CT871 Lecture 3: Class Model

- UML Class Model
  1. Class Diagram
  2. Attributes & Operations
  3. Relationships
  4. Finding Classes
  5. Relationships: Aggregation, Generalisation
  6. Organisation
- Case Study:
  7. Example Class Model: ATM

UML Class Model

- Documents the static structure of the system: relationships between system data
- A class is a description of a group of objects with common:
  - Properties (attributes)
  - Behaviour (operations)
  - Relationships to other objects (associations / aggregations)

UML Class Model

- A class model is comprised of one or more class diagrams and the supporting specifications that describe model elements including classes, relationships between classes, and interface.
- Used for a variety of purposes – from understanding requirements to describing detailed design – apply a different style in each circumstance

UML Class: Analysis & Design

Analysis

Order
Placement Date Delivery Date Order Number
Calculate Total Calculate Taxes

Design

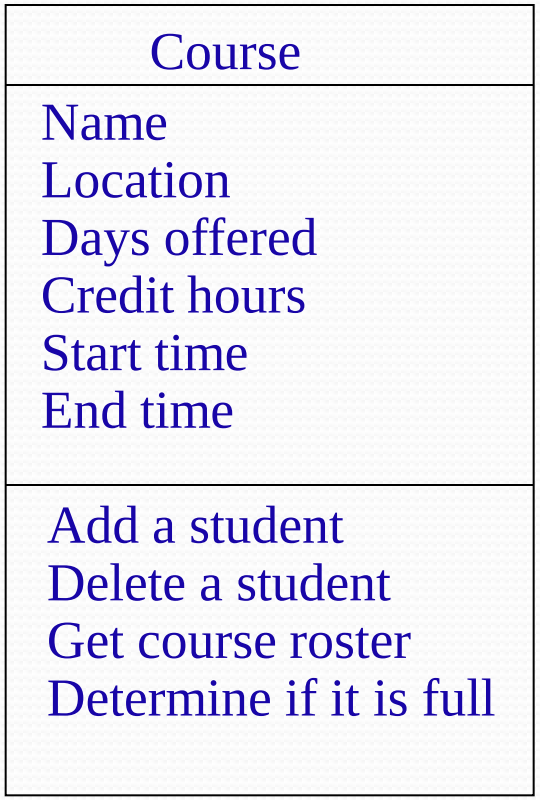
Order
- deliveryDate: Date - orderNumber: int - placementDate: Date - taxes: Currency - total: Currency
calculateTaxes(Country, State): Currency # calculateTotal(): Currency getTaxEngine() {visibility=implementation}

UML Class Model Notation

- UML represents a Class as a solid rectangle
- Three compartments:
  - Class Name and properties (optional):
    - Package
    - Stereotypes (<< >>)
  - Structure: Attributes
  - Behaviour: Operations

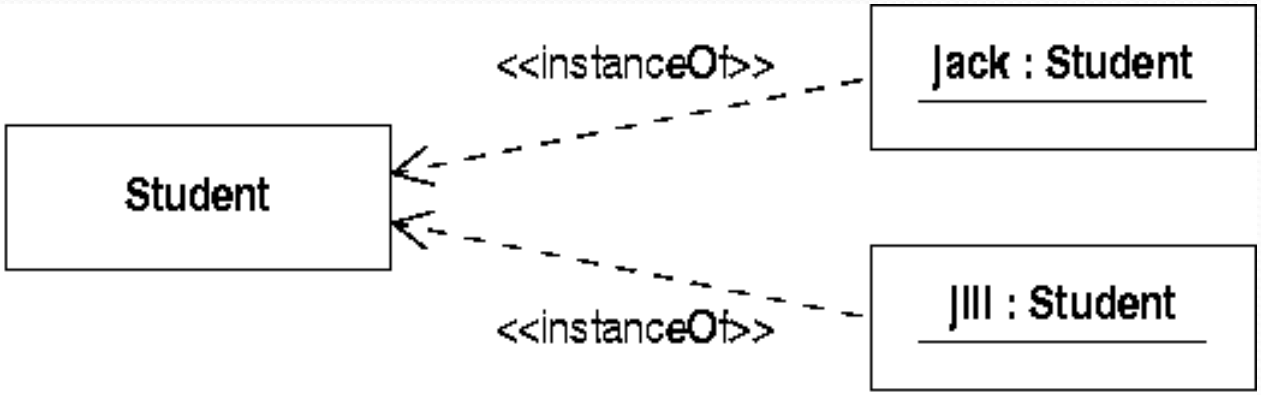
1. Classes

- Captures only one key abstraction
- A class name is a singular noun
- Class names start with an upper case letter; no underscores
- **Examples**: Course, LibraryMember, Book

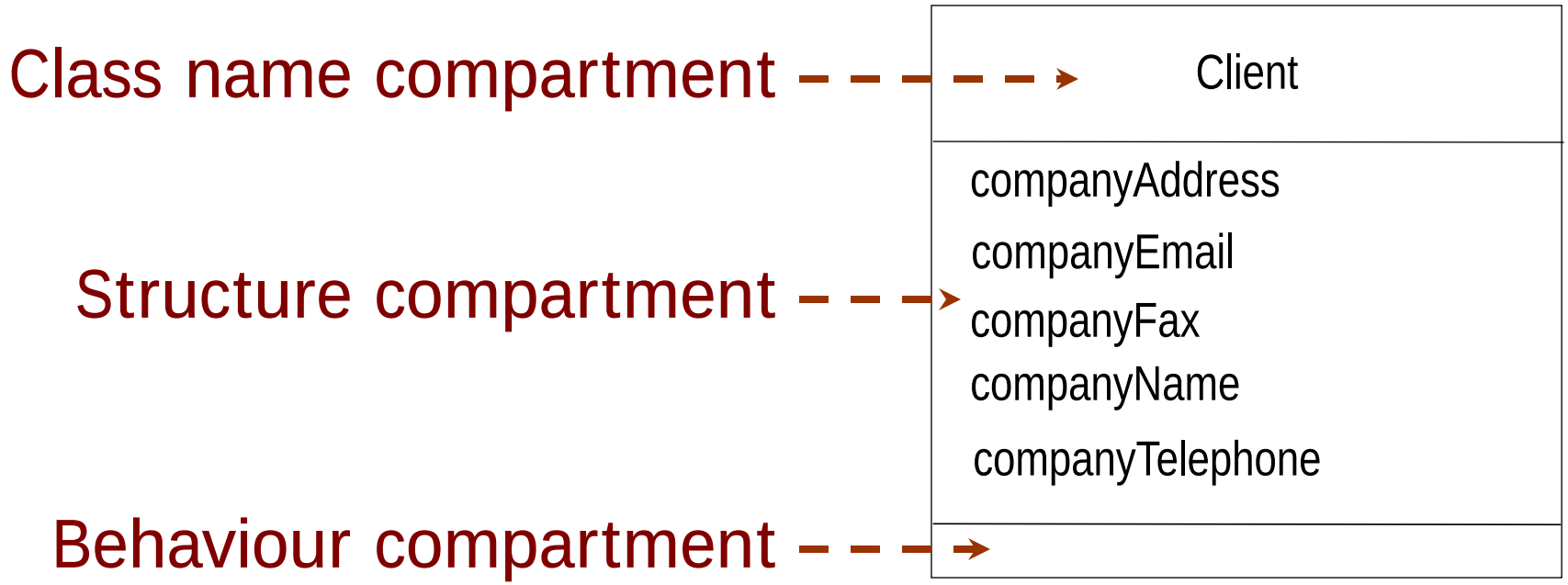


Class and Object

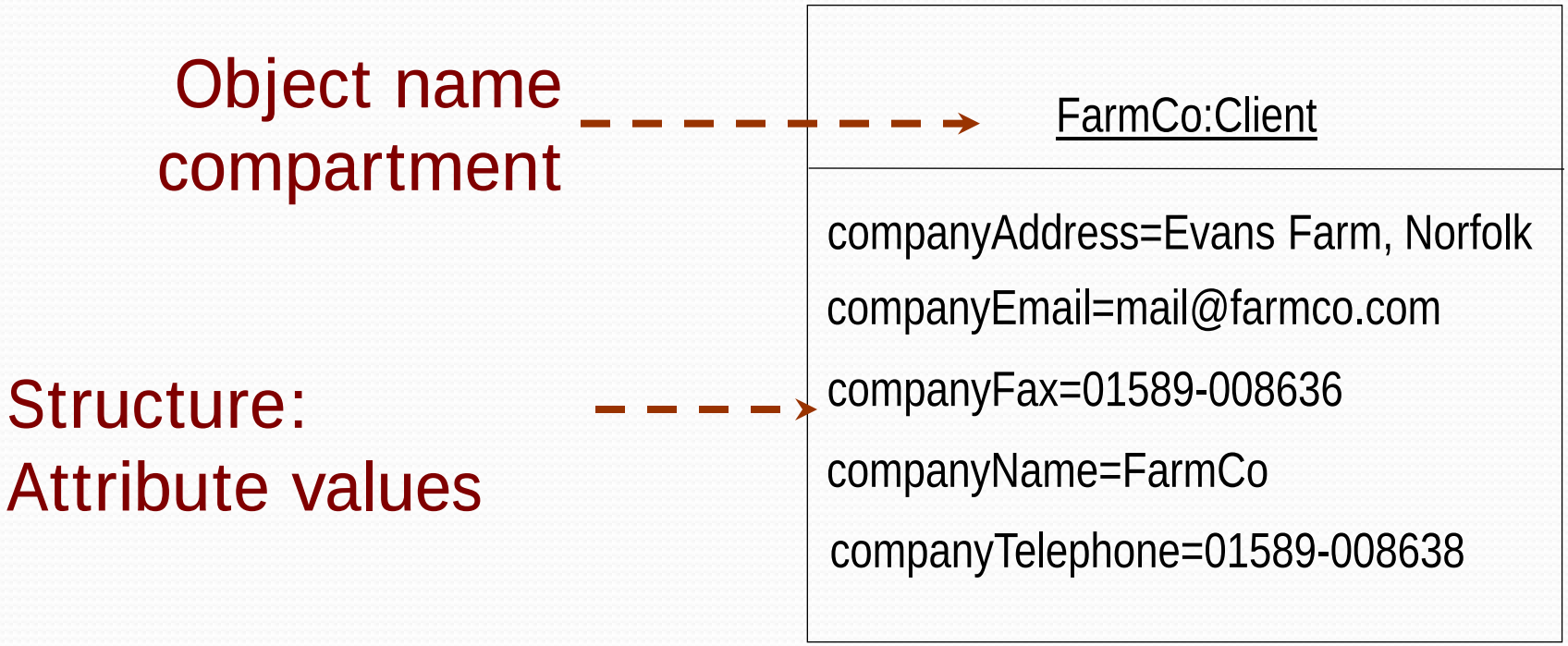
- A *class* describes a set of similar objects e.g. that share data and operations: Student
- The objects are the *instances* of the class: Jack and Jill



Example: Class Diagram



Example: Object Diagram



2. Attributes & Operations

- Having identified the classes, next we need to:
  - Define **attributes** for classes
  - Define **operations** for classes to carry out their responsibilities
  - Represent attributes and operations on class diagrams

Attributes

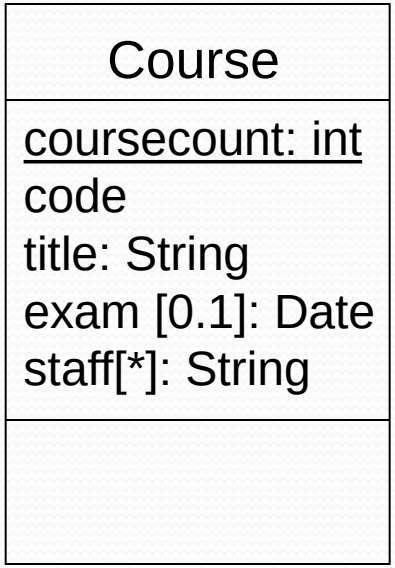
- Part of the essential description of a class
- The common structure of what the class can 'know'
- Attribute names are simple nouns or noun phrases, which must be unique within a class
- Shown in the second compartment of a class
- A data definition held by instances of a class
- Each object has its own value for each attribute in its class

Attributes

- Each attribute has a
  - Data type
  - Optional initial value
- A style guide should dictate naming conventions for attributes and operations: consistency and maintenance
- Example style guide:
  - Attributes and operations start with a lowercase letter
  - Underscores are not used

Example Attributes

- Attributes describe data fields
  - in a class, attributes can have a *type*
  - which defines the *values* that an object can hold
  - Attributes also have multiplicity and scope



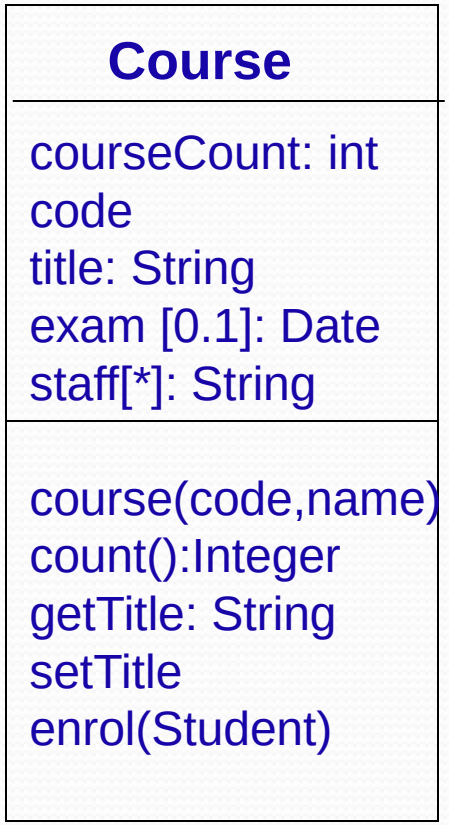
Operations

- Class responsibilities are carried out by operations
- An object sends a message to another object asking it to perform an operation; this receiver then invokes a method to perform the operation
- Operations are difficult to consider in isolation
- Operations are named:
  - to indicate their outcome: getBalance() as opposed to calculateBalance()
  - from the perspective of the supplier, not the client

Operations

- Operations can be broadly categorised as *Occur* or *Calculate* operations
- *Occur*:
  - Constructors and destructors: *Add* and *Delete*
  - Also *Change* and *Select*
- *Calculate*:
  - Perform calculations on the data values encapsulated in the same object class
  - Calculate operations can be simple (*debit account*) or complex

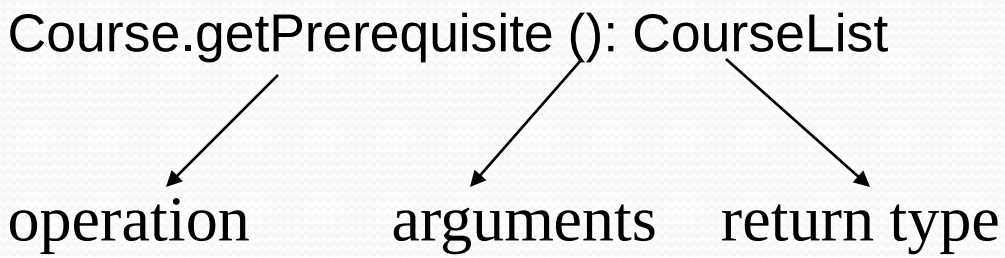
Example: Operations

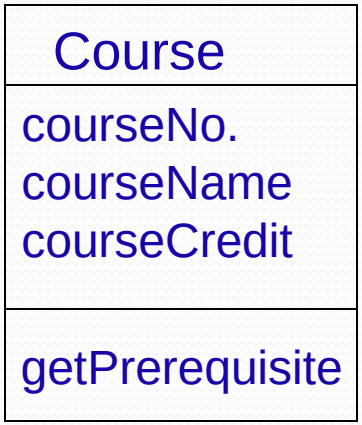


- Messages (user problem domain) are translated into operations (moving into design domain)
- A more detailed specification for each operation will be developed during design

Operations

- The operation signature consists of:
  - Operation name
  - Optional argument list
  - Return class
- Operations are shown in the third compartment of a class:
- Example:





Additional Classes

- In specifying operation signature, can find additional classes as
  - Arguments to the operation: *addLibraryMember* (John: *LibraryMemberInfo*)
  - Return class: *getbooksborrowed*(): BookList
- Can also find additional relationships
- These additional classes and relationships are added to the class model

Visibility

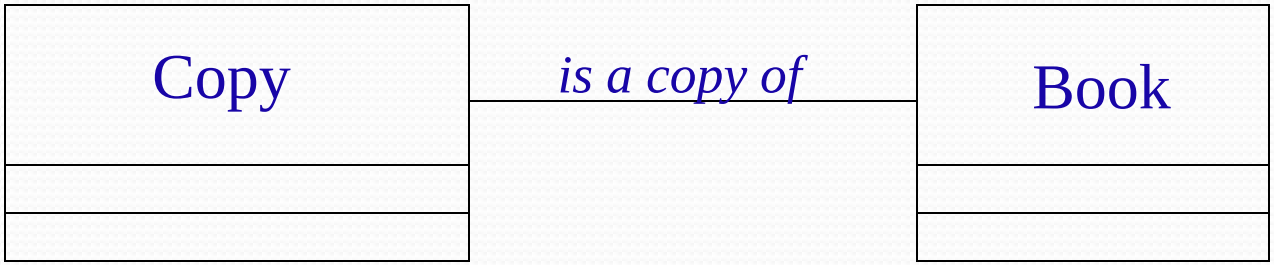
- Attributes and operations can have a *visibility*
  - parallel to Java/C++ access levels
- UML defines four levels of visibility:
  - *public* (+): visible to all objects
  - *package* (~): visible to objects in same package
  - *protected* (#): visible to instances of subclasses
  - *private* (-): visible only in same object class

3. Relationships

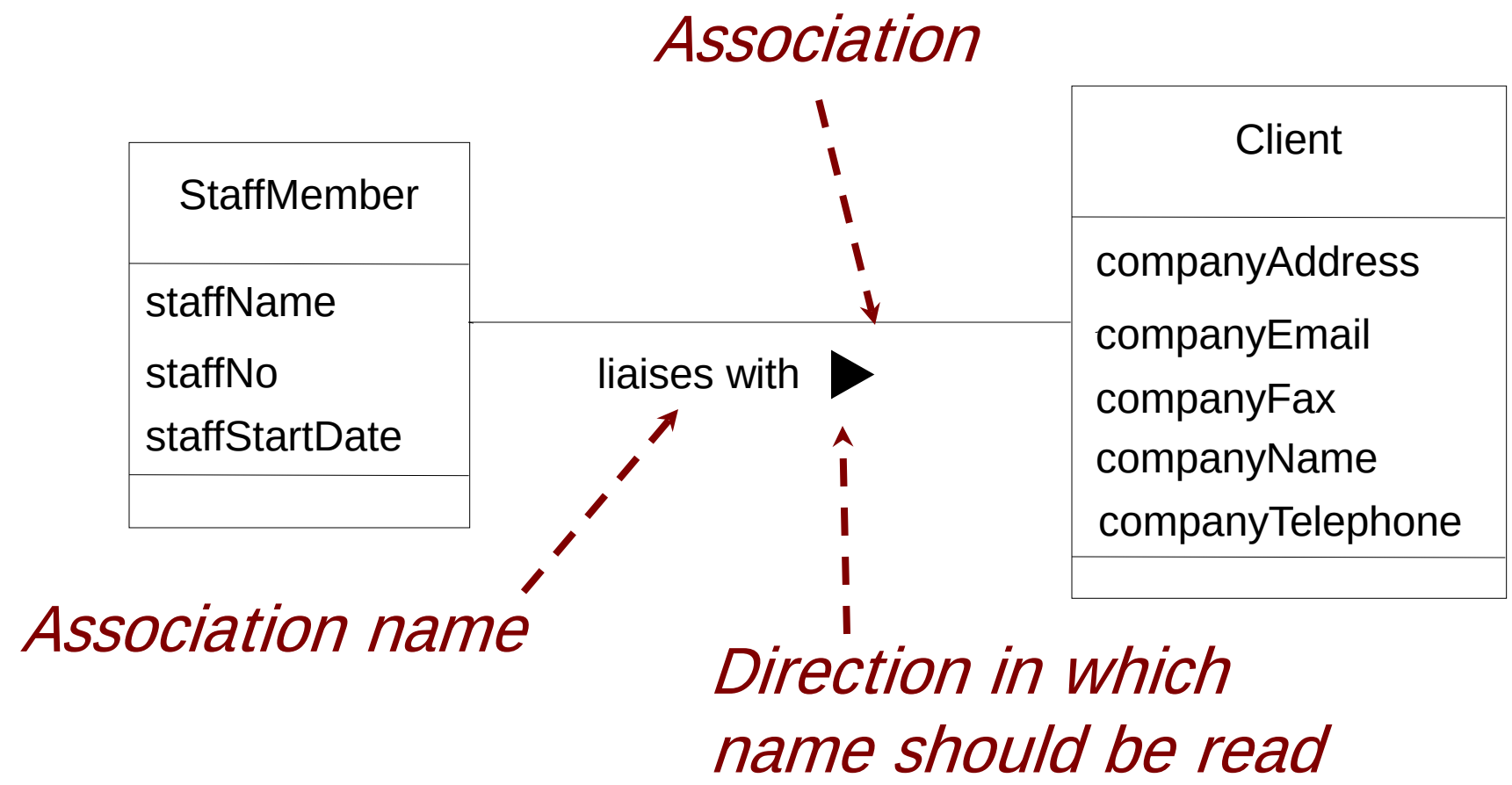
- Individual object classes now defined
- Objects contribute to the behaviour of a system by collaborating with one another
- Collaboration is accomplished through relationships
- Establish connections among classes
  - these exist because of the nature of the classes
  - estimate multiplicity

Association Relationship

- If some object of class A has to know about some object of class B (a library member borrows a book)
- Represented on class diagrams by a connecting line: instance of an association is a link
- Data may flow in either or both directions: uni or bi-directional
- Association may be named to clarify its meaning

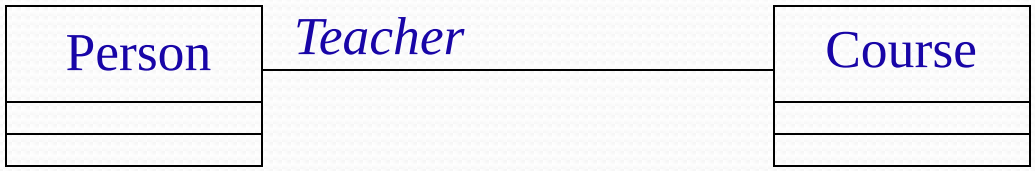


Example: Association Relationship



Roles

- A role denotes the purpose or capacity wherein one class associates with another
- Role names are typically nouns placed along the association line close to the class it modifies
- One or both ends of an association may have role names

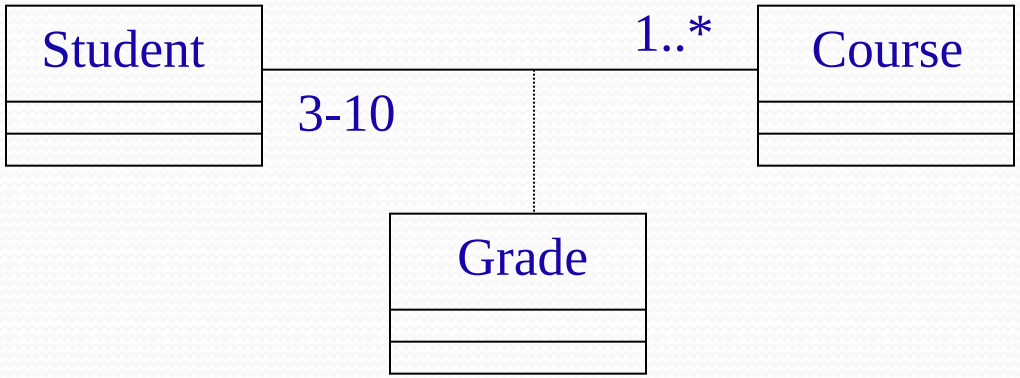


Multiplicity

- The number of instances of **one** class related to **ONE** instance of another class
- Represented by *ranges*
  - a range has lower and upper bounds, e.g. 0..9
  - * represents an unbounded multiplicity, e.g. 1..*
  - 0..* ('zero or more') is often abbreviated as *
  - 0..1 represents an optional entity
  - 1..1 is abbreviated to simply 1

Association Classes

- Association class stores information that belongs to the link between two objects where needed
- Only one association class is permitted per association

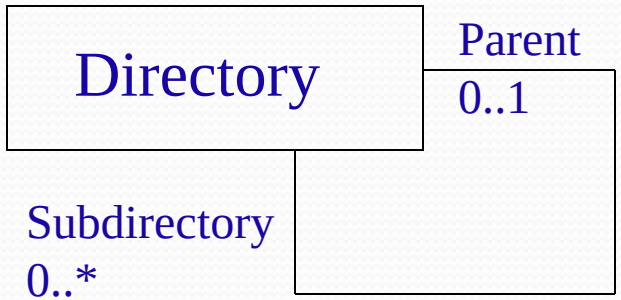


- Use the class icon, and connect it to the association line using a dashed line

Associations

- Associations are not always between two classes (binary):
  - Common for a class to have an association to itself: reflexive
  - An association can connect more than two classes

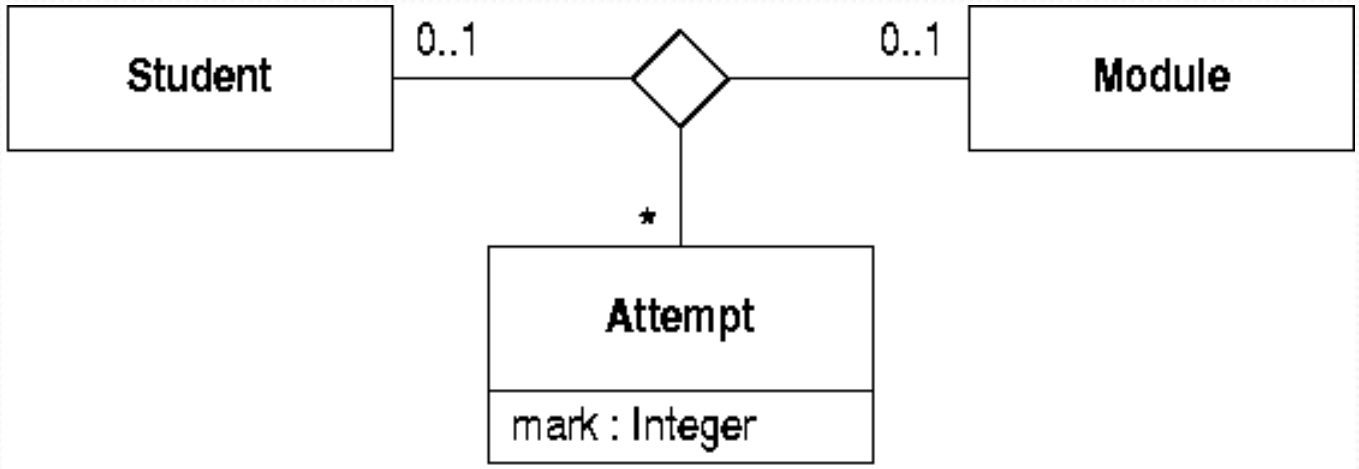
Reflexive Association



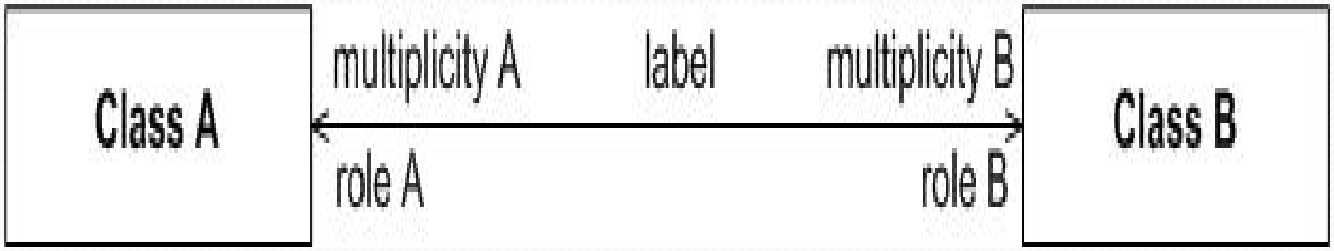
- Each Directory object can have links to zero or one Directory object that play the role subdirectory
- Each Directory object can have links to zero or one Directory object that plays the role parent

N-ary Associations

- Associations can connect more than two classes
  - a 3-way association could be used to store marks



Association Notation:



4. Finding Classes

- OOD: complexity of identifying classes of objects
- Data driven design:
  - Identification and filtering of nouns
  - Entity objects found are grouped into classes
- Responsibility driven design:
  - Identify responsibilities of the system (active processes)
- Best approach involves a combination of both

Finding Classes

- Objects and their division into classes often derive from one of the following sources:
  - Tangible, real world things
  - Roles: library member, student
  - Events: arrival, request
  - Interactions: meeting

Finding Classes

- Good analysis classes can be summarised as follows:
  - Name reflects intent
  - Crisp abstraction modeling one problem domain element
  - Has a small, well-defined set of responsibilities
  - Has high cohesion
  - Has low coupling

CRC Cards

- Class–Responsibility–Collaboration (CRC) cards help to model interaction between objects
- For a given scenario (or use case):
  - Brainstorm the objects
  - Allocate to team members
  - Role play the interaction

CRC Cards

Class Name:	
Responsibilities	Collaborations
Responsibilities of a class are listed in this section.	*Collaborations with other classes are listed here, together with a brief description of the purpose of the collaboration.*

CRC Cards

- Used to check for good design
- Class, Responsibilities and Collaborations
- Technique to help programmers think “in objects”
- CRC card: small index card; record name of class, responsibilities of the class on left hand side, and collaborators of the class on right hand side
- Example:

LibraryMember	
Responsibilities	**Collaborators**
Maintain data about copies currently borrowed Meet requests to borrow and return copies	Copy