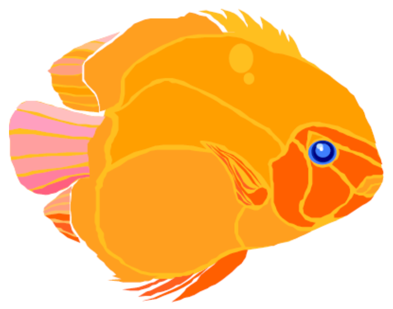
XII. Class and Object Diagrams

Classes, Attributes and Operations
Objects and Multi-objects
Generalization and Inheritance
Associations and Multiplicity
Aggregation and Composition
How to Use Class Diagrams



Presentation: N.C. Danh

Classes

- *A class describes a group of objects with*
  - ✓ *similar properties (attributes),*
  - ✓ *common behaviour (operations),*
  - ✓ *common relationships to other objects,*
  - ✓ *and common meaning ("semantics").*

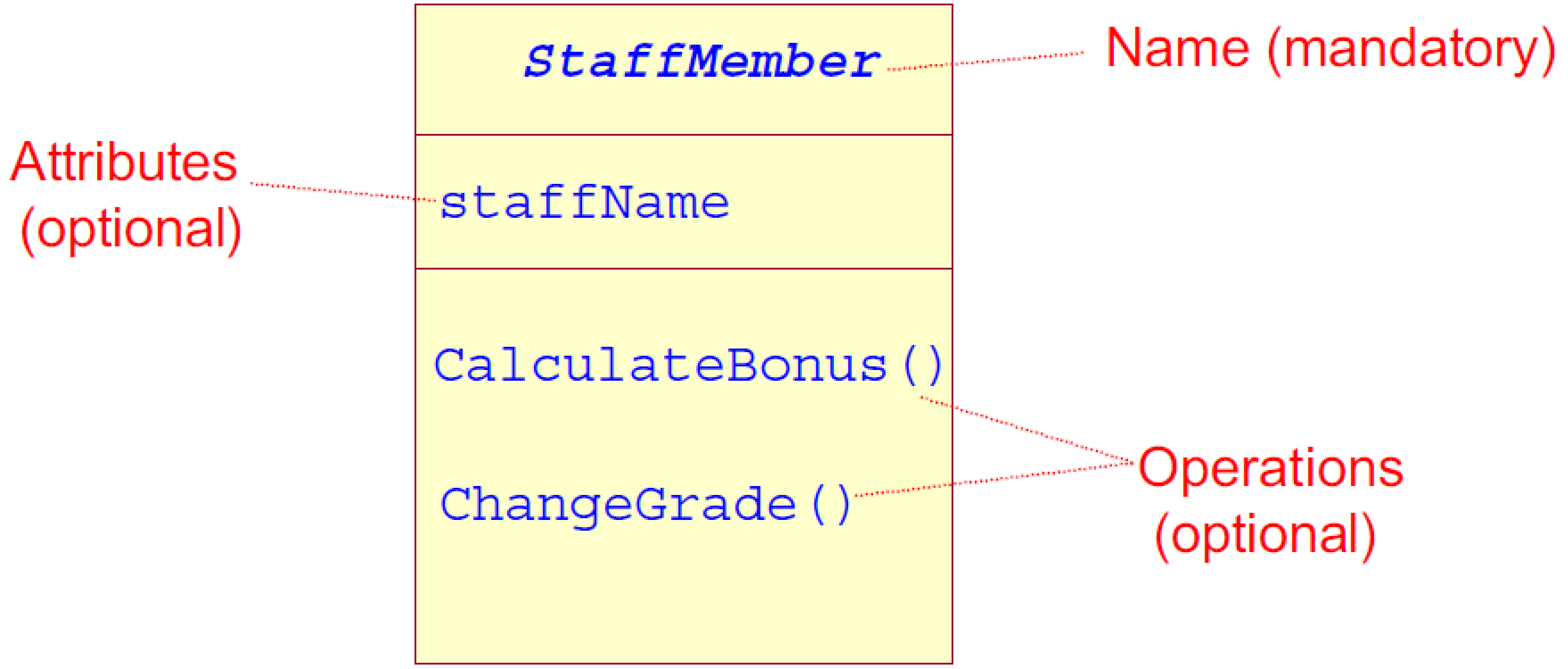
- *For example, "employee: has a name, employee# and department; an employee is hired, and fired; an employee works in one or more projects"*

Finding Classes

- *Finding classes in use case, or in text descriptions:*
  - ✓ *Look for nouns and noun phrases in the description of a use case or a problem statement;*
  - ✓ *These are only included in the model if they explain the nature or structure of information in the application.*
- *Don’t create classes for concepts which:*
  - ✓ *Are beyond the scope of the system;*
  - ✓ *Refer to the system as a whole;*
  - ✓ *Duplicate other classes;*
  - ✓ *Are too vague or too specific (few instances);*
- *Finding classes in other sources:*
  - ✓ *Reviewing background information;*
  - ✓ *Users and other stakeholders;*
  - ✓ *Analysis patterns;*
  - ✓ *CRC (Class Responsibility Collaboration) cards.*

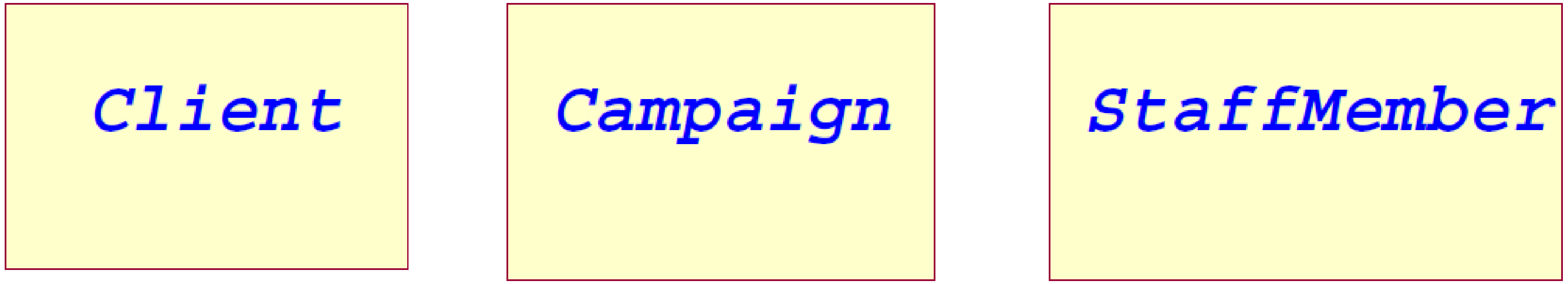
StaffMember Class for Agate

- *For example, we may want to represent the concept of a staff member for a company such as Agate in terms of the class* ***StaffMember.***



Names

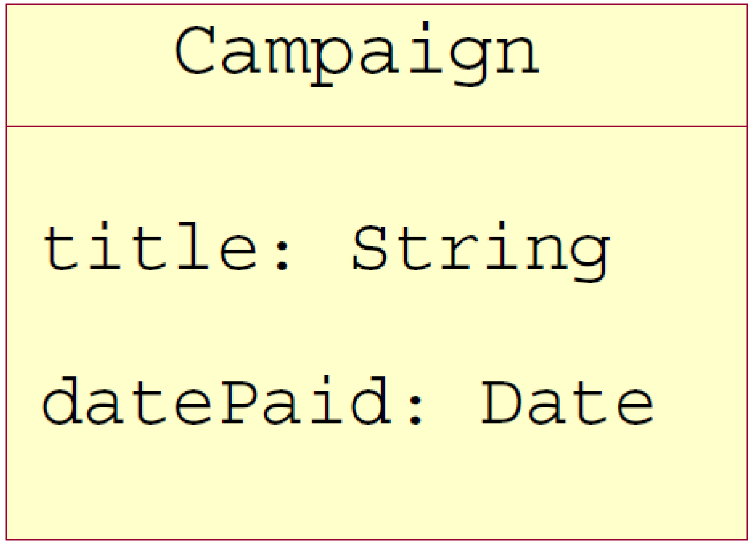
- *Every class must have a unique name*



- *Each class has instances that represent particular individuals that have the properties of the class.*
- *For example, **George, Nazim, Yijun**,… may be instances of **StaffMember**.*
- *Classes can be used to describe a part of the real world, or part of the system under design.*

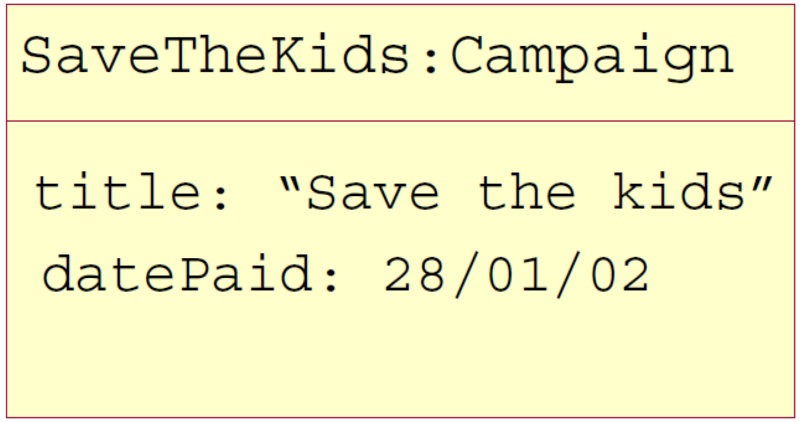
Attributes

- *Each class can have **attributes** which represent useful information about instances of a class.*
- *Each attribute has a **type**.*
- *For example, **`Campaign`** has attributes **`title`** and **`datePaid`**.*



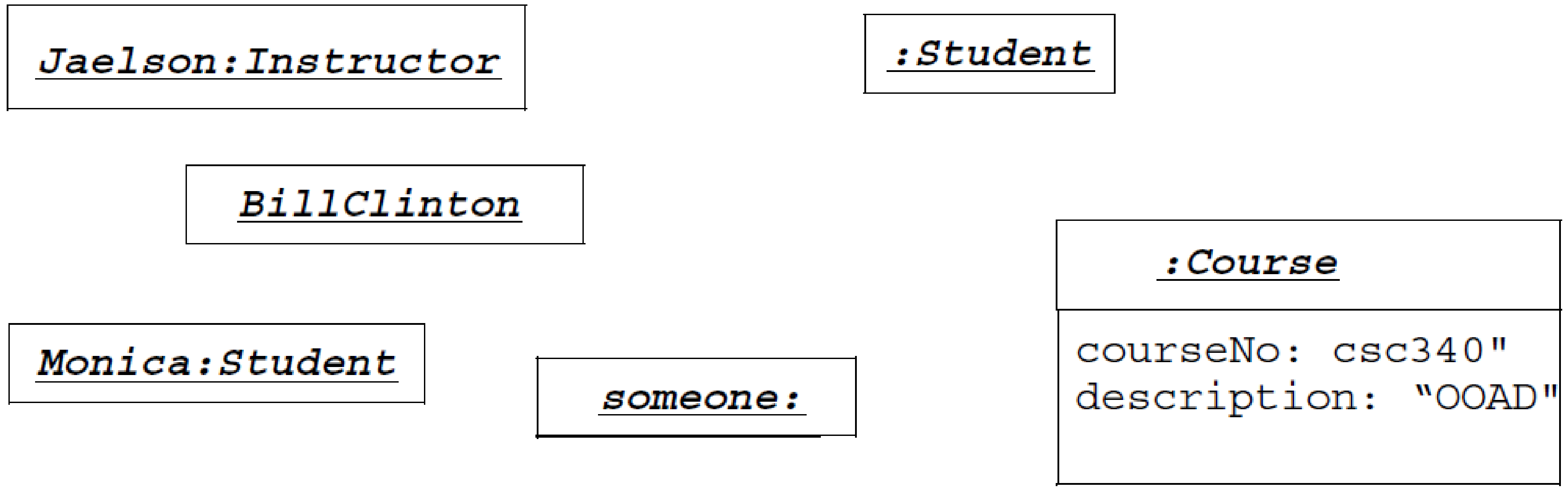
Objects and Their Attribute Values

- *The instances of a class are called **objects**.*
- *Objects are represented as shown below.*
- *Two different objects may have identical attribute values (like two people with identical name and address)*
- *Make sure that attributes are associated with the right class; for example, you don't want to have both* **`managerName`**, **`managerEmp#`** *as attributes of* **`Campaign`**! ***(…Why??)***



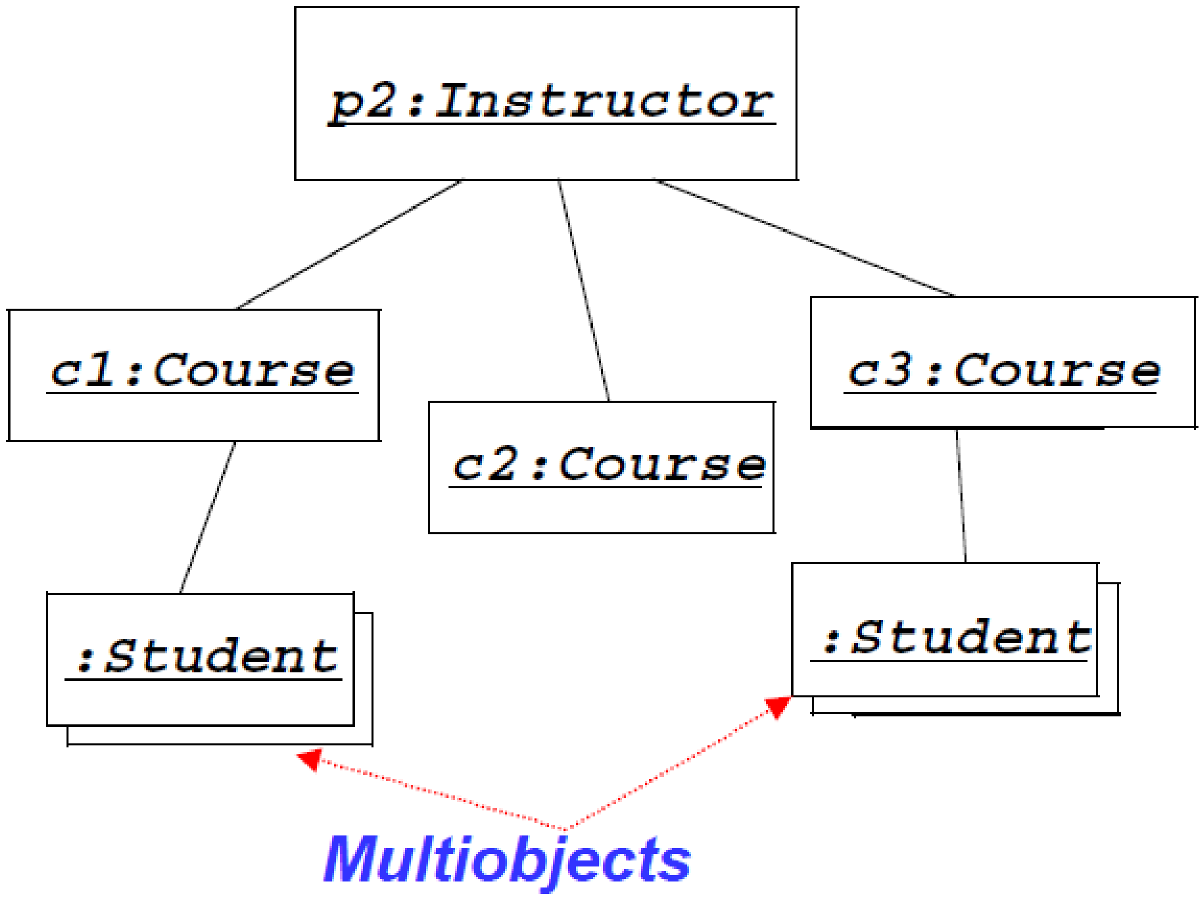
Object Diagrams

- *Model the instances of things described by a class.*
- *Each object diagram shows a set of objects and their interrelationships at a point in time.*
- *Used to model a snapshot of the application.*
- *Each object has an optional name and set of classes it is an instance of, also values for attributes of these classes.*



Multiobjects

*A **multiobject** is a set of objects, with an undefined number of elements*



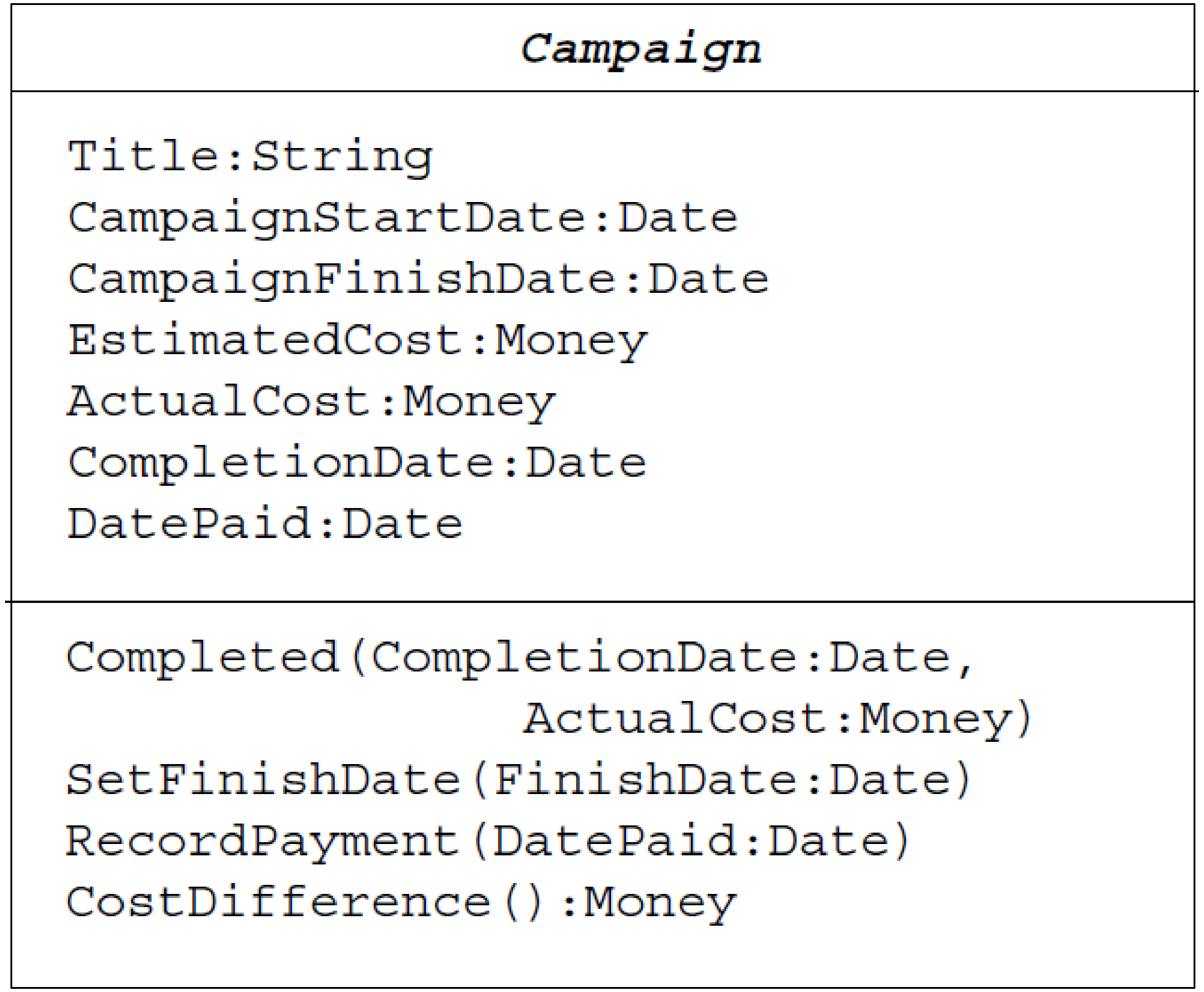
Operations

- *Often derived from action verbs in use case descriptions or problem statements.*
- *Operations describe what can be done with the instances of a class.*
- *For example, For the class **`Stone`**, we may want to associate operations **`Throw(), Kick()`** and **`WriteOn().`***
- *Some operations will carry out processes to change or do calculations with the attributes of an object.*
- *For example, the directors of Agate might want to know the difference between the estimated cost and the actual cost of a campaign*

  *➔**`Campaign`** would need an operation **`CostDifference()`***

Operations

- *Each operation has a* ***signature,*** *which specifies the types of its parameters and the type of the value it returns (if any).*



```
Campaign
---------------------------------
Title:String
CampaignStartDate:Date
CampaignFinishDate:Date
EstimatedCost:Money
ActualCost:Money
CompletionDate:Date
DatePaid:Date
---------------------------------
Completed(CompletionDate:Date,
               ActualCost:Money)
SetFinishDate(FinishDate:Date)
RecordPayment(DatePaid:Date)
CostDifference():Money
```

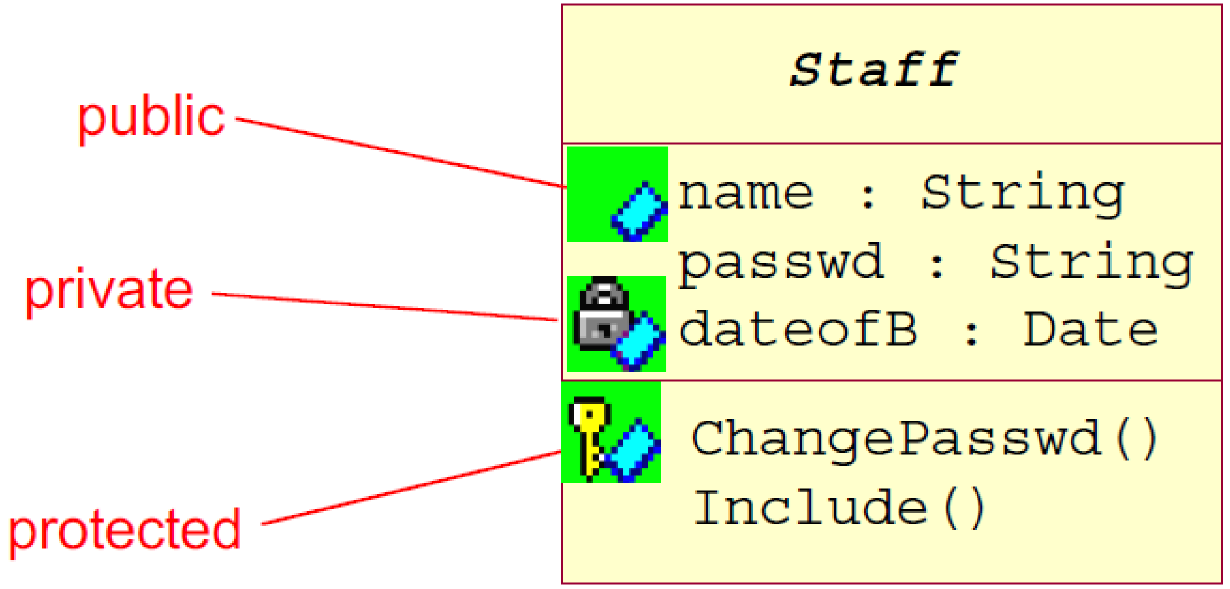
Visibility

- *As with Java, attributes and operations can be declared with different visibility modes:*

  *+ **public**: any class can use the feature (attribute or operation);*

  *# **protected**: any descendant of the class can use the feature;*

  *- **private**: only the class itself can use the feature.*



Relationships

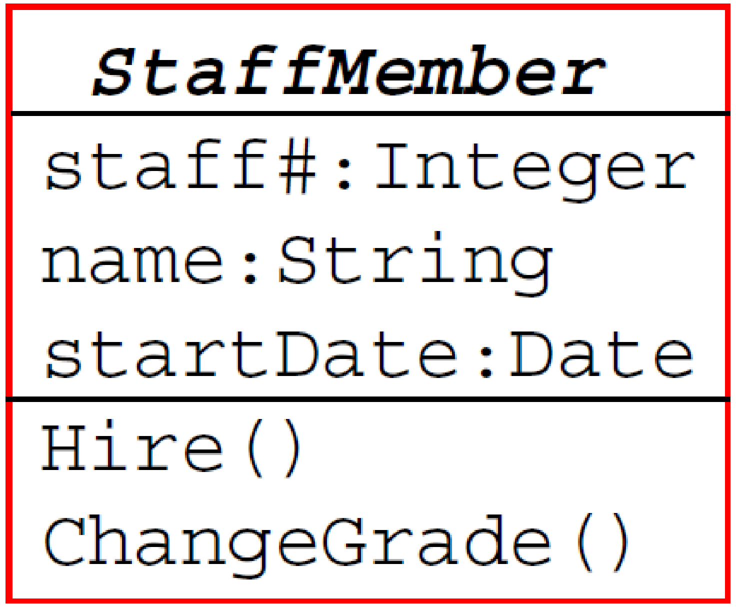
- *Classes and objects do not exist in isolation from one another*
- *A relationship represents a connection among things.*
- *In UML, there are different types of relationships:*
  - ✓ *Generalization*
  - ✓ *Association*
  - ✓ *Aggregation*
  - ✓ *Composition*
  - ✓ *…more…*

Generalization Relationship

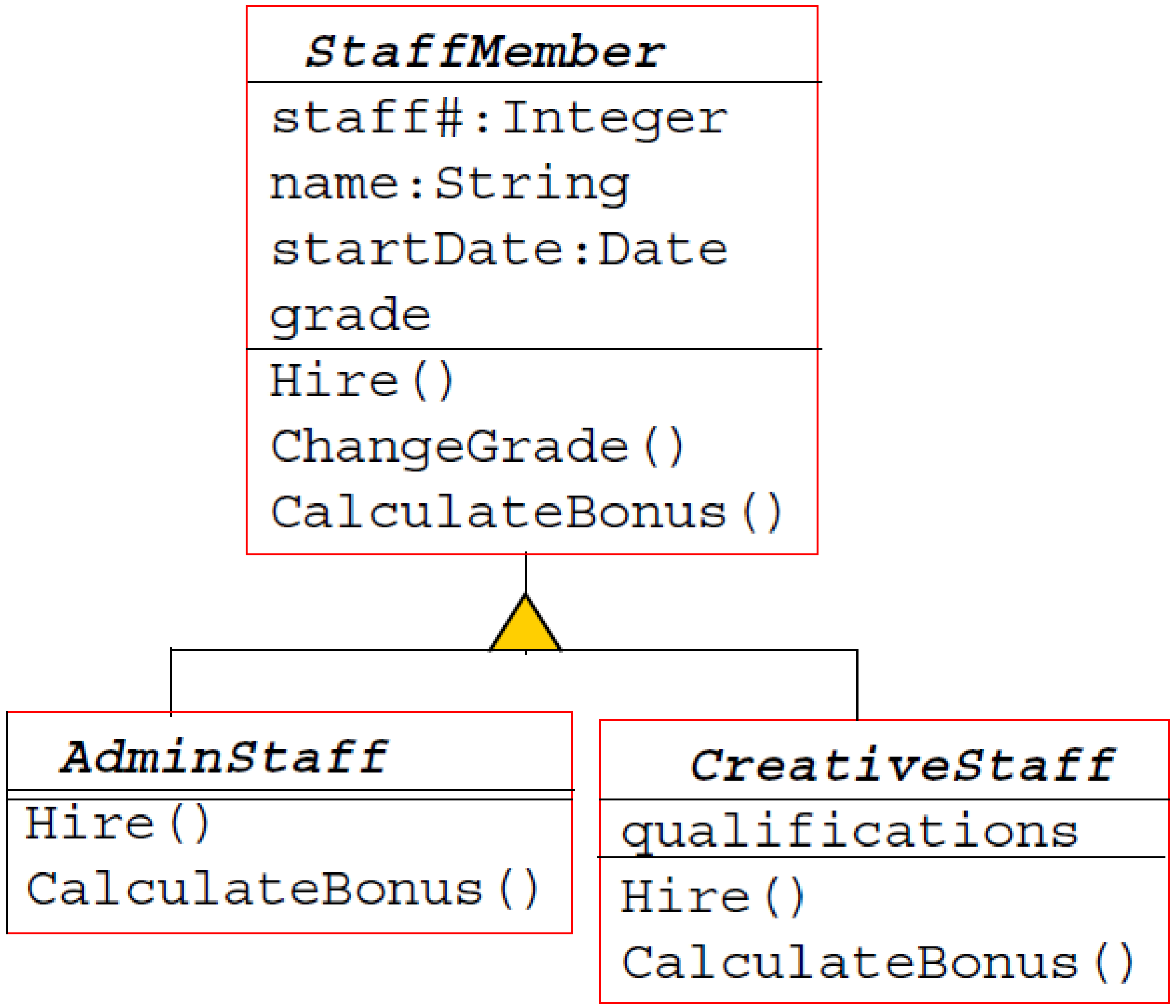
- *Generalization relates two classes when the concept represented by one class is more general than that represented by the other.*
- *For example, **`Person`** is a generalization of **`Student`**, and conversely, **`Student`** is a specialization of **`Person`**.*
- *The more general class participating in a generalization relationship is also called the **superclass** or **parent**, while the more specialized class is called **subclass** or **child.***
- *The child always inherits the structure and behavior of the parent. However, the child may also add new structure and behavior, or may modify the behavior of the parent..*

Generalization

- *It may be that in a system like Agate’s we need to distinguish between different types of staff:*
  - ✓ *creative staff and administrative staff;*
  - ✓ *and to store different data about them.*
- *For example,*
  - ✓ *Administrative staff cannot be assigned to work on or manage a campaign;*
  - ✓ *Creative staff have qualifications which we need to store;*
  - ✓ *Creative staff are paid a bonus based on the work they have done;*
  - ✓ *Administrative staff are paid a bonus based on a percentage of salary.*



Engineering Software



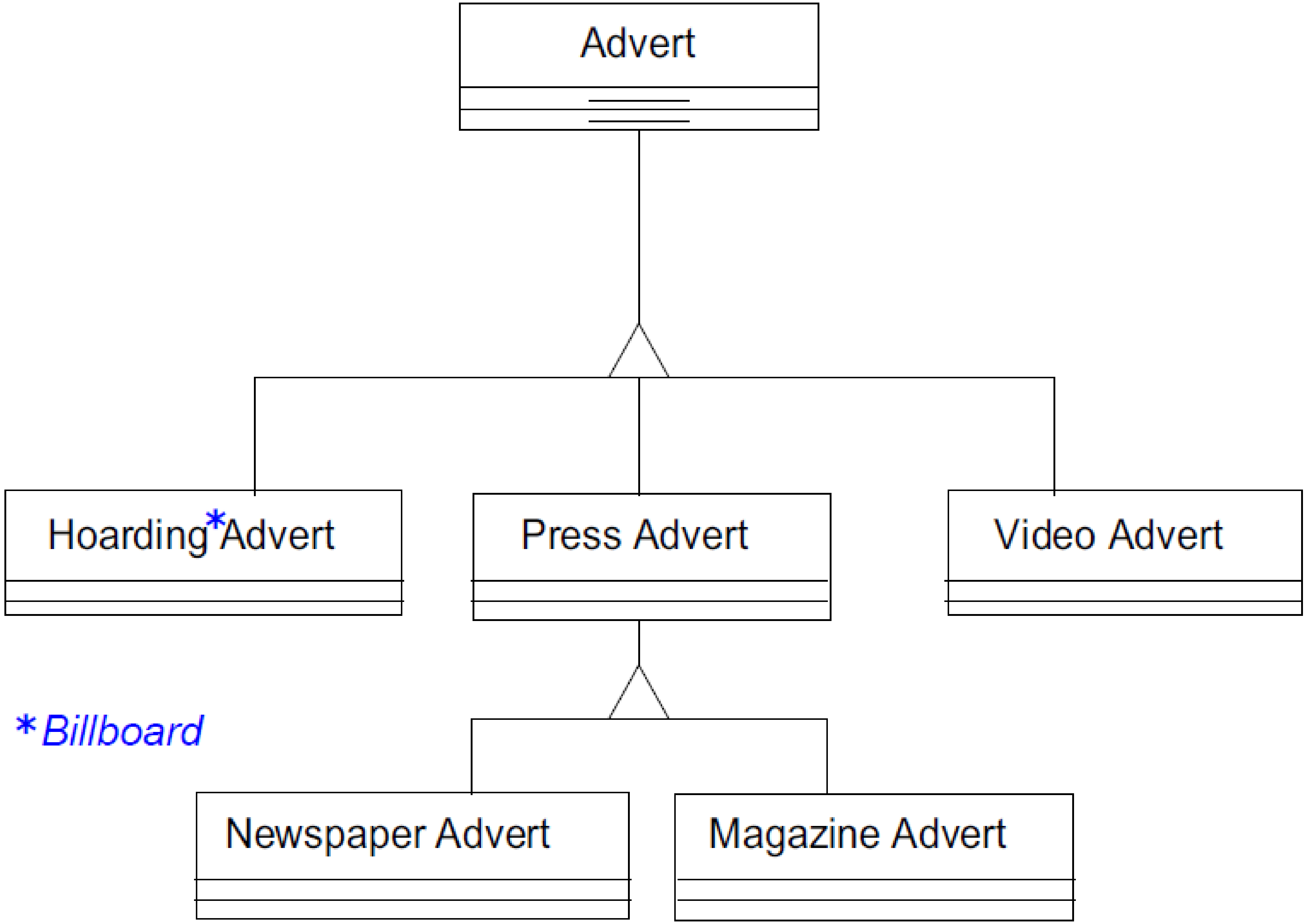
The triangle linking the classes shows inheritance; the connecting line between ***AdminStaff*** and ***CreativeStaff*** indicates that they are mutually exclusive. However, all instances of ***AdminStaff*** and ***CreativeStaff*** will have a ***staff#, name, startDate,*** while ***CreativeStaff*** will also have a ***qualifications*** attribute.

Generalization

- *Similarly, the operation* ***`CalculateBonus()`*** *is declared in* ***`StaffMember`****, but is* ***overridden*** *in each of its sub-classes.*
- *For* ***`AdminStaff`****, the method uses data from* ***`StaffGrade`*** *to find out the salary rate and calculate the bonus.*
- *In the case of* ***`CreativeStaff`****, it uses data from the campaigns that the member of staff has worked on to calculate the bonus.*
- *When the same operation is defined differently in different classes, each class is said to have its own* ***method*** *of defining the operation.*

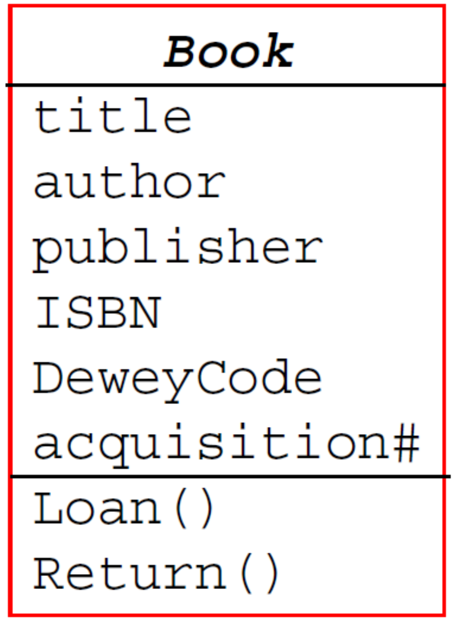
Finding Inheritance

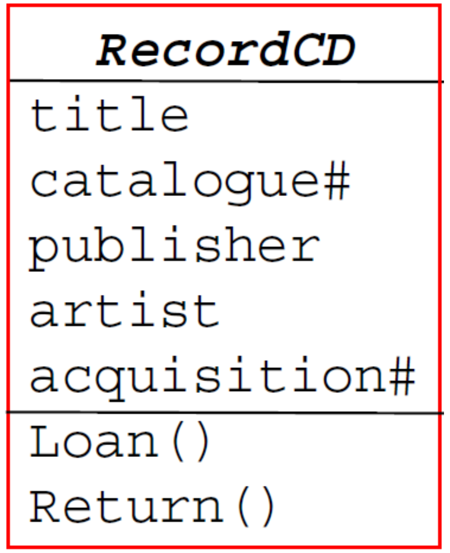
- *Sometimes inheritance is discovered top-down: we have a class, and we realize that we need to break it down into subclasses which have different attributes and operations.*
- *Here is a quote from a director of Agate: "Most of our work is on advertising for the press, that's newspapers and magazines, also for advertising hoardings, as well as for videos."*


Advert
Hoarding*Advert
Press Advert
Video Advert
*Billboard
Newspaper Advert
Magazine Advert

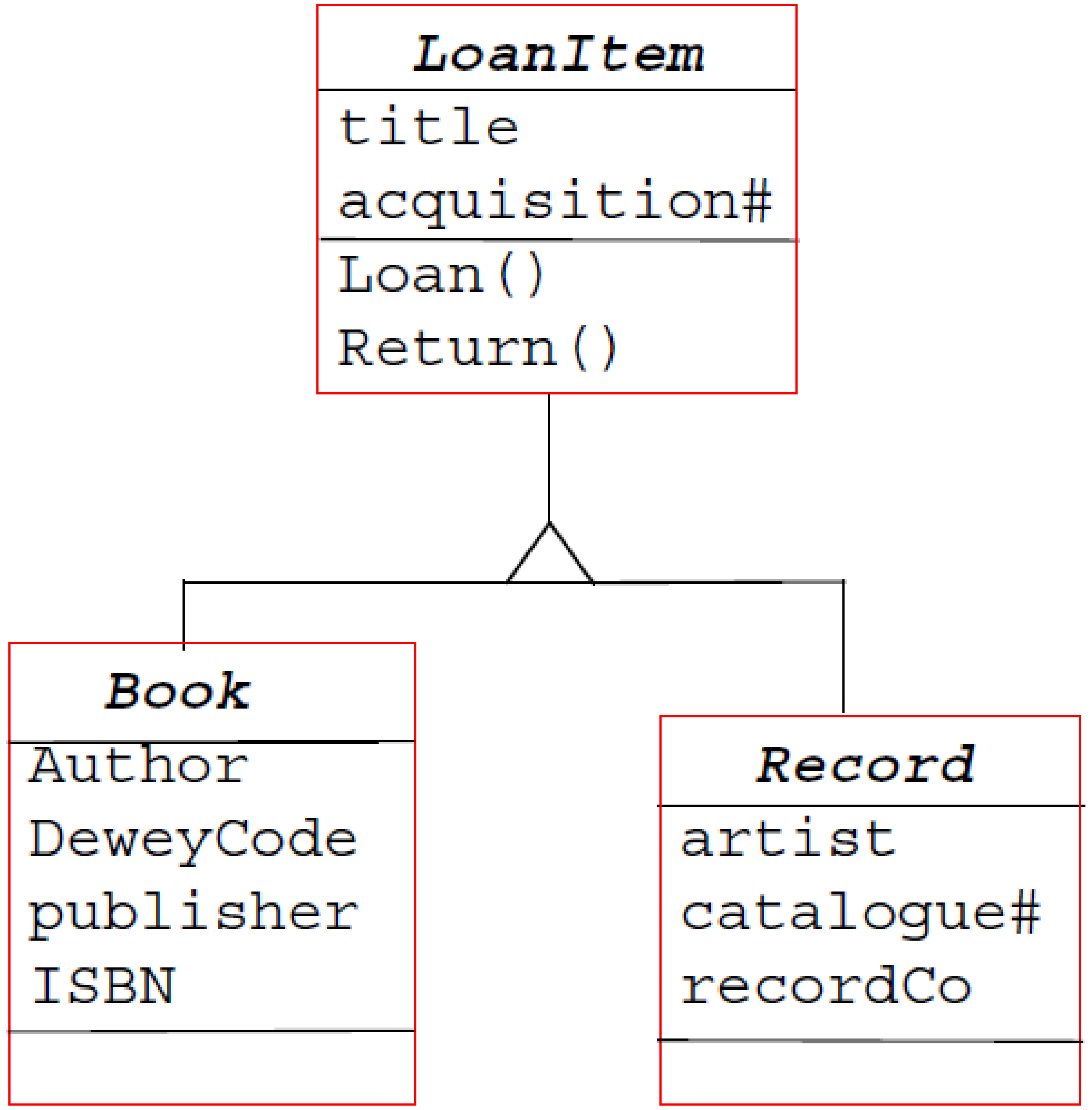
Finding Inheritance

- *Sometimes we find inheritance bottom-up: we have several classes and we realize that they have attributes and operations in common, so we group those attributes and operations together in a common super-class.*
- *Define a suitable generalization of these classes and redraw the diagram.*



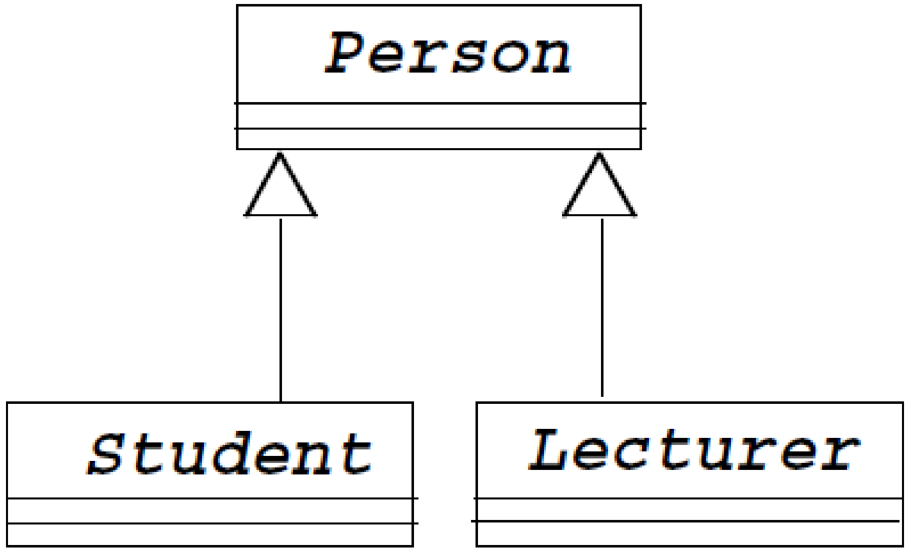


…The Solution…

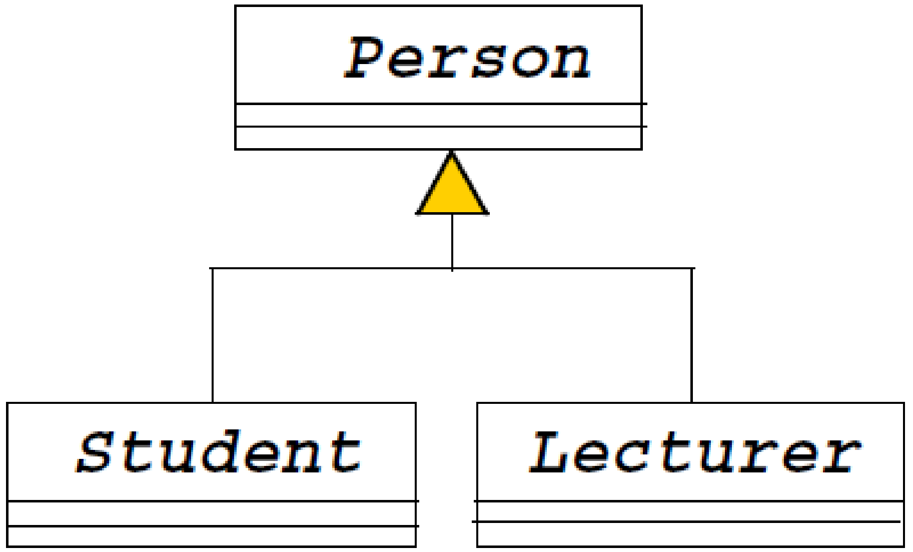


Generalization Notation

Possibly overlapping e.g., `Maria` is both `Lecturer` and `Student`



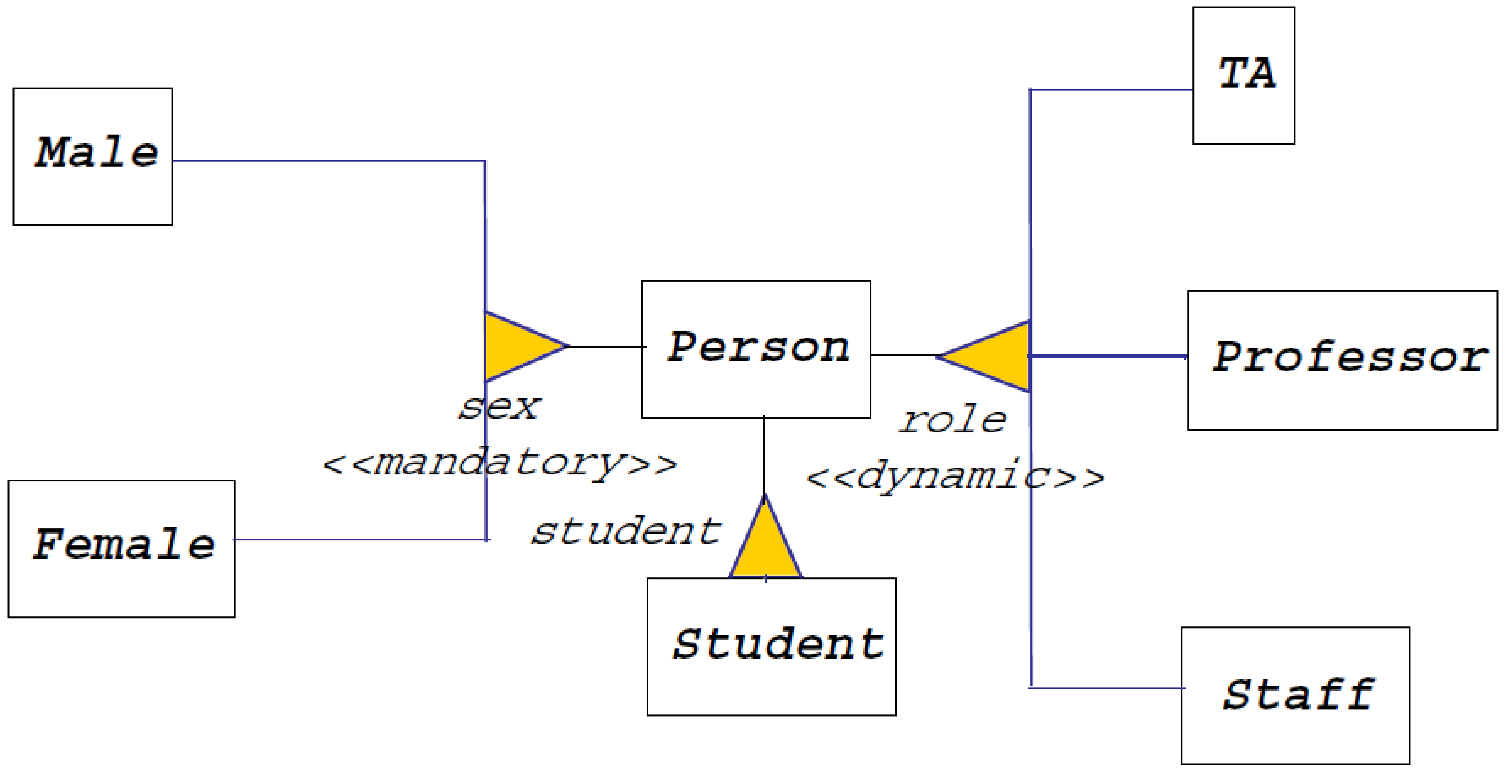
Mutually exclusive i.e., a lecturer can't be a student and vice versa



Multiple and Dynamic Classification

- *Classification refers to the relationship between an object and the classes it is an instance of.*
- *Traditional object models (e.g., Smalltalk, C++,…) assume that classification is **single** and **static**. This means that an object is an instance of a single class (and its superclasses) and this instance relationship can't change during the object's lifetime.*
- ***Multiple** classification allows an object to be an instance of several classes that are not is-a related to each other; for example, **`Maria`** may be an instance of **`GradStudent`** and **`Employee`**.*
- *If you allow multiple classification, you want to be able to specify which combinations of instantiations are allowed. This is done through **discriminators**.*
- ***Dynamic** classification allows an object to change its type during its lifetime.*

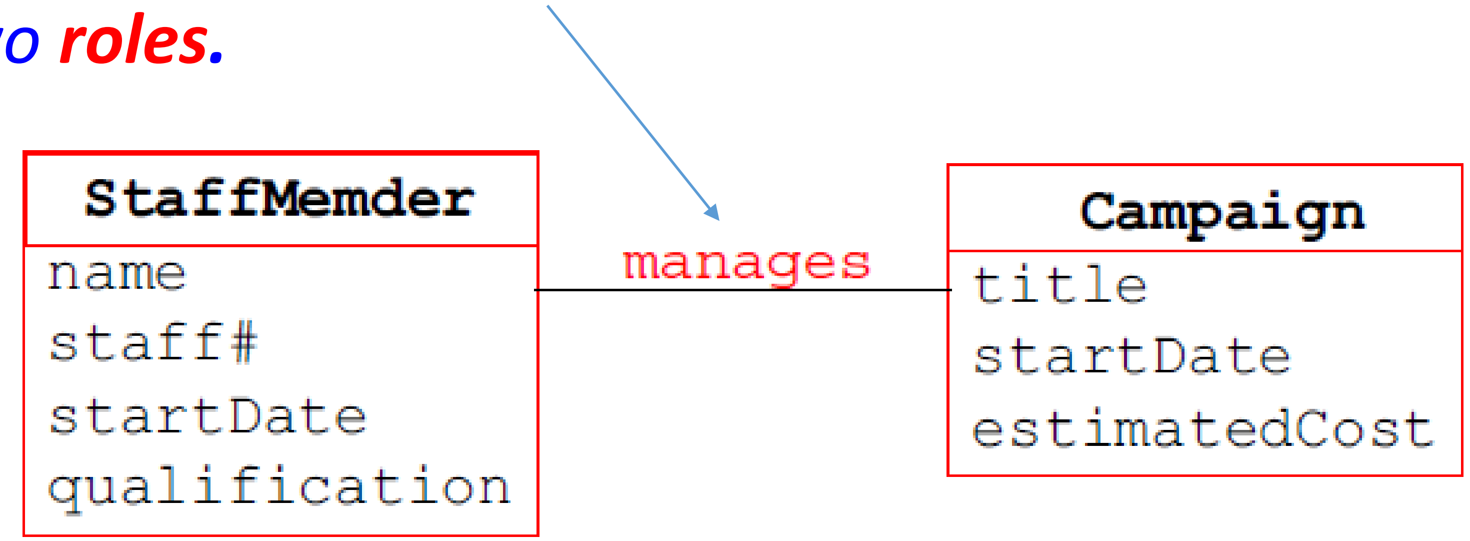
Multiple Classification



- ***Mandatory*** *means that every instance of* ***Person*** *must be an instance of* ***Male*** *or* ***Female****.*
- ***Dynamic*** *means that an object can cease to be a* ***TA*** *and may become a* ***Professor****.*

Association Relationship

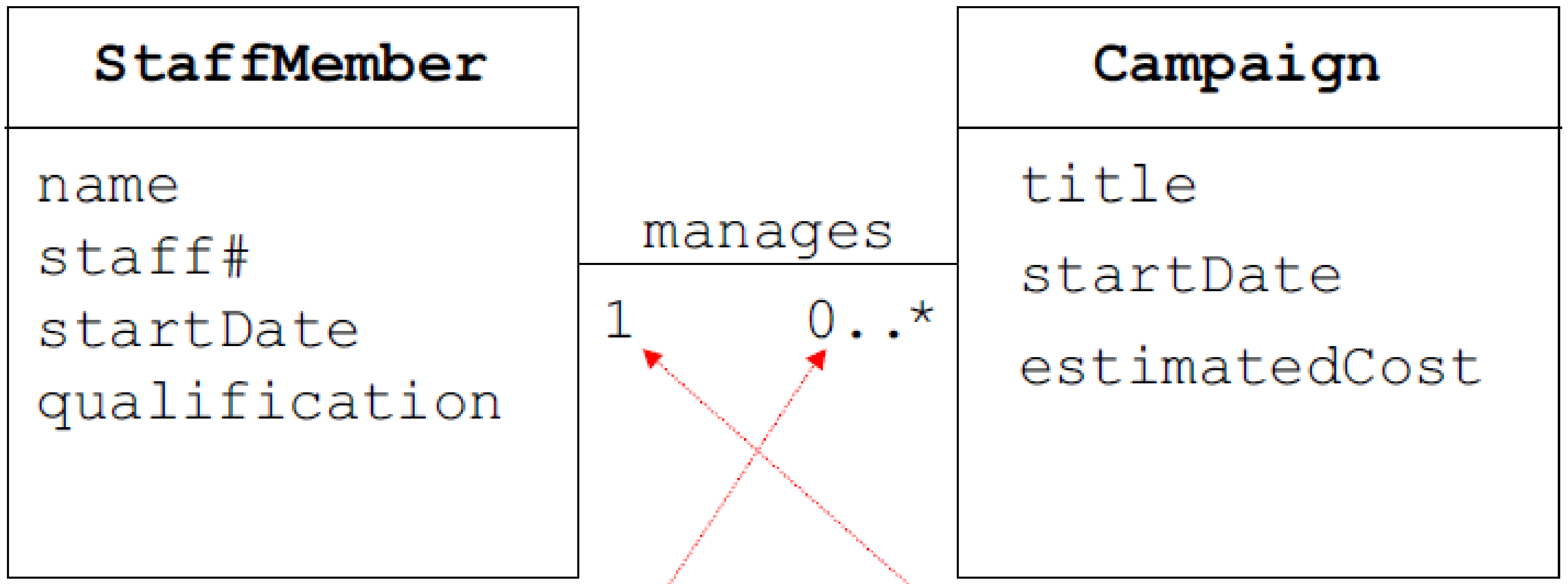
- *An association is a structural relationship which represents a binary relationship between objects.*
- *For example, a person is the child of another person, a car is owned by a person, or, a staff member manages a campaign.*
- *An association has a **name**, and may be specified along with zero, one or two **roles**.*



Association Multiplicity

- *Can a campaign exist without a member of staff to manage it?*
- *If yes, then the association is optional at the **Staff** end - zero or one*
- *If a campaign cannot exist without a member of staff to manage it*
  - ✓ *then it is not optional*
  - ✓ *if it must be managed by one and only one member of staff then we show it like this - exactly one*
- *What about the other end of the association?*
- *Does every member of staff have to manage exactly one campaign?*
- *No. So the correct multiplicity is zero or more.*
  - ✓ *Kerry Dent, a more junior member of staff, doesn't manage any campaigns…*
  - ✓ *Pete Bywater manages two…*

Associations with Multiplicity



"A staff member can manage zero or more campaigns"

"A campaign is managed by exactly one staff member"

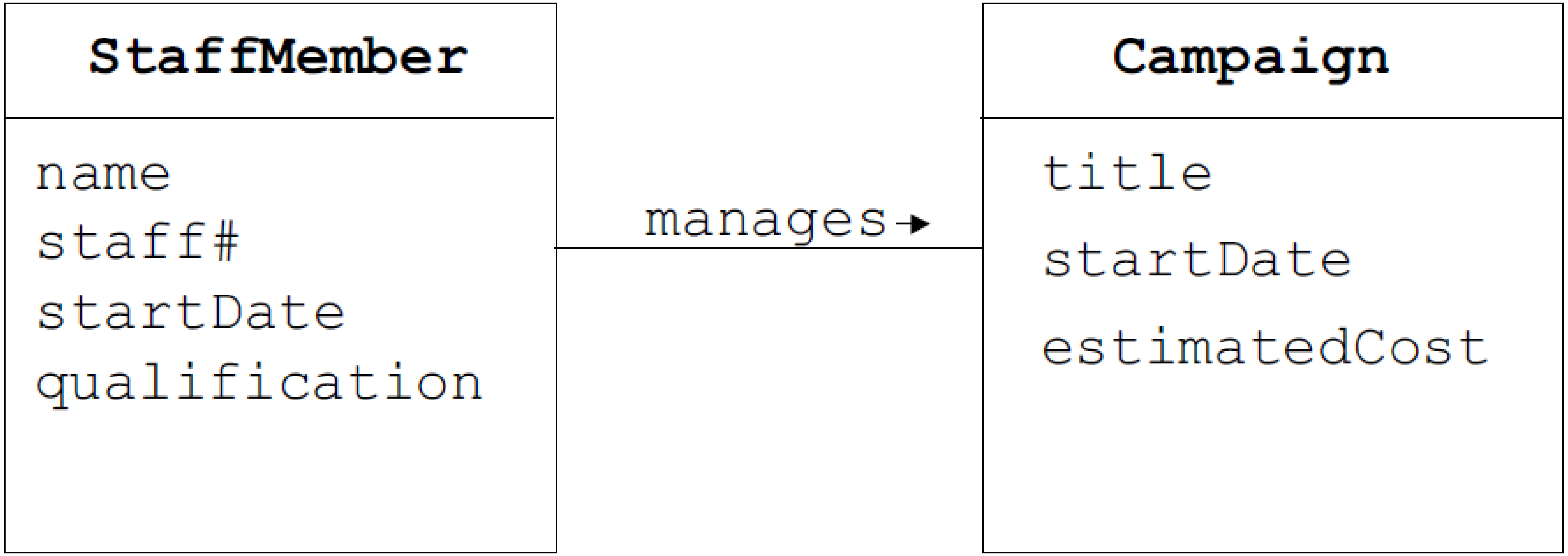
Multiplicity

- *Some examples of specifying multiplicity:*

Optional (0 or 1)	0..1		
Exactly one	1	=	1..1
Zero or more	0..*	=	*
One or more	1..*		
A range of values	1..6		
A set of ranges	1..3,7..10,15,19..*		

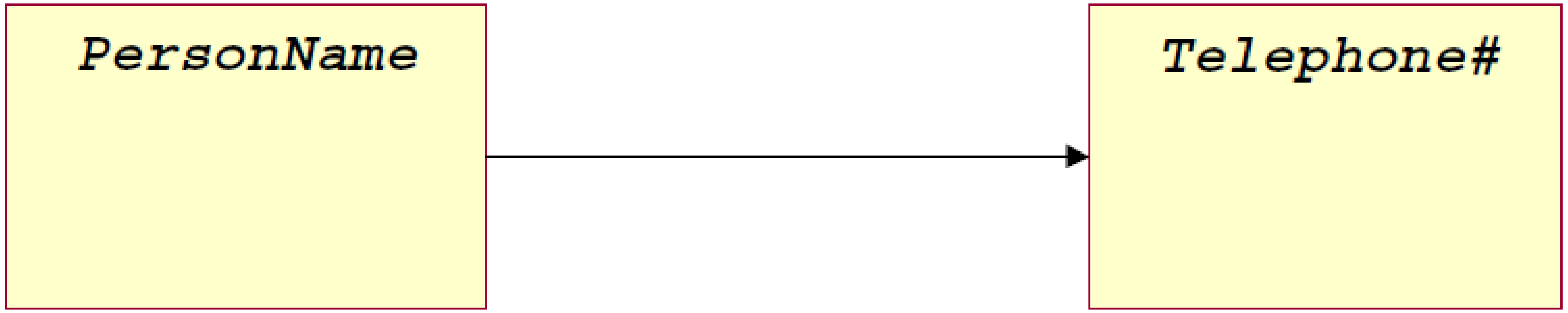
Direction of an Association

- *You can specify explicitly the direction in which an association is to be read. For example,*



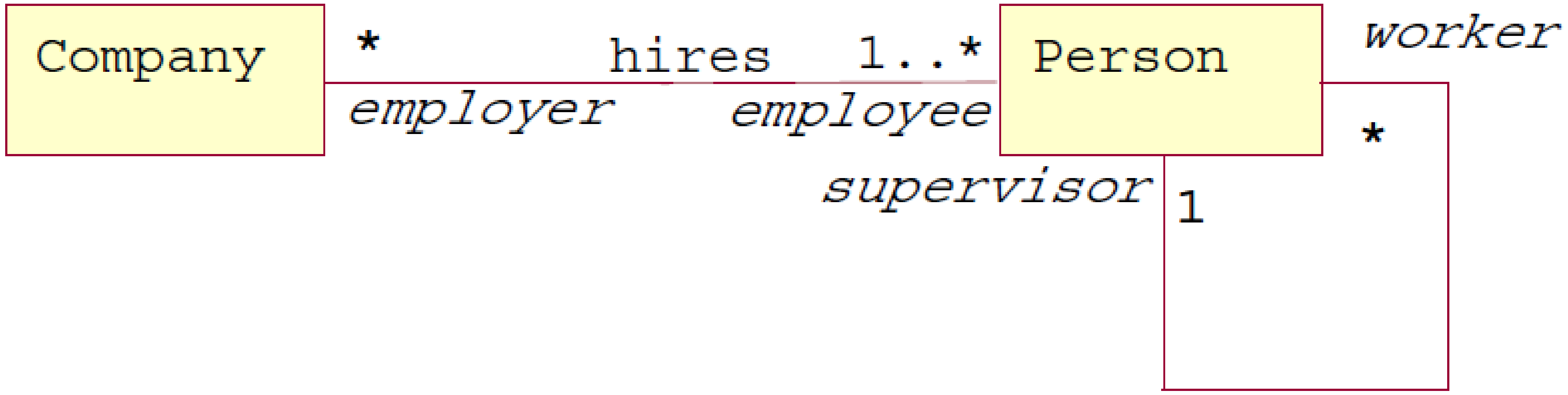
Association Navigation

- *Sometimes we want to model explicitly the fact that an association is uni-directional.*
- *For example, given a person's full name, you can get the person's telephone number, but not the other way around.*



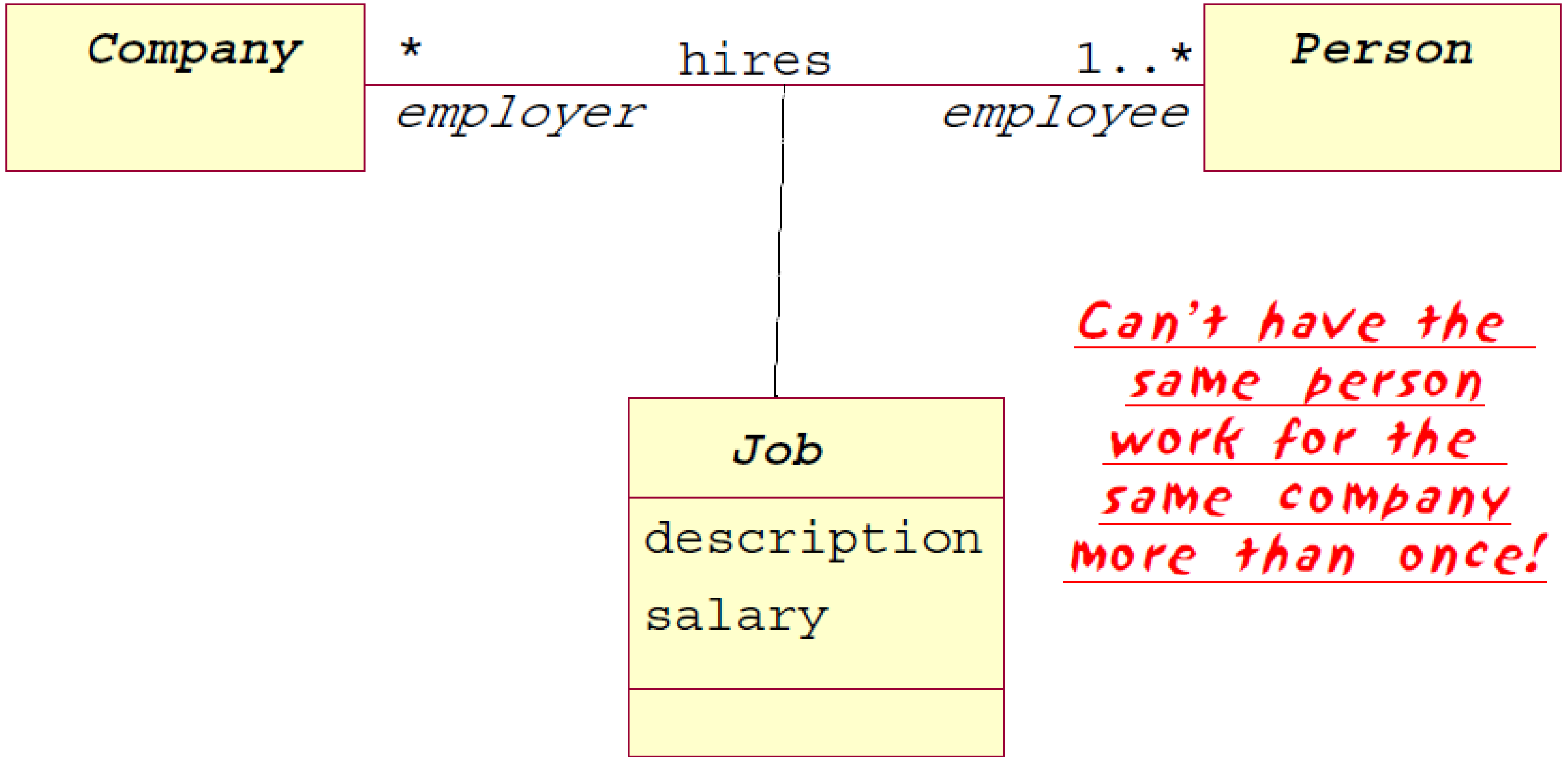
Association and Role

- *We can name explicitly the role a class in an association.*
- *The same class can play the same or different roles in other associations.*



Association Classes

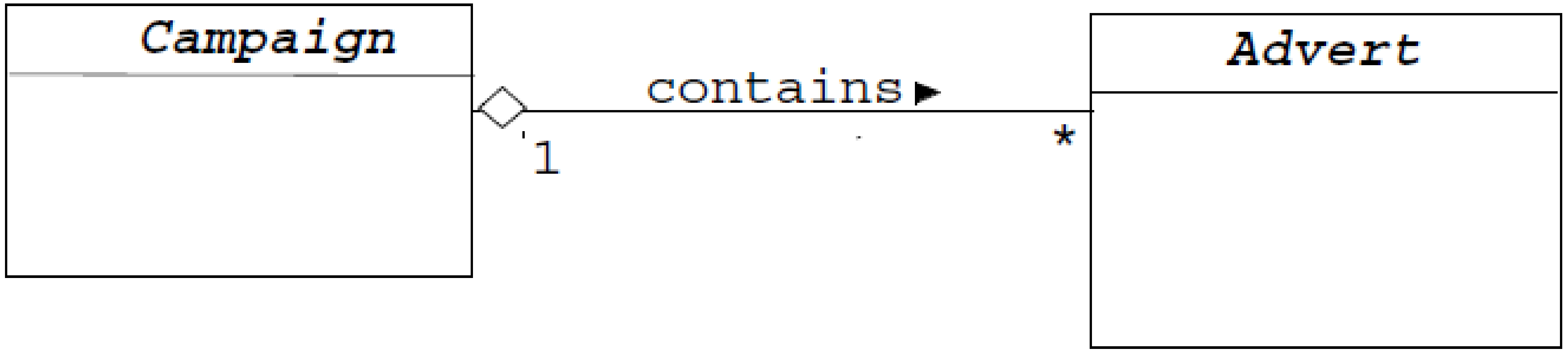
- *Sometimes we want to treat an association between two classes, as a class in its own right, with its own attributes and operations.*



Can't have the same person work for the same company more than once!

Aggregation Relationship

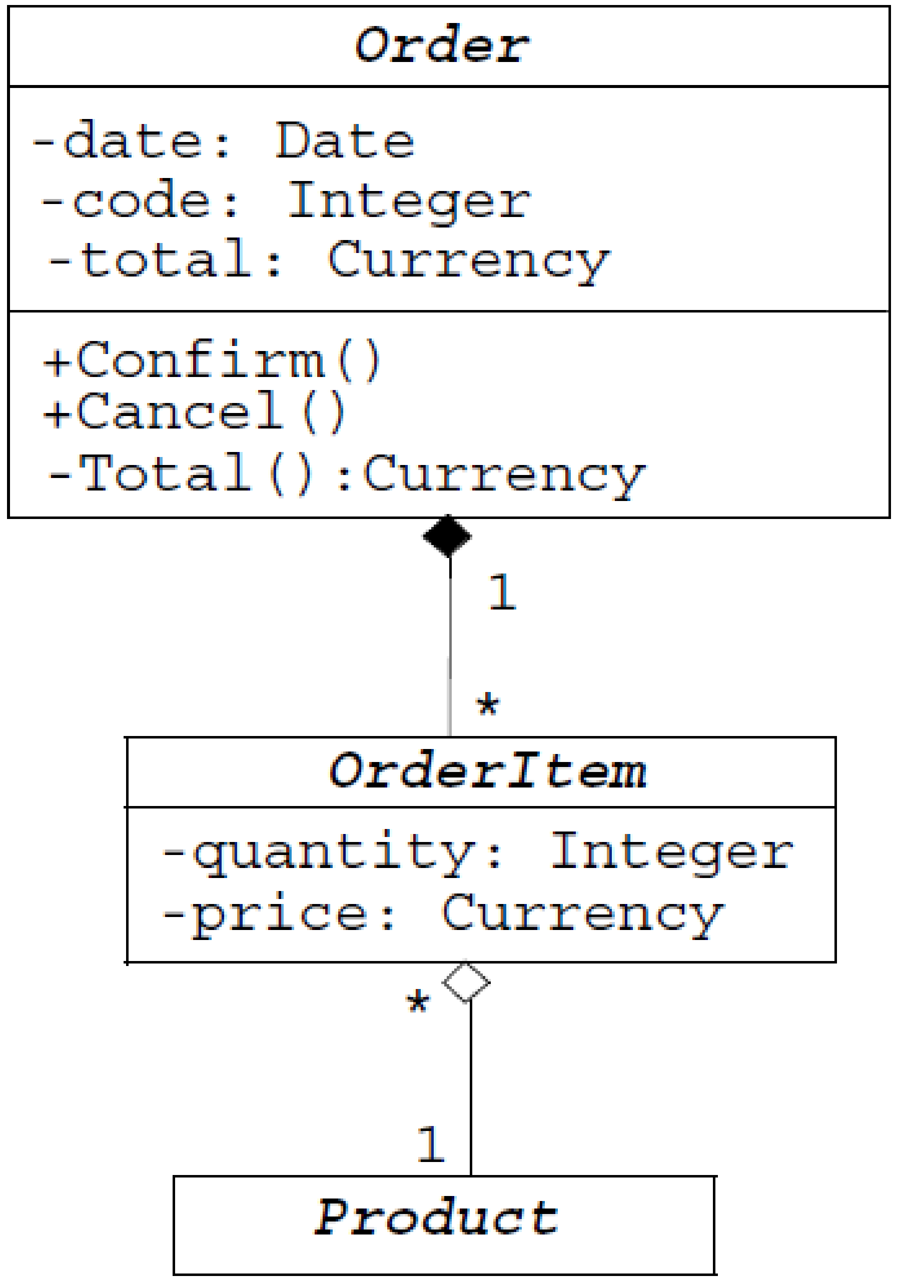
- *This is the **Has-a** or **Whole/part** relationship, where one object is the "whole", and the other (one of) the "part(s)".*



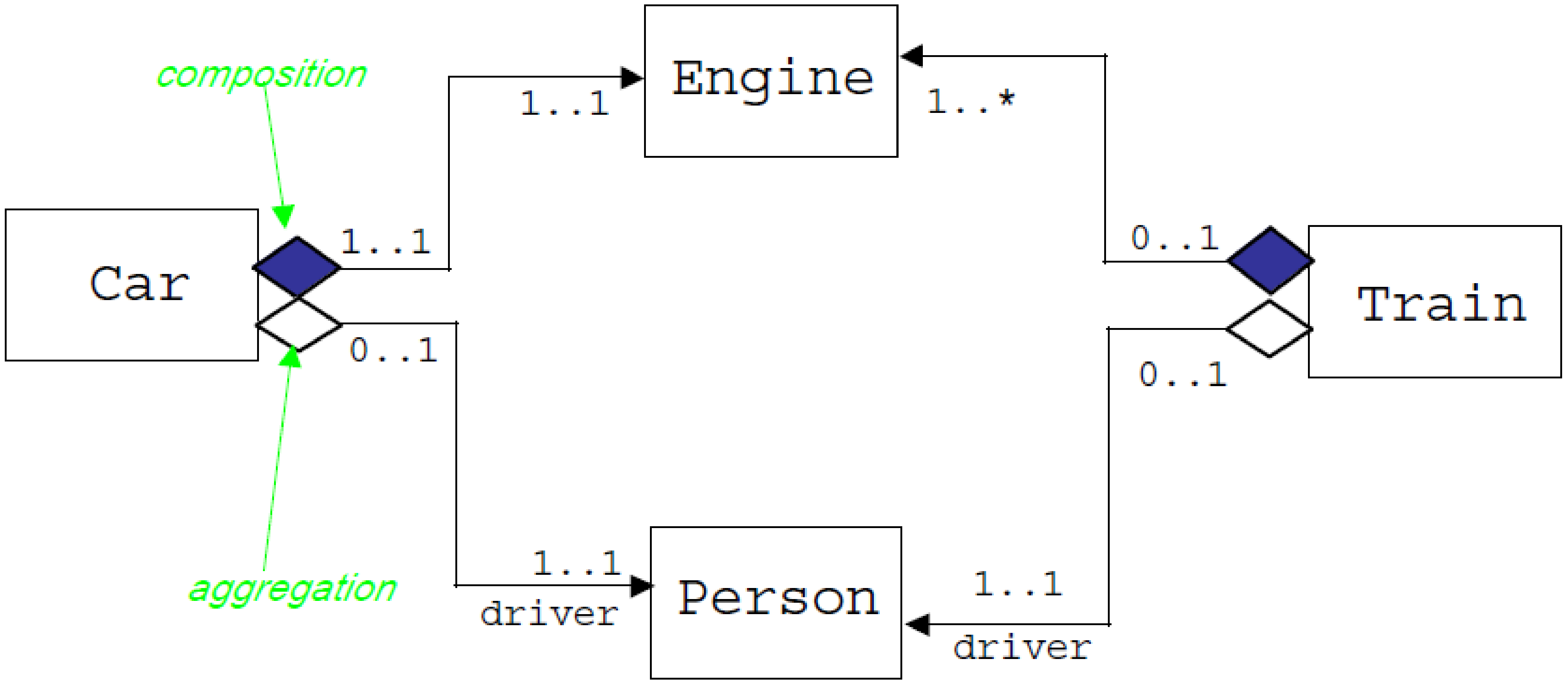
Composition Relationship

- *It is a special case of the aggregation relationship.*
- *A composition relationship implies strong ownership of the part by the whole. Also implies that if the whole is removed from the model, so is the part.*
- *For example, the relationship between a person and her head is a composition relationship, and so is the relationship between a car and its engine.*
- *In a composition relationship, the whole is responsible for the disposition of its parts, i.e. the composite must manage the creation and destruction of its parts.*

An Example

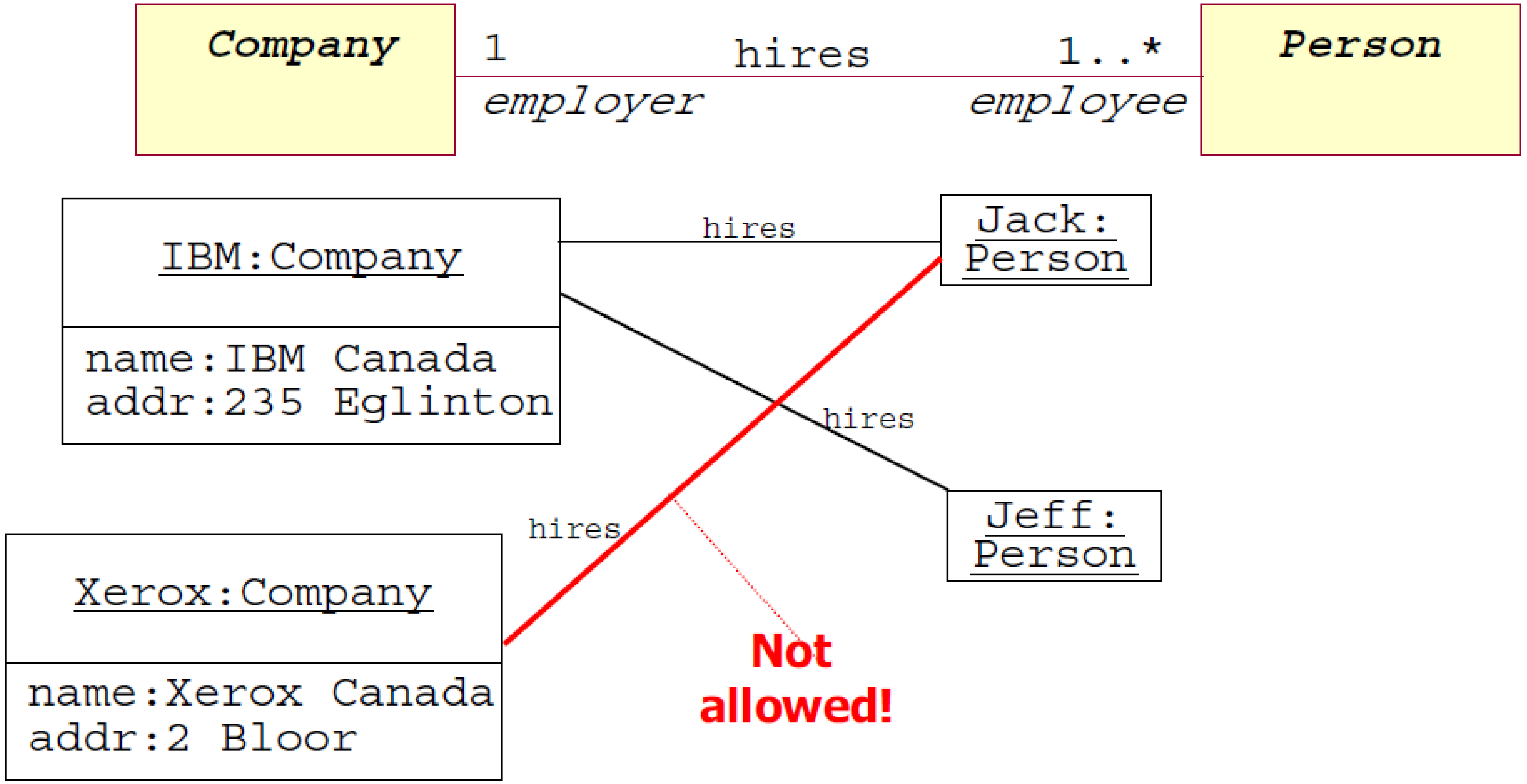

Order
-date: Date
-code: Integer
-total: Currency
+Confirm()
+Cancel()
-Total():Currency
1
*
OrderItem
-quantity: Integer
-price: Currency
*
1
Product

Another Example



Object Diagrams, Again

- *These are like class diagrams, except now we model objects, i.e., instances of the classes defined in class diagrams.*



Business Rules

- *Business rules are used to describe the properties of an application, e.g., the fact that an employee cannot earn more than his or her manager.*
- *A business rule can be:*
  - ✓ *the **description** of a concept relevant to the application (also known as a **business object**),*
  - ✓ *an **integrity constraint** on the data of the application,*
  - ✓ *a **derivation rule**, whereby information can be derived from other information within a class diagram.*

Documentation Techniques

- *Descriptive business rules can be organized into a* ***data dictionary****. This is made up of two tables: the first describes the classes of the diagram, the other describes the associations.*
- *Business rules that describe constraints can be expressed in the following form:*

  <concept> must/must not <expression on concepts>
- *Business rules that describe derivations can be expressed in the following form:*

  <concept> is obtained by <operations on concepts>

Example of a Data Dictionary

Classes	**Description**	**Attributes**	**Identifier**
EMPLOYEE	Employee working in the company.	Code, Surname, Salary, Age	Code
PROJECT	Company project on which employees are working.	Name, Budget, ReleaseDate	Name
….	….	….	….

Associations	**Description**	**Entities involved**	**Attributes**
MANAGEMENT	Associate a manager with a department.	Employee (0,1), Department (1,1)	
MEMBERSHIP	Associate an employee with a department.	Employee (0,1) Department (1,N)	StartDate
….	….	….	….

Examples of Business Rules

Constraints
(BR1) The manager of a department must belong to that department.
(BR2) An employee must not have a salary greater than that of the manager of the department to which he or she belongs.
(BR3) A department of the Rome branch must be managed by an employee with more than 10 years' employment with the company.
(BR4) An employee who does not belong to a particular department must not participate in any project.
....
Derivations
(BR5) The budget for a project is obtained by multiplying the sum of the salaries of the employees who are working on it by 3.
....

Communication and Collaboration Between Objects

- *Communication and collaboration among objects is a fundamental concept for object-orientated software.*
- *We want to decide which objects are responsible for what (within or without the system).*
- *In addition, we want to know how external users and external systems ("actors") interact with each other and the system.*
- *As well, it is often convenient to model interactions between actors; for example, the interactions between actors carrying out a business process.*

Object Interaction and Collaboration

- *Objects "own" information and behaviour, defined by operations; system objects contain data and methods which are relevant to their own **responsibilities**. They don't "know" about other objects' information, but can ask for it.*
- *To carry out business processes, objects (system or otherwise) have to work together, I.e., collaborate.*
- *Objects collaborate by sending messages to one another thereby calling operations of the other object.*
- *Objects can only send messages to one another if they "know" each other, I.e., there is an association between them.*
- *A **responsibility** is high level description of something instances of a class can do. A responsibility reflects the knowledge or information that is available to that class, either stored within its own attribute or requested via collaboration with other classes.*

VIN -- Very Important Note

- *During requirements, the system is modelled in terms of a small number of coarse-grain classes and objects which describe how the system interacts with its environment.*
- *During design, the system is modelled in greater detail in terms of many fine-grain classes and objects.*
- *To keep things clear, we will use icons to represent external objects and actors, and boxes to represent system objects.*

Responsibilities

- *It makes sense to distribute responsibility evenly among classes.*
- *For external classes, this means simpler, more robust classes to define and understand*
- *For system classes, this means:*
  - ✓ *No class is unduly complex;*
  - ✓ *Easier to develop, to test and maintain classes;*
  - ✓ *Resilient to change in the requirements of a class;*
  - ✓ *A class that it relatively small and self-contained has much greater potential for reuse.*
- *A nice way to capture class (object) responsibilities is in terms of **Class-Responsibility-Collaboration** (CRC) cards.*
- *CRC cards can be used in several different phases of software development.*
- *For now, we use them to capture interactions between objects and actors.*

Role Play with CRC Cards

- *During requirements analysis we can spend time role playing with CRC cards to try to sort out the responsibilities of objects and actors and to determine which are the other objects they need to collaborate with in order to carry out those responsibilities.*
- *Often the responsibilities start out being vague and not as precise as the operations which may only become clear as we move into design.*
- *Sometimes we need to role play the objects in the system and test out the interactions between them.*

I’m a Campaign ….

“I’m a Campaign. I know my title, start date, finish date and how much I am estimated to cost. “

“When I’ve been completed, I know how much I actually cost and when I was completed. I can calculate the difference between my actual and estimated costs.”

“When I’ve been paid for, I know when the payment was made.”

“I can calculate the contribution made to me by each member of staff who worked on me.”

This could be an external object

(call it “campaign project”)

or a system object!

I'm a CreativeStaff …

"I'm a CreativeStaff. I know my staff no, name, start date and qualification."

"I can calculate how much bonus I am entitled to at the end of the year."

Does it make sense to include

"I can calculate the contribution made to each campaign I have worked on by each member of staff who worked on it.",

or does that belong in ***`Campaign`****?*

Class: *Campaign*

Responsibilities:	Collaborating Classes
Title	
StartDate	
FinishDate	
EstimatedCost	
ActualCost	
CompletionDate	
DatePaid	
AssignManager	*CreativeStaff*
RecordPayment	
Completed	
GetCampaignContribution	
CostDifference	

Class: *CreativeStaff*

Responsibilities:	Collaborating Classes
StaffNo	
StaffName	
StaffStartDate	
Qualification	
CalculateBonus	*Campaign*
ChangeGrade	*StaffGrade*
	Grade

Additional Readings

- *[Booch99] Booch, G. et al. The Unified Modeling Language User Guide, Addison-Wesley, 1999. (Chapters 4, 5, 8, 9, 10.)*
- *[Fowler97] Fowler, M. Analysis Patterns: Reusable Object Models, Addison-Wesley, 1997.*
- *[Bellin97] Bellin, D et al. The CRC Card Book. Addison-Wesley, 1997.*