# UML Tutorial Part 2: UML Class and Interaction Diagrams

**Presented by Igor Ivković**
*iivkovic@swen.uwaterloo.ca*

## Why UML Class Diagrams?

- **UML Class Diagrams**
  - Visual specification of types of objects that exist in a system and the relationships that exist among them
  - A UML class describes a set of objects that share the same attributes, operations, relationships, and semantics
- Class diagrams may specify both the conceptual **[what]** and implementation **[how]** details of the system
- Class diagrams represent structural and **not** behavioural relationships that exist among system entities
- Class diagrams are used as a basis to develop other UML diagrams including sequence and collaboration diagrams

*2*

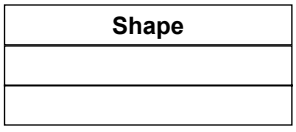
# Agenda

- Terms and Concepts
- UML Class and Interaction Diagrams Example
- Summary and References

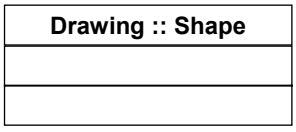
*3*

# Class Names

- **Class Names**
  - Simple Name (written as UpperCase-first Noun)



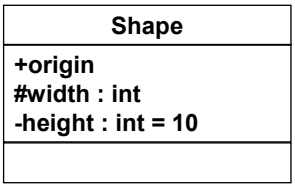

  - Path Name
    Package name :: Class name




*4*

# Class Attributes

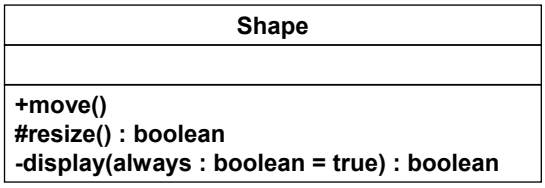
- **Class Attributes**
  - Represent named properties of a UML class
  - UML class can have many attributes of different names
  - Attribute name is generally a short noun or a noun phrase written in lowerCase-first text
  - Attribute declaration may include visibility, type and initial value: *+attributeName : type = initial-value*




*5*

# Class Operations

- **Class Operations**
  - Represent named services provided by a UML class
  - UML class can have many operations of different names
  - Operation name is generally a short verb or a verb phrase written in lowerCase-first text
  - Operation may include visibility, parameters, and return type: *+opName(param1 : type = initial_value) : return-type*




*6*

# Class Visibility

- **Class Visibility**
  - Three levels of class, attribute and operation visibility:
    - private (-), available only to the current class
    - protected (#), available to the current and inherited classes
    - public (+), available to the current and other classes

| Shape |
| --- |
| +origin<br>#width : int<br>-height : int = 0 |
| +move()<br>#resize() : boolean<br>-display(always : boolean = true) : boolean |

*7*

# Class Objects

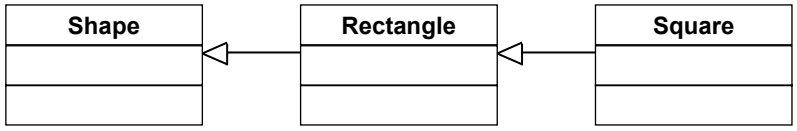
- **Class Objects**
  - Each class represents a set of objects that share the same attributes, operations, relationships, and semantics
  - For each of the class attributes, objects can have specific attribute values
  - For each of the class operations, objects may have different implementations

| s1 : Shape |
| --- |
| origin = (10, 10)<br>width = 15<br>height = 30 |
| … |

*8*

# Class Generalization

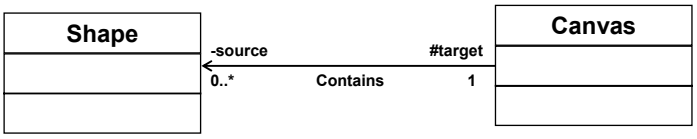
- **Class Generalization**
  - Represent a relation between a parent (a more abstract class) and a child (a more specific class)
  - Generally referred to as a “is-a-kind-of” relationship
  - Child objects may be used instead of parent objects since they share attributes and operations; the opposite is not true




*9*

# Class Association

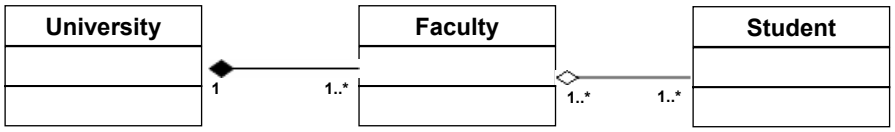
- **Class Association**
  - Represent a structural relationship between class objects and may be used to navigate between connected objects
  - Association can be binary, between two classes, or n-ary, among more than two classes
  - Can include association name, direction, role names, multiplicity, and aggregation type




*10*

# Class Aggregation

- **Class Aggregation**
  - Represent a specific, whole/part structural relationship between class objects
  - Composition (closed diamond) represents exclusive relationship between two class objects (e.g., a faculty cannot exist without nor be a part of more than one university)
  - Aggregation (open diamond) represents nonexclusive relationship between two class objects (e.g., a student is a part of one or more faculties)




*11*

# Agenda

✓ Terms and Concepts

➢ UML Class and Interaction Diagrams Example

o Summary and References

*12*

## Recall Courseware System Description

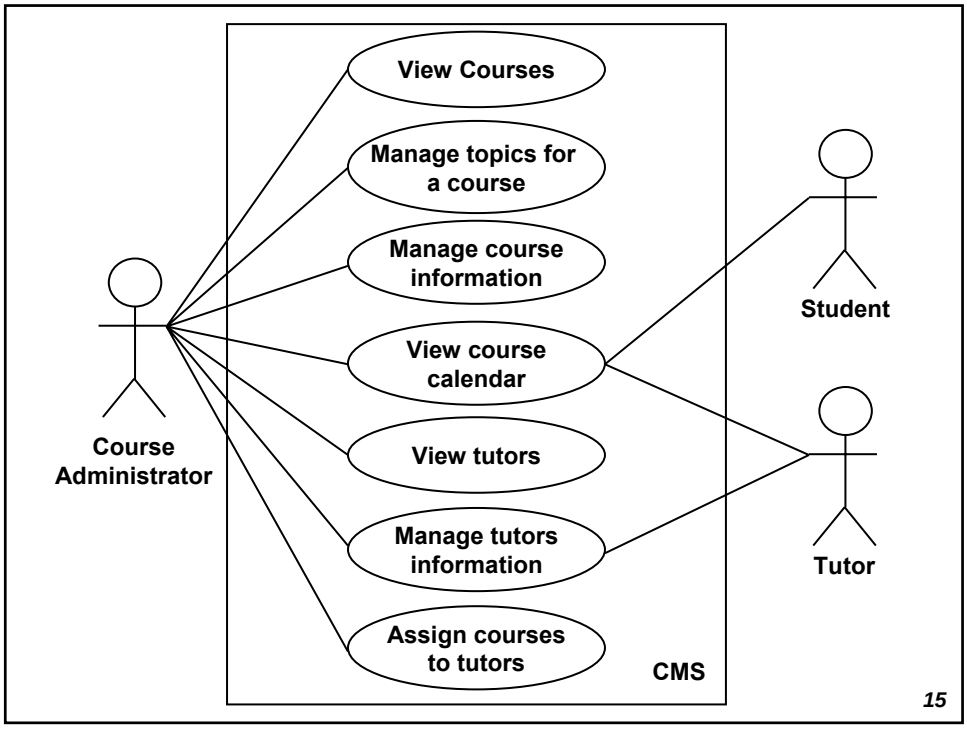
- **Informal Description**:
  - Construct the design elements for the Courseware System that can be used to manage courses and classes
  - The organization offers courses in a variety of areas such as learning management techniques and understanding software languages
  - Each course is made up of a set of topics
  - Tutors in the organization are assigned courses to teach according to the area that they specialize in and their availability
  - The organization publishes and maintains a calendar of the different courses and the assigns tutors every year
  - There is a group of course administrators in the organization who manage the courses including course content, assigning courses to tutors, and defining the course schedule

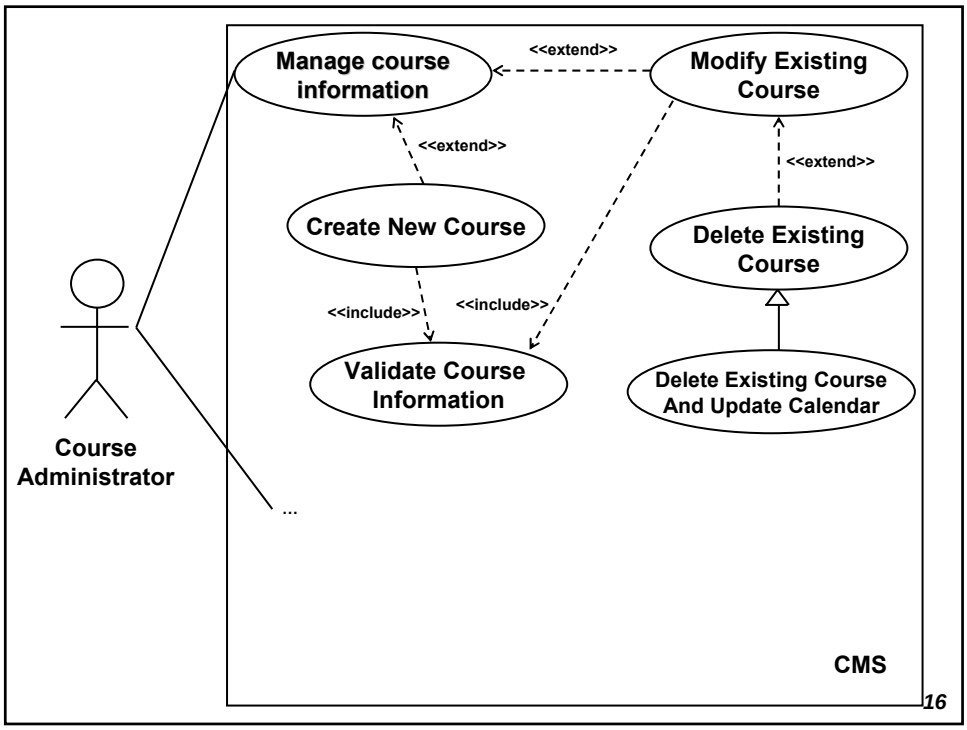
***http://www.developer.com/design/article.php/10925_2109801_4***



## Identifying UML Classes

- Based on the system descriptions, using object-oriented analysis (OOA), identify classes, attributes, and operations
  - For example, nouns / objects that share common properties and are used to enable system functionality become classes
  - Other nouns related to class nouns become class attributes
  - Verbs related to class nouns become class operations
- After identifying classes, identify applicable relationships
  - For example, identify cases where objects of one class reference (are associated with) the objects of another
  - Also identify shared (inherited) behavior between classes


View Courses
Manage topics for a course
Manage course information
View course calendar
View tutors
Manage tutors information
Assign courses to tutors
CMS
Course Administrator
Student
Tutor
15


Manage course information
<<extend>>
Modify Existing Course
<<extend>>
<<extend>>
Create New Course
Delete Existing Course
<<include>>
<<include>>
Validate Course Information
Delete Existing Course And Update Calendar
Course Administrator
...
CMS
16

## Identified Classes

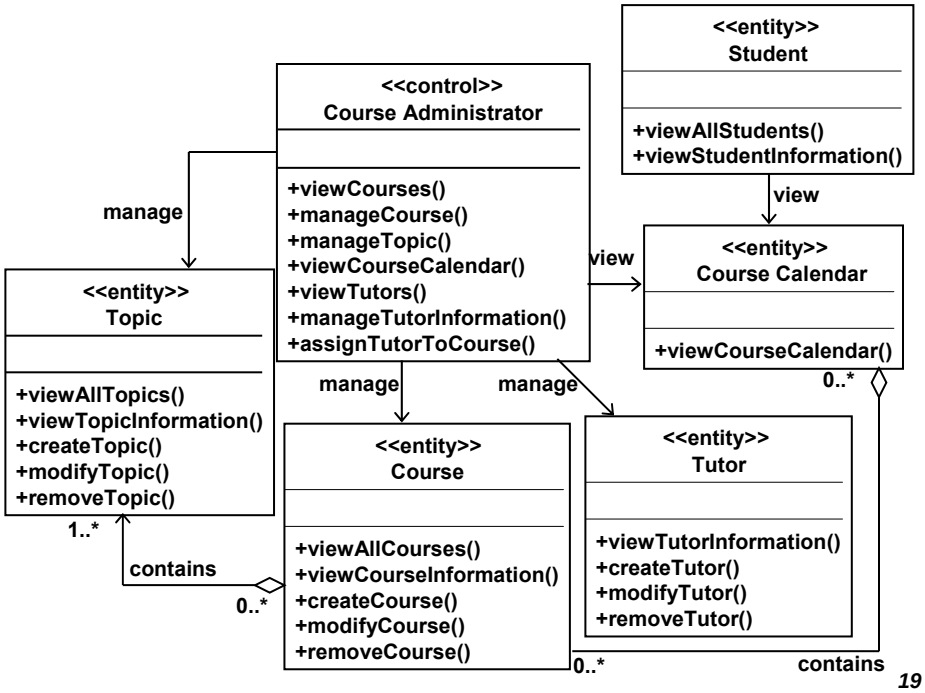
- The following classes are identified:
  - As use case actors:
    - Course Administrator
    - Student
    - Tutor
  - As system objects:
    - Course Calendar
    - Course
    - Topic

*17*

## Identified Class Operations

- The following class operations are identified:
  - For Course Administrator, using use cases:
    - View courses
    - Manage topics for a course (Manage topic)
    - Manage course information (Manage course)
    - View course calendar
    - View tutors
    - Manage tutor information (but **not** manage tutors)
    - Assign courses to tutors (assign tutors to courses)

*18*



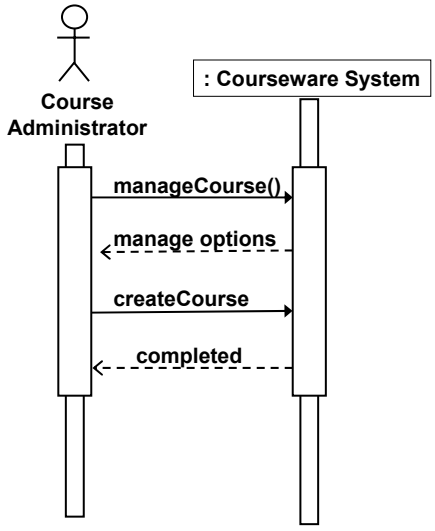

# Using Interaction Diagrams

- **Interaction Diagrams**
  - Represent interaction between class objects based on conditions and operations
  - Can also represent a use case scenario of interaction between actors and the system
- Two main subtypes: sequence and collaboration diagrams
- Sequence diagrams emphasize the temporal order of interaction and show lifetime of each object
- Collaboration diagrams emphasize layout and show interaction as numbering of steps in a scenario

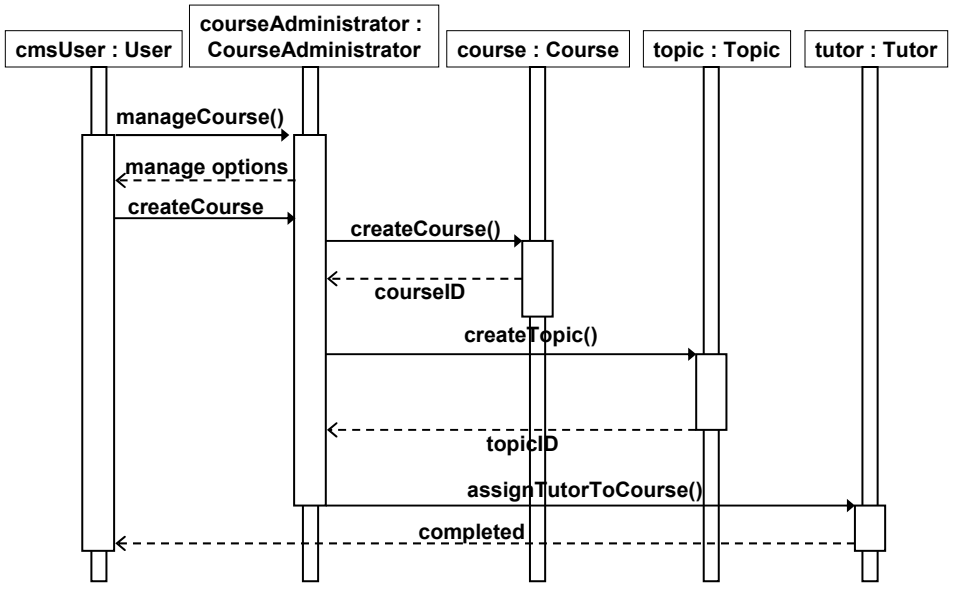
*20*

# Use Case Sequence Diagram

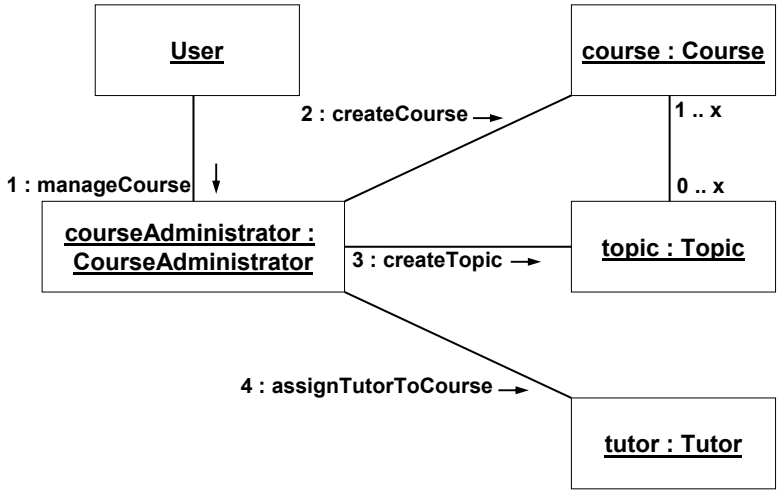
- **Use Case: Manage Course Information (UC_ID1)**
  - Typical flow of events:
    1. Course Administrator selects Create New Course
       a) System invokes Create New Course use case
  - Alternative
    1. Course Administrator selects Modify Existing Course
       a) System invokes Modify Existing Course




*21*




*22*




*23*

# Agenda

- ✓ Terms and Concepts
- ✓ UML Class and Interaction Diagrams Example
- ➢ Summary and References

*24*

## Tutorial Summary

- In this tutorial, we have introduced UML class diagrams
- We have revisited key elements of class diagrams including attributes, operations, generalization, and associations
- Through examples, we have demonstrated how to create complete UML class diagrams
- We have also shown how to demonstrate interaction between UML classes using UML sequence and collaboration diagrams



## References


- G. Booch, J. Rumbaugh and I. Jacobson. *The UML User Guide*, Addison-Wesley, 1999.
- M. Fowler and K. Scott. *UML Distilled*, Addison-Wesley, 2000.
- B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering*, Prentice Hall, 2004.