7. Object Oriented Modeling

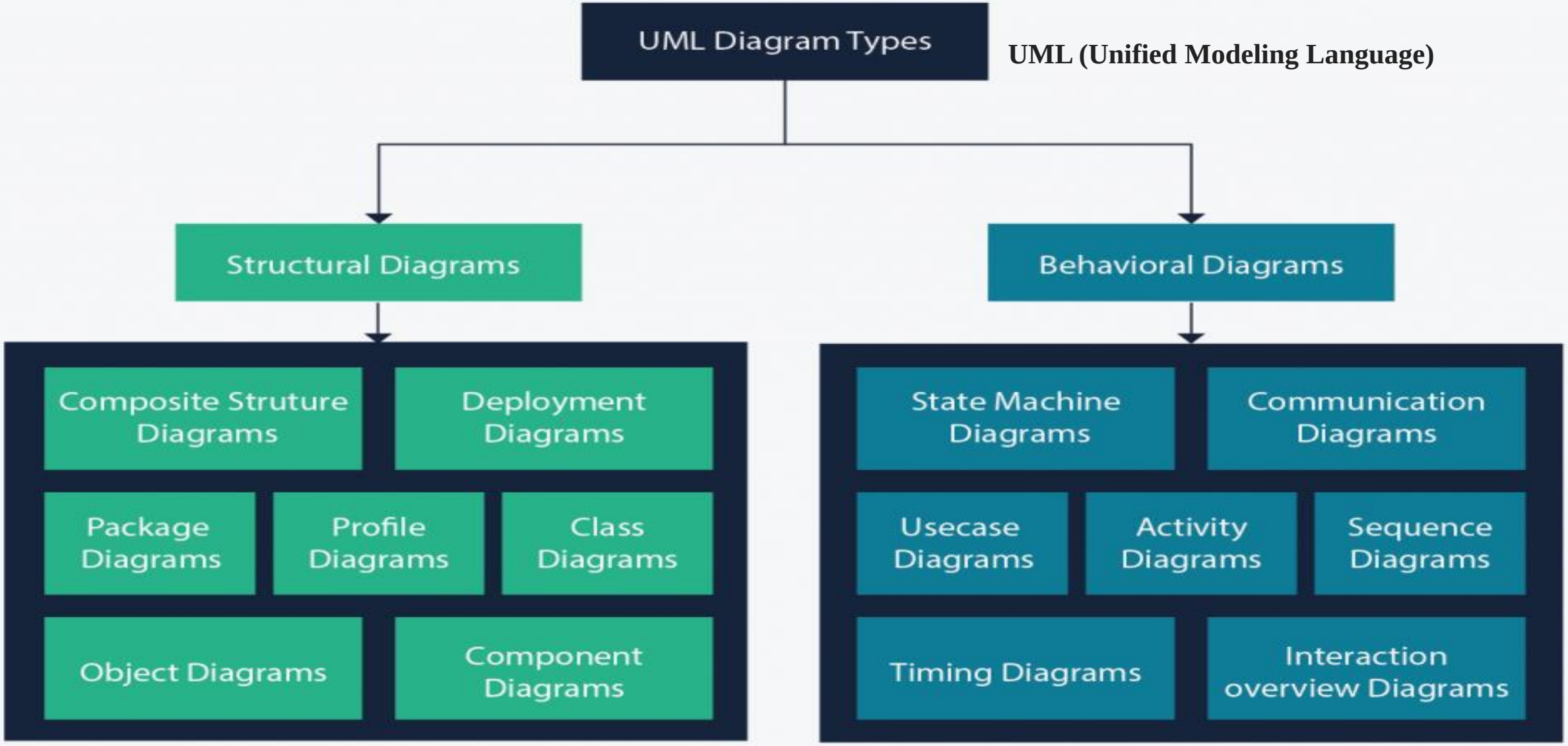
CSE333:Software Engineering

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

Learning Goals

- Understand what object-oriented systems analysis and design is and appreciate its usefulness.
- Comprehend the concepts of unified modeling language (UML), the standard approach for modeling a system in the object-oriented world.
- Apply the steps used in UML to break down the system into a use case model and then a class model.
- Diagram systems with the UML toolset so they can be described and properly designed.
- Document and communicate the newly modeled object-oriented system to users and other analysts.

UML Diagram Types



UML (Unified Modeling Language)

Diagram Types

- A **class diagram** represents a static view of the system. It describes the attributes and operations of classes.
- **Class diagrams** are the most widely used modeling diagram for object-oriented systems because they can be directly mapped with object-oriented languages.
- User, Customer, Administrator, Order, OrderDetails are classes. Each class consist of attributes and methods. Attributes describe the properties while methods describe the behaviors or operations.

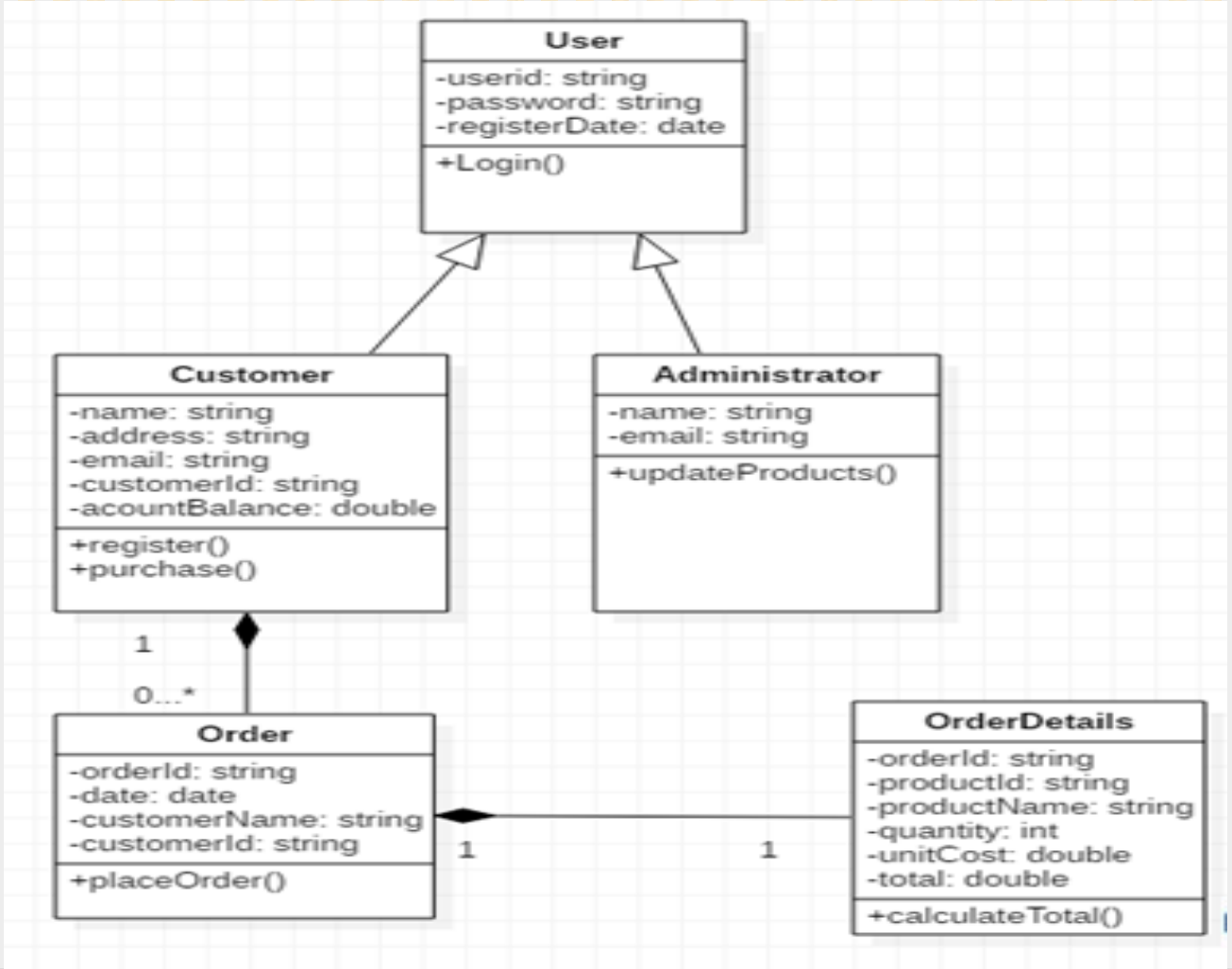
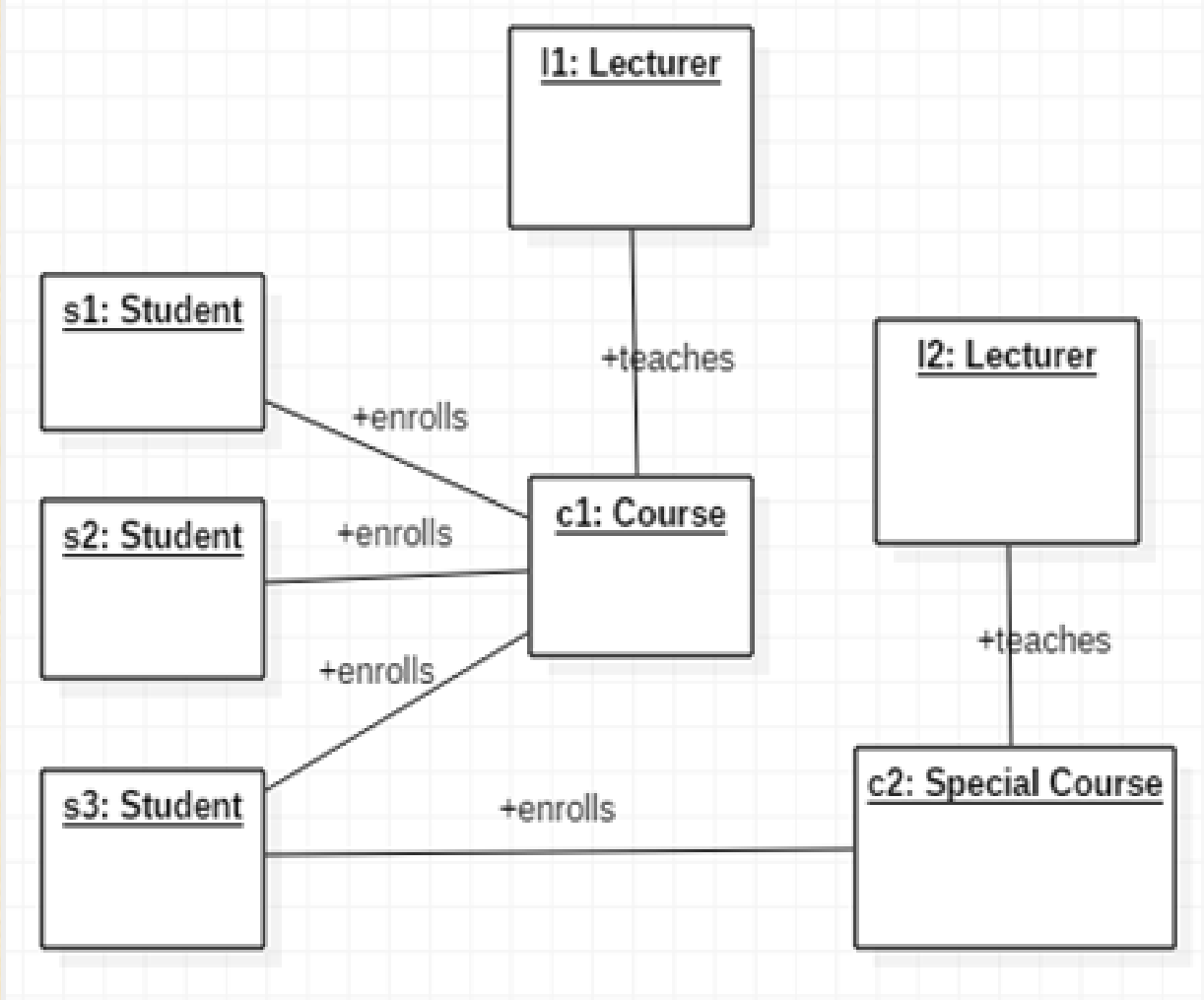
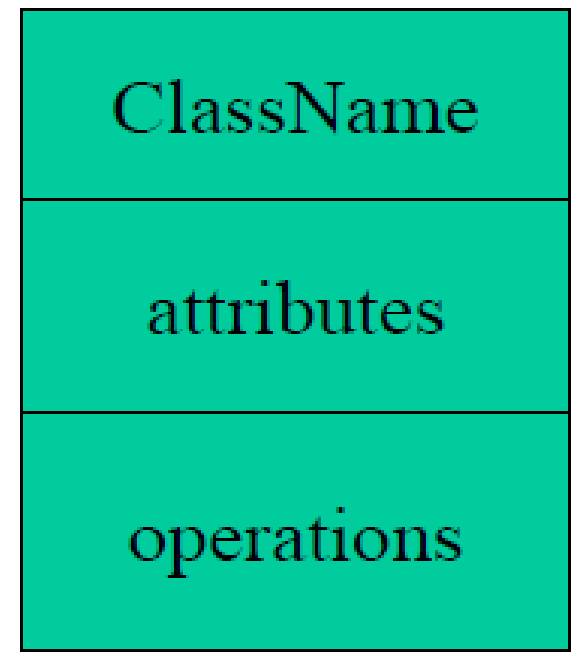


Diagram Types(Cont..)

- Another structural diagram is an object diagram. It is similar to a class diagram, but it focuses on objects.
- The basic concepts of **object diagram** are similar to a class diagram. These diagrams help to understand object behavior and their relationships at a particular moment.
- The **s1**, **s2**, and **s3** are student objects, and they enroll to c1 course object. The l1 lecturer object teaches the course **c1**. The lecturer object **l2** teaches the special course **c2**. The Student **s3** enrolls to c1 course as well as **c2** special course. This diagram illustrates how a set of objects relates to each other.

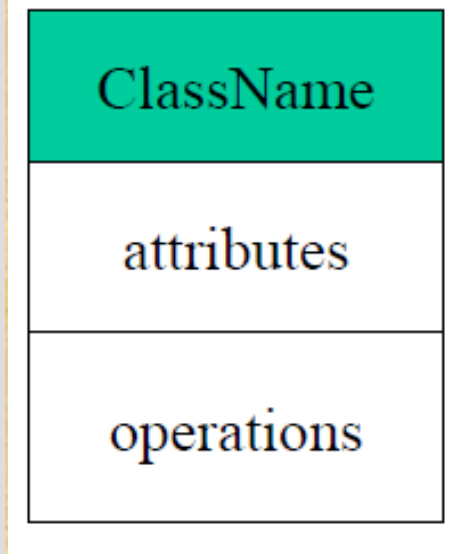


Essential elements Class Diagram



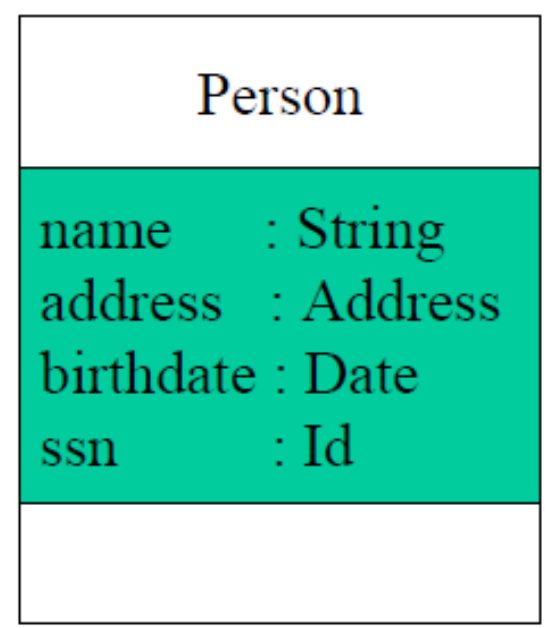
- Class Name
- Attributes
- Operations

Class Names



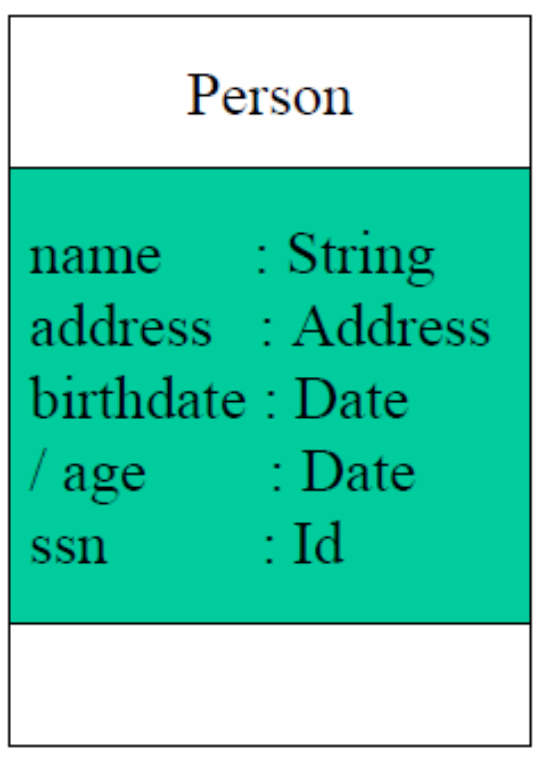
- The name of the class is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.

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)



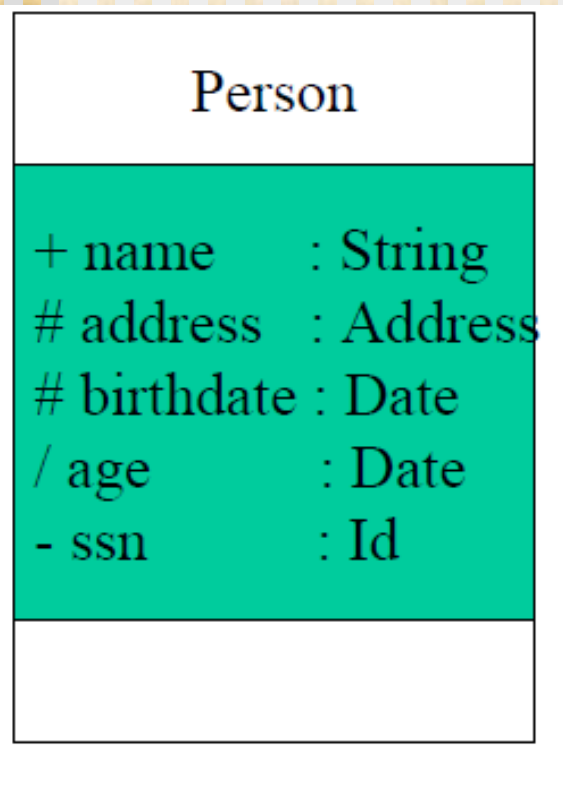
- Attributes are usually listed in the form:

  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:

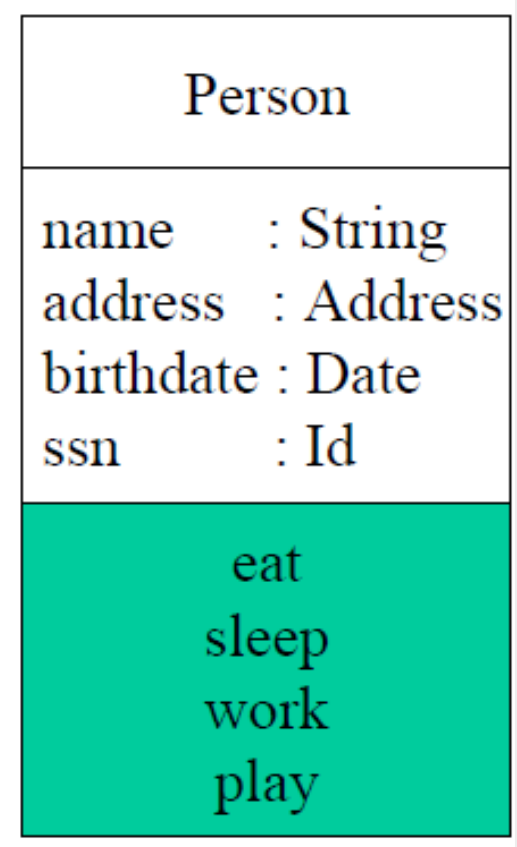
  / age : Date

Class Attributes (Cont'd)



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

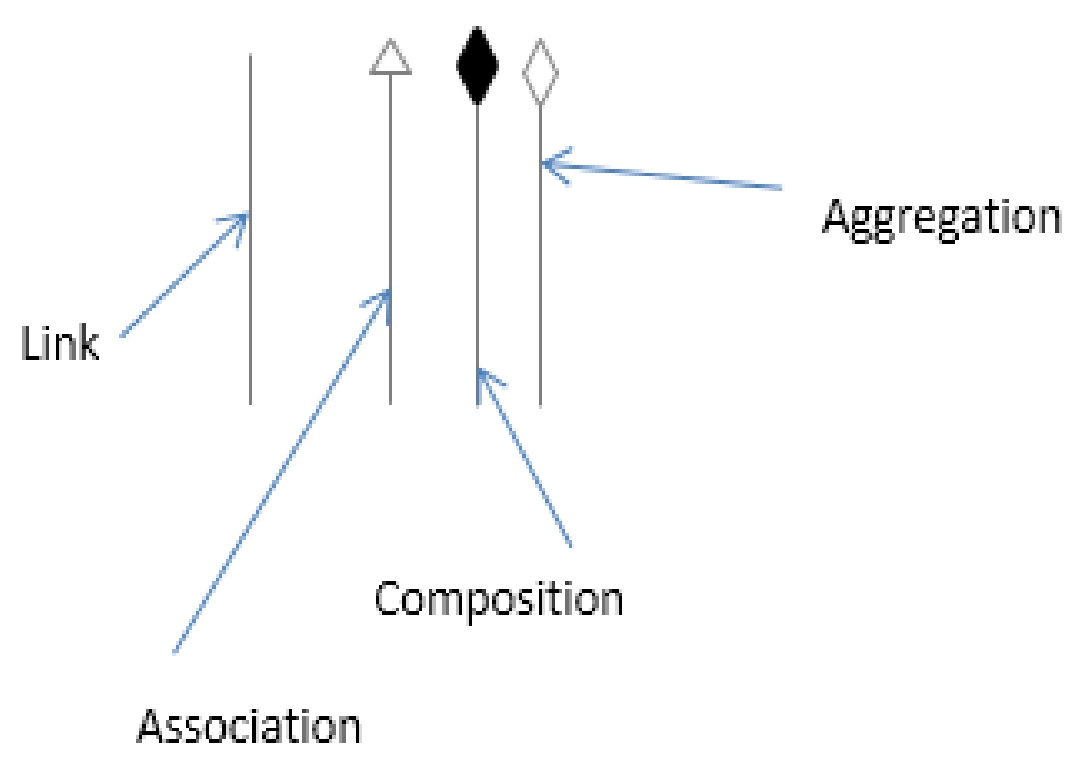
Class Operations



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

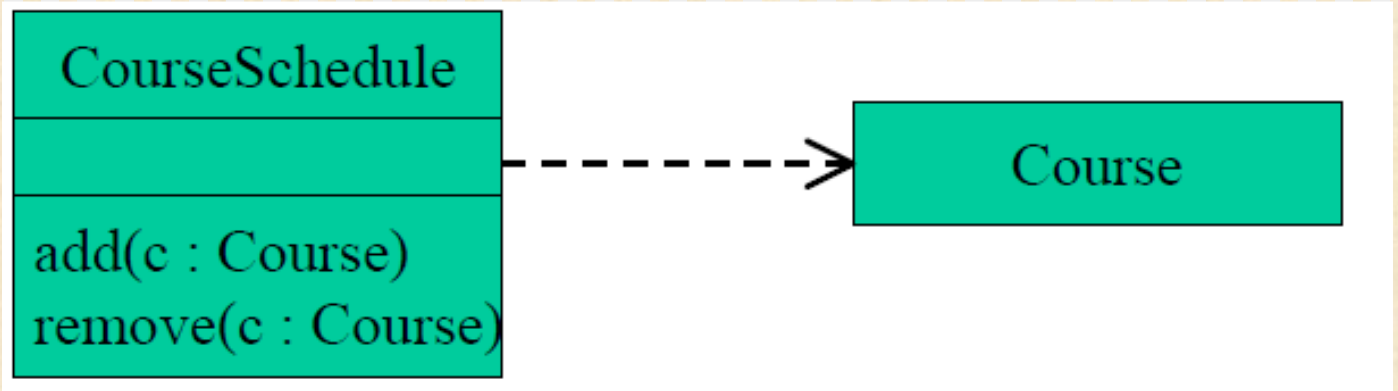
Relationships

- In UML, object interconnections (logical or physical), are modeled as relationships.
- There are three kinds of relationships in UML:
  - Dependencies
  - Generalizations
  - Associations

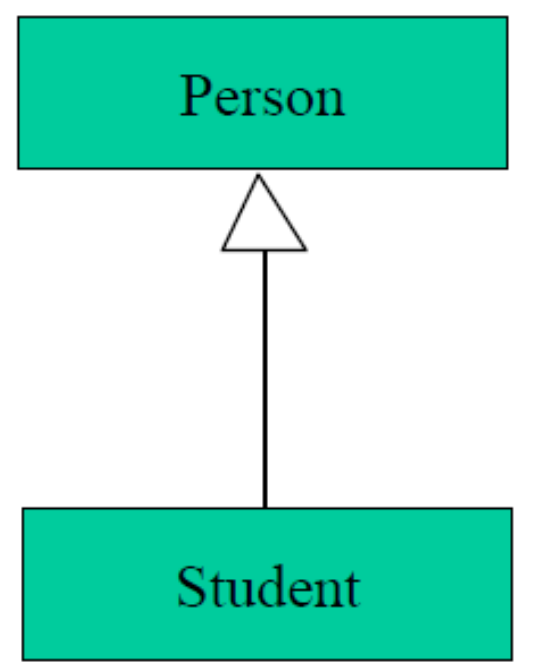


Dependency Relationships

- A *dependency* indicates a semantic/notational relationship between two or more elements.
- CourseSchedule has dependency on Course



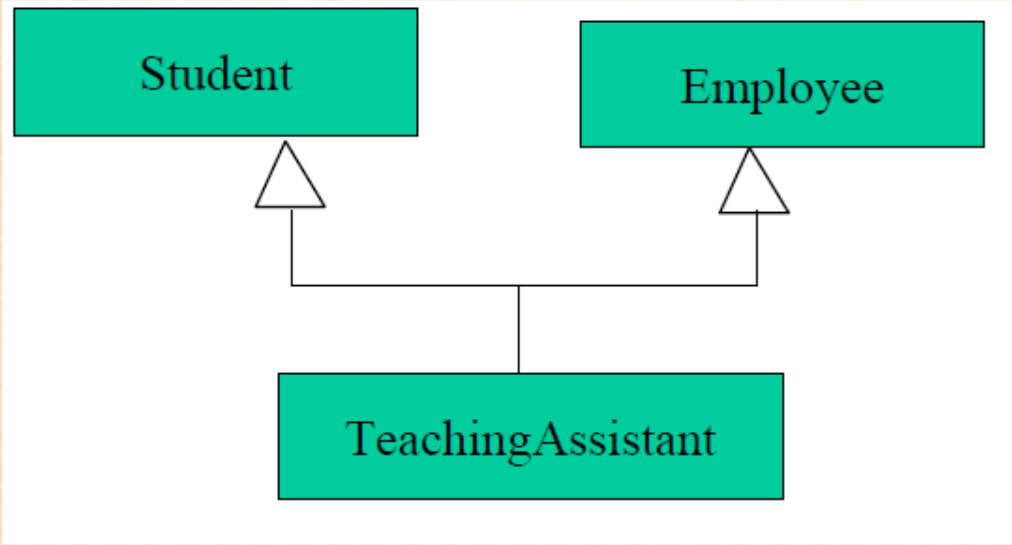
Generalization Relationships



- 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.

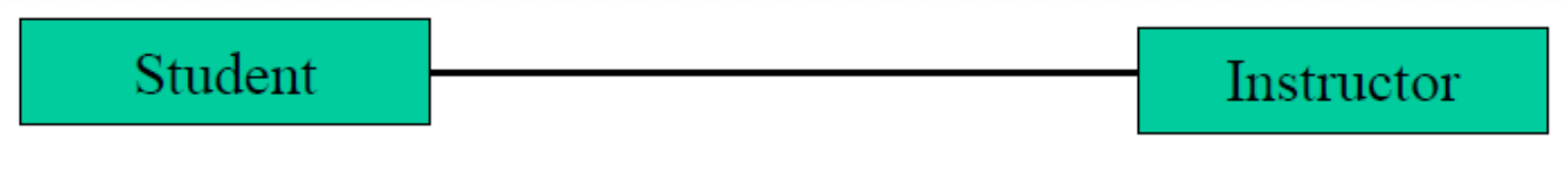
Generalization Relationships (Cont'd)

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



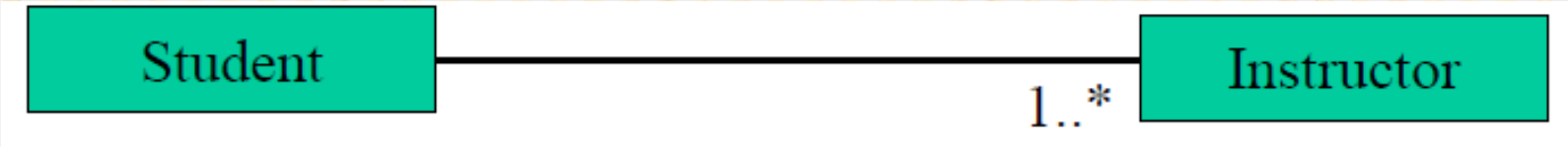
Association Relationships

- 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*****:**



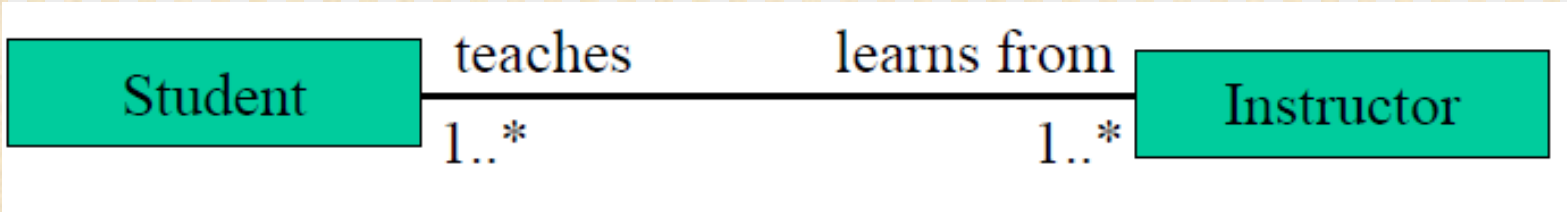
Association Relationships (Cont'd)

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

Association Relationships (Cont'd)

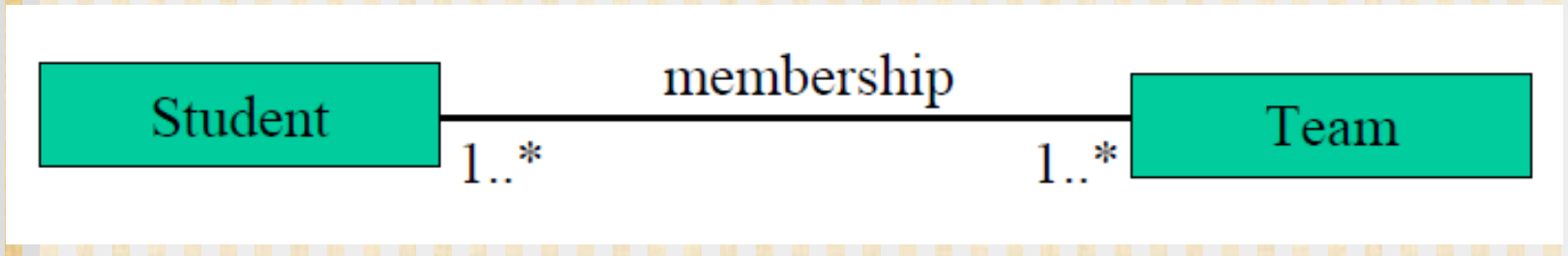
- We can also indicate the behavior of an object in an association

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



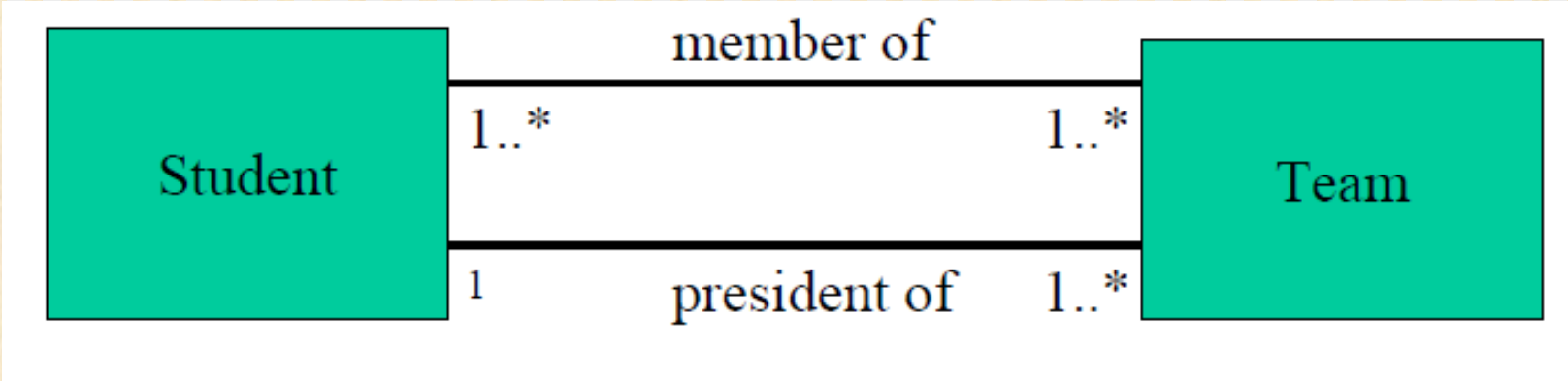
Association Relationships (Cont'd)

- We can also name the association.



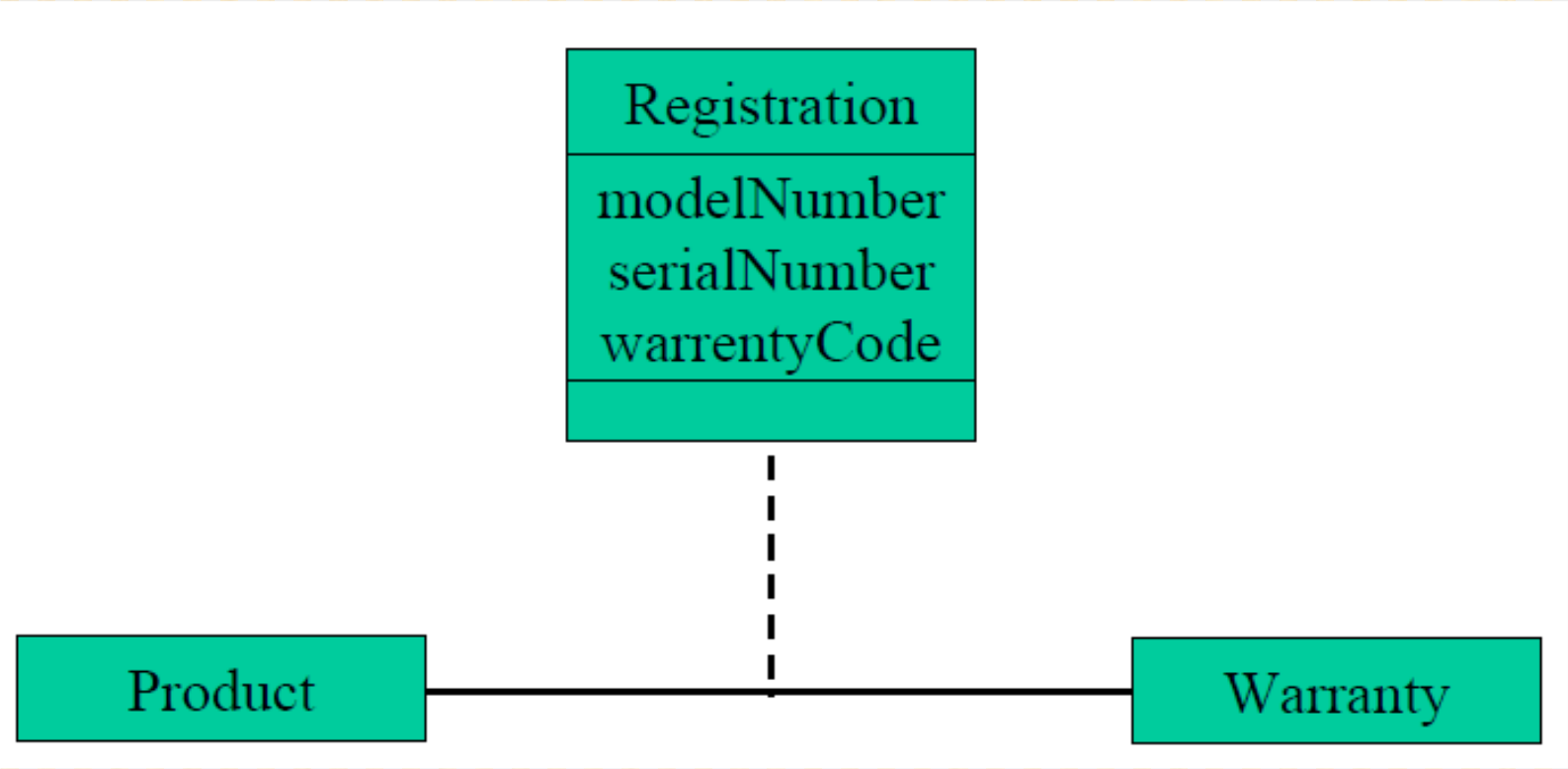
Association Relationships (Cont'd)

- We can specify dual associations.



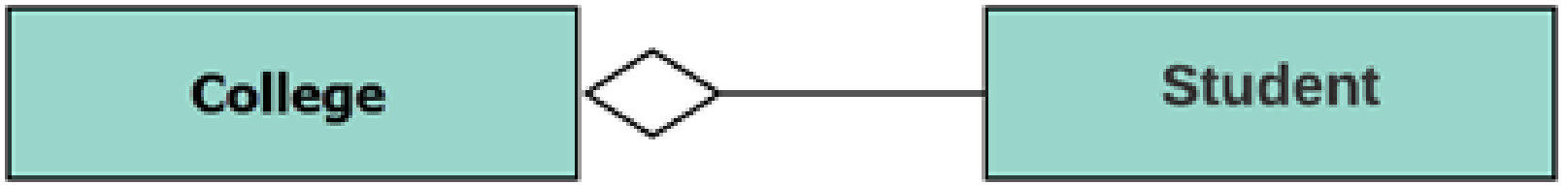
Association Relationships (Cont'd)

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



Relationships (Aggregation)

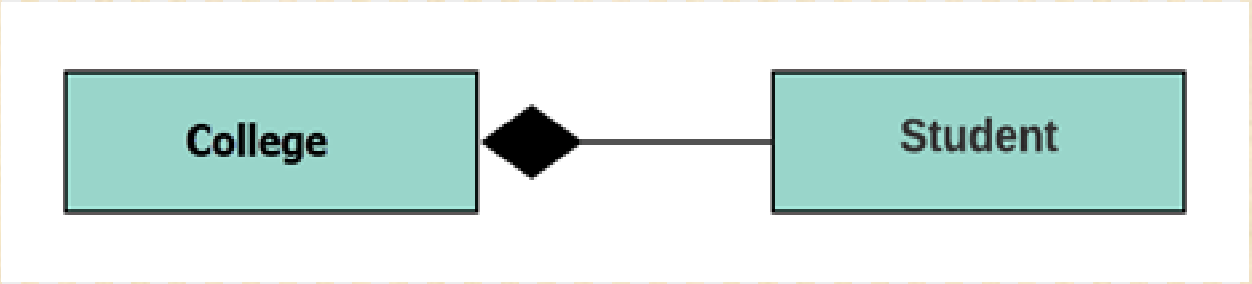
- Aggregation is a special type of association that models a whole- part relationship between aggregate and its parts.



- For example, the class college is made up of one or more student. In aggregation, the contained classes are never totally dependent on the lifecycle of the container.
- Here, the college class will remain even if the student is not available.

Relationships (Composition)

- Compositions are denoted by a filled-diamond adornment on the association.



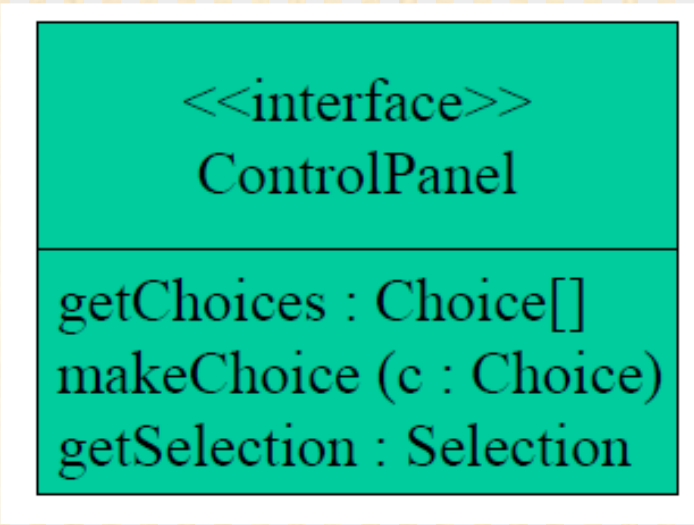
- The composition is a special type of aggregation which denotes strong ownership between two classes when one class is a part of another class.
- For example, if college is composed of classes student. The college could contain many students, while each student belongs to only one college. So, if college is not functioning all the students also removed.

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.

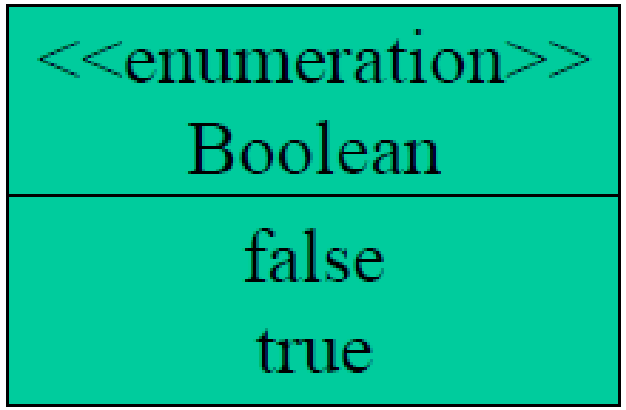
Interface Services

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

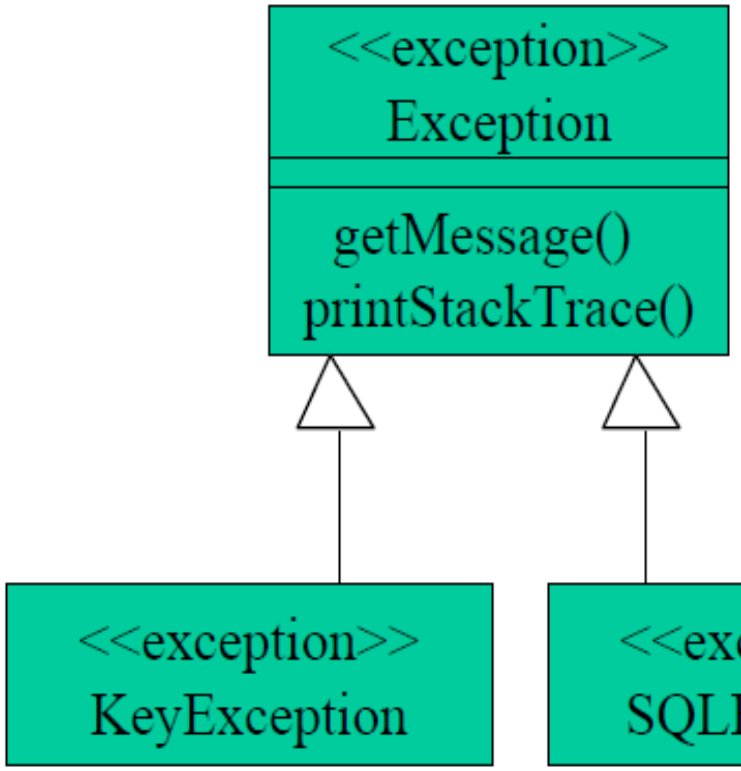


Enumeration

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



Exceptions

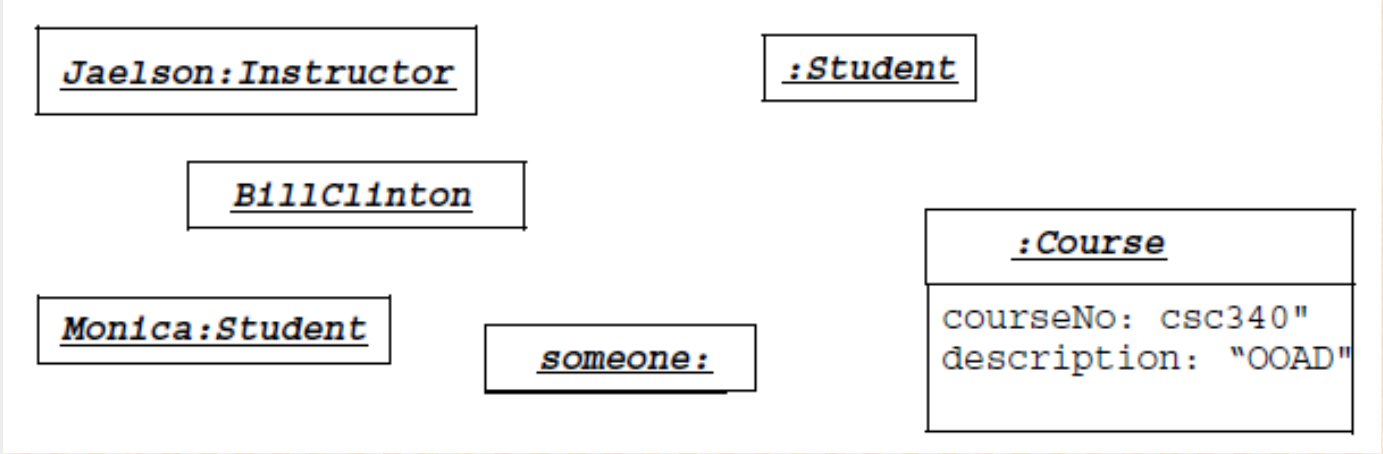


Exceptions can be modeled just like any other class.

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

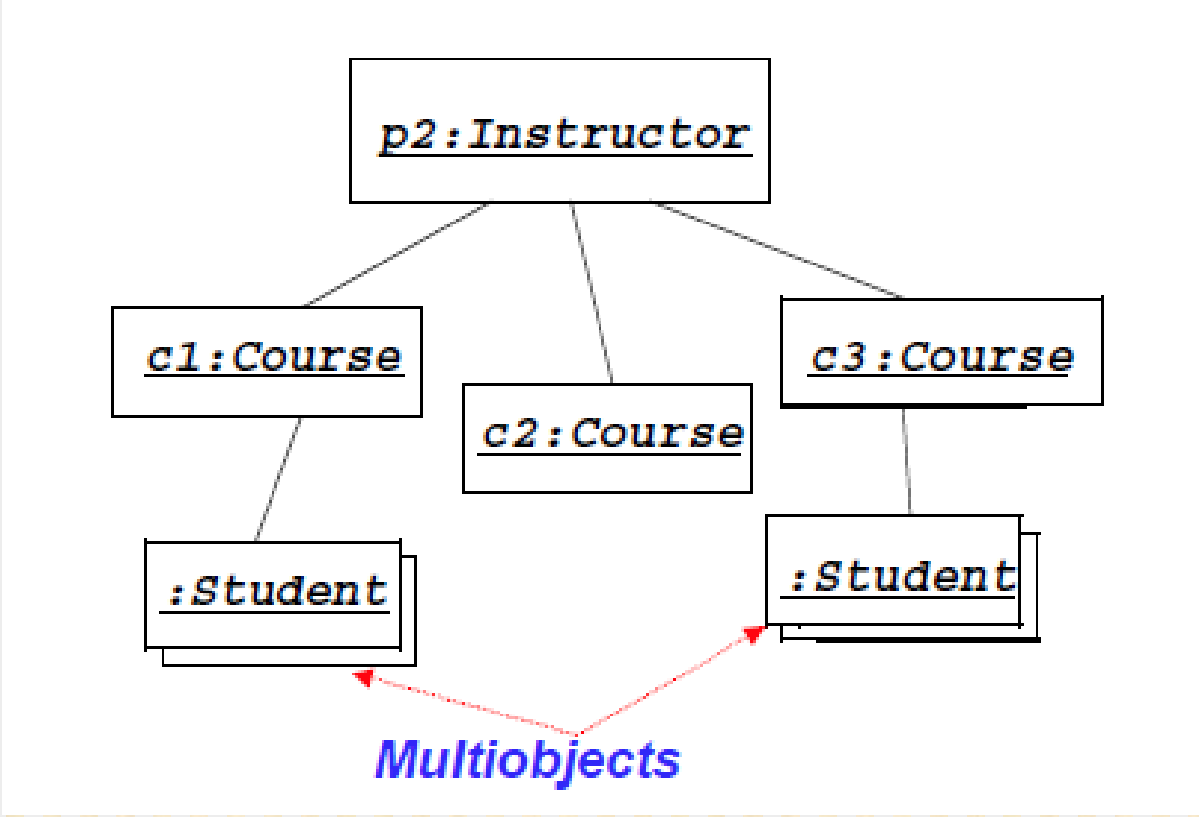
Object Diagrams

- *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.*

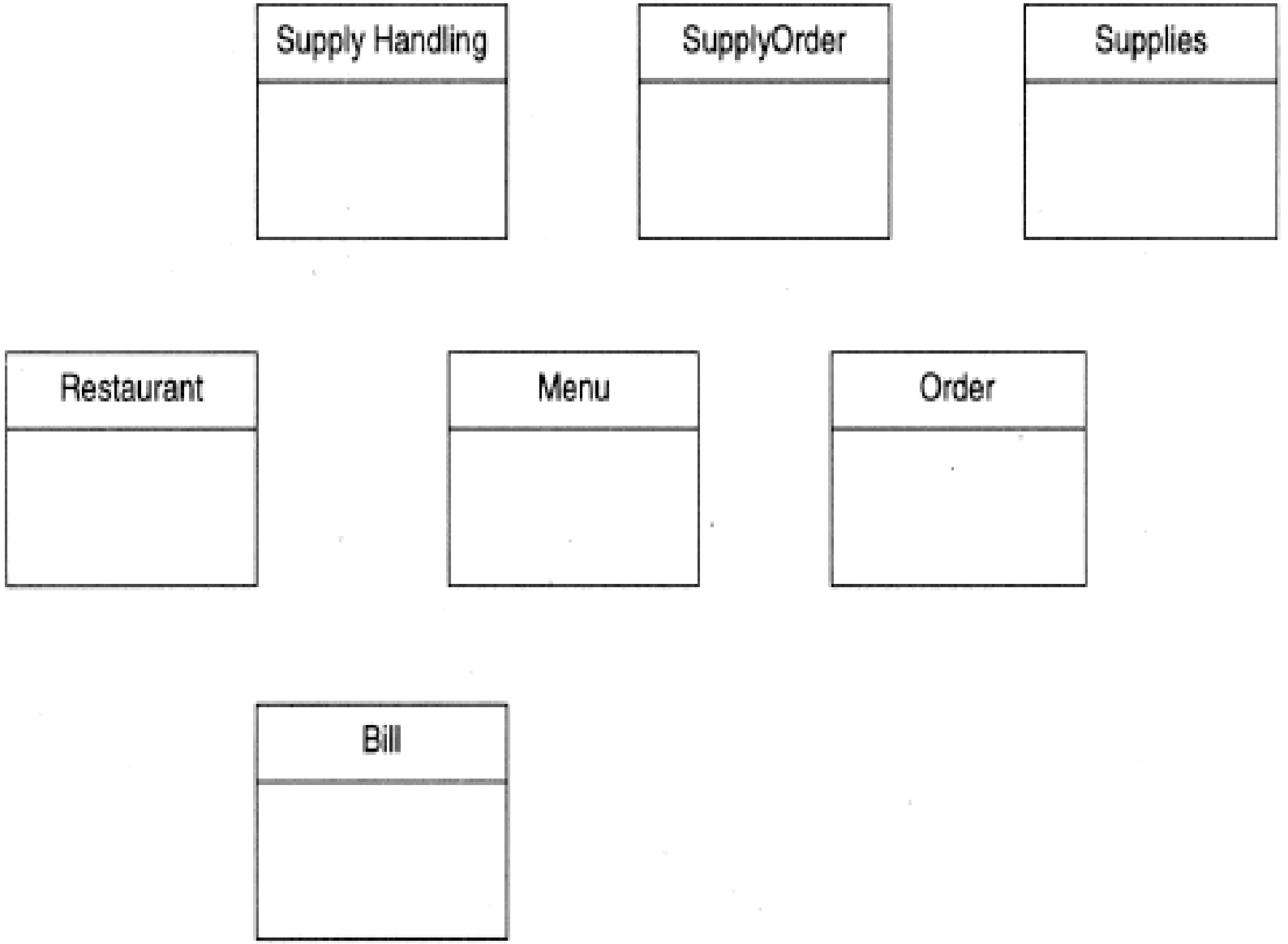


Multi objects

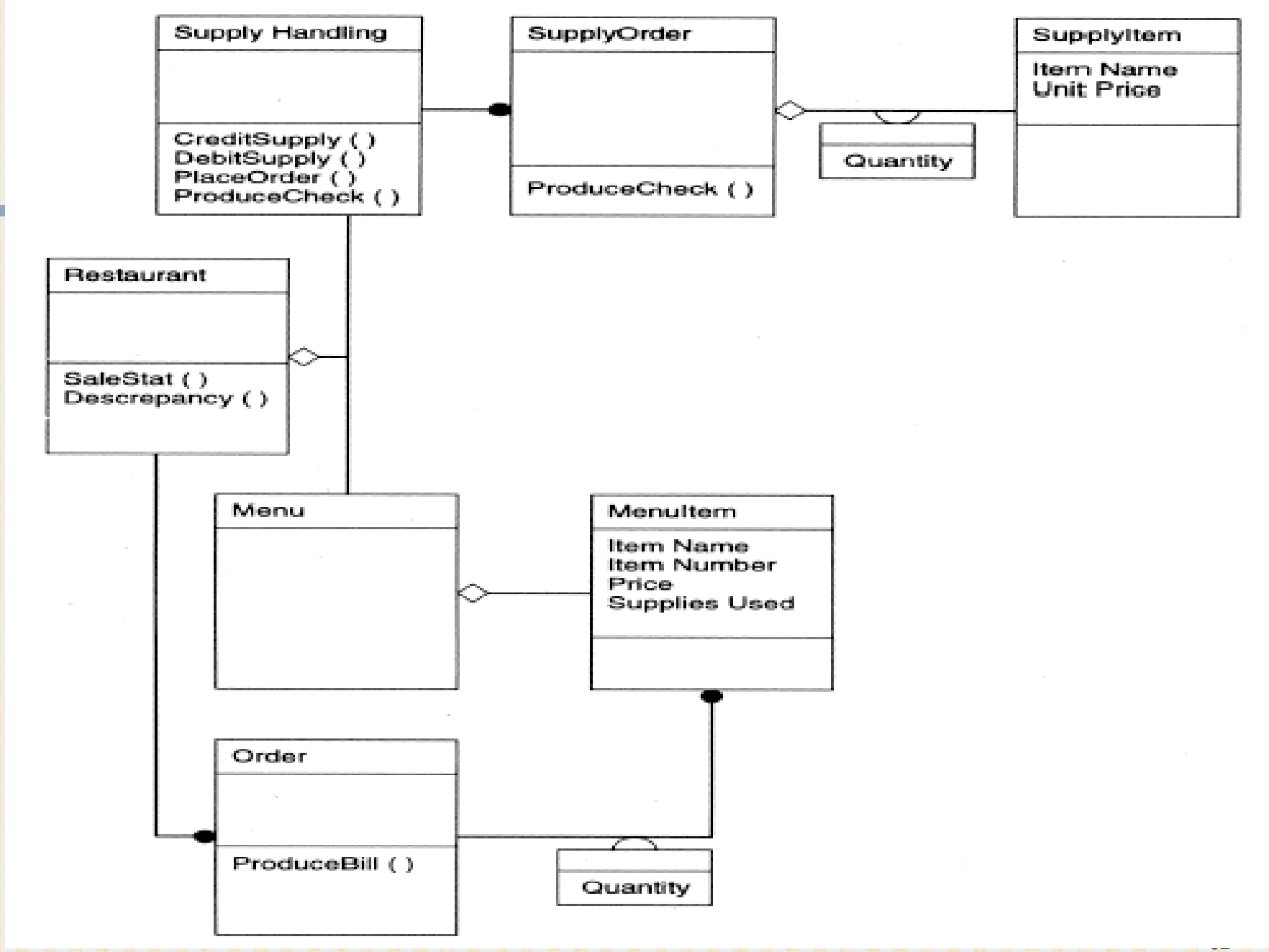
- *A **multi object** is a set of objects, with an undefined number of elements*



Restaurant example: Initial classes

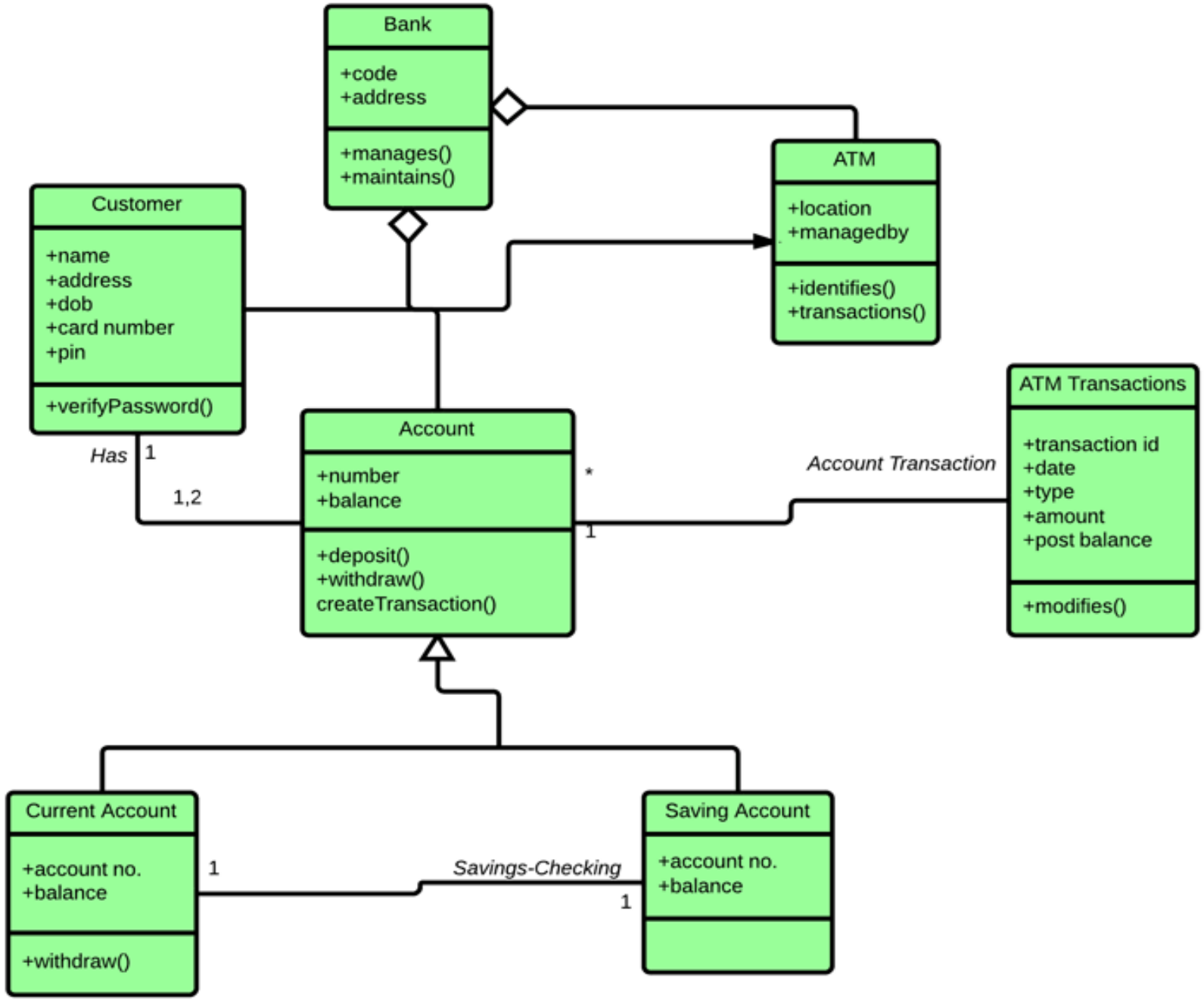


Restaurant example: Initial classes

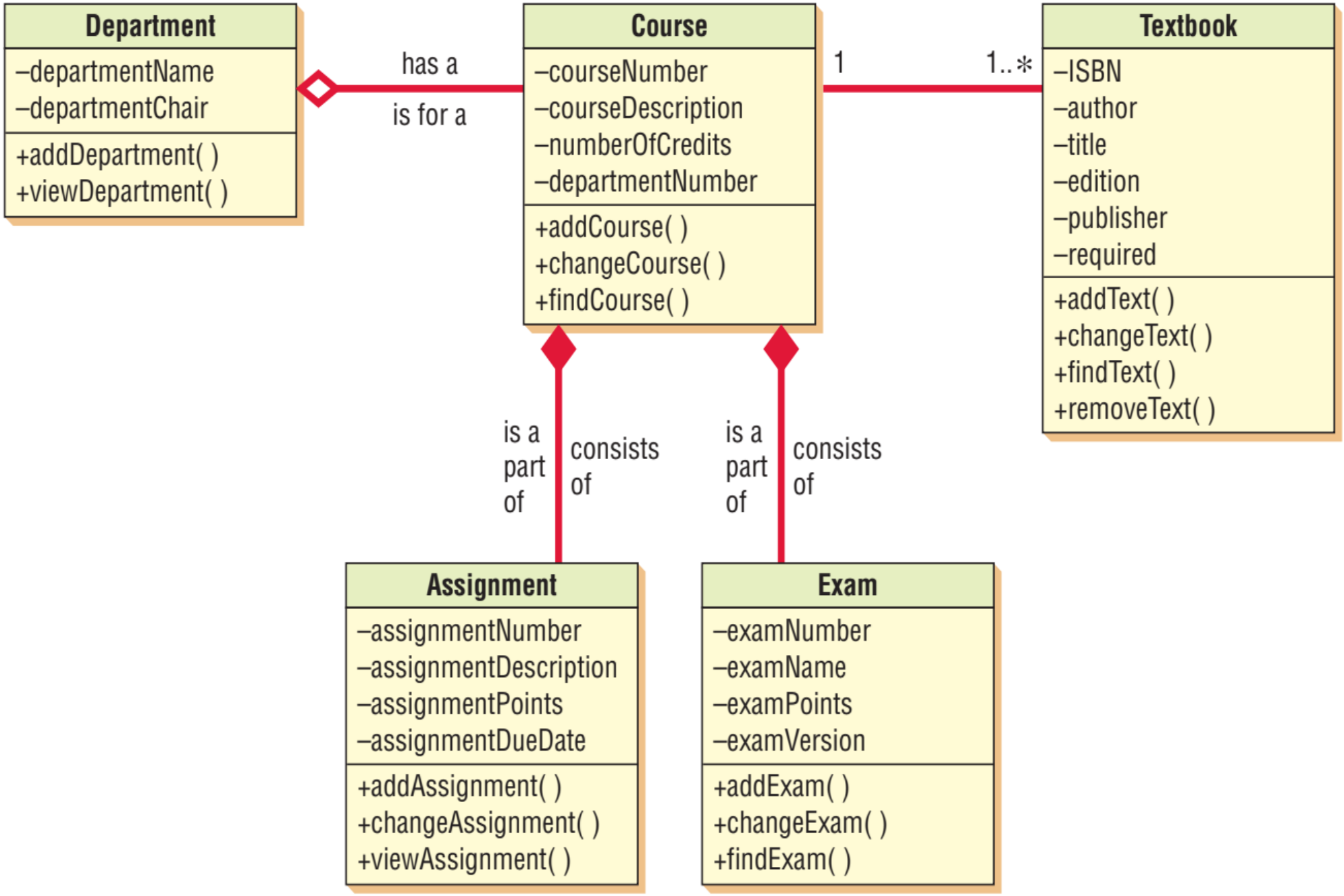


Example of UML Class Diagram

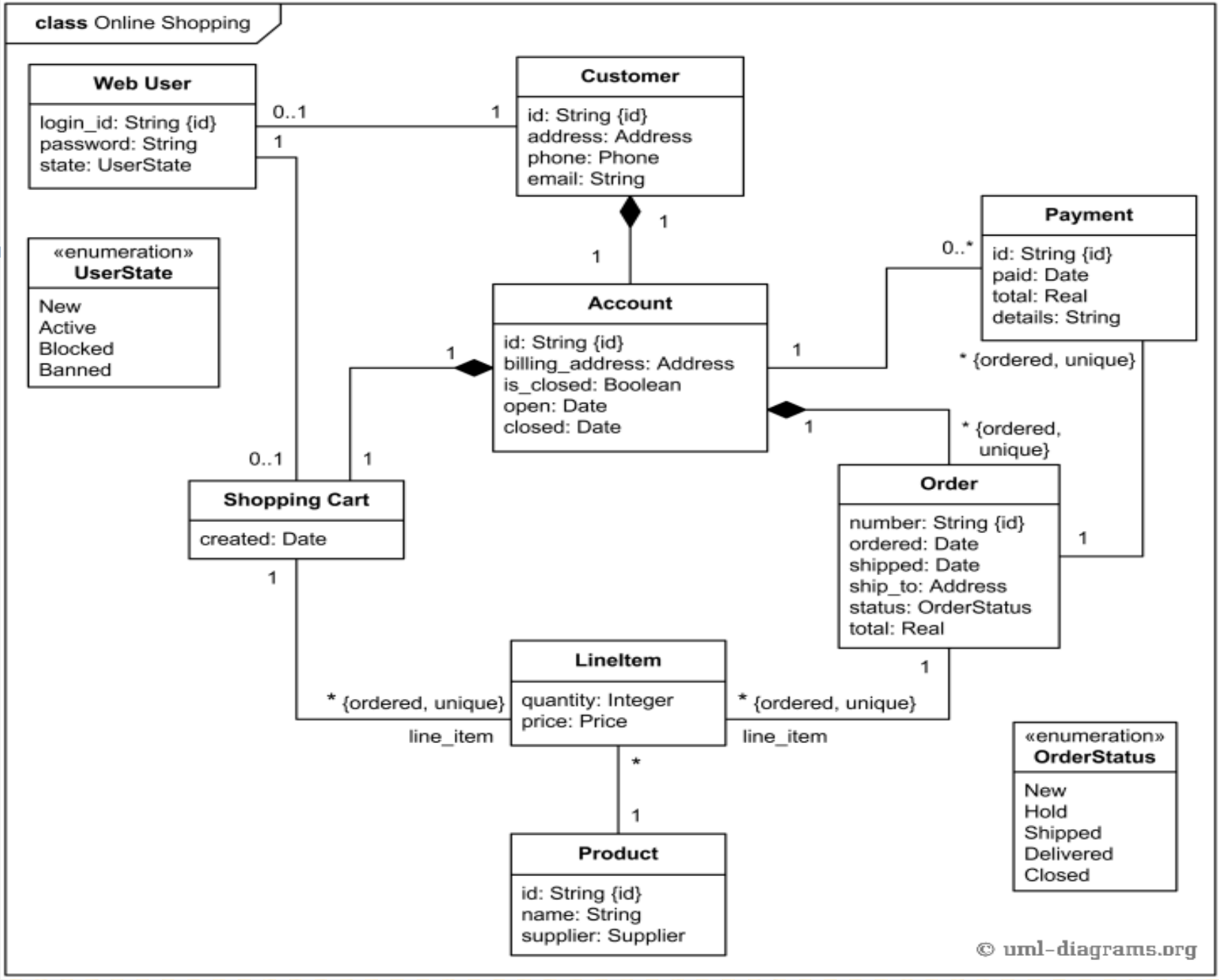
ATMs system is very simple as customers need to press some buttons to receive cash. However, there are multiple security layers that any ATM system needs to pass. This helps to prevent fraud and provide cash or need details to banking customers



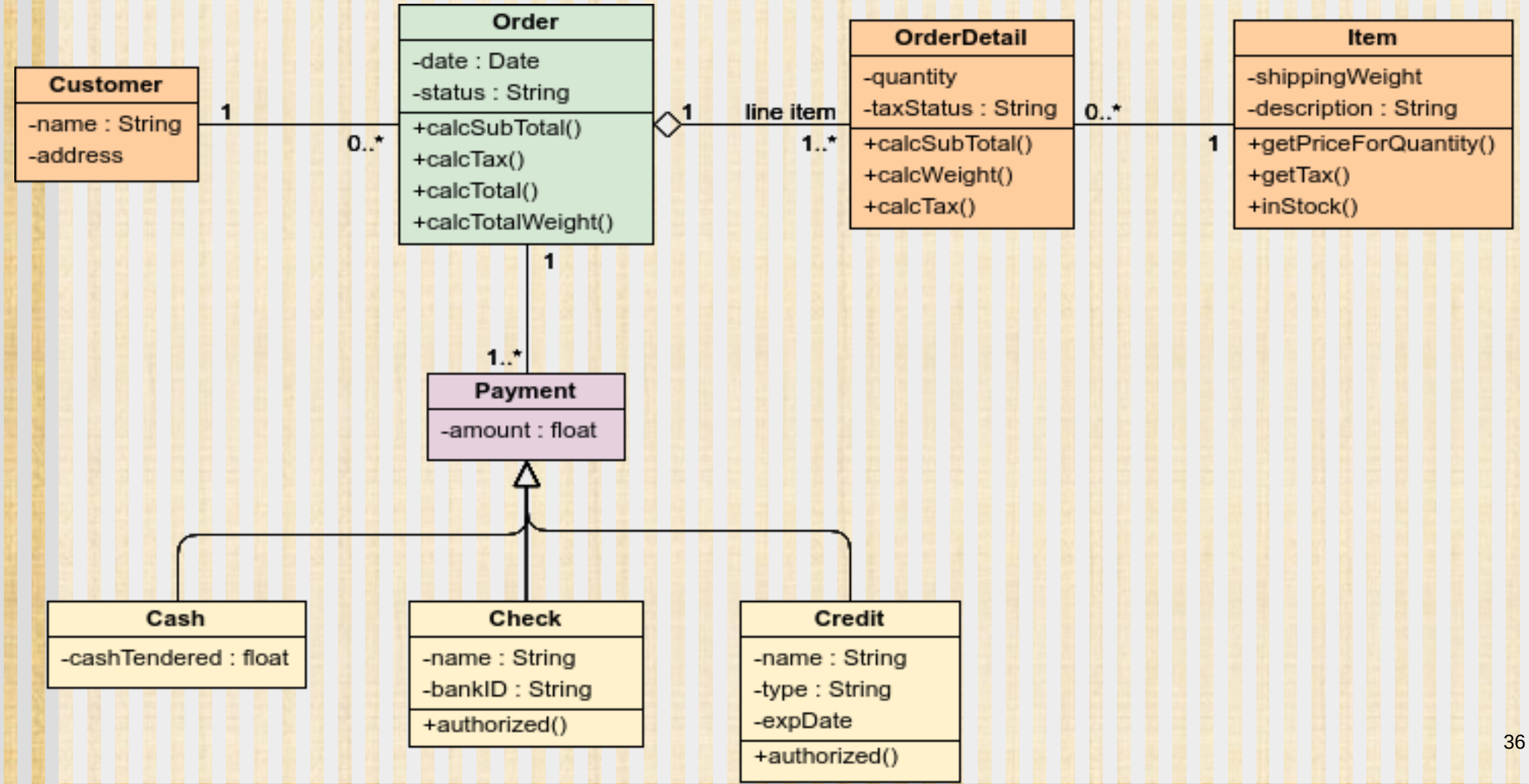
A class diagram for course offerings



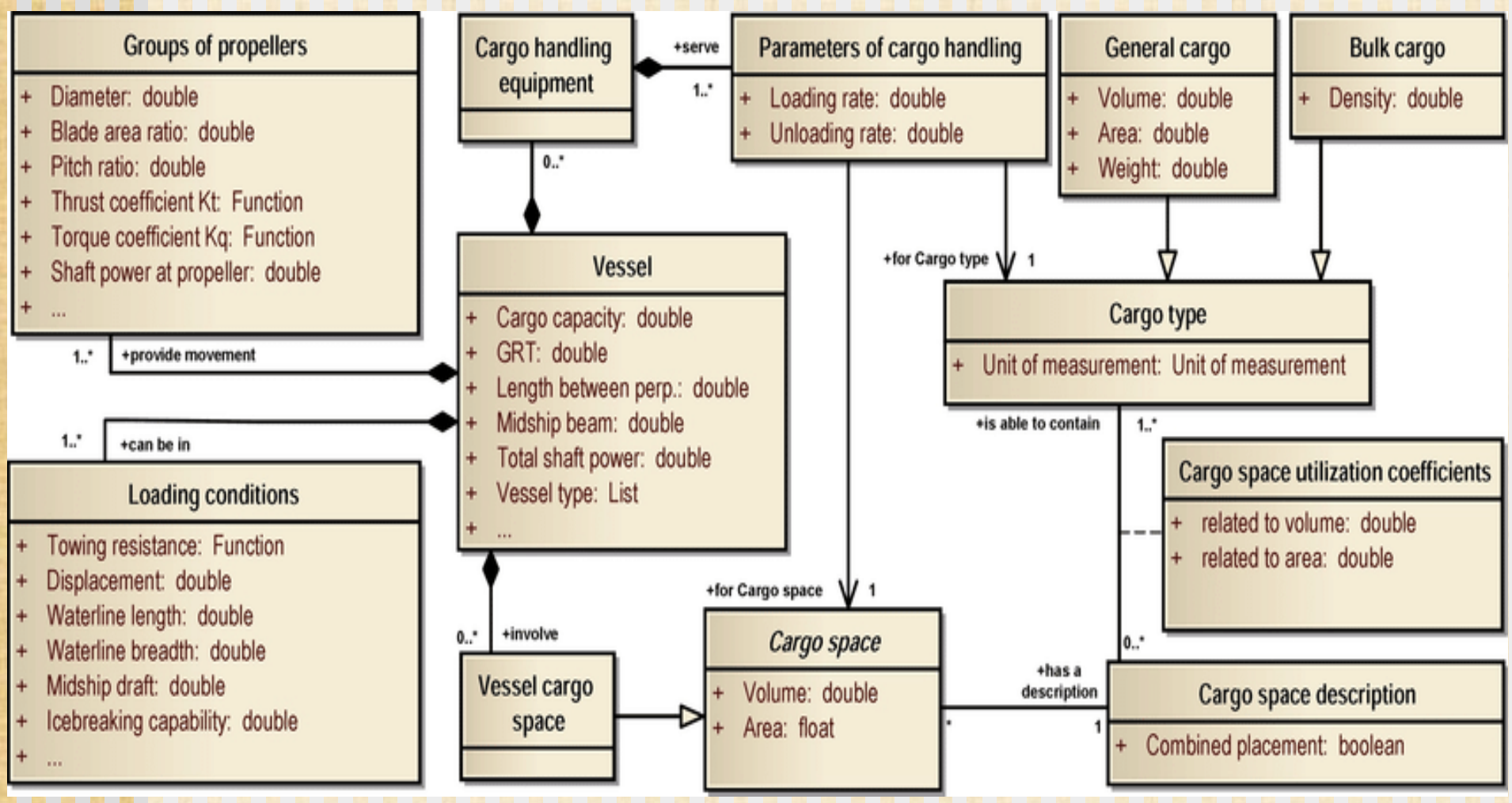
Class Diagram: Online Shopping



Class Diagram: Order Process



Ship & Cargo Object Model



References

1. **Software Engineering A practitioner's Approach** by Roger S. Pressman, 7th edition, McGraw Hill, 2010.
2. **Software Engineering by Ian Sommerville,** 9th edition, Addison-Wesley, 2011
3. **Systems Analysis and Design,** Kendall and Kendall, Fifth Edition