OOAD: Data Modeling

Presenter: Dr. Ha Viet Uyen Synh.

Objects have behaviors

In old style programming, you had:

- **data**, which was completely passive
- **functions**, which could manipulate any data

In O-O programming, an object contains both data and methods that manipulate that data

An object is *active*, not passive; it *does* things

An object is *responsible* for its own data

But it can *expose* that data to other objects

Objects have state

An object contains data

The data represent the state of the object

Data can also describe the relationship of the object to other objects

Example: a *checkingAccount* might have

A balance (the internal state of the account)

An owner (some object representing a person)

An accountNumber (used as an ID number)

Classes

A class describes a set of objects

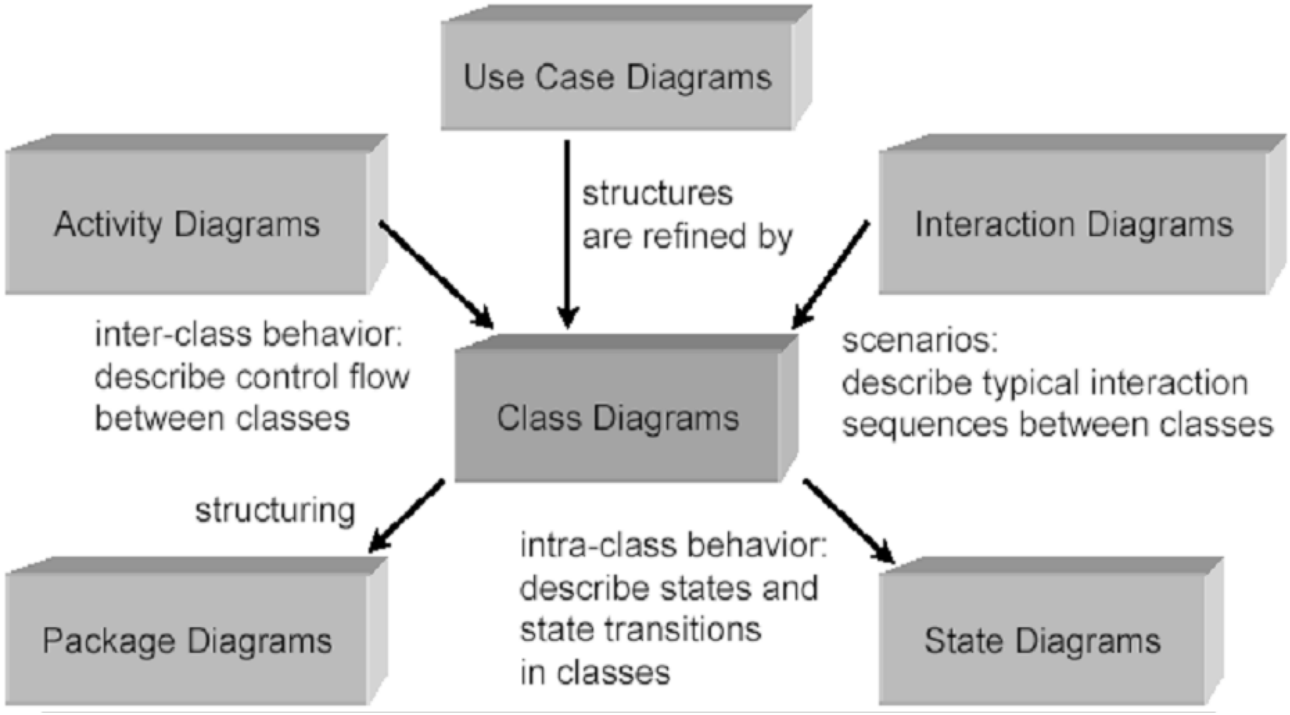
The objects are called instances of the class

A class describes:

- How to construct an object
- What data (fields) are in each object
- The methods (actions) the objects can perform

In addition, a class can have data and methods of its own (not part of the objects)

Role of Class Diagram



Class diagrams are central for analysis, design and implementation.
Class diagrams are the richest notation in UML

Part I

CLASSES

The nouns and the verbs

Start from the requirements document;

- in **function-oriented design** you would concentrate on the verbs, which correspond to actions ("do this");

- in **object-oriented design** you underline the nouns, which describe objects

Example:

The elevator will close its door before it moves to another floor.

Avoiding useless classes

The life time of an object is the most important criteria to decide if it becomes a class or not.

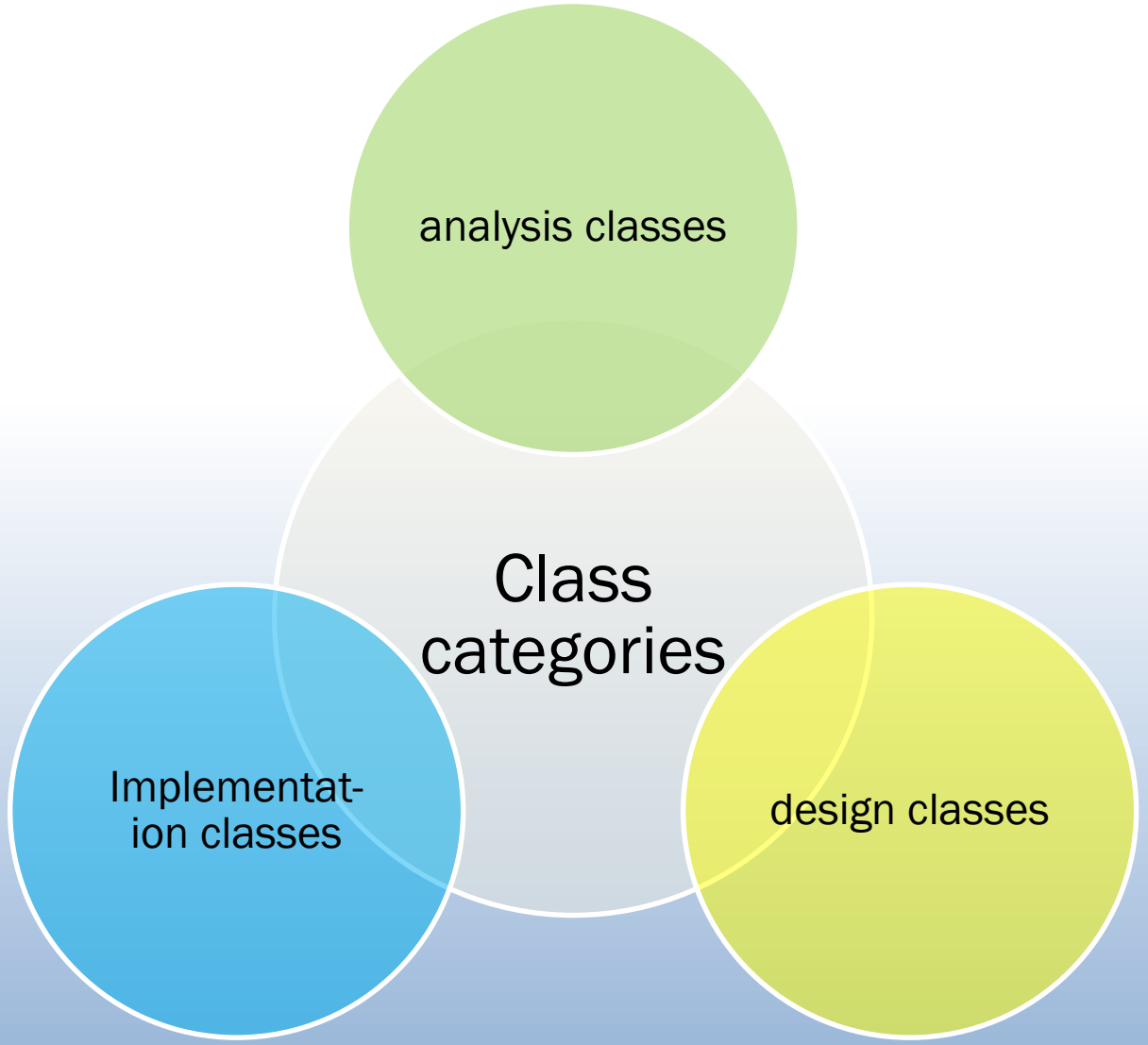
The nouns of a requirements document will cover some classes of the final design, but will also include many "false alarms": concepts that should not yield classes.

Example:

Door _ the only relevant property of elevator doors for this system is that they may be opened and closed.

Floor _ Each floor has a floor number or some floors may have special access rights defining who can visit them

GENERAL HEURISTICS FOR FINDING CLASSES



Example

Analysis classes:

sensors, devices, airplanes, employees, paychecks, tax returns, paragraphs, functions.

Implementation classes:

stack, queue, …

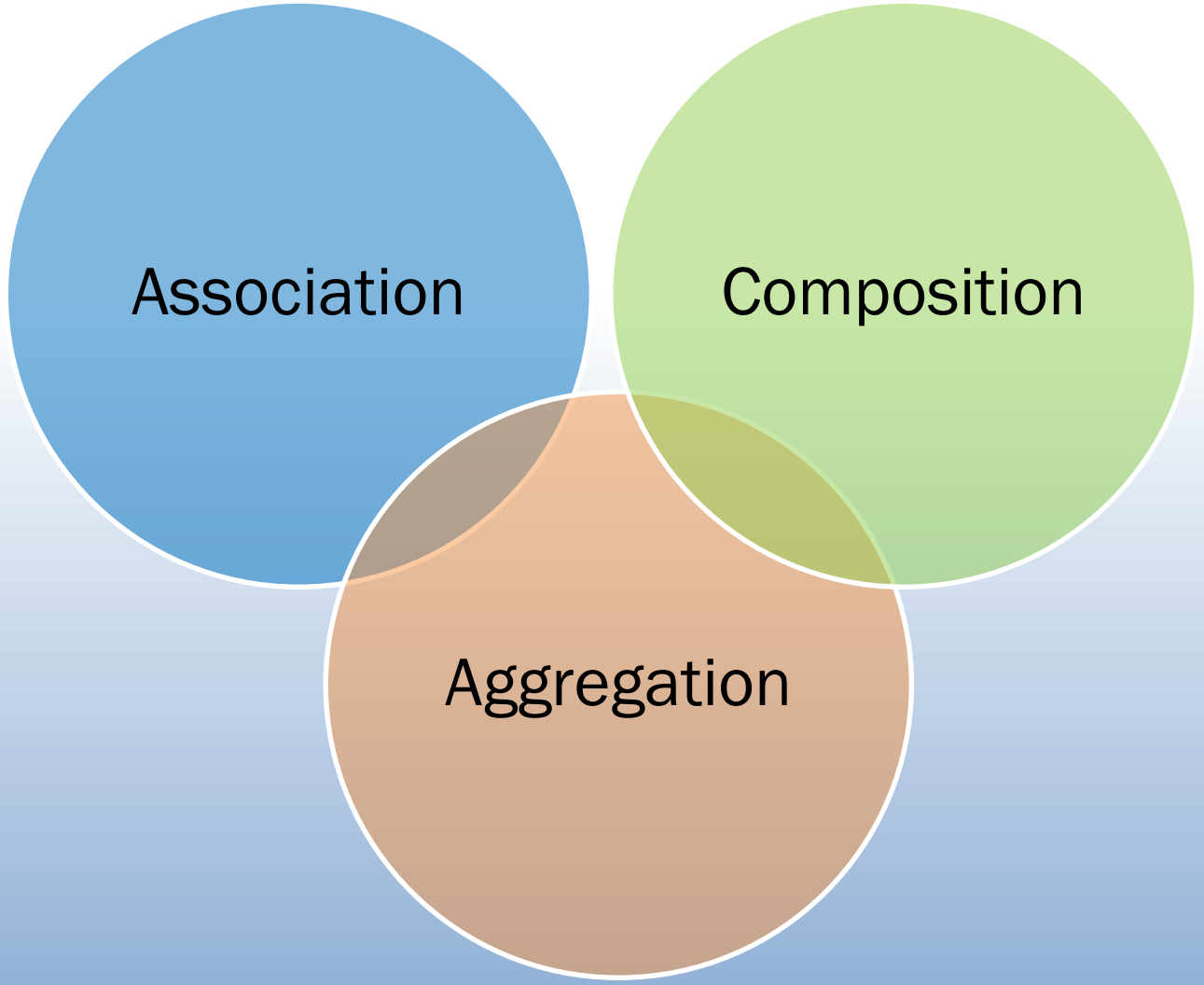
Design classes:

state, application, command, history-list, …

Part II

RELATIONSHIPS

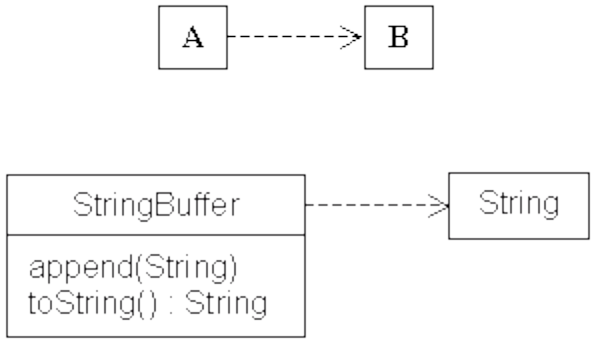
Types of Relationships



Dependencies

A dependency is one of the weakest UML relationships. A dependency between two classes implies that a change to one class might require a change to the other class. The notation for dependency is a directed dashed arrow.

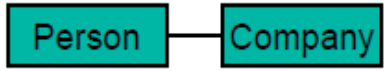
For example, if A depends on B, a change in the interface of B might require a change to A



Associations

Associations represent relationships between **instances** of classes.

"Companies *employ* persons."

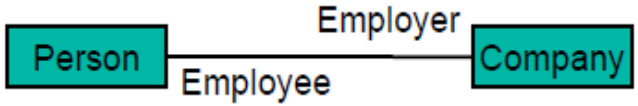


From the conceptual perspective, associations represent conceptual relationships between **classes**.

"Peter and Mary *are employed* **by** IBM."

Instances are marked by underlining.

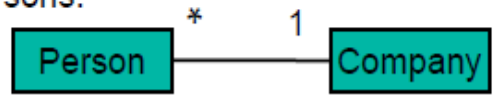
Each association has two **roles** that may be named with a label.



Multiplicities indicate how many objects may participate in a relationship.

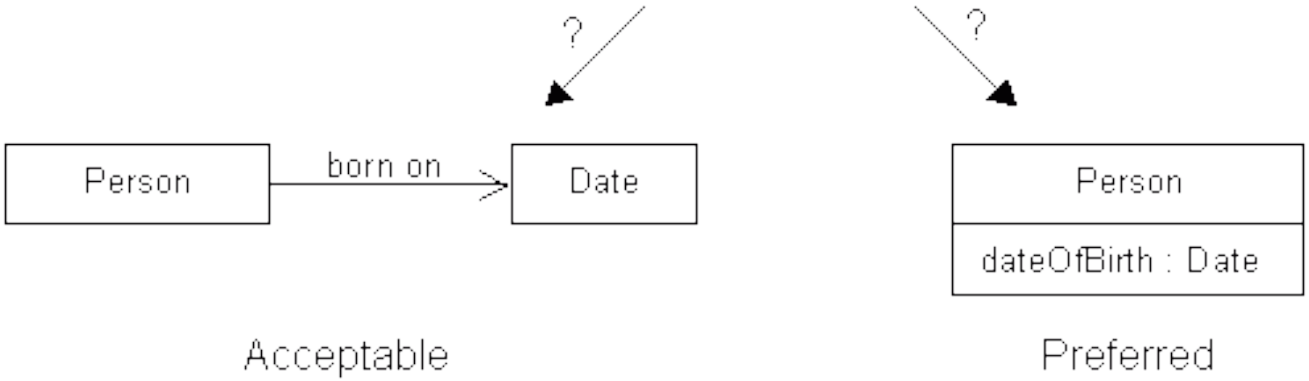
"A person is employed by a (exactly one) company."

"A company may employ many persons."



Associations vs. Attributes

```
class Person {
    private dateOfBirth Date;
    . . .
}
```



Acceptable

Preferred

Associations: Multiplicities

The * represents the range 0..Infinity.

The 1 stands for 1..1.

“An order must have been placed by exactly one customer.”

For more general multiplicities you can have

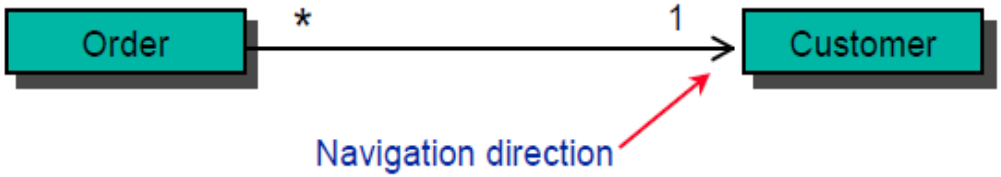
a **single** number like 11 soccer players,

a **range**, for example, 2..4 players for a canasta game,

a discrete **set** of numbers like 2,4 for doors in a car.

Navigability

To indicate navigability with associations, arrows are added to the lines.



In **a specification view**, this would indicate that an order has a responsibility to tell which customer it is for, but a customer has no corresponding ability to tell you which orders it has.

In **an implementation view**, one would indicate, that order contains a pointer to customer but customer would not point to orders.

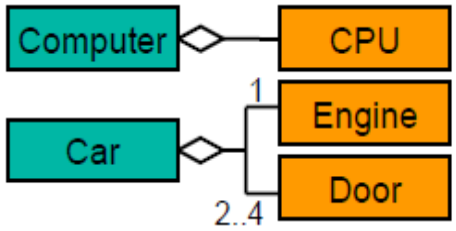
If a navigability exists in only one direction it is called **uni-directional association**, otherwise **bi-directional association**.

Aggregation

Aggregation is the part-of relationship.

“A CPU is part of a computer.”

“A car has an engine and doors as its parts.”



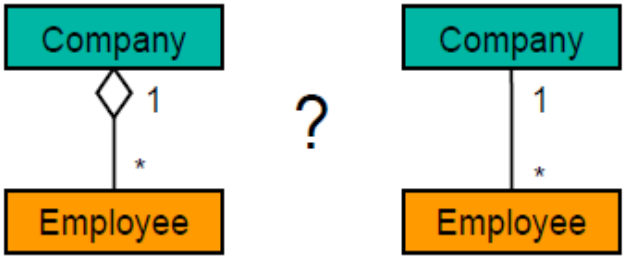
Aggregation vs. attributes :

Attributes describe properties of objects, e.g. speed, price, length.

Aggregation describe assemblies of objects.

Aggregation vs. association:

Is a company an aggregation over its employees or is it an association between its employees?



Composition

Composition is a stronger version of aggregation:

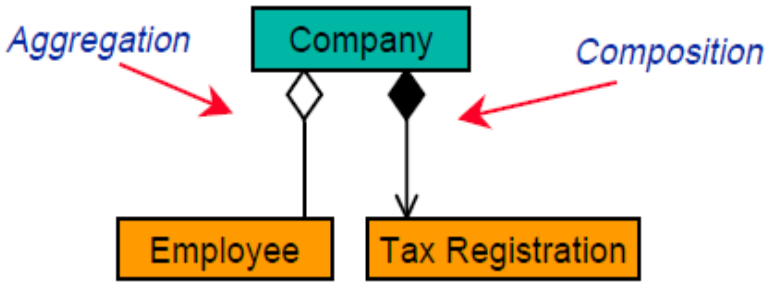
The part object may belong to only one whole.

The parts usually live and die with the whole.

Example:

Aggregation: A company has employees. The employees may change the company.

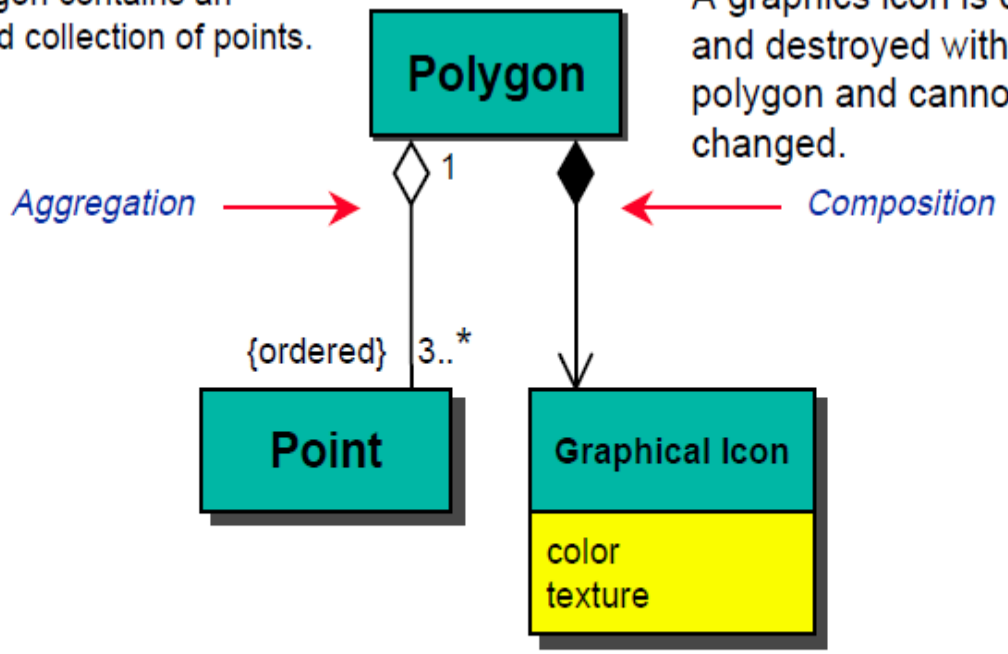
Composition: The company has a tax registration. The tax registration is tied to the company and dies with it.



Aggregation vs Composition

A polygon contains an ordered collection of points.

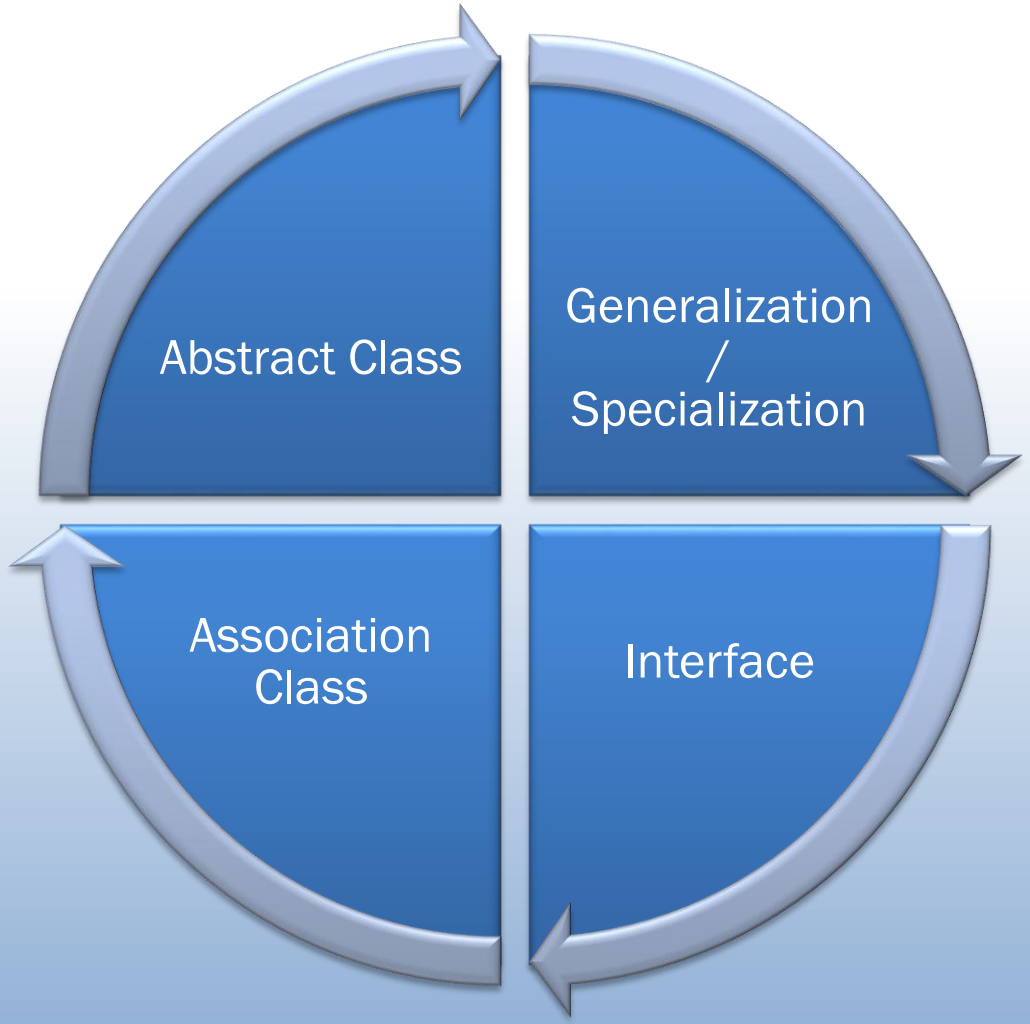
A graphics icon is created and destroyed with the polygon and cannot be changed.



These points may be changed as the polygon is edited.

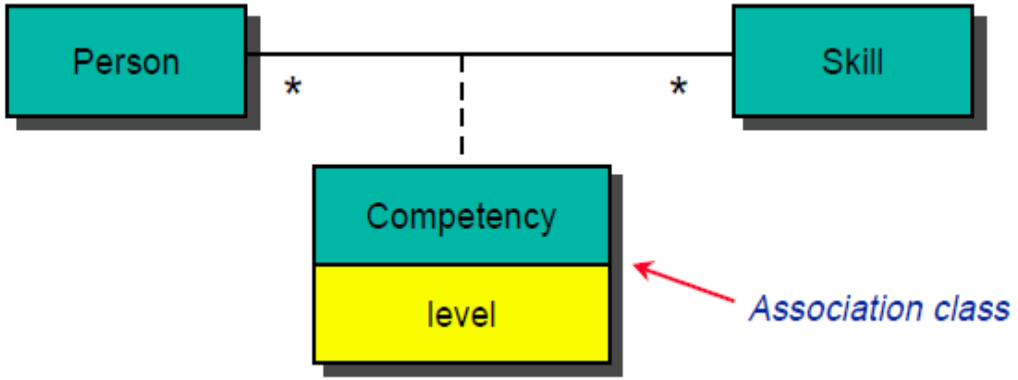
The attributes can be changed, but the icon cannot be replaced by another object.

Some Particular Types of Classes



Association Classes

Association classes allow you to model associations by classes, i.e., to **add attributes**, operations and other features to associations

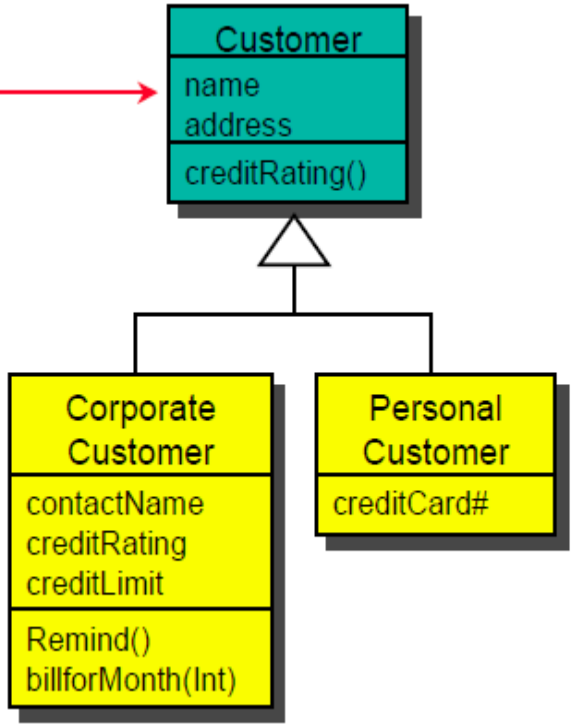


Generalization / Specialization

Generalization captures similarities between several classes in a superclass. **Specialization** refines and adds differences in subclasses.

Similarities are placed in a general superclass.

The differences are separated in specialized subclasses.



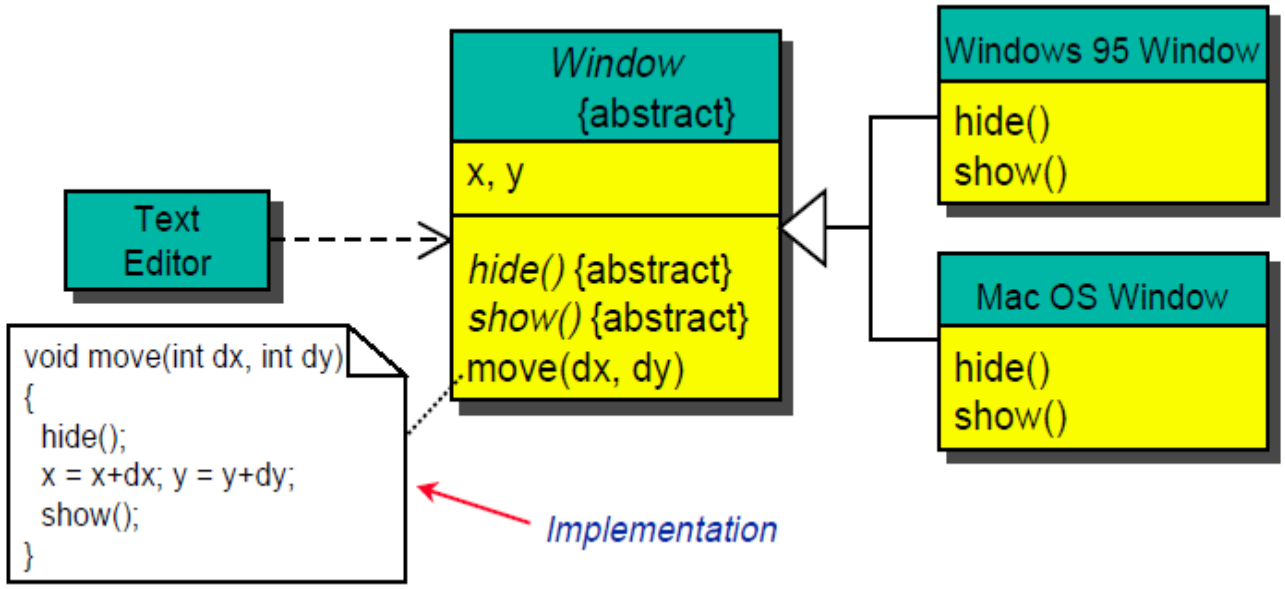
Abstract Class

An **abstract** class is a class without a **(full) implementation**.

Some **methods** are **deferred**, i.e., they are not implemented.

The deferred methods are implemented by **subclasses** only.

Example: The window move operation is implemented by using hide and show methods which are implemented by subclasses.



Interface

An **interface** is a (abstract) class with **no implementation.**

An interface is implemented (refined) by (different) classes.

The implementation can be changed without changing the clients.

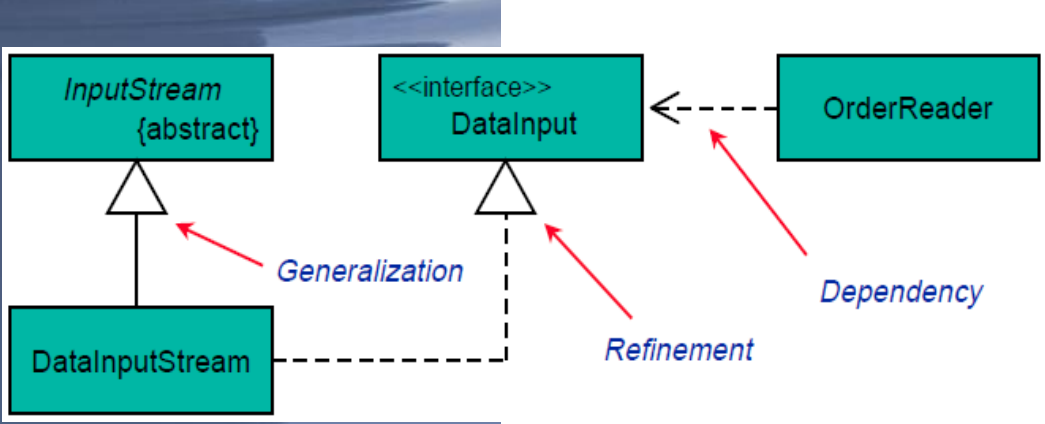
Example from Java class libraries:

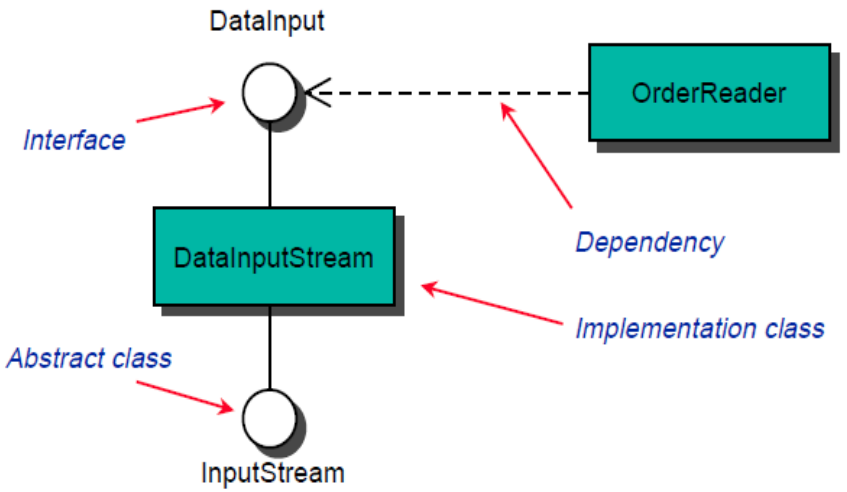
InputStream is an abstract class, i.e., some methods are deferred.

DataInput is an interface, i.e., it implements no methods.

DataInputStream is a subclass of InputStream; it implements the deferred methods of InputStream, and the methods of the interface DataInput.

OrderReader uses only those methods of DataInputStream that are defined by the interface DataInput.





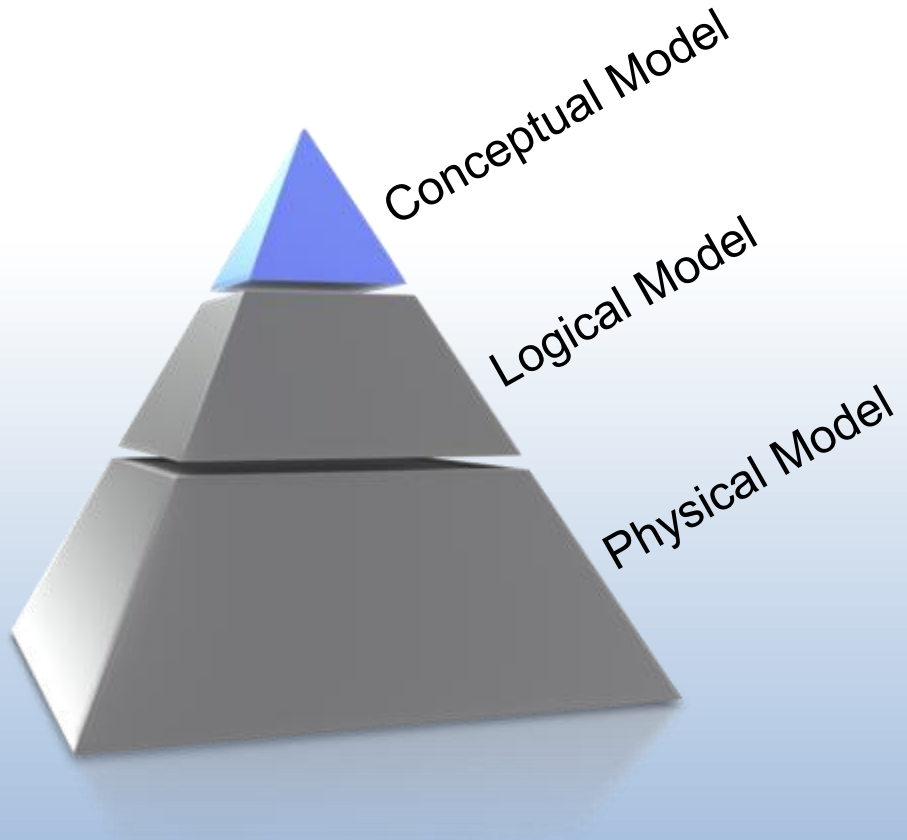
Part III

WHEN TO USE CLASS DIAGRAMS

Perspectives

	Analysis	Design
Database	Conceptual Model	
Software Component		Specification Model Implementation Model

Database Analysis & Design



Conceptual Model

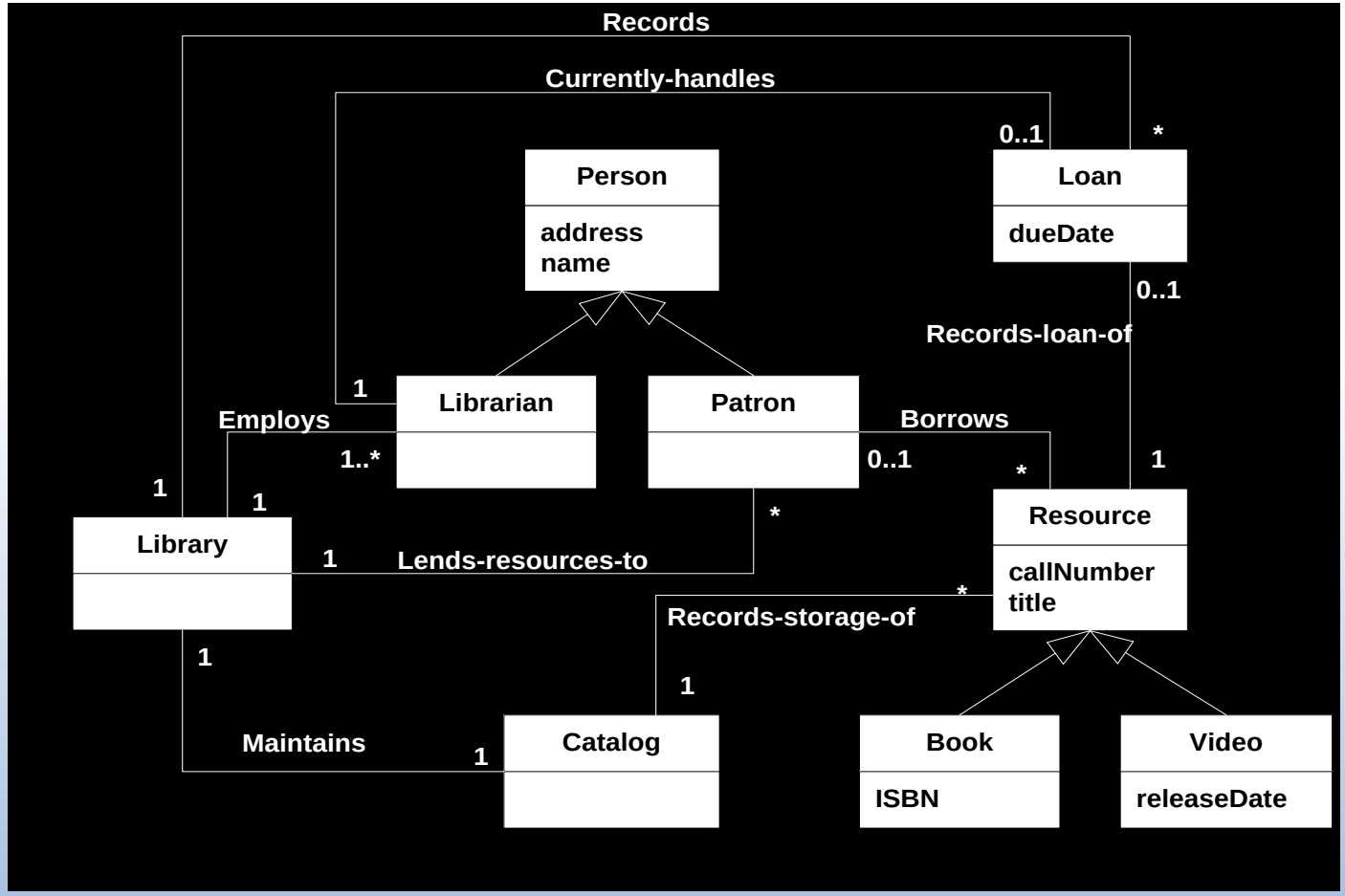
Conceptual model is used to describe a domain of concepts

A detailed model that captures the overall structure of data in an organization

Independent of any database management system (DBMS) or other implementation considerations

In conceptual model, Entity Relationship Diagram is replaced by Class Diagram

Conceptual Model Example



Logical Model

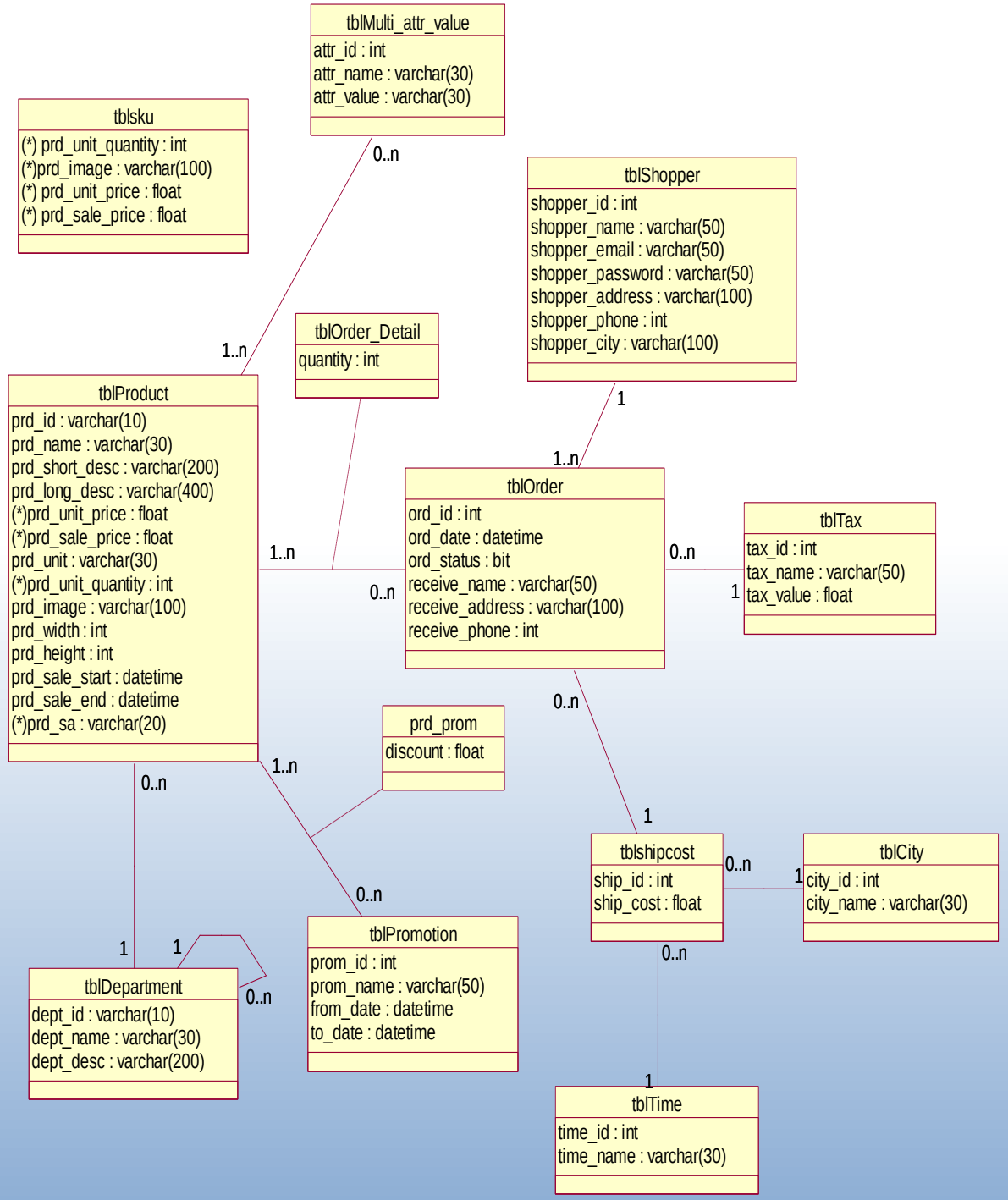
Based upon the Conceptual Model

Logical Model is independent with DBMSs.

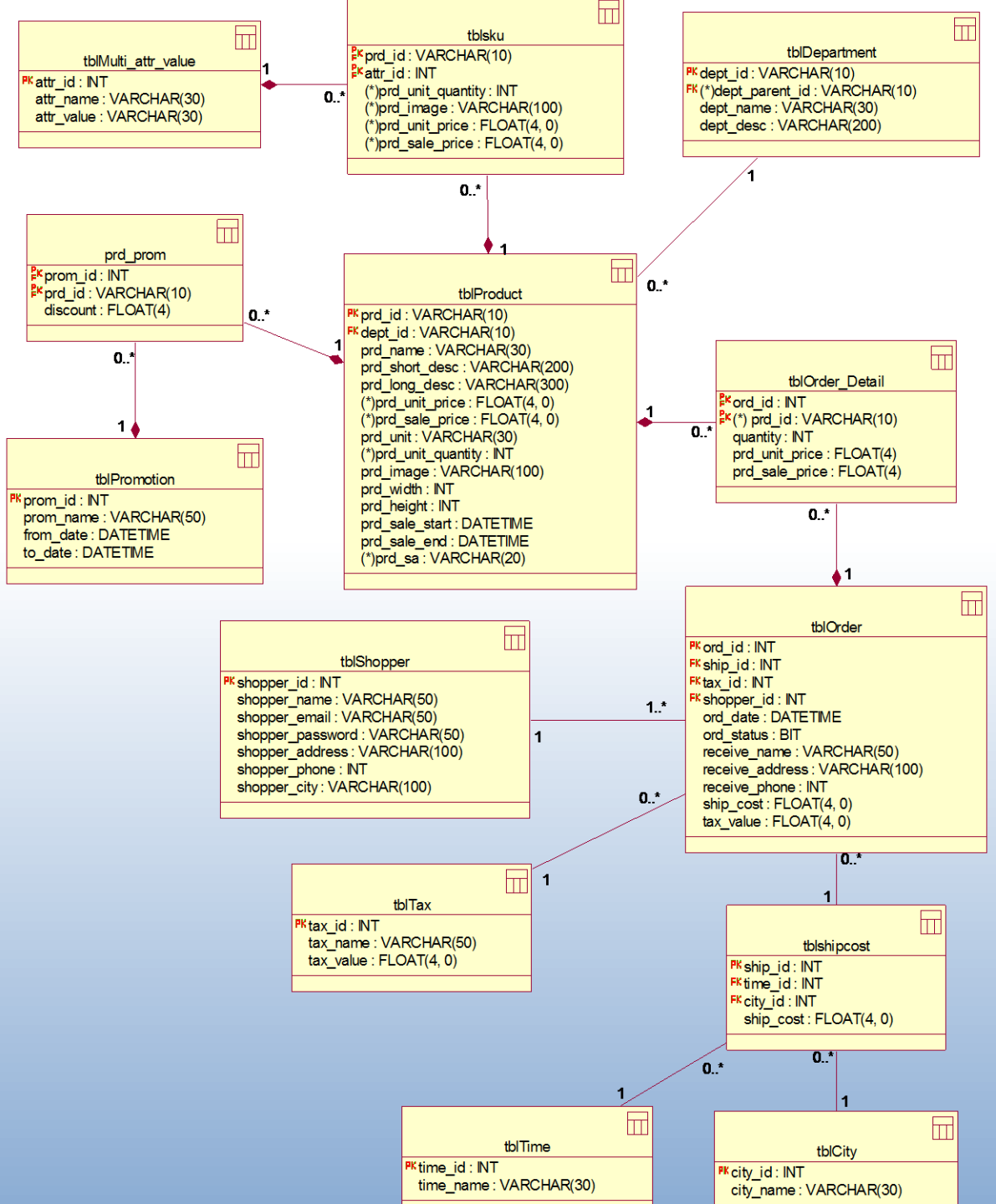
In Logical Model, there are some foreign keys.

A Logical Model is a Relational Database.

Conceptual Model



Logical Model



Physical Model

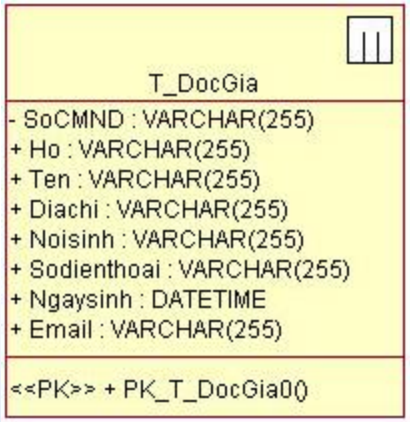
Based upon the logical model.

Physical Model depends on a particular DBMS.

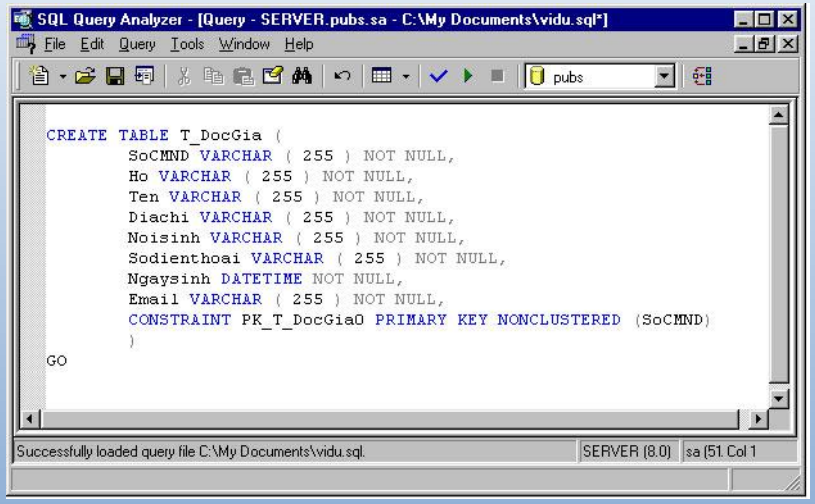
A Physical Model is a SQL script file.

Example

Conceptual Model



Logical Model



Physical Model

Notes

1. A conceptual model is not a picture of Software components or Classes in an object-oriented programming language. It illustrates real-world concepts.

2. Each many-to-many relationship have to be modeled with an association class.

3. Conceptual model need to get the third normal form (3NF).

Normalizations

The process of converting complex data structures into simple, stable data structures

First Normal Form (1NF) _ Unique rows, No multivalued attributes, All relations are in 1NF.

Second Normal Form (2NF) _ Each nonprimary key attribute is identified by the whole key (called full functional dependency).

Third Normal Form (3NF) _ Nonprimary key attributes do not depend on each other (i.e. no transitive dependencies).

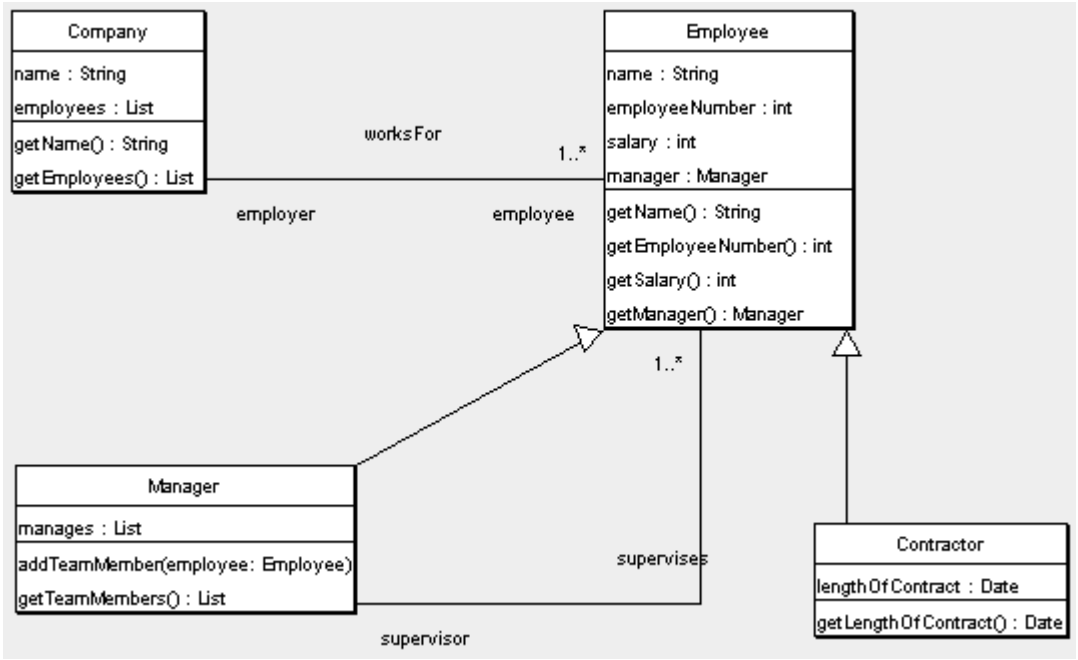
Specification Model vs Implementation Model

Specification Model and Implementation Model are used to describe software components.

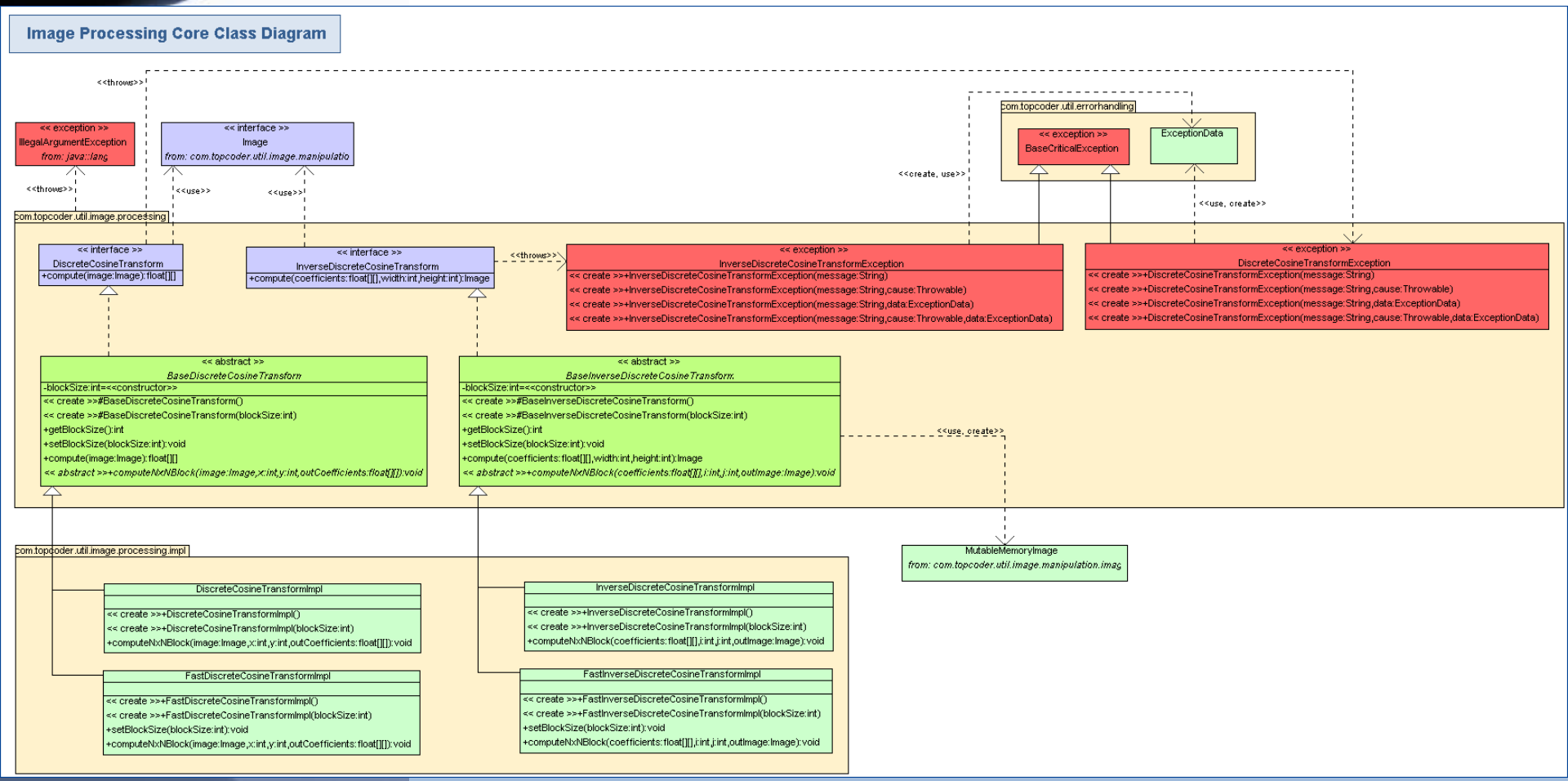
Specification Model describes software abstractions with specifications and interfaces. *It's independent with programming languages.*

Implementation Model describes software implementations in a particular programming language.

Specification Model Example



Implementation Model Example



Sequence Diagrams vs Class Diagrams

```
public class M {

  public static void main(String[] args) {
    M m = new M();
    m.f();
  }
  public void f() {
    Subject s = new Subject();
    Observer o1 = new Observer();
    Observer o2 = new Observer();
    s.addObserver(o1);
    s.addObserver(o2);

    s.changeState();
  }
}
```

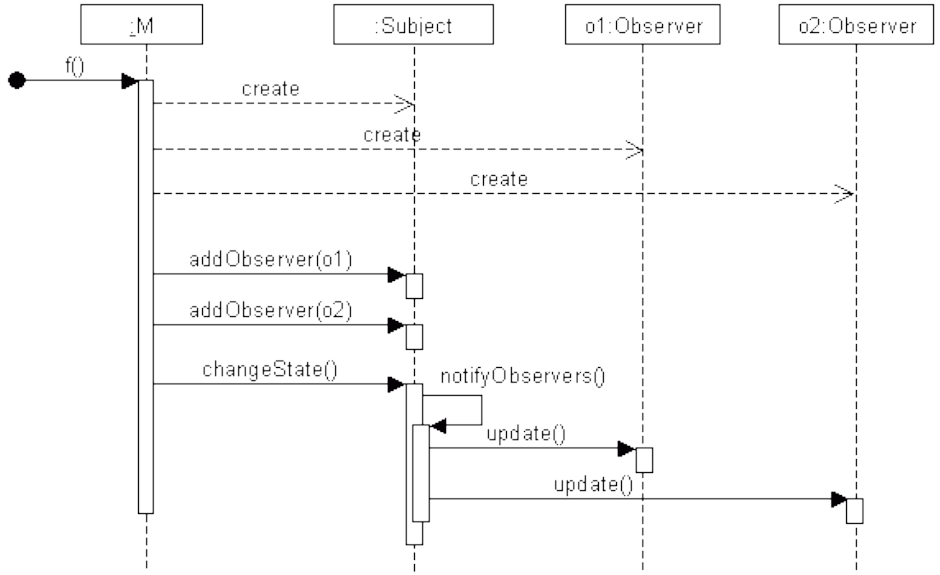
```
class Observer {
    public void update()()
}
```

```
class Subject {
  private Collection c;

  public void addObserver(Observer o) {
    c.add(o);
  }

  public void changeState() {
    // Change state of subject
    notifyObservers();
  }

  public void notifyObservers() {
    Iterator i = c.iterator();
    while (i.hasNext()) {
      Observer o = (Observer)i.next();
      o.update();
    }
  }
}
```



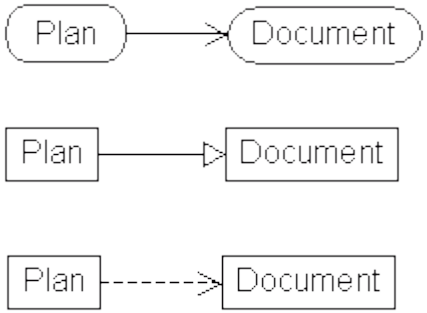
Exercise #1

Match the UML model with the appropriate interpretation.

___ A plan is dependent on a document.

___ Document first, then plan.

___ A plan is a special type of document.



Exercise #2

Build relationships between these classes: Person, Car-Owner, Car

Exercise #3

Build relationships between these classes: Employee, Supervisor, Administrator, Engineer, Permanent, Temporary. If required, you can add a new class.

Exercise #4

Create an UML class diagram that models the data relationships described in the following paragraph.

To be a collector you have to have one or more collections. Each collection must have 2 or more items. Each collection belongs to one collector. A collection is made up of items owned. A particular item may be in more than one collection (i.e. an old Coke sign may be in both a Coke memorabilia collection and a sign collection.)

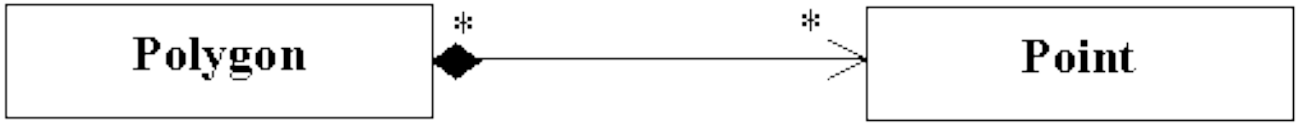
Exercise #5

Use the following class diagram to answer the questions that following it. If there isn't enough information in the diagram to answer the question, state as much

1. Can you have a company without any employees?
2. Can a person be unemployed?
3. Can a person be self-employed?

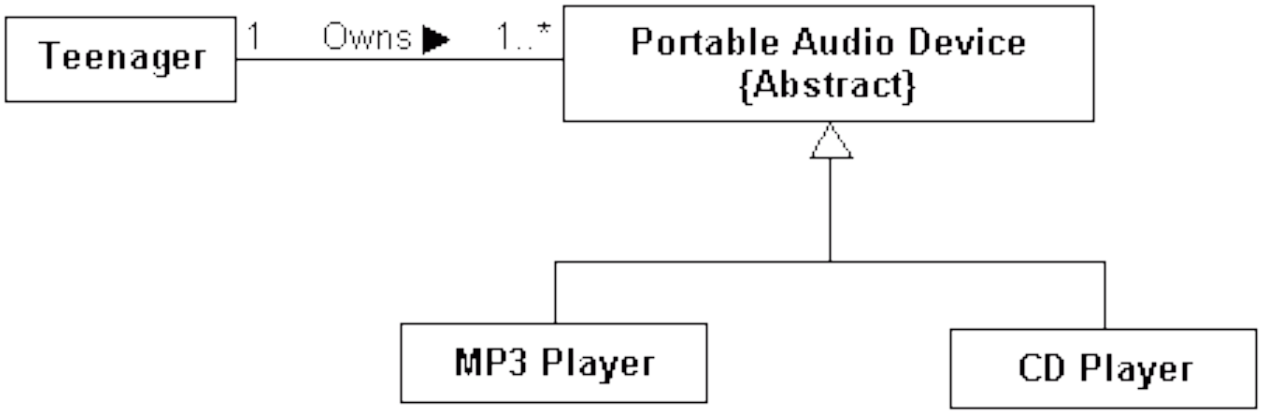
Exercise #6

What, if anything, is wrong with the following diagram:



Exercise #7

Use the following figure to answer the questions that follow.



a. How many portable audio devices does each teenager own?

b. Can there be a portable audio device not owned by any teenager?

c. Can there be an MP3 player player not owned by any teenager?

Exercise #8

What, if anything, is wrong with the following model?

Mark each of the statements below true or false, according to the diagram above

a. (True / False) A player must play for either an NBA team or an NFL team but can't play for both at the same time.

b. (True / False) A player can play for two NBA or NFL teams simultaneously.

c. (True / False) A player can be traded among teams.

d. (True / False) A player must always be a part of a one team.

Exercise #9

Based on the following description, draw a class diagram of an information system for a company called North Marina.

The company offers docking spots for the boats. Boat owners rent docking spots based on an annual agreement for each boat, which can be renewed. An owner can have several boats, however each boat is assigned a unique docking spot stated in the agreement.

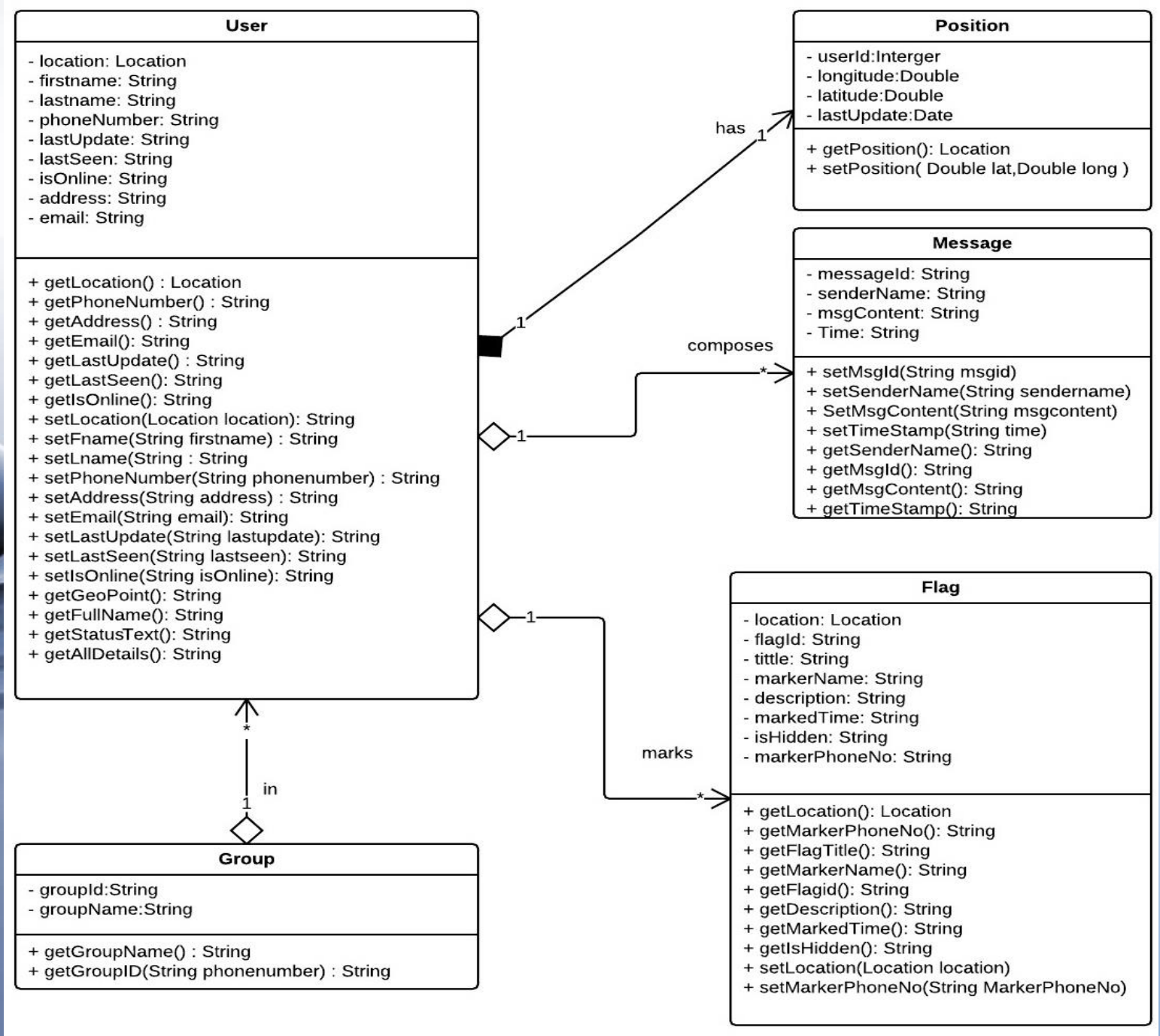
North Marina also offers boat repair service which requires an owner to sign a contract with a company representative responsible for repair. In your class diagram, model the relationships between the classes: DOCKING SPOT, BOAT, AGREEMENT, OWNER, CONTRACT and REPRESENTATIVE.

a. Indicate minimum and maximum cardinalities and degree of each relationship;

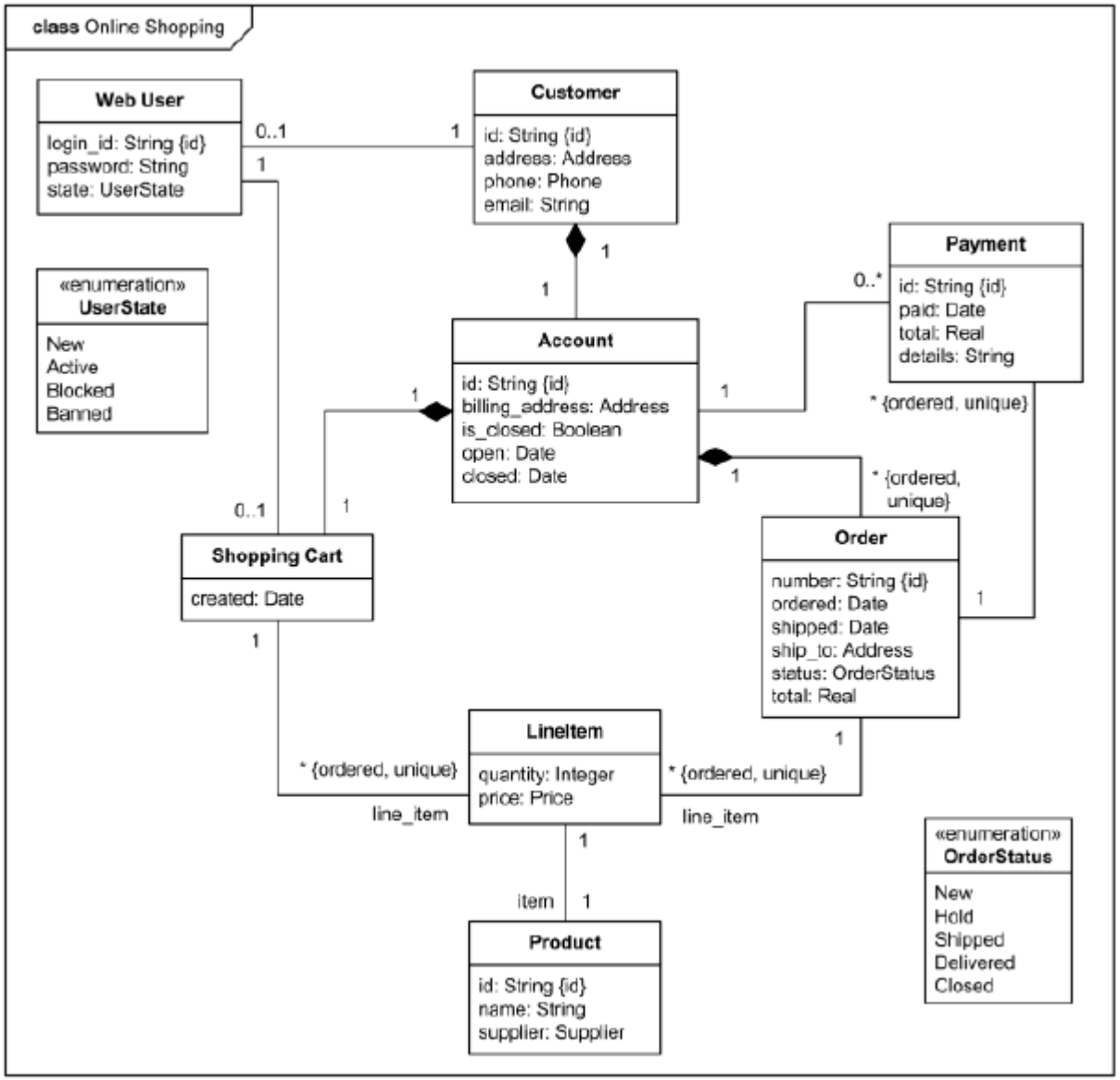
b. Show the attributes of each class;

c. If required, present a solution to avoid many-to-many relationships.

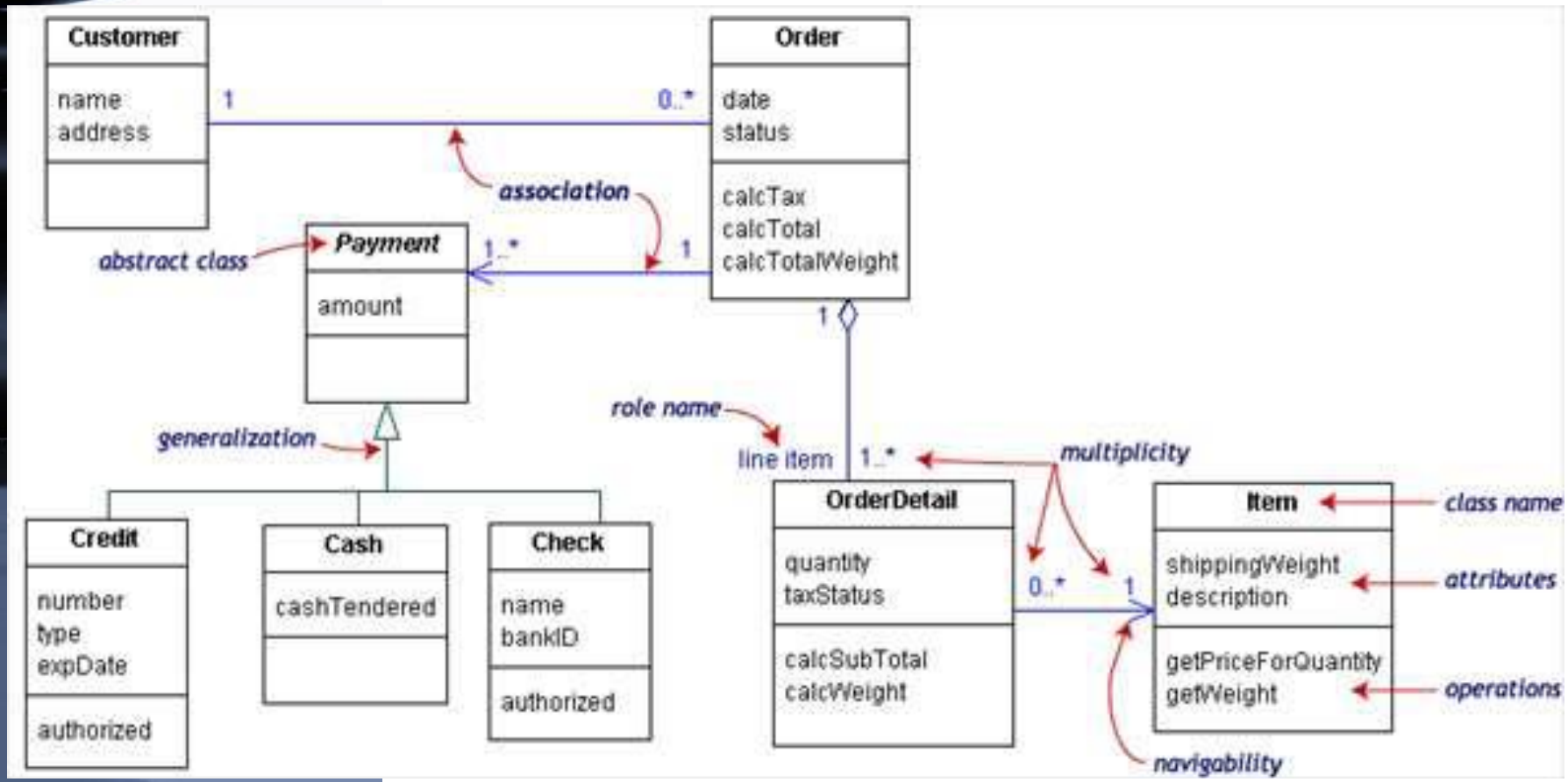
Ex. #10


User
- location: Location
- firstname: String
- lastname: String
- phoneNumber: String
- lastUpdate: String
- lastSeen: String
- isOnline: String
- address: String
- email: String
+ getLocation() : Location
+ getPhoneNumber() : String
+ getAddress() : String
+ getEmail(): String
+ getLastUpdate() : String
+ getLastSeen(): String
+ getIsOnline(): String
+ setLocation(Location location): String
+ setFname(String firstname) : String
+ setLname(String : String
+ setPhoneNumber(String phonenumber) : String
+ setAddress(String address) : String
+ setEmail(String email): String
+ setLastUpdate(String lastupdate): String
+ setLastSeen(String lastseen): String
+ setIsOnline(String isOnline): String
+ getGeoPoint(): String
+ getFullName(): String
+ getStatusText(): String
+ getAllDetails(): String
Position
- userId:Interger
- longitude:Double
- latitude:Double
- lastUpdate:Date
+ getPosition(): Location
+ setPosition(Double lat,Double long)
has
1
1
Message
- messageId: String
- senderName: String
- msgContent: String
- Time: String
+ setMsgId(String msgid)
+ setSenderName(String sendername)
+ SetMsgContent(String msgcontent)
+ setTimeStamp(String time)
+ getSenderName(): String
+ getMsgId(): String
+ getMsgContent(): String
+ getTimeStamp(): String
composes
*
1
1
Flag
- location: Location
- flagId: String
- tittle: String
- markerName: String
- description: String
- markedTime: String
- isHidden: String
- markerPhoneNo: String
+ getLocation(): Location
+ getMarkerPhoneNo(): String
+ getFlagTitle(): String
+ getMarkerName(): String
+ getFlagid(): String
+ getDescription(): String
+ getMarkedTime(): String
+ getIsHidden(): String
+ setLocation(Location location)
+ setMarkerPhoneNo(String MarkerPhoneNo)
marks
*
1
*
in
1
Group
- groupId:String
- groupName:String
+ getGroupName() : String
+ getGroupID(String phonenumber) : String

Exercise #11


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..*
* {ordered, unique}
Account
id: String {id}
billing_address: Address
is_closed: Boolean
open: Date
closed: Date
1
1
1
* {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
item
1
Product
id: String {id}
name: String
supplier: Supplier
«enumeration»
OrderStatus
New
Hold
Shipped
Delivered
Closed

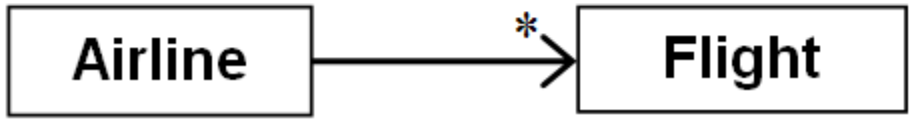
Exercise #12


Customer
name
address
1
0..*
Order
date
status
calcTax
calcTotal
calcTotalWeight
association
abstract class
Payment
amount
1..*
1
1
role name
line item
1..*
multiplicity
generalization
Credit
number
type
expDate
authorized
Cash
cashTendered
Check
name
bankID
authorized
OrderDetail
quantity
taxStatus
calcSubTotal
calcWeight
0..*
1
Item
shippingWeight
description
getPriceForQuantity
getWeight
class name
attributes
operations
navigability

Exercise #13

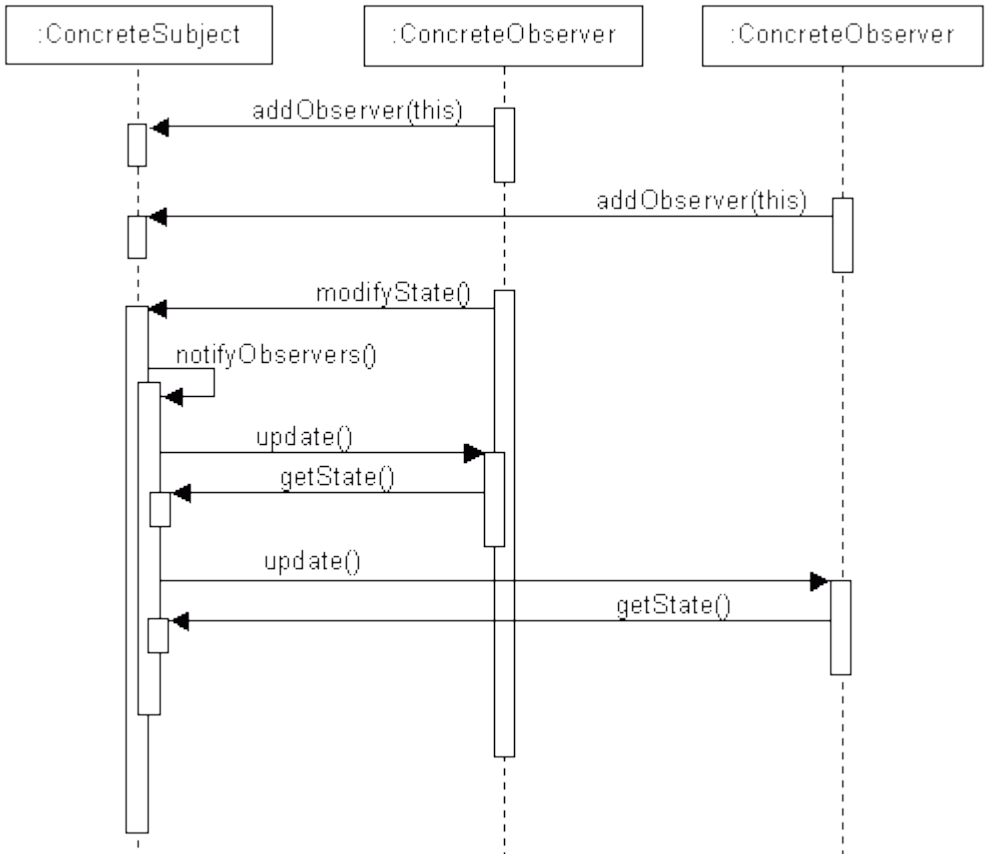
Is the following class diagram valid for the given code fragment? Explain. If invalid, show a valid class diagram.

```
class Airline {
   Flight [] getFlights() { . . . }
}
```



Exercise #14

Write the code fragment that corresponds to the following sequence diagram.



Exercise #15

Draw an UML class diagram to express the structural relationships in the following program and draw an UML sequence diagram to express the dynamic behavior.

```
import java.util.Vector;

public class Driver {
    private StringContainer b = null;

    public static void main(String[]
args){
        Driver d = new Driver();
        d.run();
    }

    public void run() {
        b = new StringContainer();
        b.add("One");
        b.add("Two");
        b.remove("One");
    }
}
```

```
class StringContainer {
    private Vector v = null;

    public void add(String s) {
        init();
        v.add(s);
    }

    public boolean remove(String s) {
        init();
        return v.remove(s);
    }

    private void init() {
        if (v == null)
            v = new Vector();
    }
}
```

Any Questions?

✉ hvusynh@hcmiu.edu.vn