Class Diagrams

Topic # 8

Chapter 16 – Craig Larman

Class Diagrams

- It provides a conceptual model of the system
- **Class diagram**: A graphical depiction of a system's static object structure, showing object classes that the system is composed of as well as the relationships between those object classes.
- The purpose of class diagram is to model the static view of an application.
- Class diagrams are the only diagrams which can be directly mapped with object-oriented languages.
- Widely used at the time of construction.

Basics of UML Class Diagrams

- A software application is comprised of classes and a diagram depicting the relationship between each of these classes would be the class diagram.
- A class diagram is a pictorial representation of the detailed system design

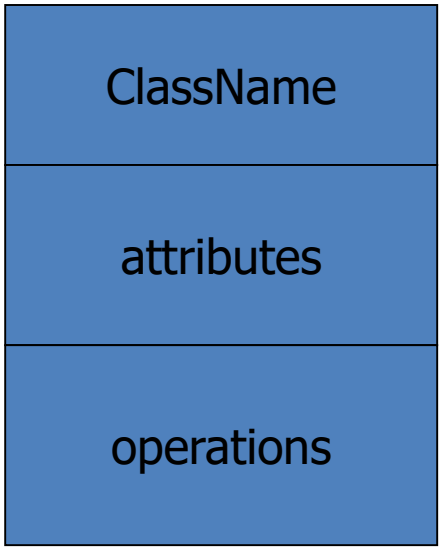
Relationship between Class Diagram and Use Cases

- How does a class diagram relate to the use case diagrams that that we learned before?
- When you designed the use cases, you must have realized that the use cases talk about "what are the requirements" of a system.
- The aim of designing classes is to convert this "what" to a "how" for each requirement
- Each use case is further analyzed and broken up that form the basis for the classes that need to be designed

Elements of a Class Diagram

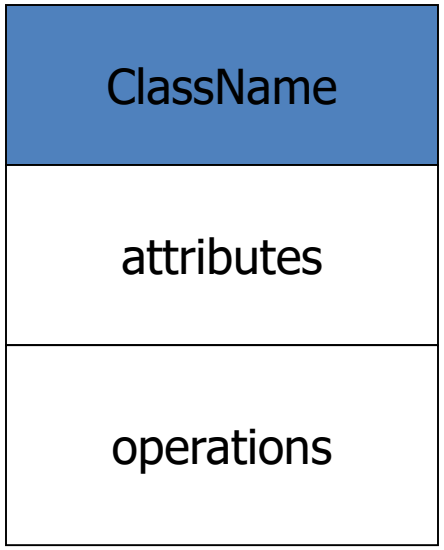
- A class diagram is composed primarily of the following elements that represent the system's business entities:
  - Class
  - Class Relationships

Classes



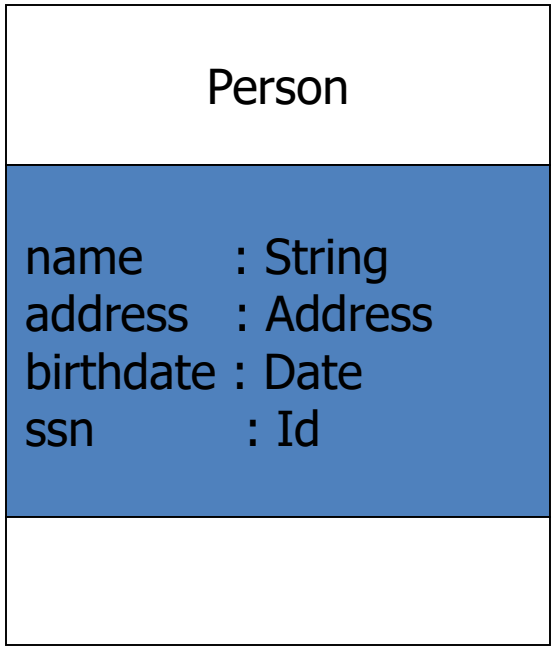
- A class represents an entity of a given system that provides an encapsulated implementation of certain functionality of a given entity. These are exposed by the class to other classes as methods. Apart from functionality, a class also has properties that reflect unique features of a class. The properties of a class are called attributes.
- 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

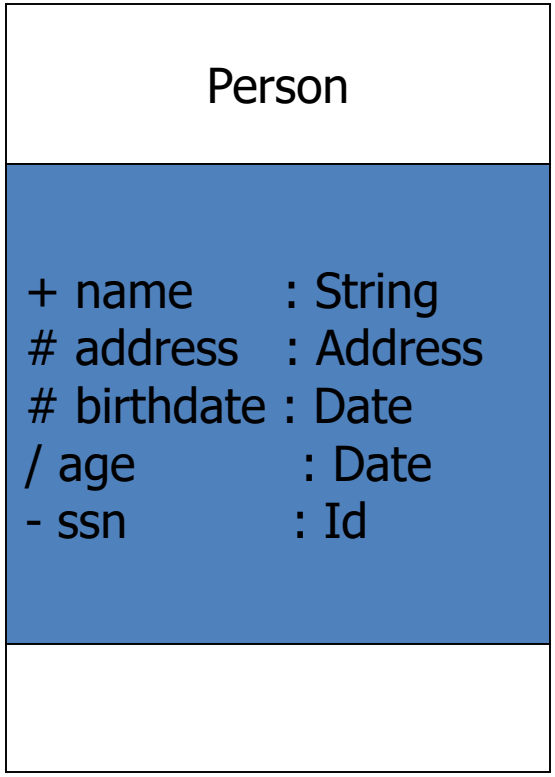


- An *attribute* is a named property of a class that describes the object being modeled.
- In the class diagram, attributes appear in the second compartment just below the name-compartment.

- Attributes are usually listed in the form:

attributeName : Type

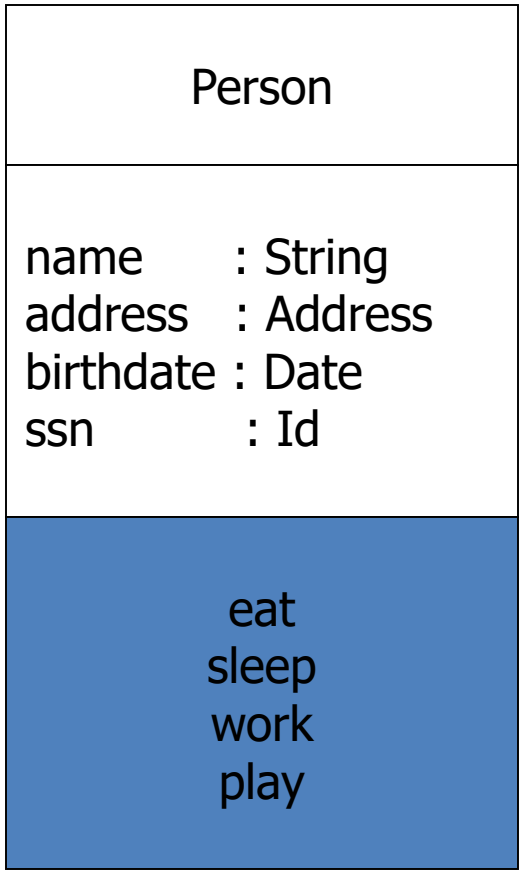
Class Attributes-Visibility(Access Specifiers)



Attributes can be:

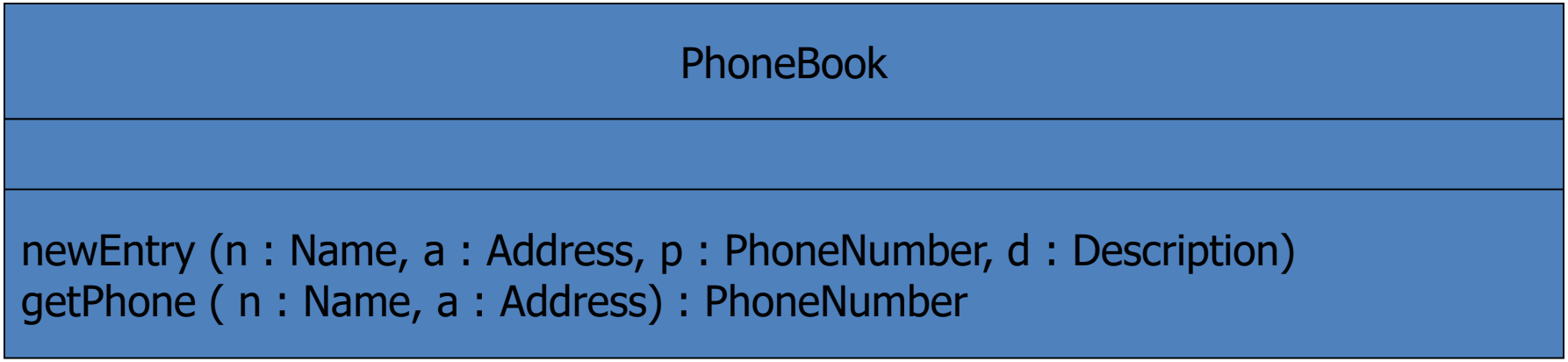
+ public
# protected
- private
~ package
/ derived

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.

The full UML syntax for attribute list is

name : attribute type

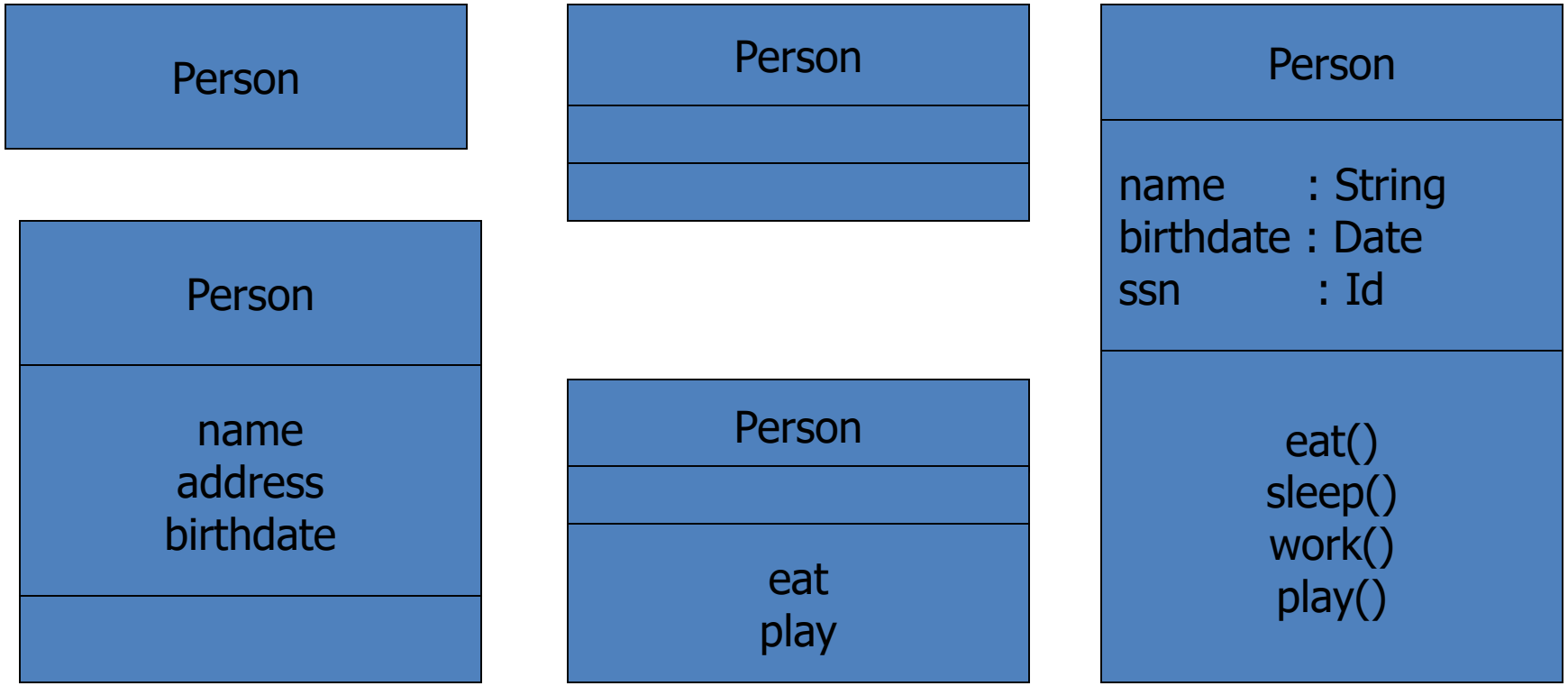
flightNumber : Integer

The full UML syntax for operations is

visibility name (parameter-list) : return-type

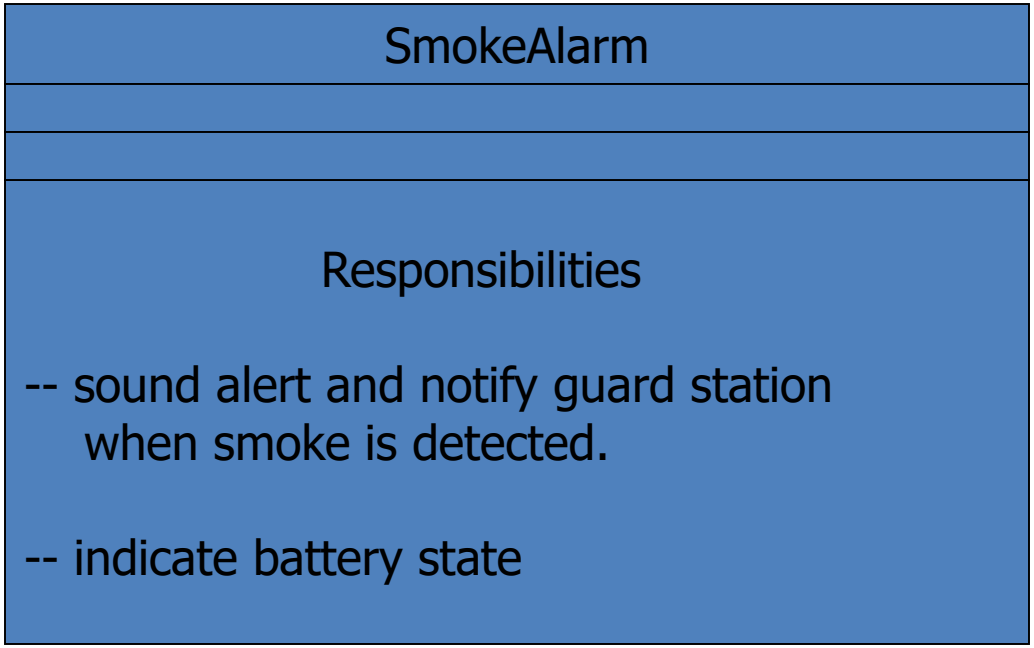
Depicting Classes

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

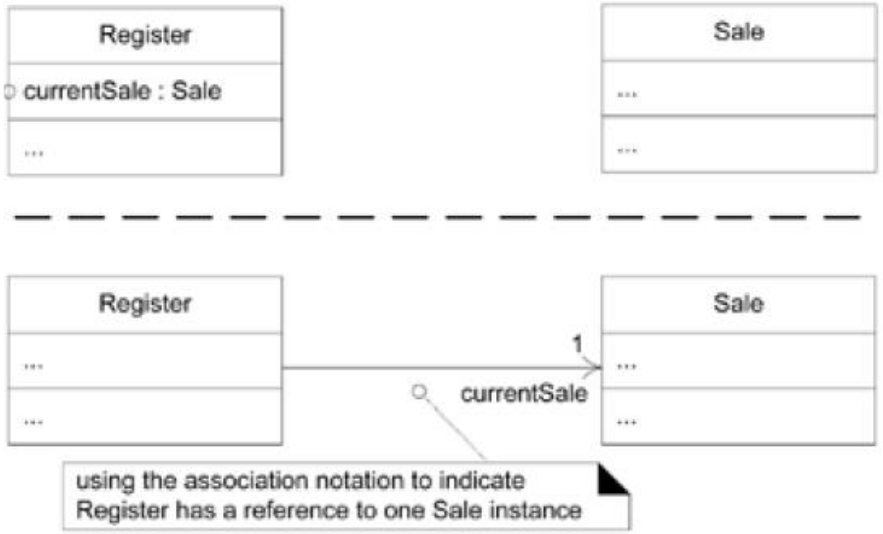


Class Responsibilities

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.



Attribute notations

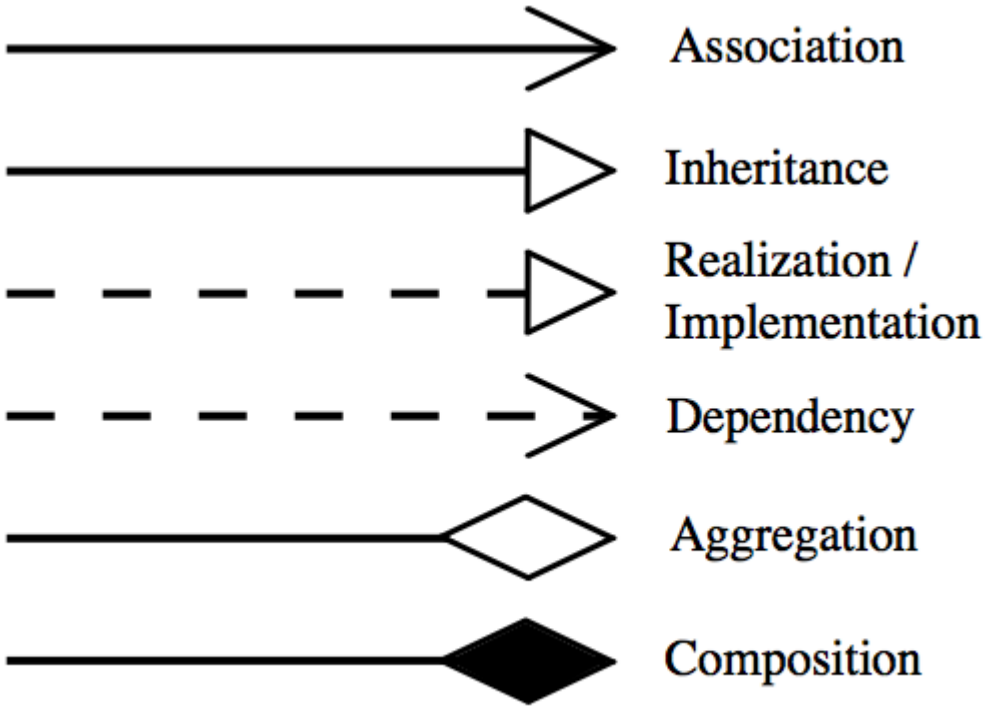



```
public class Register
{
private int id;
private Sale currentSale;
private Store location;
// …
}
```

Relationships

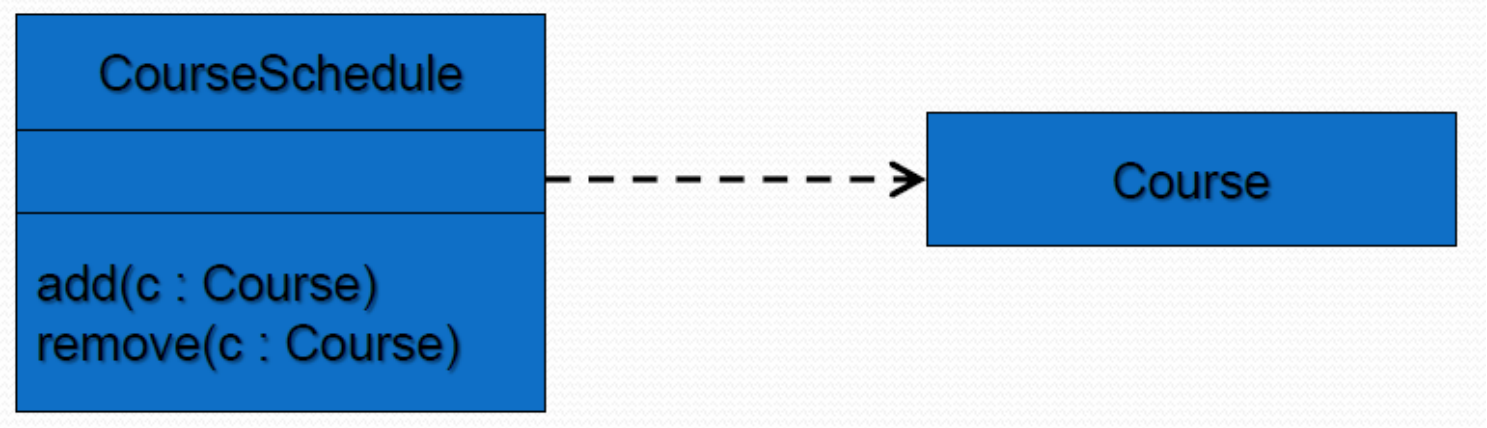
In UML, object interconnections (logical or physical), are modeled as relationships.

There are six kinds of relationships in UML:



Dependency Relationships

- A dependency indicates a semantic relationship between two or more elements.
- In the following example: there is a dependency from Course Schedule to Course as Course is used in add and remove operations of Course Schedule.

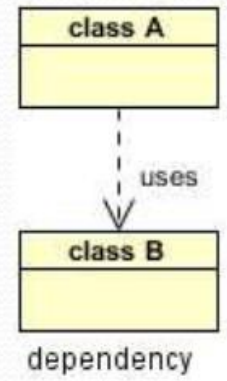


Dependency Relationship

- **Dependency** is represented when a reference to one class is passed in as a method parameter to another class. For example, an instance of class B is passed into a method of class A:
- Class A uses class B
- Dependency is a relationship between two things in which change in one element also affects the other.

Import class B

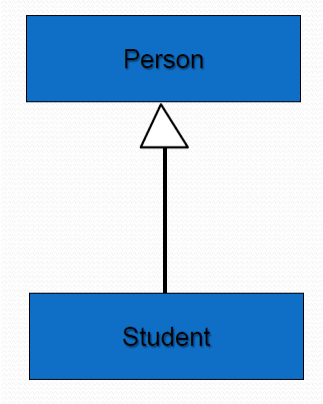
```
1public class A {
2
3    public void doSomething(B b) {
```



Generalization Relationship

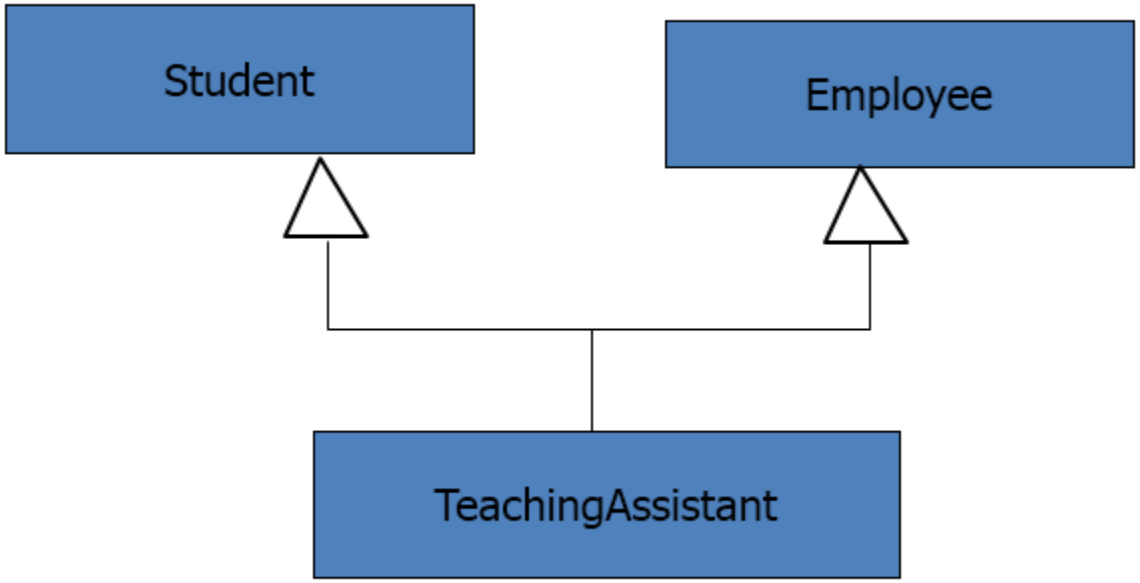
- A generalization connects a subclass to its superclass.
- It denotes an inheritance of attributes & behavior from superclass to the subclass and indicates a specialization in the subclass of the more general superclass.

```
public class Person {
...
} // class Person
public class Student extends Person {
....
} // class Student
```



Generalization

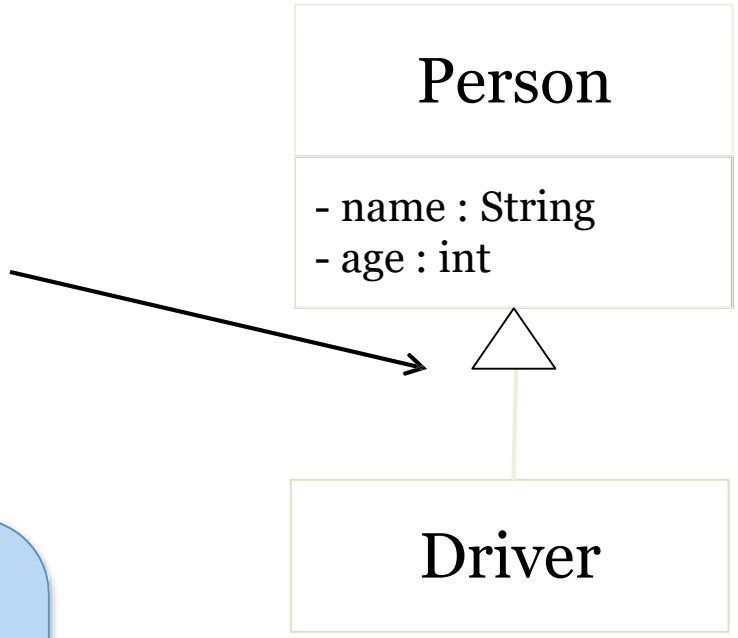
- UML permits a class to inherit from multiple super-classes, although some programming languages (e.g. Java)don't permit multiple inheritance.



UML Class Diagrams: Generalization

Drivers are a type of person. Every person has a name and an age.

Note: we use a special kind of arrowhead to represent generalization



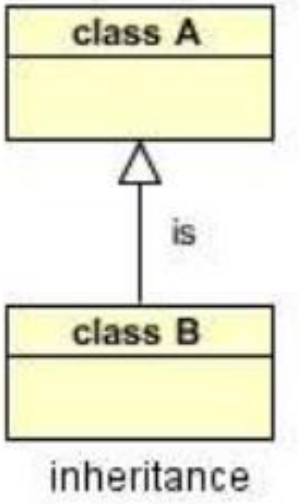

```
public Person {
...
} // class Person
public class Driver extends Person{
....
} // class Driver
```

We assume that Driver **inherits** all the properties and operations of a Person (as well as defining its own)

Inheritance: Generalization

- Class B is a Class A (or class A is extended by class B)

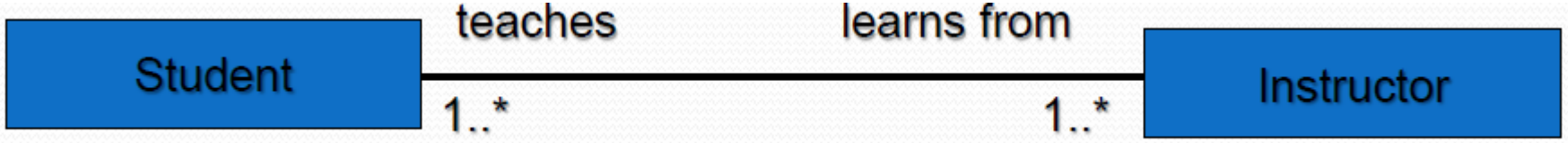
```
1public class A {
2
3    ...
4
5} // class A
6
7public class B extends A {
8
9    ....
10
11} // class B
```



Association Relationship

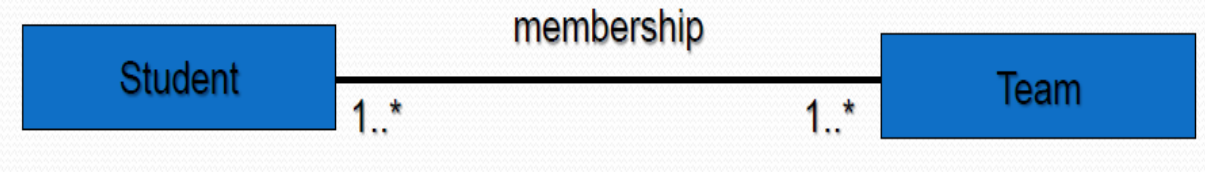
- If two classes in a model need to communicate with each other, there must be a link between them.
- An association denotes the link.
- Usually an object provides services to several other objects hence an object keeps associations with other objects to delegate tasks
- The **multiplicity** of an association can be indicated by adding multiplicity adornments to the line denoting the association.
- Example: A student has one or more instructors

- The behavior of an object can also be indicated in an association (i.e. the

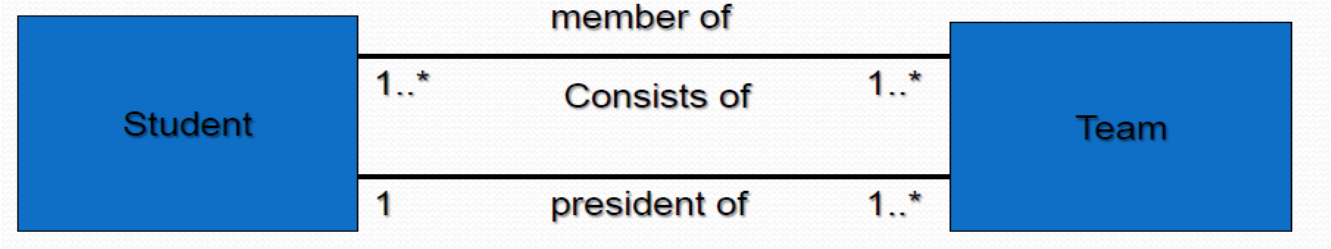


Association Relationship

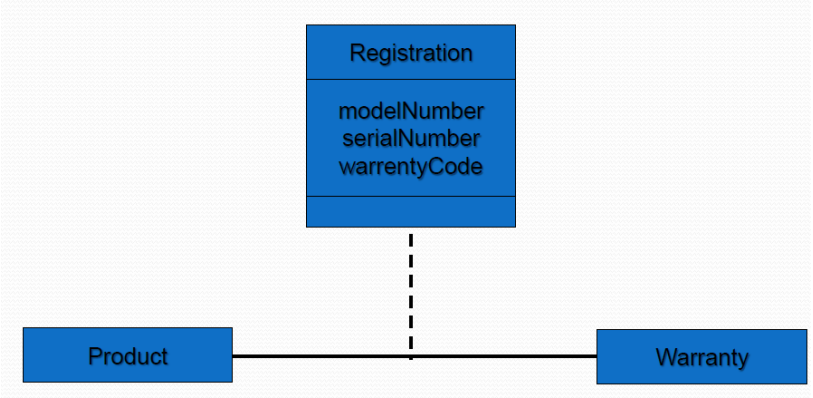
- The association can also be named.



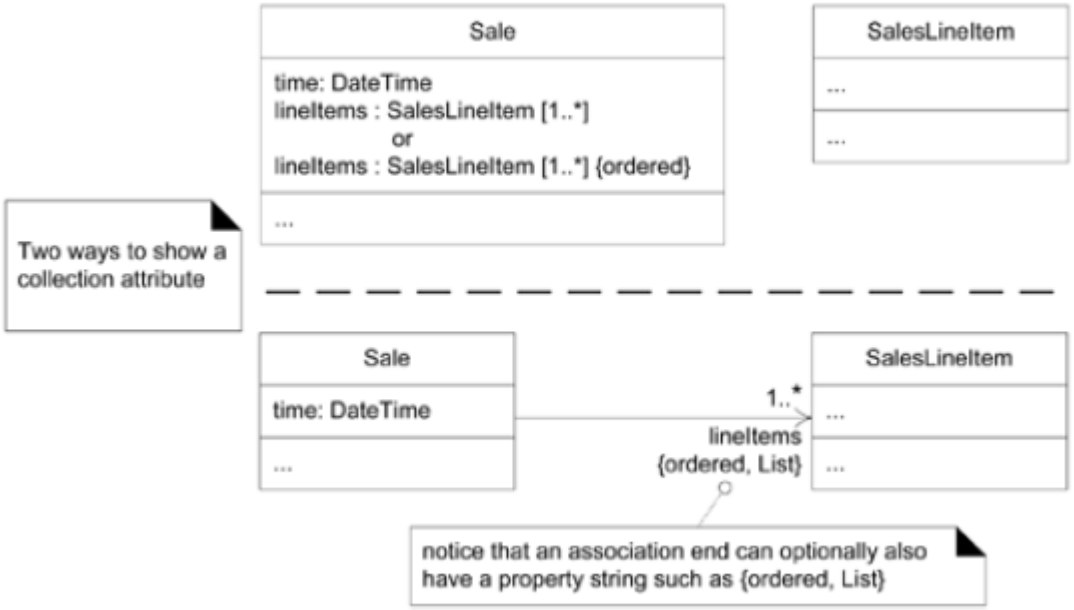
- Dual association can also be specified.



- Association can also be object themselves, called link classes or association classes.
- * semester, course & course offered



Collection attribute




```
public class Sale
{
private List<SalesLineItem> lineItems =
                            new ArrayList<SalesLineItem>();

// …
}
```

UML Note symbol

- Note symbols can be used in any UML diagram.
- **Note symbols** is displayed as dog eared rectangle with a dashed line to the explained element.

Multiplicity

- Multiplicity (how many are used)
- * ⇒ 0, 1, or more
- 1 ⇒ 1 exactly
- 2..4 ⇒ between 2 and 4,
- inclusive 3..* ⇒ 3 or more

Kinds of Association

- Object Association
  - Simple Association: Is simply called as “association”
  - Composition
  - Aggregation

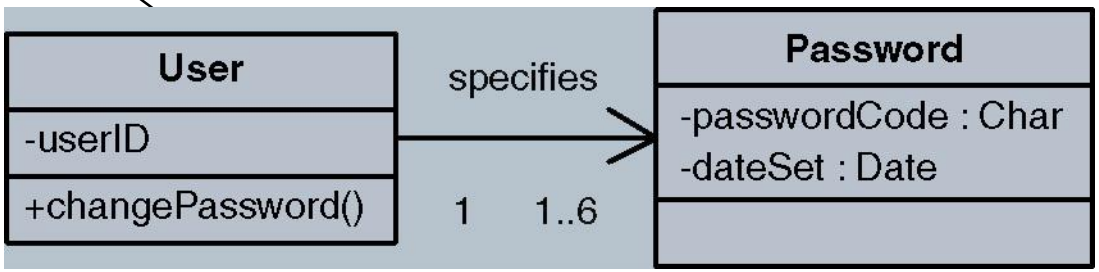
Kinds of Simple Association

- w.r.t navigation
  - One-way Association
  - Two-way Association
  - Self association
- w.r.t number of objects
  - Binary Association
  - Ternary Association
  - N-ary Association

Design Relationships - Navigability

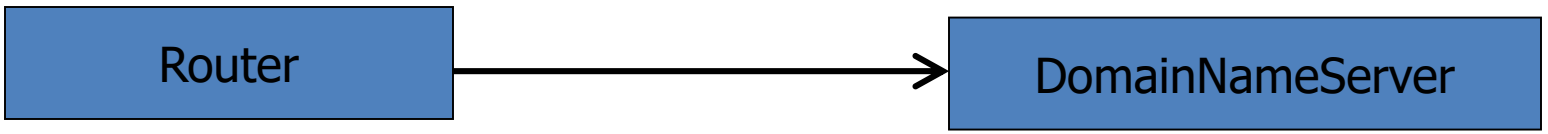
- Classes with associations can navigate (send messages) to each other.
- By default the associations are bidirectional.
- Sometimes you want to limit the message sending to only one direction.
- Illustrated with an arrow pointing in the direction a message can be sent.

Given a User, you can find that user's current password for authentication. But given a password, you cannot find the corresponding user.



One-way Association

- We can constrain the association relationship by defining the *navigability* of the association.
- In one way association, We can navigate along a single direction only
- Denoted by an arrow towards the server object
- Here, a *Router* object requests services from a *DNS* object by sending messages to (invoking the operations of) the server. The direction of the association indicates that the server has no knowledge of the *Router*.



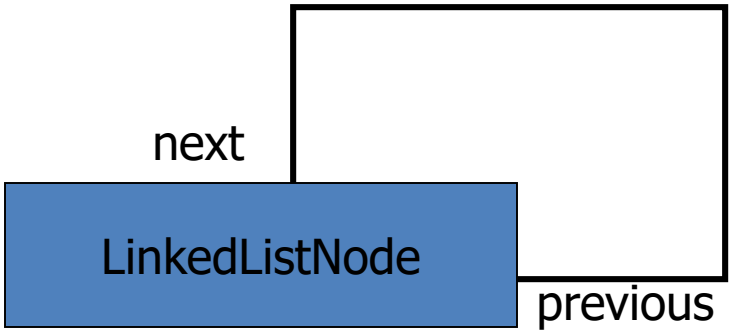
Two-way Association

- We can navigate in both directions
- Denoted by a line between the associated objects

- Employee works for company
- Company employs employees

Self Association

A class can have a *self association/ reflexive Association.*



Two instances of the same class:
Pilot
Aviation engineer

w.r.t Objects

Binary Association

- Associates objects of exactly two classes

- Denoted by a line, or an arrow between the associated objects

Employee Company

- Association “works-for” associates objects of exactly two classes

Ternary Association

- Associates objects of exactly three classes.
- Denoted by a diamond with lines connected to associated objects.

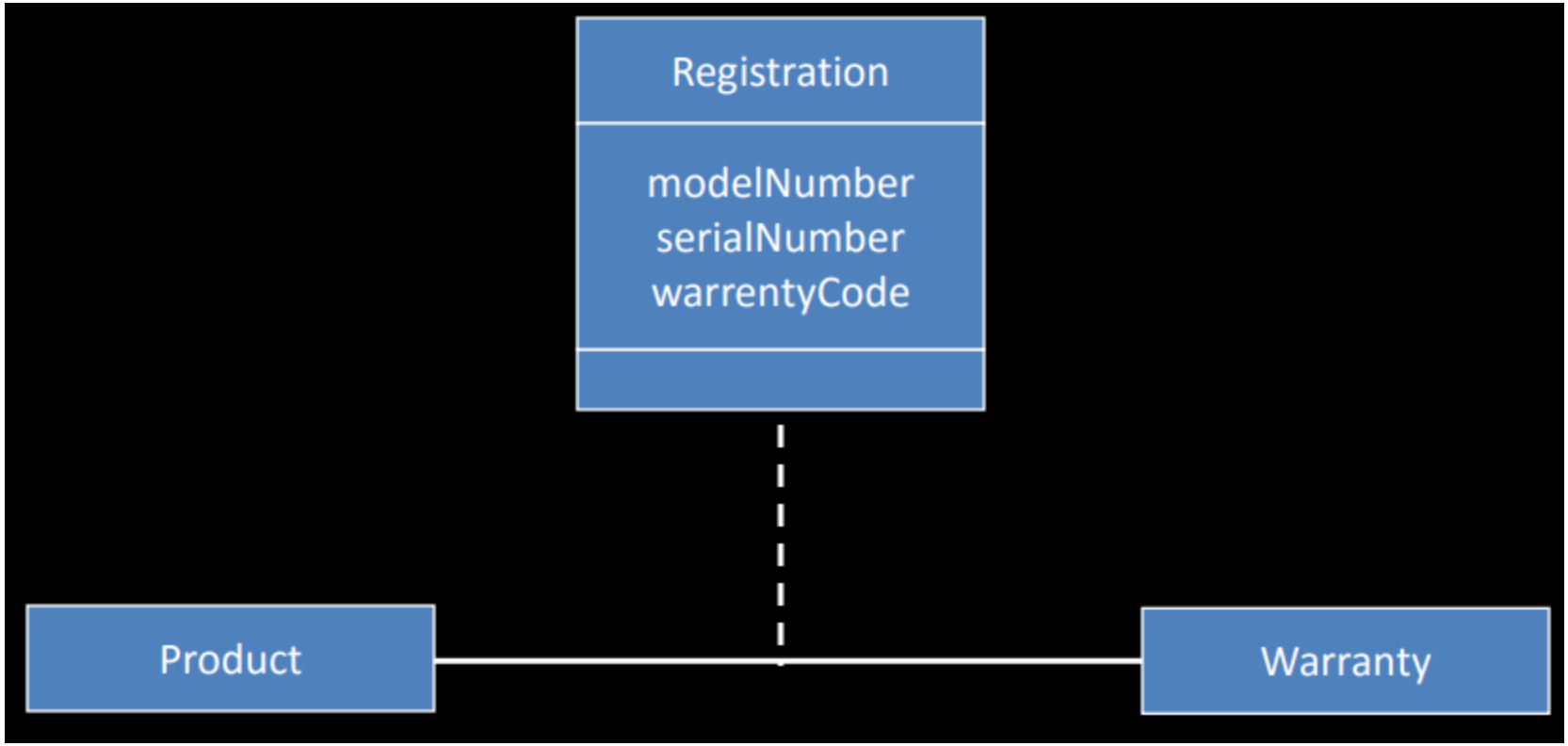
Student

Teacher

Course

Association link

- Associations can also be objects themselves, called link classes or an association classes.
- A link is an instance of an association.



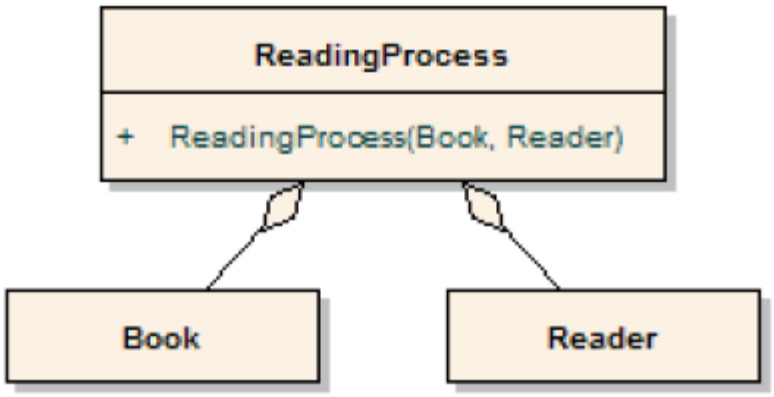
N-ary Association

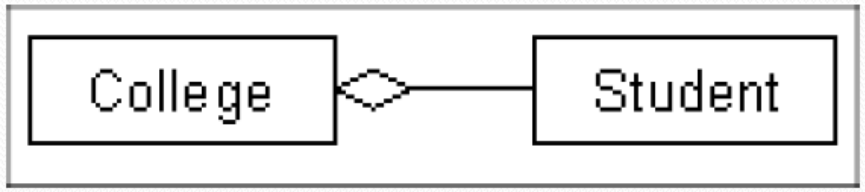
- An association between 3 or more classes
- Practical examples are very rare

Association Relationship: Aggregations

- Objects that contain other objects can be modeled 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.

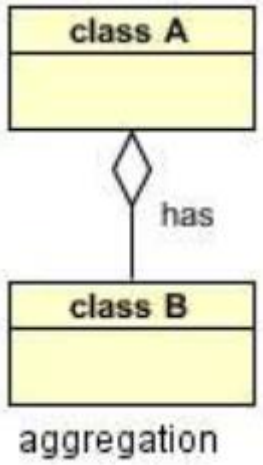
class ReadingProcess





Aggregation

- Class A has class B
- If class A stores the reference to class B for later use we would have a different relationship called **Aggregation**,
- A more common and more obvious example of Aggregation would be via setter injection.

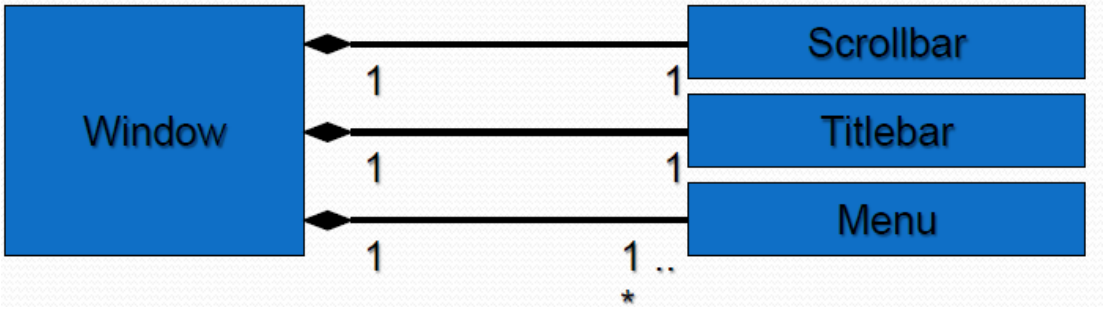


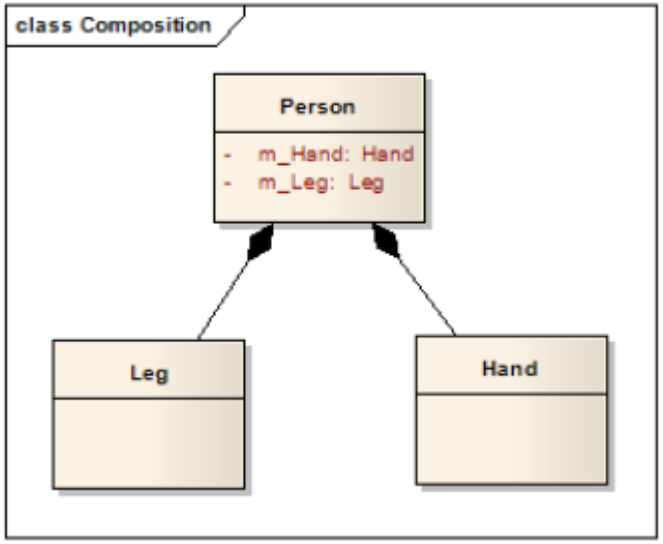
aggregation

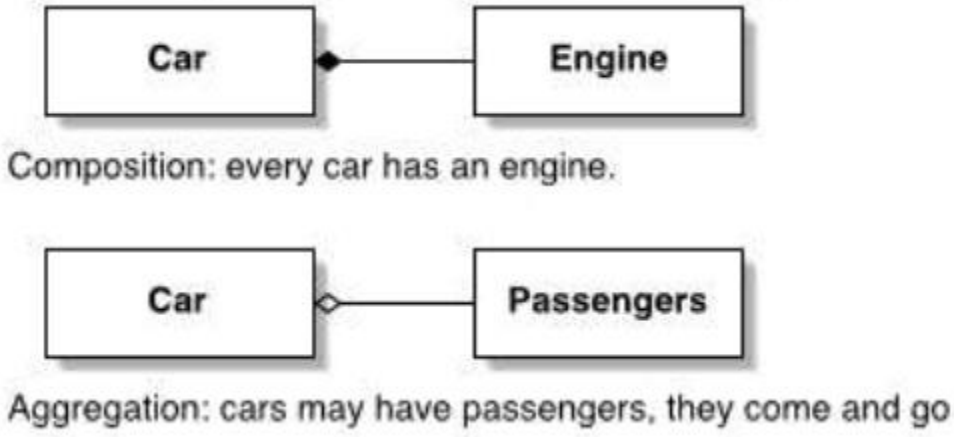
```
1public class A {
2
3    private B _b;
4
5    public void setB(B b) { _b = b; }
```

Association Relationship: Composition

- 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.

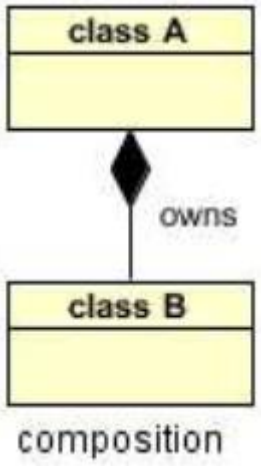






Composition

- Class A owns class B
- In Composition the containing object is responsible for the creation & life cycle of the contained object (either directly or indirectly).

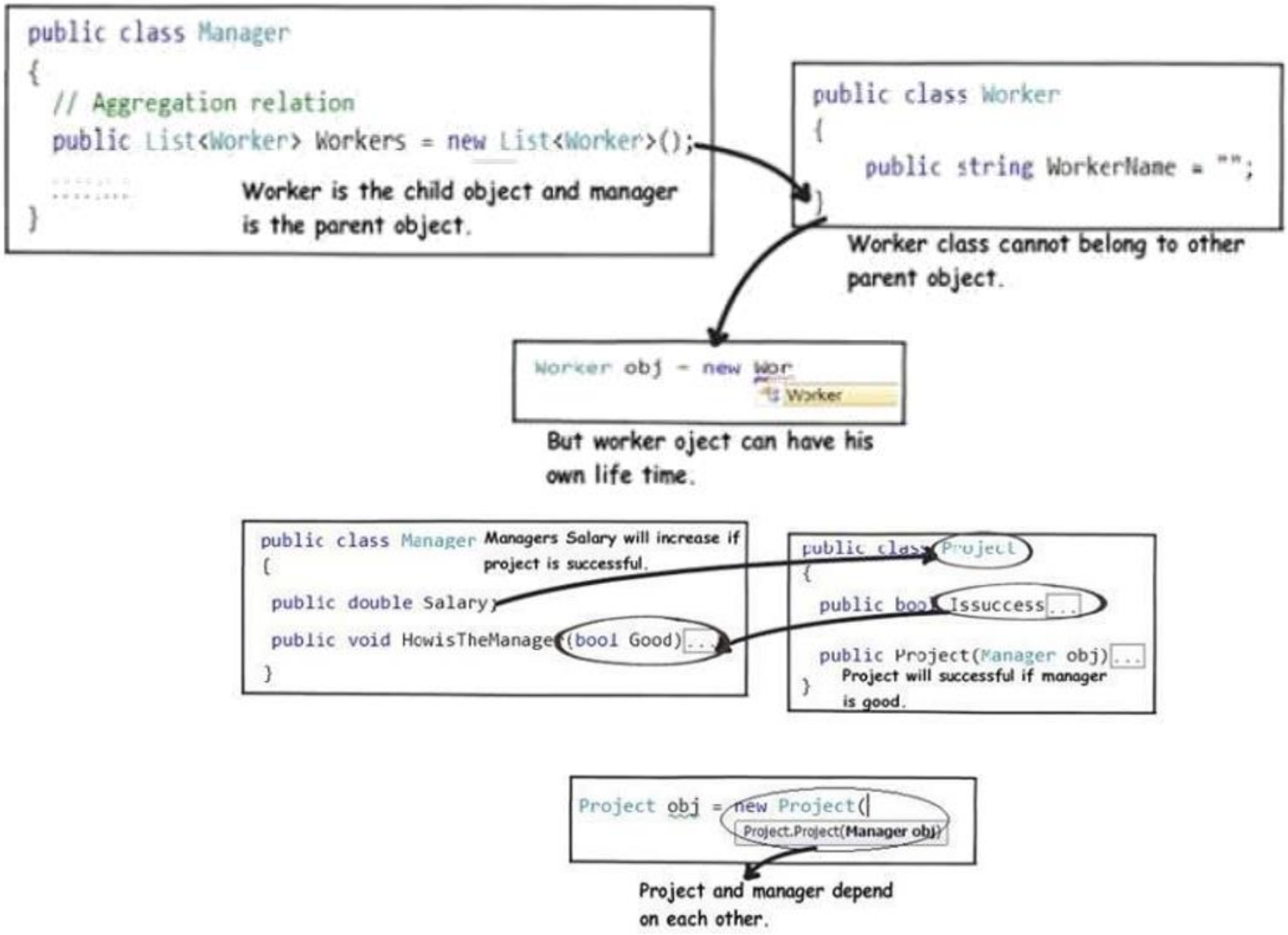


Via Member Initialization:

```
1public class A {
2
3    private B _b = new B();
```

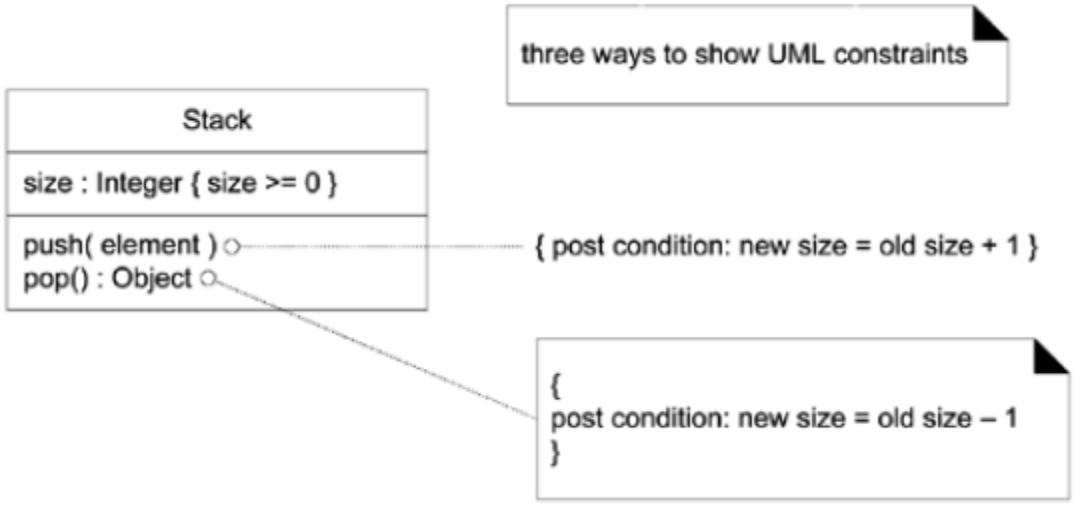
```
1public class A {
2
3    private B _b;
4
5    public A() {
6        _b = new B();
7    } // default constructor
```

Association & Composition



Constraints

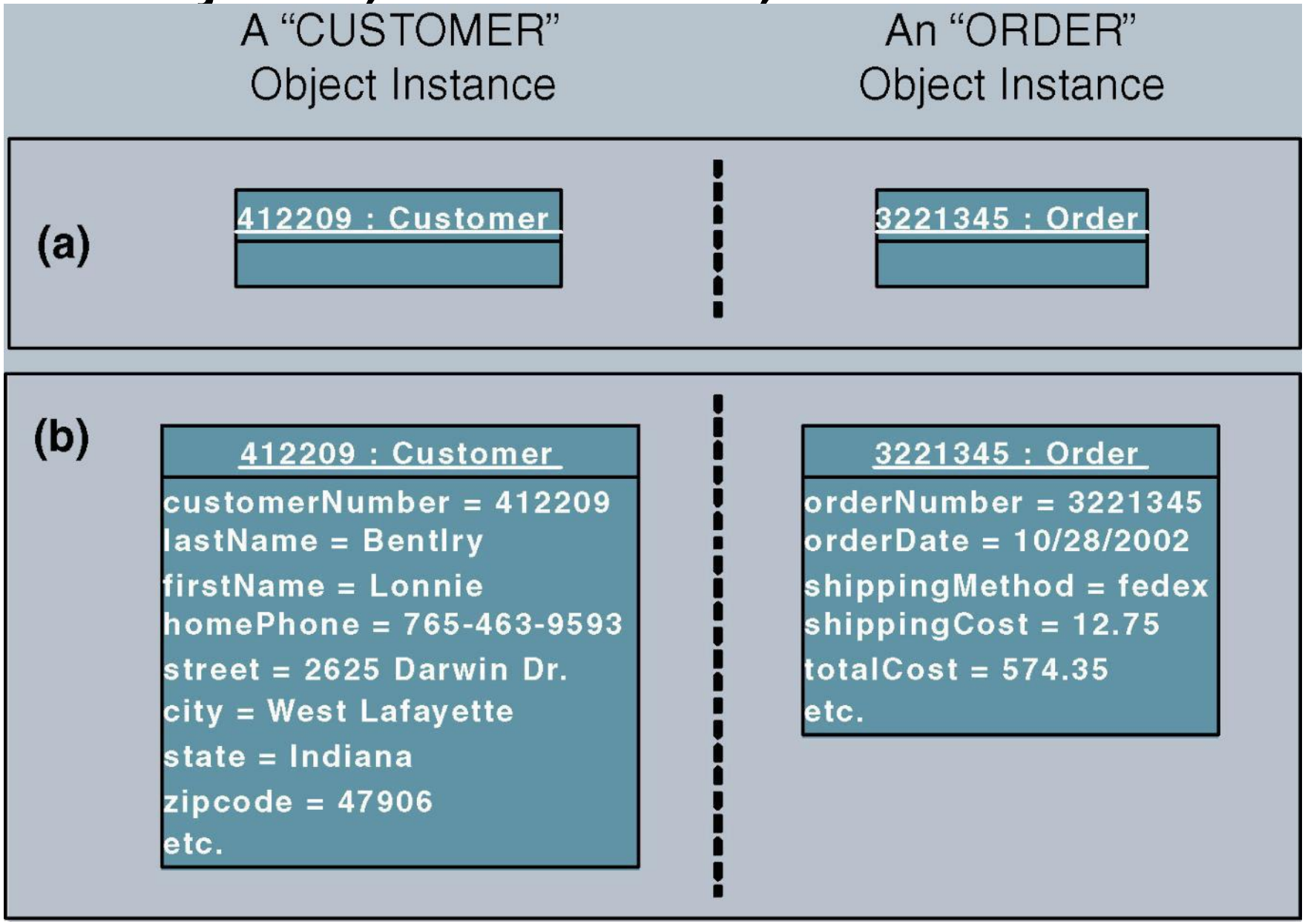
- A UML constraint is a restriction or constraint on a UML element.
- It is visualized in text between braces.



Objects & Instances

- **Object**: Something that is or is capable of being seen, touched, or otherwise sensed, and about which users store data and associate behavior.
  - Person, place, thing, or event
  - Employee, customer, instructor, student
  - Warehouse, office, building, room
  - Product, vehicle, computer, videotape
  - Abstract nouns etc.
- **Object instance**: Each specific person, place, thing, or event, as well as the values for the attributes of that object.

Objects, Attributes, & Instances



Persistent and Transient Object Classes

Persistent class – a class that describes an object that outlives the execution of the program that created it.

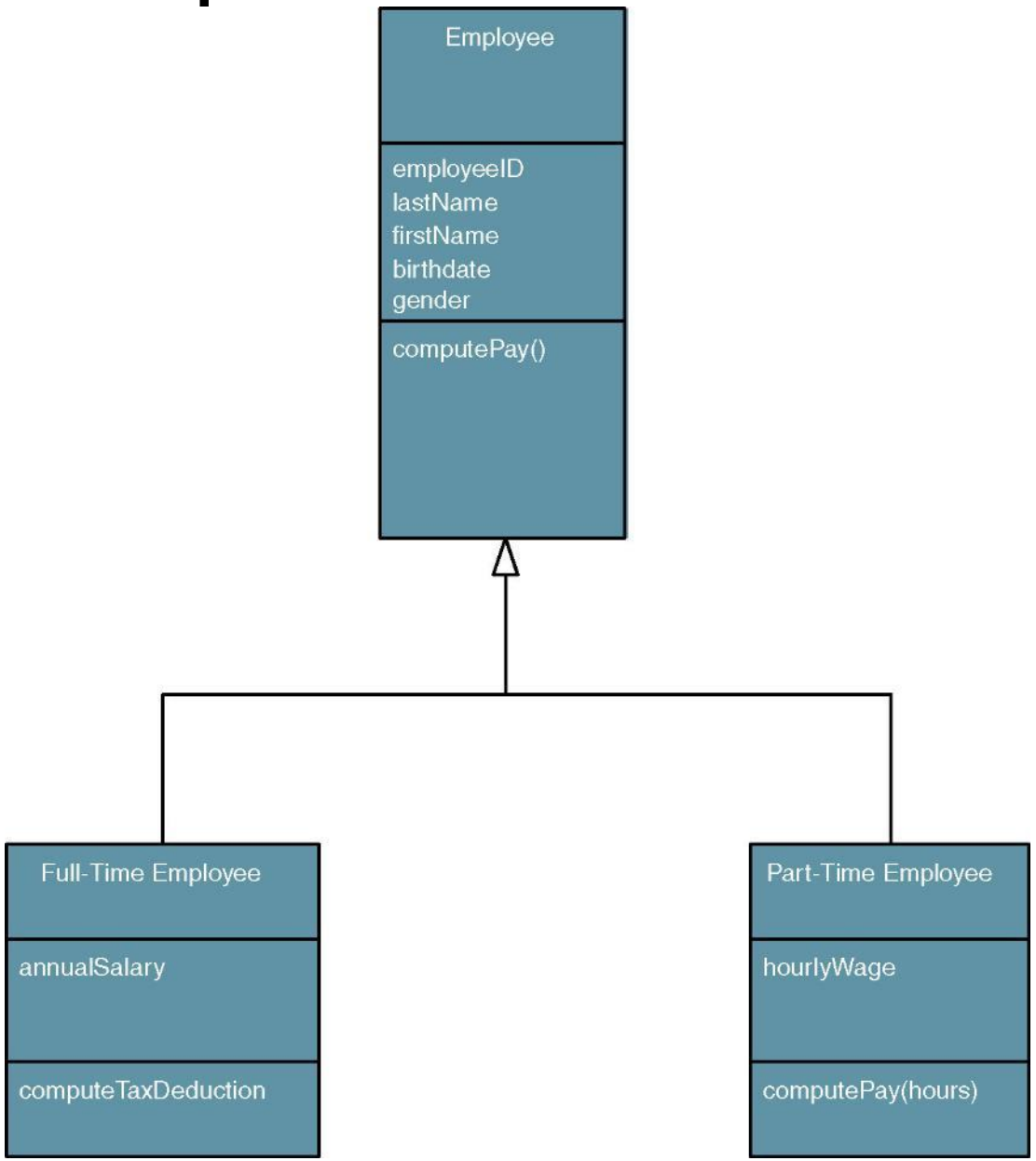
- Stored permanently as in a database

Transient object class – a class that describes an object that is created temporarily by the program and lives only during that program's execution.

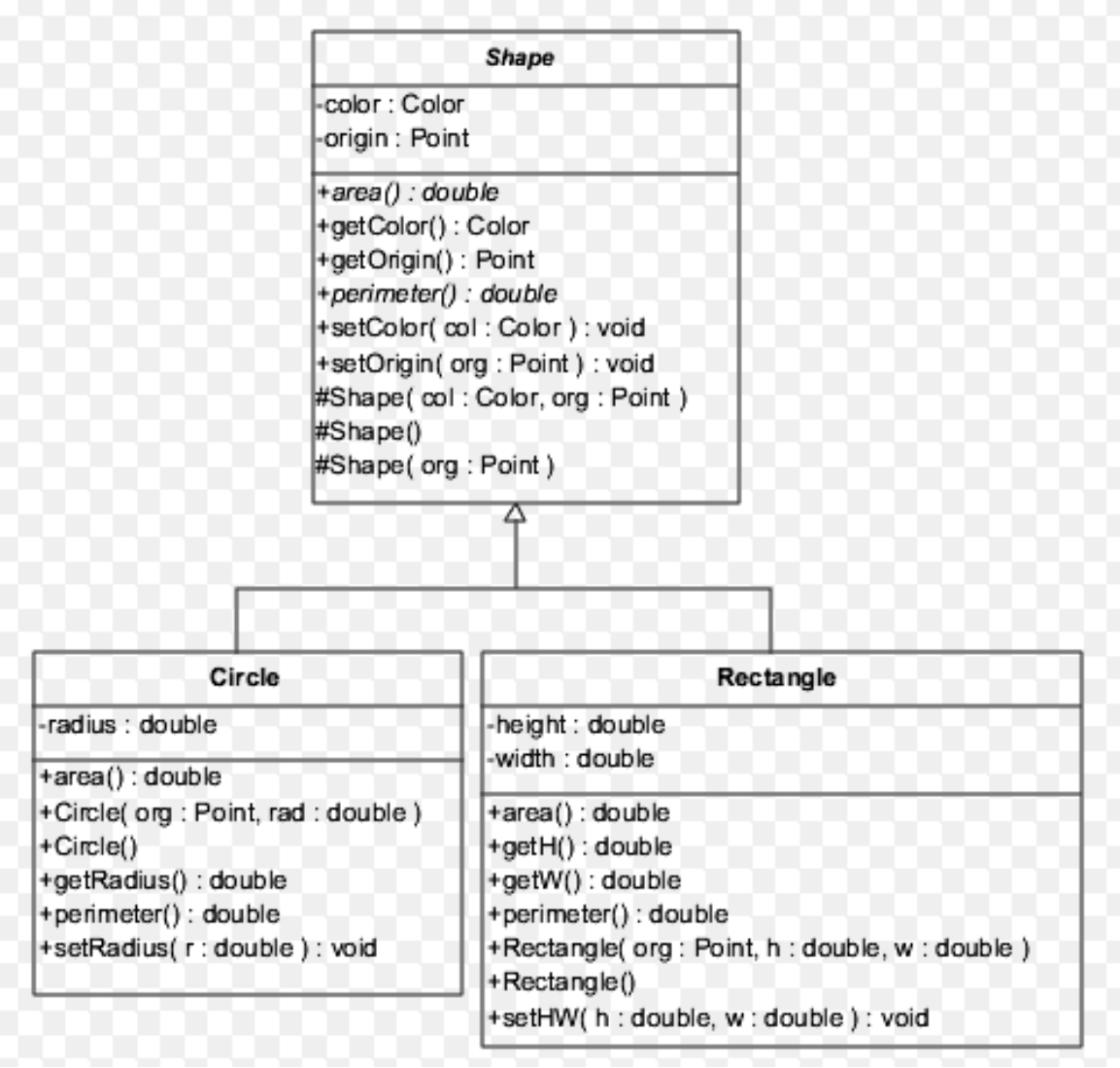
Polymorphism

- **Polymorphism**: Literally meaning "many forms," the concept that different objects can respond to the same message in different ways.

- **Override**: A technique whereby a subclass (subtype) uses an attribute or behavior of its own instead of an attribute or behavior inherited from the class (super type).

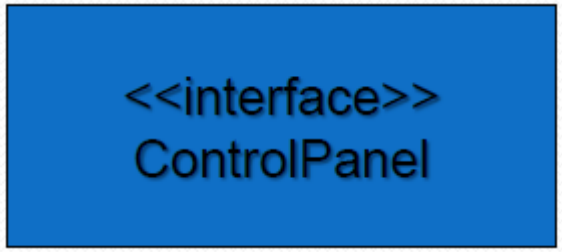


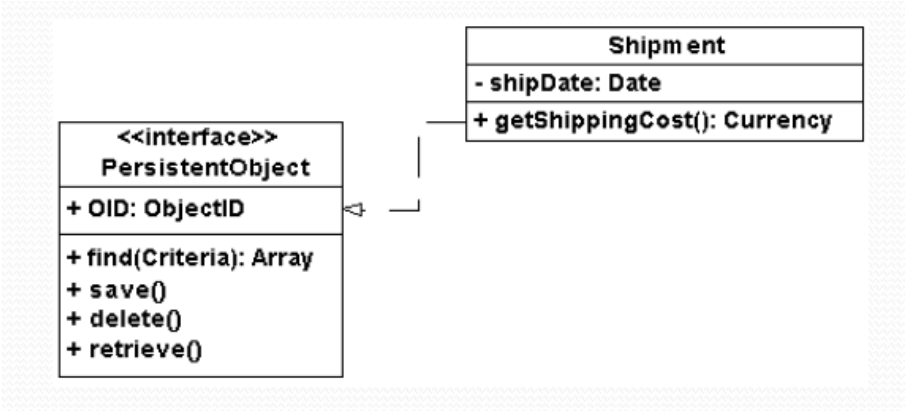
Class Diagram: Example



Interfaces

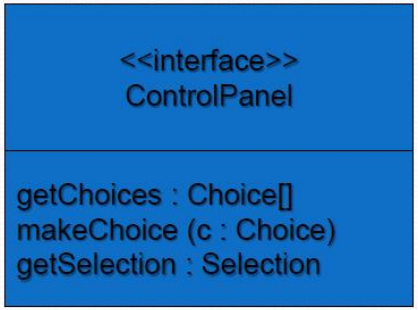
- An interface is a named set of operations that specifies the behavior of objects without showing their inner structure.
- It can be rendered in the model by a one- or two- compartment rectangle, with the stereotype <<interface>> above the interface name.
- To realize an interface a class or component must implement the operations and attributes defined by the interface.





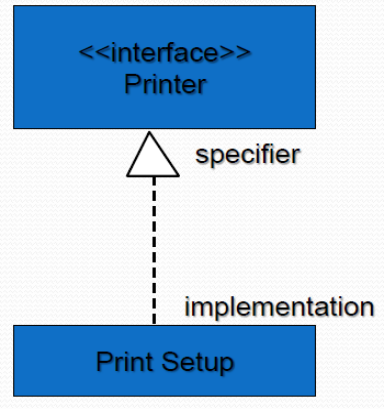
Interfaces Services

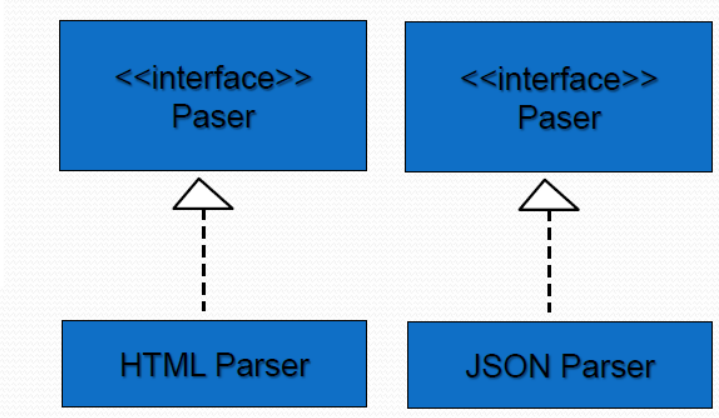
- Interfaces do not get instantiated. They have no state. Rather, they specify the services offered by a related class



INTERFACE REALIZATION RELATIONSHIP:

- A realization relationship connects a class with an interface that specifies its behavioral specification.
- It is rendered by a dashed line with a hollow triangle towards the specifier.

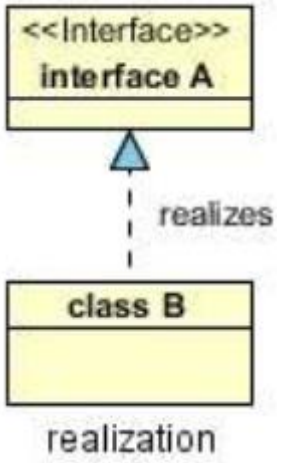




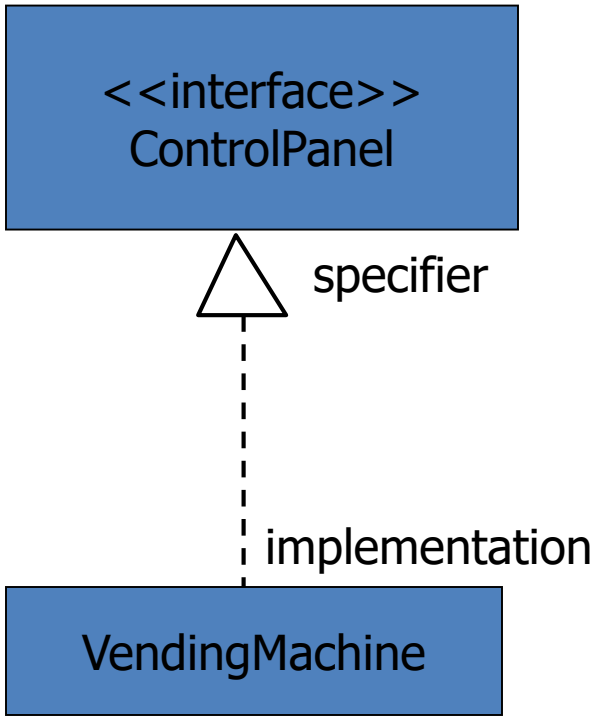
Interfaces

- **Realization:** class B realizes class A (or class A is realized by class B)

```
1public interface A {
2
3    ...
4
5} // interface A
6
7public class B implements A {
8
9    ...
10
11} // class B
```



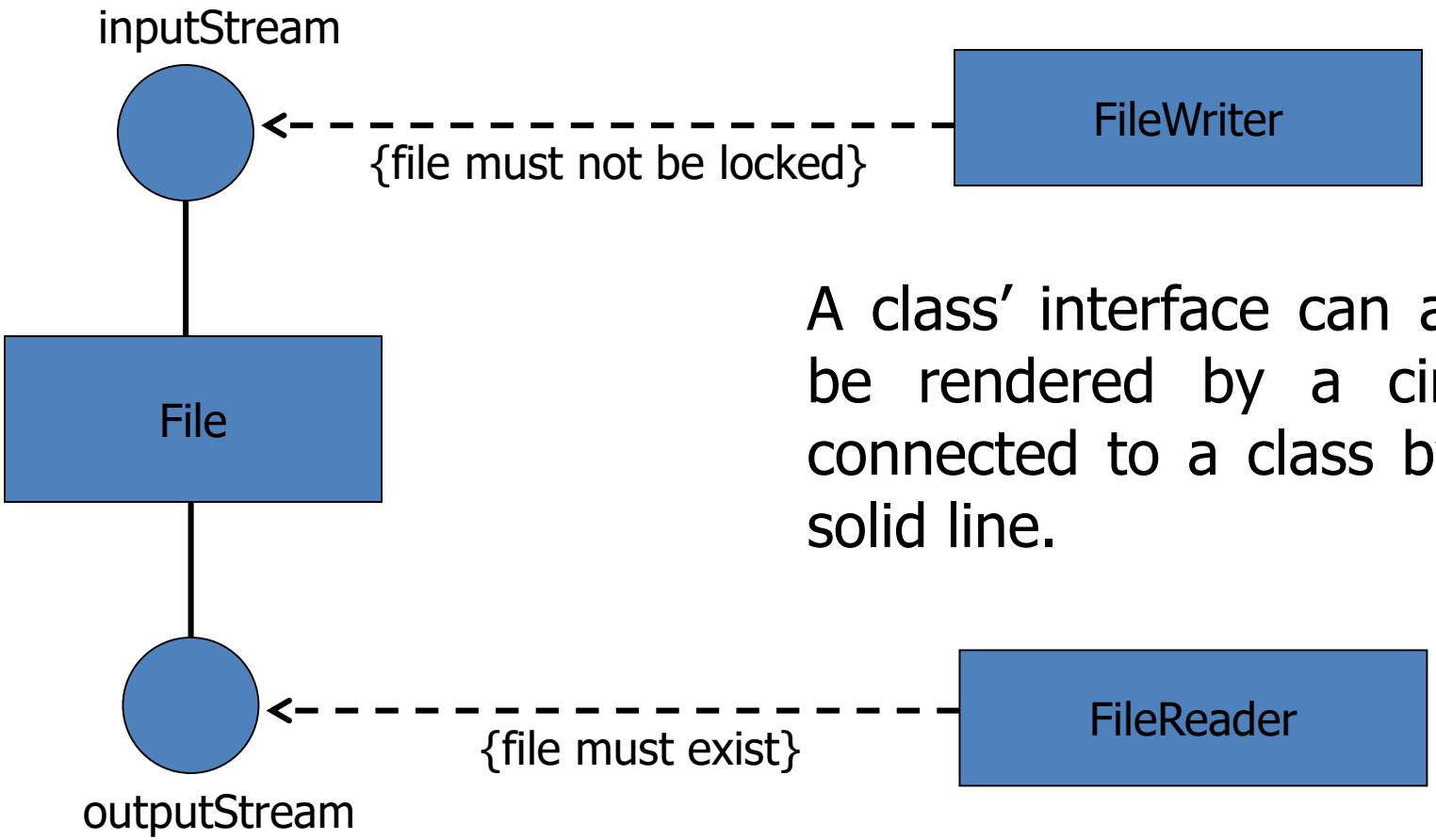
Interface Realization Relationship



A *realization* relationship connects a class with an interface that supplies its behavioral specification. It is rendered by a dashed line with a hollow triangle towards the specifier.

```
public interface A {
...
} // interface A
public class B implements A {
...
} // class B
```

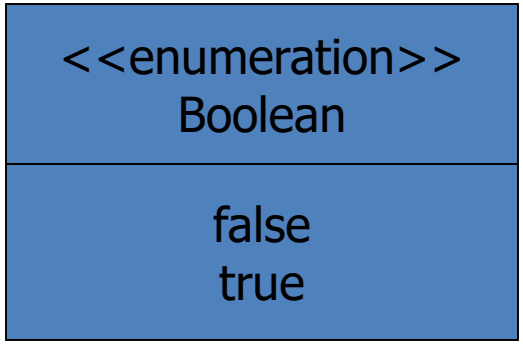
Interfaces


inputStream
FileWriter
{file must not be locked}
File
A class' interface can also be rendered by a circle connected to a class by a solid line.
FileReader
{file must exist}
outputStream

Keywords, Abstract and Final Classes

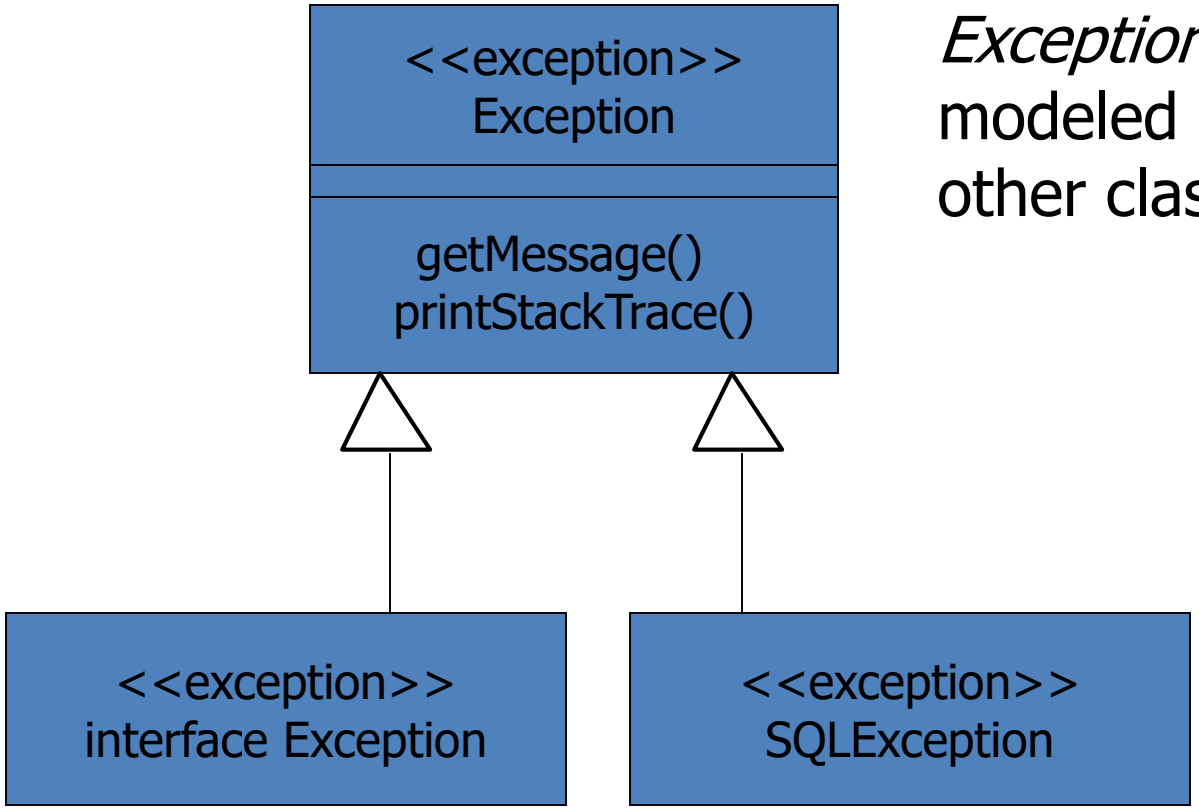
- UML keyword is a textual adornment to categorize a model element.
- For e.g. to categorize interface <<interface>> keyword is used. <<actor>> keyword is used to categorize actor.
- Most keywords are shown in guillemet but some are shown in curly brackets like {abstract} and {ordered}.
- Abstract classes and operations can be shown either with the {abstract} tag or by italicizing the name.
- Final classes and operations that cant be overridden in subclasses are shown with the {leaf} tag.

Enumeration



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

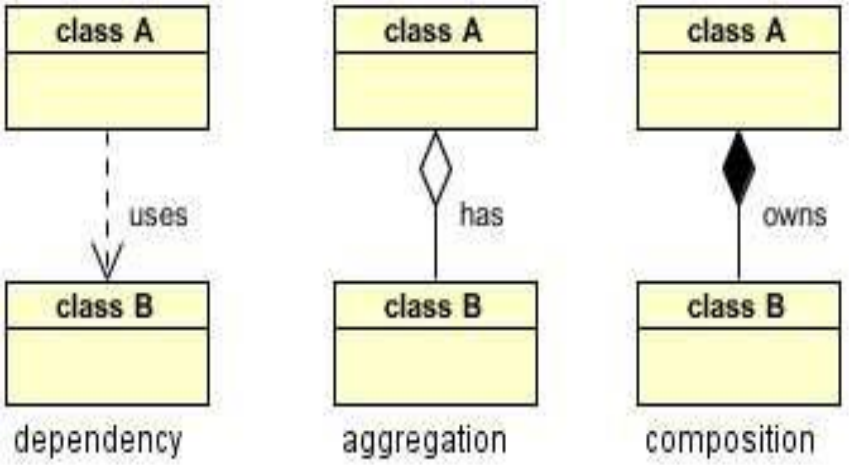
Exceptions

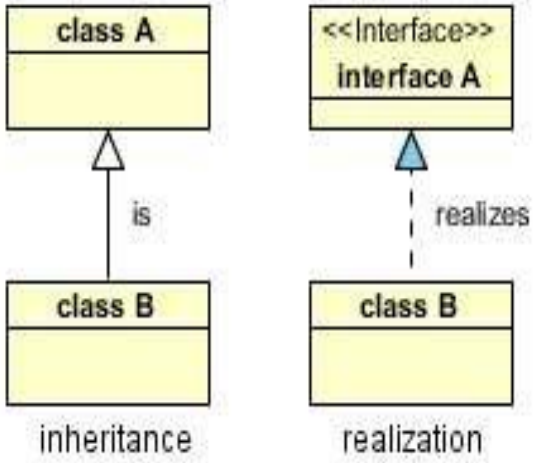


Exceptions can be modeled just like any other class.

Relationships in a NutShell

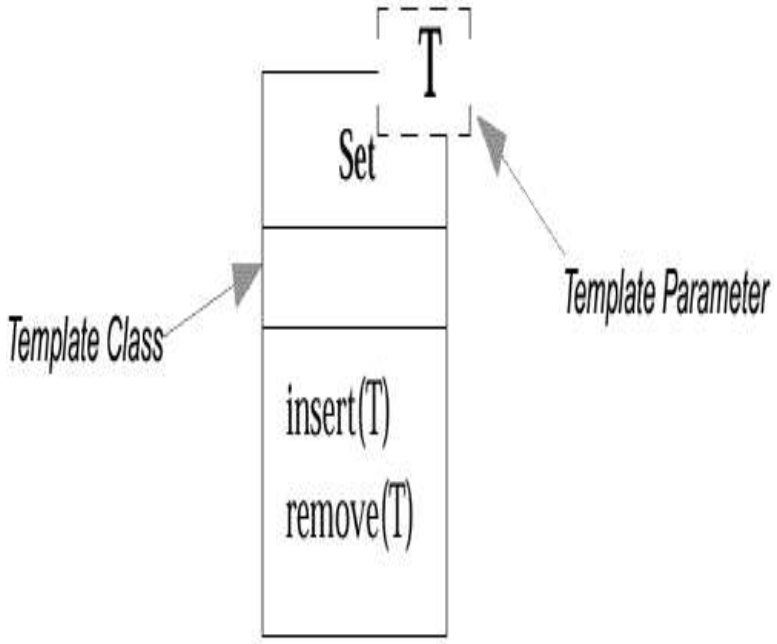
- Dependency : class A uses class B
- Aggregation : class A has a class B
- Composition : class A owns a class B
- Inheritance : class B is a Class A (or class A is extended by class B)
- Realization : class B realizes Class A (or class A is realized by class B)





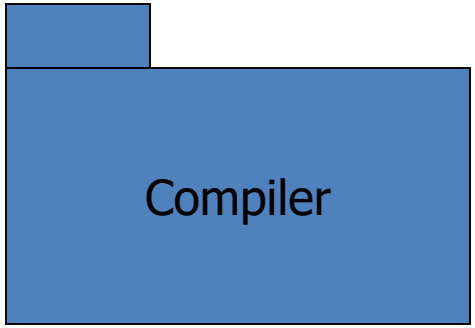
Parameterized Class

- A *parameterized class* or *template* defines a family of potential elements.
- To use it, the parameter must be bound.
- A *template* is rendered by a small dashed rectangle superimposed on the upper-right corner of the class rectangle. The dashed rectangle contains a list of formal parameters for the class.




```
class Set <T> {
void insert (T newElement);
void remove (T anElement);
```

Packages

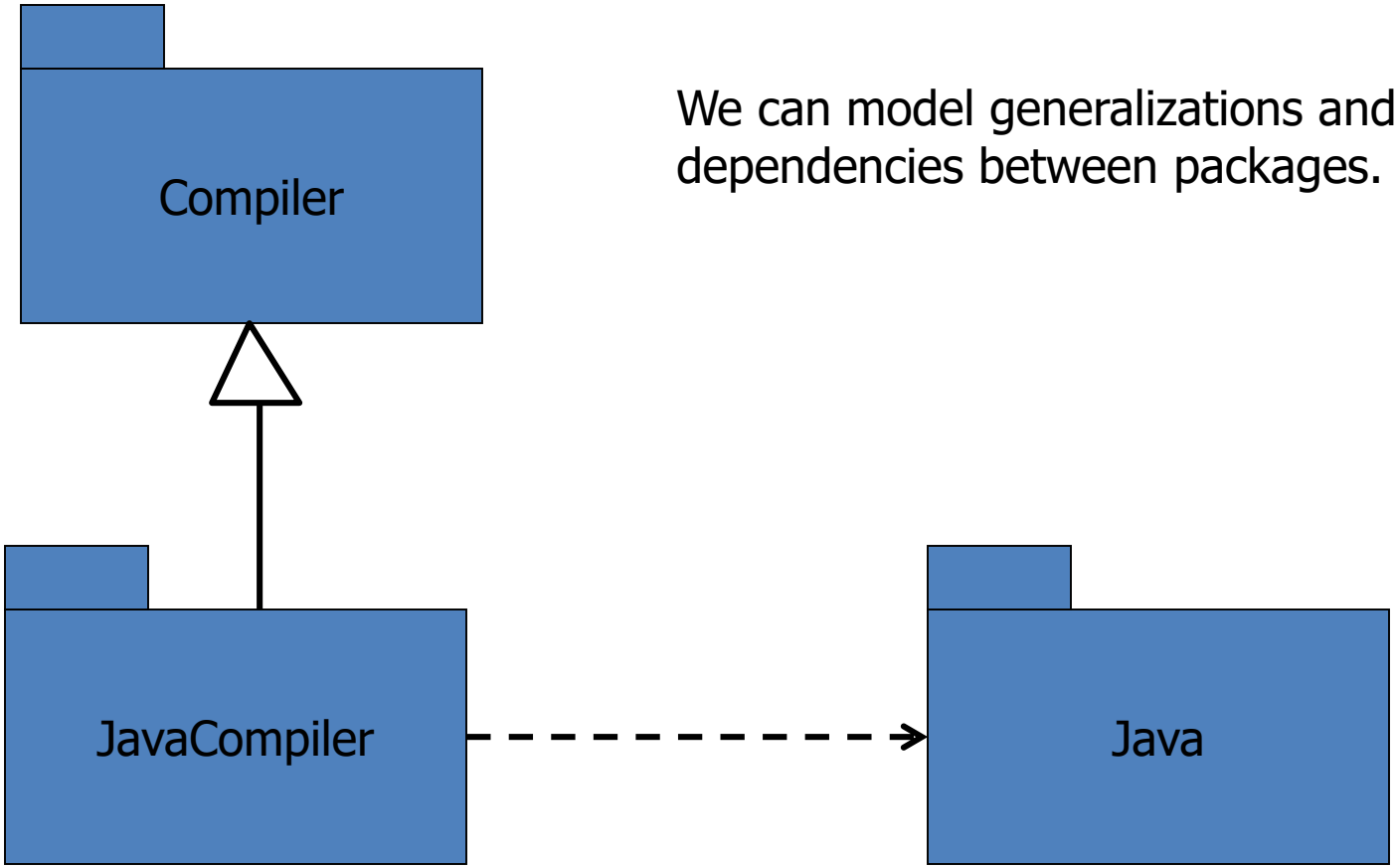


A *package* is a container-like element for organizing other elements into groups.

A package can contain classes and other packages.

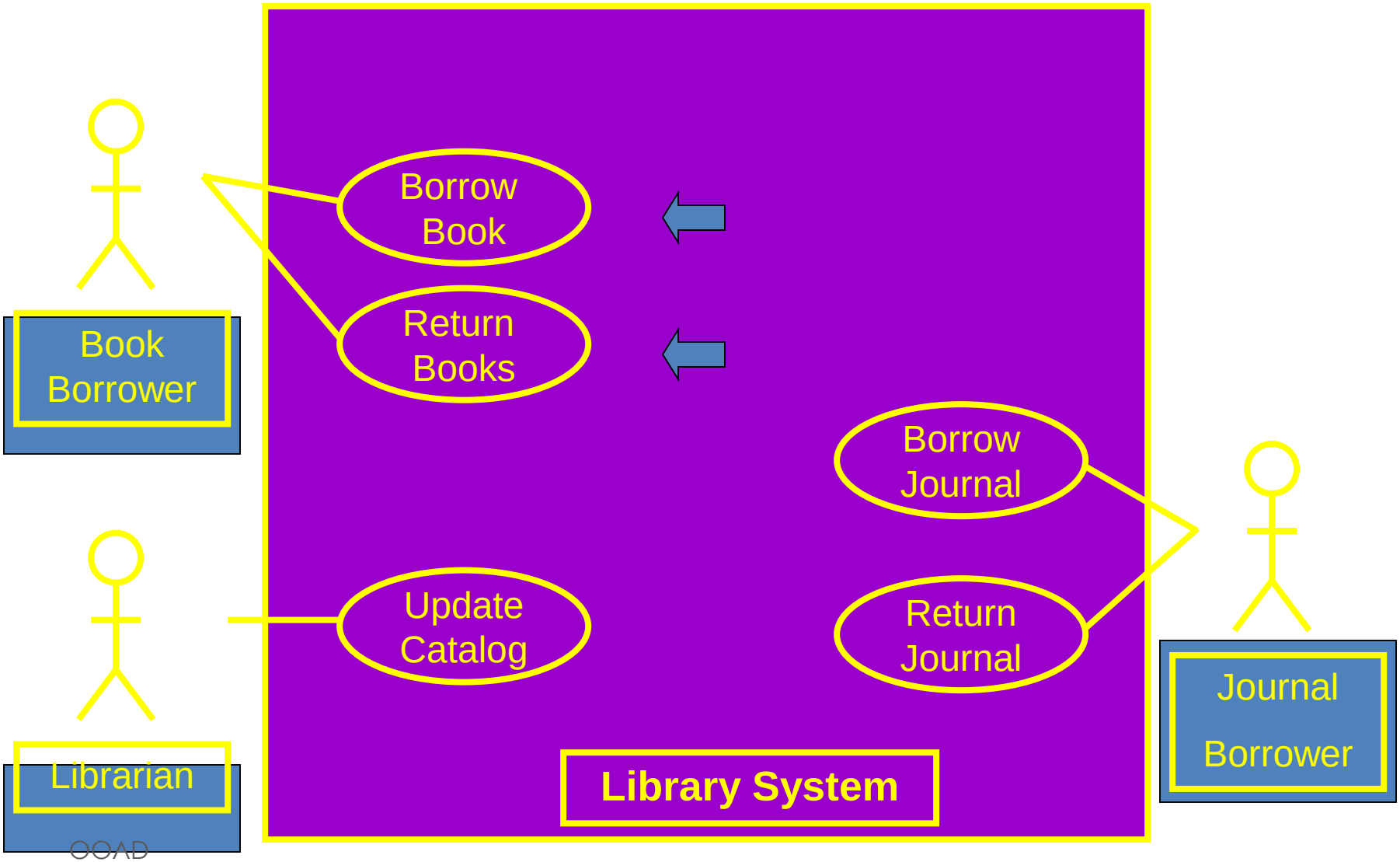
Packages can be used to provide controlled access between classes in different packages.

Packages (Cont'd)



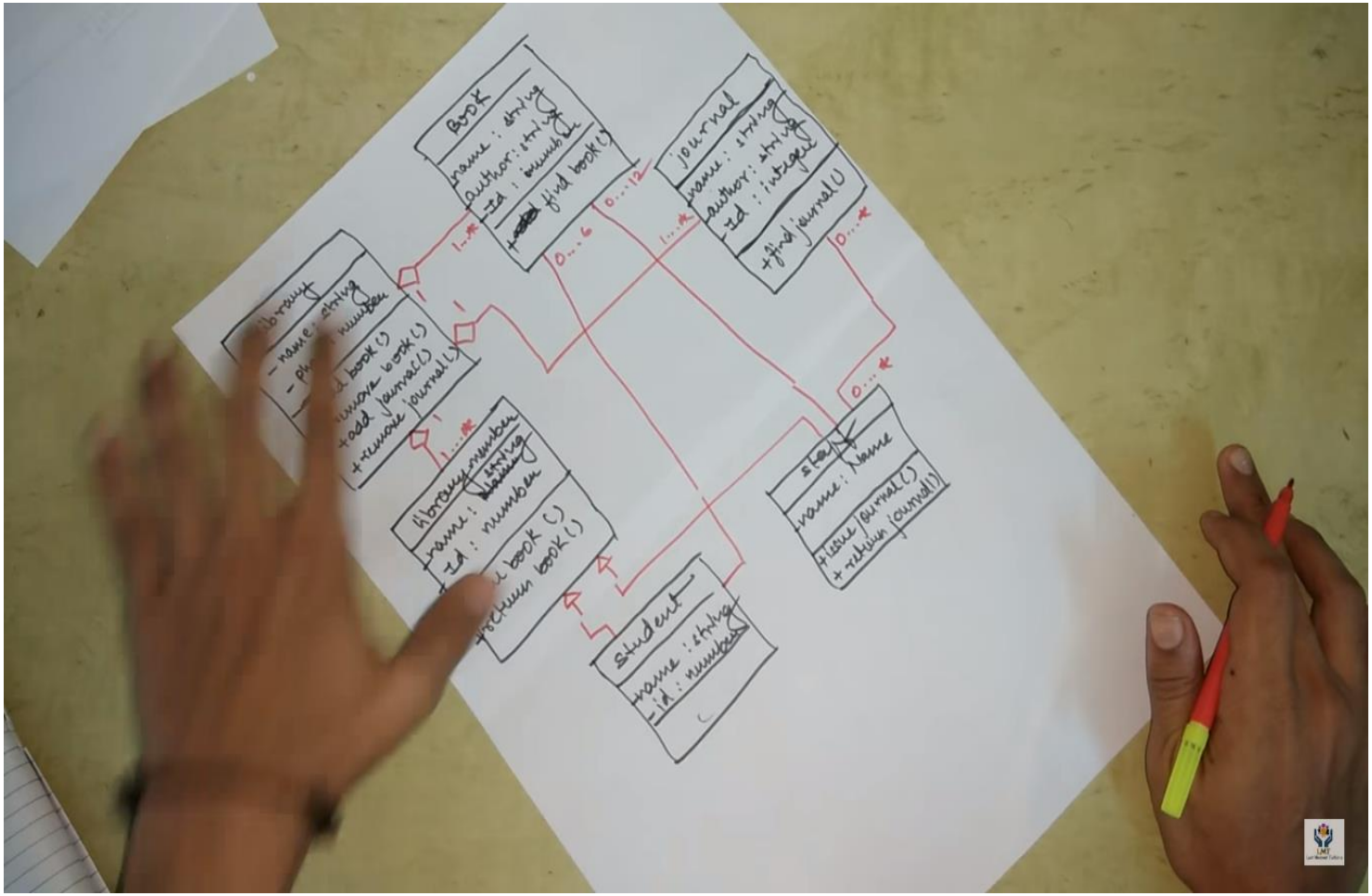
We can model generalizations and dependencies between packages.

Use Diagram for a Library System



Exercise

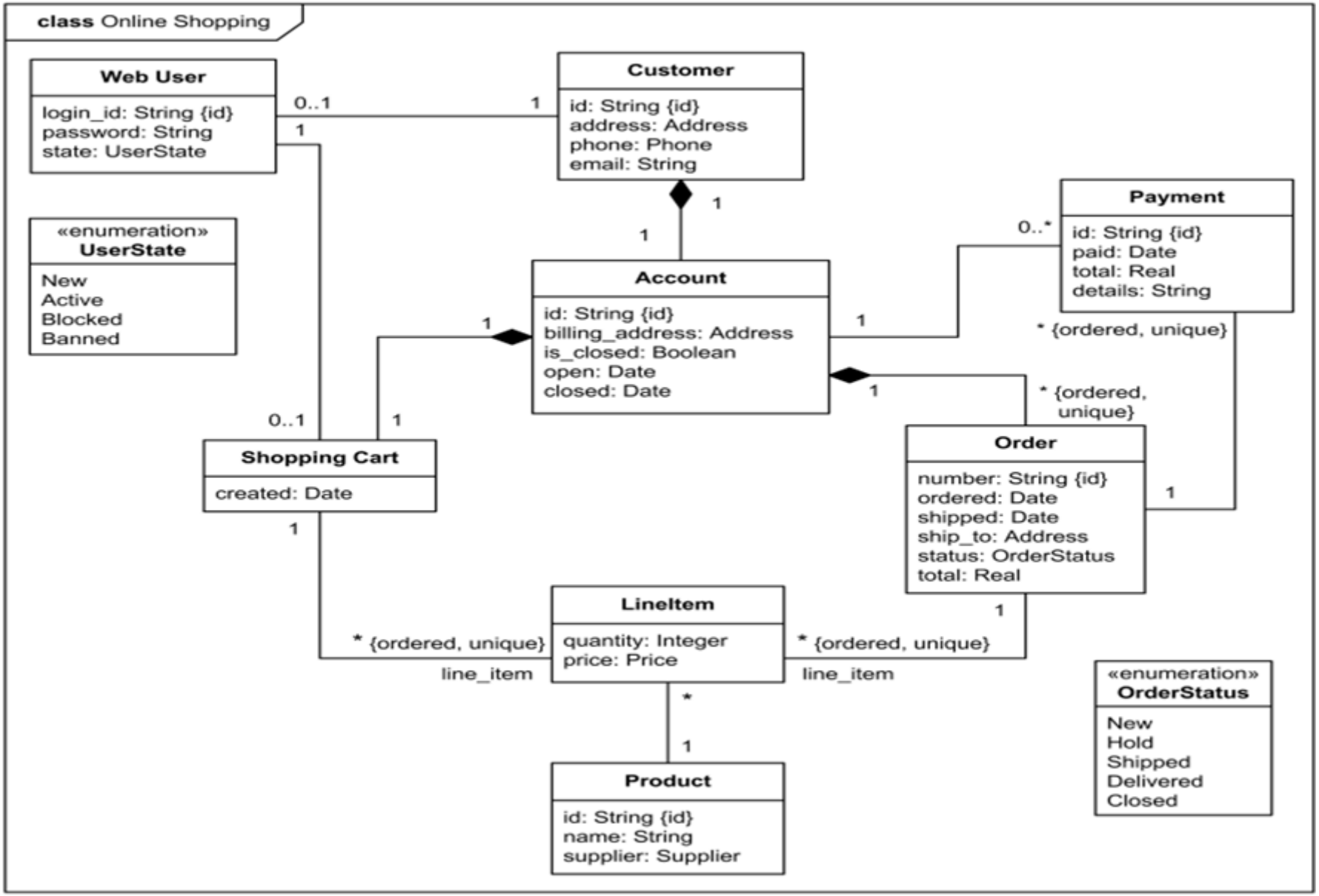
- Draw a class diagram of a campus library management system. Attributes of library include name, phone number. Library contains books and journals that can be added or removed form the library.
- Each book and journal has an id, name, author name and publisher.
- Library member can issue and return the book.
- Library member can be student or staff. Students can issue 4 books at a time and staff can have 8 books.
- Journals are available for staff only.


Book
+name : string
-author : string
-id : number
+find book()
journal
+name : string
-author : string
-id : integer
+find journal()
student
+name : string
-id : number
+name : Name
+issue journal()
+return journal()
1...*
0...6
0...12
1...*
0...*
0...*
1
1
1...*

Your Turn: Exercise

- Draw the UML **class diagram** showing the domain model for online shopping. The purpose of the diagram is to introduce some common terms, "dictionary" for online shopping - Customer, Web User, Account, Shopping Cart, Product, Order, Payment, etc. and relationships between. It could be used as a common ground between business analysts and software developers.
- 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. 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 (new, hold, shipped, delivered, closed). 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 Activity



Object Association Matrix

	CLUB MEMBER	MEMBER ORDER	MEMBER ORDERED PRODUCT	PRODUCT
CLUB MEMBER		Places zero to many	Has purchased zero to many	XX
MEMBER ORDER	Is placed by one and only one		Contains one to many	XX
MEMBER ORDERED PRODUCT	Was purchased by one and only one	Is part of one and only one		Relates to one and only one
PRODUCT	XX	XX	Sold as zero to many	

READING

Chapter 16 – Applying UML and Pattern by Craig Larman 3rd Edition

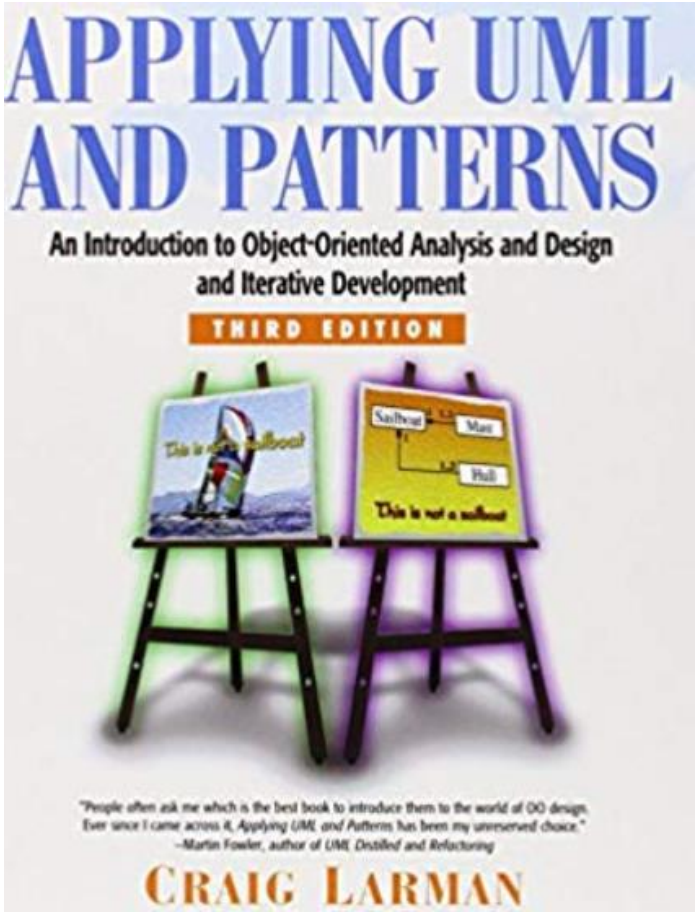
Chapter 16. UML Class Diagrams

To iterate is human, to recurse, divine.

anonymous

Objectives

- Provide a reference for frequently used UML class diagram notation.



END OF TOPIC 8

-COMING UP!!!!!!

-Midterm Examination

-RUP

-Activity Diagrams