UNIVERSITY of GUELPH

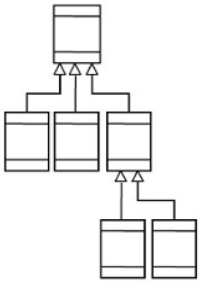
ENGINEERING

UML Class Diagrams

Fattane Zarrinkalam

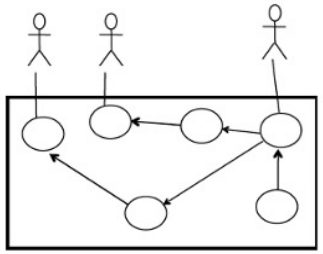
Fowler: UML Distilled: A Brief Guide to the Standard Object Modeling Lang. 3rd Ed. (Chapter 3 & 5)

UML Diagrams



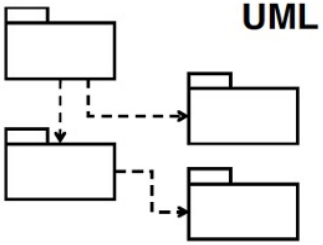
UML Class Diagrams

- information structure
- relationships between data items
- modular structure for the system



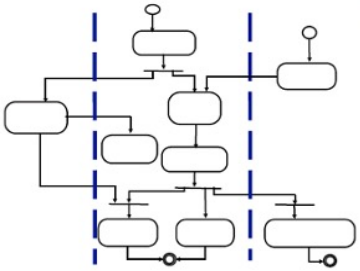
Use Cases

- user's view
- Lists functions
- visual overview of the main requirements



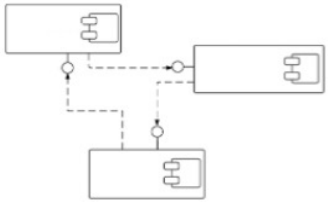
UML Package Diagrams

- Overall architecture
- Dependencies between components



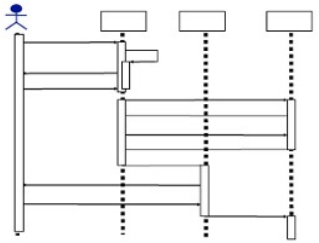
Activity diagrams

- business processes;
- concurrency and synchronization;
- dependencies between tasks;



UML Component Diagrams

- static implementation view of a system
- Interfaces between components



UML Sequence Diagrams

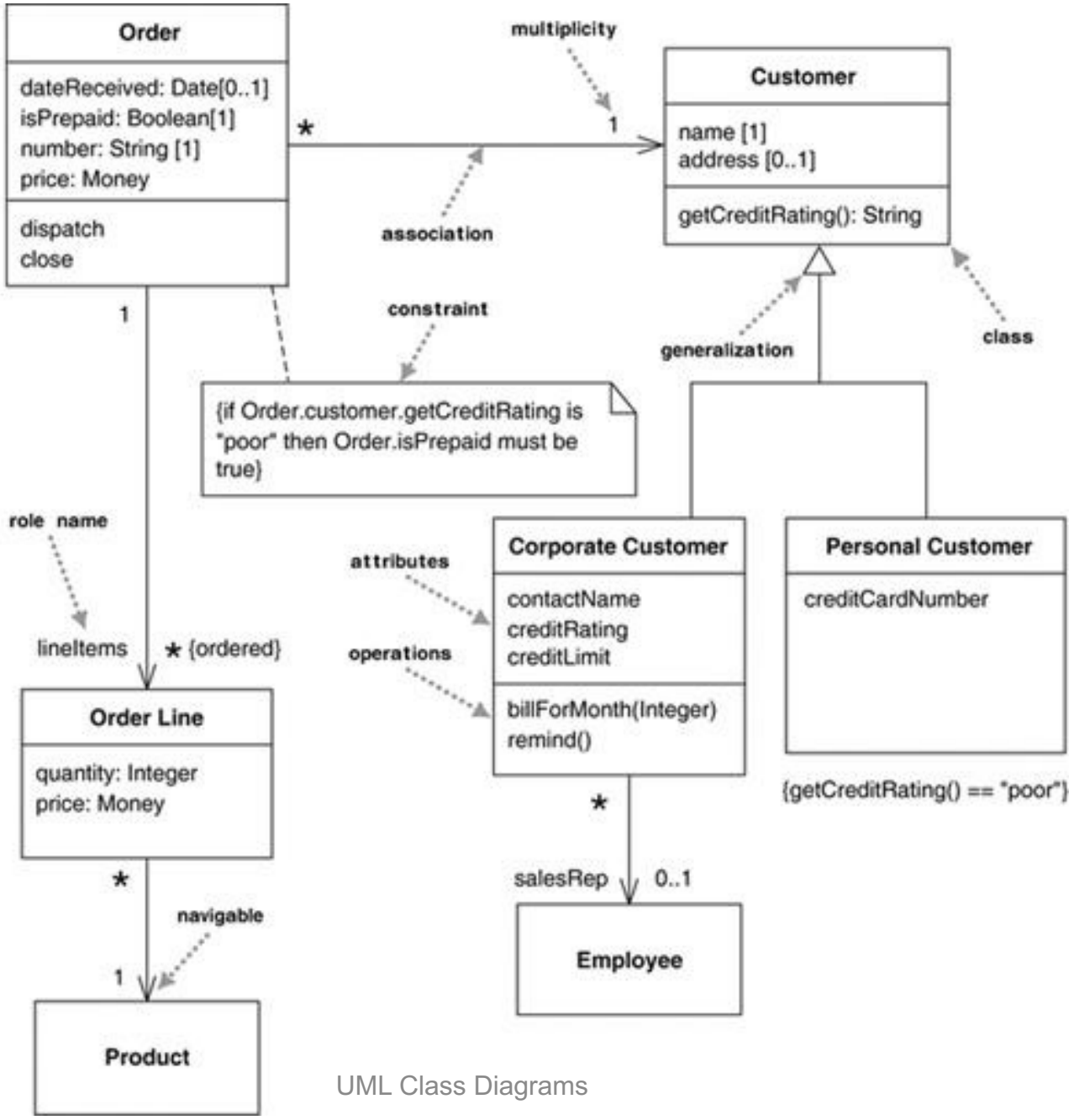
- individual scenario
- interactions between users and system
- Sequence of messages

Class Diagram

- A class diagram describes
  - types of objects in the system and the
  - various kinds of static relationships that exist among them.
  - the properties and operations of a class
  - the constraints that apply to the way objects are connected.

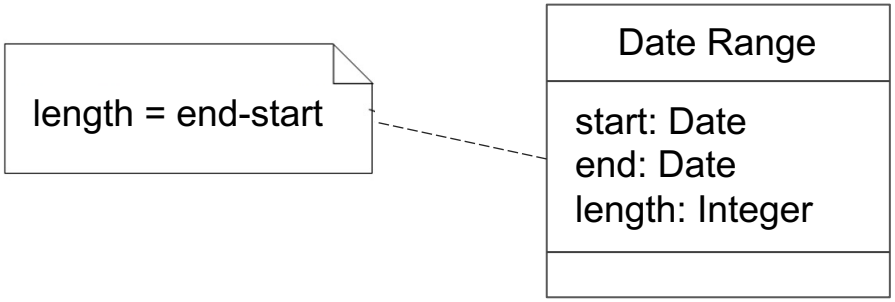
- The UML uses the term feature as a general term that covers properties and operations of a class.

A simple class diagram



Annotations

- Comments
  - can be used to add comments within a class description
- Notes



{length must not be more than three months}

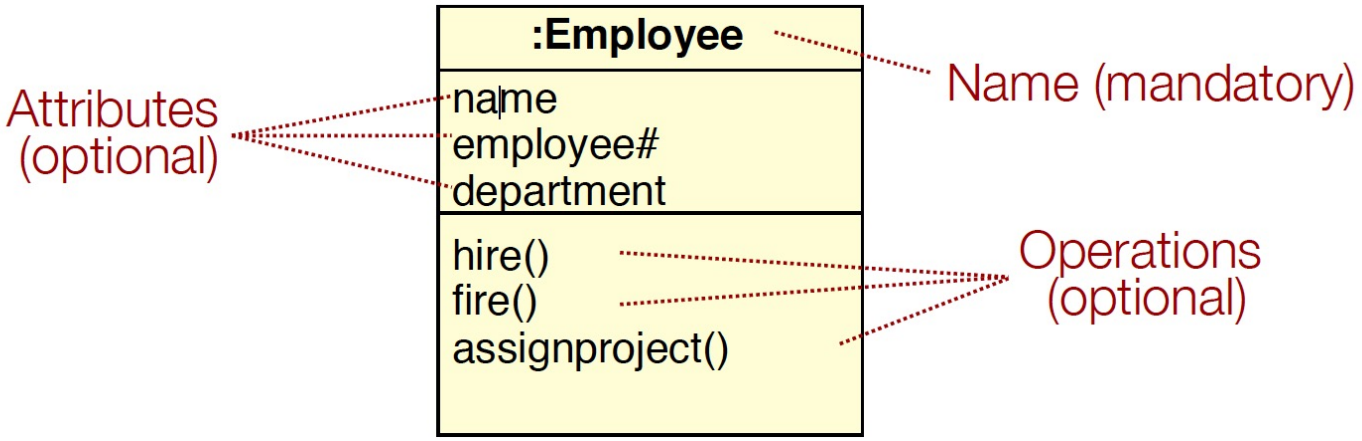
- Constraint Rules
  - Any further constraints {in curly braces}

Classes

Class

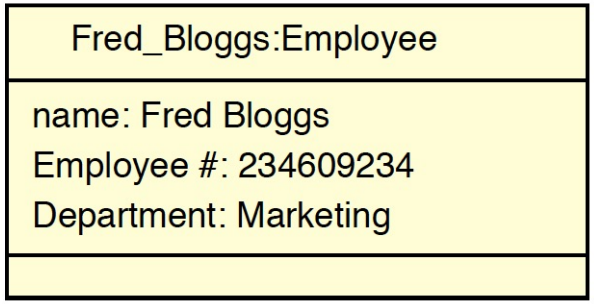
- A class describes a group of objects with common structure and behaviour.

- Examples
  - Employee: has a name, employee # and department; an employee is hired, and fired; an employee works in one or more projects



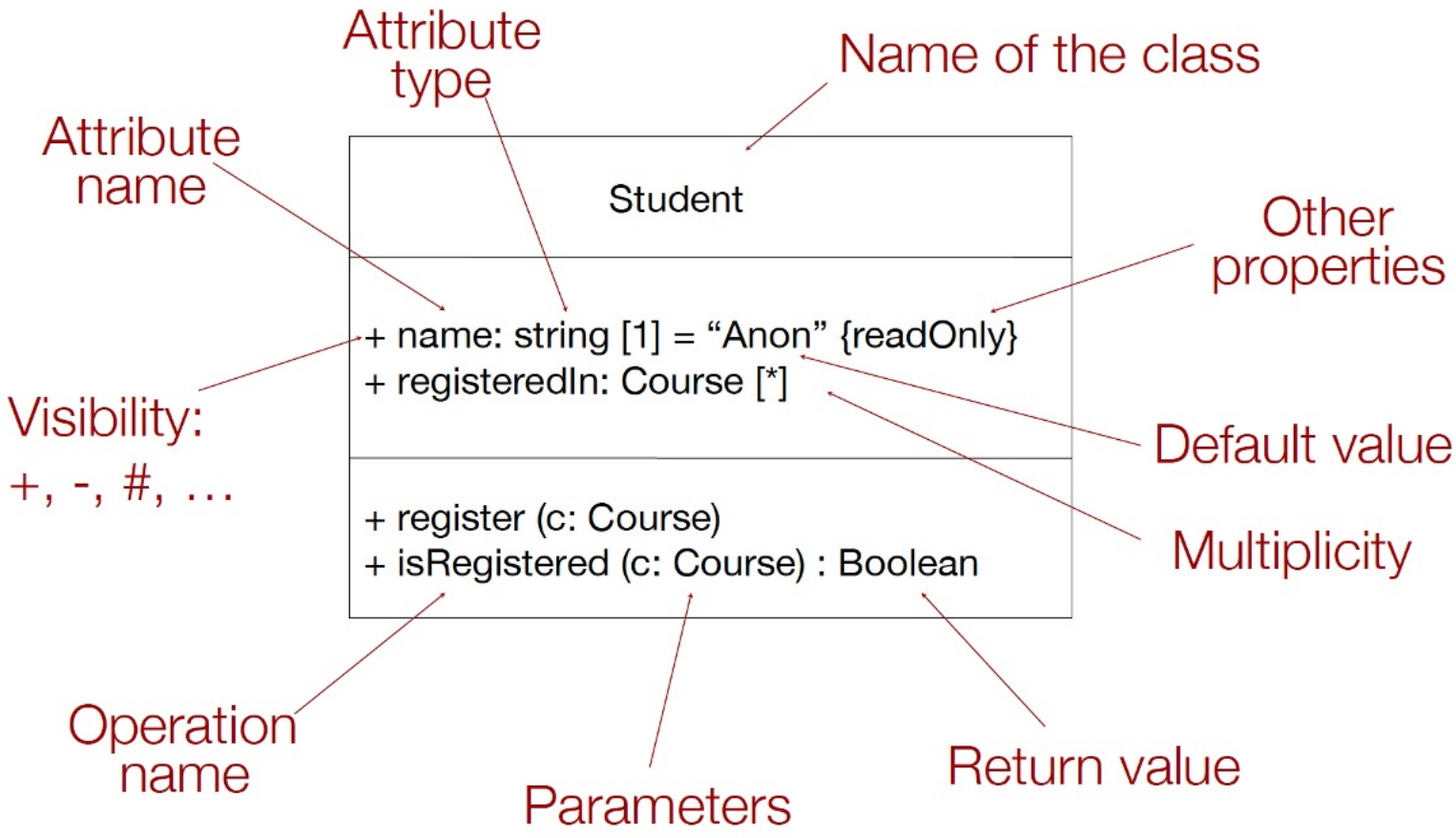
Objects vs. Classes

- The instances of a class are called objects.
  - Objects are represented as:

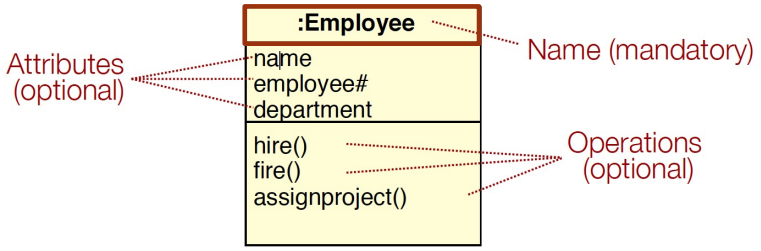


  - Two different objects may have identical attribute values (like two people with identical name and address)

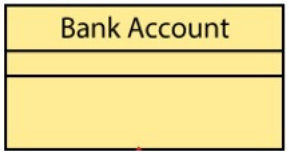
Anatomy of a Class

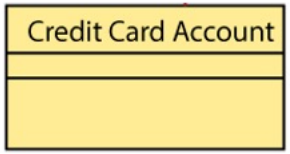


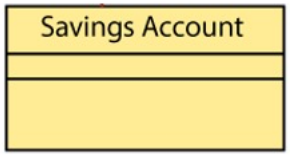
Name of the Class



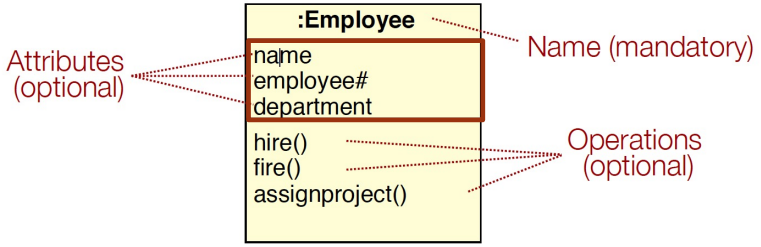
- Appears in the first partition of the class
- The only mandatory information in a class







Attributes



- An attribute is what an object of a class knows.
- An attribute is expressed using the following UML syntax:

visibility name: type multiplicity = default {property-string}

Two examples of this is:

- name: String [1] = ”Anon" {readOnly}
+ registeredIn: Course [*]

- Only the name is necessary.

Attributes :: Visibility

visibility name: type multiplicity = default {property-string}

- Is optional
- Has no default value
- Indicates whether the attribute is accessible from outside the class.

- It may be one of the following:
  - + Public visibility; the attribute is accessible from outside its class.
  - - Private visibility; the attribute is inaccessible from outside its class.
  - # Protected visibility; the attribute is accessible by classes that have a generalization relationship to its class but is otherwise inaccessible from outside its class.

- name: String [1] = ”Anon" {readOnly}
+ registeredIn: Course [*]

Attributes :: Name

visibility name: type multiplicity = default {property-string}

- Mandatory
- How the class refers to the attribute—roughly corresponds to the name of a field in a programming language.

- name: String [1] = ”Anon" {readOnly}
+ registeredIn: Course [*]

Attributes :: Type

visibility name: type multiplicity = default {property-string}

- Is optional
- Has no default value
- Indicates the type of data an attribute may hold.

- If you don't show a type for an attribute, you should omit the colon. The type of an attribute may be another class.
- UML provides the following data types:
  - Boolean: A true or false value
  - Integer: An integer number
  - Real: A real number
  - String: A sequence of characters

- name: String [1] = ”Anon" {readOnly}
+ registeredIn: Course [*]

Attributes :: Multiplicity

visibility name: type multiplicity = default {property-string}

- Is optional
- Has a default value of 1
- Indicates the number of values an attribute may hold.

- The multiplicity is shown as a [*lower-bound .. upper-bound*] string in which a single asterisk indicates an unlimited range;

```
- address: String [1 .. 2]
- name: String [1] = ”Anon" {readOnly}
+ registeredIn: Course [*]
```

Attributes :: Default value

visibility name: type multiplicity = default {property-string}

- Is optional
- Has no default value
- Indicates the default value an attribute may hold.

- name: String [1] = ”Anon" {readOnly}

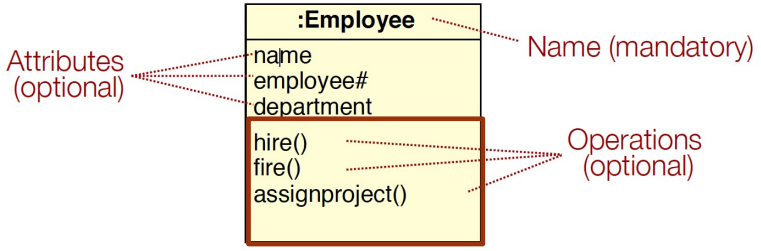
Attributes :: {property-string}

visibility name: type multiplicity = default {property-string}

- Is optional
- Has no default value
- It allows you to indicate additional properties for the attribute.

- name: String [1] = ”Anon" {readOnly}
- semesters : Semester [1 .. 8] {ordered}

Operations



- An operation is described in a class's third compartment using the following UML syntax:

visibility operation_name (parameter_list) : return_type {property-string}

Two examples of this is:

+ register (c: Course)

+ isRegistered (c: Course) : Boolean

- Only the operation_name is necessary.

Operations :: parameter_list

visibility operation_name (parameter_list) : return_type {property-string}

- Is optional
- Has no default value

- Is a comma-separated list indicating the parameters that hold the values passed to or received from the operation.

+ register (c: Course)
+ isRegistered (c: Course) : Boolean

- Each parameter is a text string having the following syntax:

name : type = default value

Operations

addEmailAddress

addEmailAddress (theEmailAddress)

addEmailAddress (theEmailAddress : String)

addEmailAddress (theEmailAddress : String = "")

addEmailAddress (theEmailAddress : String = "")

addEmailAddress (theEmailAddress : String = "") : Boolean

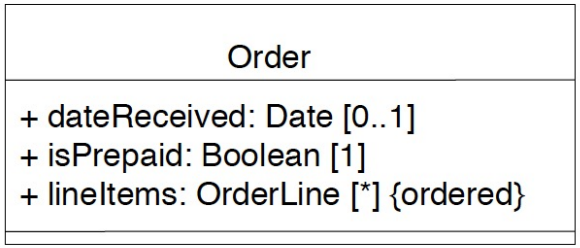
+ addEmailAddress (theEmailAddress : String = "") : Boolean

Relationships

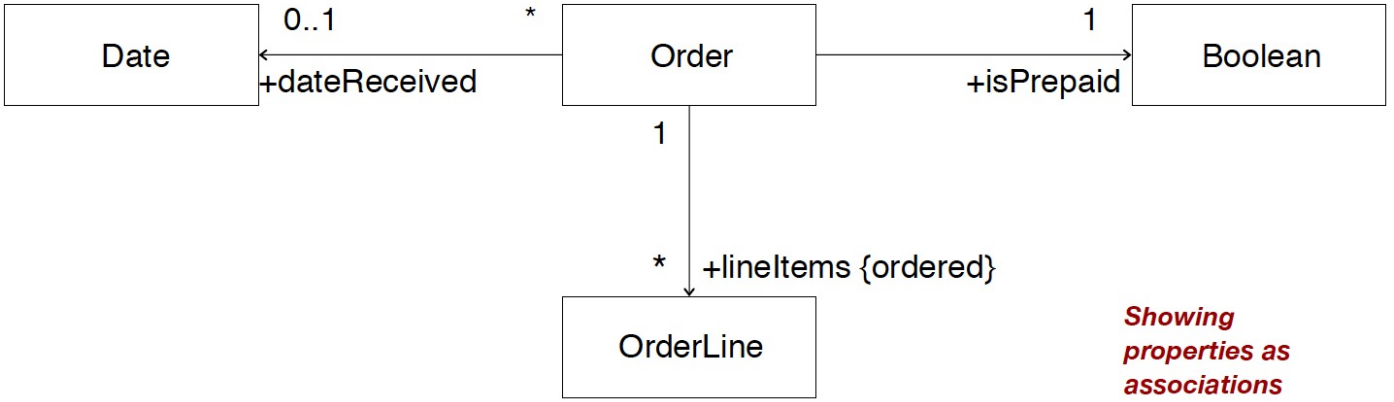
Relationships in class diagram

- In UML class diagram, there are different types of relationships:
  - Association
  - Aggregation and Composition
  - Generalization
  - Dependency
  - Realization

Association



Showing properties as attributes

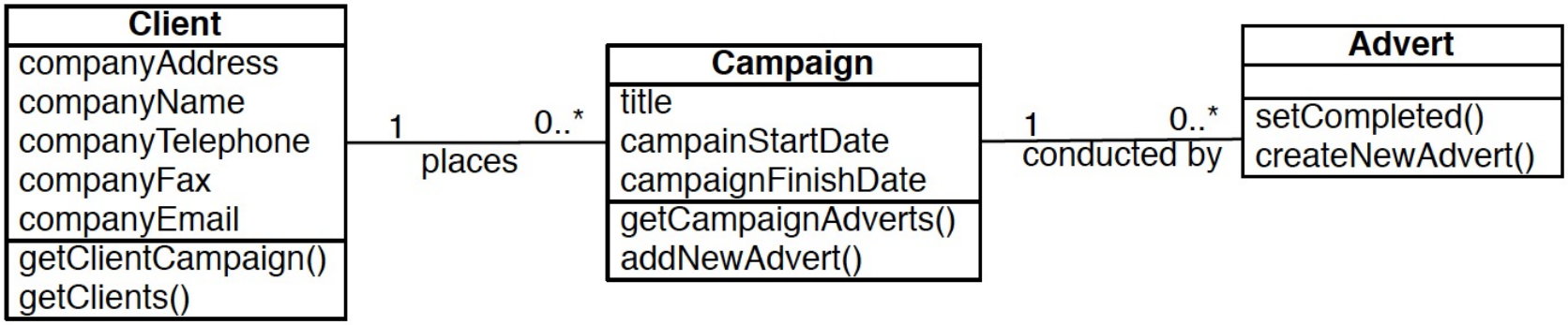


Showing properties as associations

Association

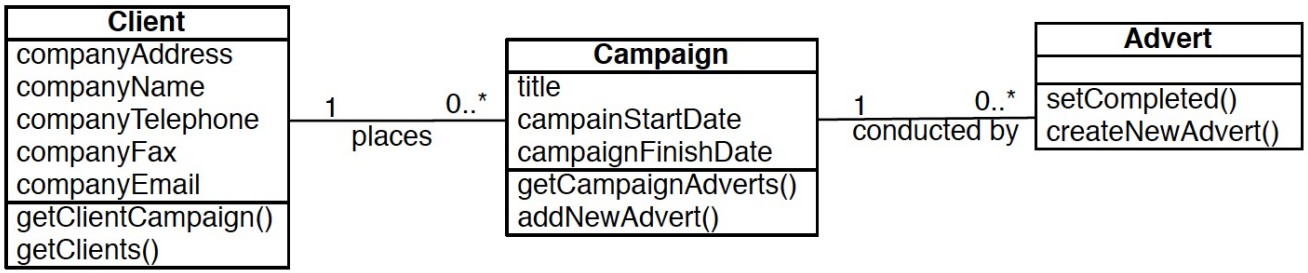
- Showing properties as attributes or associations?
  - Attributes for small things, such as dates or Booleans
  - Associations for more significant classes, such as customers and orders.

Association

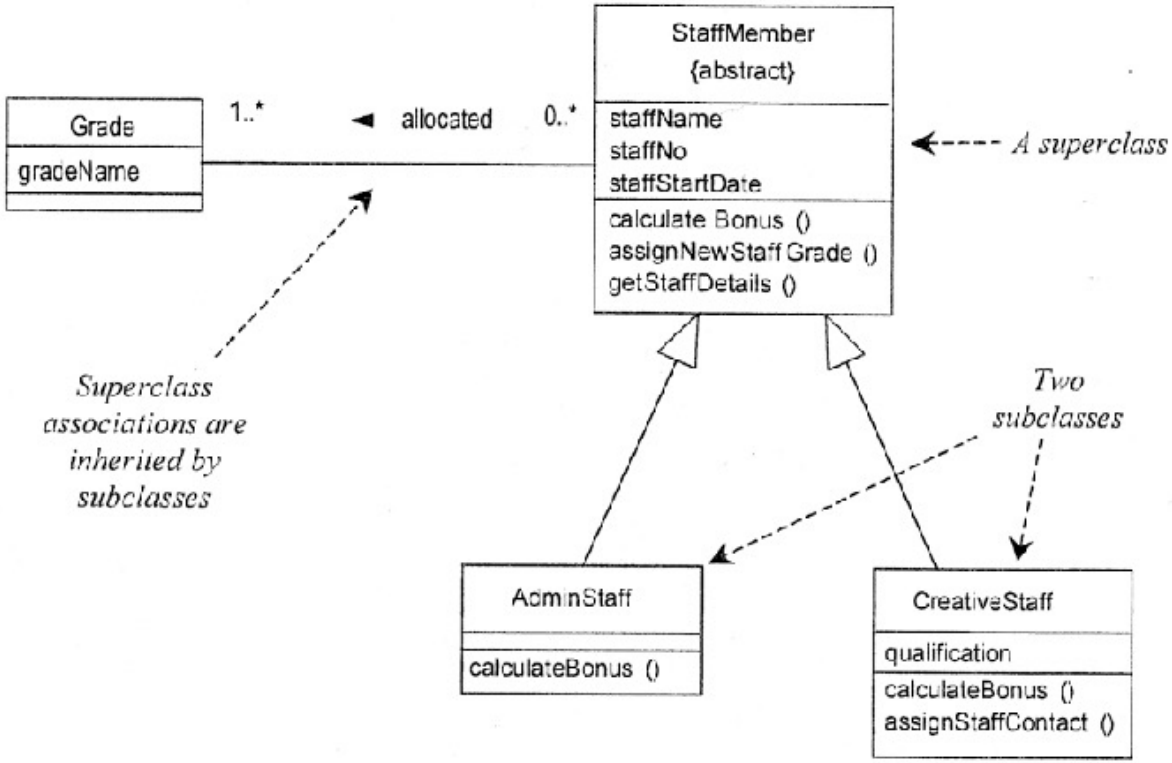


Association Multiplicity

- Ask questions about the associations:
  - Can a campaign exist without a member of client to manage it?
    - If yes, then the association is optional at the Client end - zero or more (0..*)
    - If no, then it is not optional - one or more (1..*)
    - If it must be managed by one and only one member of client-exactly one (1)
- What about the other end of the association?
  - Does every member of client have to manage exactly one campaign?
    - No. So the correct multiplicity is zero or more.



Generalization



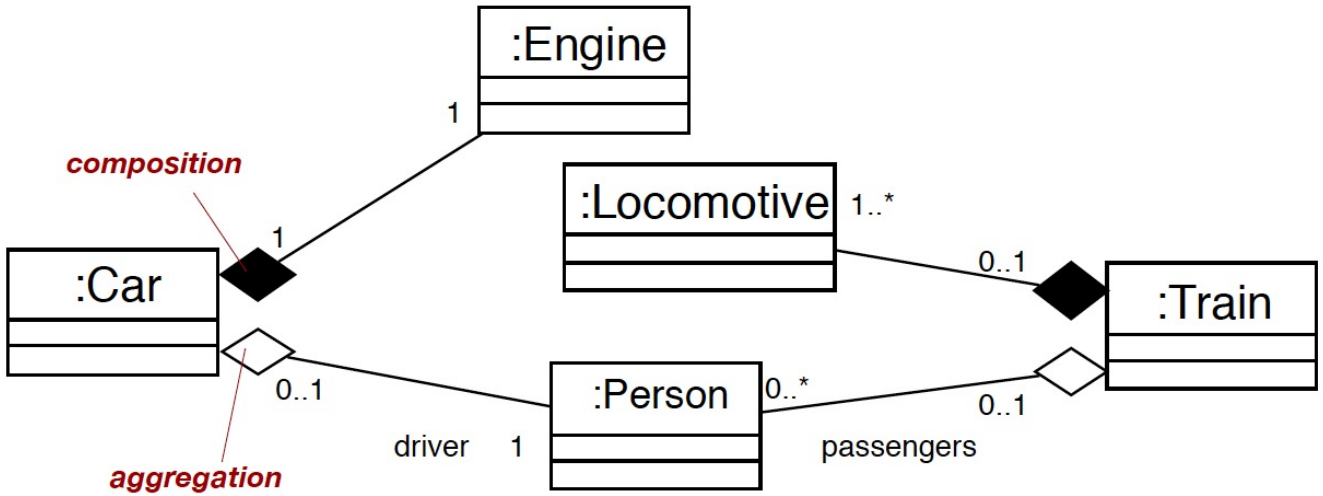
- Subclasses inherit attributes, associations, & operations from the superclass
  - A subclass may override an inherited aspect
    - e.g. AdminStaff & CreativeStaff have different methods for calculating bonuses
  - Superclasses may be declared {abstract}, meaning they have no instances

Generalization

- Abstract class:
  - a class that cannot be directly instantiated. Instead, you instantiate an instance of a subclass.
  - Typically, an abstract class has one or more operations that are abstract.
- An abstract operation has no implementation; it is pure declaration so that clients can bind to the abstract class.
- The most common way to indicate an abstract class or operation in the UML is to *italicize* the name.
- You can also make properties abstract, indicating an abstract property or accessor methods. Italics are tricky to do on a whiteboards, so you can use the label: {abstract}.

Aggregation and Composition

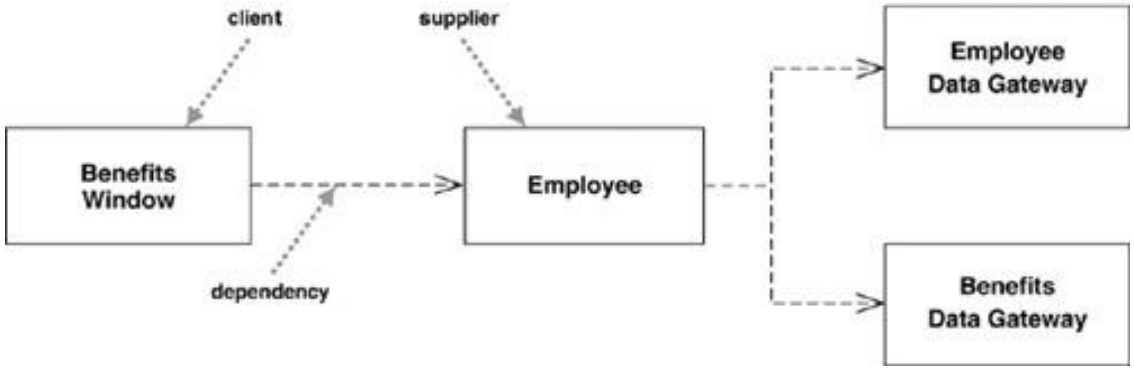
- Aggregation
  - This is the “Has-a” or “Whole/part” relationship
- Composition
  - Strong form of aggregation that implies ownership:
    - if the whole is removed from the model, so is the part
    - the whole is responsible for the disposition of its parts



Dependency

- A dependency exists between two elements if changes to the definition of one element (the supplier) may cause changes to the other (the client).

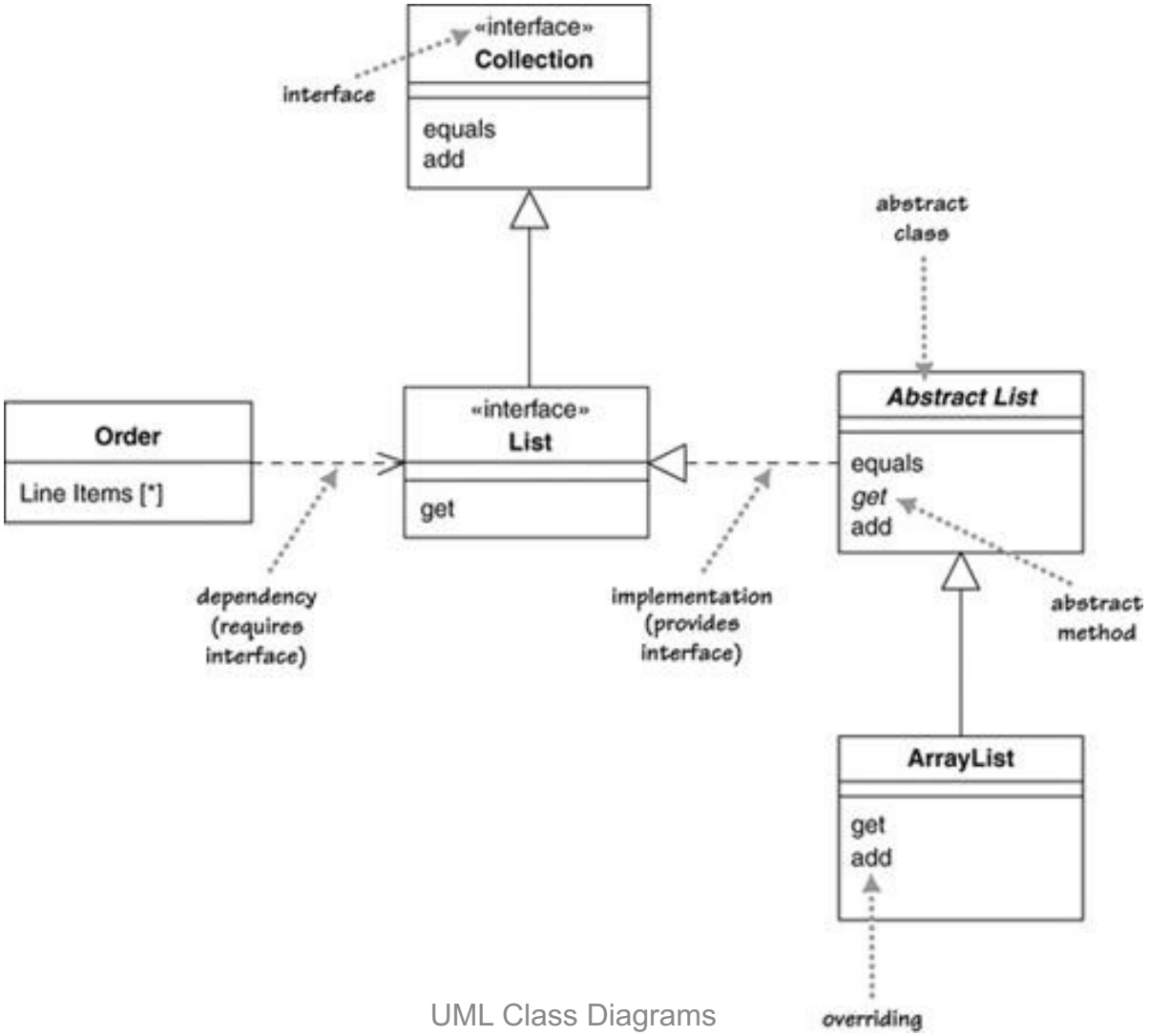
- Example Dependency types:
  - <<call>>
  - <<create>>
  - <<instantiate>>
  - <<parameter>>



Realization

- Interface class:
  - An interface is a class that has no implementation; that is, all its features are abstract.
  - You mark an interface with the keyword «interface».
- Classes have two kinds of relationships with interfaces:
  - Realization
    - by implementing the interface or implementing a subtype of the interface.
  - Dependency
    - A class requires an interface if it needs an instance of that interface in order to work. Essentially, this is having a dependency on the interface.

Realization



Summary

Summary

- A class diagram describes the types of objects in the system and the various kinds of static relationships that exist among them.
- Classes
  - Name
  - Attributes
  - Operations
- Relationships
  - Association
  - Generalization
  - Aggregation and Composition
  - Dependency
  - Realization