OO Techniques & UML Class Diagrams
SE3A04 – Tutorial

Jason Jaskolka

Department of Computing and Software
Faculty of Engineering
McMaster University
Hamilton, Ontario, Canada
jaskolj@mcmaster.ca

October 9, 2012

Outline

1. Review of Object-Orientation

2. UML Notation for Class Diagrams

3. Examples

4. Questions

Object-Orientation

Definition (Class)

A **class** is a construct that is used as a blueprint to create instances of itself. Classes usually contain a list of attributes (state variables) and a list of methods.

Definition (Object)

An **object** is an instance of a class. Objects of the same class share the same set of attributes, yet will typically differ in what those attributes contain.

Definition (Method)

A **method** is a procedure or function associated with a class which enable a class object's behaviour.

UML

Definition (UML)

The **Unified Modelling Language (UML)** is a means of precisely expressing requirements, analysis models and designs, in a platform independent manner.

- UML is the result of a successful unification of the many object-oriented analysis and design notations which arose during the 1980's and 1990's
- Version 1.1 was endorsed by the **Object Management Group (OMG)**, representing many companies and organizations, as a standard for object-oriented analysis and design
- A major revision of UML, to version 2.0, was published in 2004
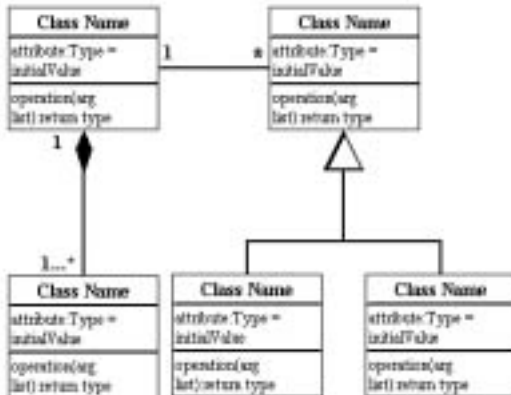- As of March 2011, UML is in version 2.4 beta

Class Diagrams

Definition (Class Diagram)

Class diagrams are the backbone of almost every object oriented method, including UML. They describe the static structure of a system.
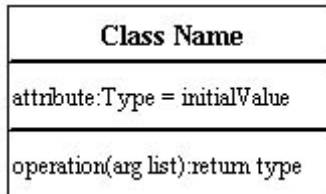
- Purpose is to depict the classes within a model
- Classes have attributes, operations and relationships with other classes
  - All can be graphically depicted using UML class diagrams
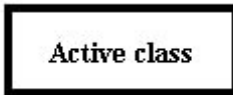
Class Diagrams

Class

- **Classes** represent an abstraction of entities with common characteristics
- Illustrate classes with rectangles divided into compartments
- Place the name of the class in the first partition (centred, bolded, and capitalized)
- List the attributes in the second partition
- Write operations into the third partition

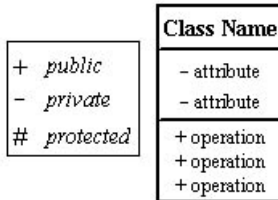| **Class Name** |
|---|
| attribute:Type = initialValue |
| operation(arg list):return type |

Active Class

- **Active classes** initiate and control the flow of activity, while passive classes store data and serve other classes
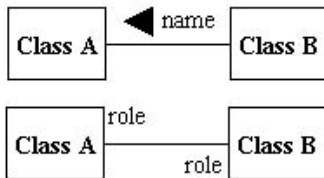- Illustrate active classes with a thicker border

Visibility

- Use visibility markers to signify who can access the information contained within a class
- **Private visibility** hides information from anything outside the class partition
- **Public visibility** allows all other classes to view the marked information
- **Protected visibility** allows child classes to access information they inherited from a parent class
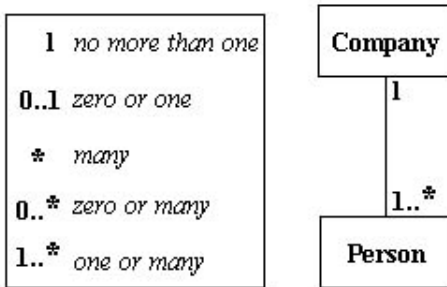
Associations

- **Associations** represent static relationships between classes
- Place association names above, on, or below the association line
- Use a filled arrow to indicate the direction of the relationship
- Place roles near the end of an association
- **Roles** represent the way the two classes see each other
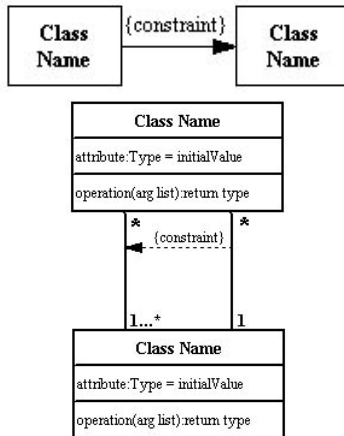- **Note**: It is uncommon to name both the association and the class roles

Multiplicity

- Place multiplicity notations near the ends of an association
- These symbols indicate the number of instances of one class linked to one instance of the other class
- For example, one company will have one or more employees, but each employee works for one company only
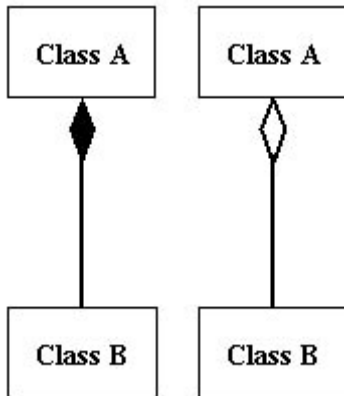
Constraints

- Place constraints inside curly braces {}

Aggregation and Composition

- **Aggregation** is a relationship in which the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other

- Use a hollow diamond to represent a simple aggregation relationship

- **Composition** is a special type of aggregation that denotes a strong ownership between Class A, the whole, and Class B, its part

- Illustrate composition with a filled diamond

- The diamond end in both a composition and aggregation relationship points toward the "whole" class or the aggregate
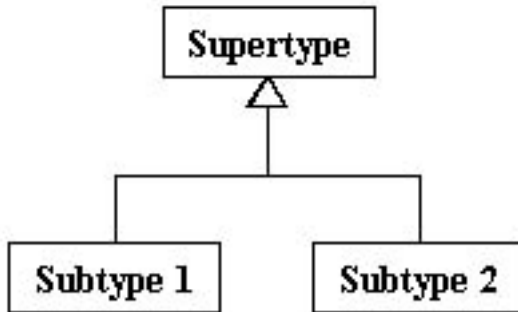
Aggregation and Composition

Generalization

- **Generalization** is another name for *inheritance* or an "is a" relationship
- It refers to a relationship between two classes where one class is a specialized version of another
- Use a hollow arrow head to denote a generalization relationship
- For example, a *Circle* and a *Square* are both shapes; so the class *Circle* and *Square* would have a generalization relationship with the class *Shape*
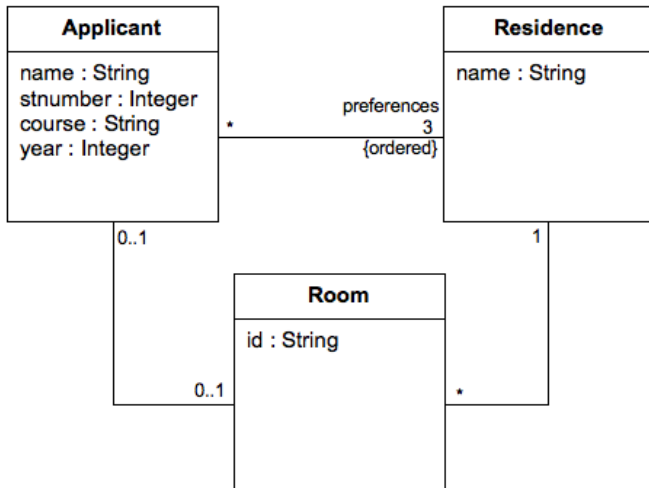
Generalization

Allocation of Student Rooms

Example (Allocation of Student Rooms)

Draw a class diagram to represent the following system:

- Over the summer holiday, university students can book college hall accommodation online. They must specify their name, student number, course, year, and identify three college residences as their preferences.

- The system makes an allocation of students to rooms before the start of the term, trying, where possible, to allocate students to a room in one of their preferred halls.
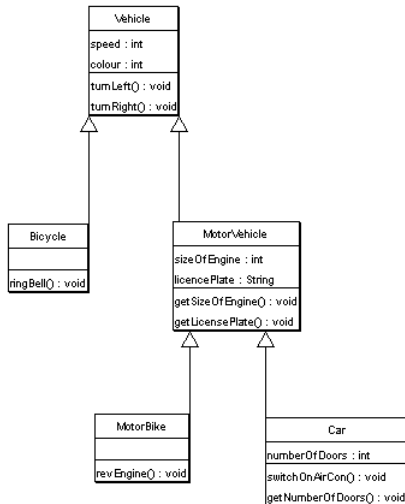
Allocation of Student Rooms

Vehicles

Example (Vehicles)

Draw a class diagram to represent an imaginary application that must model different kinds of vehicles such as bicycles, motor bikes and cars.
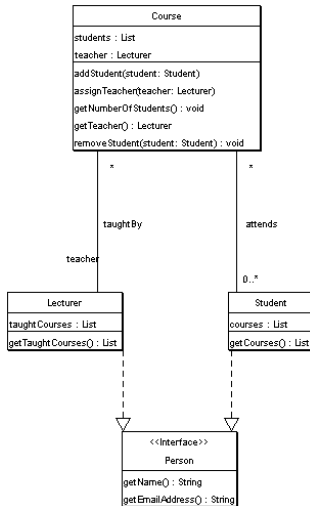
Vehicles

University Courses

Example (University Courses)

Draw a class diagram for an imaginary application that models university courses.
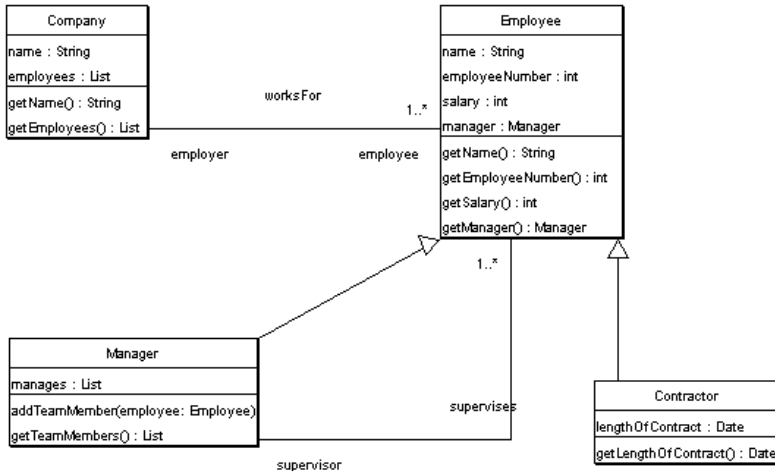
University Courses

Company Payroll

Example (Company Payroll)

Draw a class diagram that models an imaginary company payroll system.
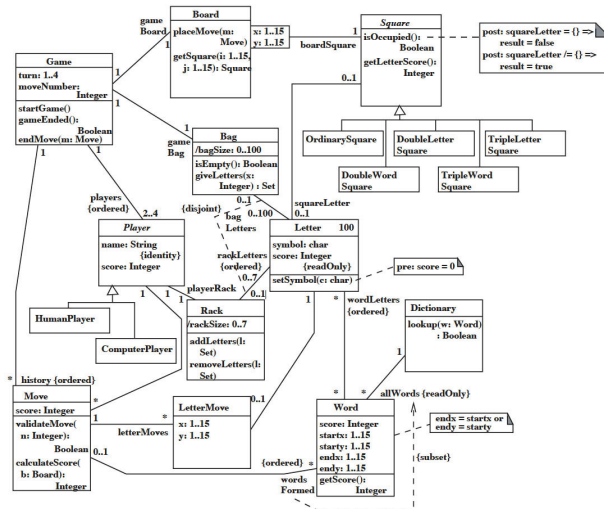
Company Payroll

Scrabble

Example (Scrabble)

Draw a class diagram for a Scrabble system.

Scrabble

Questions

- Questions?