UNIVERSITÀ DEGLI STUDI DI TERAMO

UML Class Diagram

Prof. ssa Romina Eramo
Università degli Studi di Teramo
Dipartimento di Scienze della Comunicazione
reramo@unite.it

Models

- The goal is not to build documents but software that satisfies the user's requests with predictable times and methods
- Templates help build good software
- Proven and Accepted Engineering Technical Modeling
  - Mathematical models for forecasts
  - Models for building cars, cars

Models

- What is it?
  - A model is a simplification of reality
- Because
  - Models are built to better understand the system that is being developed

Models

- Purposes
  - They help to describe/visualize a system as it is and as we want it to be
  - Specify the structure or behavior of a system
  - They give a template that guides you to build a system
  - Document the decisions made

Models

Models of complex systems are built because they cannot be understood in their entirety

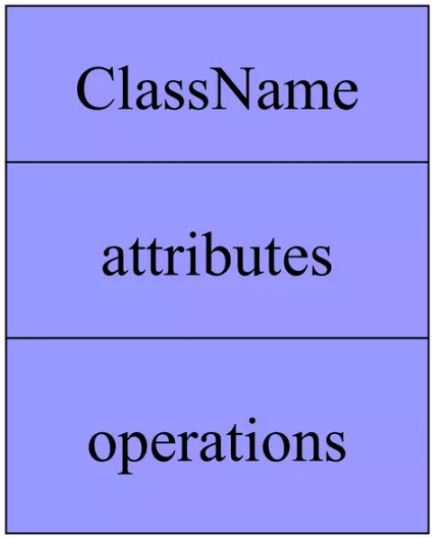
UML: What is it?

- Acronym Unified Modeling Language
  - Language: language
  - Unified: standard, unified model
  - Modeling: For Modeling
- Language for
  - Visualize
  - Specify
  - Build
  - Document
  - the elaborations of a software system

Class Diagrams

- Concept of class diagram
- Creating class diagram

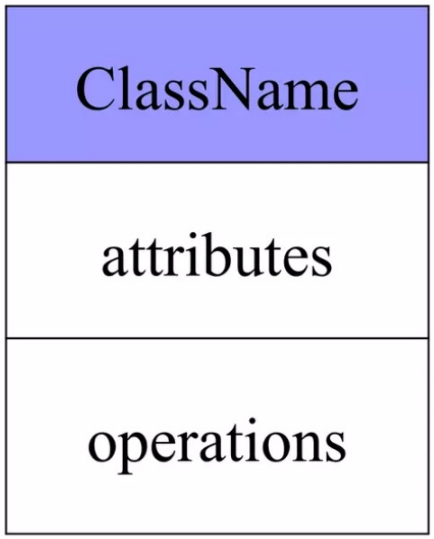
Classes



A *class* is a description of a set of objects that share the same attributes, operations, relationships, and semantics.

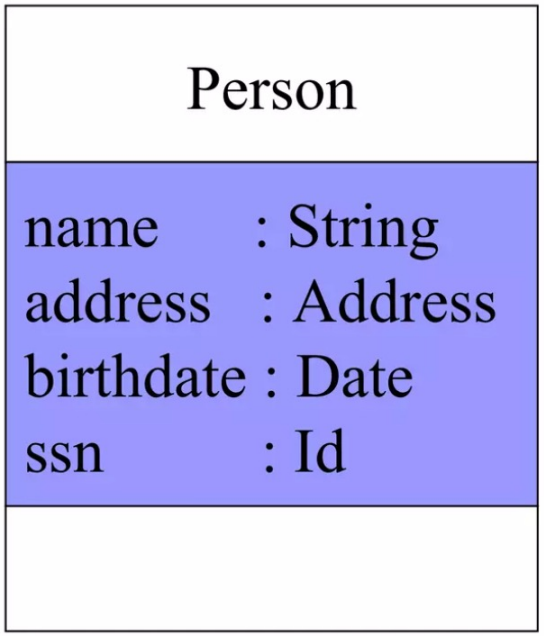
Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

Class Names



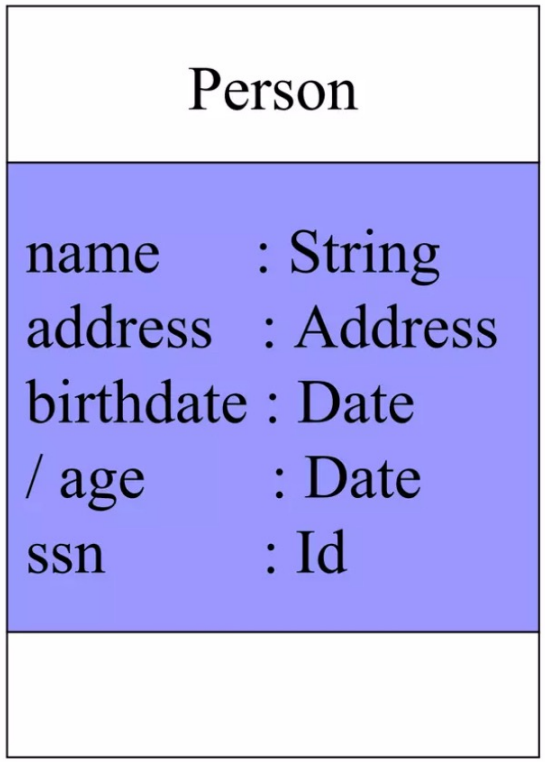
The name of the class is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.

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)



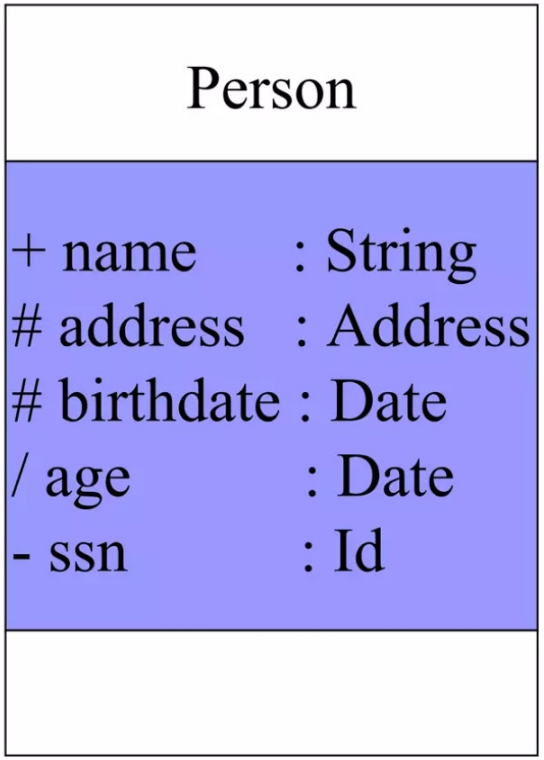
Attributes are usually listed in the form:

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

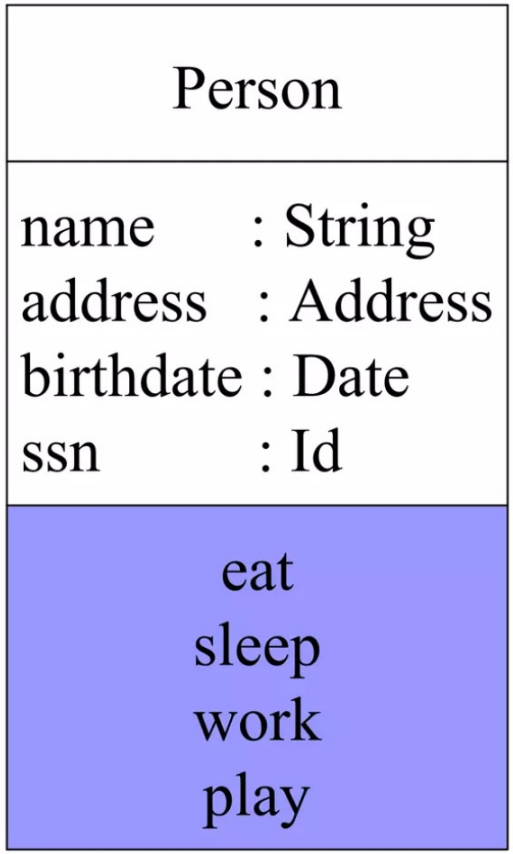
Class Attributes (Cont'd)



Attributes can be:

+ public
# protected
- private
/ derived

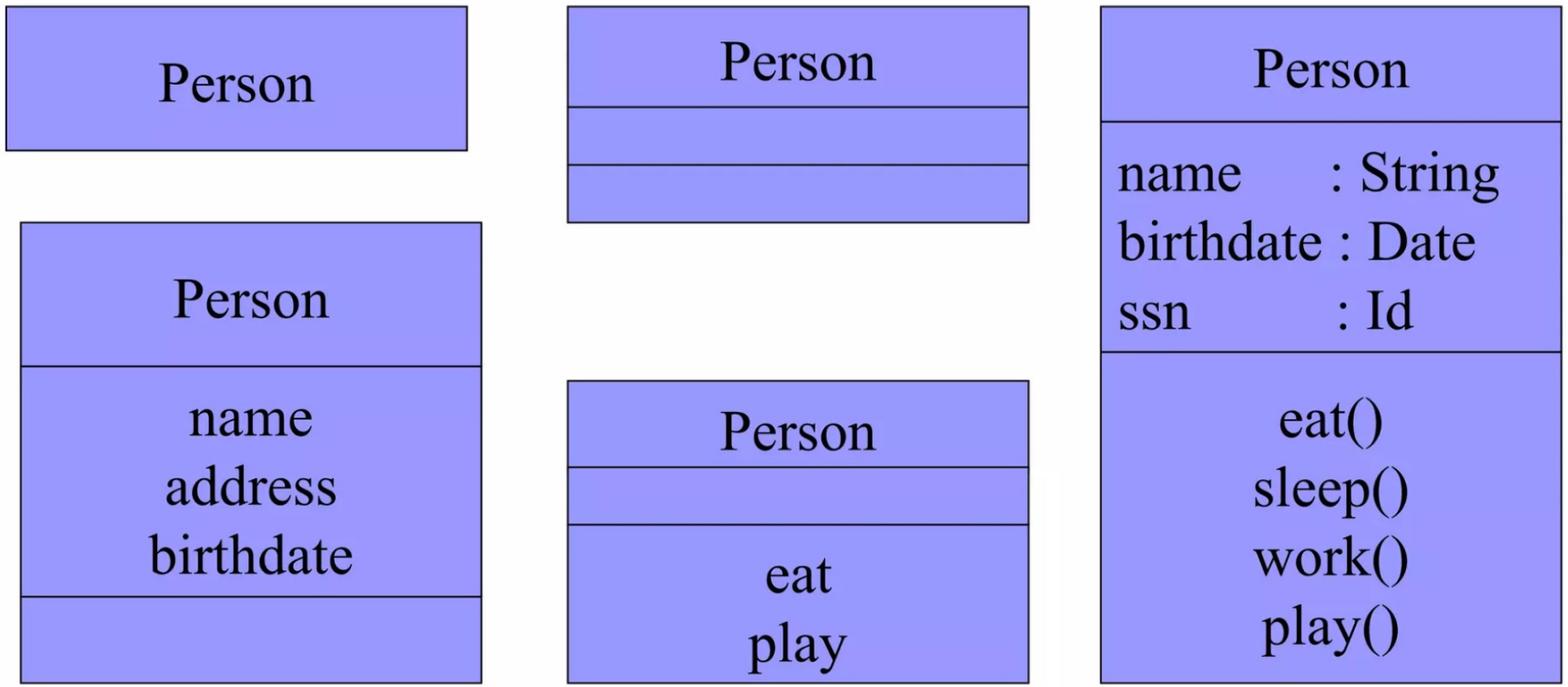
Class Operations



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

Depicting Classes

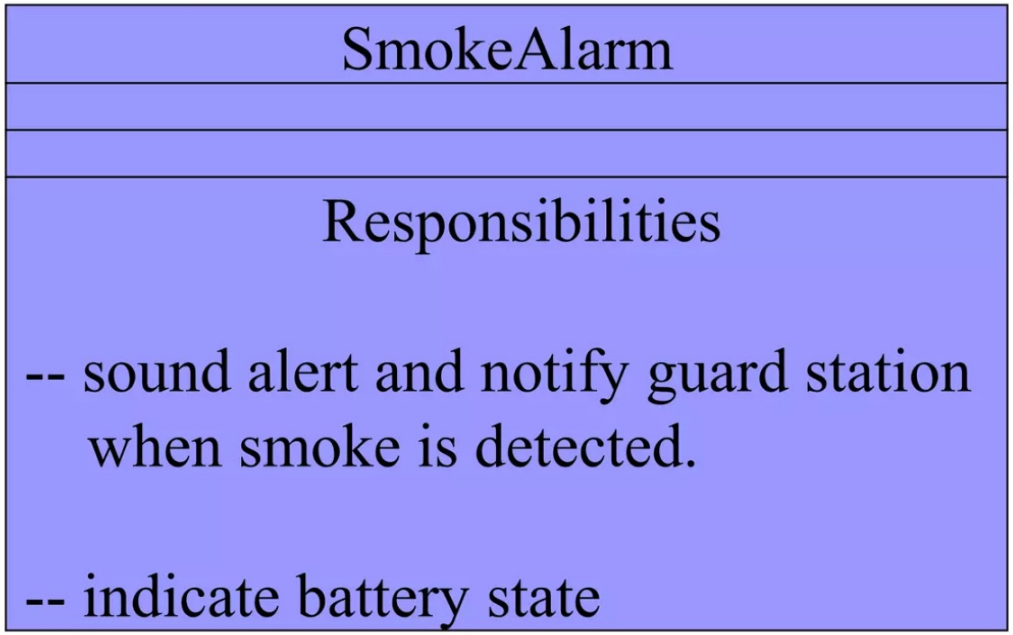
When drawing a class, you needn't show attributes and operation in every diagram.



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.



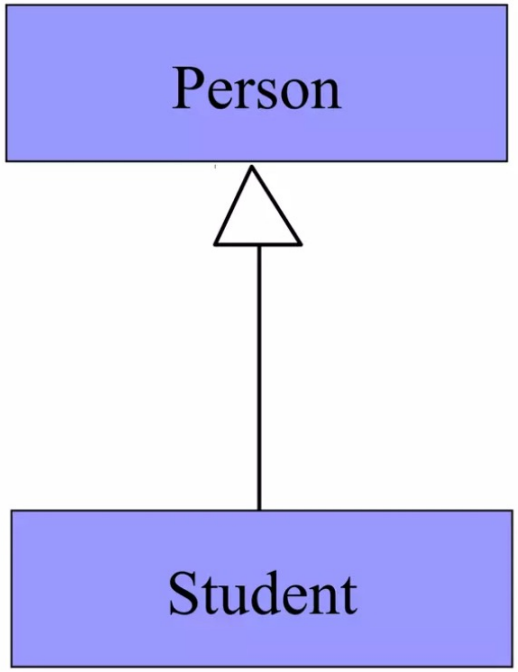
Relationships

In UML, object interconnections (logical or physical), are modeled as relationships.

There are three kinds of relationships in UML:

- dependencies
- generalizations
- associations

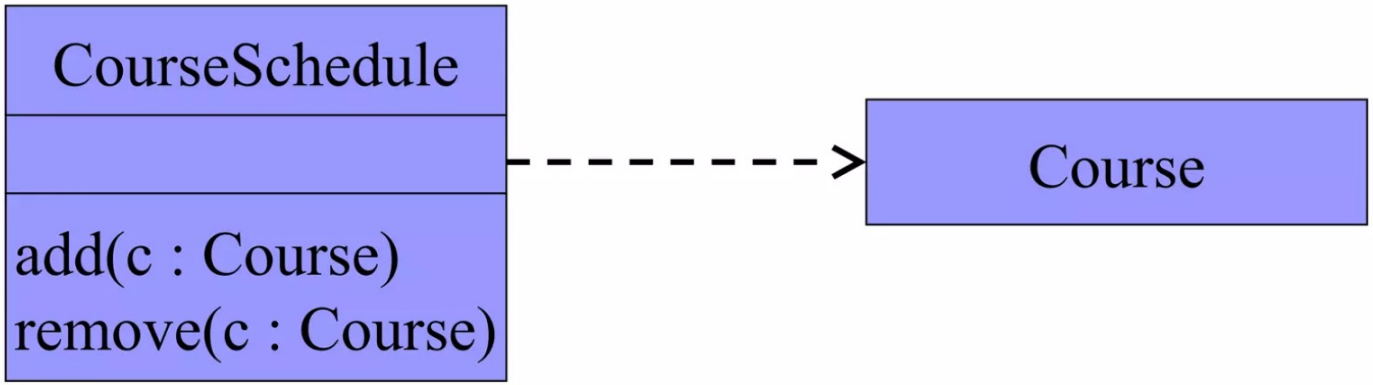
Generalization Relationships



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.

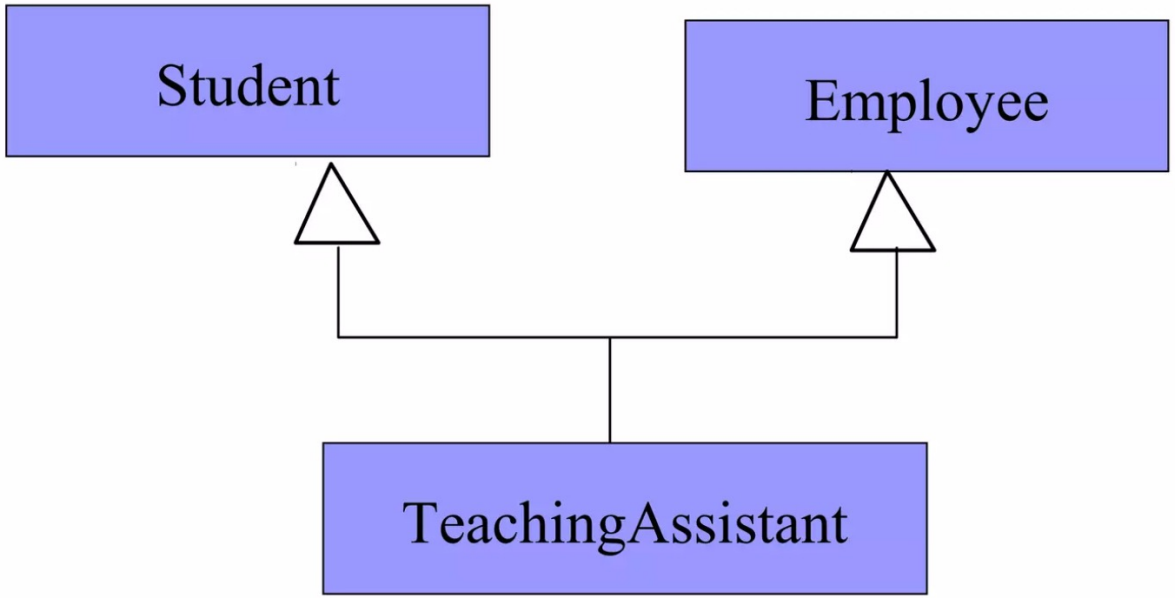
Dependency Relationships

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*.



Generalization Relationships (Cont'd)

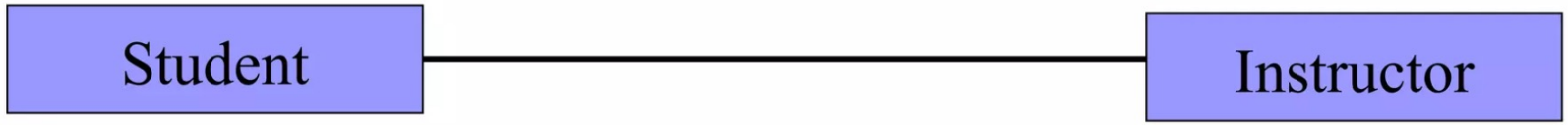
UML permits a class to inherit from multiple super classes, although some programming languages (*e.g.,* Java) do not permit multiple inheritance.



Association Relationships

If two classes in a model need to communicate with each other, there must be link between them.

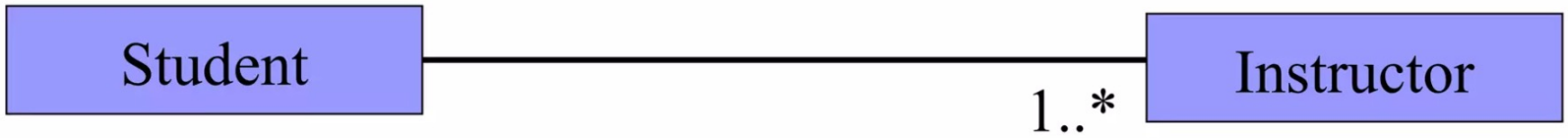
An *association* denotes that link.



Association Relationships (Cont'd)

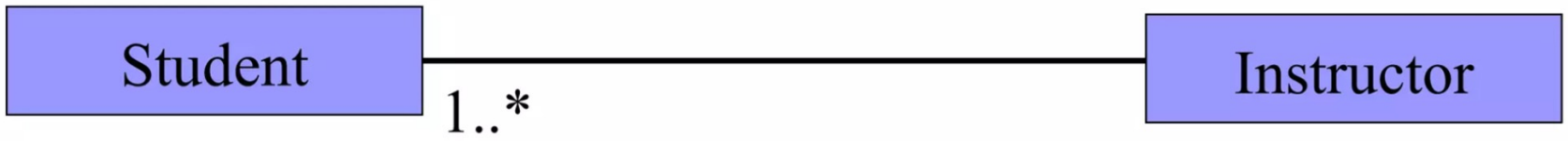
We can indicate the *multiplicity* of an association by adding *multiplicity adornments* to the line denoting the association.

The example indicates that a *Student* has one or more *Instructors*:



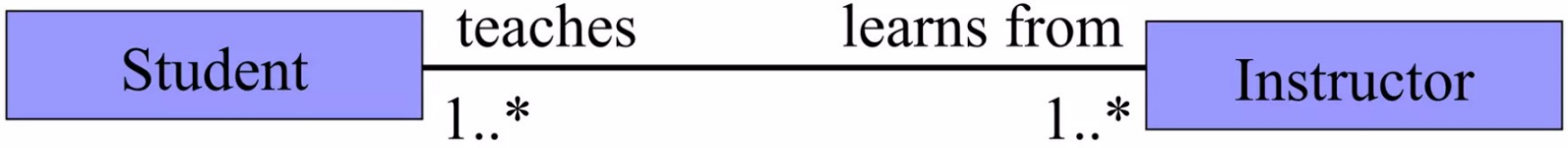
Association Relationships (Cont'd)

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



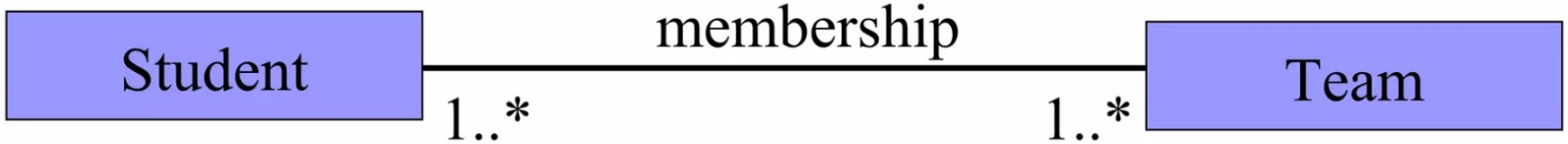
Association Relationships (Cont'd)

We can also indicate the behavior of an object in an association (*i.e.,* the *role* of an object) using *role names.*



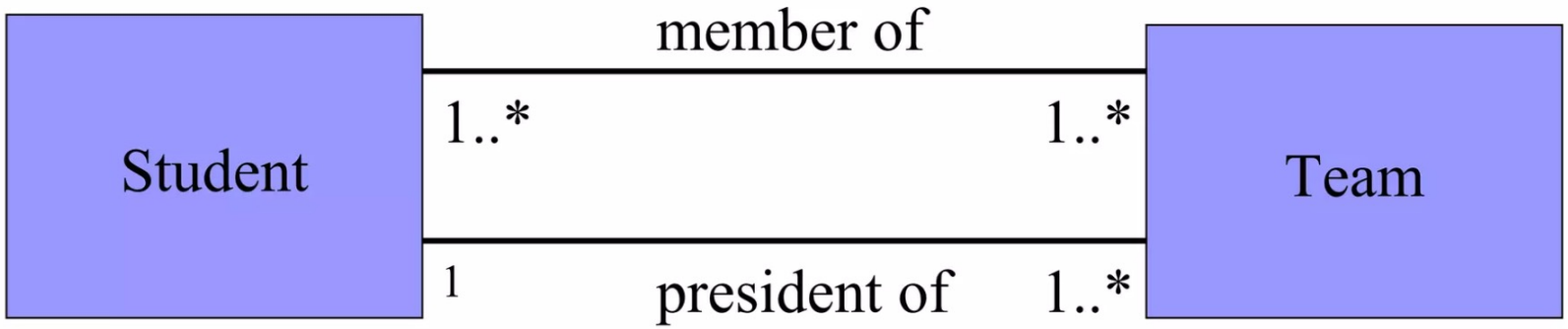
Association Relationships (Cont'd)

We can also name the association.



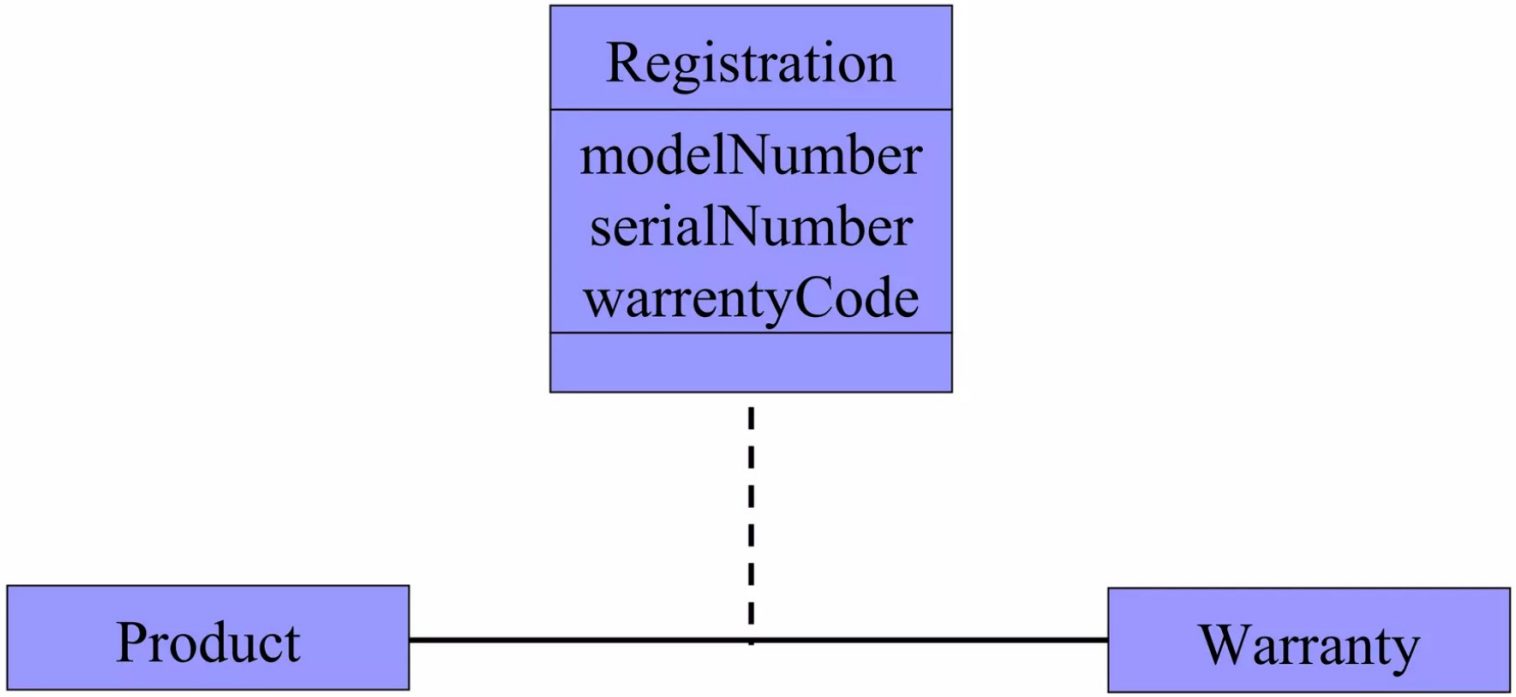
Association Relationships (Cont'd)

We can specify dual associations.

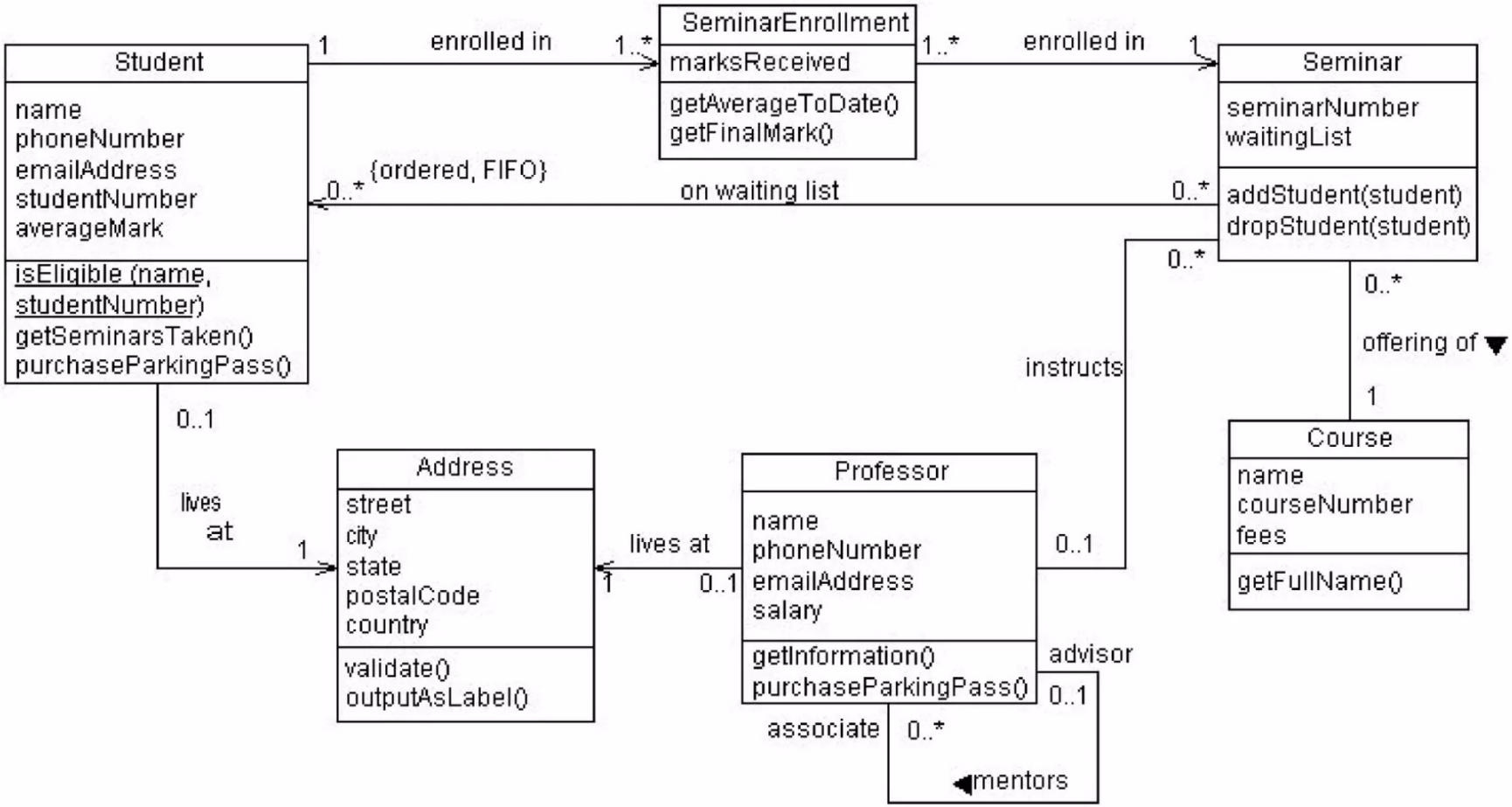


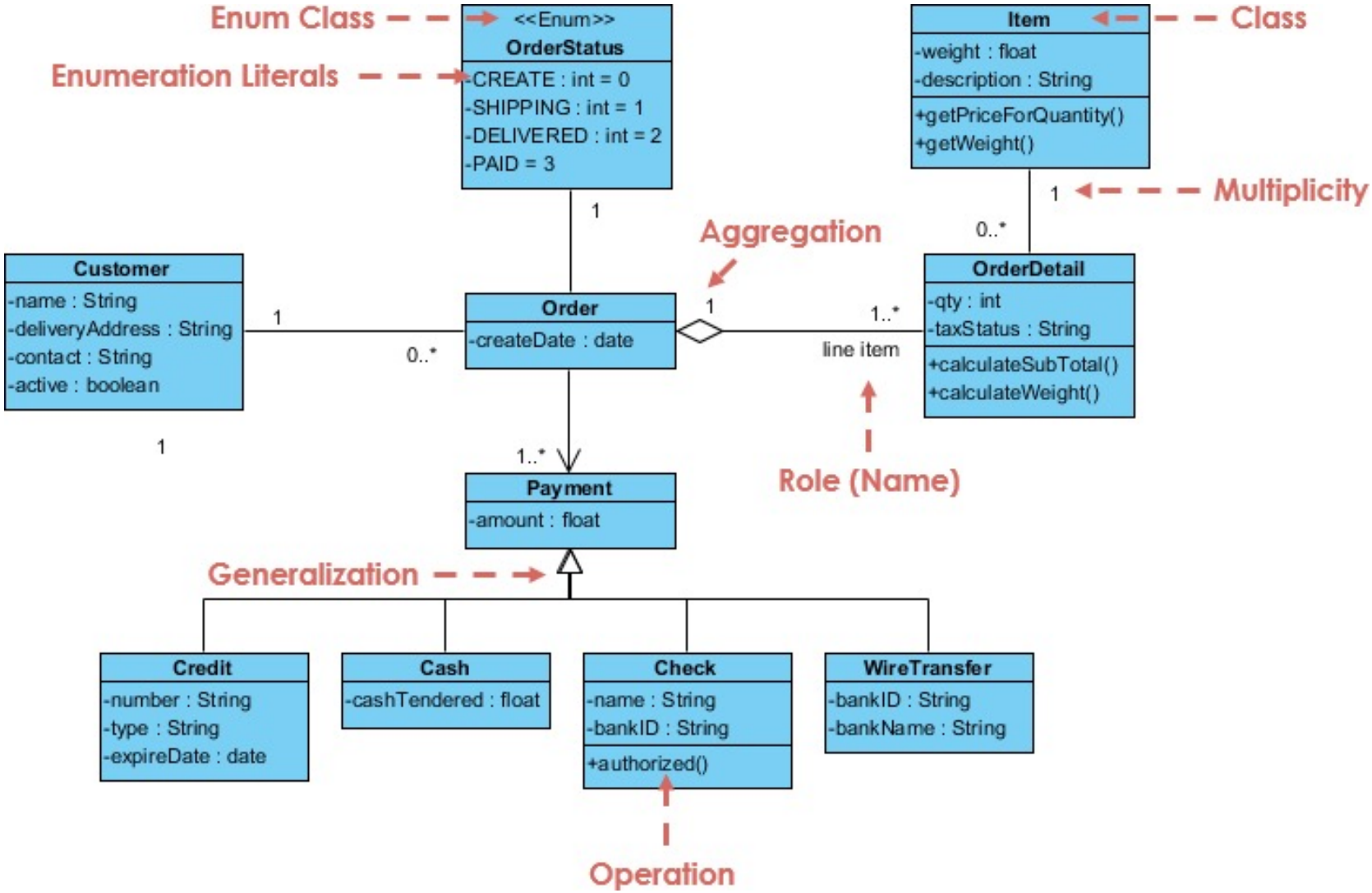
Association Relationships (Cont'd)

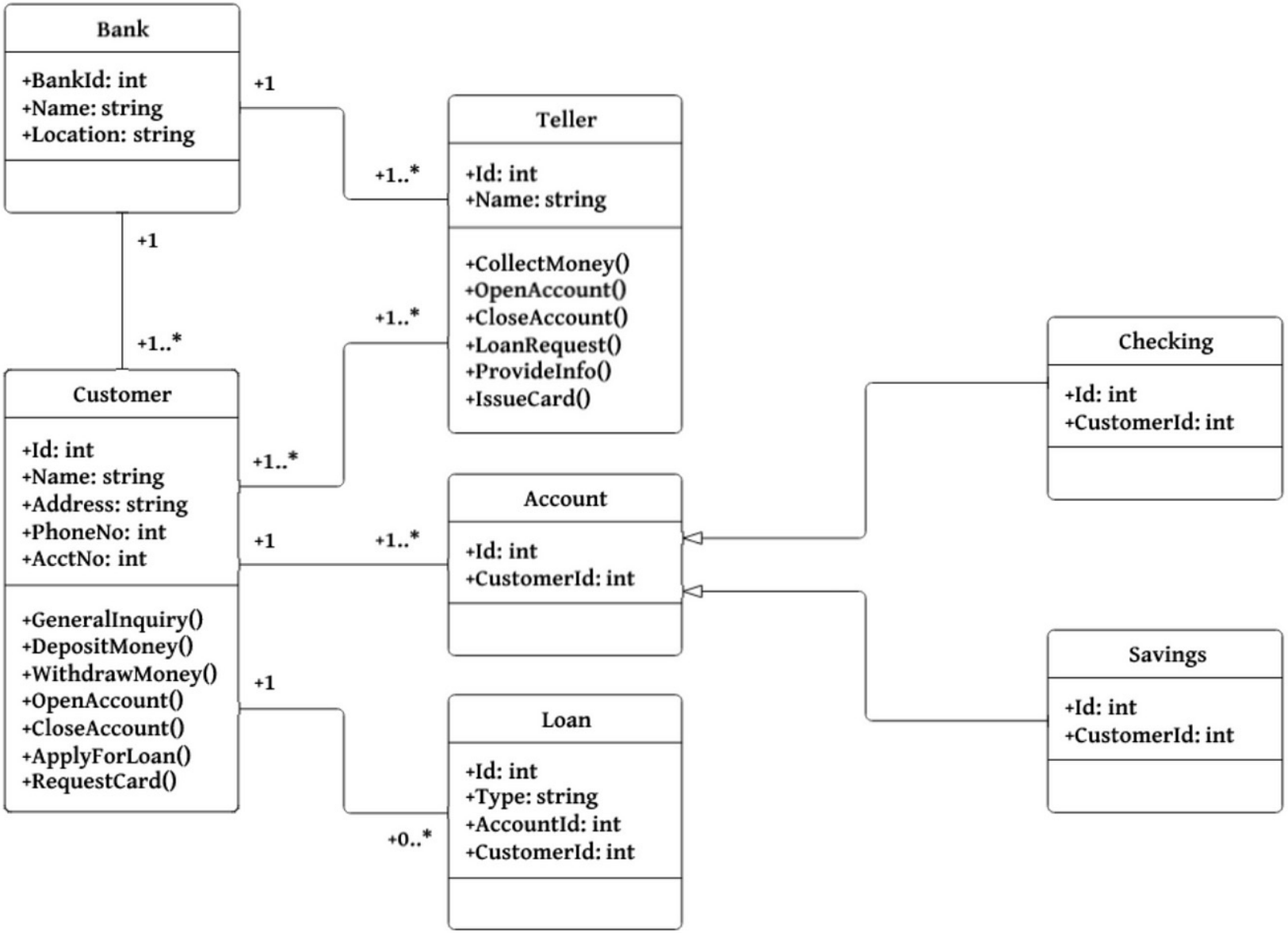
Associations can also be objects themselves, called *link classes* or an *association classes*.



Example


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)
0..*
{ordered, FIFO}
on waiting list
0..*
0..*
instructs
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()
0..1
advisor
0..1
associate
0..*
mentors


Enum Class
<<Enum>>
OrderStatus
Enumeration Literals
-CREATE : int = 0
-SHIPPING : int = 1
-DELIVERED : int = 2
-PAID = 3
Item
Class
-weight : float
-description : String
+getPriceForQuantity()
+getWeight()
1
Multiplicity
0..*
Aggregation
Customer
-name : String
-deliveryAddress : String
-contact : String
-active : boolean
1
0..*
Order
-createDate : date
1
1..*
line item
OrderDetail
-qty : int
-taxStatus : String
+calculateSubTotal()
+calculateWeight()
1
1..*
Role (Name)
Payment
-amount : float
Generalization
Credit
-number : String
-type : String
-expireDate : date
Cash
-cashTendered : float
Check
-name : String
-bankID : String
+authorized()
WireTransfer
-bankID : String
-bankName : String
Operation


Bank
+BankId: int
+Name: string
+Location: string
+1
+1..*
Teller
+Id: int
+Name: string
+CollectMoney()
+OpenAccount()
+CloseAccount()
+LoanRequest()
+ProvideInfo()
+IssueCard()
+1
+1..*
+1..*
+1..*
Customer
+Id: int
+Name: string
+Address: string
+PhoneNo: int
+AcctNo: int
+GeneralInquiry()
+DepositMoney()
+WithdrawMoney()
+OpenAccount()
+CloseAccount()
+ApplyForLoan()
+RequestCard()
+1
+1..*
Account
+Id: int
+CustomerId: int
+1
+0..*
Loan
+Id: int
+Type: string
+AccountId: int
+CustomerId: int
Checking
+Id: int
+CustomerId: int
Savings
+Id: int
+CustomerId: int