

# CS 213 – Software Methodology

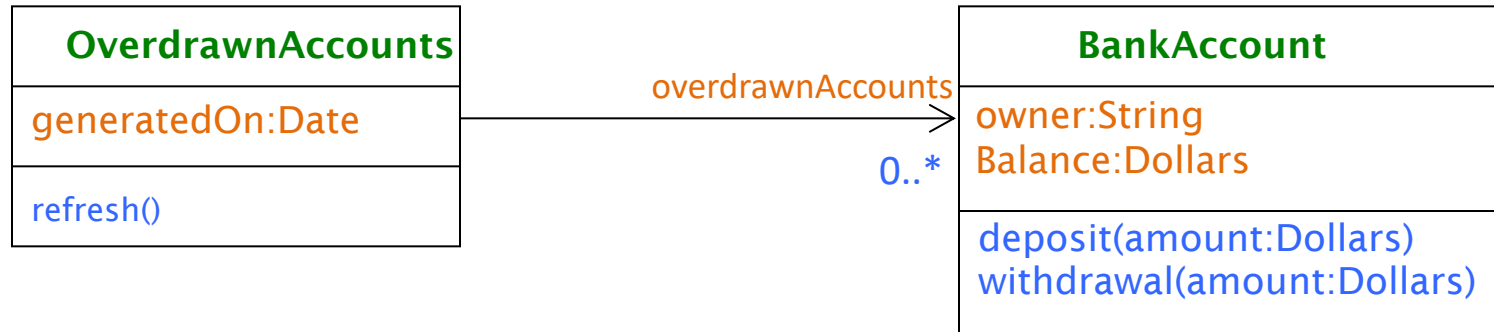
*Sesh Venugopal*

UML Class Diagram - II

# Unidirectional Association

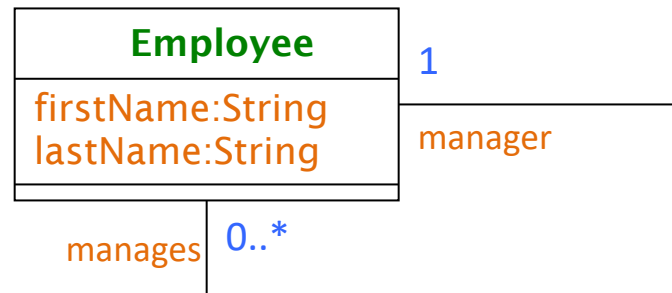
(From From “UML basics: The class diagram” by Donald Bell)

<http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/>



OverdrawnAccounts *knows about* BankAccount,  
but BankAccount does not know about OverdrawnAccounts => in  
the implementation OverdrawnAccounts would have a list of BankAccount  
instances, but BankAccount would not refer back to OverDrawnAccounts

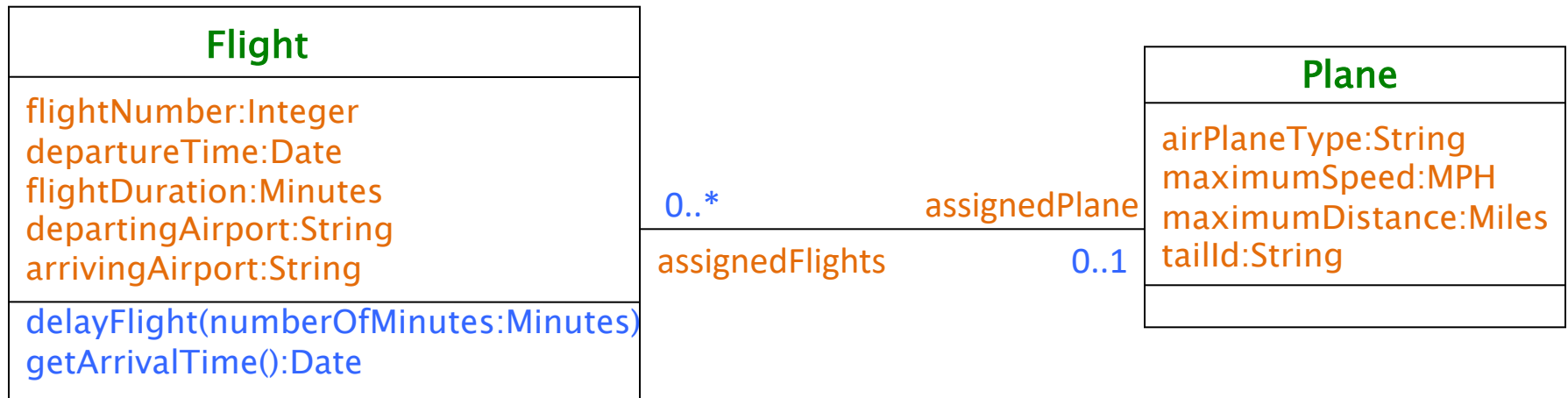
## Reflexive Association



# Bidirectional Association and Navigability

From “UML basics: The class diagram” by Donald Bell

<http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/>

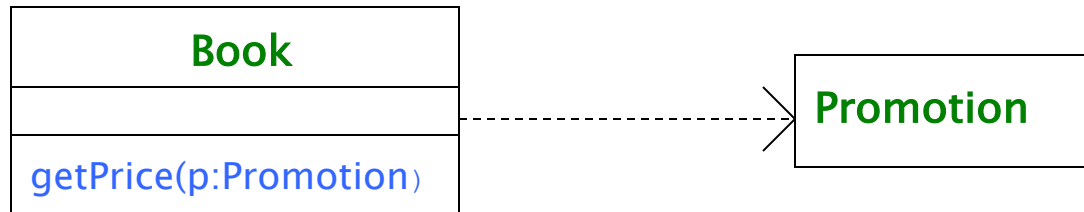


*Flight knows about Plane*, and *Plane knows about Flight*. When implemented, it will be easy to know which Plane is assigned to a Flight, and which Flights a Plane has been assigned to.

In other words, it is possible to **navigate** from Flight to Plane, and vice versa

# Dependency

- Class A depends on class B if A uses B in such a way that a change in B will effect A

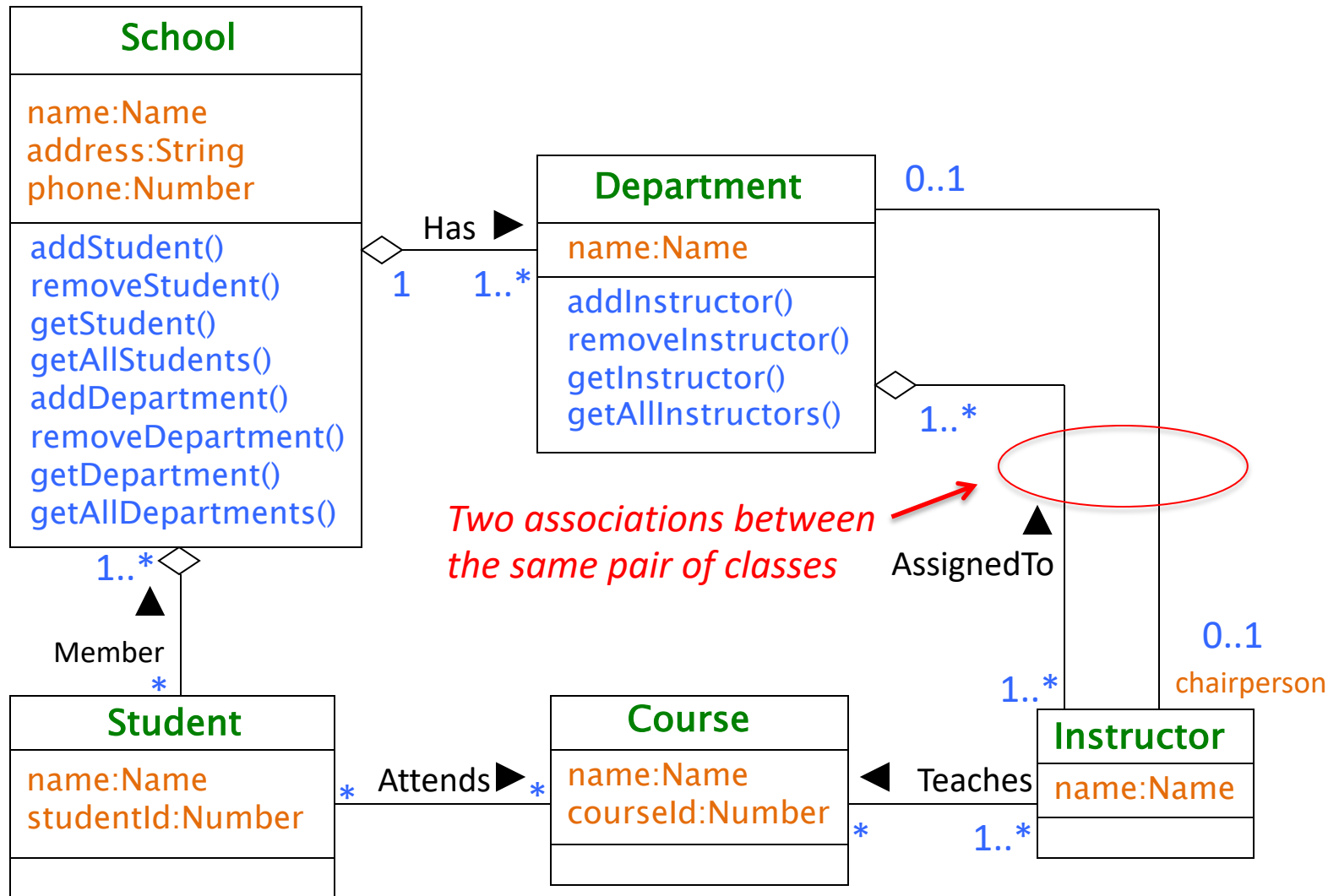


The Book class uses (depends on) on the Promotion class – changing Promotion would affect Book

- Say A depends on B. Typically then, B would appear as a parameter, return type, or local variable in a method of A
- Dependencies are the weakest kind of relationships, and the “dependee” class (e.g. Promotion) is subordinate to the “depender/dependent” class (e.g. Book) In other words, the Book class can be defined meaningfully even without the Promotion class

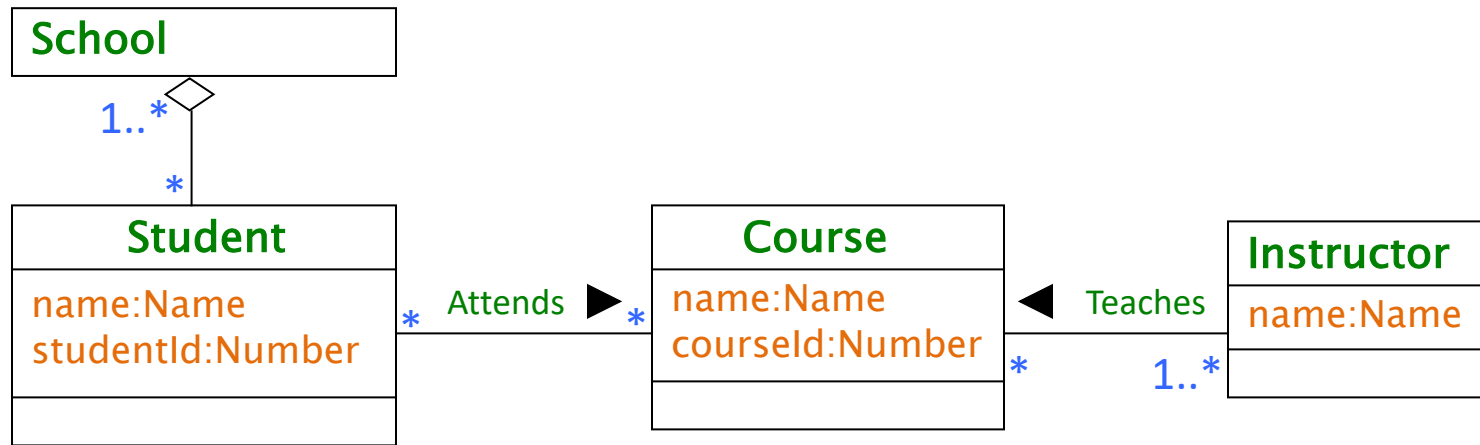
# Modeling a University

From “The Unified Modeling Language User Guide” by Booch, Rumbaugh, and Jacobson



# Modeling a University

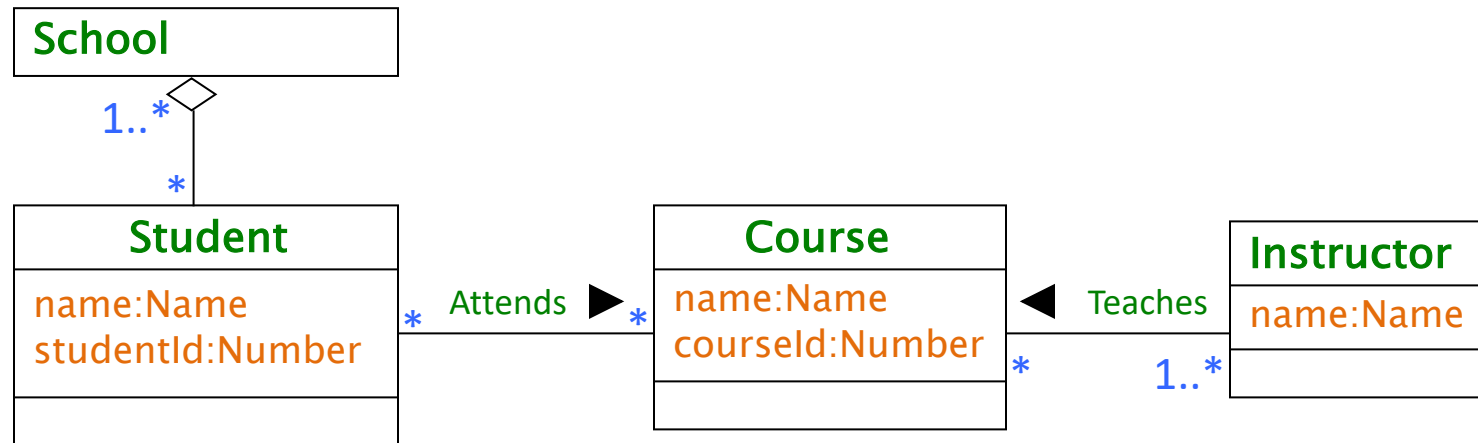
## *Relationship data*



Write the Student and Course classes with just fields (no methods)

# Modeling a University

## *Relationship data*



```
public class Student
{
    Name name;
    Number studentId;
    Course[] courses;
    School[] schools;
    ...
}
```

```
public class Course {
    Name name;
    Number courseId;
    ArrayList<Student> students;
    Instructor[] instructors;
    ...
}
```

**Data needed to store a relationship between two classes DO NOT show up as UML attributes in either class, because they are NOT inherent properties of either of the classes in the relationship.**

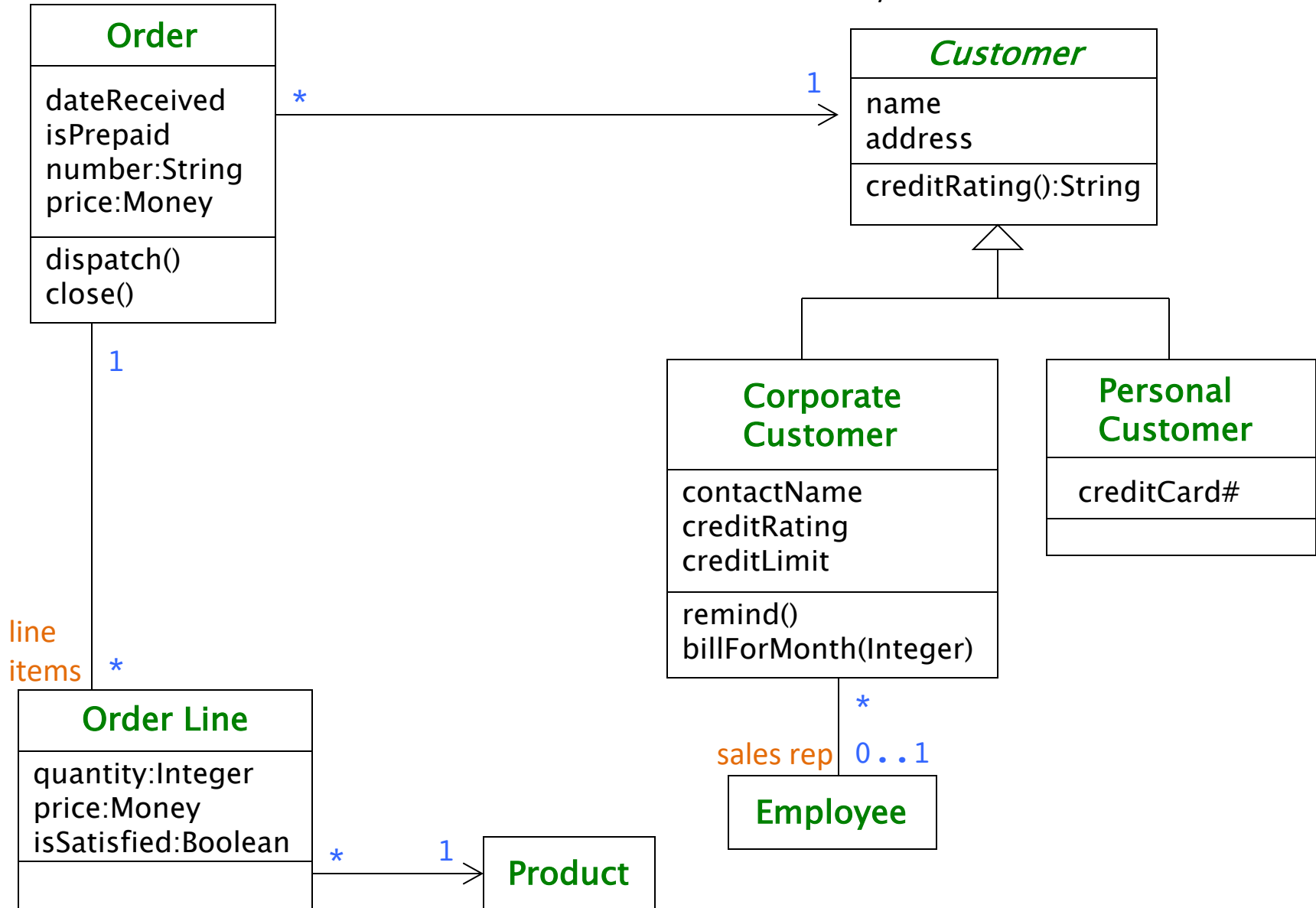
# Modeling Orders for Products

Draw a UML Class Diagram to model orders for products. Customers who place orders can be personal customers or corporate customers. An order could have multiple products (items) in it, and at any point in time the company needs to know which of these items have been shipped.



# Modeling Orders for Products

From "UML Distilled" by Martin Fowler with Kendall Scott

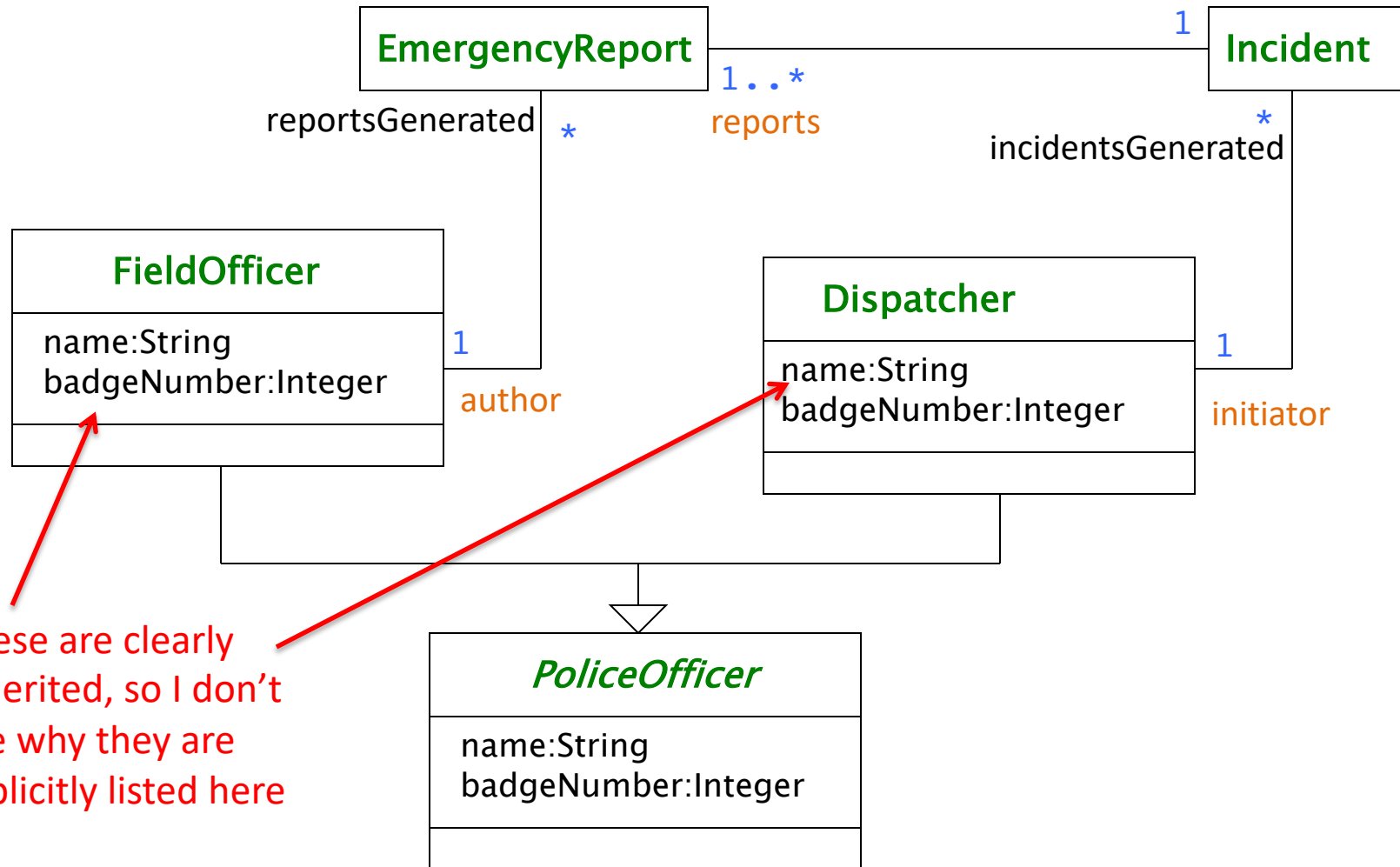


# Modeling Police Incidents

Draw a UML Class Diagram to model the handling of incidents by the dispatchers and field officers of a police department. Dispatchers log incidents, and field officers write emergency reports, as many as each incident requires.

# Abstract Class/Inheritance/Bidirectional Association

From “Object-Oriented Software Engineering” 2nd ed. by Bruegge and Dutoit

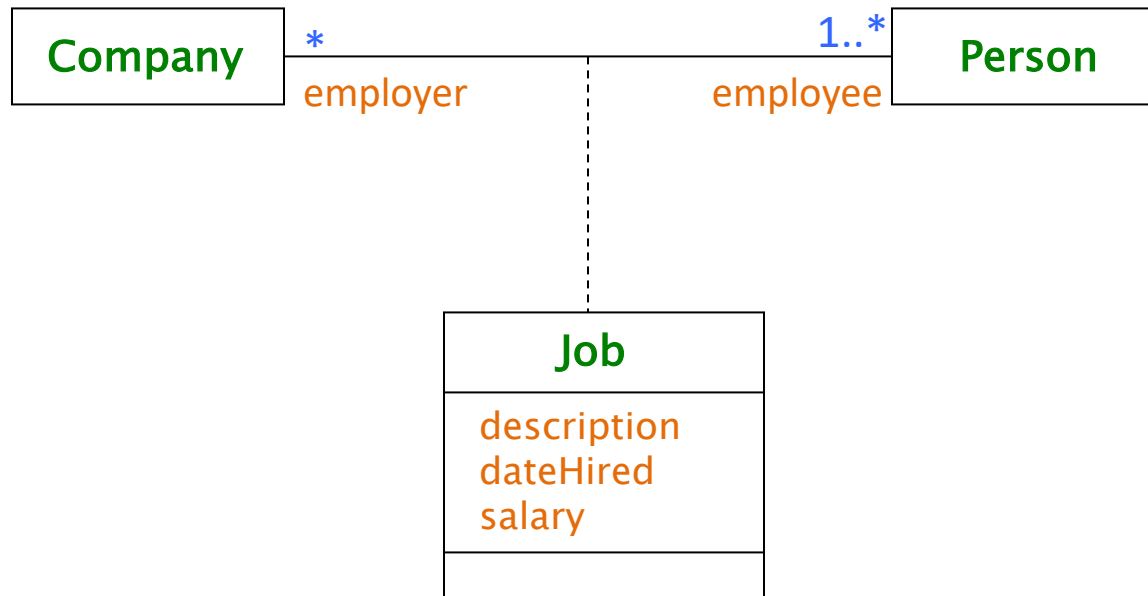


These are clearly inherited, so I don't see why they are explicitly listed here

# Association Class

- At times, an association between two classes itself has properties - an **association class** is used to model these properties.

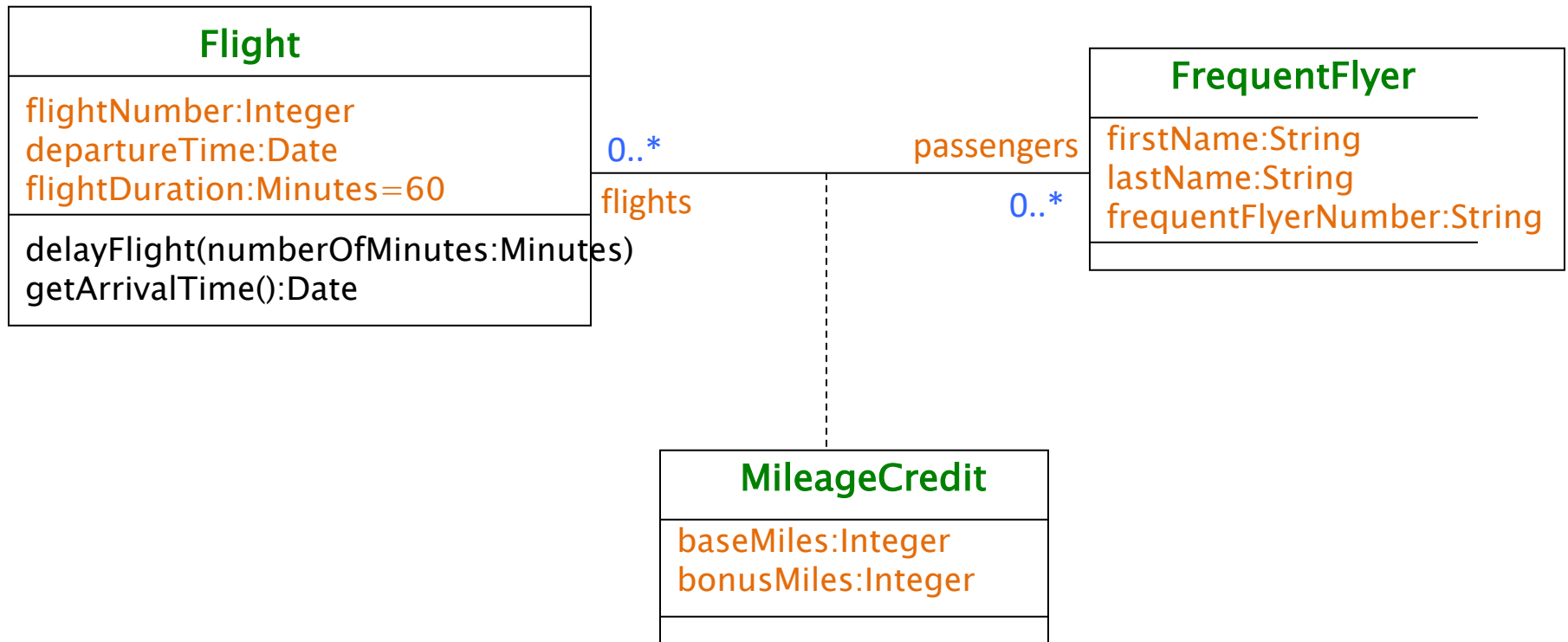
From “The Unified Modeling Language User Guide” by Booch, Rumbaugh, and Jacobson



# Association Class: Example

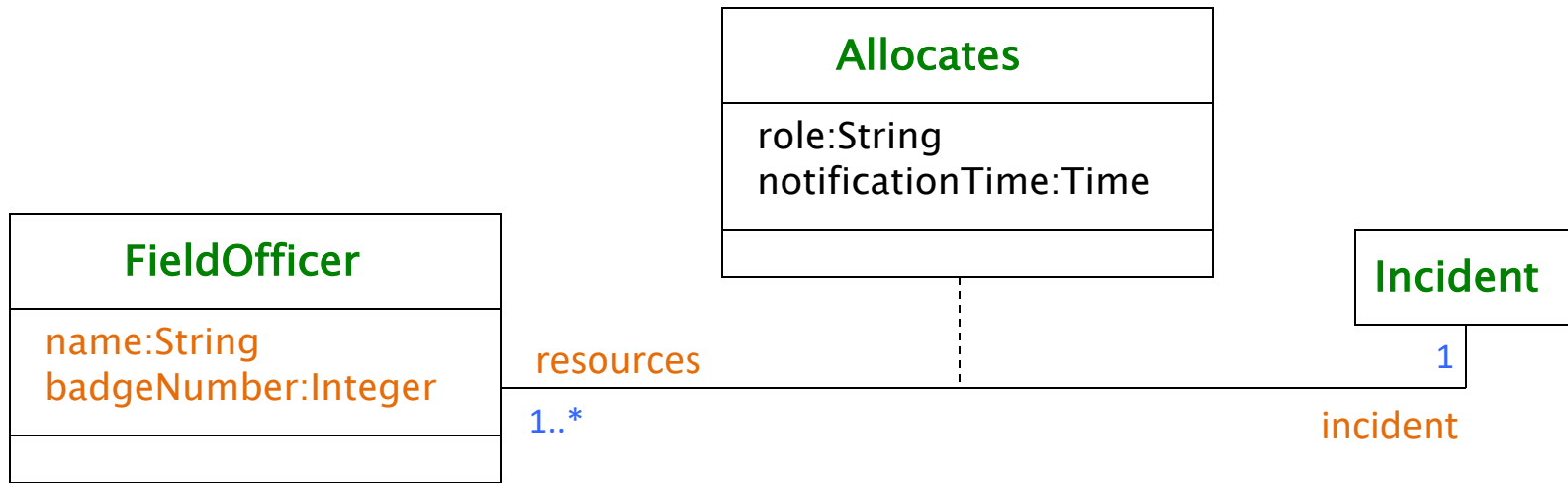
From “UML basics: The class diagram” by Donald Bell

<http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/>

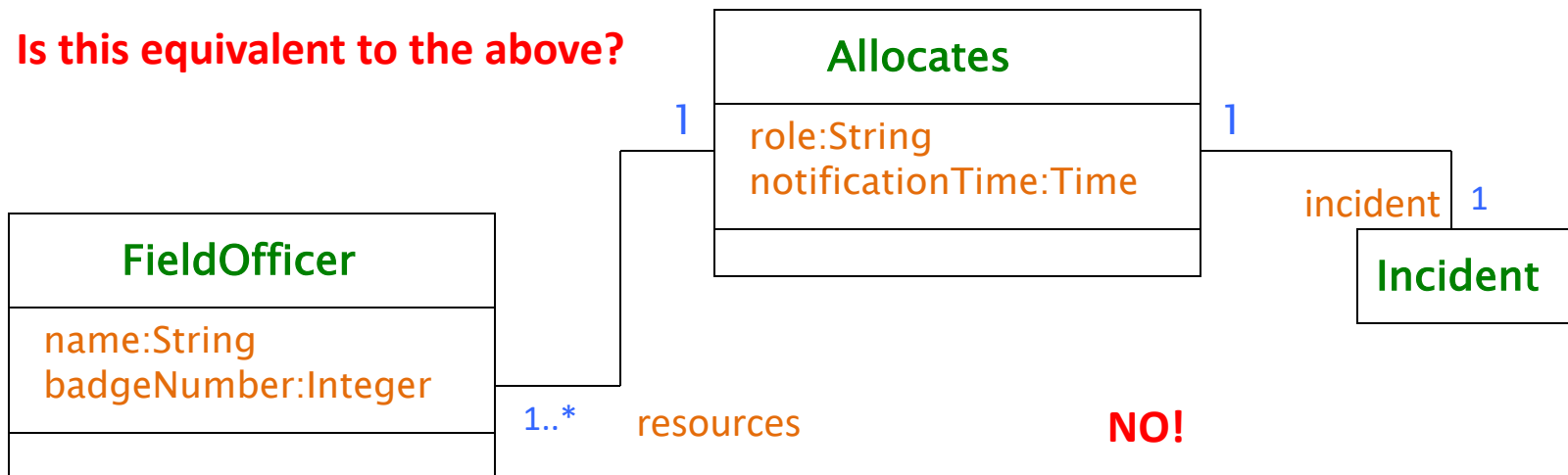


# Association Class: Example

From “Object-Oriented Software Engineering” 2nd ed. by Bruegge and Dutoit



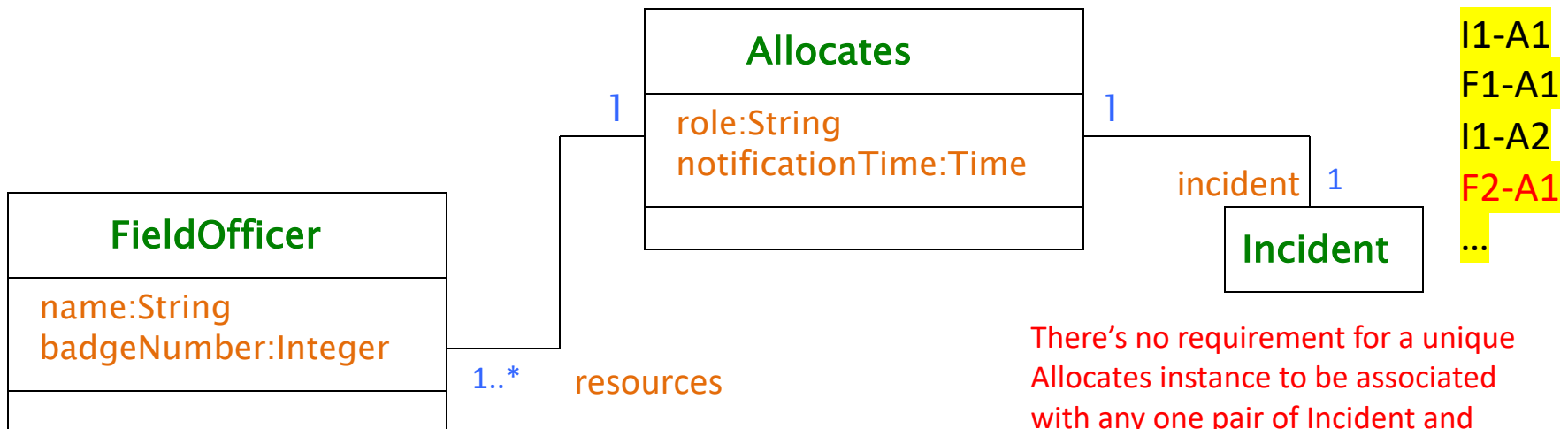
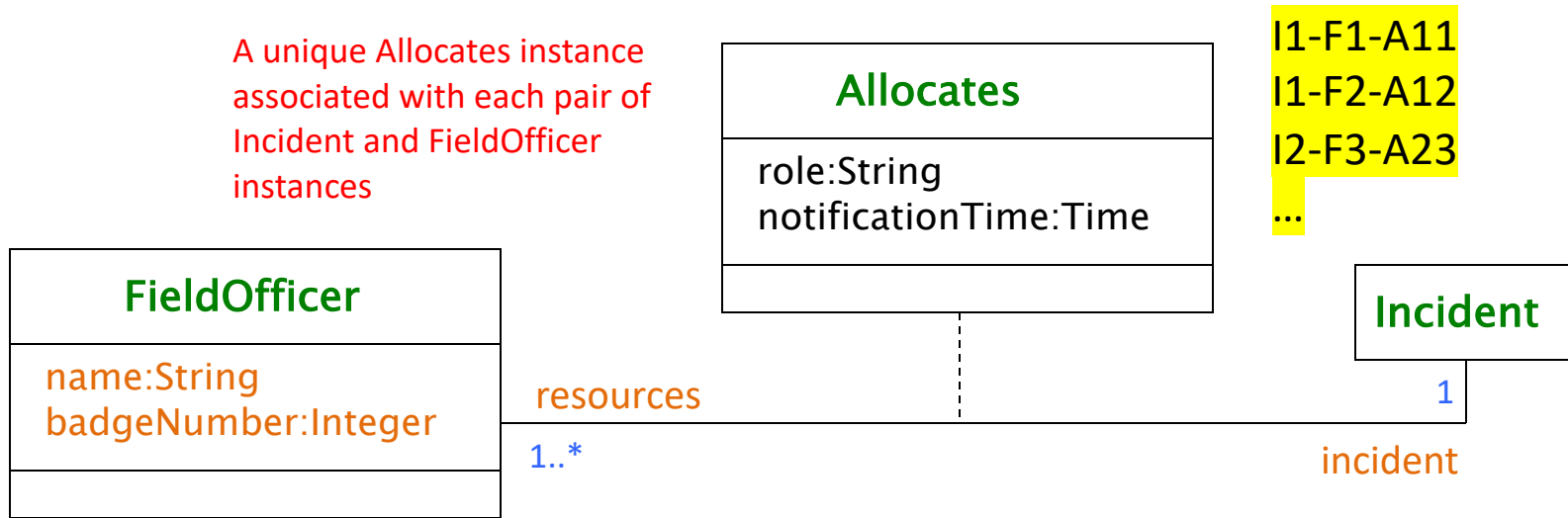
Is this equivalent to the above?



# Association Class: Example

From “Object-Oriented Software Engineering” 2nd ed. by Bruegge and Dutoit

A unique Allocates instance associated with each pair of Incident and FieldOfficer instances



There's no requirement for a unique Allocates instance to be associated with any one pair of Incident and FieldOfficer instances