Lecture 11: Class Diagrams

Kenneth M. Anderson
Object-Oriented Analysis and Design
CSCI 6448 - Spring Semester, 2001

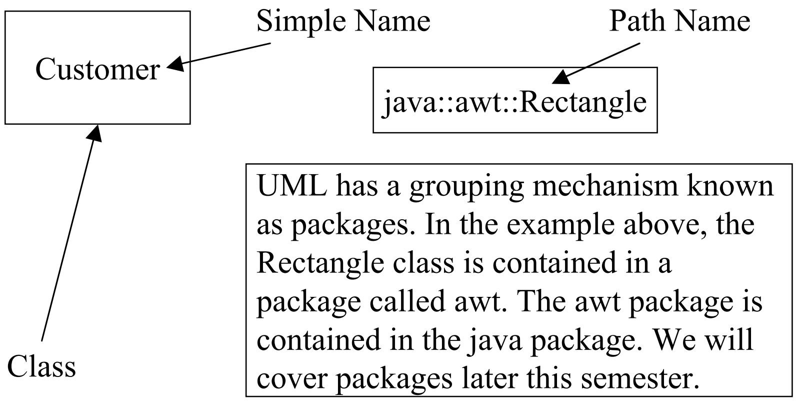
Goals for this Lecture

- Examine Classes In Depth
  - Including associations
- Review UML Notation for Class Diagrams

Classes

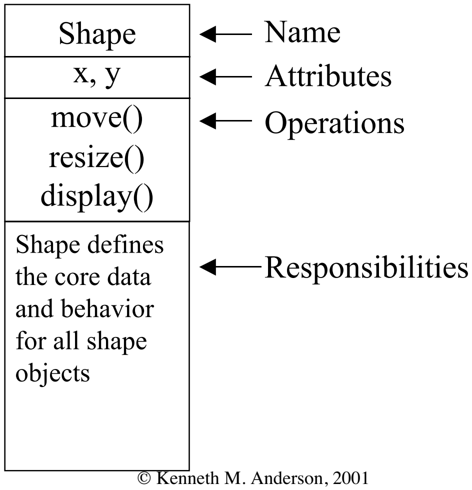
- Classes are used to identify the common characteristics of a particular type of individual
  - Recall from lecture 3 that in order to understand a domain, we need a unique set of individuals from which we can build our descriptions
- In UML, classes consist of
  - Names, Attributes, Operations, Responsibilities
  - and participate in various types of associations

Class Diagram: Basics



UML has a grouping mechanism known as packages. In the example above, the Rectangle class is contained in a package called awt. The awt package is contained in the java package. We will cover packages later this semester.

Class Diagram: Basics, continued

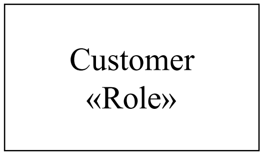


Attributes and Operations

- UML Attributes/Operations start with names
  - x, y, move(), draw()
- You can then add types and parameters
  - x: integer
  - intersect(x, y): boolean
- Finally, you can add defaults
  - x : integer = 5
  - intersect(x: integer = 0, y: integer = 0): boolean
- They can be organized using stereotypes, as shown on page 52 of the UML User Guide
- Note: The *signature* of an operation consists of its name, parameters, and return type

Customizing with Stereotypes

- Stereotypes are a UML extension mechanism
  - You will see them used pretty much everywhere to customize and extend the basic UML notation
  - For instance, you can indicate that a class represents a particular user role with stereotypes like this



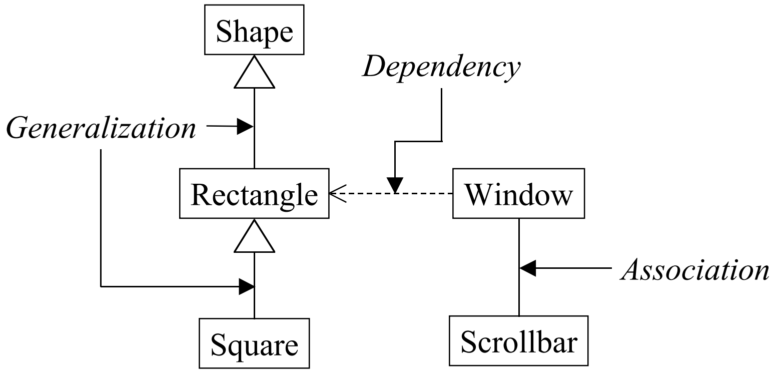
Classes in Analysis and Design

- After domain analysis, your designations serve as excellent candidates for classes
  - Start by identifying the names of the most important classes and list responsibilities
  - As analysis continues, you can add attributes and operations (without types)
  - In design, you will flesh out the classes with more information such as types and method signatures
- Your classes, thus, set the scope for your object-oriented designs

Relationships

- Classes can participate in many types of relationships
  - Generalizations
    - As described in lecture 7
  - Dependencies
    - Similar to a module “uses” relationship
    - A class requires another class to function correctly
  - Associations
    - Structural Relationships among instances

Examples



Generalization

- A parent-child relationship
  - The child shares features with the parent but may add additional attributes and behavior
  - A child can substitute for a parent
  - A child can also override the behaviors of a parent but this conflicts with substitutability
- Also known as an “is-a” relationship
  - A rectangle is a shape
  - A square is a rectangle
- A class with no parents is a root class; A class with no children is a leaf class

Dependency

- A dependency is a “using” relationships that asserts that a change in one class may affect another class that uses it
  - A typical instance of a dependency is when a class appears as an argument in the signature of another class
  - If a class has multiple dependencies, you can distinguish among them using stereotypes

Association

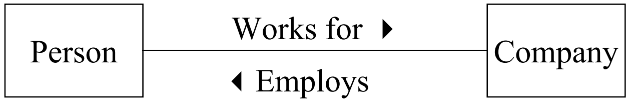
- An association is a structural relationship between instances of classes
  - objects of one class are connected to objects of another class
  - Given an association, you can navigate from an object of one class to an object of the class at the other end
  - It is legal for a class to have an association that begins and ends with itself
    - Examples: stacks, queues, and lists

More on Associations

- Associations can have “adornments”
  - name
  - role
  - multiplicity
  - aggregation

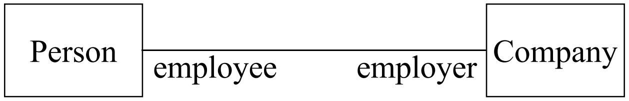
Association Names

- Names can be used to indicate the nature of the association
  - An arrow can be used to indicate the direction of the relationship
    - Typically, names are not reversible
      - to change the direction of the arrow, a new name must be used



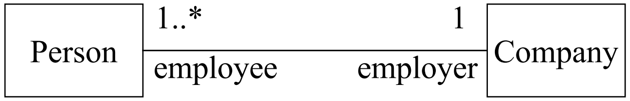
Association Roles

- A class that participates in an association plays a particular role
- These roles can be given explicit names
  - and may lead to the creation of interfaces
- A class can participate in multiple associations (and thus roles) at once



Multiplicity

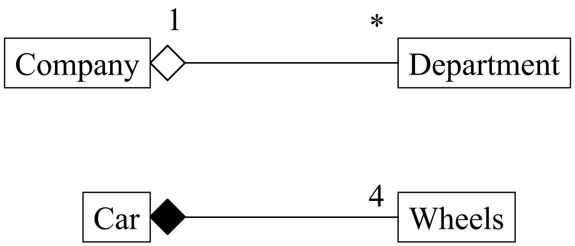
- An association represents a structural relationship among objects
  - You can specify how many objects participate in a particular association using multiplicity
  - To interpret a multiplicity always assume a “1” is at the opposite end of the association, for example,
    - a person may have only one employer
    - a company may have one or more employees



Aggregation

- Some associations have a “whole/part” type relationship, or a “has-a” relationship
  - These are known as aggregations
    - indicated with a diamond at the “whole” end
    - a white diamond is a “simple” aggregation and does not imply a relationship between the lifetimes of the objects
    - a black diamond is a “composition,” a stronger form of aggregation which does imply a relationship between the lifetimes of the objects
      - e.g. destroy the whole and you destroy the parts
  - Note: this information “overrides” the information I presented in lecture 7

Examples



Class Activity Session

- Create the following class diagrams
  - An alphabet with 26 letters
  - A department that employs multiple types of employees: professors, admins, office managers, and graduate students; professors manage students; office managers manage admins; each professor has an admin that assists them
  - An e-mail program that can contain multiple mailboxes, each with multiple messages
  - A class hierarchy of Shapes (including ovals, rectangles, arcs, lines, and points) with a special Shape known as a connector that can connect any two shapes
  - Finally, write a textual description of the class diagram that Dr. Anderson will show in lecture