Section 2.3
UML Class Diagrams

1. Documentation
2. Types of documentation
3. Class diagrams

2.3.1 Documentation

- True fact: nobody likes to document their code
- But what happens when the lead programmer...
  - gets hit by a bus
  - decides to become an organic produce farmer
  - quits for a better job

Documentation (cont.)

- Result:
  - nobody on the team knows enough to finish the product
  - the customer cancels the contract
  - your company goes bankrupt
  - you lose your job

- Moral of the story:
  - document your code!
  - make everyone document theirs too

2.3.2 Types of Documentation

- Software development life cycle activities:
  - requirements analysis
  - design
  - implementation
  - testing

- Each activity requires documentation

- Type of documentation is different at every stage

Types of Documentation (cont.)

- Requirements analysis
  - functional and non-functional requirements
    - they are a specification of what the system will do
    - they are user-visible constraints on the system
  - use cases
    - they model the interactions between the user and the system
  - high-level object model
    - this models the high-level concepts manipulated by the system
    - these concepts include major entity, control, and boundary objects
  - dynamic model
    - this models the system behaviour from an external point-of-view

Types of Documentation (cont.)

- Design
  - subsystem decomposition
    - this is a specification of high-level subsystems based on architecture
  - detailed object model
    - the low-level objects required to implement subsystem interfaces
- Implementation
  - program comments
- Testing
  - test plan
  - test cases

Program Comments

- All your code should:
  - be self-documenting
  - read like the story of your program
    - first this happens, then that happens, ...
- Comments should describe:
  - the purpose of every class
  - details of complex or critical members

Test Cases

- Test cases describe:
  - what isolated portion of the program you are testing
  - the test input
  - the expected output
- Unit testing
  - test one function or class separately
- System testing
  - test that each requirement is implemented and works correctly

2.3.3 Class Diagrams

- Unified Modelling Language (UML)
  - it is a family of notations used to represent OO models
  - it is a major tool used to document OO design
  - it facilitates communication between developers
  - it is programming language independent

Class Diagrams (cont.)

- There are many types of UML design diagrams
  - UML class diagram
    - attributes and operations
    - associations (relationships) between classes
  - UML activity diagram / sequence diagram
    - behaviour of program
    - interaction between classes
  - UML state machine diagrams

Class Diagrams (cont.)

- UML class diagrams are **not** tied to a specific language
  - given class diagram, program can be written in any OO language
- UML class diagrams represent:
  - classes
    - attributes
    - operations
  - associations
    - relationships between classes

Class Diagrams (cont.)

- Classes
  - data structure in OO language to represent single concept
  - attributes
    - generic term for “data members”, “instance variables”, etc.
  - operations
    - generic term for “member functions”, “instance methods”, etc.
  - access specifiers for attributes and operations
    - private: **–**
    - protected: **#**
    - public: **+**

Class Diagrams (cont.)

- Associations
  - these are the relationships between classes
  - composition
    - "has-a" relationship
    - terminology:
      - container: the class that "has" instance(s) of the other class
      - containee: the class that is contained within the other class
  - inheritance
    - "is-a" relationship
    - terminology
      - super-class: the generalized class
      - sub-class: the specialized class

Class Diagrams (cont.)

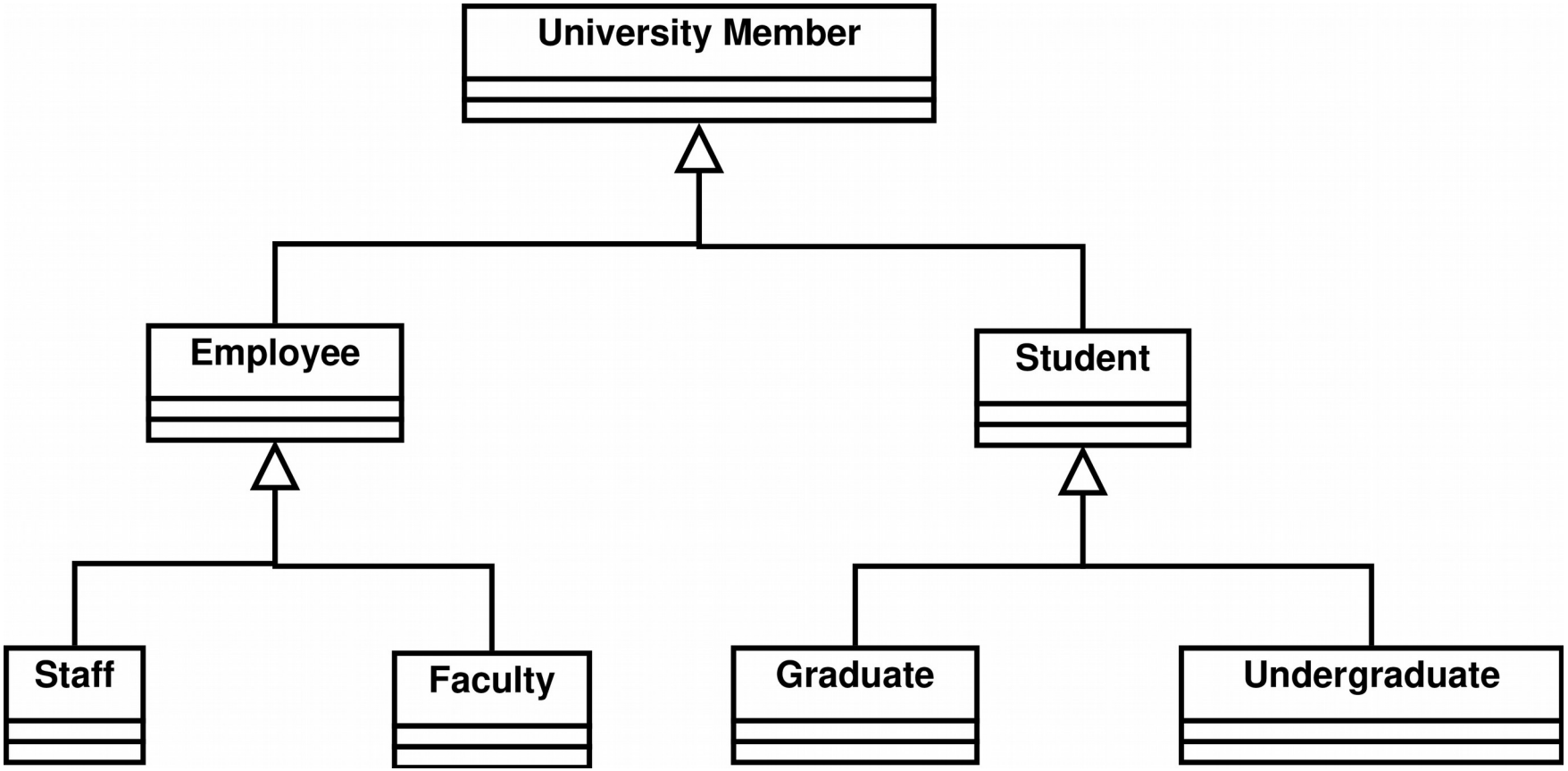
- Characteristics of composition associations
  - directionality
    - indicates which class is the container and which is the containee
    - unidirectional
      - the container has one or more instances of the containee
      - it is represented by line with an *arrow from container to containee*
    - bidirectional
      - the container has one or more instances of the containee, and
      - the containee has one or more instances of the container
      - it is represented by line with *no arrow*
  - multiplicity
    - the number of containee instances in the container
    - values: 0, 1, many (*), or a range

Class Diagrams (cont.)

- Important convention
  - we do **not** show collection classes in UML
    - that's because they are *implied* using multiplicity
  - we must still show the multiplicity on the correct classes
  - example:
    - in real life: a Library contains many Books
    - in UML:
      - show Library class with a 0-to-many association to Book
      - do **not** show the BookArray class at all
    - in program:
      - Library object contains a BookArray object
      - BookArray object contains a collection of Book objects

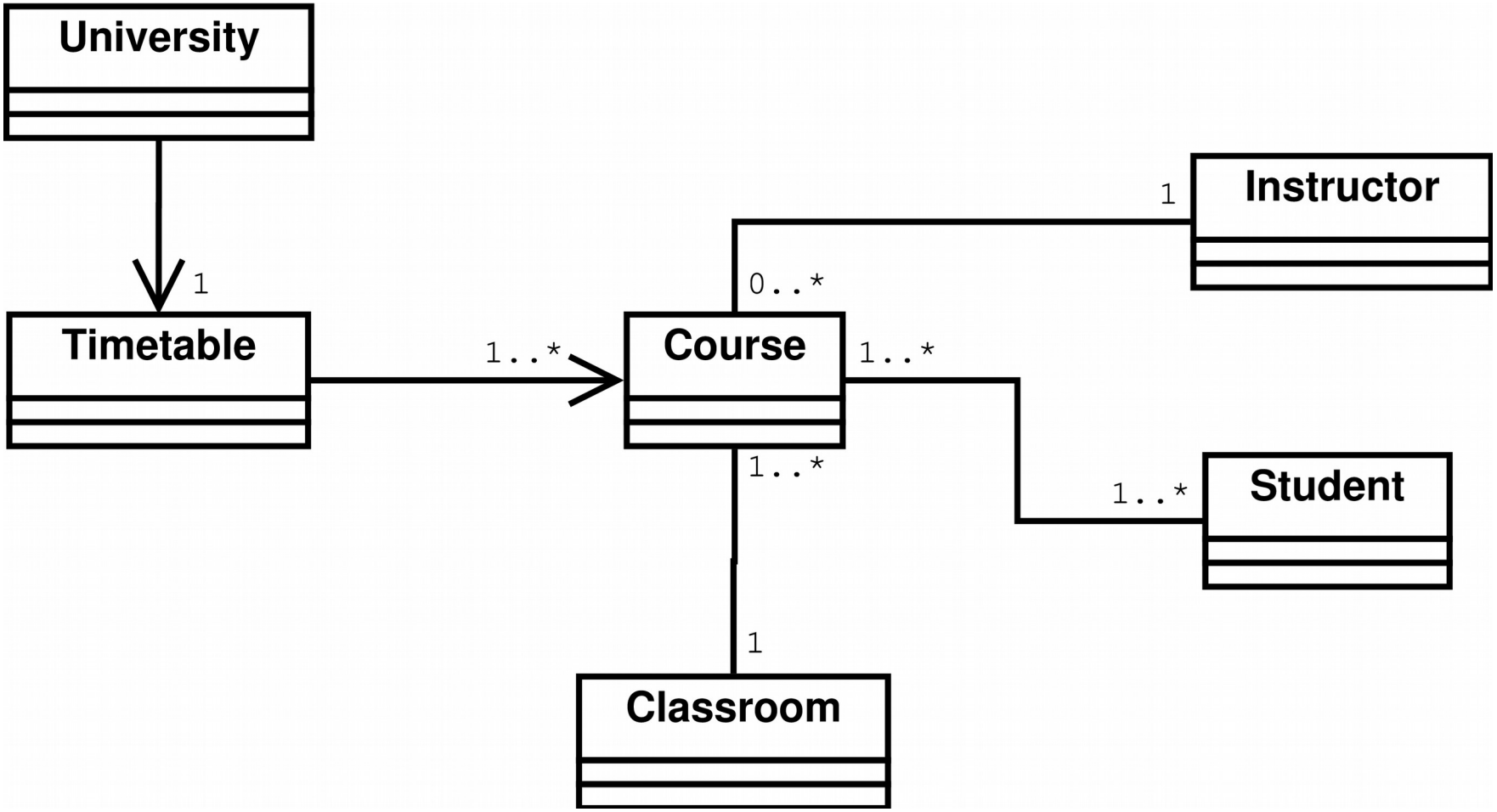
Class Diagrams (cont.)

- Inheritance



Class Diagrams (cont.)

- Composition



Class Diagrams (cont.)