# Object-Oriented Modeling with UML

Lecture # 29

## Object-Oriented Modeling Methods

- Shlaer/Mellor - 1988
- Coad/Yourdon – 1991
- Booch - 1991
- OMT by Rumbaugh et. al. – 1991
- Wirfs-Brock – 1991
- And many more

## Unification Effort

- 1994 key researchers Grady Booch, James Rambaugh, and Ivar Jacobson joined hands to unify their respective methodologies
- In 1997 Object Management Group (OMG) finally approved the Unified Modeling Language (UML)

## Unified Modeling Language - 1

- Visualizing, specifying, constructing, and documenting object-oriented systems is exactly the purpose of the unified modeling language or UML
- The rules of UML focus on the conceptual and physical representation of a system

## Unified Modeling Language - 2

- Process independent
- Notation has well-defined semantics
- It has become the de-facto standard for modeling
- Many vendors provide tools that support different modeling views

## Unified Modeling Language - 3

- UML provides a very rich set of concept areas
  - Static structure
  - Dynamic behavior
  - Implementation constructs
  - Model organization
  - Extensibility mechanisms

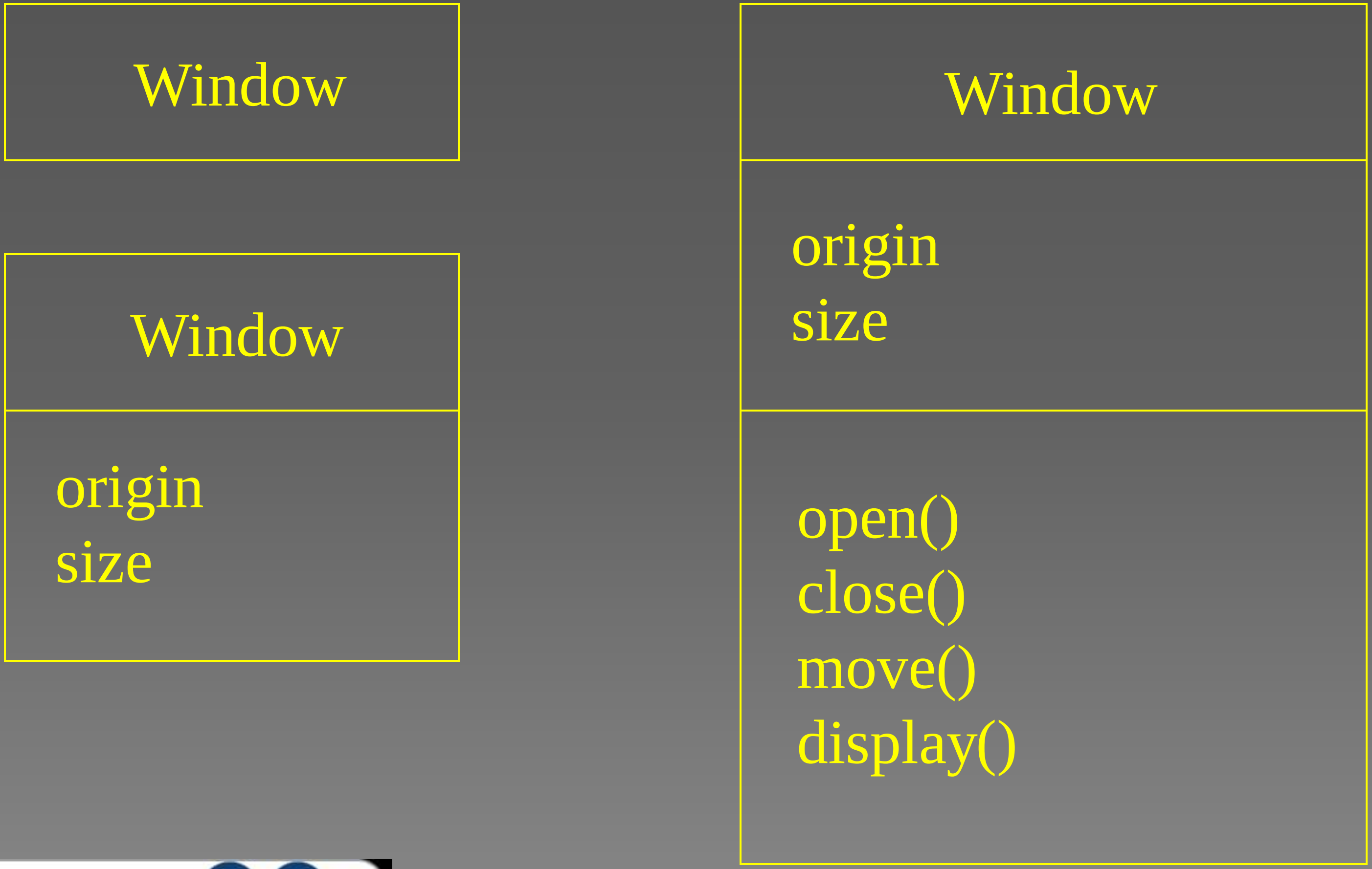
## Static Structure - 1

- Any precise model must first define the universe of discourse, that is, the key concepts from the application, their internal properties, and their relationships to each other
- This set of constructs is the static view

## Static Structure - 2

- The application concepts are modeled as classes, each of which describes a set of discrete objects that hold information and communicate to implement behavior
- The information they hold is modeled as attributes; the behavior they perform is modeled as operations

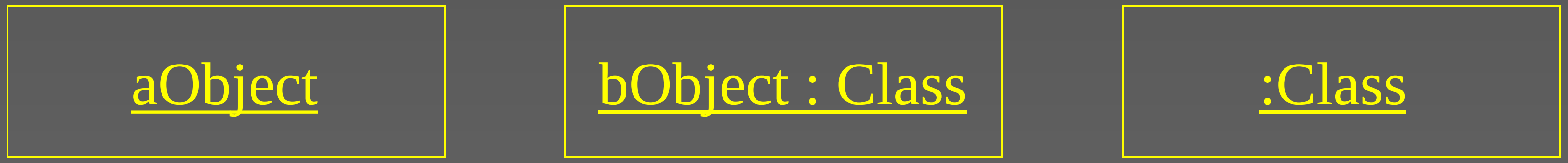
# UML Notation for Classes



## Objects and Classes

- An object is an instantiation of a class
- It has an identity, state, and behavior

# UML Notation for Objects



## Relationships Between Objects - 1

- A relationship is a connection among things. In object-oriented modeling, the three most important relationships are dependencies, generalizations, and associations
- Graphically, a relationship is rendered as a path, with different kinds of lines used to distinguish the kinds of relationships

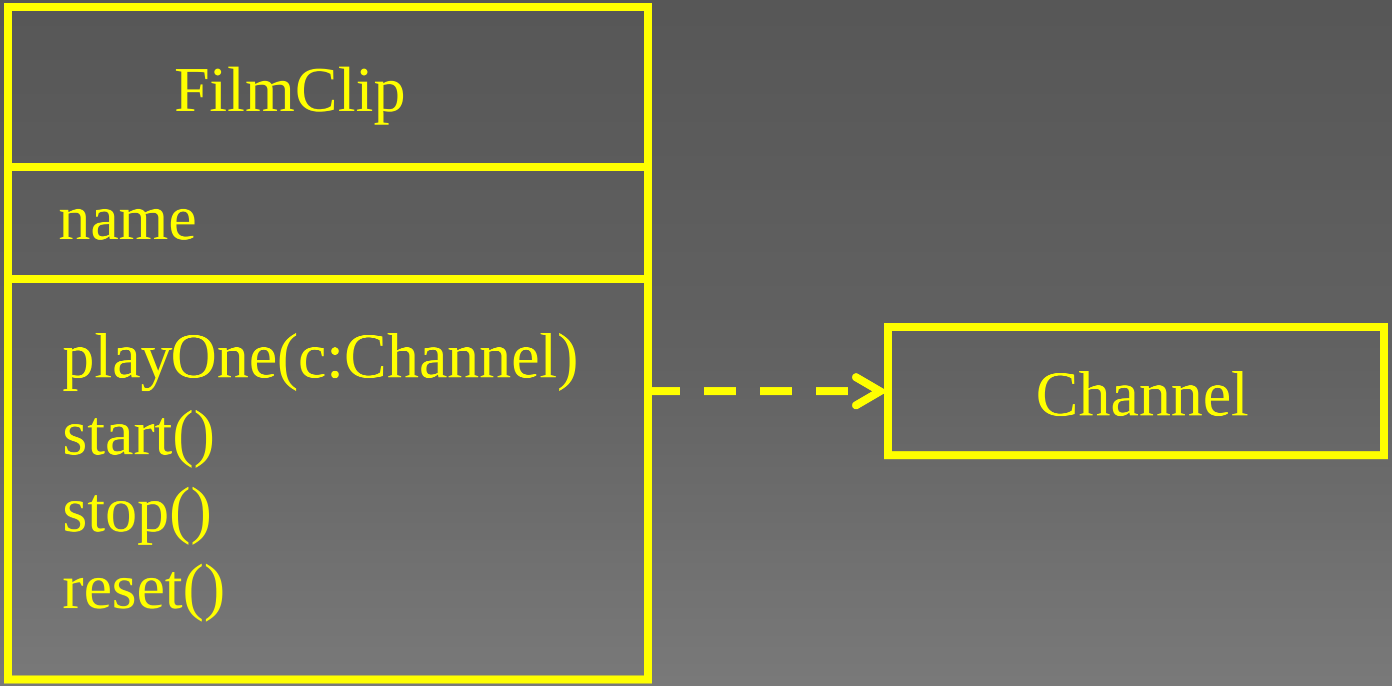
## Relationships Between Objects - 2

- Dependencies, generalizations, and associations are all static things defined at the level of classes
- In the UML, these relationships are usually visualized in class diagrams
- These relationships convey the most important semantics in an object-oriented model

## Dependency Relationship

- A dependency is a using relationship that states that a change in specification of one thing may affect another thing that uses it, but not necessarily the reverse
- Graphically, a dependency is rendered as a dashed line, directed to the thing being depended on
- Use dependencies when you want to show one thing using another thing

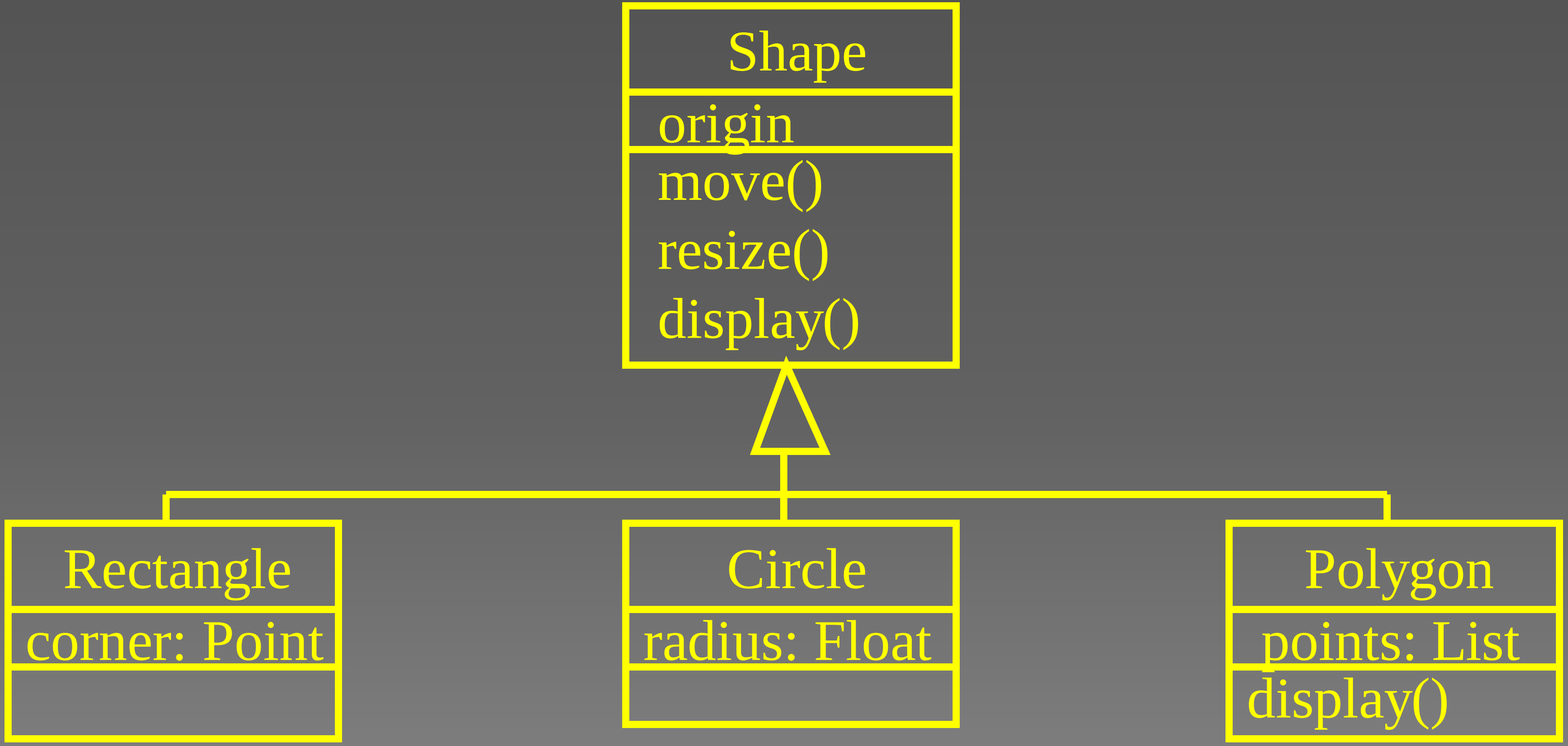
## Dependency Relationship



## Generalization Relationship

- A generalization is a relationship between a general thing (called a super class or parent) and a more specific kind of that thing (called the subclass or child)
- Generalization is sometimes called an ‘is-a-kind-of’ relationship

## Generalization Relationship



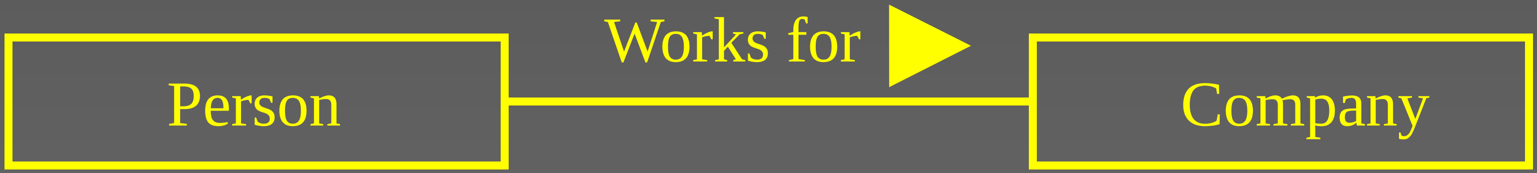
## Association Relationship - 1

- An association is a structural relationship that specifies that objects of one thing are connected to objects of another. Given an association connecting two classes, you can navigate from an object of one class to an object of the other class, and vice versa
- Graphically, an association is rendered as a solid line connecting the same of different classes

## Association Relationship - 2

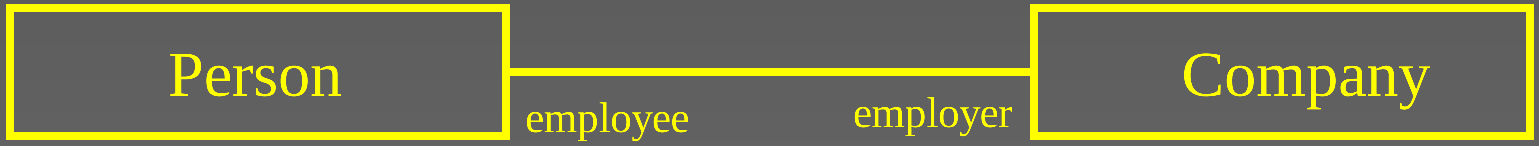
- Use associations when you want to show structural relationships
- An association can have four adornments
  - Name
  - Role
  - Multiplicity
  - Aggregation
    - Captures the ‘whole-part’ relationship
    - Composition – a stronger ‘whole-part’ relationship

## Association Relationship: Name



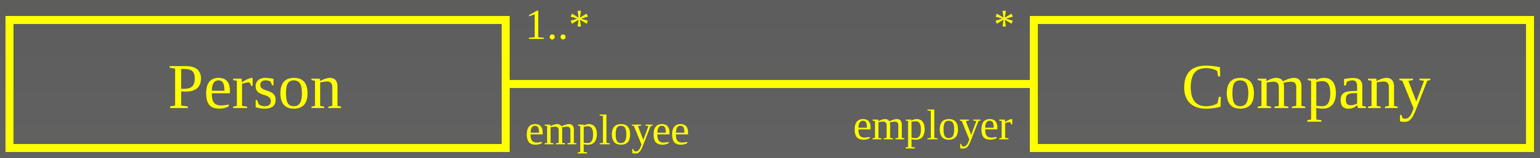
**Association Names**

## Association Relationship: Role



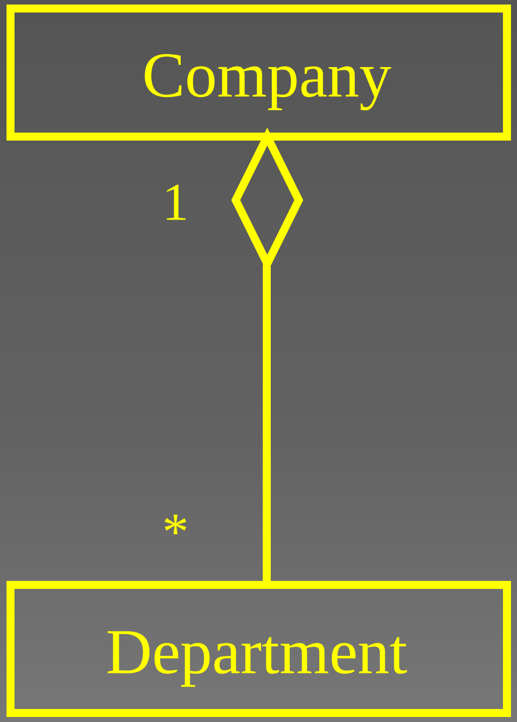
**Roles**

## Association Relationship: Multiplicity



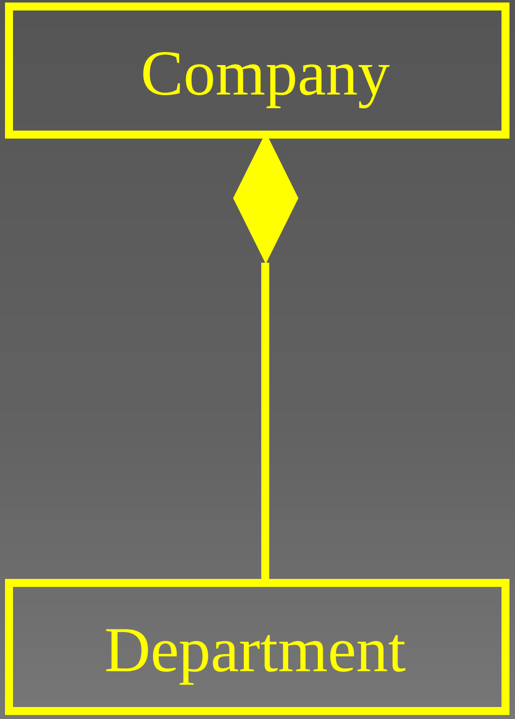
**Multiplicity**

## Association Relationship: Aggregation



**Aggregation**

## Association Relationship: Composition



**Composition**

## Hints and Tips - 1

- Use dependencies only when the relationships you are modeling are not structural
- Use generalization only when you have ‘is-a-kind-of’ relationship; multiple inheritance can often be replaced with aggregation

## Hints and Tips - 2

- Keep your generalization relationships generally balanced; inheritance latices should not be too deep (more than five levels or so should be questioned) nor too wide (instead, look for the possibility of intermediate abstract classes)

## Hints and Tips - 3

- Beware of introducing cyclical generalization relationships
- Use associations primarily where there are structural relationships among objects

## Class Inheritance and Object Composition

# Class Inheritance: Advantages

- Class inheritance is defined statically at compile-time
- Class inheritance is straightforward to use, as it is supported directly by object-oriented programming languages
- Class inheritance makes it easy to modify the implementation being reused

## Class Inheritance: Disadvantages - 1

- You cannot change the implementations inherited from parent class at run-time, because inheritance is defined at compile-time
- Parent classes often define at least part of their subclasses' physical representation. Any change in the parent's implementation will force the subclass to change

## Class Inheritance: Disadvantages - 2

- Inheritance breaks encapsulation
- Implementation dependencies can cause problems when you're trying to reuse a subclass

## Object Composition: Advantages - 1

- Object composition is defined dynamically at run-time through objects acquiring references to other objects
- Composition requires objects to respect each others’ interfaces, that requires you to carefully define interfaces of classes

## Object Composition: Advantages - 2

- Encapsulation is not broken
- Very few implementation dependencies

# Object Composition: Advantages - 3

- Object composition has a positive affect on the system design
  - Classes are encapsulated and focused on one task
  - Classes and class hierarchies will remain small
  - There will be more objects than classes, and the system's behavior will depend on their interrelationships instead of being defined in one class

## *Favor object composition over class inheritance*

## Summary

- We discussed Object-Oriented Modeling using UML
- We can model classes and objects in UML
- The relationships between classes can be modeled using UML notation
- The adornments can be added to the relationships among classes for modeling additional information about the relationships
- Object composition should be favored over class inheritance