Reading UML Class models

Software Requirements and Design

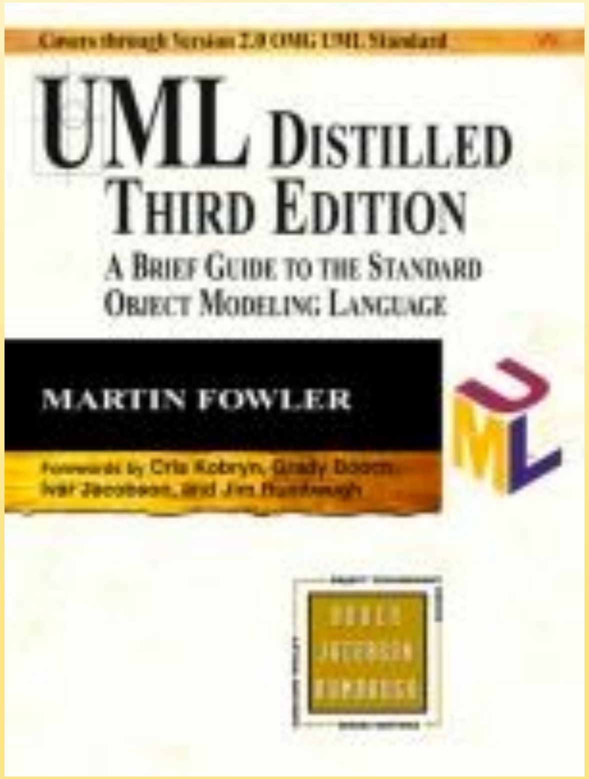
CITS4401

Lecture 9

Key Topics this week

1. UML Class diagrams (what they are for and how to read them)

2. Discovering **objects** (noun discovery method) and **associations** (Class, Responsibilities, Collaboration (CRC) method)


Covers through Version 2.0 OMG UML Standard
UML Distilled
Third Edition
A Brief Guide to the Standard Object Modeling Language
Martin Fowler
Forewords by Cris Kobryn, Grady Booch, Ivar Jacobson, and Jim Rumbaugh

Requirements Analysis Framework

- Requirements analysis generates an analysis model with 3 parts
  - functional model: use cases & scenarios
  - static analysis object model: class & object diagrams
  - dynamic model: statechart & sequence diagrams (later in the unit)
- This week we will look at **static class models**
- Goal is to investigate the problem domain as far as possible before moving to the solution domain (for design & implementation)

1. UML Class diagrams

In this section we will learn how to READ and understand a UML class diagram

Why build class models?

Why do we build class models? In order to …

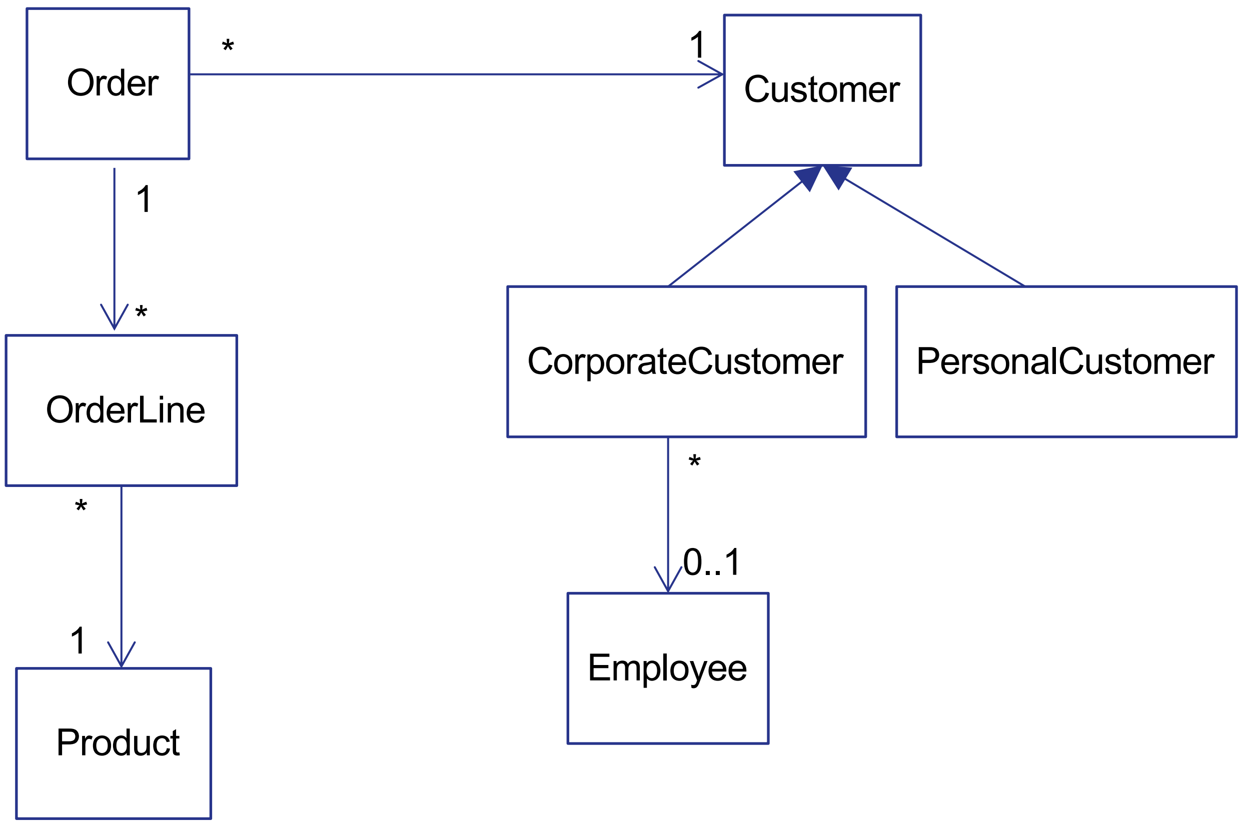
1. Build, as quickly and cheaply as possible, a system that satisfies our current requirements;

2. Build a system which will be easy to maintain and adapt to future requirements

What is a UML class diagram?

- A **class diagram** describes the **types of objects** in the system and the various kinds of **static relationships** that exist among them.
- Class diagrams also show the **properties and operations of a class** and the constraints that apply to the way objects are connected
- Fowler Ch 3

UML class diagram for an online shop (Example 1)

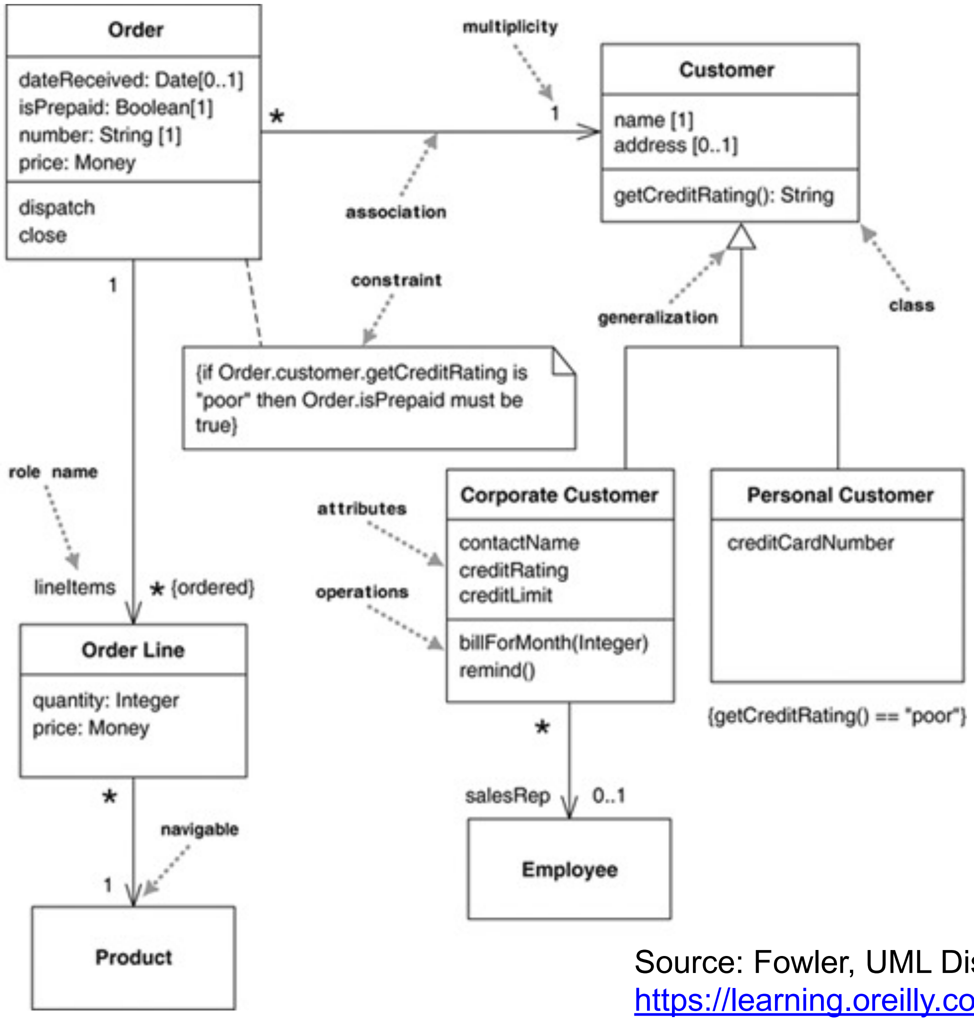


UML Class diagram interpretation

- 1 customer has 0 or more orders, but each order has a single customer
- An order comprises one or more order lines
- Many order lines refer to a product

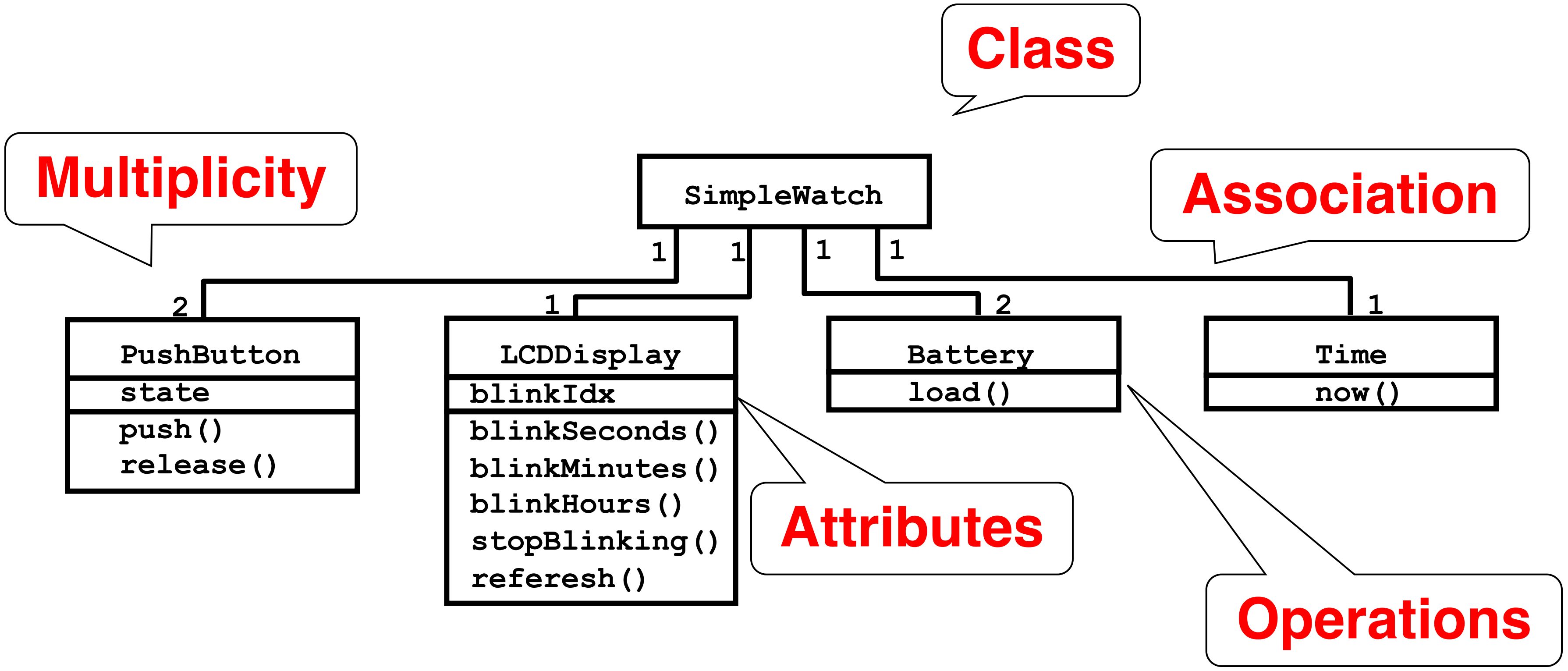
- Corporate customer and personal customer are special types of customer
- A corporate customer is supported by 0 or 1 sales reps who are order company employees – that is they may have a sales rep or may not

Example 1 with more detail



Source: Fowler, UML Distilled, Figure 3.1
https://learning.oreilly.com/library/view/uml-distilled-a/0321193687

Example 2

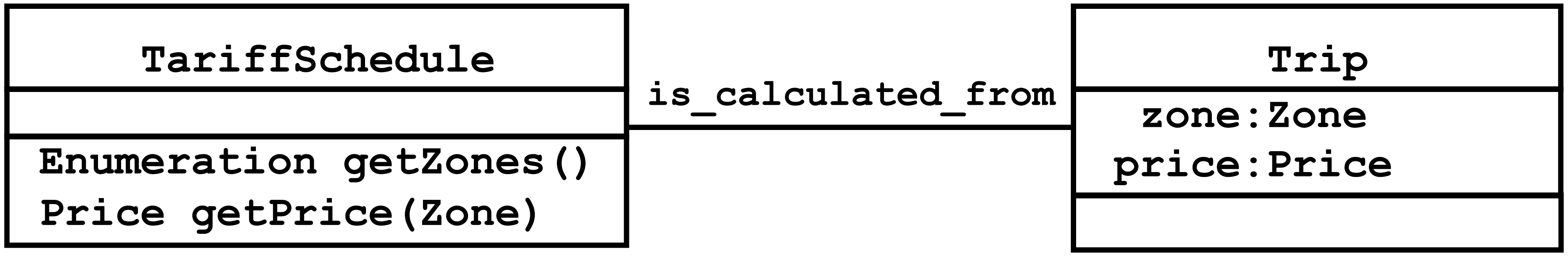


Class diagrams represent the structure of the system

classes

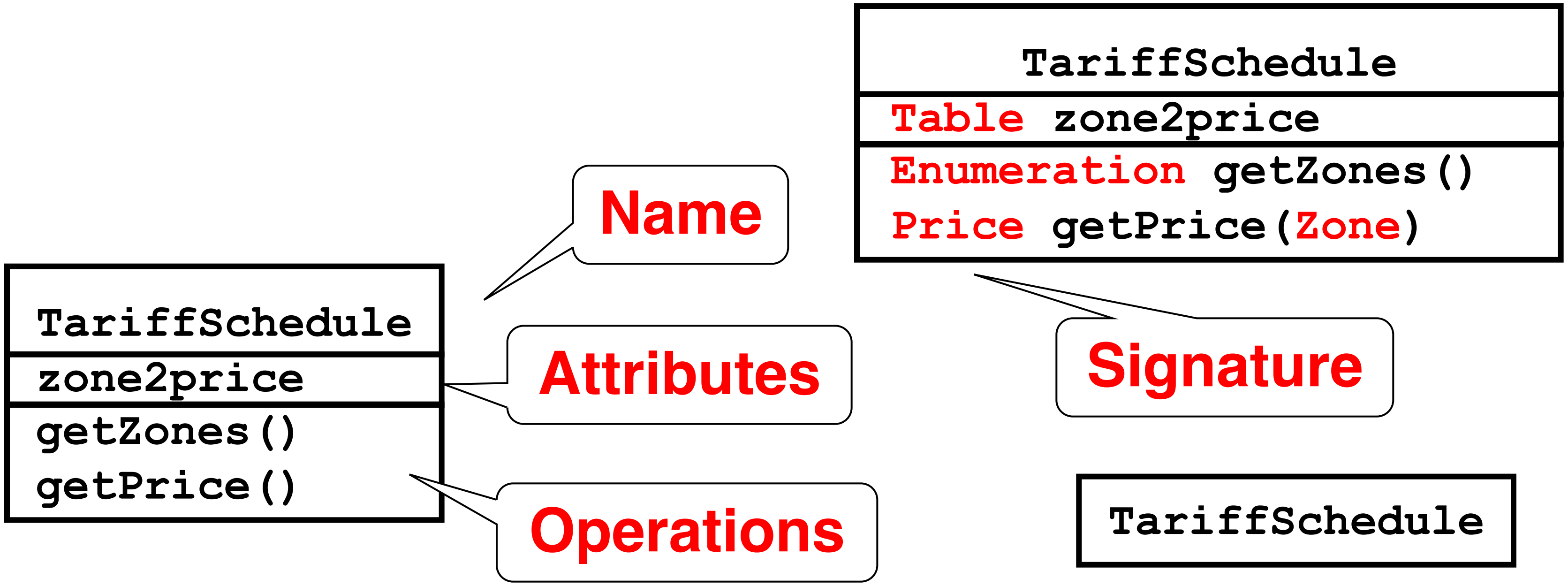
classes, associations, attributes, generalization, and constraints

Class Diagrams



- Class diagrams represent the structure of the system.
- Class diagrams are used
  - during requirements analysis to model problem domain concepts
  - during system design to model subsystems and interfaces
  - during object design to model classes.

Classes

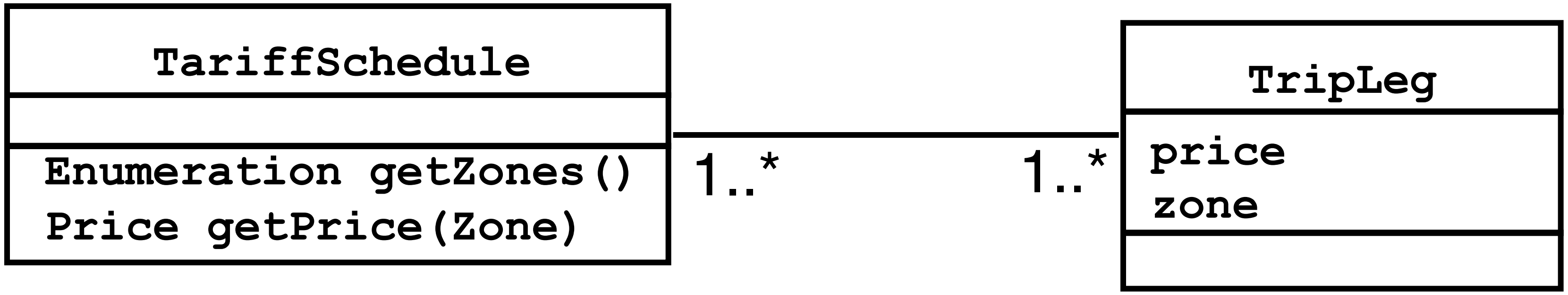


- A ***class*** represent a concept.
- A class encapsulates state ***(attributes)*** and behavior ***(operations).***
- Each attribute has a ***type***.
- Each operation has a ***signature***.
- The class name is the only mandatory information.

associations

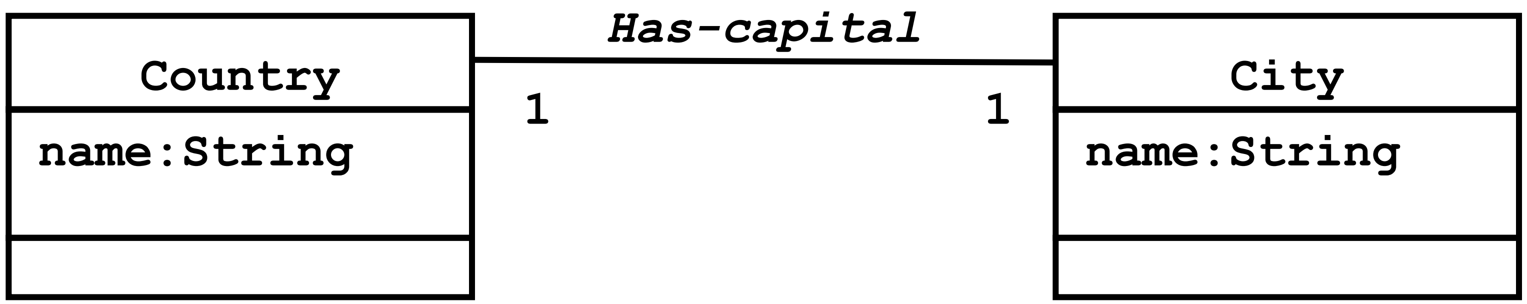
classes, **associations,** attributes, generalization, and constraints

Associations

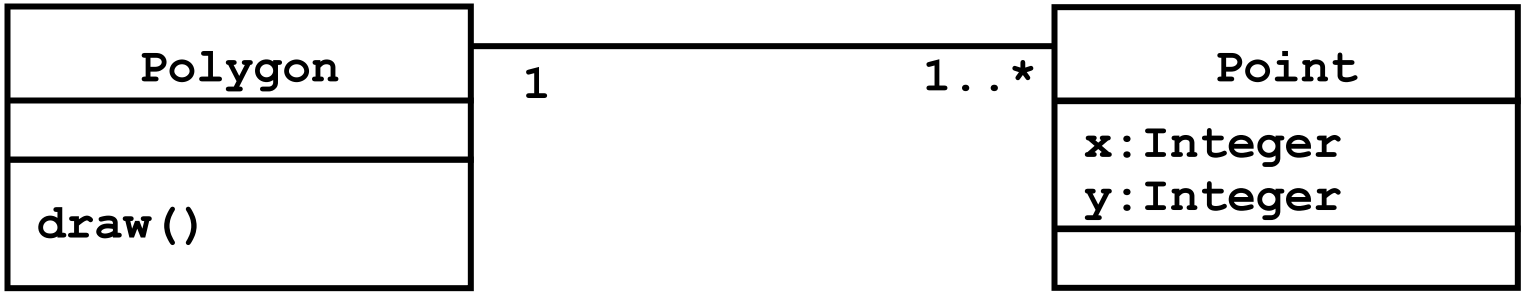


- Associations denote relationships between classes
- *A is associated with B* is: *A has to know about B*
- The multiplicity of an association end denotes how many objects the source object can legitimately reference.

1-to-1 and 1-to-Many Associations



1-to-1 association



1-to-many association

Multiplicity (cont)

The **multiplicity** of a property is an indication of how many objects may fill the property.

The most common multiplicities you will see are

1	(An order must have exactly one customer.)
0..1	(A corporate customer may or may not have a single sales rep.)
*****	(A customer need not place an Order; there is no upper limit to the number of Orders. That is, zero or more orders.)

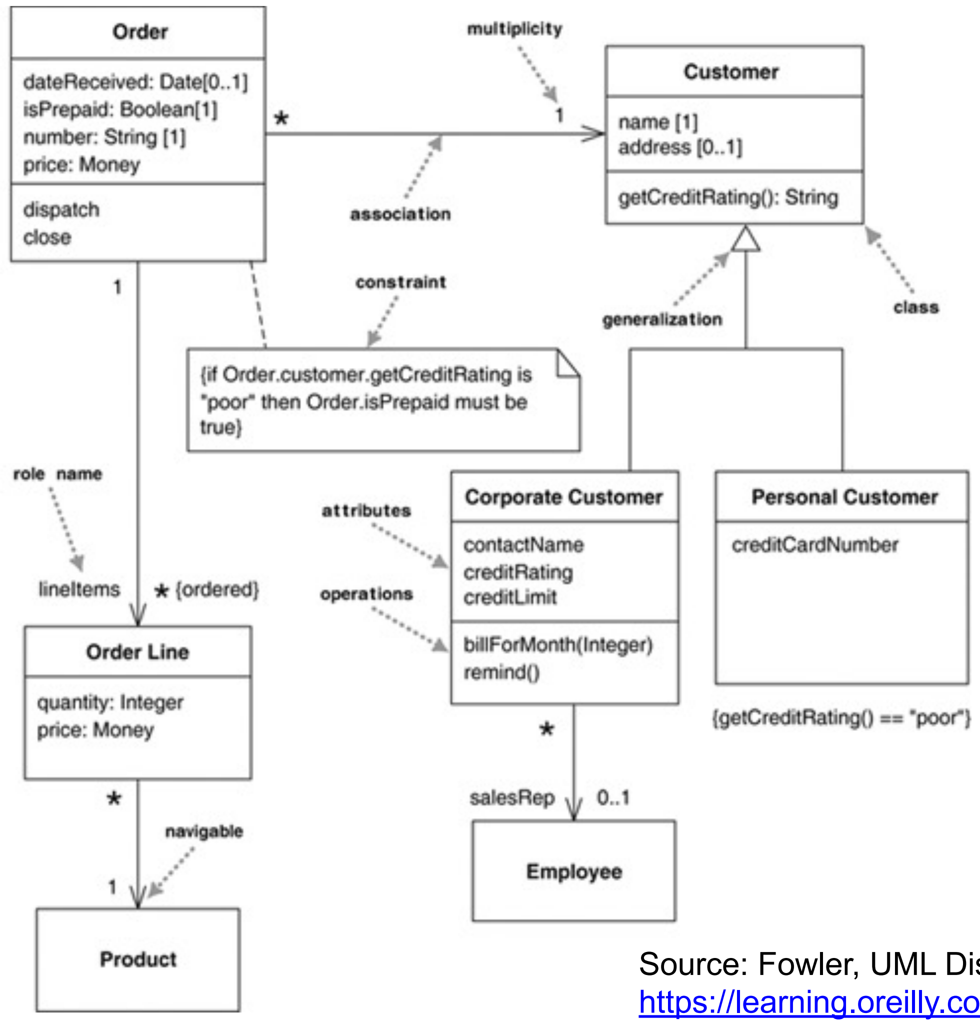
If I have a multivalued property, I prefer to use a plural form for its name.

Multiplicity (cont)

In attributes, you come across various terms that refer to the multiplicity.

- **Optional** implies a lower bound of 0.
- **Mandatory** implies a lower bound of 1 or possibly more.
- **Single-valued** implies an upper bound of 1.
- **Multivalued** implies an upper bound of more than 1: usually *.

Practice example: Multiplicities



Source: Fowler, UML Distilled, Figure 3.1
https://learning.oreilly.com/library/view/uml-distilled-a/0321193687

attributes

classes, associations, **attributes**, generalization, and constraints

Attributes

- **Properties** represent structural features of a class. As a first approximation, you can think of properties as corresponding to fields in a class
- The **attribute** notation describes a property as a line of text within the class box itself.
- Properties are a single concept, but they appear in two quite distinct notations: attributes and associations.
- Although they look quite different on a diagram, they are really the same thing.

Attributes or Associations

Source: Fowler, UML Distilled
https://learning.oreilly.com/library/view/uml-distilled-a/0321193687

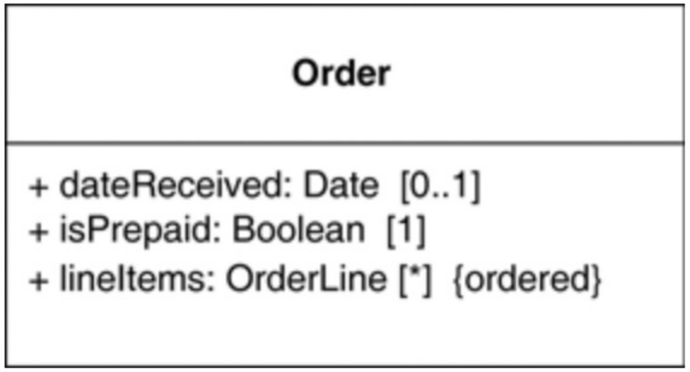

Order
+ dateReceived: Date [0..1]
+ isPrepaid: Boolean [1]
+ lineItems: OrderLine [*] {ordered}

Figure 3.2. Showing properties of an order as attributes

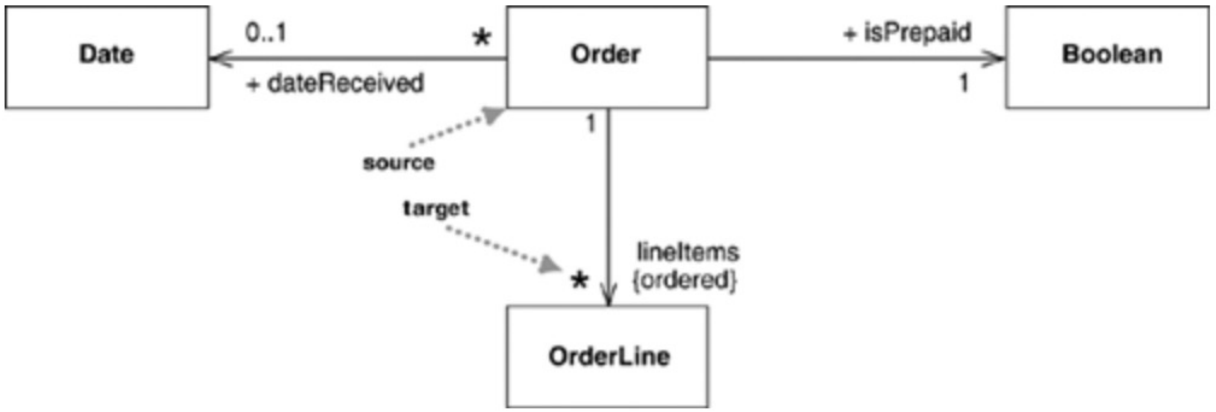


Figure 3.3. Showing properties of an order as associations

Which to use? [Fowler]

With two notations for the same thing, the obvious question is,
Why should you use one or the other?
In general, I tend to use

attributes for small things,
such as dates or Booleans—in general, value types (page 73)—and

associations for more significant classes, such as customers and orders.

I also tend to prefer to use class boxes for classes that are significant for the diagram, which leads to using associations, and attributes for things less important for that diagram.

The choice is much more about emphasis than about any underlying meaning.

Programming Interpretation

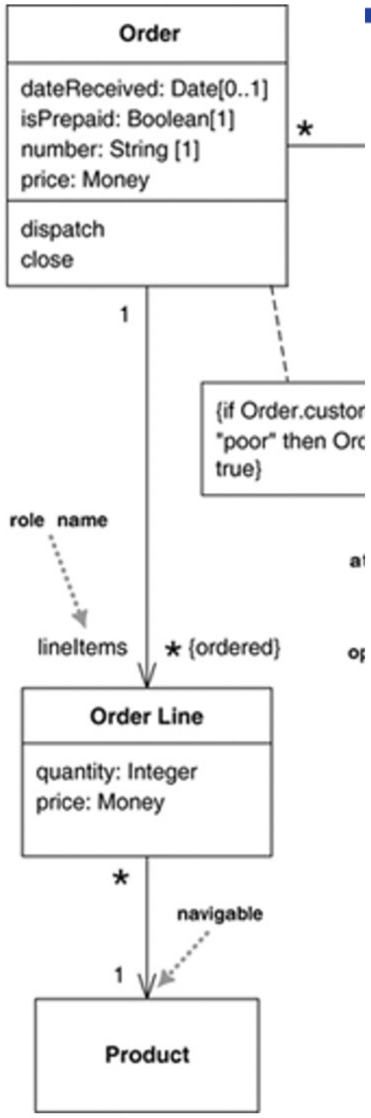
As with anything else in the UML, there's no one way to interpret properties in code.

The most common software representation is that of a field or property of your programming language.

So the Order Line class from Figure 3.1 would correspond to something like the following in Java:

```
public class OrderLine...
private int quantity;
private Money price;
private Order order;
private Product product
```

If an attribute is multivalued, this implies that the data concerned is a collection (ArrayList etc in Java)

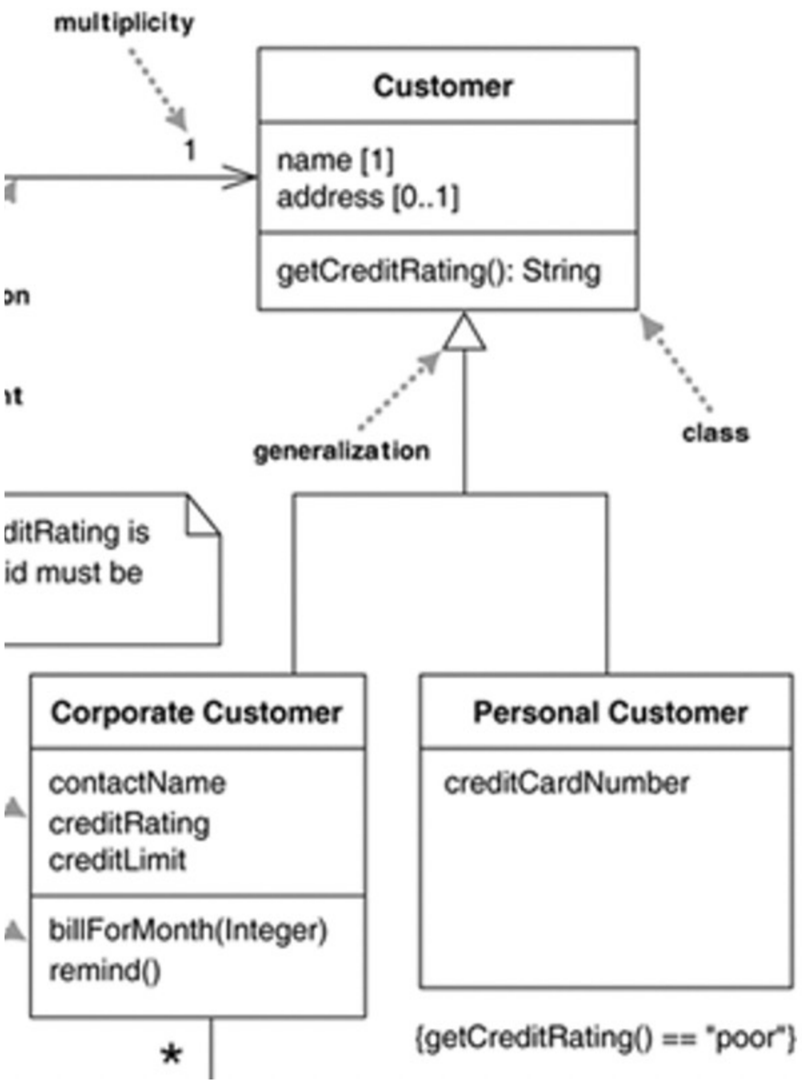


generalization

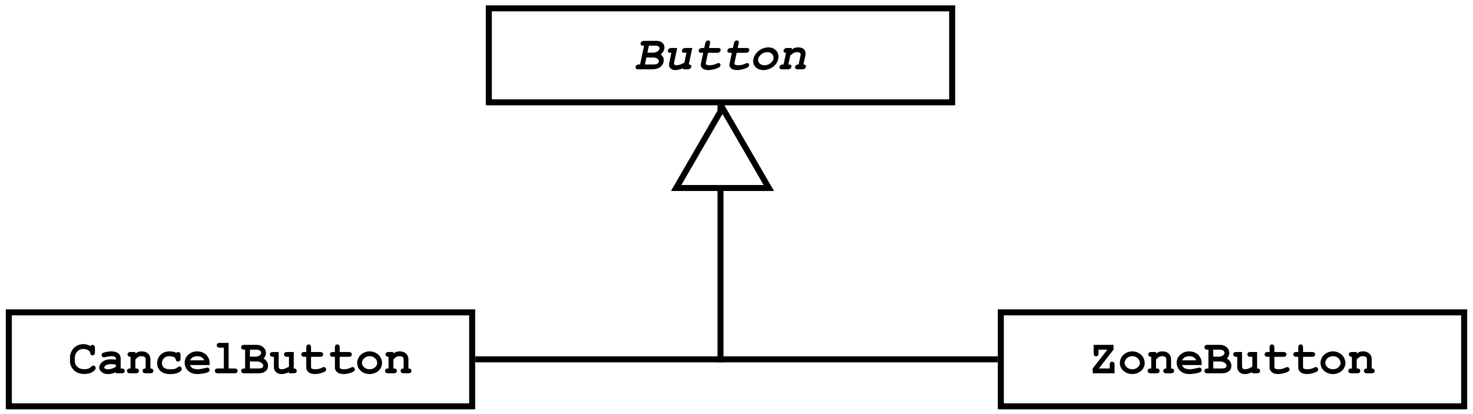
classes, associations, attributes, **generalization**, and constraints

What is generalization?

- A typical example of **generalization** involves the personal and corporate customers of a business.
- They have differences but also many similarities.
- The similarities can be placed in a general Customer class (the supertype),
- Personal Customer and Corporate Customer are subtypes.



Generalization



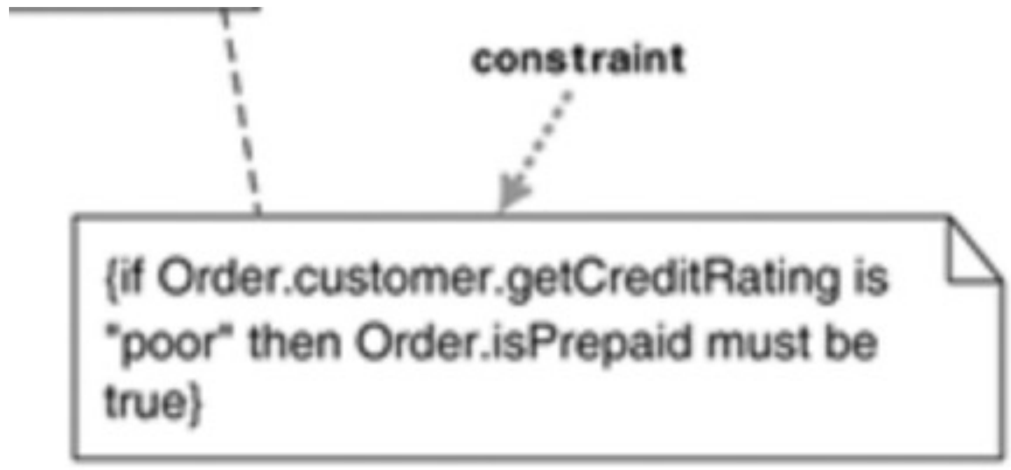
- Generalization relationships denote inheritance between classes.
- The children classes inherit the attributes and operations of the parent class.
- Generalization simplifies the model by eliminating redundancy.

constraints

classes, associations, attributes, generalization, and **constraints**

Constraints

- UML class diagrams indicate constraints via association, attributes and generalizations
- But other types of constraints may also need to be capture
- UML allows you to use *anything* to describe constraints
- The only rule is you put them inside curly braces {}
- Natural or formal language can be used to describe the constraint


constraint
{if Order.customer.getCreditRating is "poor" then Order.isPrepaid must be true}

Summary

1. UML Class diagrams (what they are for and how to read them)
2. Discovering objects (noun discovery method)
3. Discovering associations (Class, Responsibilities, Collaboration (CRC) method)

UML class diagrams

UML Class Diagrams describe the **static** structure of the system

- classes,
- class attributes,
- associations between classes,
- association roles and multiplicity

classes: *features and facts about the problem domain which matter in the system we are building to support it*

Reference: UML Distilled by Martin Fowler, Chapter 3

Our focus

- Class diagrams are the backbone of the UML, so you will find yourself using them all the time.
- But the trouble with class diagrams is that they are so rich, they can be overwhelming to use.
- In CITS4401 we will study these elements:

  classes, associations, attributes, generalization, and constraints

- For more detail: see UML Distilled by Martin Fowler, Chapter 3 & 5
- And some info on the hidden slides – not examinable but for interest

When to Use Class Diagrams
[Martin Fowler]

- Class diagrams are the backbone of the UML, so you will find yourself using them all the time.
- The trouble with class diagrams is that they are so rich, they can be overwhelming to use. Here are a few tips.
- **Don't** try to use all the notations available to you.

  Start with the **simple stuff**: classes, associations, attributes, generalization, and constraints. Introduce other notations only when you need them.
- I've found conceptual class diagrams very useful in exploring the language of a business. For this to work, you have to work hard on **keeping software out of the discussion** and keeping the notation very simple.

When to use (2)

- **Don't** draw models for everything; instead, concentrate on the key areas. It is better to have a few diagrams that you use and keep up to date than to have many forgotten, obsolete models.

- The biggest danger with class diagrams is that you can focus exclusively on **structure** and ignore **behaviour.**

  Therefore, when drawing class diagrams to understand software, always do them in conjunction with some form of behavioural technique. If you're going well, you'll find yourself swapping between the techniques frequently.

- Next week we will study 2 behavioural UML models: sequence diagrams and state charts.

Recommended reading

UML Distilled by Martin Fowler, Chapter 3

UWA Unit Readings or
https://my.safaribooksonline.com/book/software-engineering-and-development/uml/0321193687

Barnes and Kolling, Objects First with Java, Chapter 15

B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering – Using UML, Patterns, and Java,* 3rd ed., Prentice Hall, 2010

- Section 2.2
- Section 2.4
- Section 5.3