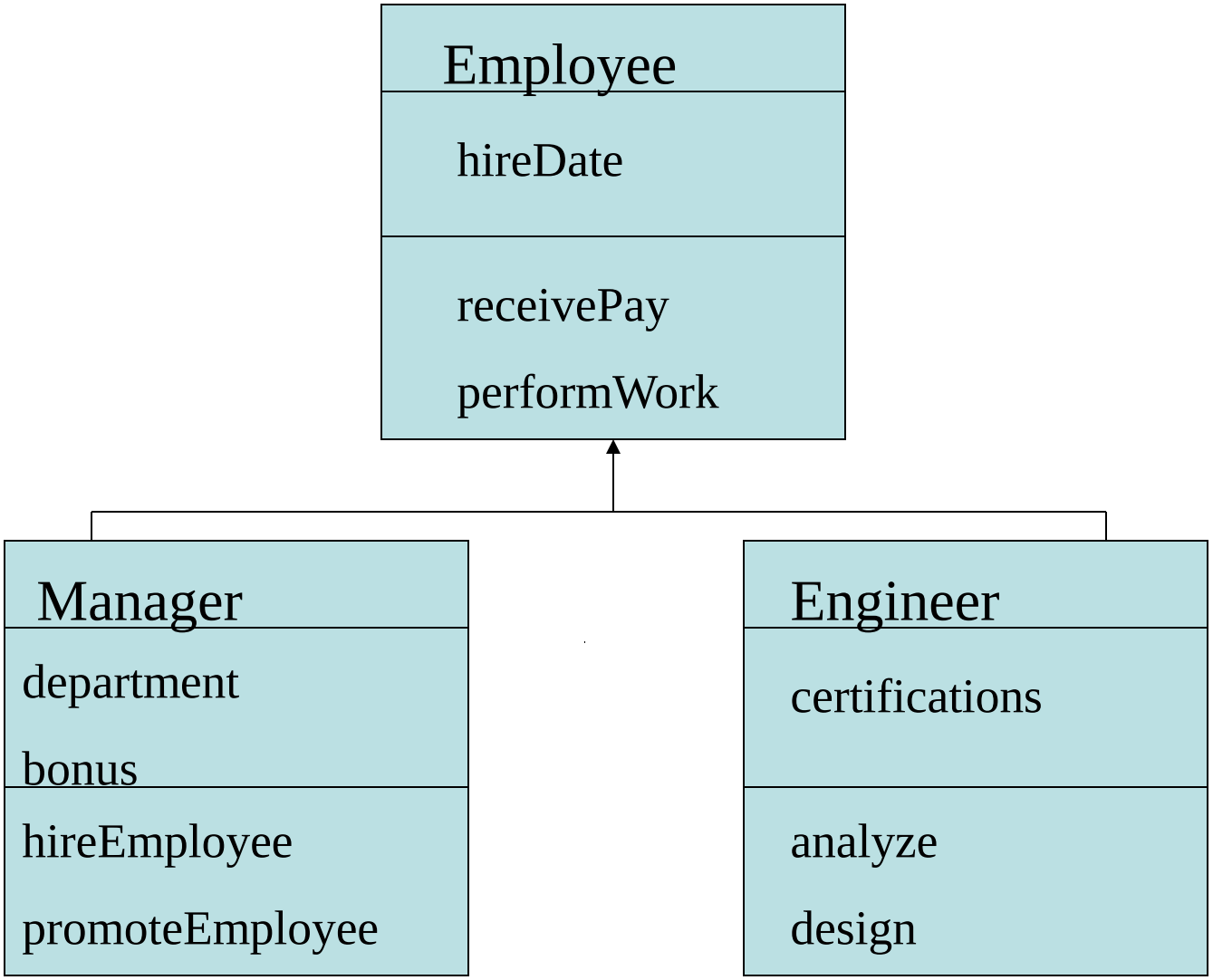
Generalization Relationship

- Enables the analyst to create classes that inherit attributes and operations of other classes
- Represented by *a-kind-of* relationship

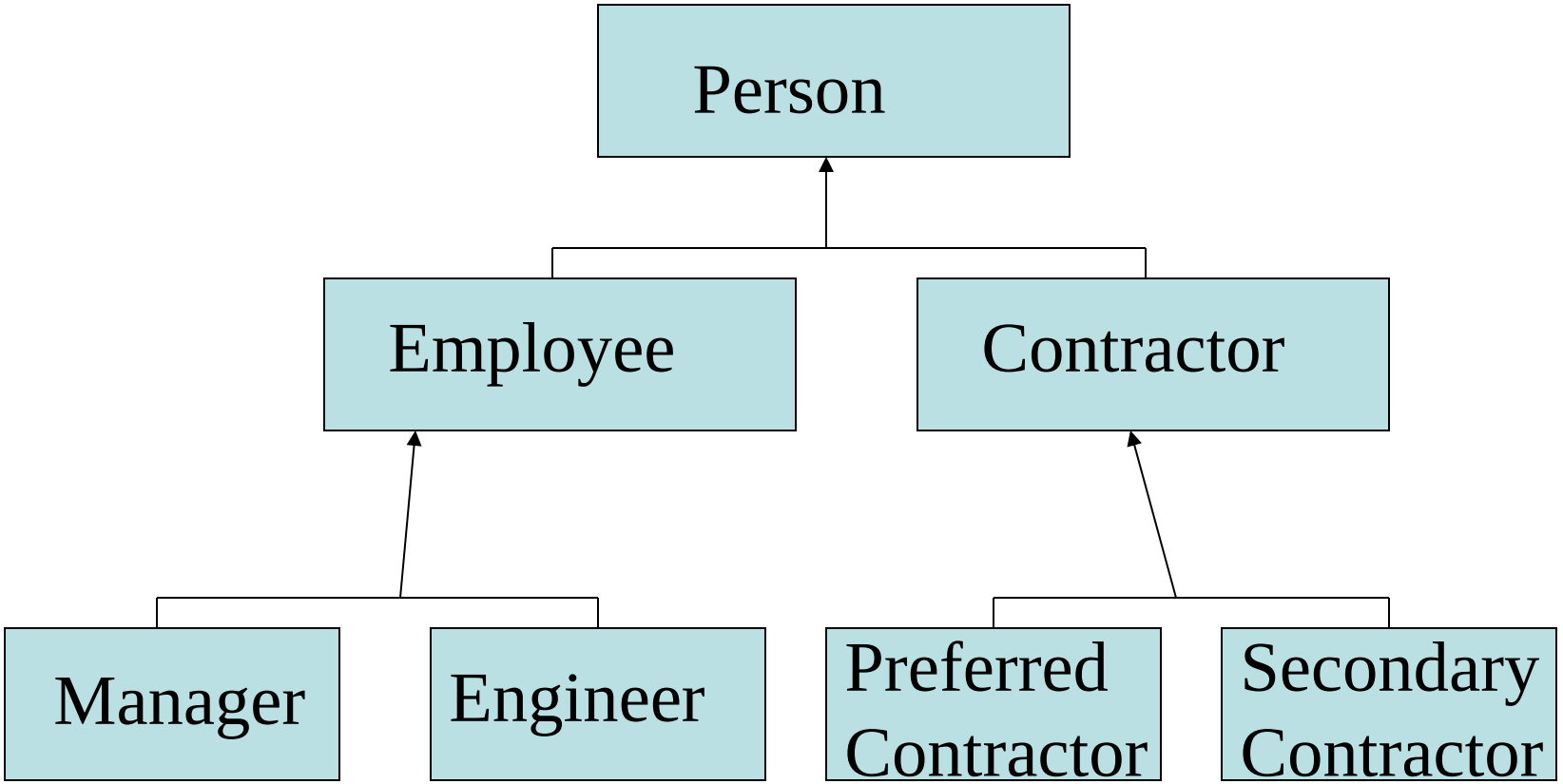
Generalization Relationship



Generalization Relationship

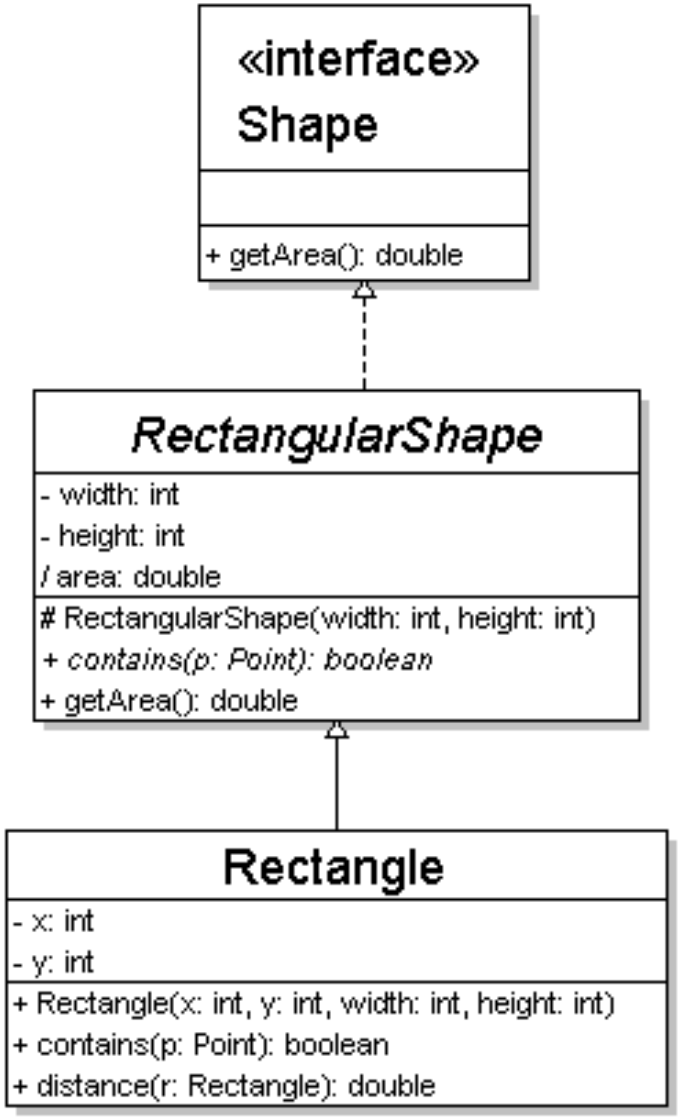


Generalization Relationship



Generalization relationships

- generalization (inheritance) relationships
  - hierarchies drawn top-down with arrows pointing upward to parent
  - line/arrow styles differ, based on whether parent is a(n):
    - class:
      solid line, black arrow
    - abstract class:
      solid line, white arrow
    - interface:
      dashed line, white arrow

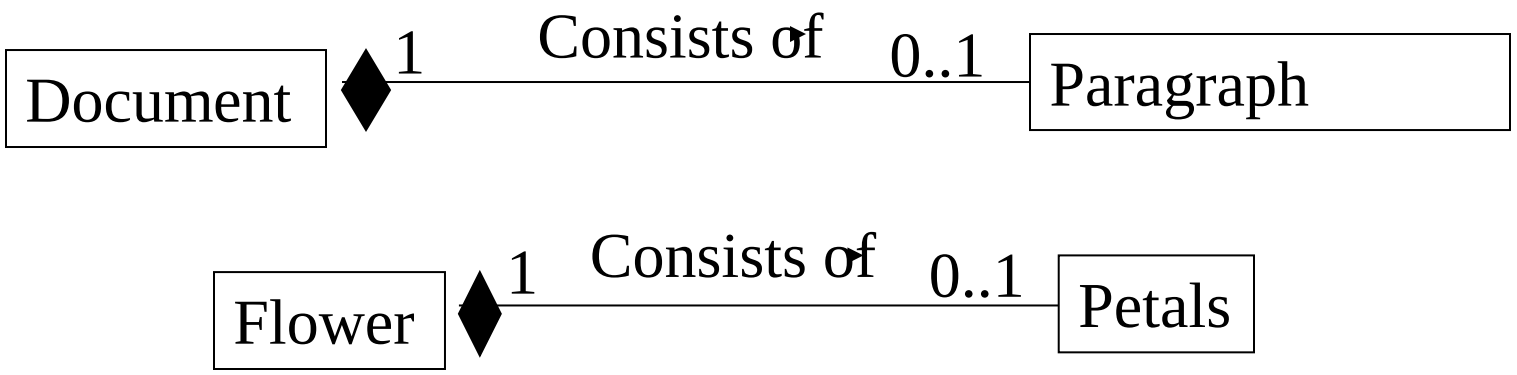


Aggregation Relationship

- Specialized form of association in which a whole is related to its part(s)
- Represented by *a-part-of* relationship

Aggregation Relationship

- “whole-part” or “a- part-of” relationship.
- Objects representing the components of something are associated with an object representing the entire assembly.
- Tightly coupled form of association.
- Denoted by placing a diamond nearest the class representing the aggregation



versus Composition

- **Composition** is a form of aggregation with additional constraints:
  - A constituent part can belong to **at most one** assembly (whole).
    - it has a coincident lifetime with the assembly.
    - Deletion of an assembly object triggers automatically a deletion of all constituent objects via composition.
  - Composition implies ownership of the parts by the whole.
    - Parts cannot be shared by different wholes.

Example

- Whole / Part relationships
  - Say, we model Flower HAS_A Petal
  - Flower contains many Petals
  - Flower is the Whole, Petal is the Part
  - Depicted as:

  - Physical Containment – Composition / Strong Aggregation
- Member relationship
  - Say, we model Library HAS Users
  - Library enrolls many Users
  - Library does not contain the Users
  - Depicted as:

  - Conceptual Containment – Weak Aggregation

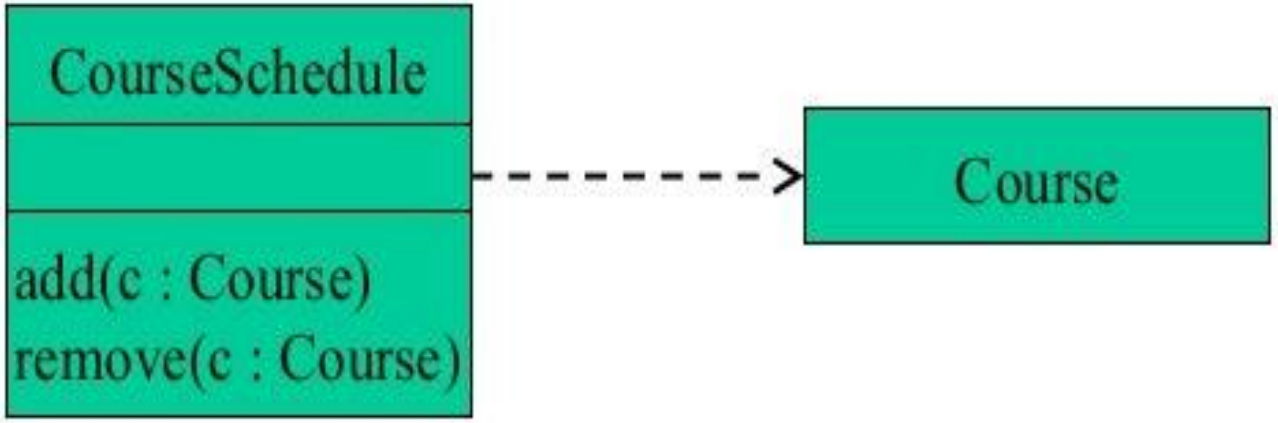
Dependency Relationship

- Two or more elements are dependent on each other.

- If we make a change to a particular element, then it is likely possible that all the other elements will also get affected by the change.

Dependency Relationships

A ***dependency*** indicates a semantic relationship between two or more elements. The dependency from *CourseSchedule* to *Course* exists because *Course* is used in both the **add** and **remove** operations of *CourseSchedule*.



Multiplicity

- Documents how many instances of a class can be associated with one instance of another class

Multiplicity

- Denotes the **minimum number.. maximum number** of instances

Exactly one	1		
Zero or more	0..*	or	0..m
One or more	1..*	or	1..m
Zero or one	0..1		
Specified range	2..4		
Multiple, disjoint ranges			1..3, 5

Class Diagram

- 2 approaches to deriving class diagrams:
  - from use cases and their scenarios
  - from Class-responsibility-collaboration (CRC) cards
    - For each class, developer fills in an index card that shows the name of the class, its functionality (responsibility), and a list of other classes it invokes to achieve that functionality (collaboration)
    - Also includes the attributes of the class

Guidelines for Analyzing Use Cases

- A common or improper noun implies a class of objects
- A proper noun implies an instance of a class
- A collective noun implies a class of objects made up of groups of instances of another class

Guidelines for Analyzing Use Cases (2)

- An adjective implies an attribute of an object
- A doing verb implies an operation
- A being verb implies a relationship between an object and its class
- A having verb implies an aggregation or association relationship

Class Diagram

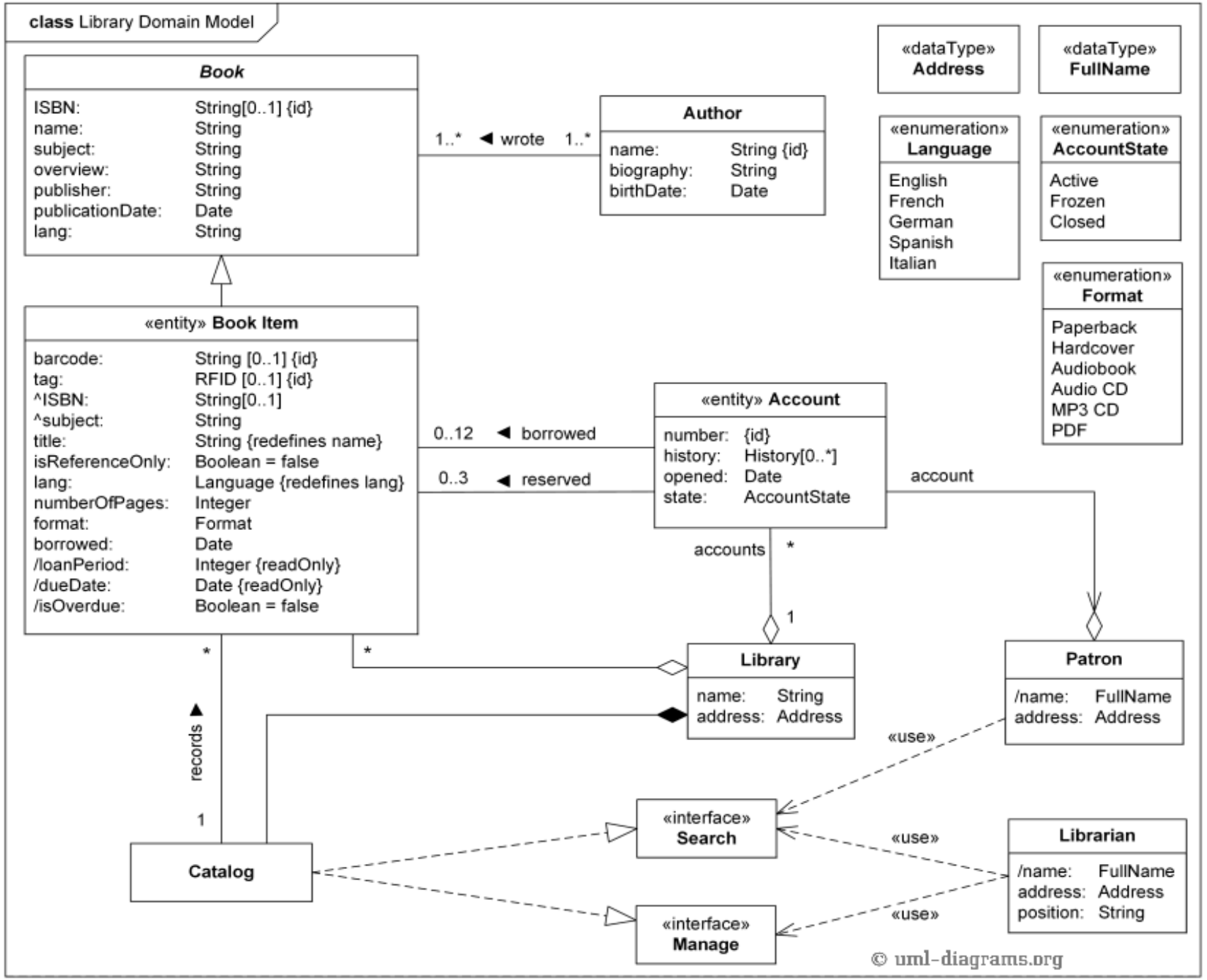
- Ensure that the classes are both necessary and sufficient to solve the underlying problem
  - no missing attributes or methods in each class
  - no extra or unused attributes or methods in each class
  - no missing or extra classes

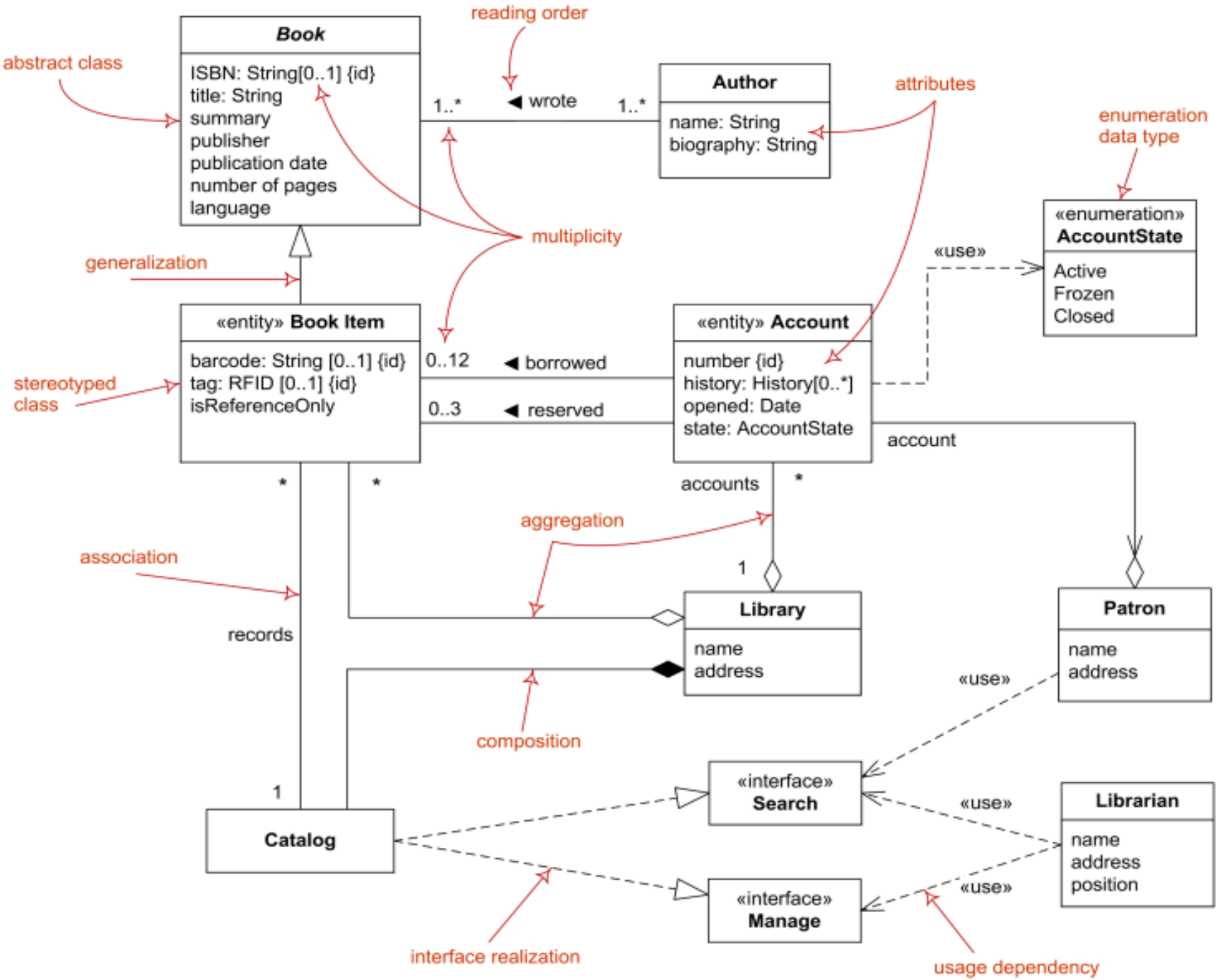
Library Management System

- Each physical library item - book, tape cassette, CD, DVD, etc. could have its own item number.
- To support it, the items may be **barcoded**.
- The purpose of barcoding is to provide a unique and scannable identifier that links the barcoded physical item to the electronic record in the **catalog**.

Library Management System

- Barcode must be physically attached to the item, and barcode number is entered into the corresponding field in the electronic item record.
- Barcodes on library items could be replaced by **RFID tags**.
- The RFID tag can contain item's identifier, title, material type, etc.
- It is read by an RFID reader, without the need to open a book cover or CD/DVD case to scan it with barcode reader.


class Library Domain Model
Book
ISBN: String[0..1] {id}
name: String
subject: String
overview: String
publisher: String
publicationDate: Date
lang: String
1..* wrote 1..*
Author
name: String {id}
biography: String
birthDate: Date
«dataType» Address
«dataType» FullName
«enumeration» Language
English
French
German
Spanish
Italian
«enumeration» AccountState
Active
Frozen
Closed
«enumeration» Format
Paperback
Hardcover
Audiobook
Audio CD
MP3 CD
PDF
«entity» Book Item
barcode: String [0..1] {id}
tag: RFID [0..1] {id}
^ISBN: String[0..1]
^subject: String
title: String {redefines name}
isReferenceOnly: Boolean = false
lang: Language {redefines lang}
numberOfPages: Integer
format: Format
borrowed: Date
/loanPeriod: Integer {readOnly}
/dueDate: Date {readOnly}
/isOverdue: Boolean = false
0..12 borrowed
0..3 reserved
«entity» Account
number: {id}
history: History[0..*]
opened: Date
state: AccountState
account
accounts
*
1
*
*
records
1
Library
name: String
address: Address
Patron
/name: FullName
address: Address
«use»
«interface» Search
«use»
Librarian
/name: FullName
address: Address
position: String
Catalog
«interface» Manage
«use»
© uml-diagrams.org


Book
ISBN: String[0..1] {id}
title: String
summary
publisher
publication date
number of pages
language
abstract class
reading order
wrote
1..*
1..*
Author
name: String
biography: String
attributes
enumeration data type
«enumeration» AccountState
Active
Frozen
Closed
«use»
multiplicity
generalization
«entity» Book Item
barcode: String [0..1] {id}
tag: RFID [0..1] {id}
isReferenceOnly
stereotyped class
0..12
borrowed
0..3
reserved
«entity» Account
number {id}
history: History[0..*]
opened: Date
state: AccountState
account
accounts
*
*
*
aggregation
association
1
Library
name
address
Patron
name
address
records
composition
«use»
1
«interface» Search
«use»
Librarian
name
address
position
Catalog
«interface» Manage
«use»
interface realization
usage dependency

Automated Teller Machine Example

- Design the software to support a computerized banking network including both human cashiers and automatic teller machines to be shared by a consortium of banks.
- Each bank provides its own computer to maintain its own accounts and process transactions against them.
- Cashier stations are owned by individual banks and communicate directly with their own banks computers.
- Human cashiers enter account and transaction data.
- ATMs communicate with a central computer which clears transactions with the appropriate banks.

Automated Teller Machine Example

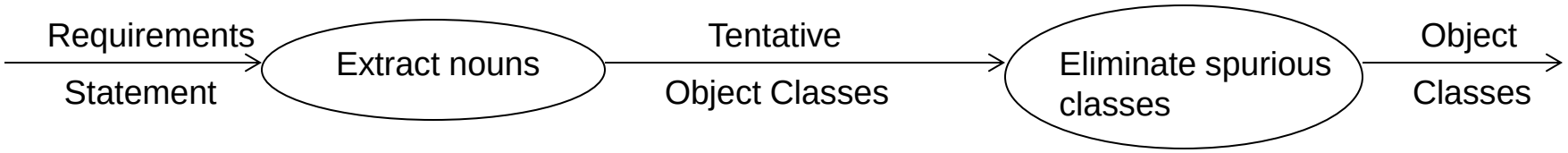
- An ATM accepts a cash card, interacts with user, communicate with the central system to carry out the transaction, dispenses cash, and prints receipts.
- The system requires appropriate recordkeeping and security provisions.
- The system must handle concurrent accesses to the same account correctly.
- The bank will provide their own software for their own computer; you are to design the software for the ATM and the network.
- The cost of the shared system will be apportioned to the banks according to the number of customers with cash cards.

Class Diagram

- Steps performed in constructing a class diagram
  1. Identify objects and classes.
  2. Keep the right classes
  3. Prepare data dictionary.
  4. Identify associations between objects.
  5. Keep the right associations
  6. Identify attributes of objects and links.
  7. Organize and simplify object classes using inheritance.
  8. Verify that access path exist for likely queries.
  9. Iterate and refine the model.

1. Identify classes and objects

- Objects include physical entities.
  - Houses, employees, machines.
- All classes must make sense in the application domain.
- Avoid computer implementation construct.
  - Linked list, subroutines.
- Not all classes are explicit in the problem statement.
- Some are implicit in the application domain or general knowledge.
- classes often correspond to nouns.



1. Identify classes and objects

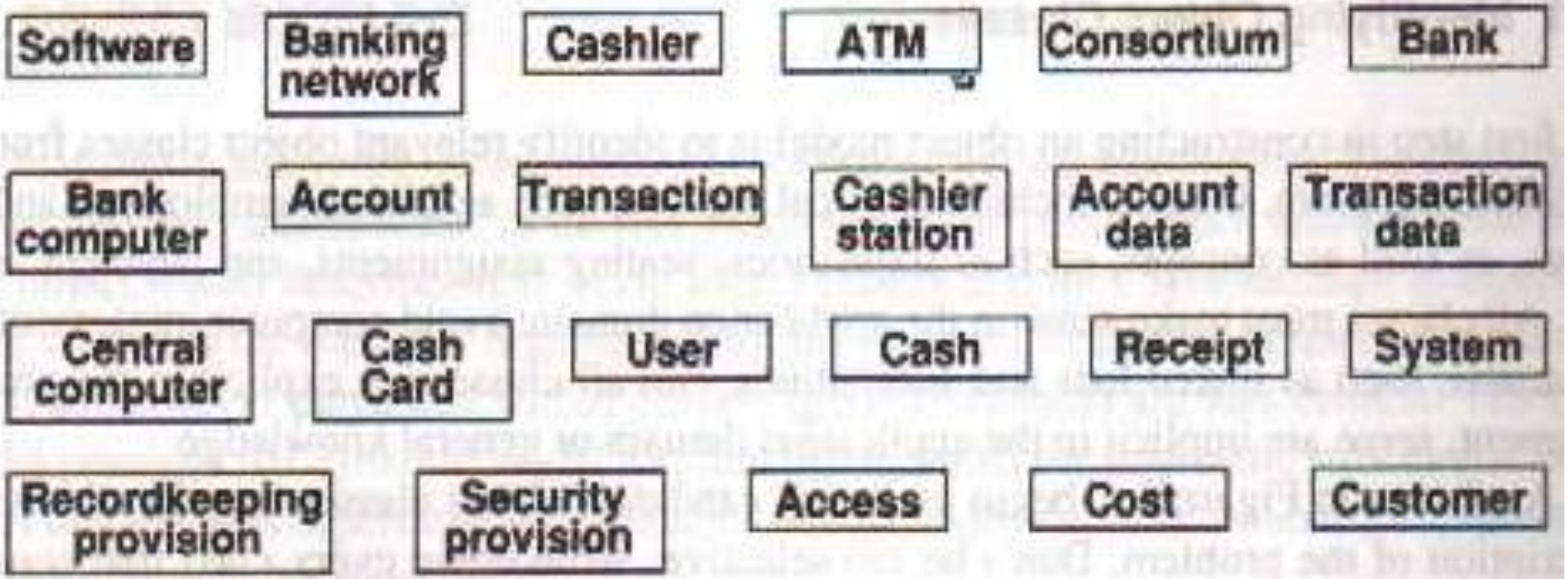


Figure 8.5 ATM classes extracted from problem statement nouns

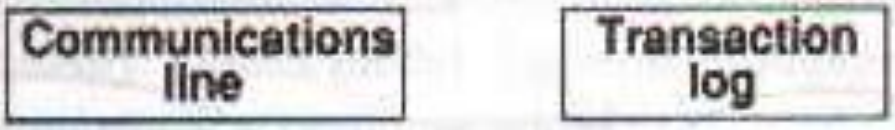


Figure 8.6 ATM classes identified from knowledge of problem domain

2. Keep the Right Classes

- Discard unnecessary and incorrect classes.

- Redundant classes
  - If two classes express the same information the most descript name should be kept.
    - Eg : customer and user are redundant.
      customer is retained
- Irrelevant classes
  - If a class has little or nothing to do with the problem, it should be eliminated.
    - Eg : cost is outside the scope of the ATM transaction software.

2. Keep the Right Classes

- Discard unnecessary and incorrect classes.

- Vague classes
  - A classes should be specific.
    - Eg : recordkeeping provision is part of transaction.
- Attributes
  - Names that primarily describe individual objects should be related as attributes.
    - Eg : account data, receipt, cash

2. Keep the Right Classes

- Operations
  - If a name describes an operation that is applied to objects and not manipulated in its own right, then it is not a class.
    - Eg : if we are simply building telephone, then call is a part of dynamic model.
- Roles
  - The name of the class should reflect its intrinsic nature and not a role that it plays in an association.
    - Eg : In a car manufacturer's database owner, driver, lessee are role names and person is a class.
- Implementation constructs
  - Constructs extraneous to the real world should be eliminated from the analysis model.

2. Keep the Right Classes

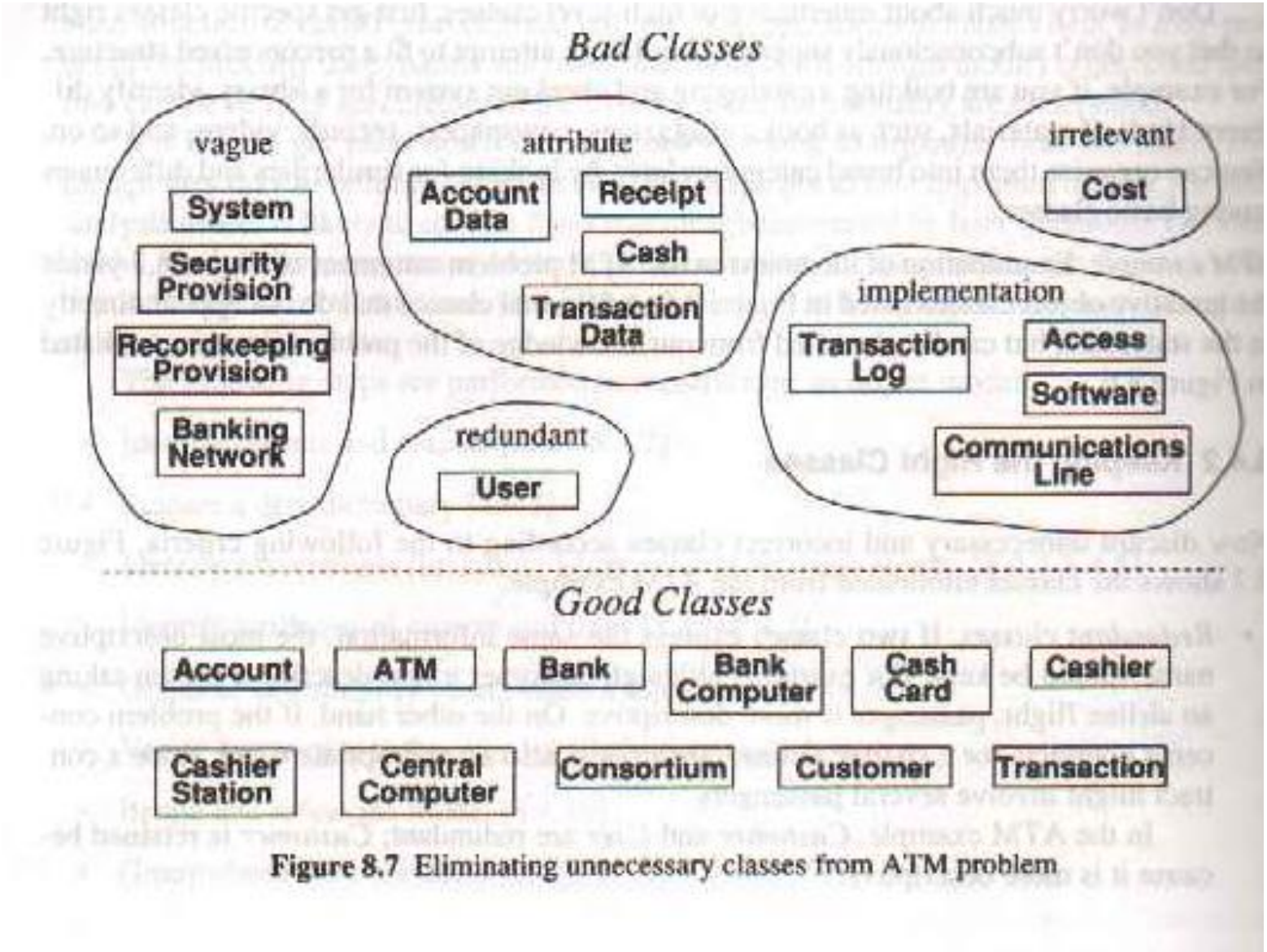


Figure 8.7 Eliminating unnecessary classes from ATM problem

3. Prepare a Data Dictionary.

Account—a single account in a bank against which transactions can be applied. Accounts may be of various types, at least checking or savings. A customer can hold more than one account.

ATM—a station that allows customers to enter their own transactions using cash cards as identification. The ATM interacts with the customer to gather transaction information, sends the transaction information to the central computer for validation and processing, and dispenses cash to the user. We assume that an ATM need not operate independently of the network.

Bank—a financial institution that holds accounts for customers and that issues cash cards authorizing access to accounts over the ATM network.

Bank computer—the computer owned by a bank that interfaces with the ATM network and the bank's own cashier stations. A bank may actually have its own internal network of computers to process accounts, but we are only concerned with the one that talks to the network.

Cash card—a card assigned to a bank customer that authorizes access of accounts using an ATM machine. Each card contains a bank code and a card number, most likely coded in accordance with national standards on credit cards and cash cards. The bank code uniquely identifies the bank within the consortium. The card number determines the accounts that the card can access. A card does not necessarily access all of a customer's accounts. Each cash card is owned by a single customer, but multiple copies of it may exist, so the possibility of simultaneous use of the same card from different machines must be considered.

3. Prepare a Data Dictionary.

Account—a single account in a bank against which transactions can be applied. Accounts may be of various types, at least checking or savings. A customer can hold more than one account.

ATM—a station that allows customers to enter their own transactions using cash cards as identification. The ATM interacts with the customer to gather transaction information, sends the transaction information to the central computer for validation and processing, and dispenses cash to the user. We assume that an ATM need not operate independently of the network.

Bank—a financial institution that holds accounts for customers and that issues cash cards authorizing access to accounts over the ATM network.

Bank computer—the computer owned by a bank that interfaces with the ATM network and the bank's own cashier stations. A bank may actually have its own internal network of computers to process accounts, but we are only concerned with the one that talks to the network.

Cash card—a card assigned to a bank customer that authorizes access of accounts using an ATM machine. Each card contains a bank code and a card number, most likely coded in accordance with national standards on credit cards and cash cards. The bank code uniquely identifies the bank within the consortium. The card number determines the accounts that the card can access. A card does not necessarily access all of a customer's accounts. Each cash card is owned by a single customer, but multiple copies of it may exist, so the possibility of simultaneous use of the same card from different machines must be considered.

Cashier—an employee of a bank who is authorized to enter transactions into cashier stations and accept and dispense cash and checks to customers. Transactions, cash, and checks handled by each cashier must be logged and properly accounted for.

Cashier station—a station on which cashiers enter transactions for customers. Cashiers dispense and accept cash and checks; the station prints receipts. The cashier station communicates with the bank computer to validate and process the transactions.

Central computer—a computer operated by the consortium which dispatches transactions between the ATMs and the bank computers. The central computer validates bank codes but does not process transactions directly.

Consortium—an organization of banks that commissions and operates the ATM network. The network only handles transactions for banks in the consortium.

Customer—the holder of one or more accounts in a bank. A customer can consist of one or more persons or corporations; the correspondence is not relevant to this problem. The same person holding an account at a different bank is considered a different customer.

4. Identify Associations between objects

- Any dependency between two or more classes is an association.
- A reference from one class to another is an association.
- Associations often correspond to verb phrases.

Verb phrases:

Banking network includes cashiers and ATMs
Consortium shares ATMs
Bank provides bank computer
Bank computer maintains accounts
Bank computer processes transaction against account
Bank owns cashier station
Cashier station communicates with bank computer
Cashier enters transaction for account
ATMs communicate with central computer about transaction
Central computer clears transaction with bank
ATM accepts cash card
ATM interacts with user
ATM dispenses cash
ATM prints receipts
System handles concurrent access
Banks provide software
Cost apportioned to banks

Implicit verb phrases:

Consortium consists of banks
Bank holds account
Consortium owns central computer
System provides recordkeeping
System provides security
Customers have cash cards

Knowledge of problem domain:

Cash card accesses accounts
Bank employs cashiers

Figure 8.9 Associations from ATM problem statement

5. Keep the Right Associations

- Association between eliminated classes
  - If one of the classes in the association has been eliminated, then the association must be eliminated or restated in terms of other classes.
    - Eg : eliminate ATM prints receipts, ATM dispenses cash
- Irrelevant or implementation associations
  - Eliminate any associations that are outside the problem domain or deal with implementation constructs.
    - Eg : system handles concurrent access is an implementation concept.
- Actions
  - An association should describe a structural property of the application domain, not a transient event.
    - Eg : ATM accepts cash card - describes part of the interaction cycle between an

      ATM and a customer.

5. Keep the Right Associations

- Ternary associations
  - Most associations between three or more classes can be decomposed into binary associations.
- Derived associations
  - Omit associations that can be defined in terms of other associations.
  - Multiple paths between classes often indicate derived associations which are composition of primitive associations.
    - Eg : consortium shares ATMs is a composition of the associations consortium owns central computer and central computer communicates with ATMs.

Semantics of Associations

- Misnamed associations
  - Names are important to understand and should be chosen with great care.
- Role names
  - An association between two instances of the same class requires role names to distinguish the instances.
- Qualified associations
  - A qualifier distinguishes objects on the “many” side of an association.
- Multiplicity
  - Multiplicity often changes during analysis.
- Missing associations
  - Add any missing associations that are discovered.

Initial Diagram

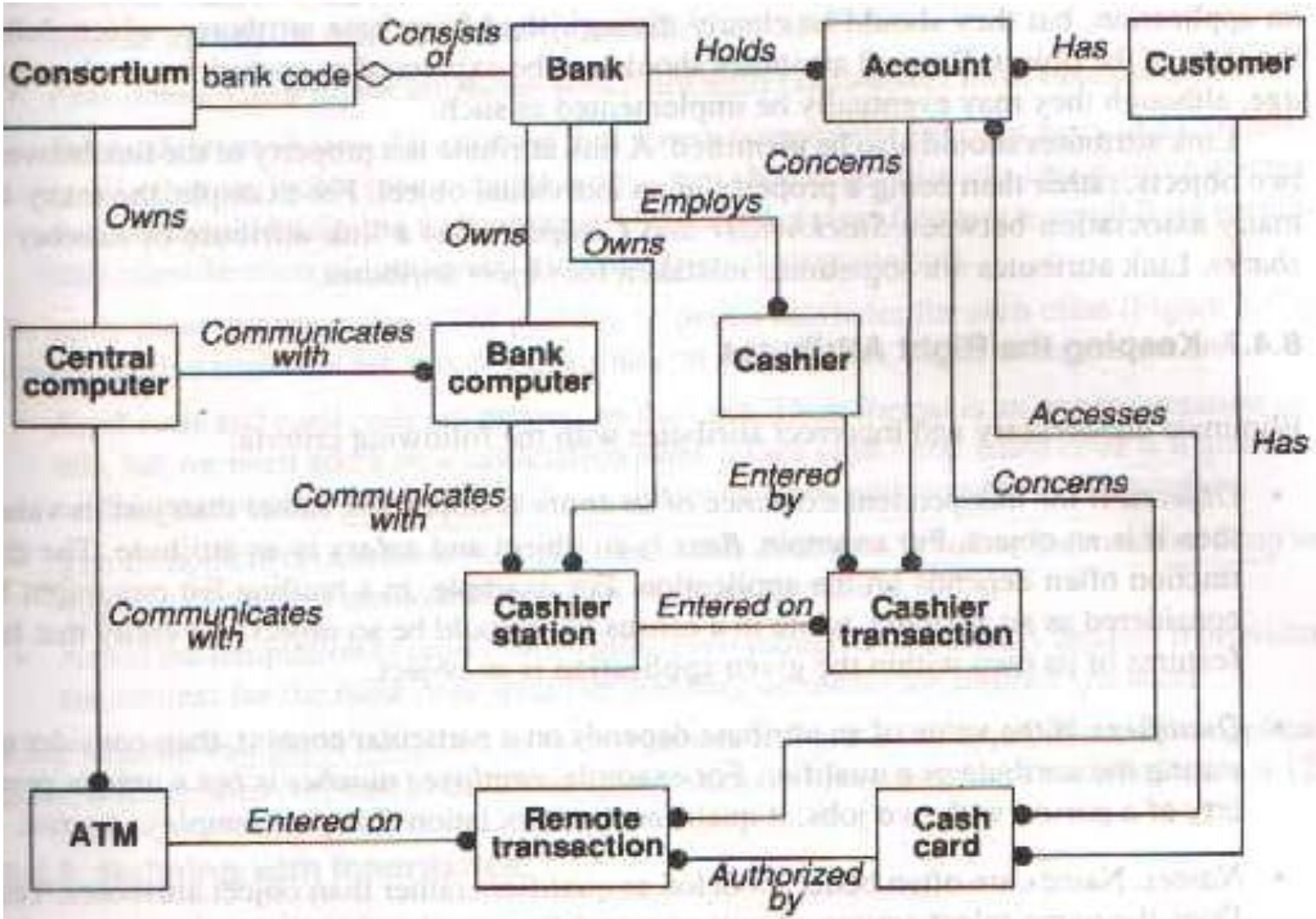


Figure 8.11 Initial object diagram for ATM system

6. Identify attributes

- Attributes usually correspond to nouns followed by possessive phrases.
  - Eg : color of the car
- Attributes less likely to be fully described in the problem statement.
- Only consider attributes that directly relate to a particular application.
- Give each attribute a meaningful name.
- Derived attributes should be omitted or cleared labeled.
- Link attributes should also be identified.

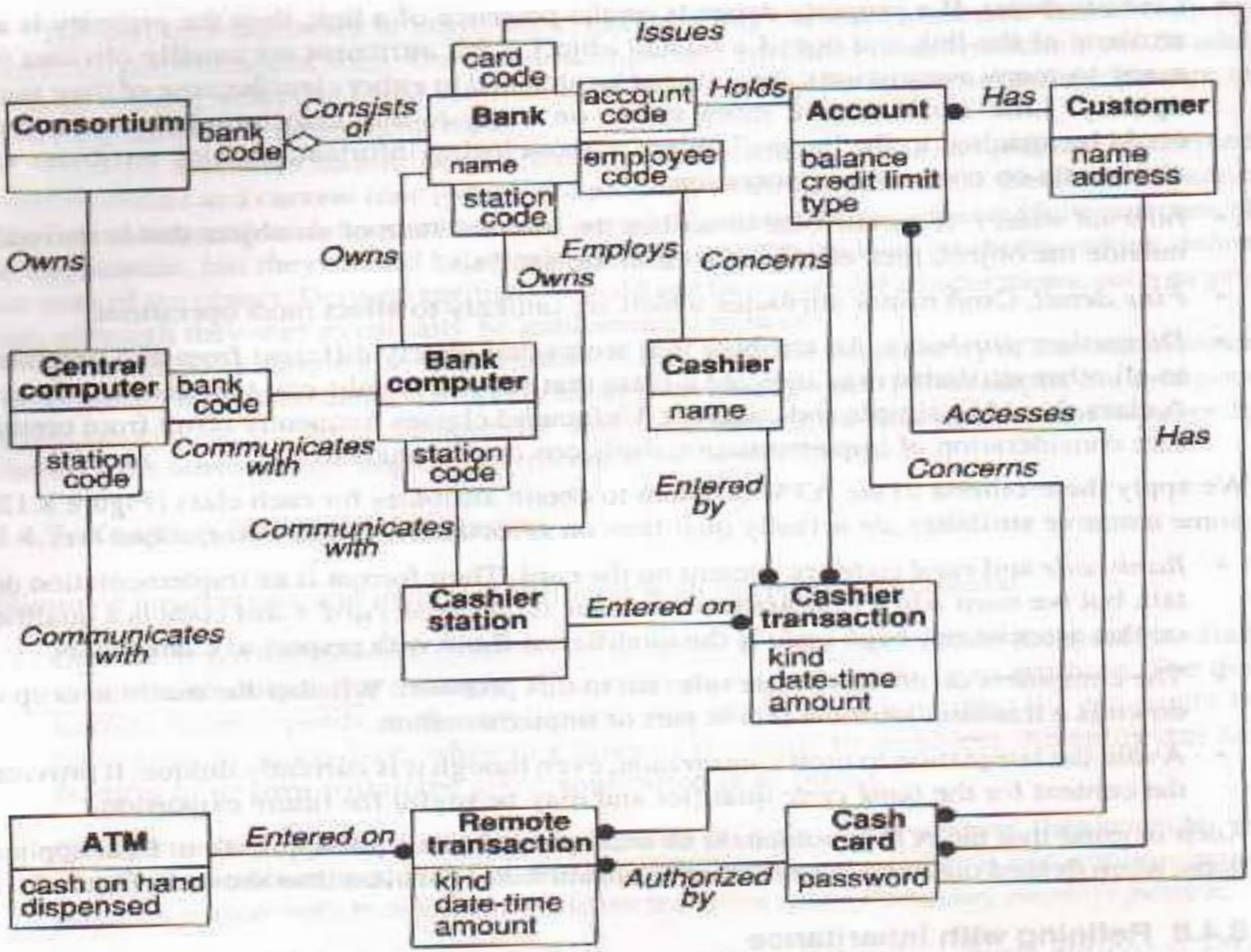


Figure 8.12 ATM object model with attributes

7. Refine with Inheritance

- Inheritance can be added in two directions.
  - By generalizing common aspects of existing classes into a superclass.
  - By refining existing classes into specialized subclasses.
- Entry station generalizes cashier station and ATM.

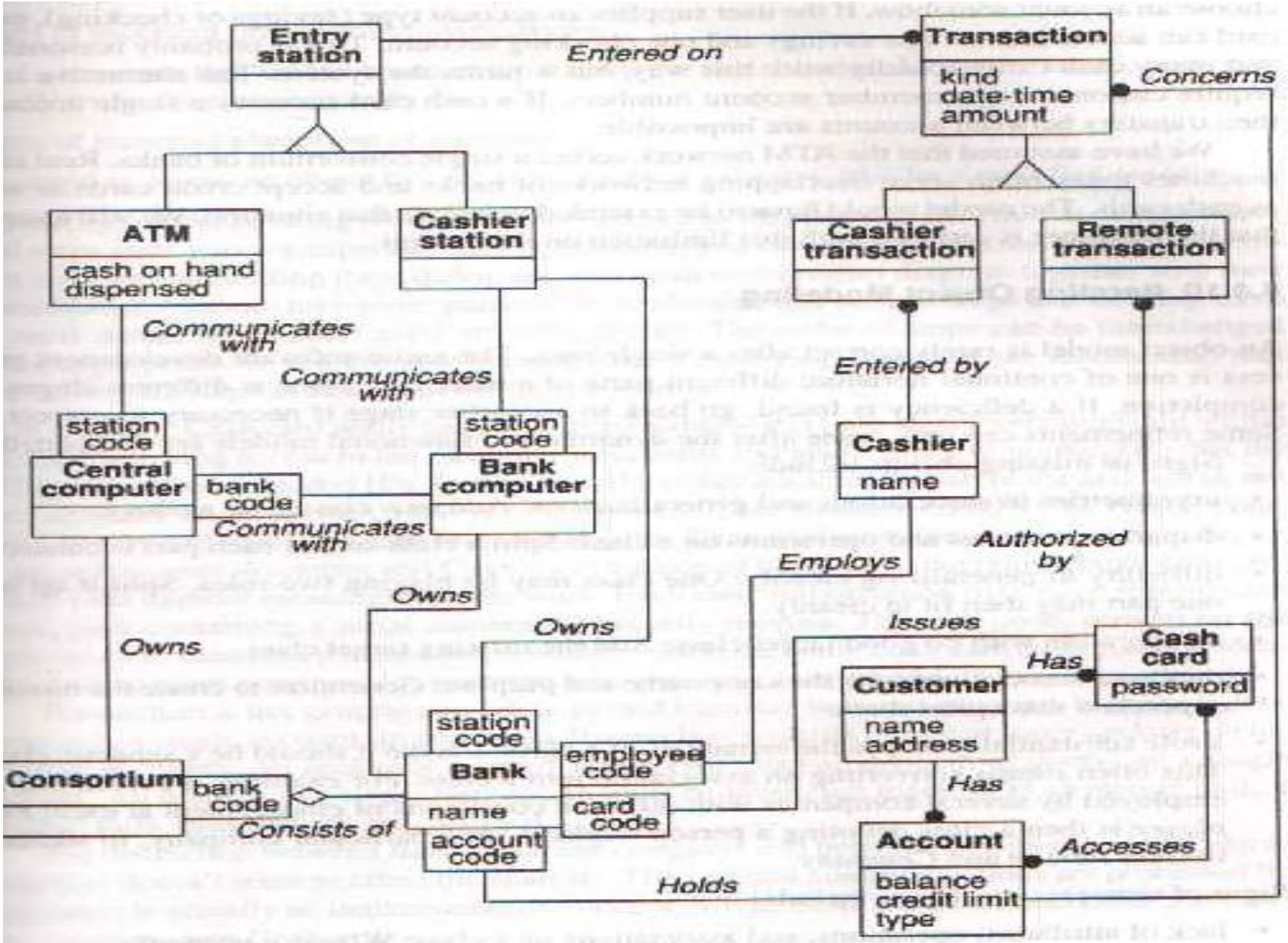


Figure 8.13 ATM object model with attributes and inheritance

8. Test the access path

- Trace access path through the object model diagram to see if they yield sensible results.
  - A cash card itself does not uniquely identify an account, so the user must choose an account.
  - If the user supplies an account type, each card can access at most one saving and one checking account.

9. Iterate and refine the model

- An object model is rarely correct after a single pass.
- The entire software development process is one of continual iteration.
- Different parts of a model are often at different stages of completion.
- If a deficiency is found, go back to an earlier stage if necessary to correct it.
- Some refinements can only come after the dynamic and functional models are completed.

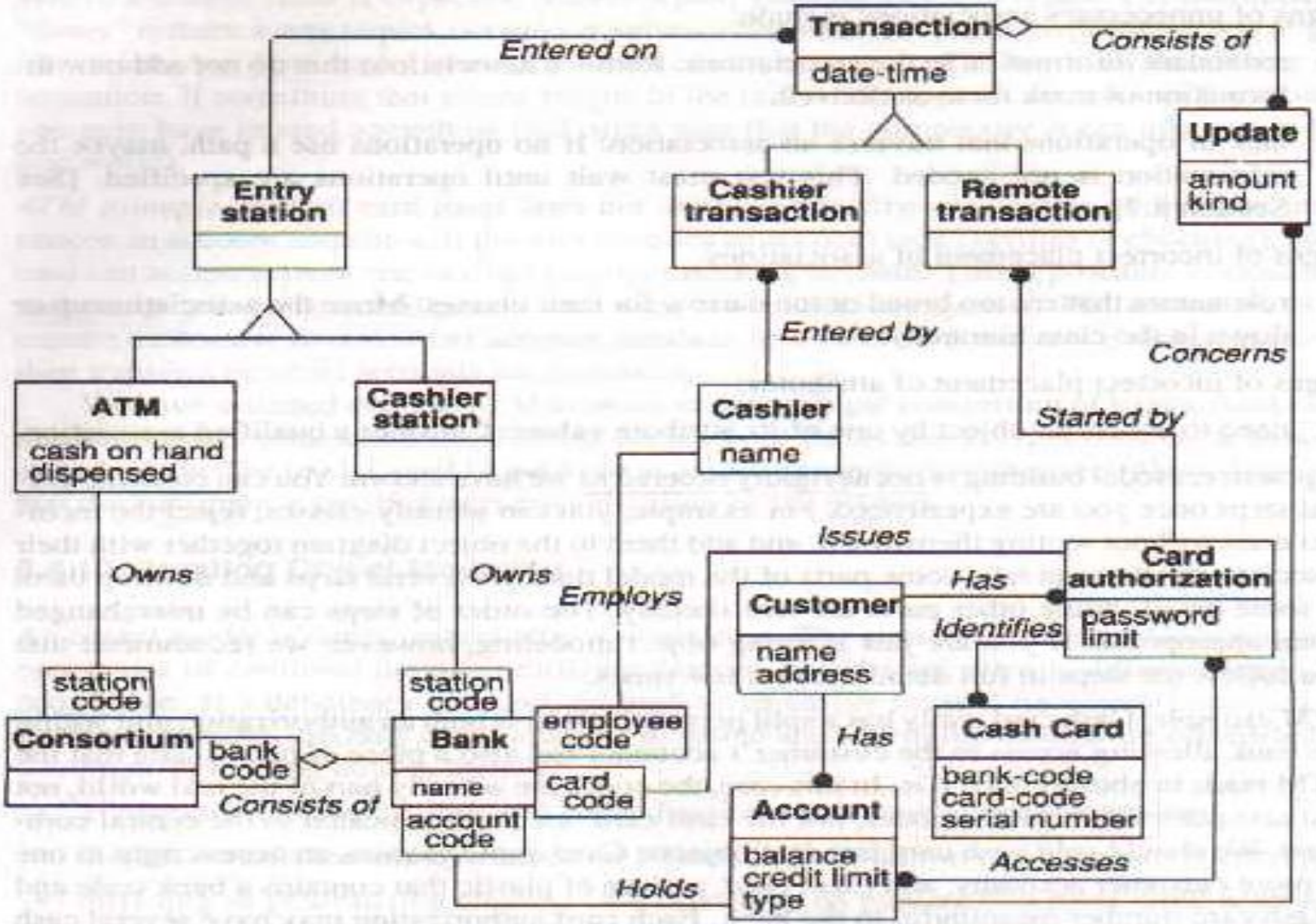


Figure 8.14 ATM object model after further revision

Grouping Classes into Modules

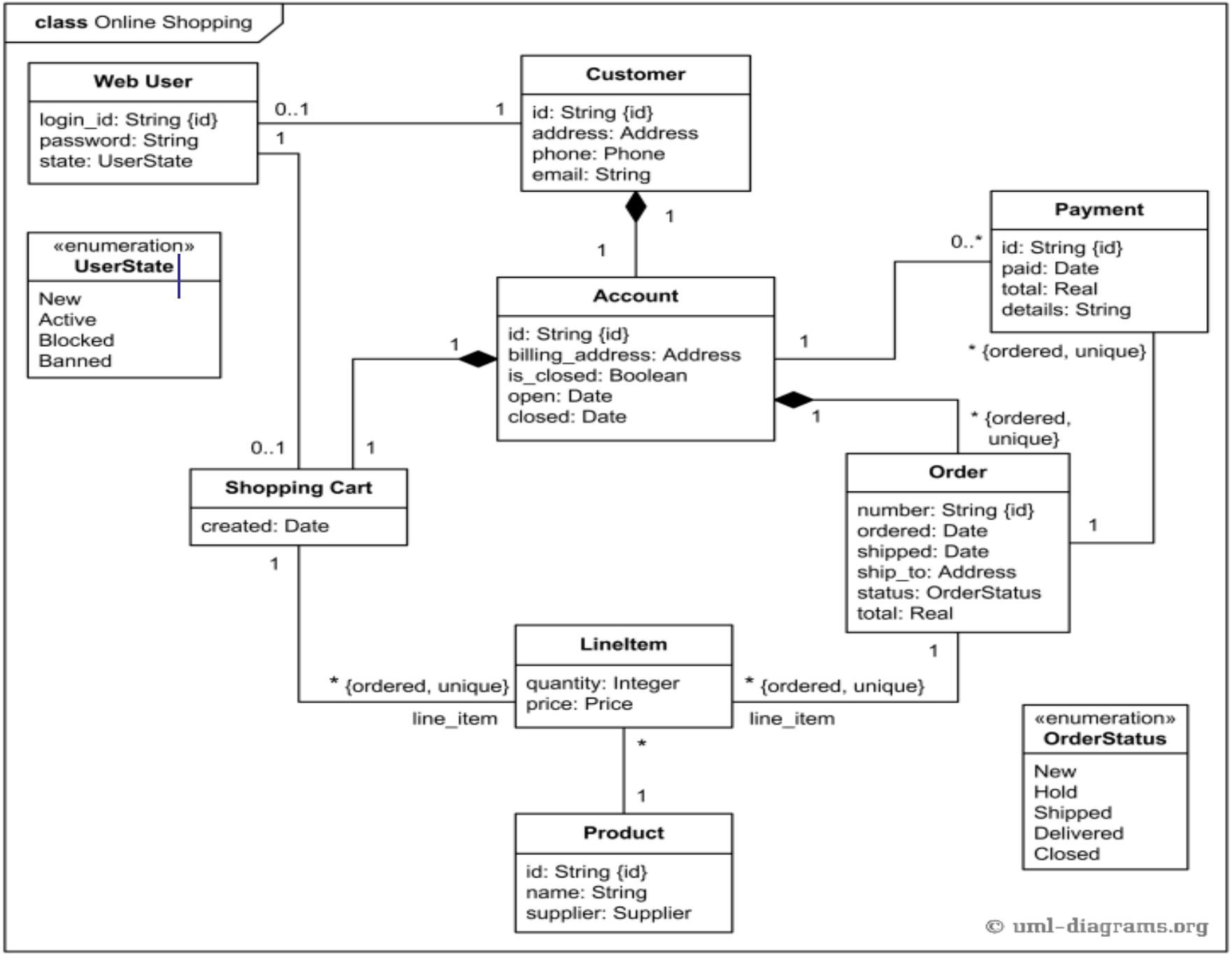
- Diagrams may be divided into sheets of uniform size for convenience in drawing, printing, and viewing.
- Tightly coupled classes should be grouped together.
- A module is a set of classes that captures some logical subset of the entire model.
- The modules might be
  - Teller – cashier, entry station, cashier station, ATM
  - Accounts – account, cash card, card authorization, customer, transaction, update, cashier transaction, remote transaction
  - Banks – consortium, bank.

Online Shopping

Each customer has unique id and is linked to exactly one **account**.
Account owns shopping cart and orders.
Customer could register as a web user to be able to buy items online.
Customer is not required to be a web user because purchases could also be made by phone or by ordering from catalogues.
Web user has login name which also serves as unique id. Web user could be in several states - new, active, temporary blocked, or banned, and be linked to a **shopping cart**.
Shopping cart belongs to account. Account owns customer orders.

Online Shopping

Customer may have no orders. Customer orders are sorted and unique.
Each order could refer to several **payments**, possibly none. Every payment has unique id and is related to exactly one account.
Each order has current order status.
Both order and shopping cart have **line items** linked to a specific product.
Each line item is related to exactly one product. A product could be associated to many line items or no item at all


class Online Shopping
Web User
login_id: String {id}
password: String
state: UserState
0..1
1
1
Customer
id: String {id}
address: Address
phone: Phone
email: String
1
1
«enumeration»
UserState
New
Active
Blocked
Banned
Account
id: String {id}
billing_address: Address
is_closed: Boolean
open: Date
closed: Date
1
1
0..*
Payment
id: String {id}
paid: Date
total: Real
details: String
* {ordered, unique}
1
1
* {ordered, unique}
0..1
1
Shopping Cart
created: Date
1
Order
number: String {id}
ordered: Date
shipped: Date
ship_to: Address
status: OrderStatus
total: Real
1
1
LineItem
quantity: Integer
price: Price
* {ordered, unique}
line_item
* {ordered, unique}
line_item
*
1
Product
id: String {id}
name: String
supplier: Supplier
«enumeration»
OrderStatus
New
Hold
Shipped
Delivered
Closed
© uml-diagrams.org

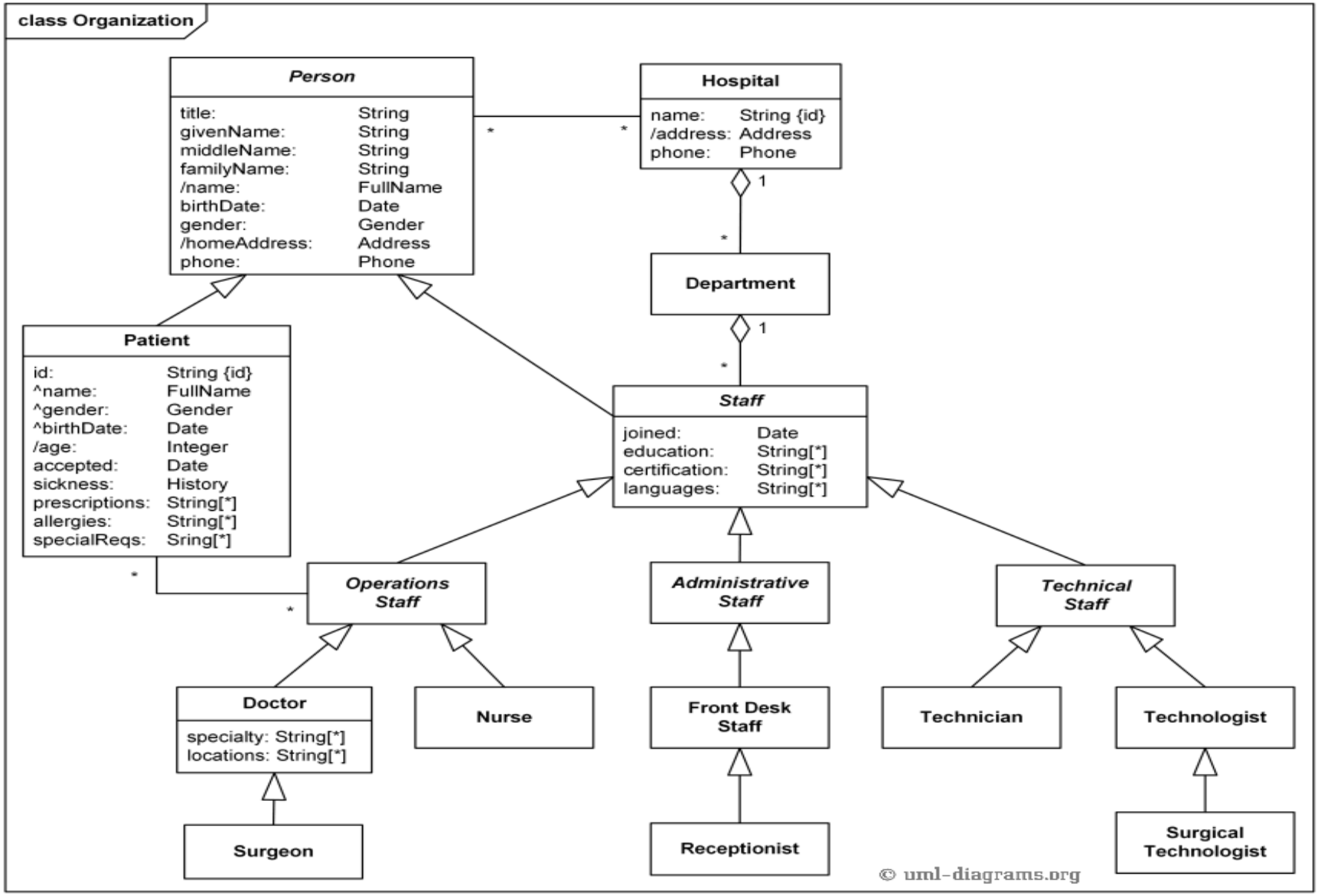
Hospital organization

The domain model for the **Hospital Management System** is represented by several class diagrams.

The purpose of the diagram is to show and explain hospital structure, staff, relationships with patients, and patient treatment terminology.

On the diagram below a *Person* could be associated with different *Hospitals*, and a Hospital could employ or serve multiple Persons. *Person* class has derived attributes *name* and *homeAddress*. Name represents full name and could be combined from title, given (or first) name, middle name, and family (or last) name. *Patient* class has derived attribute *age* which could be calculated based on her or his birth date and current date or hospital admission date.
The *Patient* class inherits attributes from the *Person* class. Several inherited attributes *name*, *gender*, and *birthDate* are shown with prepended *caret* '^' symbol (new notation introduced in UML 2.5).

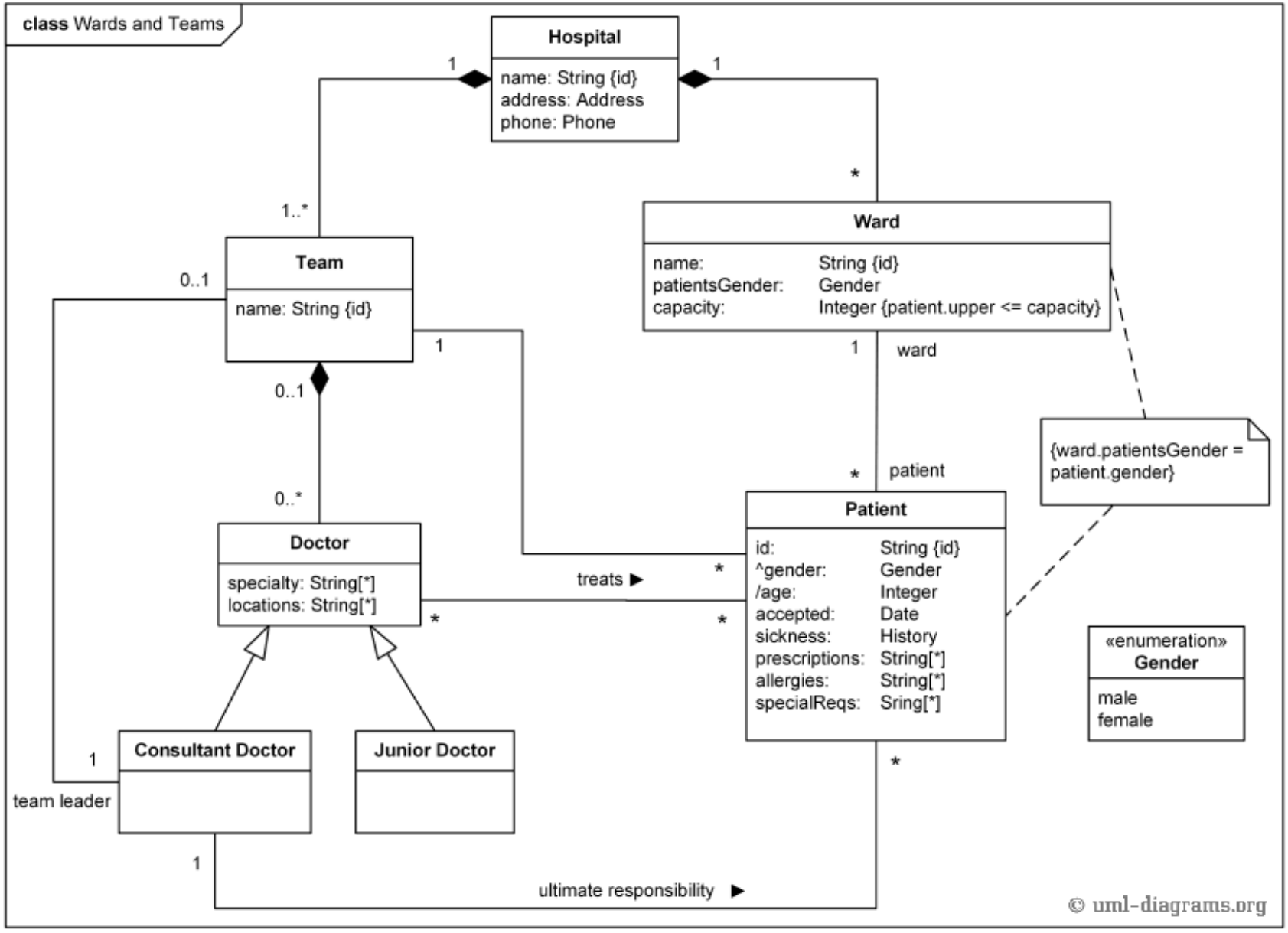
Hospital organization



Staff, Relationships With Patients

Ward is a division of a hospital or a suite of rooms shared by patients who need a similar kind of care. In a hospital, there are a number of wards, each of which may be empty or have on it one or more **patients**. Each ward has a unique name. Diagram below shows it using **{id}** modifier for ward's name.
Wards are differentiated by **gender** of its patients, i.e. male wards and female wards. A ward can only have patients of the gender admitted to it. Gender is shown as enumeration. Ward and patient have constraint on Gender.
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 organised into **teams**. Each team has a unique name or code and is headed by a **consultant doctor** or **attending physician**. Consultant doctor or attending physician is the senior doctor who has completed all of his or her specialist training, residency and practices medicine in a clinic or hospital, in the specialty learned during residency. She or he can supervise fellows, residents, and medical students. The rest of the team are all junior doctors. Each doctor could be a member of no more than one team.

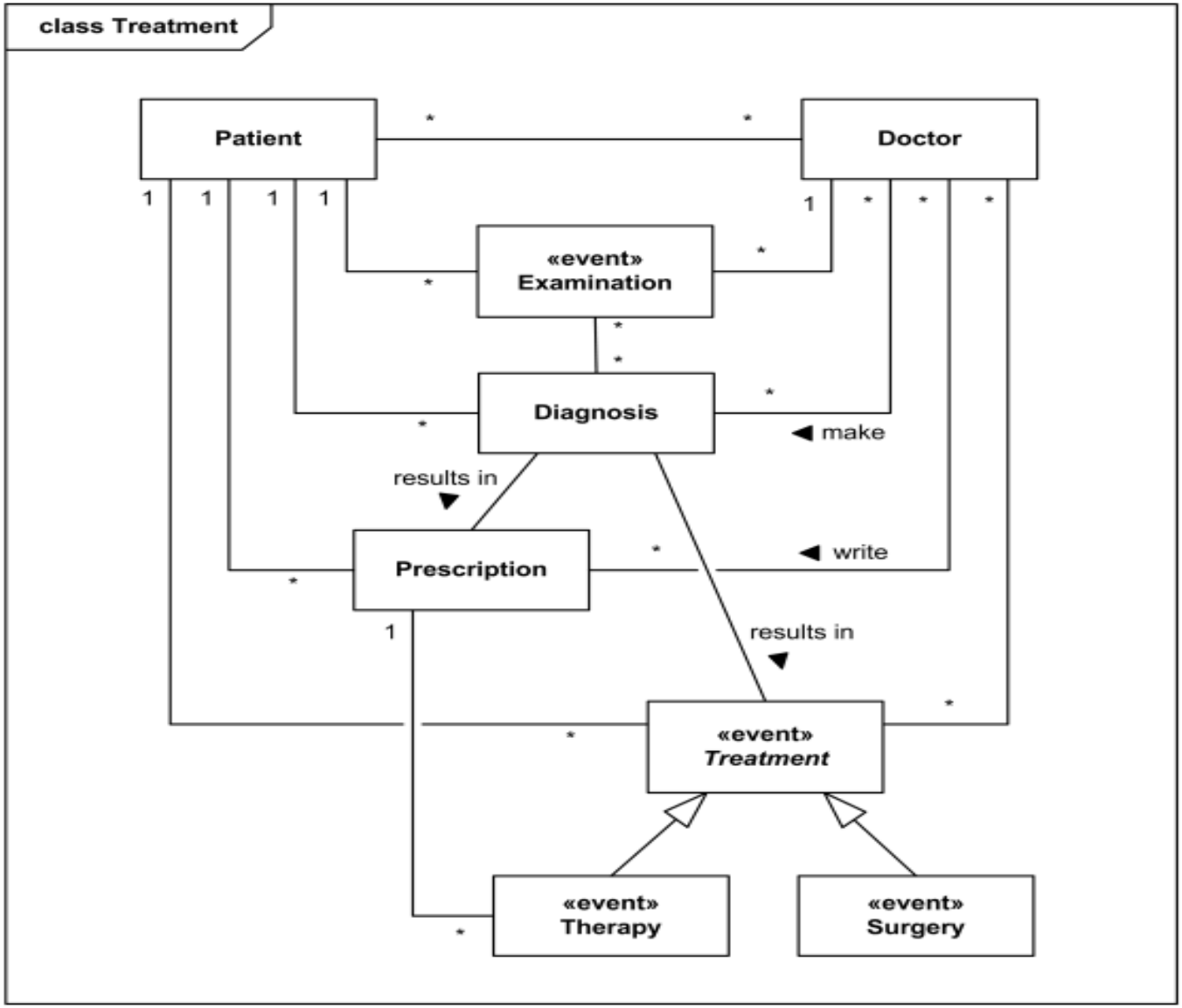
Staff, Relationships With Patients



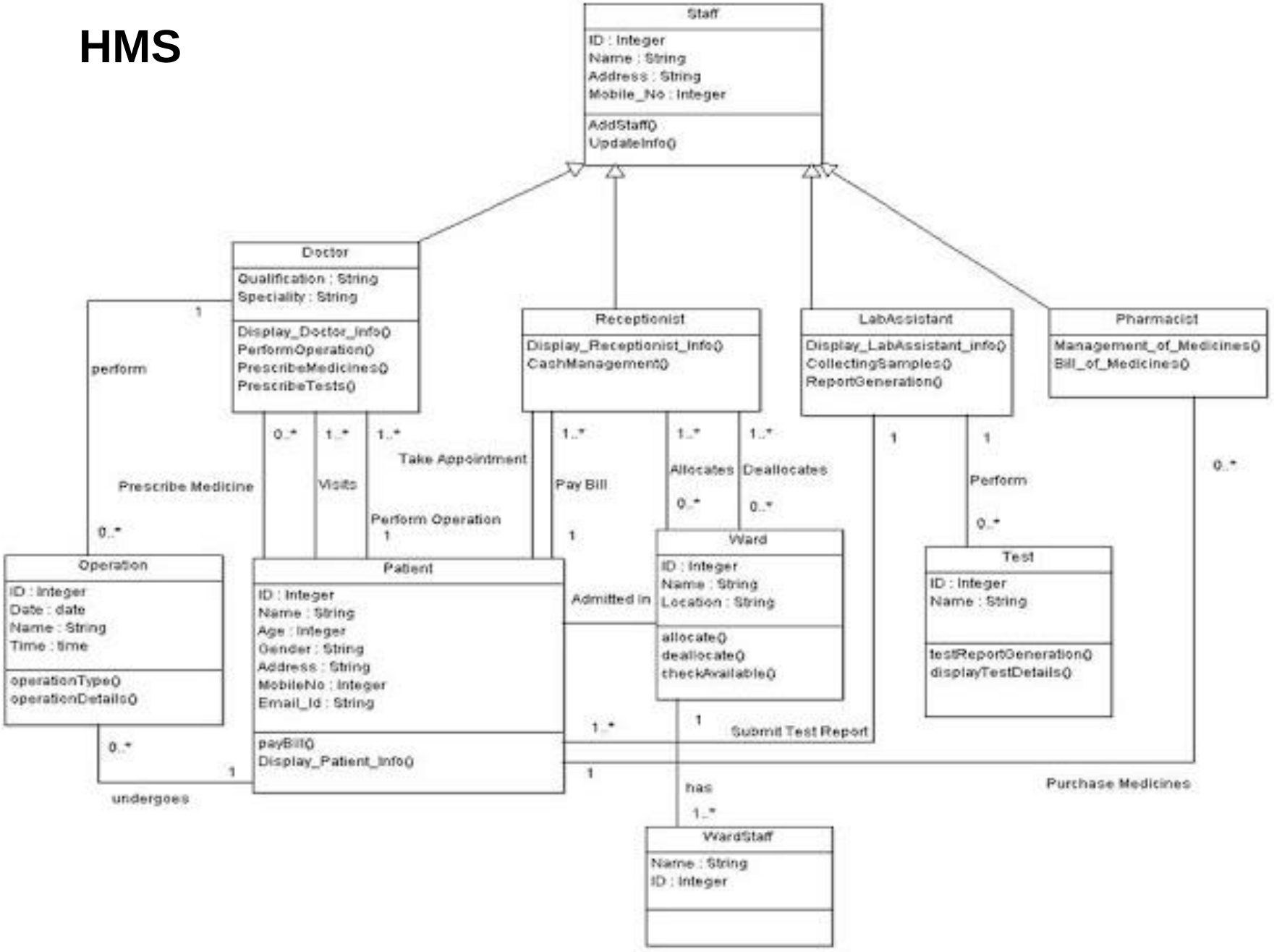
Patient Treatment

Each **patient** is on 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.

Patient Treatment



HMS


Staff
ID : Integer
Name : String
Address : String
Mobile_No : Integer
AddStaff()
UpdateInfo()
Doctor
Qualification : String
Speciality : String
Display_Doctor_Info()
PerformOperation()
PrescribeMedicines()
PrescribeTests()
Receptionist
Display_Receptionist_Info()
CashManagement()
LabAssistant
Display_LabAssistant_info()
CollectingSamples()
ReportGeneration()
Pharmacist
Management_of_Medicines()
Bill_of_Medicines()
perform
Prescribe Medicine
Visits
Take Appointment
Perform Operation
Pay Bill
Allocates
Deallocates
Perform
Admitted in
Submit Test Report
Purchase Medicines
undergoes
has
Operation
ID : Integer
Date : date
Name : String
Time : time
operationType()
operationDetails()
Patient
ID : Integer
Name : String
Age : Integer
Gender : String
Address : String
MobileNo : Integer
Email_Id : String
payBill()
Display_Patient_Info()
Ward
ID : Integer
Name : String
Location : String
allocate()
deallocate()
checkAvailable()
Test
ID : Integer
Name : String
testReportGeneration()
displayTestDetails()
WardStaff
Name : String
ID : Integer