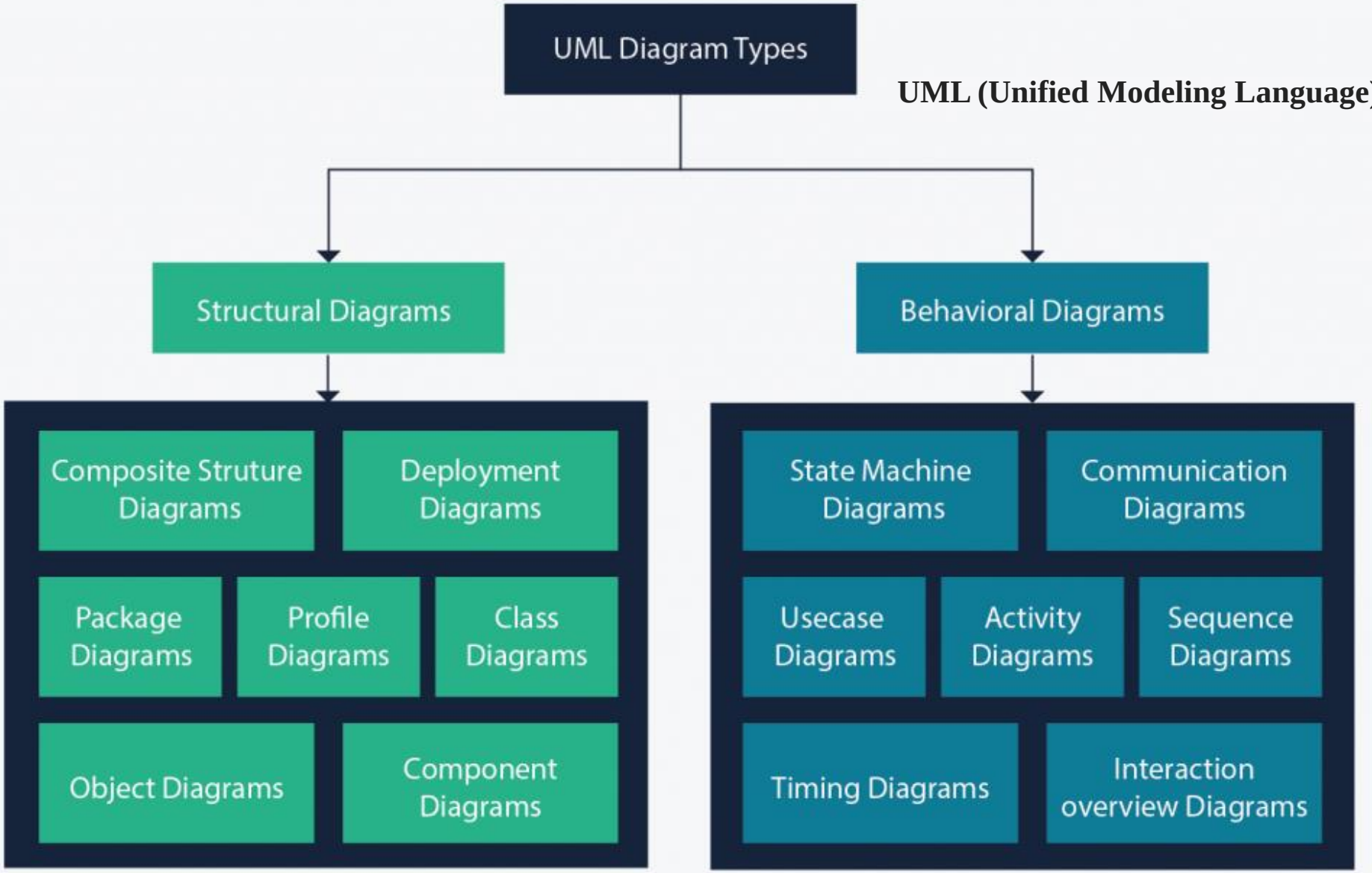
7. Object Oriented Modeling

Abdus Sattar
Assistant Professor
Department of Computer Science and Engineering
Daffodil International University
Email: abdus.cse@diu.edu.bd

Discussion Topics:

- **UML diagram types**
- **Structured diagram, Behavioral diagram**
- **A Class description with elements**
- **UML Relationship of Object interconnections**
- **Practicing exercise on object model diagram from case study**

UML Diagram Types



UML (Unified Modeling Language)

Diagram Types

- **Structure diagrams** show the things in the modeled system. In a more technical term, they show **different objects** in a system.

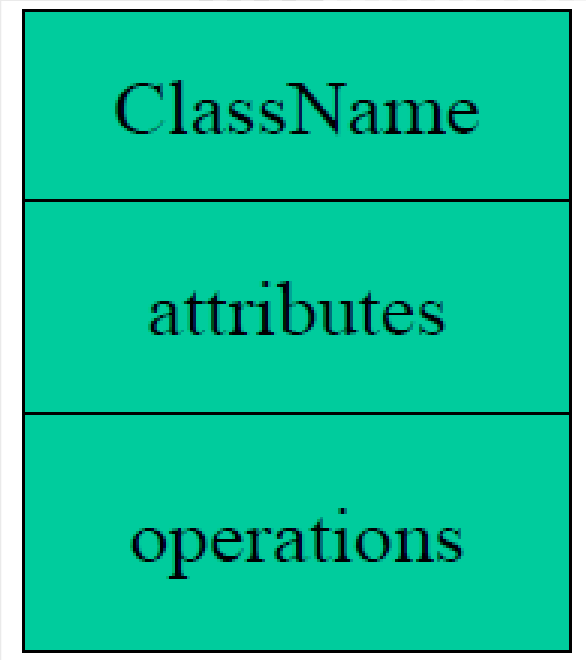
- **Behavioral diagrams** show what should happen in a system. They describe how the objects **interact with each other** to create a functioning system.

Diagram Types

- **Class diagram** is a graph of classifier elements connected by their various static relationships. A "class" diagram may also contain **interfaces, packages, relationships**, and even instances, such as objects and links.

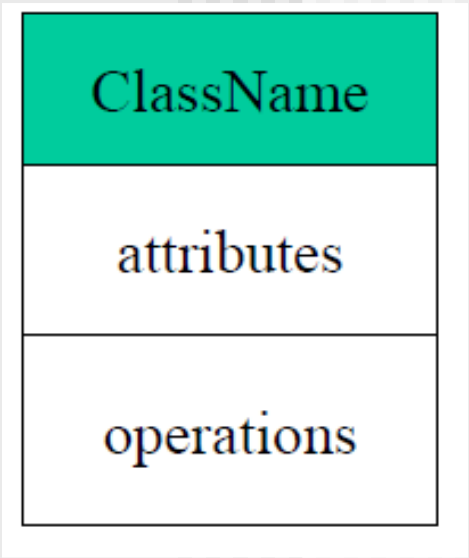
- **Object diagram** on the other hand is a graph of instances, including **objects and data values**. A static object diagram is an instance of a class diagram. It shows a snapshot of the detailed state of a system at a point in time. The use of object diagrams is fairly limited, mainly to **show examples of data structures**.

Classes



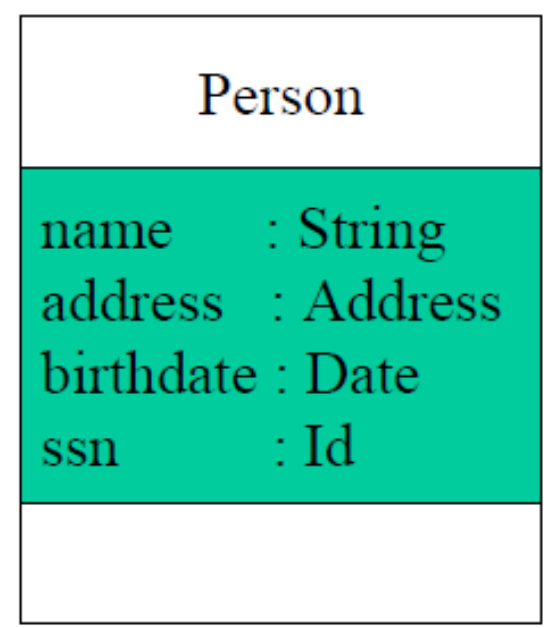
- A *class* is a description of a set of objects that share the same attributes, operations, relationships, and semantics.
- Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

Class Names



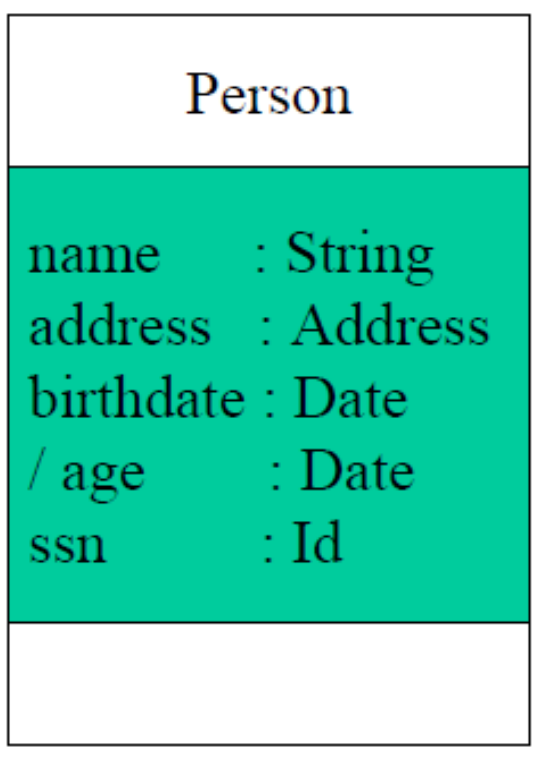
- The name of the class is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.

Class Attributes



- An *attribute* is a named property of a class that describes the object being modeled.
- In the class diagram, attributes appear in the second compartment just below the name-compartment.

Class Attributes (Cont'd)



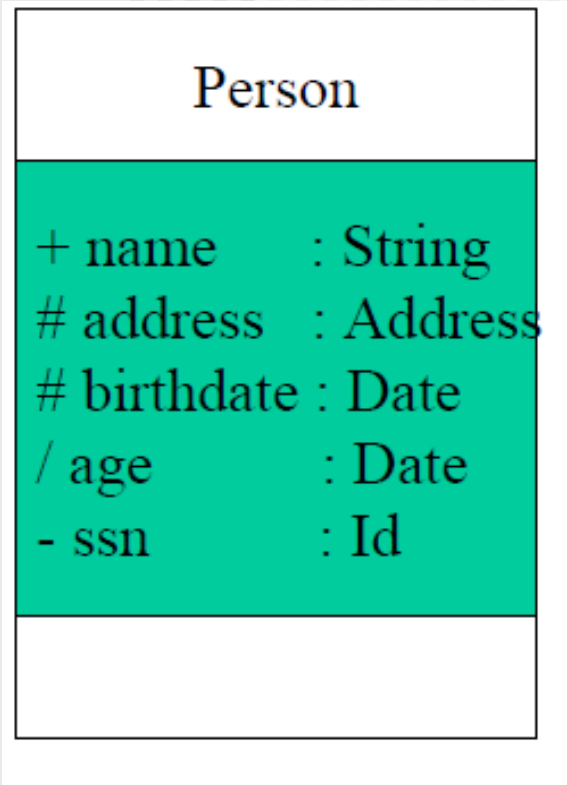
- Attributes are usually listed in the form:

  attributeName : Type

- A ***derived* attribute** is one that can be computed from other attributes, but doesn't actually exist. For example, a Person's age can be computed from his birth date. A derived attribute is designated by a preceding '/' as in:

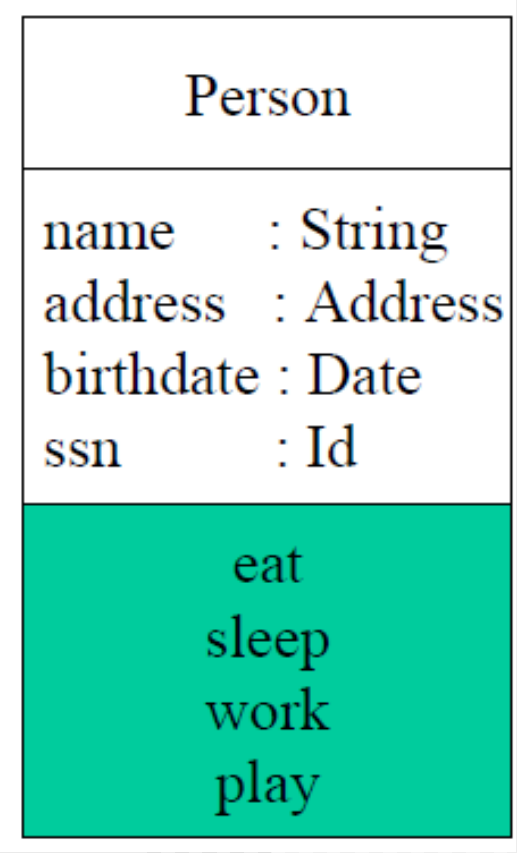
  / age : Date

Class Attributes (Cont'd)



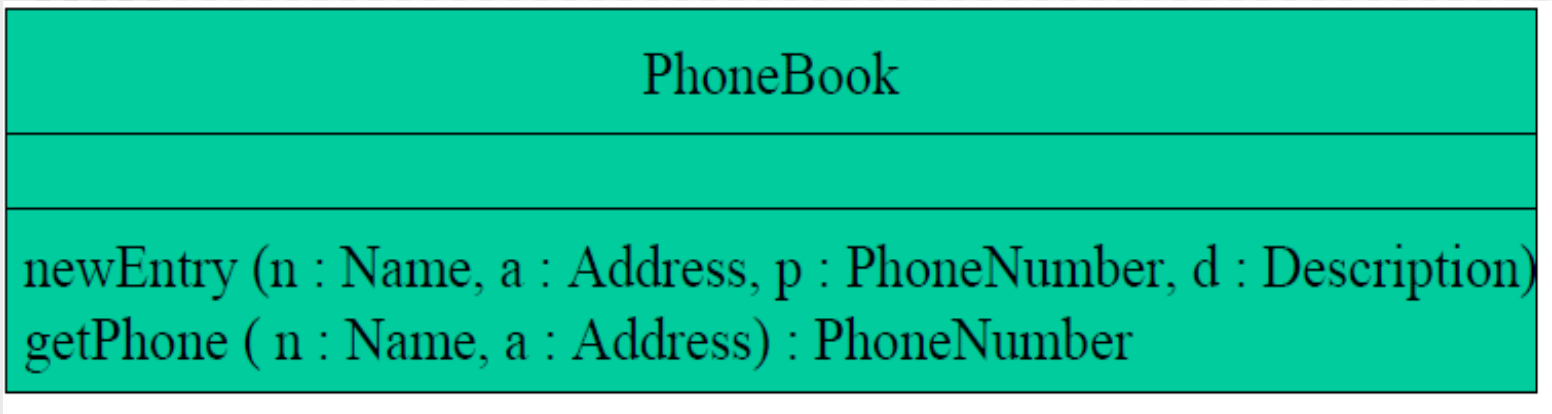
- Attributes can be:
  - + public
  - # protected
  - - private
  - / derived

Class Operations



- *Operations* describe the class behavior and appear in the third compartment.

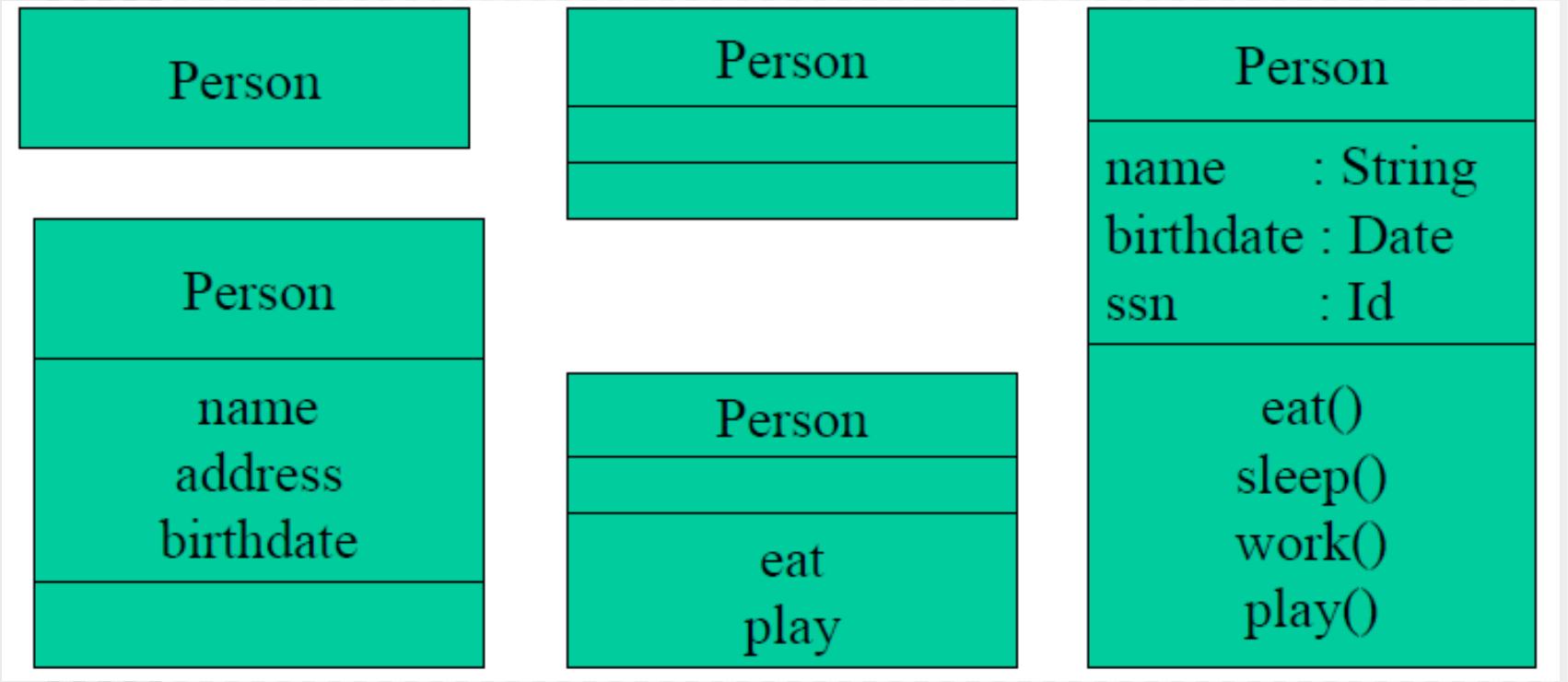
Class Operations (Cont'd)



- You can specify an operation by stating its signature: listing the **name, type, and default value** of all parameters, and, in the case of functions, a return type.

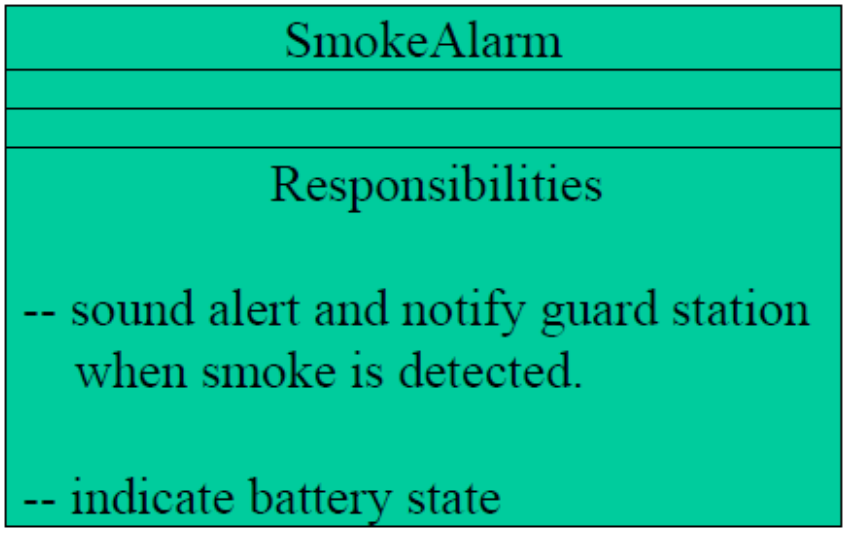
Describing Classes

- When drawing a class, you needn't show attributes and operation in every diagram.



Class Responsibilities

- A class may also include its responsibilities in a class diagram.
- A responsibility is a contract or obligation of a class to perform a particular service.

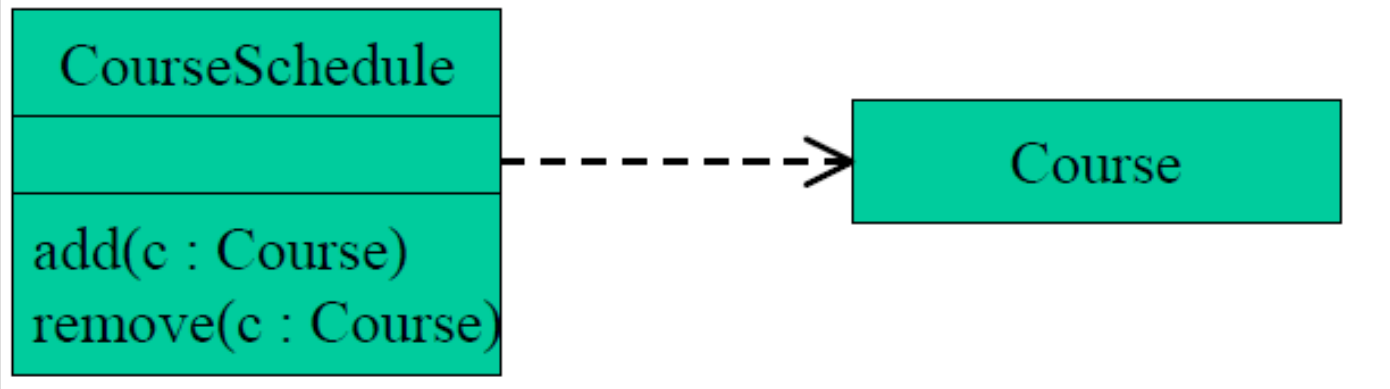


Relationships

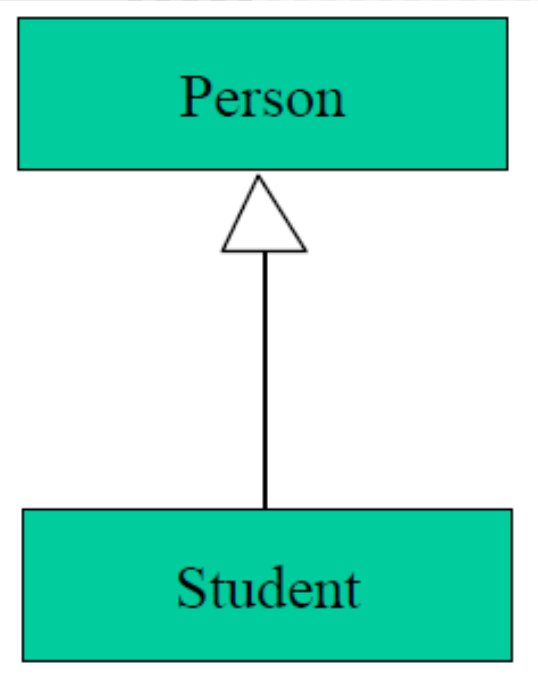
- In UML, object interconnections (logical or physical), are modeled as relationships.
- There are three kinds of relationships in UML:
  - dependencies
  - generalizations
  - associations

Dependency Relationships

- A *dependency* indicates a semantic/notational 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*.



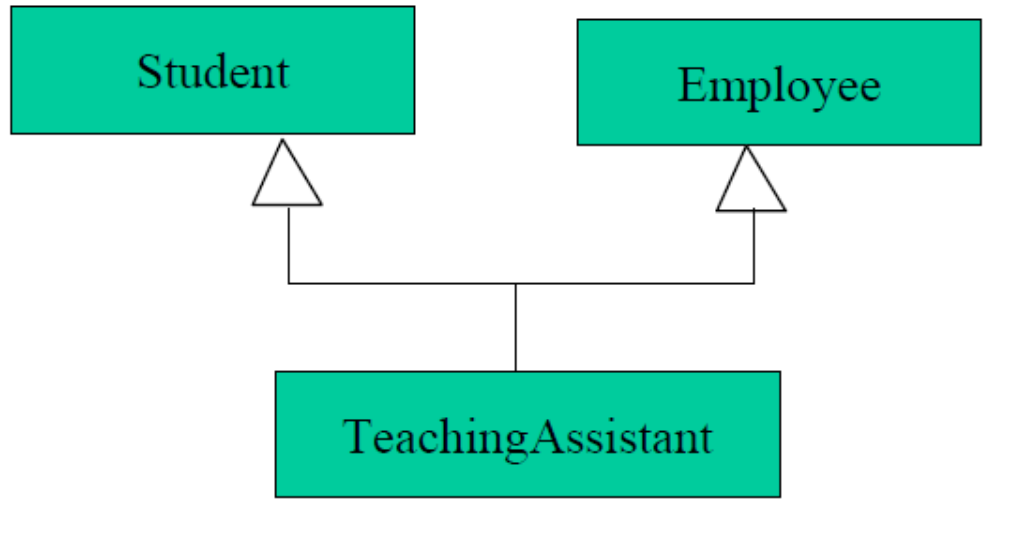
Generalization Relationships



- A *generalization* connects a subclass to its superclass.
- It denotes an inheritance of attributes and behavior from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass.

Generalization Relationships (Cont'd)

- UML permits a class to inherit from multiple super-classes, although some programming languages (*e.g.*, Java) do not permit multiple inheritance.



Association Relationships

- If two classes in a model need to communicate with each other, there must be link between them.
- An *association* denotes that link.

Association Relationships (Cont'd)

- We can indicate the *multiplicity* of an association by adding *multiplicity adornments* to the line denoting the association.
- The example indicates that a *Student* has one or more *Instructors*:

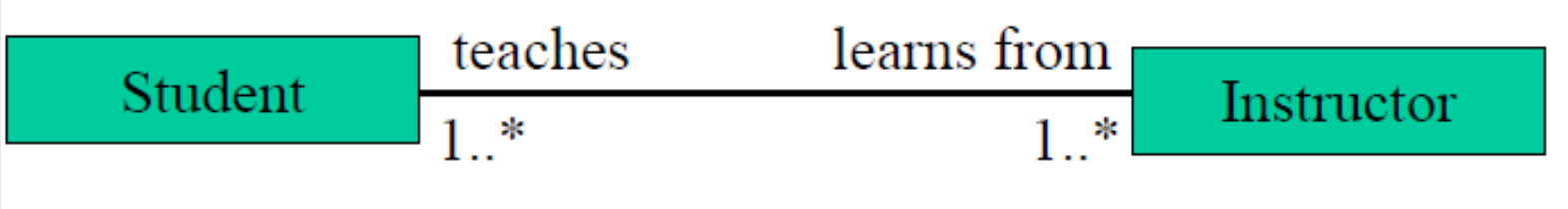
Association Relationships (Cont'd)

- The example indicates that every *Instructor* has one or more
- *Students*:

Association Relationships (Cont'd)

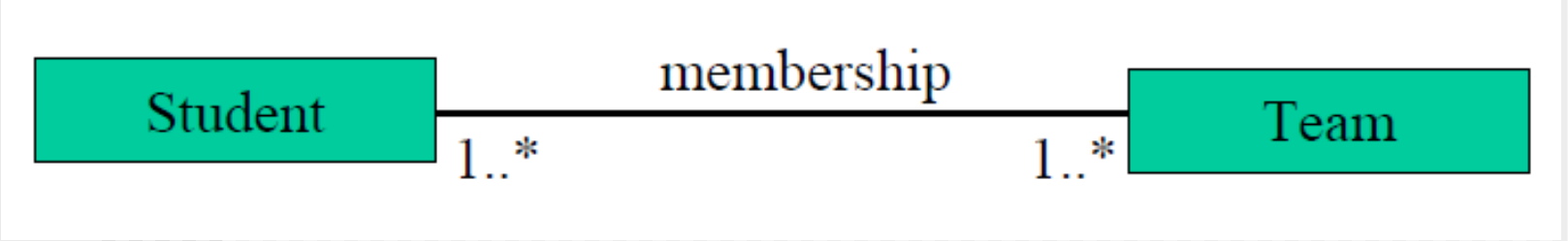
- We can also indicate the behavior of an object in an association

(*i.e.*, the *role* of an object) using *rolenames.*



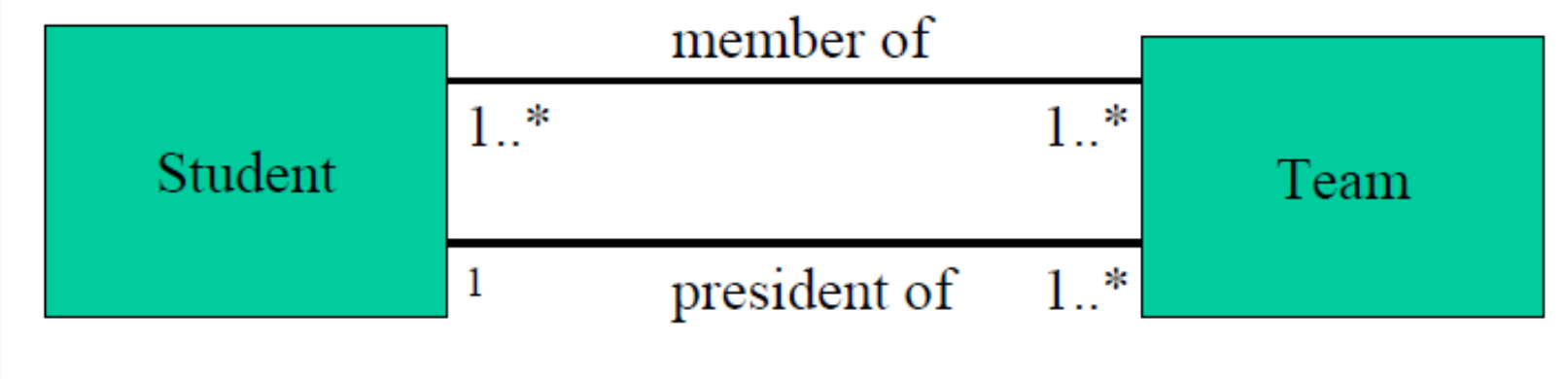
Association Relationships (Cont'd)

- We can also name the association.



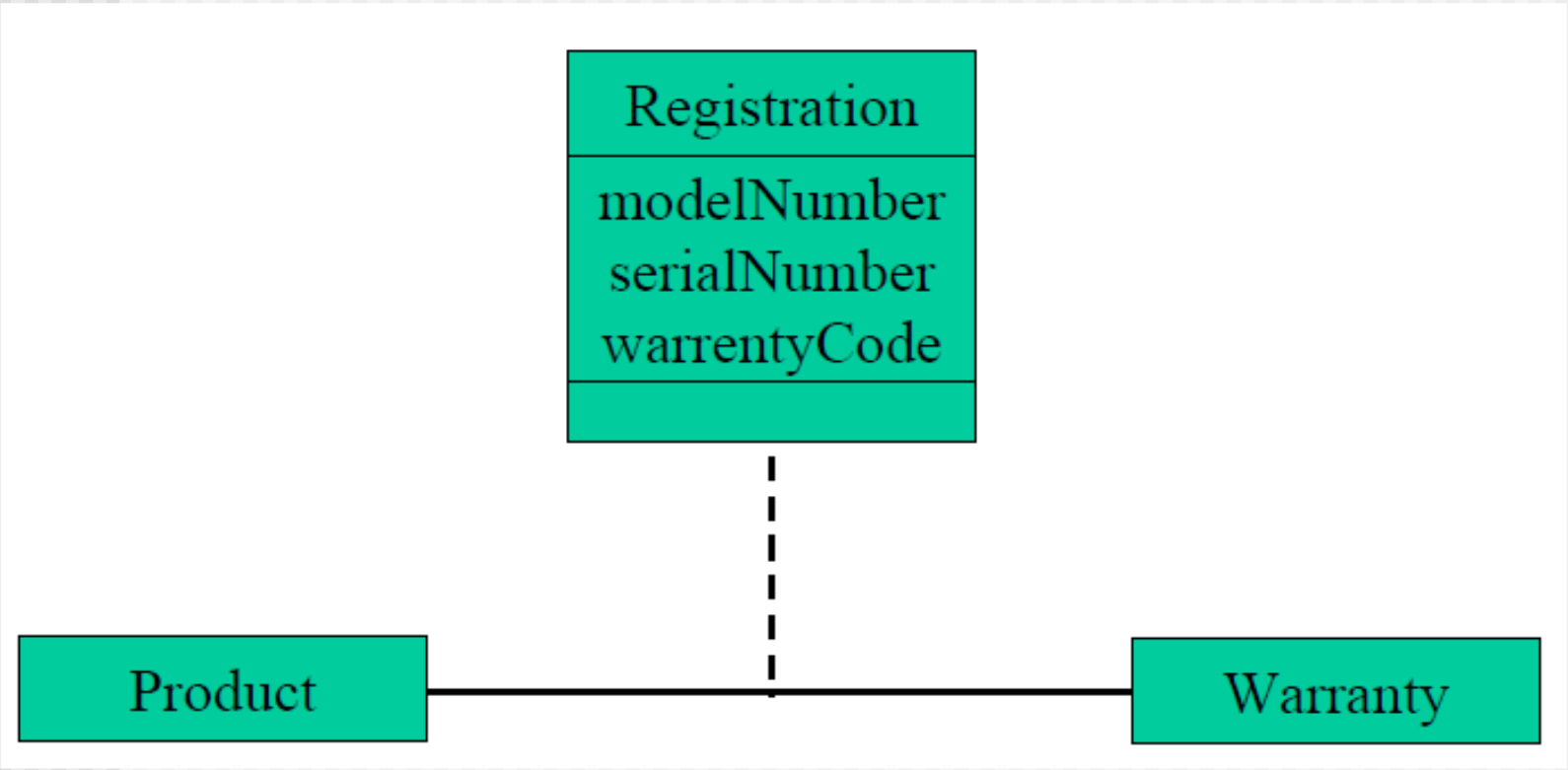
Association Relationships (Cont'd)

- We can specify dual associations.



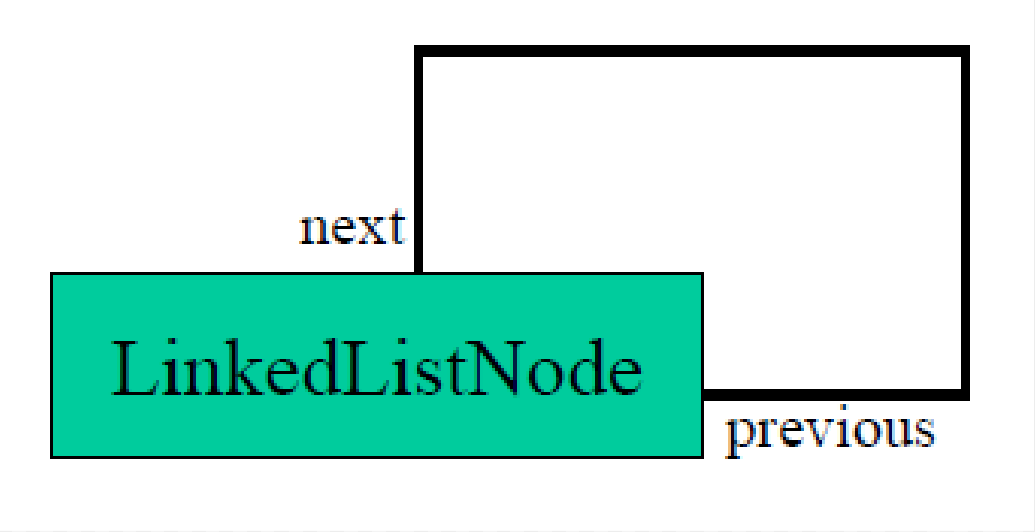
Association Relationships (Cont'd)

- Associations can also be objects themselves, called *link classes* or an *association classes*.



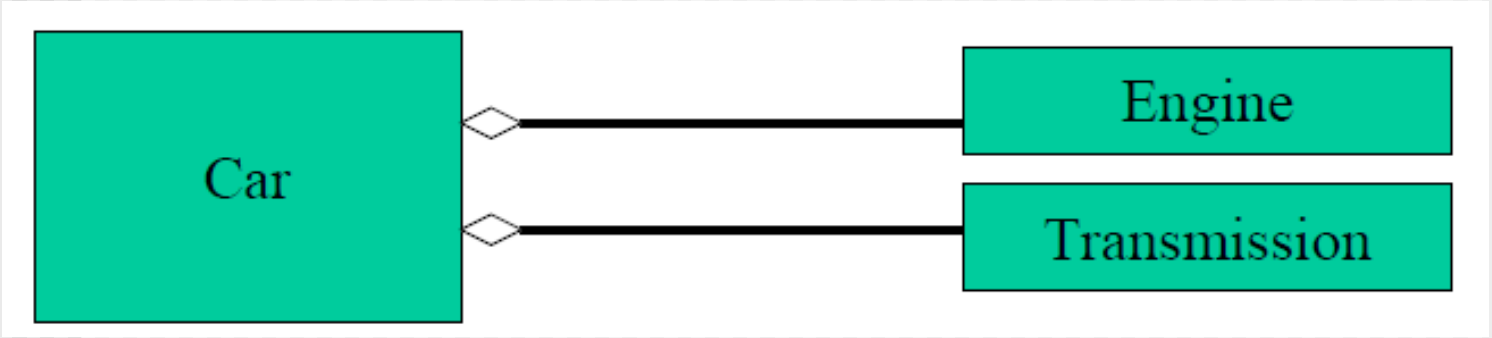
Association Relationships (Cont'd)

- A class can have a *self association*.



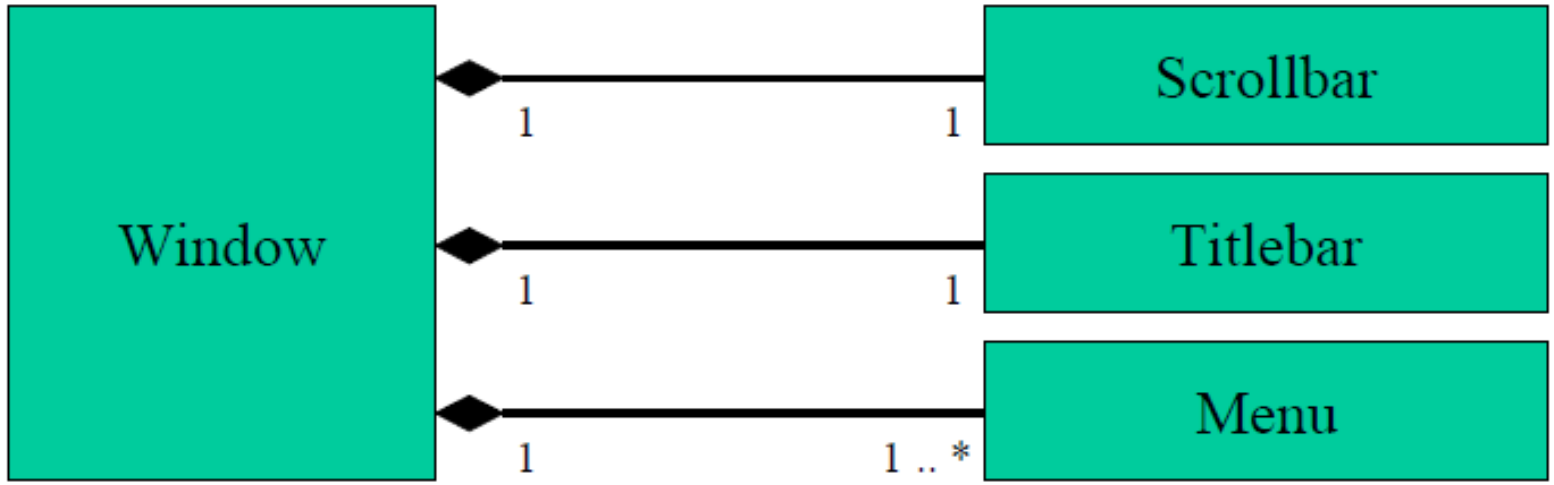
Association Relationships (Cont'd)

- We can model objects that contain other objects by way of special associations called *aggregations* and *compositions.*
- An *aggregation* specifies a whole-part relationship between an aggregate (a whole) and a constituent part, where the part can exist independently from the aggregate. Aggregations are denoted by a hollow-diamond adornment on the association.



Association Relationships (Cont'd)

- A *composition* indicates a strong ownership and coincident lifetime of parts by the whole (*i.e.*, they live and die as a whole).
- Compositions are denoted by a filled-diamond adornment on the association.

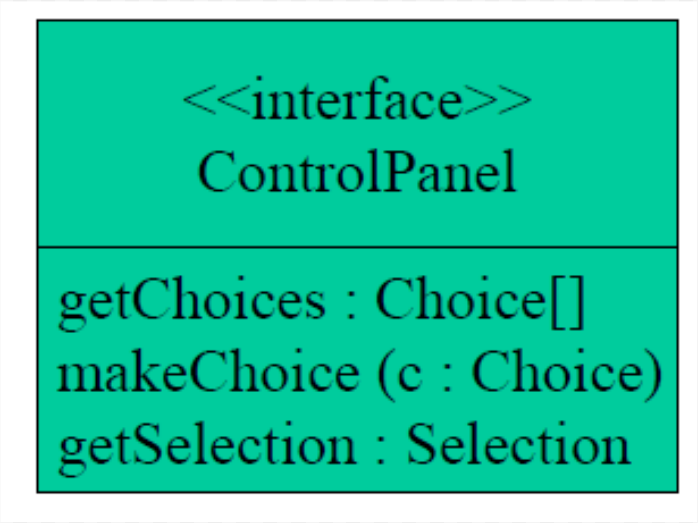


Interfaces

- An *interface* is a named set of operations that specifies the behavior of objects without showing their inner structure. It can be rendered in the model by a one- or two-compartment rectangle, with the *stereotype* <<interface>> above the interface name.

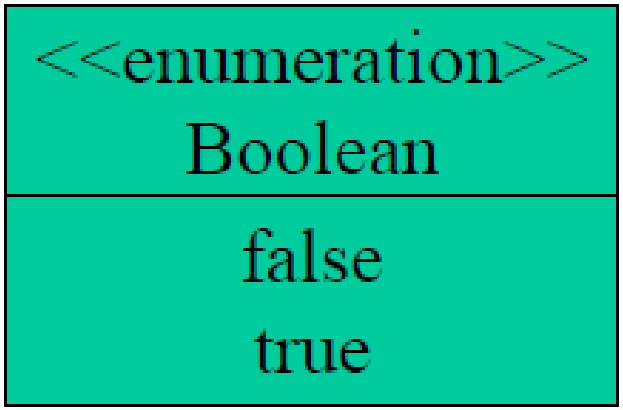
Interface Services

- Interfaces do not get instantiated. They have no attributes or state. Rather, they specify the services offered by a related class.

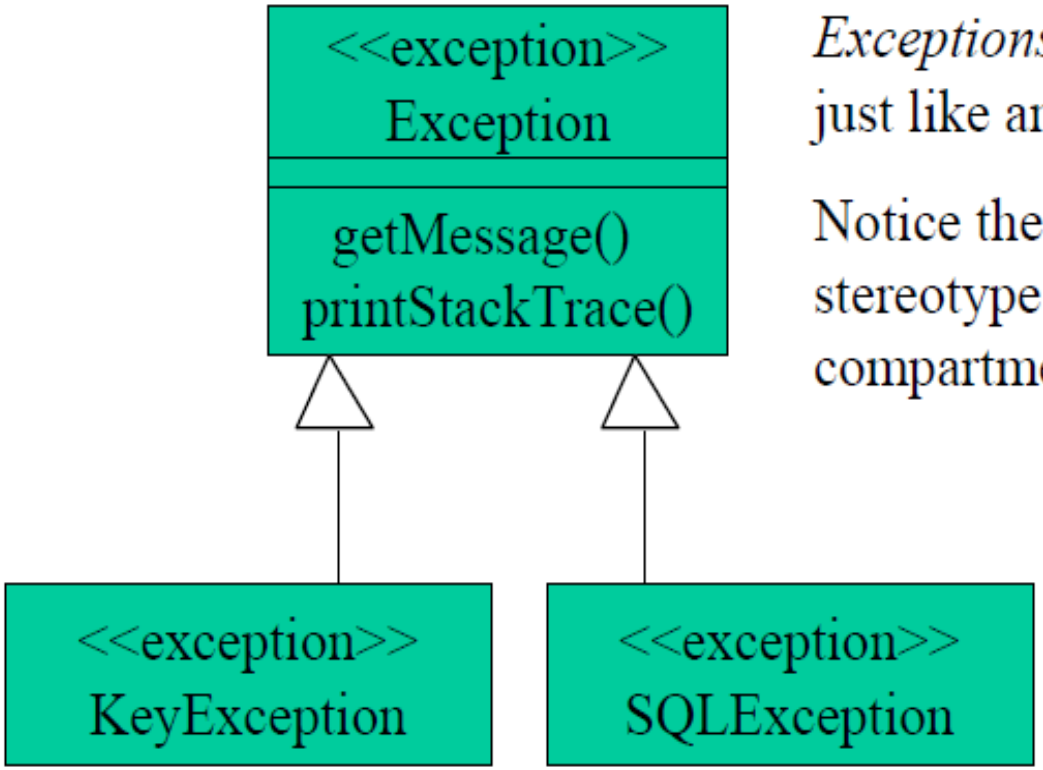


Enumeration

- An *enumeration* is a user-defined data type that consists of a name and an ordered list of enumeration literals.



Exceptions

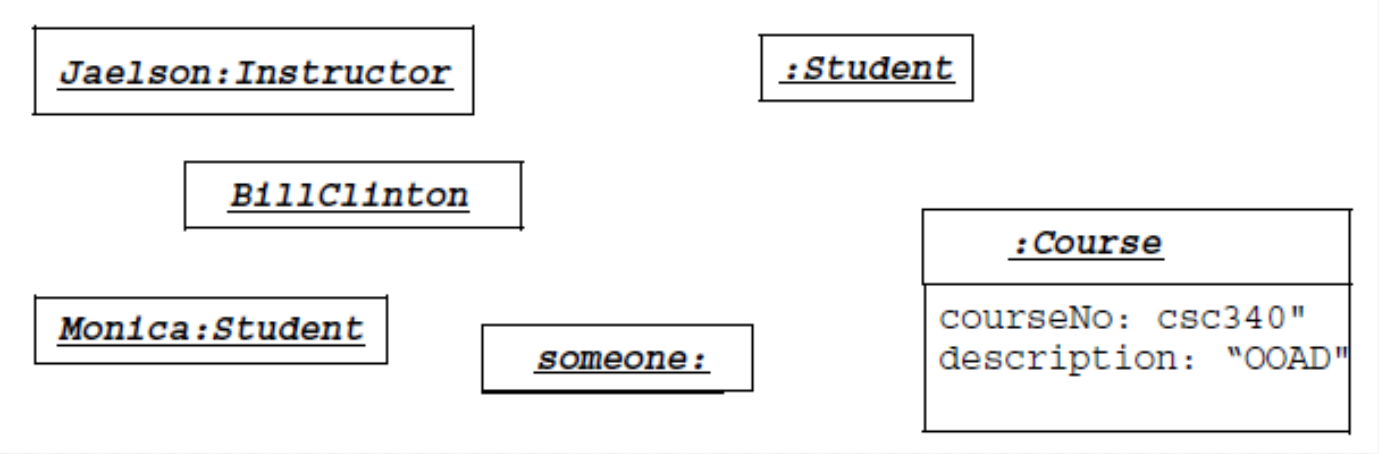


Exceptions can be modeled just like any other class.

Notice the <<exception>> stereotype in the name compartment.

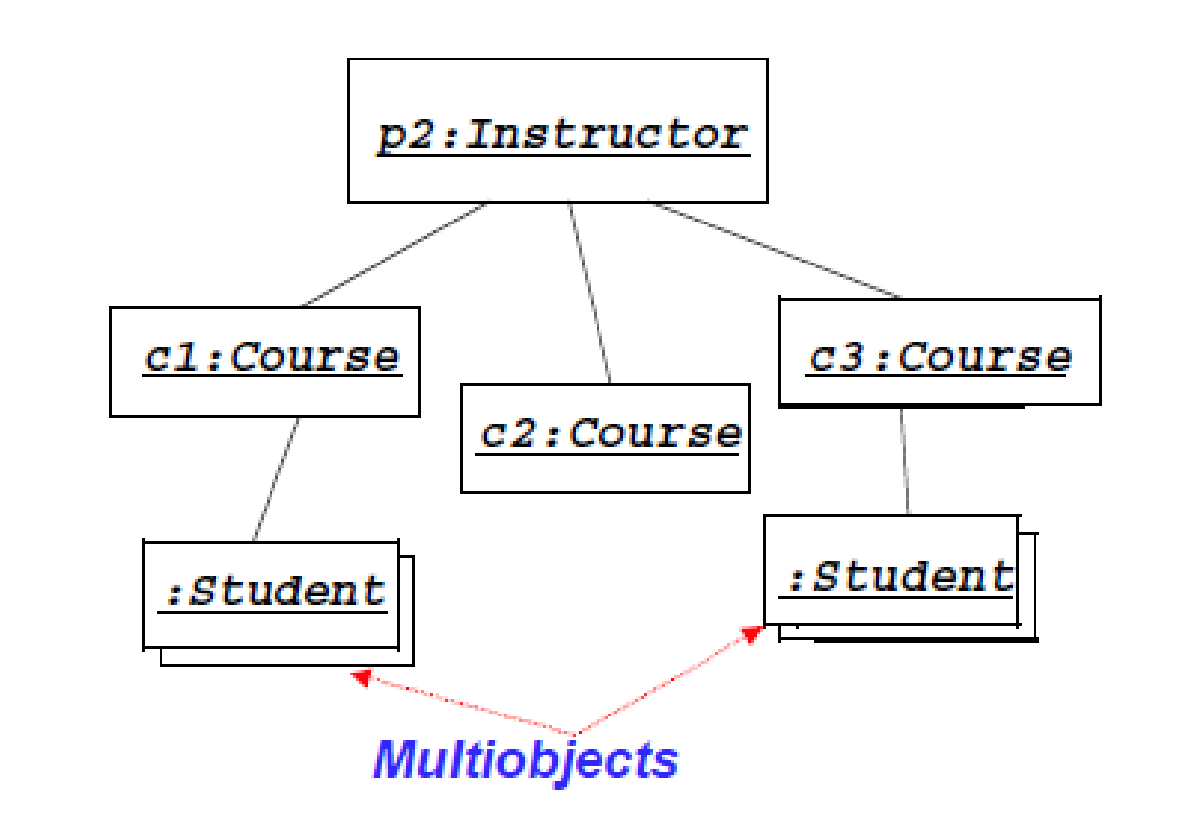
Object Diagrams

- *Model the instances of things described by a class.*
- *Each object diagram shows a set of objects and their interrelationships at a point in time.*
- *Used to model a snapshot of the application.*
- *Each object has an optional name and set of classes it is an instance of, also values for attributes of these classes.*

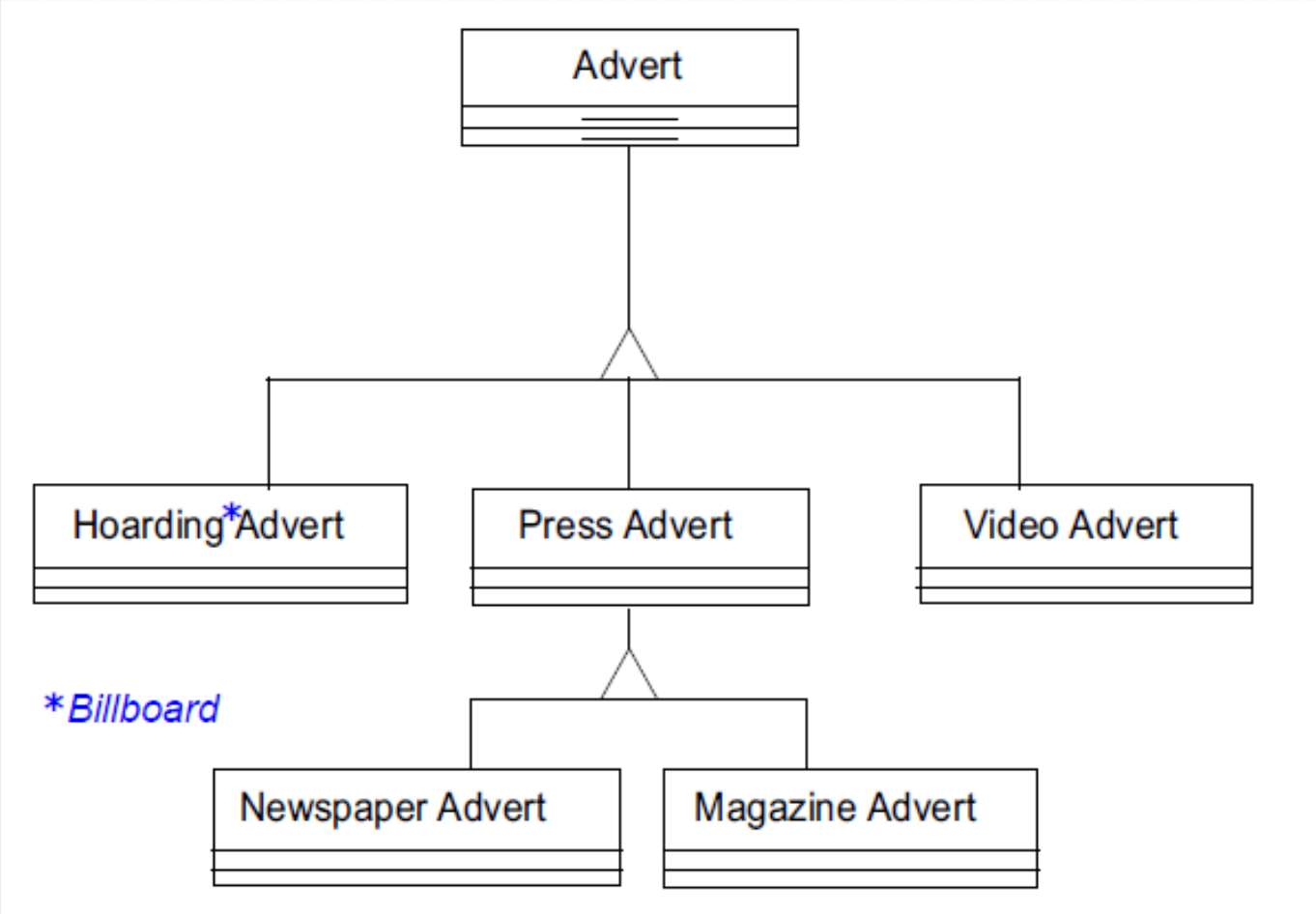


Multi objects

- *A **multi object** is a set of objects, with an undefined number of elements*

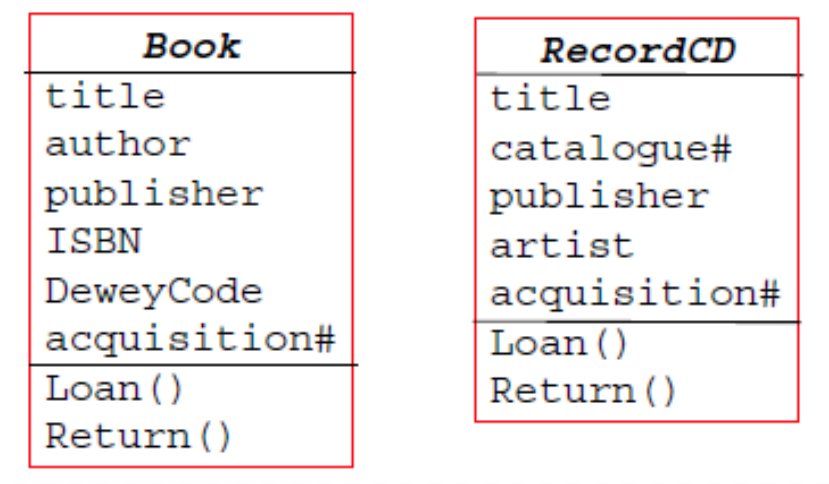


Finding Inheritance

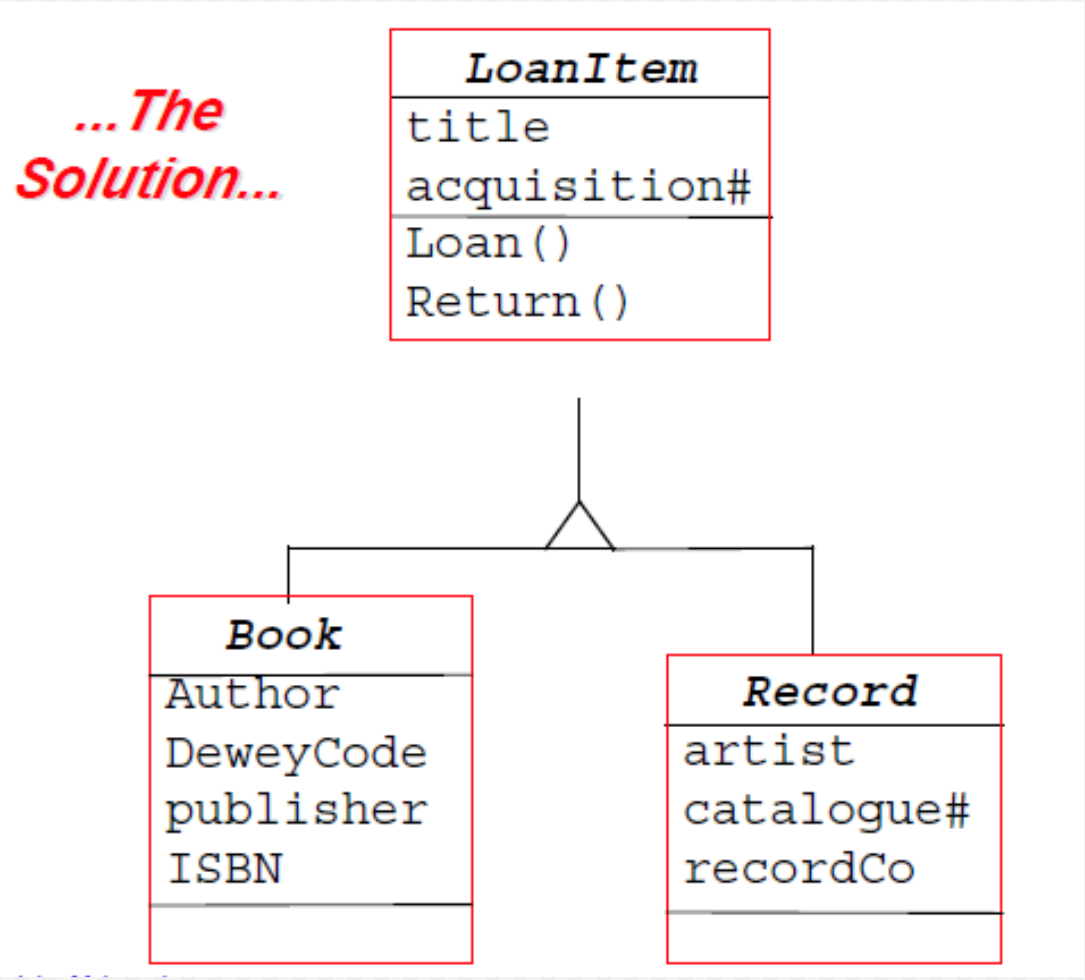


Finding Inheritance

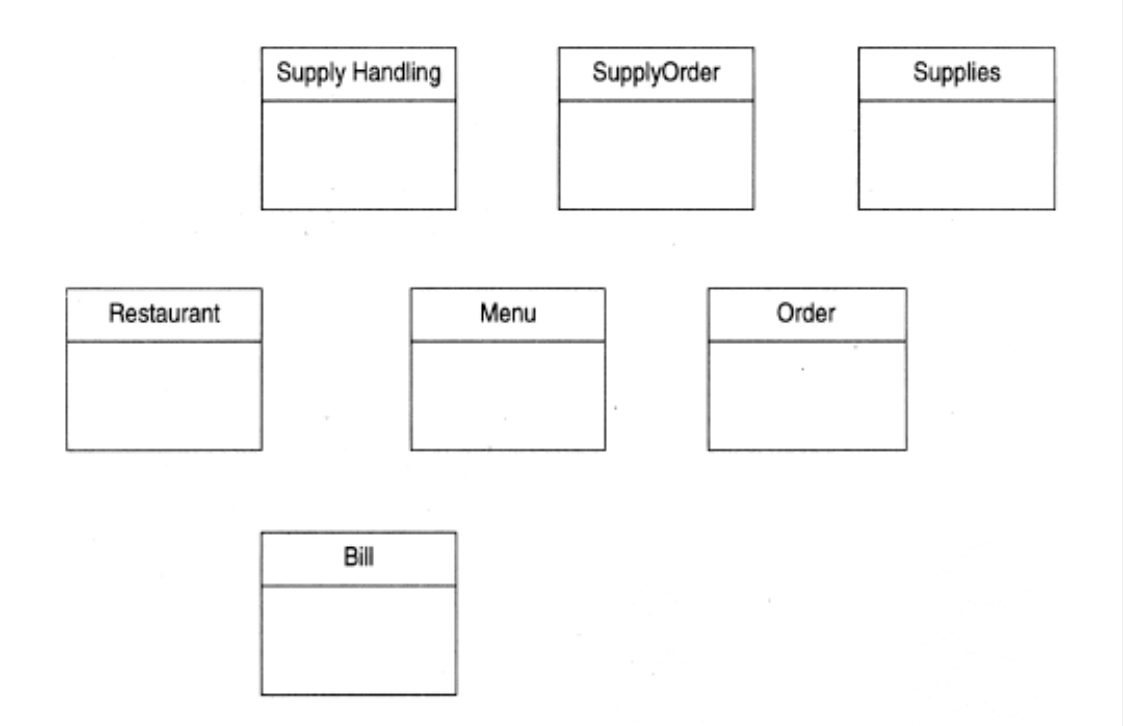
- *Sometimes we find inheritance bottom-up: we have several classes and we realize that they have attributes and operations in common, so we group those attributes and operations together in a common super-class.*
- *Define a suitable generalization of these classes and redraw the diagram.*



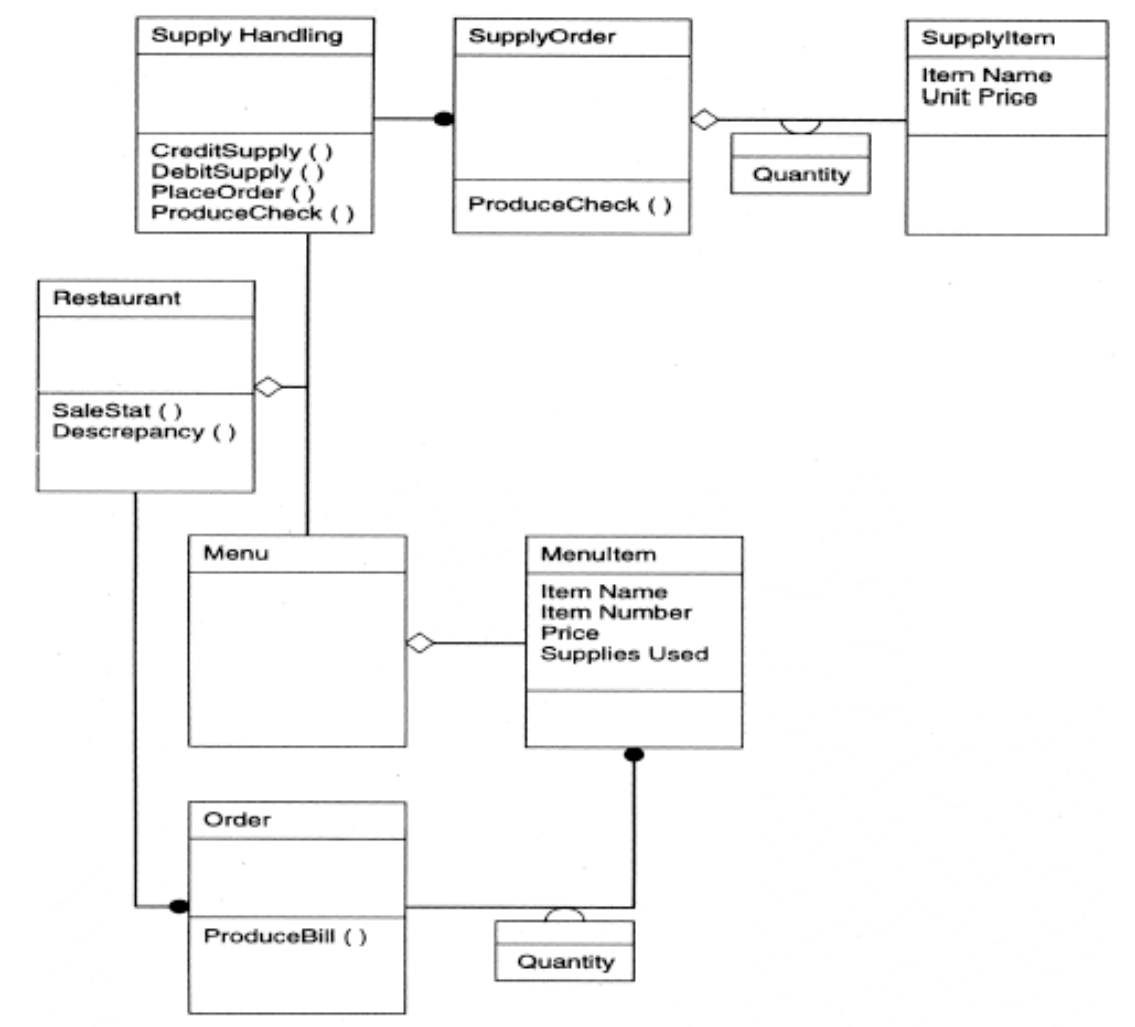
Finding Inheritance

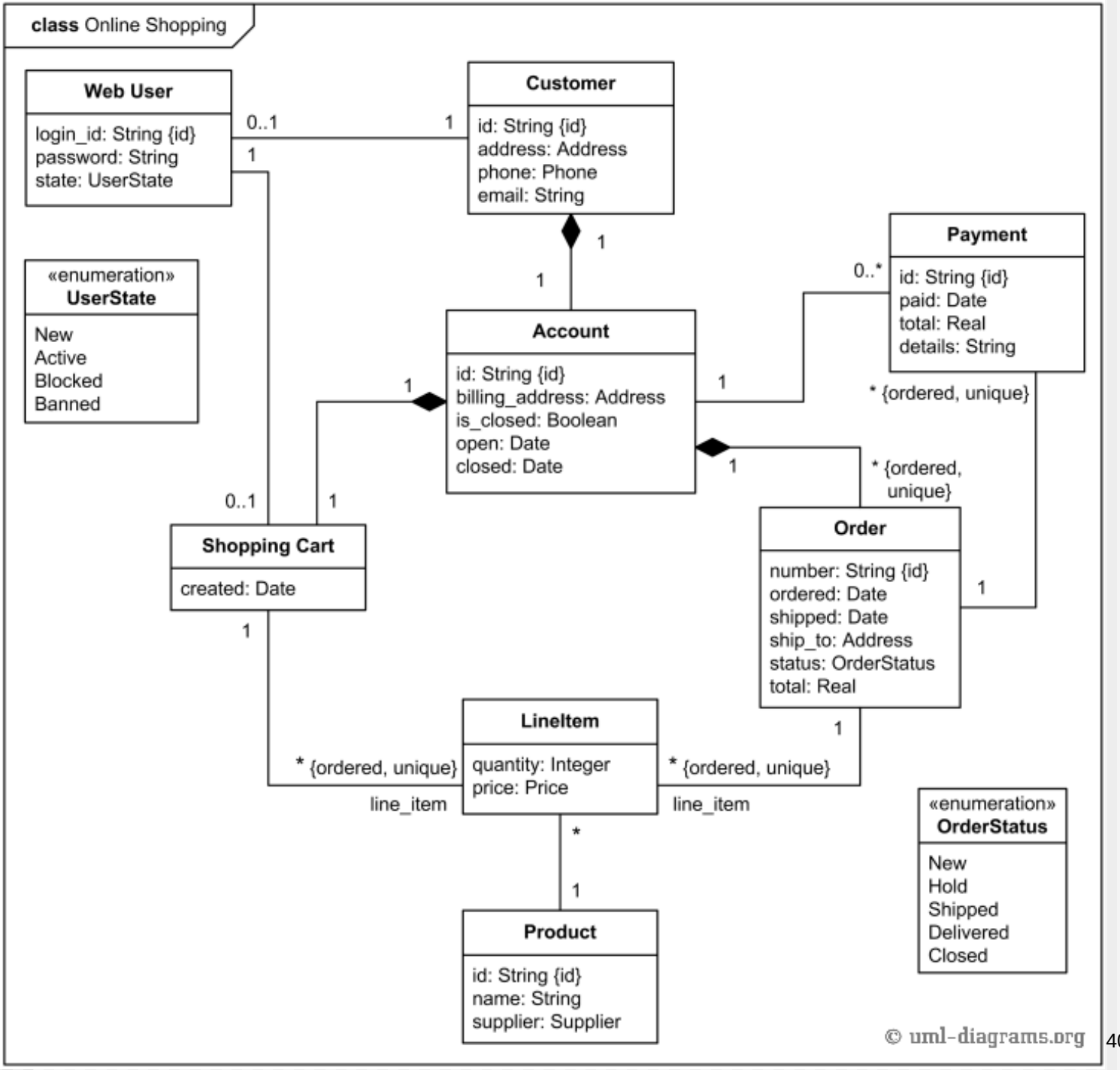


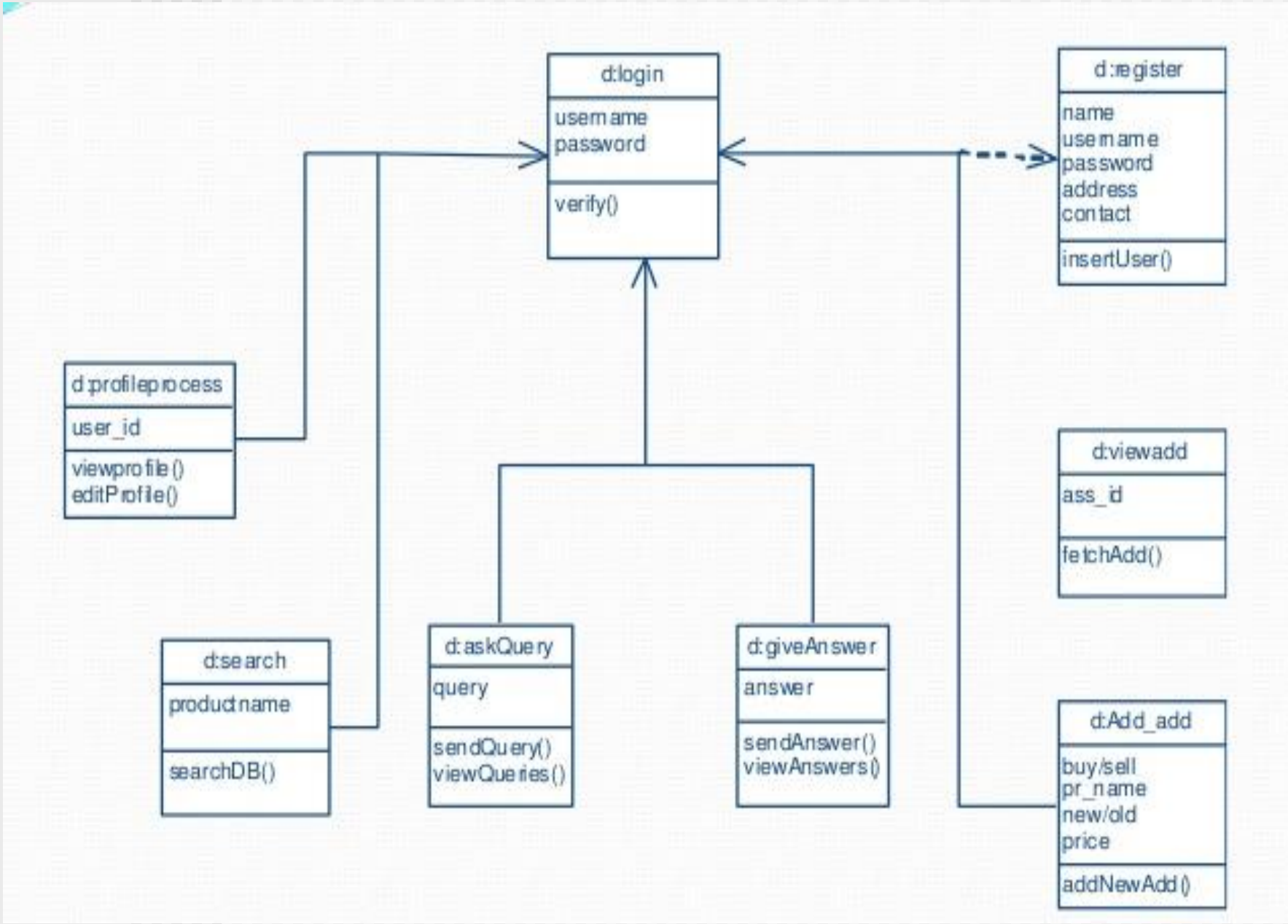
Restaurant example: Initial classes



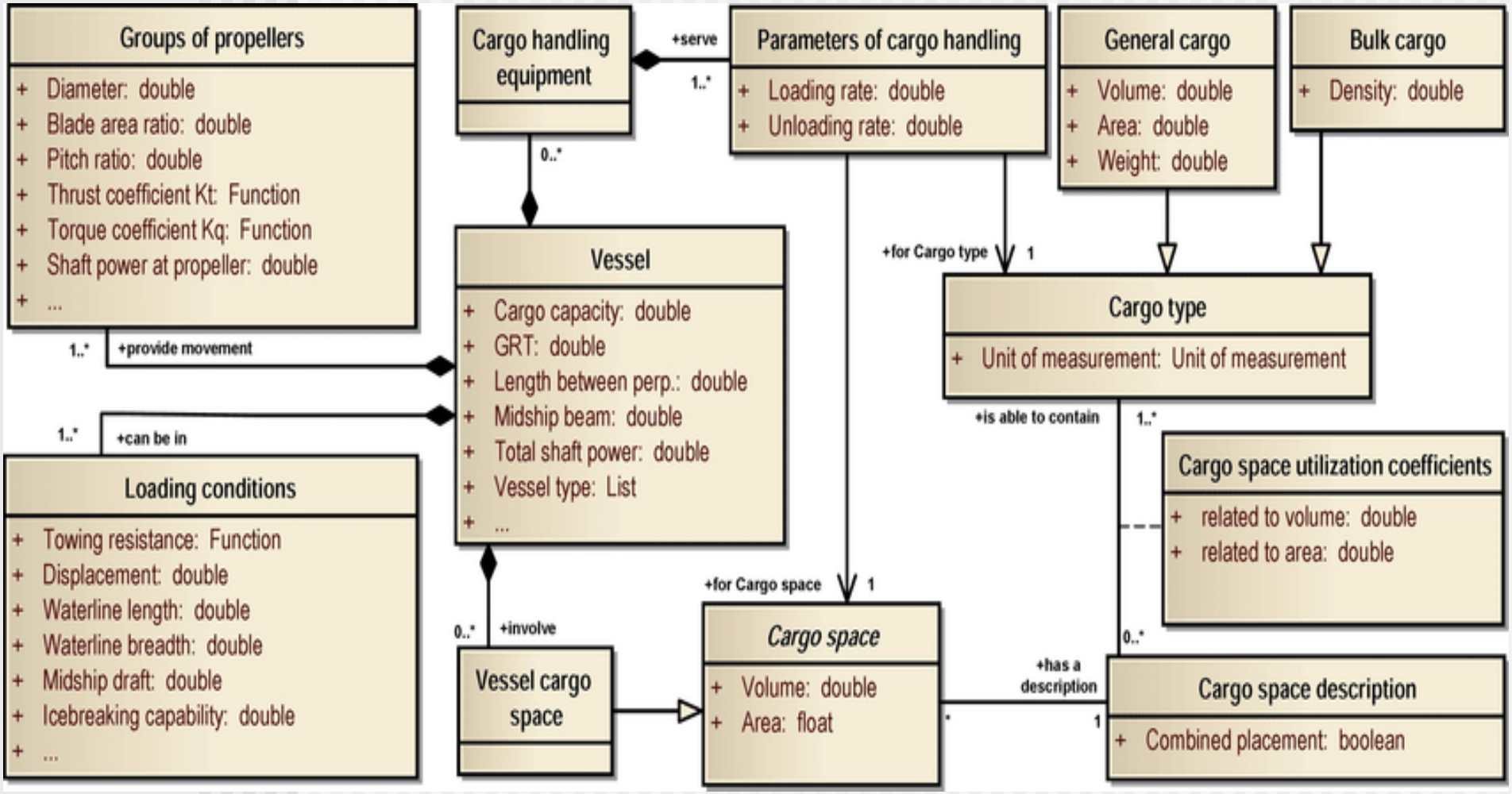
Restaurant example: Initial classes



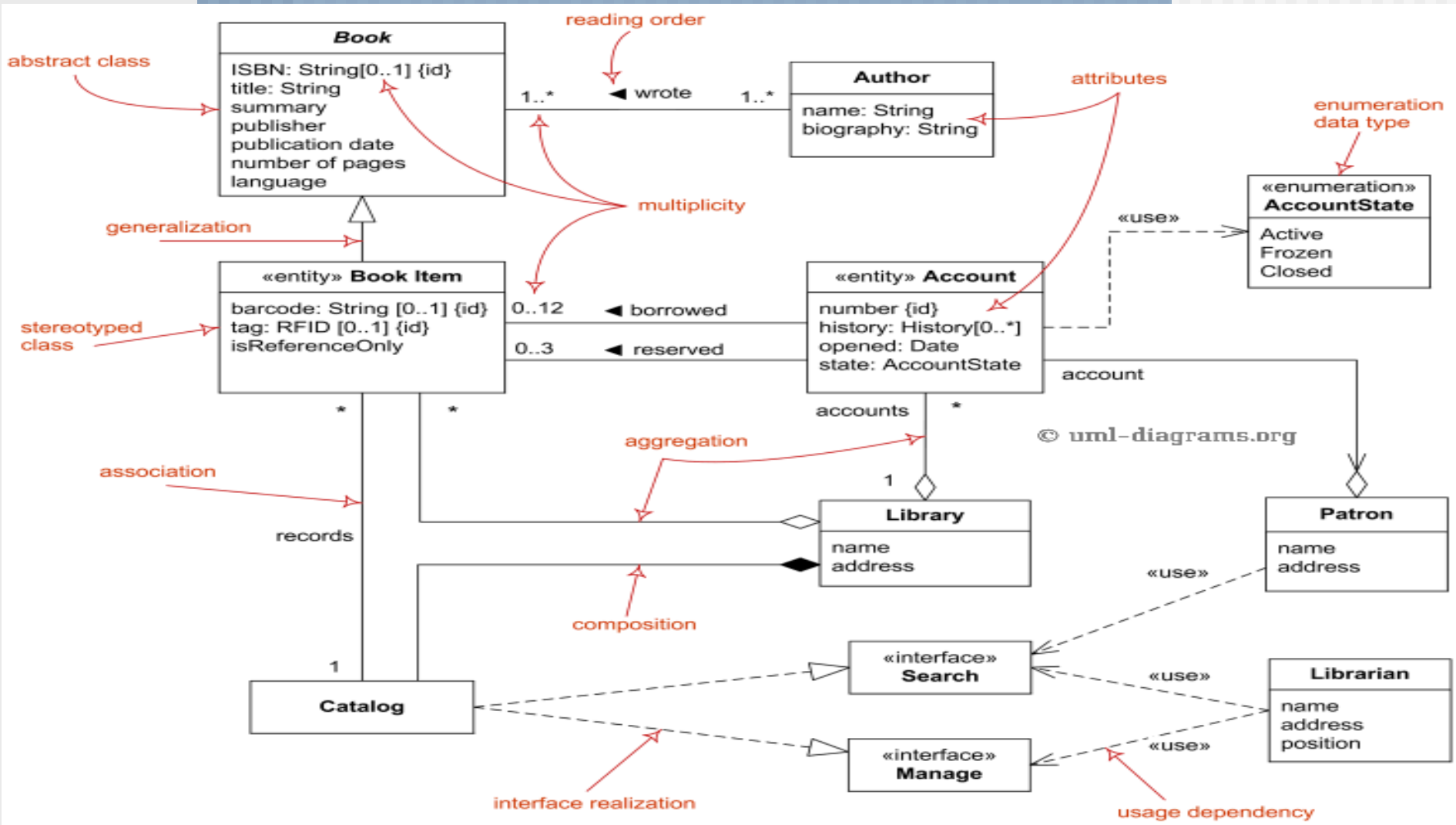

class Online Shopping
Web User
login_id: String {id}
password: String
state: UserState
0..1
1
Customer
id: String {id}
address: Address
phone: Phone
email: String
1
1
«enumeration»
UserState
New
Active
Blocked
Banned
Payment
id: String {id}
paid: Date
total: Real
details: String
0..*
Account
id: String {id}
billing_address: Address
is_closed: Boolean
open: Date
closed: Date
1
1
1
1
* {ordered, unique}
* {ordered, unique}
0..1
1
Shopping Cart
created: Date
Order
number: String {id}
ordered: Date
shipped: Date
ship_to: Address
status: OrderStatus
total: Real
1
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


d:login
username
password
verify()
d:register
name
username
password
address
contact
insertUser()
d:profileprocess
user_id
viewprofile()
editProfile()
d:viewadd
ass_id
fetchAdd()
d:search
productname
searchDB()
d:askQuery
query
sendQuery()
viewQueries()
d:giveAnswer
answer
sendAnswer()
viewAnswers()
d:Add_add
buy/sell
pr_name
new/old
price
addNewAdd()

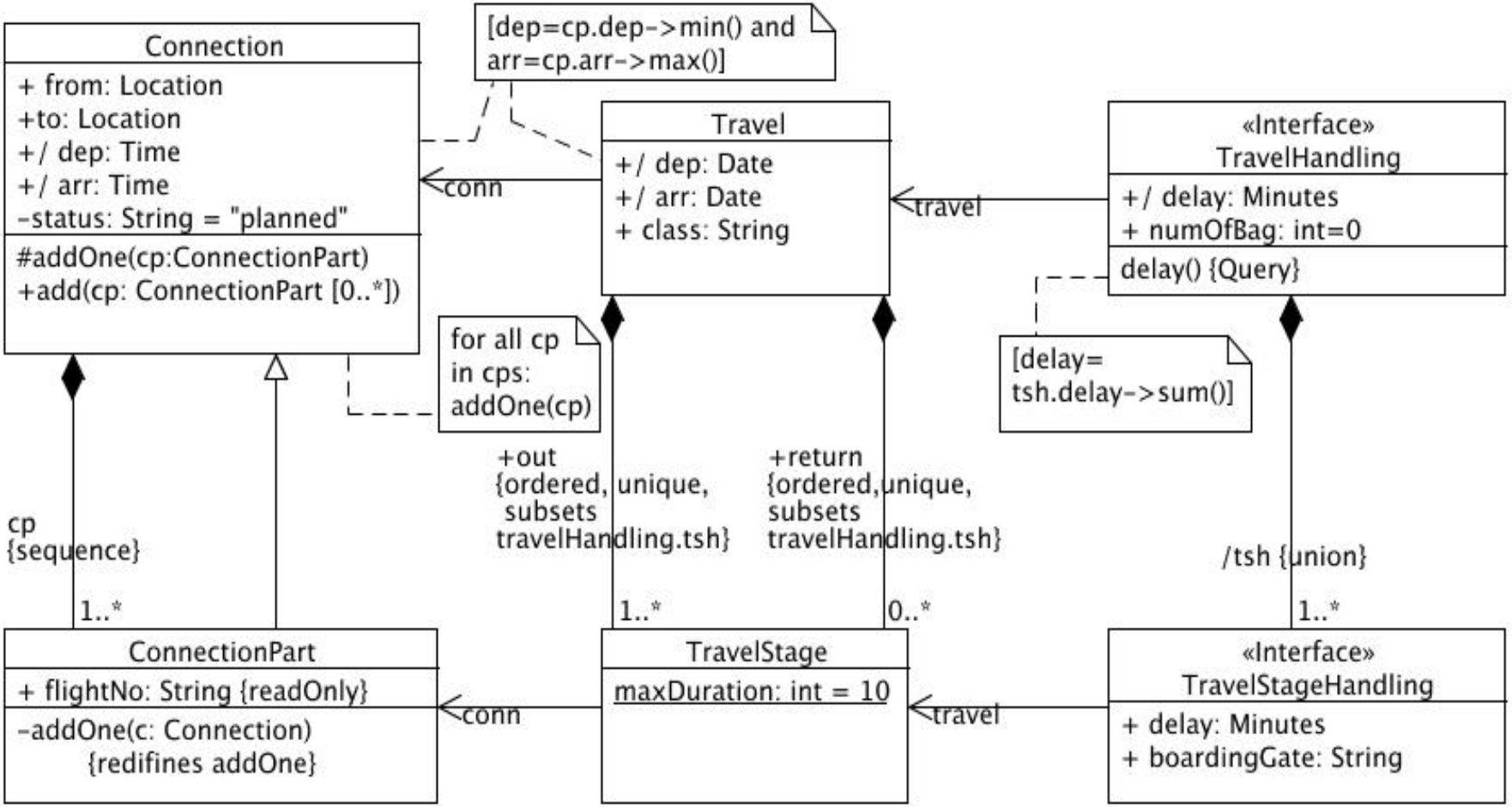
Ship & Cargo object model

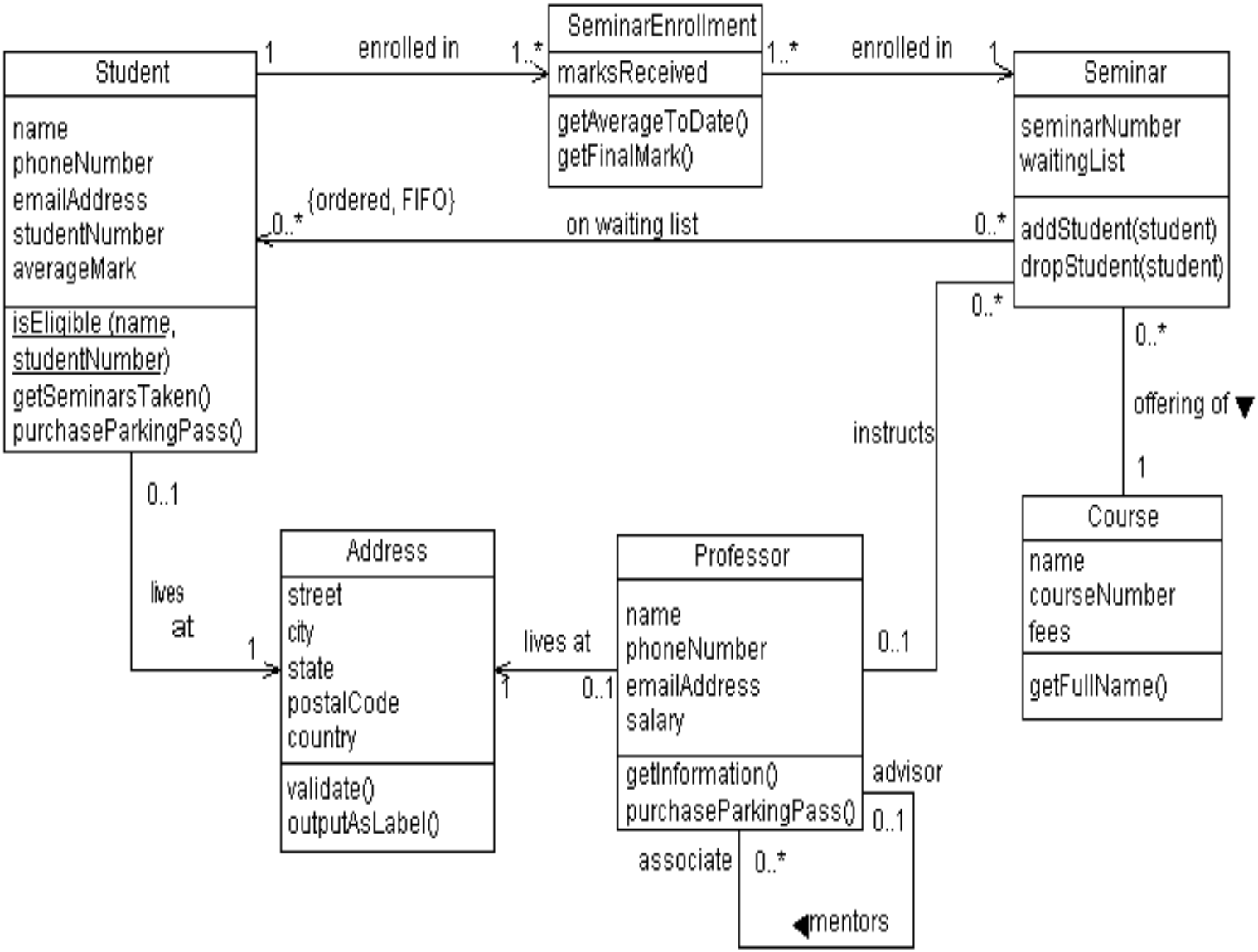


Library Management object model



Air Ticket Reservation Design Model




Student
name
phoneNumber
emailAddress
studentNumber
averageMark
isEligible (name, studentNumber)
getSeminarsTaken()
purchaseParkingPass()
1
enrolled in
1..*
SeminarEnrollment
marksReceived
getAverageToDate()
getFinalMark()
1..*
enrolled in
1
Seminar
seminarNumber
waitingList
addStudent(student)
dropStudent(student)
{ordered, FIFO}
0..*
on waiting list
0..*
0..*
instructs
0..1
0..*
offering of
1
Course
name
courseNumber
fees
getFullName()
0..1
lives at
1
Address
street
city
state
postalCode
country
validate()
outputAsLabel()
lives at
1
0..1
Professor
name
phoneNumber
emailAddress
salary
getInformation()
purchaseParkingPass()
advisor
0..1
associate
0..*
mentors

References

[Booch99] Booch, Grady, James Rumbaugh, Ivar Jacobson,
The Unified Modeling Language User Guide, Addison Wesley, 1999
[Rambaugh99] Rumbaugh, James, Ivar Jacobson, Grady Booch, The Unified
Modeling Language Reference Manual, Addison Wesley, 1999
[Jacobson99] Jacobson, Ivar, Grady Booch, James Rumbaugh, The Unified
Software Development Process, Addison Wesley, 1999
[Fowler, 1997] Fowler, Martin, Kendall Scott, UML Distilled
(Applying the Standard Object Modeling Language),
Addison Wesley, 1997.
[Brown99] First draft of these slides were created by James Brown.