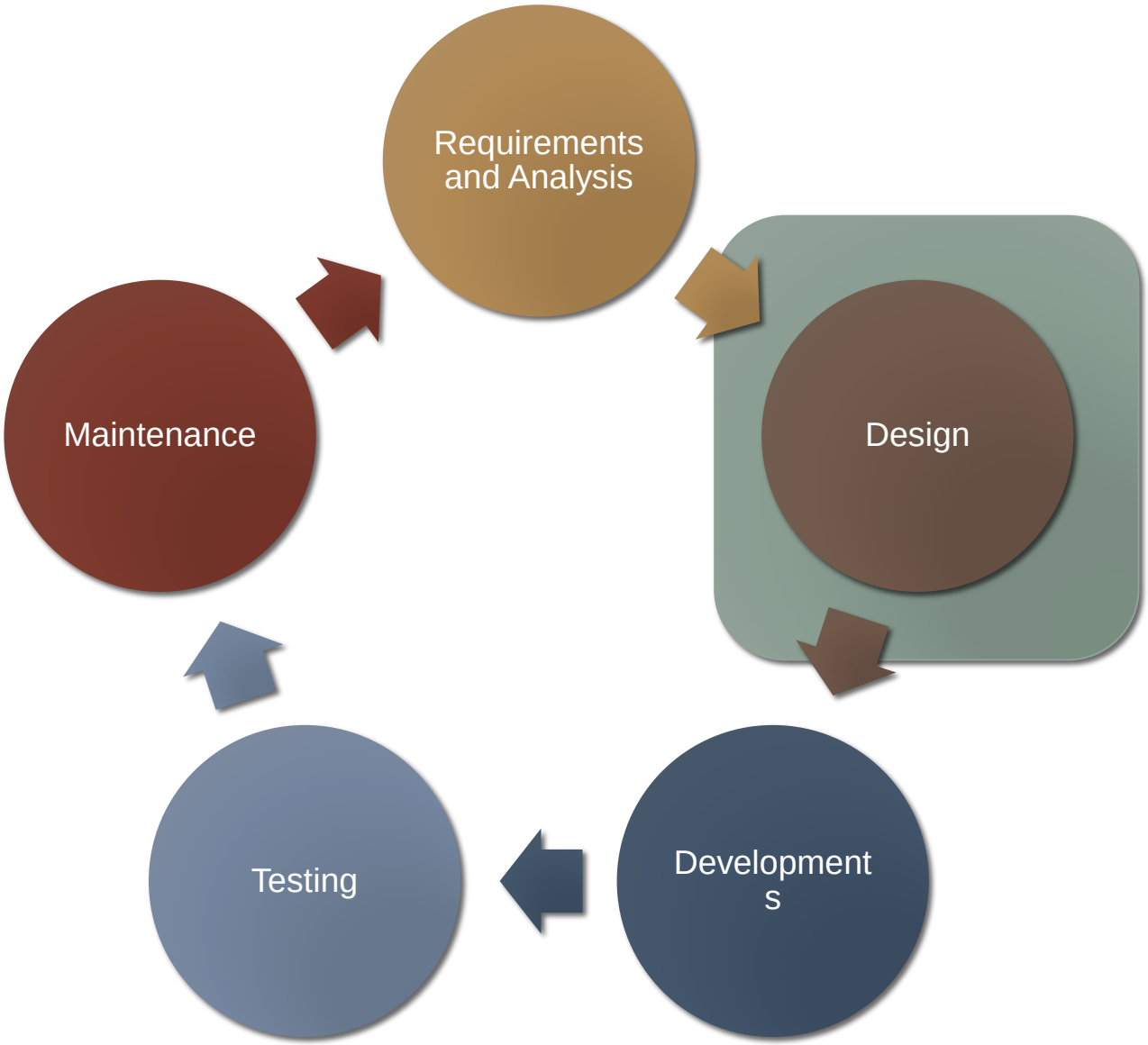
# (11224) INTRODUCTION TO SOFTWARE ENGINEERING

**Lecture 6:** The Unified Modeling Language (part 1)

Shereen Fouad

# Software Processes



# Design phase

- specifying the structure of how a software system will be written and function, without actually writing the complete implementation

- a transition from "what" the system must do, to "how" the system will do it
  - What classes will we need to implement a system that meets our requirements?
  - What fields and methods will each class have?
  - How will the classes interact with each other?

# What is Unified Modeling Language (UML)?

- Visual language intended to capture both structural and behavioural aspects of a software system design.
- Its main purpose is to assist software engineers in designing their systems by providing a more abstract language than source code
- UML can be used to describe:
  - the organization of a program
  - how a program executes
  - how a program is used
  - how a program is deployed over a network
  - …and more

# UML is complex

- UML is a big, complicated diagramming language
- Different demands on such a unified language, have resulted in a large, complex, specified standard that has at least 14 different diagram types.
- 7 describe static structure of systems, and 7 describe behaviour.

# Overview of UML Diagrams

## Structural

: element of spec. irrespective of time

- Class
- Component
- Deployment
- Object
- Composite structure
- Package

## Behavioral

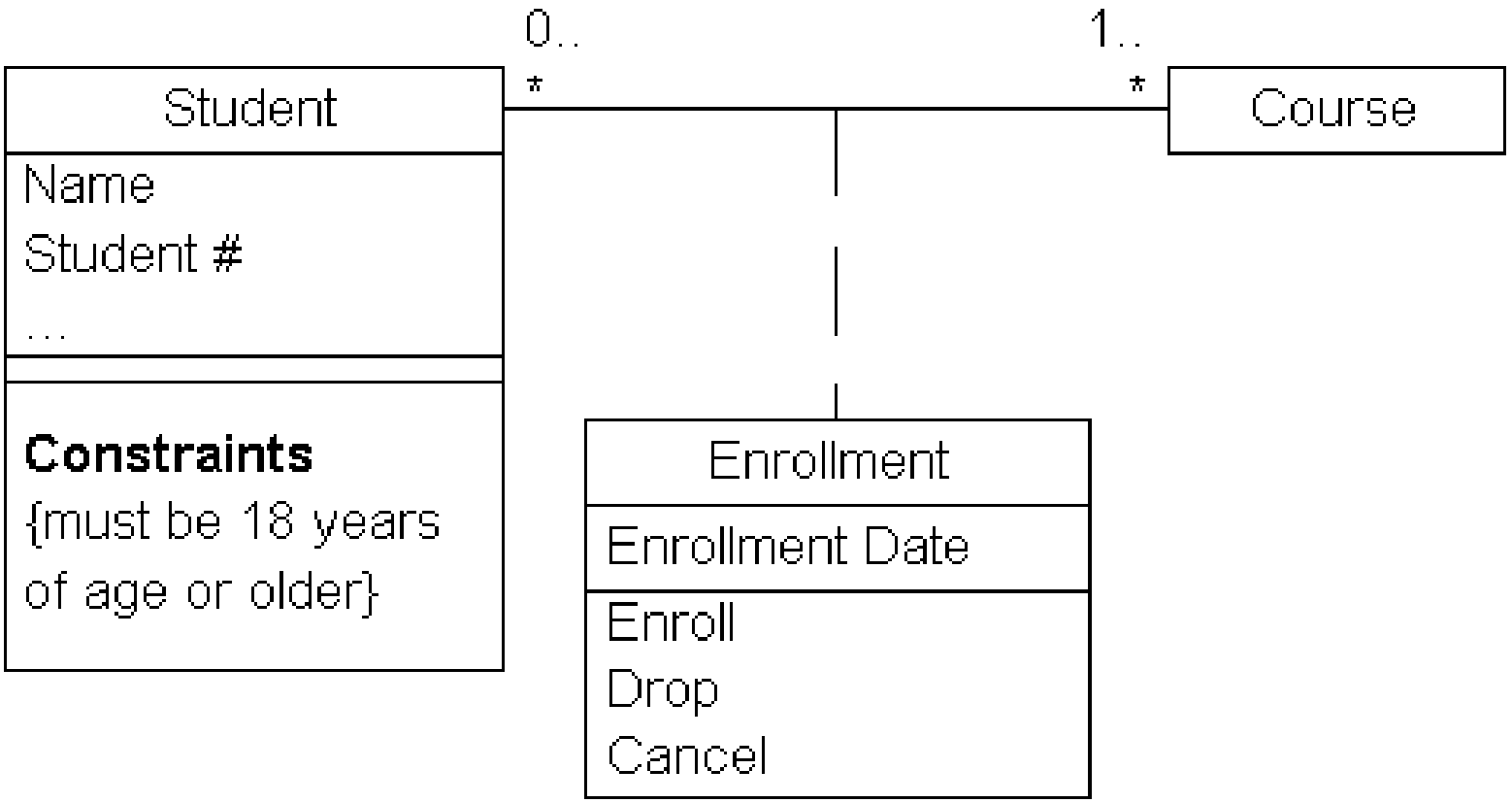
: behavioral features of a system / business process

- Activity
- State machine
- Use case
- Interaction

# Examples of UML Structure Diagrams

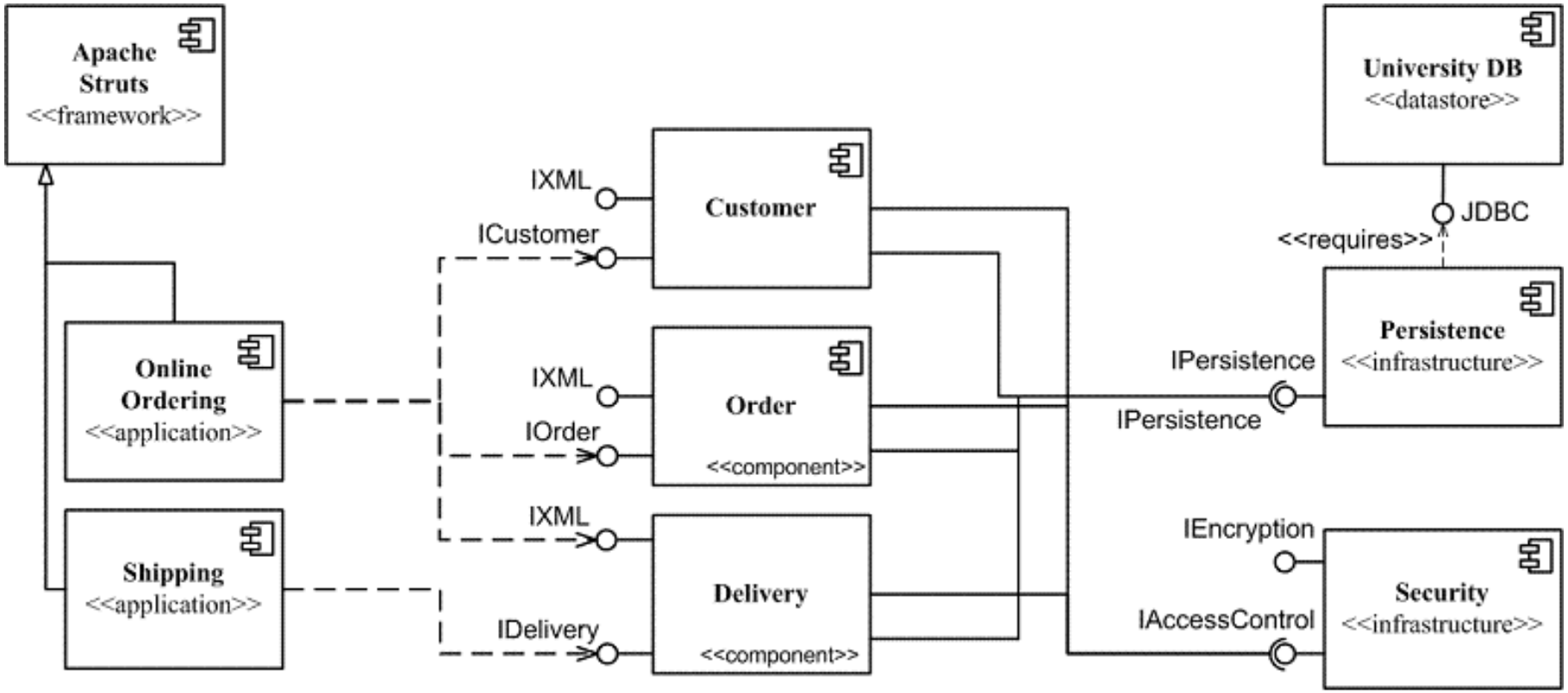
# Class diagram (today)

- The most common diagram. It identifies the system's classes and the relationships between them.



# Component diagram

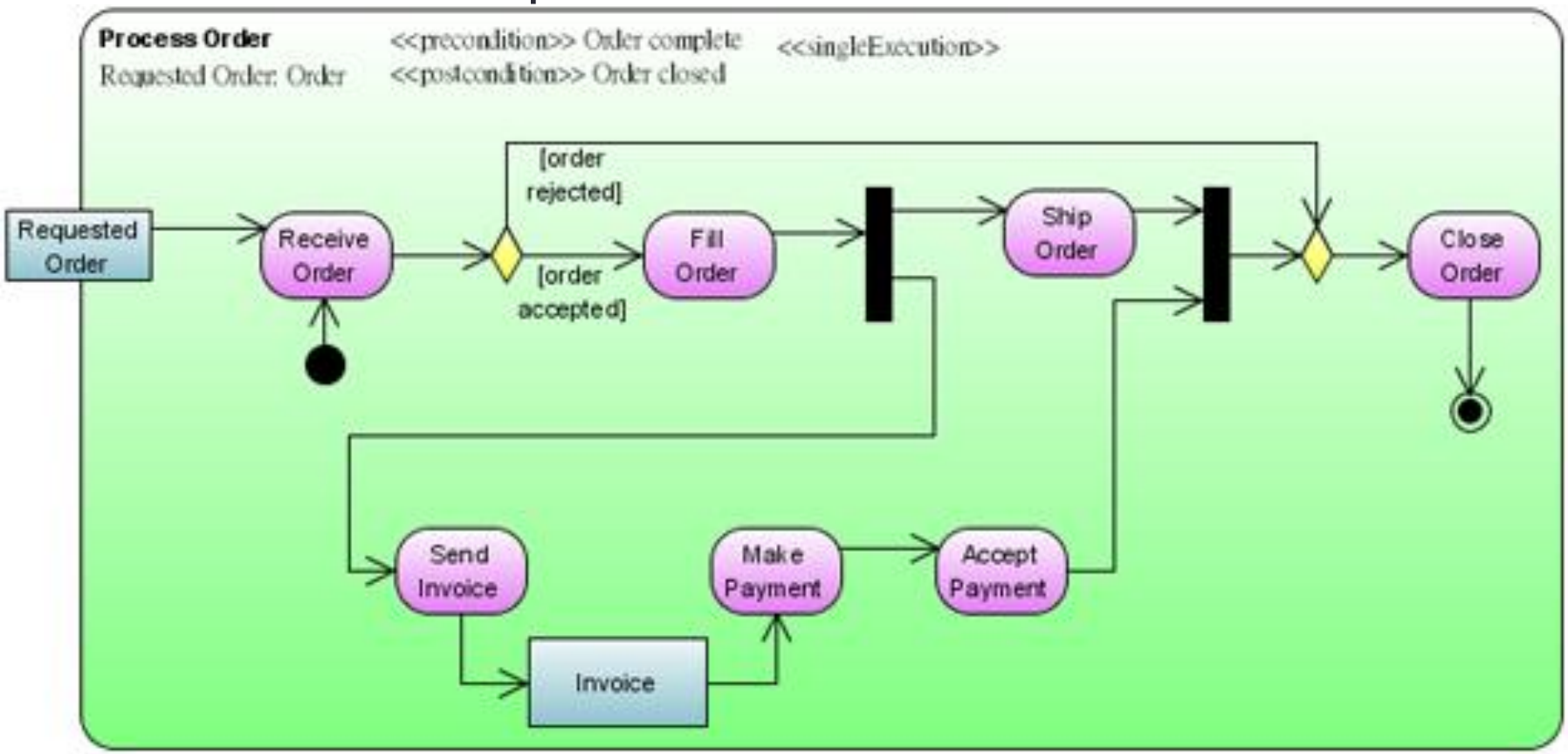
- This shows how a system is decomposed into reusable software components and the relationships between them.



# Examples of UML Behaviour Diagrams

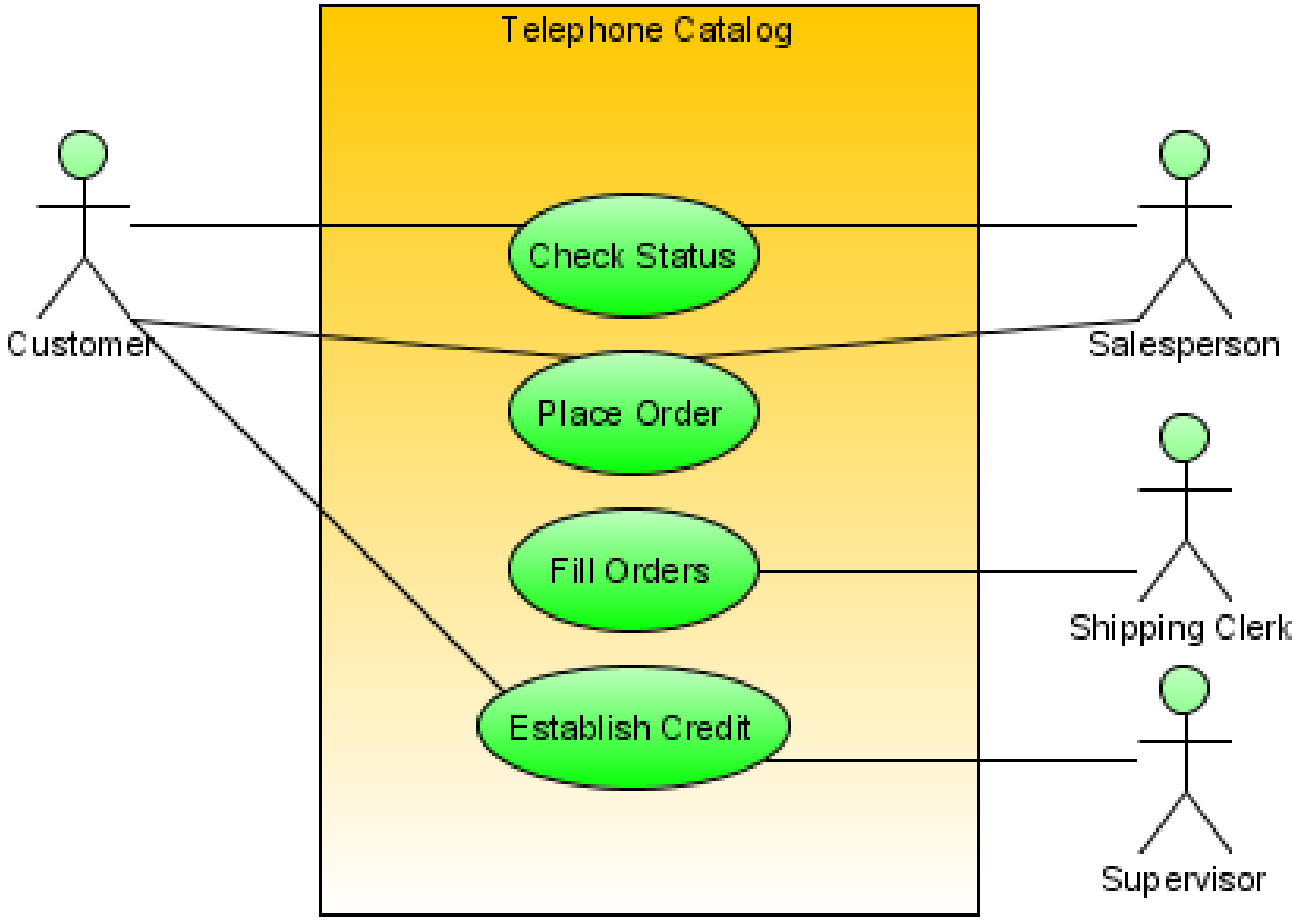
# Activity diagram

This is used to describe sequential and parallel flow of control in a system, such as the exploring complex business rules and operations, describing the use case also the business process.



# Use cases diagram

- This presents an overview, without the textual detail, of the use cases in a system and their interdependencies.



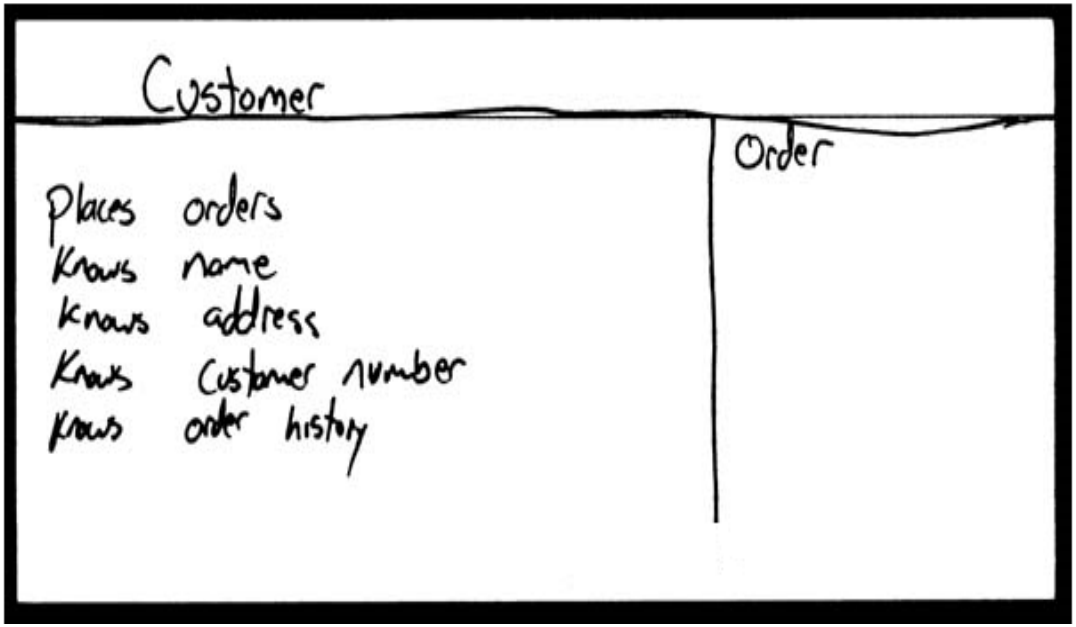
This lecture will cover only one kind of UML diagram, the class diagram

# UML Class Diagrams

- UML Class diagrams represent the structure of the system
- Shows a picture of the classes in an OO system,
  - classes
  - Attributes
  - operations (or methods),
  - Relationships among the classes.
- UML diagrams have a precise syntax just like programming languages

# How do we design classes?

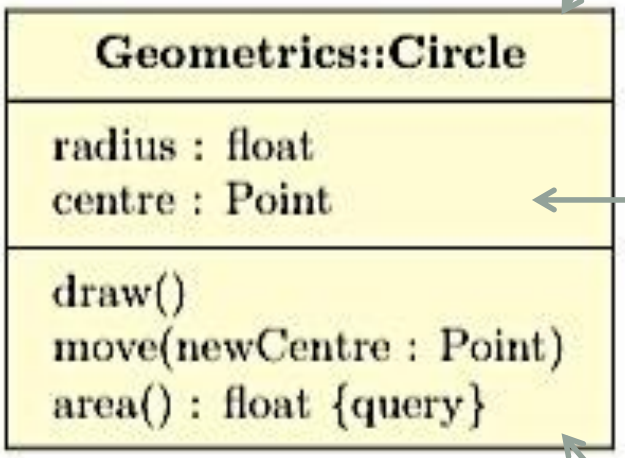
- class identification from project spec / requirements
  - nouns are potential classes, objects, fields
  - verbs are potential methods or responsibilities of a class



# Classes

- The simplest form allowed of a class in a class diagram is to write the class name inside a box.
- The class name must begin with an uppercase character:

# Classes




- Class name,

(Mandatory)

- Attributes (corresponding to member fields in Java),

(Optional)

- Operations (methods in Java), (Optional)

# Attributes I

- A variable is written as:

  *visibility  name  :  type*

  where:
  - +  means public visibility
  - #  means protected visibility
  - -  means private visibility
  - <*blank*> means default (package) visibility
- Example:  +length:int

# Attributes II

- Static variables are underlined
- An initial value can be shown with =*value*
- Example:

- **What do I mean by?**

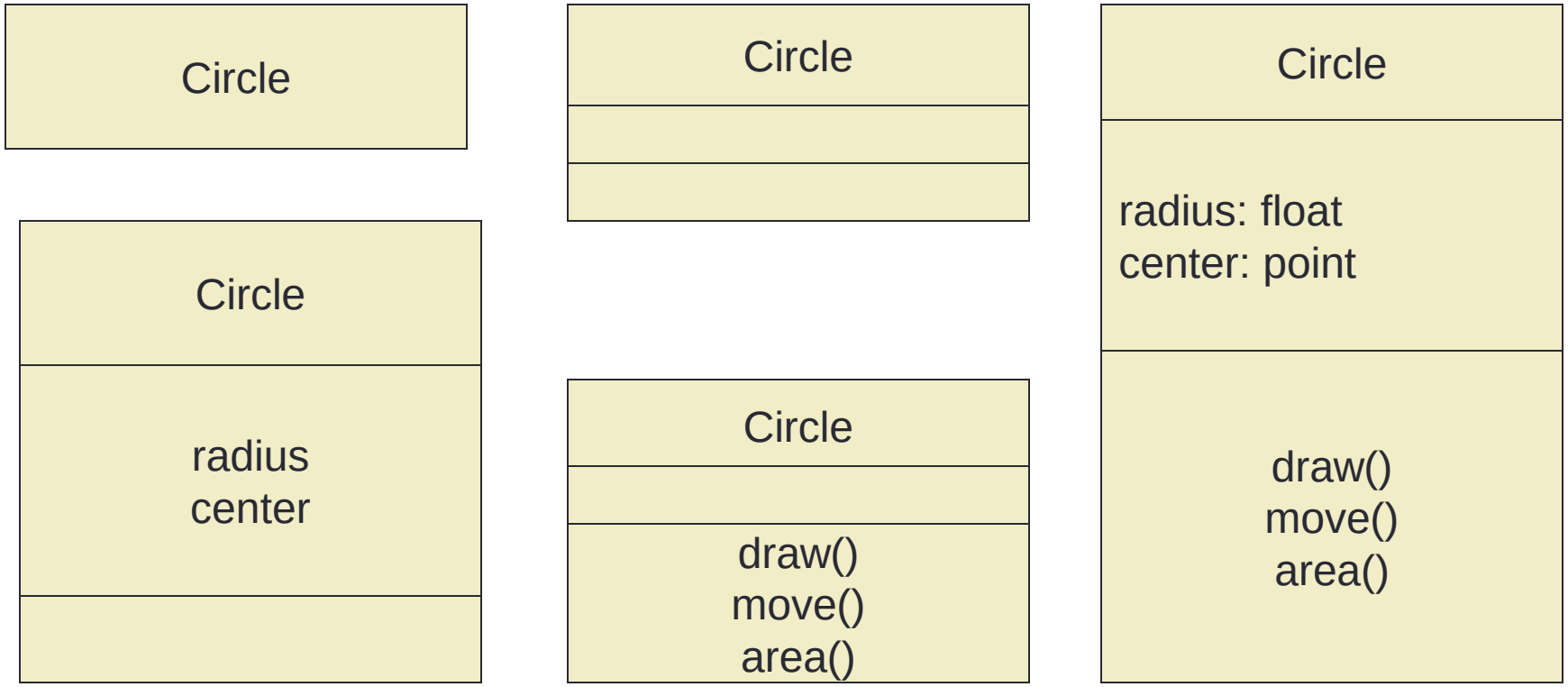
- -numberOfEmployees:int=10

# Methods

- Methods are written as:
  *visibility name (parameters) : returnType*
  where
  - visibility is the same as variables (+, -, #, blank)
  - parameters are given as *name:type*
  - if the returnType is void, it is omitted
  - constructors are preceded by «constructor»
- may omit trivial (get/set) methods
- but don't omit any methods from an interface!
- should not include inherited methods

# Depicting Classes

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




Classes cannot be drawn with only two parts, or with four or more parts.

# Associations

- Associations denote relationships between classes
- Identify which classes reference which other classes.
- They are drawn as a solid line between the two classes.

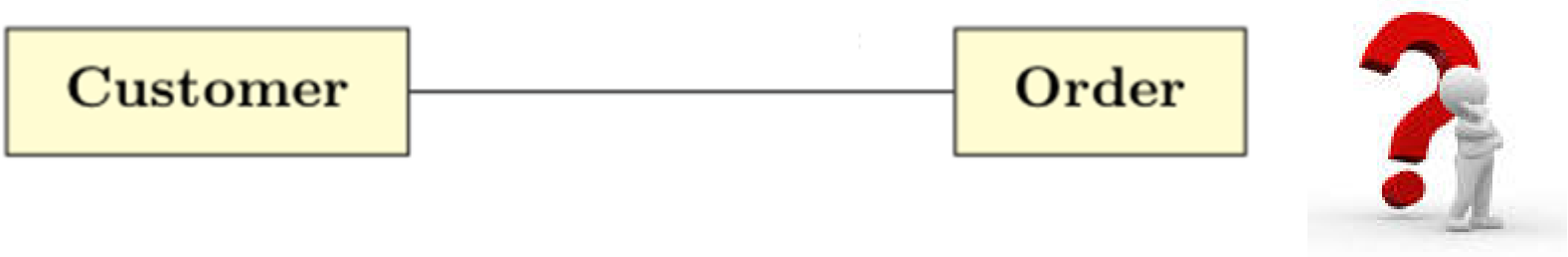

- Indicates that objects at one end of an association "recognize" objects at the other end and may send messages to them.

# Cardinality of Association

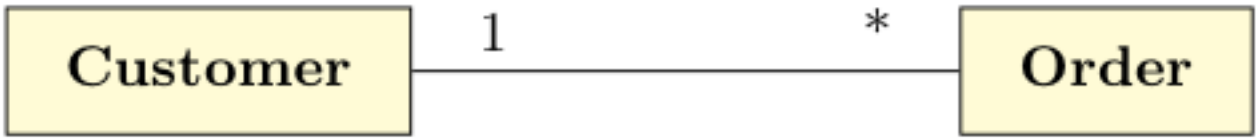
- Deciding how many objects of one class is connected to how many of another.
- Can you provide a verbal expression of the relationship between a customer and an order?

# Cardinality of Association

- Deciding how many objects of one class is connected to how many of another.
- Can you provide a verbal expression of the relationship between a customer and an order?




- This is called multiplicity or cardinality information.



# 1-to-1 and 1-to-many Associations


1-to-1 association


1-to-many association

# Many-to-Many Associations

# From Problem Statement To Class Diagram

- Problem Statement: A stock exchange lists many companies. Each company is uniquely identified by a ticker symbol.

- What is the association cardinality?

# “lower..upper” Cardinality of Association

- The most general form of a multiplicity specification is as a range “lower..upper” :
  - lower can be any integer greater or equal to 0
  - upper can be any integer greater than 0 or “*”, where the latter means there is no upper limit imposed.
- A number of shorthand notations are used: “1” is shorthand for “1..1”, “*” is shorthand for “0..*”.


- If no multiplicity is specified, a multiplicity of “1..1” is assumed.

# Naming of Association

- To clarify its meaning, an association may be named.
  - The name is represented as a label placed midway along the association line.
  - Usually a verb or a verb phrase.


“Customer owns Order”:

# Optional, Mandatory or Multivalued Association

- If lower multiplicity is 0 then the association is optional:
  - in this case we have to allow for the existence of objects that sometimes will be connected to another object and sometimes will not.
- If the lower multiplicity is 1 then the association is mandatory:
  - if this is the case then we have to ensure that no object of this type can exist without being connected to another object.
- If the upper multiplicity is greater than 1 then the association is multivalued:
  - in this case we have to provide a mechanism (e.g. arrays, lists, sets, etc.) to allow the object to be connected to whole collections of other objects.

# Quiz !!

- A class is divided into which of these compartments ?
- a) Name Compartment
- b) Attribute Compartment
- c) Operation Compartment
- d) All of the mentioned

- What is multiplicity for an association?
- a) The multiplicity at the target class end of an association is the number of instances that can be associated with a single instance of source class
- b) The multiplicity at the target class end of an association is the number of instances that can be associated with a number instance of source class
- c) All of the mentioned
- d) None of the mentioned

# Quiz !!

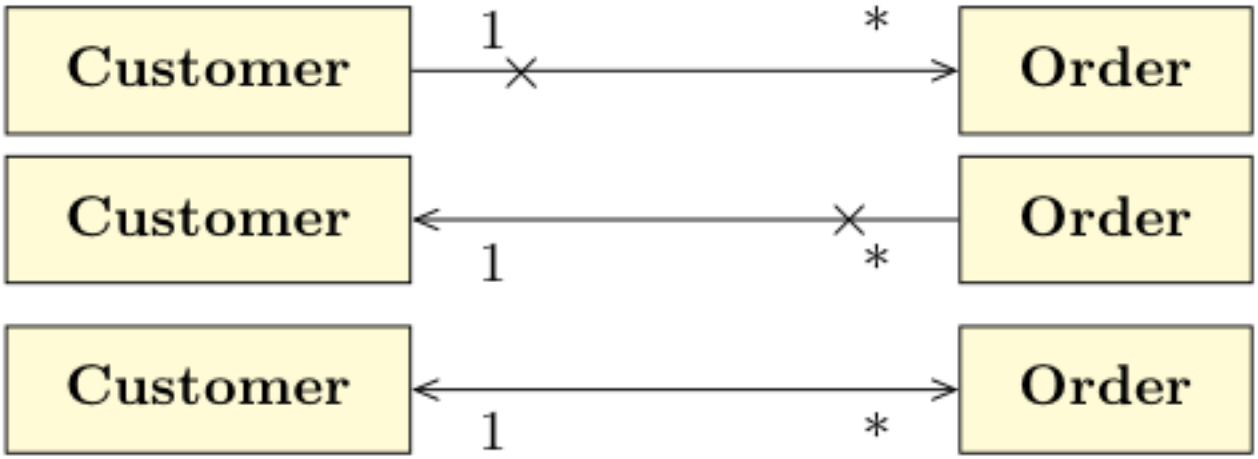
- Which among these are the rules to be considered to form Class diagrams?
- a) Class symbols must have at least a name compartment
- b) Compartment can be in random order
- c) Attributes and operations can be listed at any suitable place
- d) None of the mentioned

# Navigability of Association

- Some call this “direction of visibility”
- Does each class really need to store a reference to each other?
- One reason for having an association between classes:
  Messages between objects of those classes
- But, often “knowledge” indicated by association is only in one direction
  - Example: In a computer system, a User needs access to his/her Password
  - From a Password object we should not be able to get back to a User!

# Navigability of Association

- Arrow heads on the end of the association lines indicate to which target the associations can be navigated.
- Crosses on the line indicate that the association can not be navigated to the corresponding target.
- If no arrows or crosses are on one end of the line, the navigability in that direction is unspecified.



# Navigability of Association


- Customer object will have a property containing the Order objects that this Customer object owns. However, the Order objects will not have any property identifying its owning Customer.


- Order object will have a property to identify its owning Customer, but there would be no reverse direction references.


- Association is bidirectional. Both the Customer can find its collection of Order objects, and each Order object can find its owning Customer object.

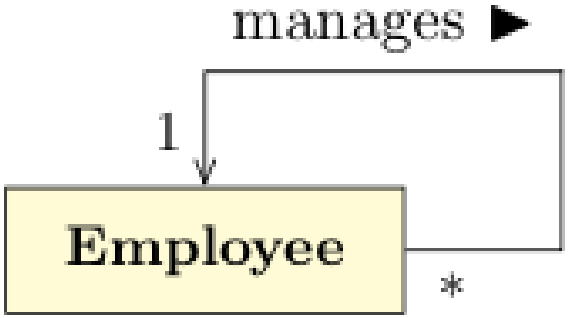
# Degree of Association

- **Binary Association**
  - Association between two classes
- **Recursive Association**
  - Recursive association is between a class and itself.
- **N-ary Association**
  - N-ary association is between three or more classes

- Example:
- Each employee has a reference to the other employee who manages him or her.
- What is the degree of association?

# Recursive Associations

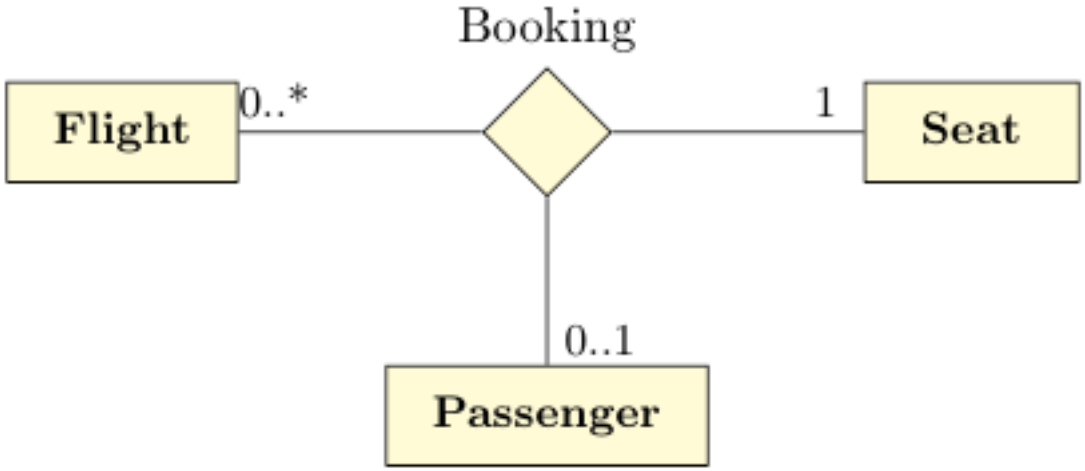
- Recursive association is between a class and itself.




- Each employee has a reference to the other employee who manages him or her.
- Clockwise: “Employee (i.e. the manager) manages many Employees”. Anticlockwise: “Employee is managed by one Employee”.

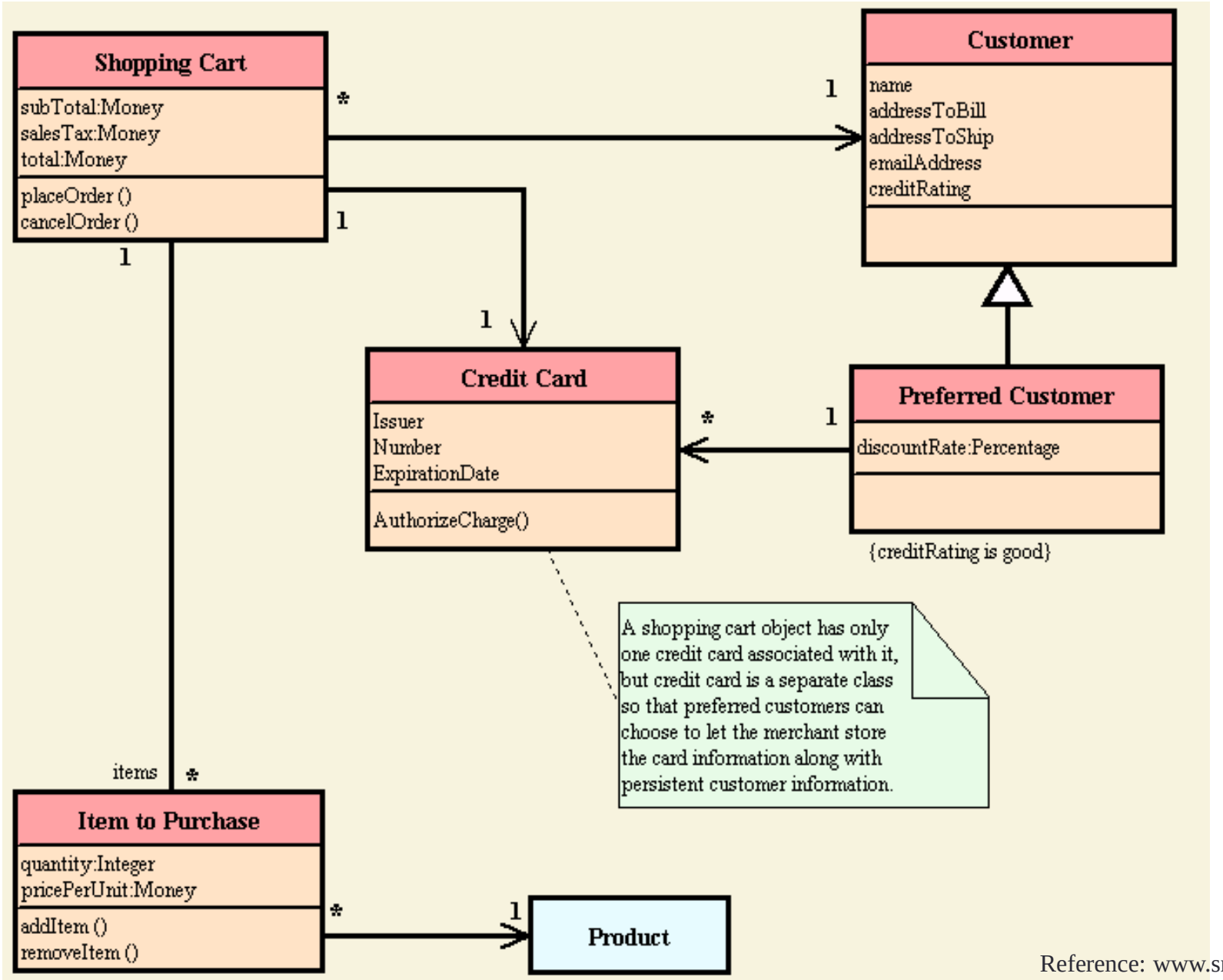
# N-ary Associations

- For a given flight and seat, there is 0 or 1 passengers.
- For a given passenger and seat, there may be any number of flights (e.g. the passenger may have the same seat number on more than one flights or there may be no flights for which this passenger has a seat with this seat number).
- For a given flight and passenger , there is only 1 seat.



# N-ary Associations

- A large open diamond is placed at the junction of the association.
- Cardinality information for N-ary associations are defined in the UML standard but are often misunderstood and their specified.
- Typically, you should only use an N-ary association if it is really necessary, and then you should add further comments or constraints to clarify your intended meaning.




Reference: www.smartdraw.com

# References

- A number of slides in this talk is based on:
  - Alan P. Sexton hand-outs (Introduction to Software Engineering. The University of Birmingham. Spring Semester 2014)