

# Unified Modeling Language (UML) crash course



Version 1.0 Oct 2005

## Note well

- This slide set presents a **very small fraction** of UML capabilities
- Further readings
  - ♦ [www.cetus-links.org](http://www.cetus-links.org)
  - ♦ M.Fowler, K. Scott, "UML Distilled 2<sup>nd</sup> ed.", Addison-Wesley
- ArgoUml, UML design tool
  - ♦ <http://argouml.tigris.org>
- Omondo UML, Eclipse plugin for UML
  - ♦ <http://www.omondo.com>



4

## Learning objectives

- Understand the concepts of UML model and UML diagram
  - ♦ What is a UML Class Diagram?
- Understand the steps of development process
  - ♦ How to translate specs to code?



2

## Models and diagrams

- It is important to distinguish between a UML **model**, and a (set of) UML **diagram(s)**
- A diagram is a graphical representation of the information in the model, but the model exists independently
- Use Case Diagram, Collaboration Diagram, Activity Diagram, Sequence Diagram, Deployment Diagram, Component Diagram, **Class Diagram**, StateChart Diagram



5

## Intro

- UML is a standardized modeling and specification language by the **Object Management Group (OMG)**
- **Graphical notation** to specify, visualize, construct and document an object-oriented system
- Support throughout **many development phases** (analysis and requirements, high-level design, detailed design, implementation, deployment ...)
- Integrates the concepts of Booch, OMT and OOSE, and coalesces them into a single, common and **widely used modeling language**



3

## Models and diagrams

- There are **three prominent models** of the UML system development
- **Functional Model** – Showcases the functionality of the system from the User's Point of View
- **Object Model** – Showcases the structure and substructure of the system using objects, attributes, operations, and associations
- **Dynamic Model** – Showcases the internal behaviour of the system

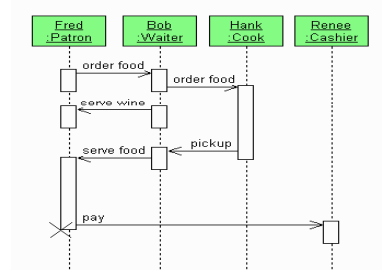


6

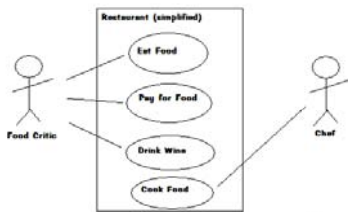
## Models and diagrams

Functional Model	Use Cases Diagrams
Object Model	Class Diagrams
Dynamic Model	Sequence Diagrams, Activity Diagrams, Statechart Diagrams

## Sequence Diagram



## Use Case Diagram

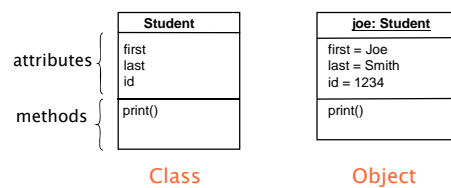


## Class diagram

## Class Diagram



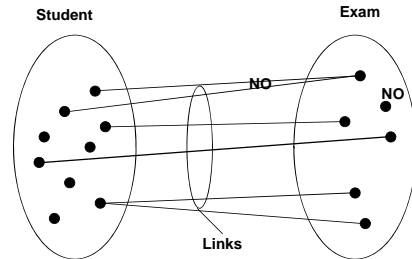
## Class and object



## Class/Object Diagrams

- Class diagrams
  - Shows relationships among (part of the) application classes
  - Classes and Associations
- Object diagrams
  - Shows relationships among (part of the) application objects
  - Objects and Links

## Example

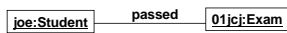


## Class/Object Diagrams

- Class diagrams



- Object diagrams



## Types of association

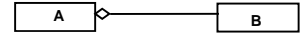
- Use

- B uses A



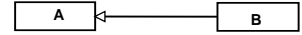
- Aggregation (*part of*)

- B is part of A

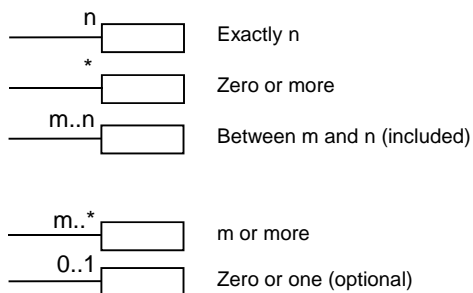


- Inheritance (*is a*)

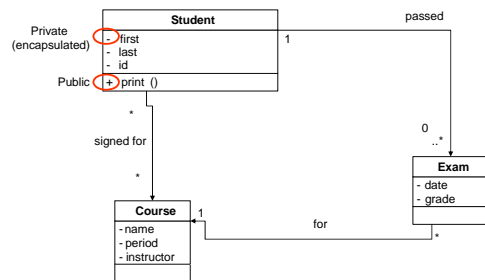
- B is a child of A



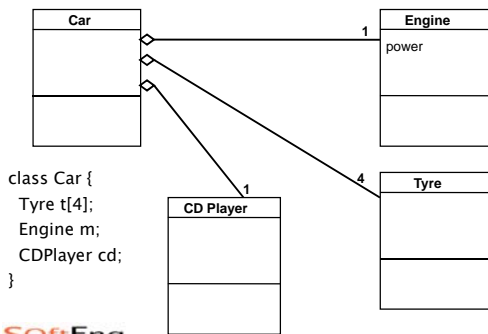
## Multiplicity of assoc. ends



## Use



## Aggregation



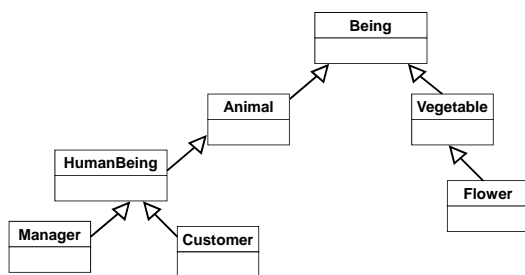
```

class Car {
    Tyre t[4];
    Engine m;
    CDPlayer cd;
}
    
```

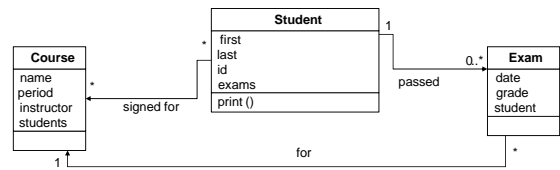
## Analysis

- Identify classes
  - ♦ Substantives and real objects (having attributes)
- Identify attributes
  - ♦ Substantives, physical properties
- Identify methods
  - ♦ Delegation, information hiding
- Identify associations

## Inheritance



## UML analysis



## Process

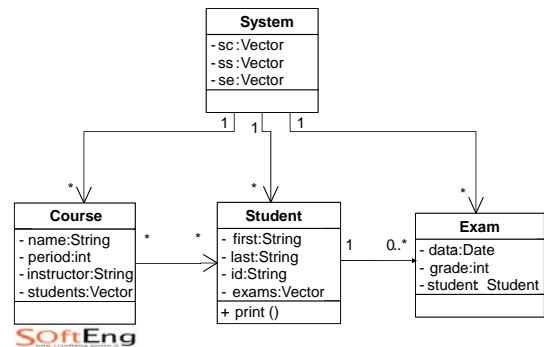
## Design

- Add/modify classes for
  - ♦ User Interface / Graphical user Interface
  - ♦ DB access
  - ♦ Net distribution
  - ♦ Efficiency/Optimization

## OO – Design Heuristics

- All data should be hidden within its class
- Keep related data and behavior in one place
- Model the real world whenever possible
- Eliminate classes that are outside the system
- Avoid all-powerful (omnipotent) classes
- Minimize the number of messages sent between two classes

## UML low-level design



## More OO Design Heuristics

- If a class contains objects of another class, then the containing class should be in charge of sending messages to the contained objects
- The containment relationship should always imply a uses relationship
- A class must know what it contains, but it should not know who contains it

## How to implement associations

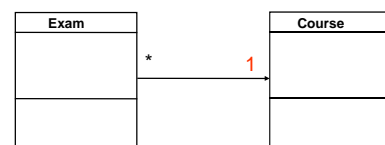


## Low level design

- Implement classes
- Implement attributes
  - ♦ Define the type
- Implement methods
  - ♦ Define the prototype
- Implement associations

## Association : 1

- From Exam towards Course



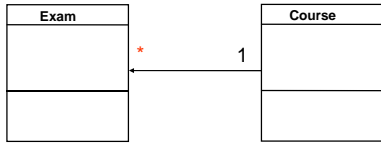
```

Class Exam {
    Course c;
    setCourse(Course c){
        this.c=c;}
}

Class Course {
}
    
```

## Association :n

- From Course towards Exams

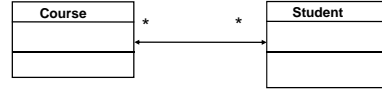


```

Class Course {
    Vector exams;
    Course(){ exams = new Vector(); }
    addExam(Exam e){ exams.add(e); }
}
    
```

## Association n:m

- Both directions

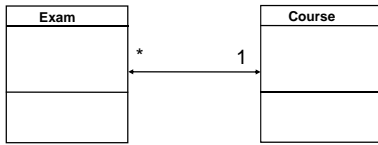


```

Class Course {
    Vector students;
    Course(){
        students = new Vector();
    }
    addStudent(Student s){
        students.add(s);
    }
}
Class Student {
    Vector courses;
    Students(){
        courses = new Vector();
    }
    addCourse(Course c){
        courses.add(c);
    }
}
    
```

## Association 1:n

- Both directions



```

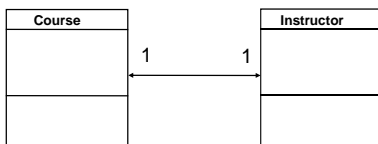
Class Exam {
    Course c;
    setCourse(Course c){
        this.c=c;
    }
}
Class Course {
    Vector exams;
    Course(){ exams = new Vector(); }
    addExam(Exam e){ exams.add(e); }
}
    
```

## Wrap-up session

- UML is a graphical notation for modeling and documenting OO systems
- Class diagram
  - Classes and associations
- Three types of associations
- Developing is not "just coding"!
  - Use the process to tackle the req. spec.

## Association 1:1

- Both directions



```

Class Course {
    Instructor i;
}
Class Instructor {
    Course c;
}
    
```