

# UML Class Diagrams

- May also contain interfaces, objects, and relationships
- Sometimes called "static structure diagrams"
- [Unified Modeling Language \(UML\)](#),  
*version 2.1.2*
- Examples taken from *UML Reference Manual*

# Classes

- Every behavior a system provides should be associated with a class
- Candidate for classes include domain objects, roles, events, interactions
- Denoted by rectangle (or an icon), normally containing three, vertically stacked, compartments
  - Name
  - Attributes
  - Operations
  - (Responsibilities, exceptions)

# Name Compartment

- Stereotype, if any
  - Extra, meta-model-defined labels
  - Enclosed by «» (*guillemets*)
- Name
  - Italics for abstract classes
- Properties in braces
  - {abstract}, {leaf}, {root}

# Example Class

<p><i>Window</i> {abstract, author=Joe, status=tested}</p>
<p>+size: Area = (100,100) #visibility: Boolean = true <u>+default-size: Rectangle</u> <u>#maximum-size: Rectangle</u> -xptr: XWindow*</p>
<p><u>+display ()</u> <u>+hide ()</u> <u>+create ()</u> -attachXWindow(xwin:Xwindow*)</p>

# Attributes

- **Visibility**
  - + (public), - (private),  
# (protected), ~ (package)
- **Name**
- **Optional multiplicity and ordering**
- **Type**
- **Initial value**
- **Derivation (/)**
- **Properties**
  - Class scope (underlining)
  - {frozen}, {changeable}, {addOnly}

# Operations

- Visibility
- Name
- Return type
- Parameter list
  - Kind (`in`, `out`, `inout`), name, type, default value
- Properties
  - Class scope
  - `{query}`
  - `{concurrency} = sequential | guarded | concurrent`
  - `{abstract}`
- `<<signal>>`

# More Example Classes

<b>Rectangle</b>
p1:Point p2:Point
«constructor» Rectangle(p1:Point, p2:Point) «query» area (): Real aspect (): Real ... «update» move (delta: Point) scale (ratio: Real) ...

<b>Reservation</b>
<b>operations</b> guarantee() cancel () change (newDate: Date)
<b>responsibilities</b> bill no-shows match to available rooms
<b>exceptions</b> invalid credit card

# Interfaces

- Interfaces
  - Implementers append labeled "knob"
  - <<use>>
  - <<interface>>
  - Realization relationship
    - Triangle with dashed line

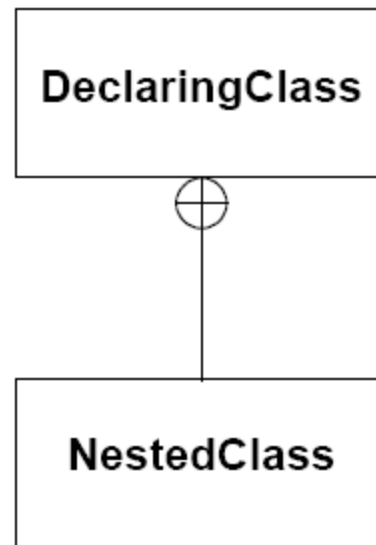


# Parameterized Classes

- One or more type parameters (templates)
  - Dashed rectangle overlaying upper right hand corner
  - `<<bind>>` realization relationship

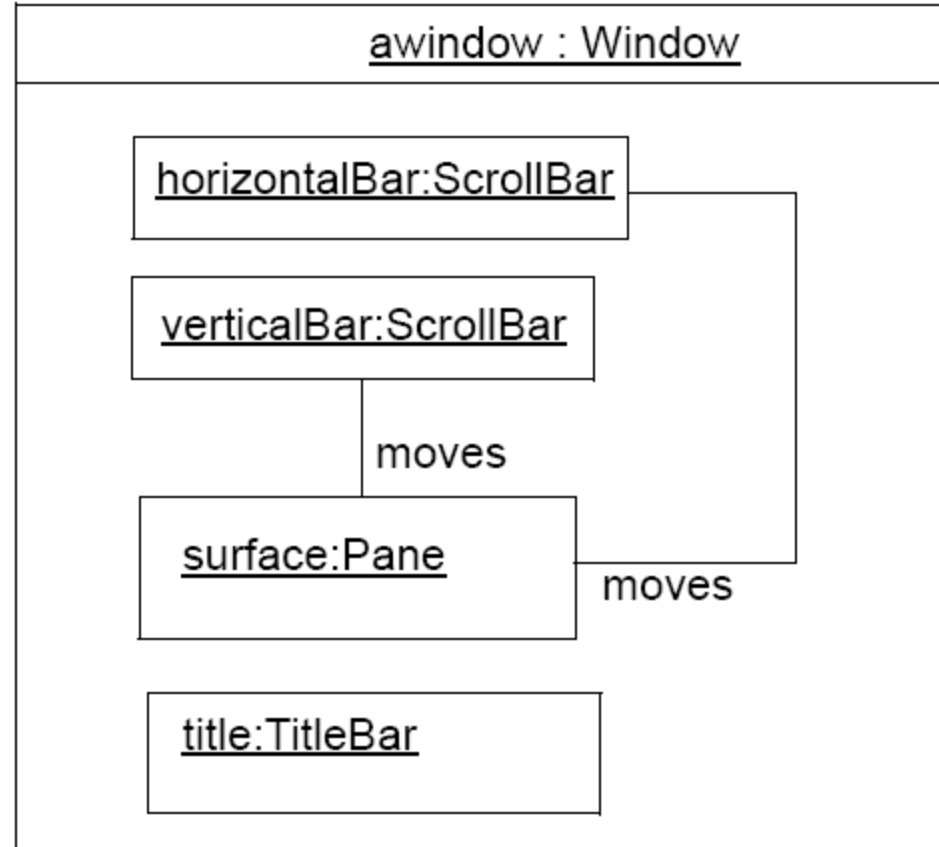
# Nested Classes

- Nested: Separate class rectangles with anchor



# Composite Objects

- Nesting mechanism
- Class diagrams within class rectangle



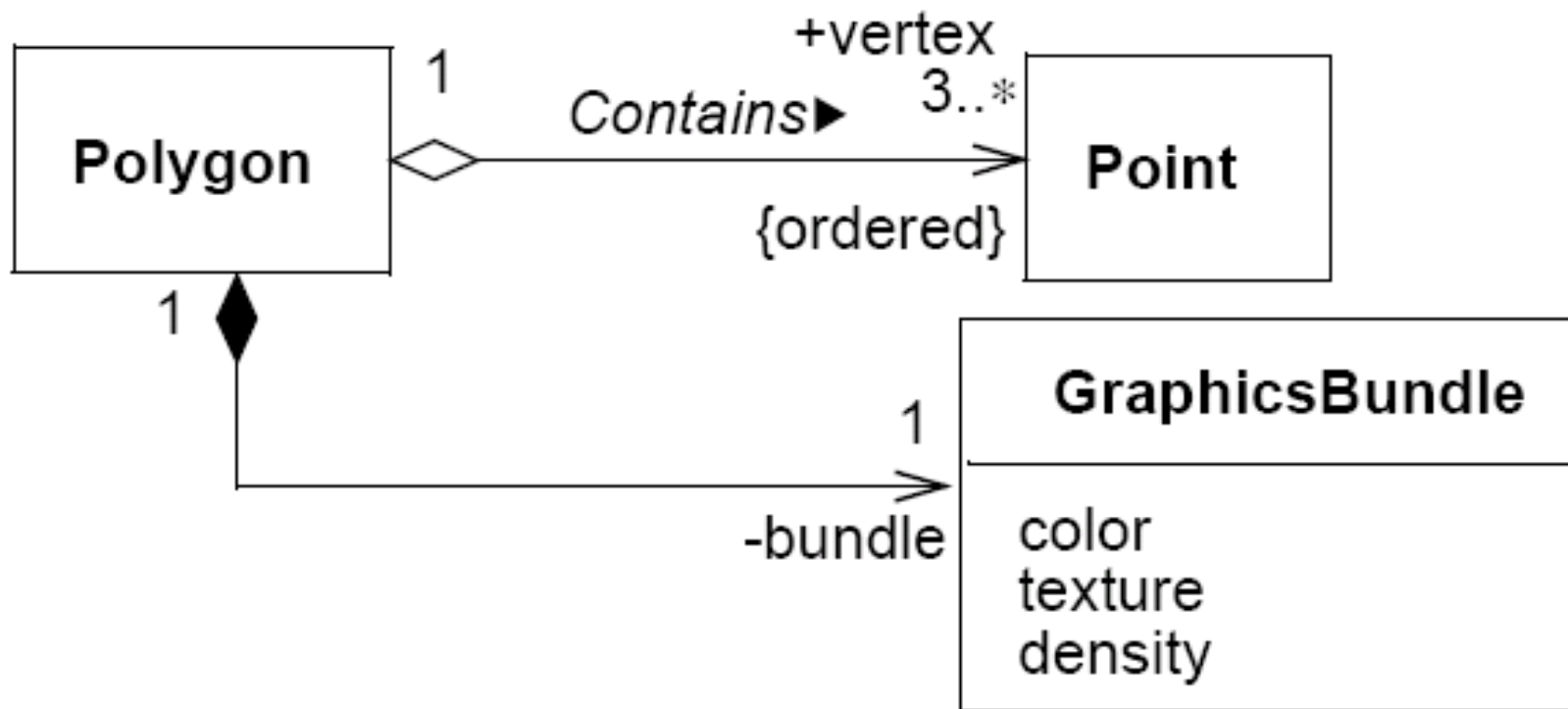
# Relationships

- Association
  - Aggregation
  - Composition
  - Solid, labeled line, possibly directional via arrowhead
- Generalization
  - Solid line with triangle
- Dependency
  - Dashed line with arrowhead
- Realization
  - Between specification and implementation
  - Dashed line ending in a triangle

# Associations

- Name
- Class (for association classes)
- Aggregation (open diamond) or composition (filled diamond)
- Reading direction (filled triangle next to name)
- Navigatability (arrowhead)
- Multiplicity, ordering
- Ternary and higher (rhombus)
- Role names
- Qualification (rectangle on end)

# Association Example

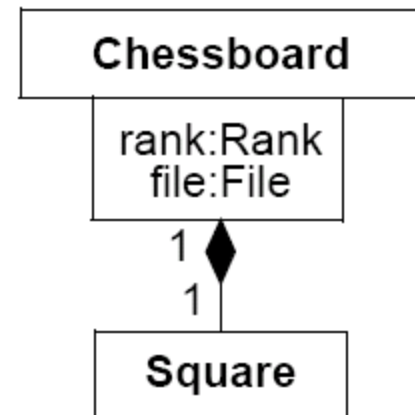
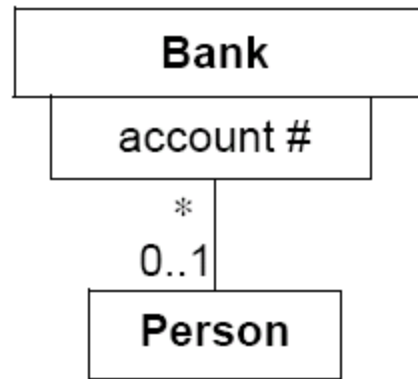


# Association Adornments

- Reading direction
- Name
- Roles
- Multiplicity
- Aggregation / composition
- Association constraints
  - `implicit`, `ordered`, `changeable`,  
`frozen`, `addOnly`

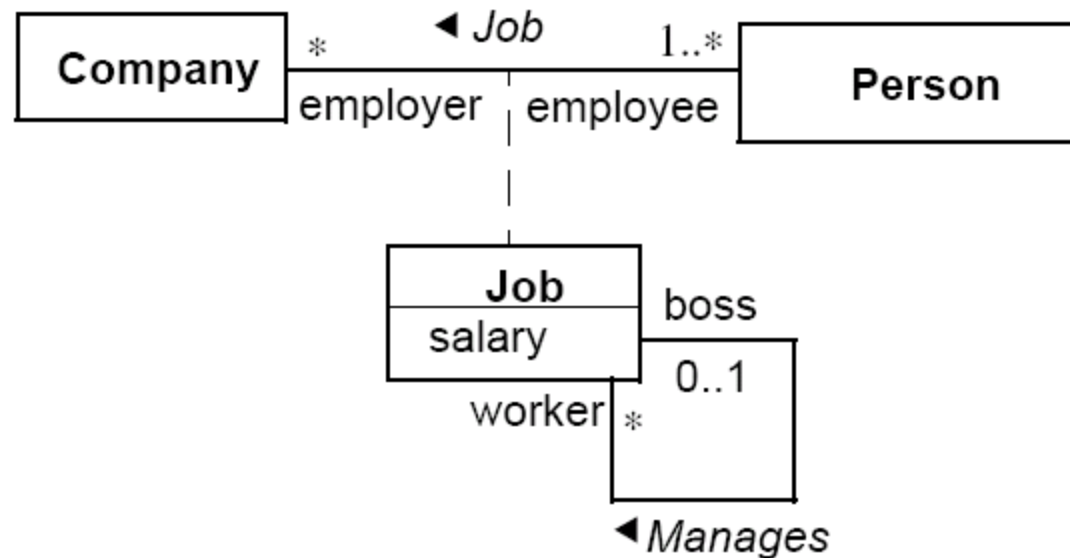
# Qualifiers

- Attribute or attributes used to partition a set of instances across an association
  - Like keys; serves as an indexing mechanism



# Association Class

- An association class is an association that also has class properties (or a class that has association properties)

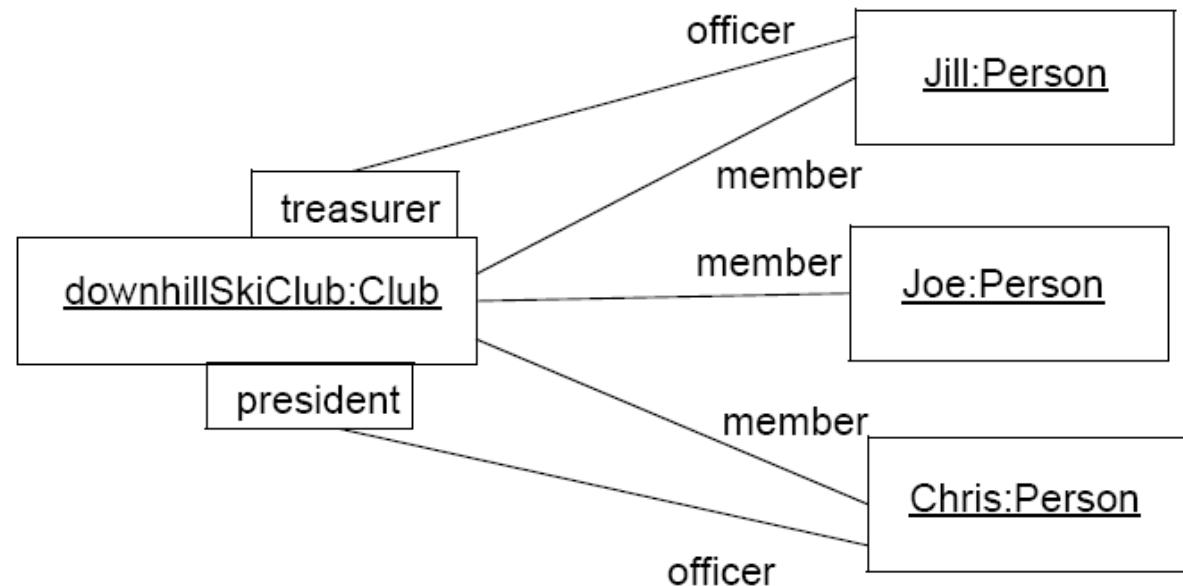


# Composition

- Aggregation by itself does not impose any semantic restrictions on the classes involved, it just implies a whole-part relationship
- Composition is a stronger form of aggregation (filled diamond) that does impose restrictions
  - A class can only belong to one composition
  - Responsibility for management (lifetime)
  - Transitive
- An attribute of class C with type A implies a composition association between C and A
  - Choose one or the other (association for analysis diagram, attribute for design diagram)

# Links

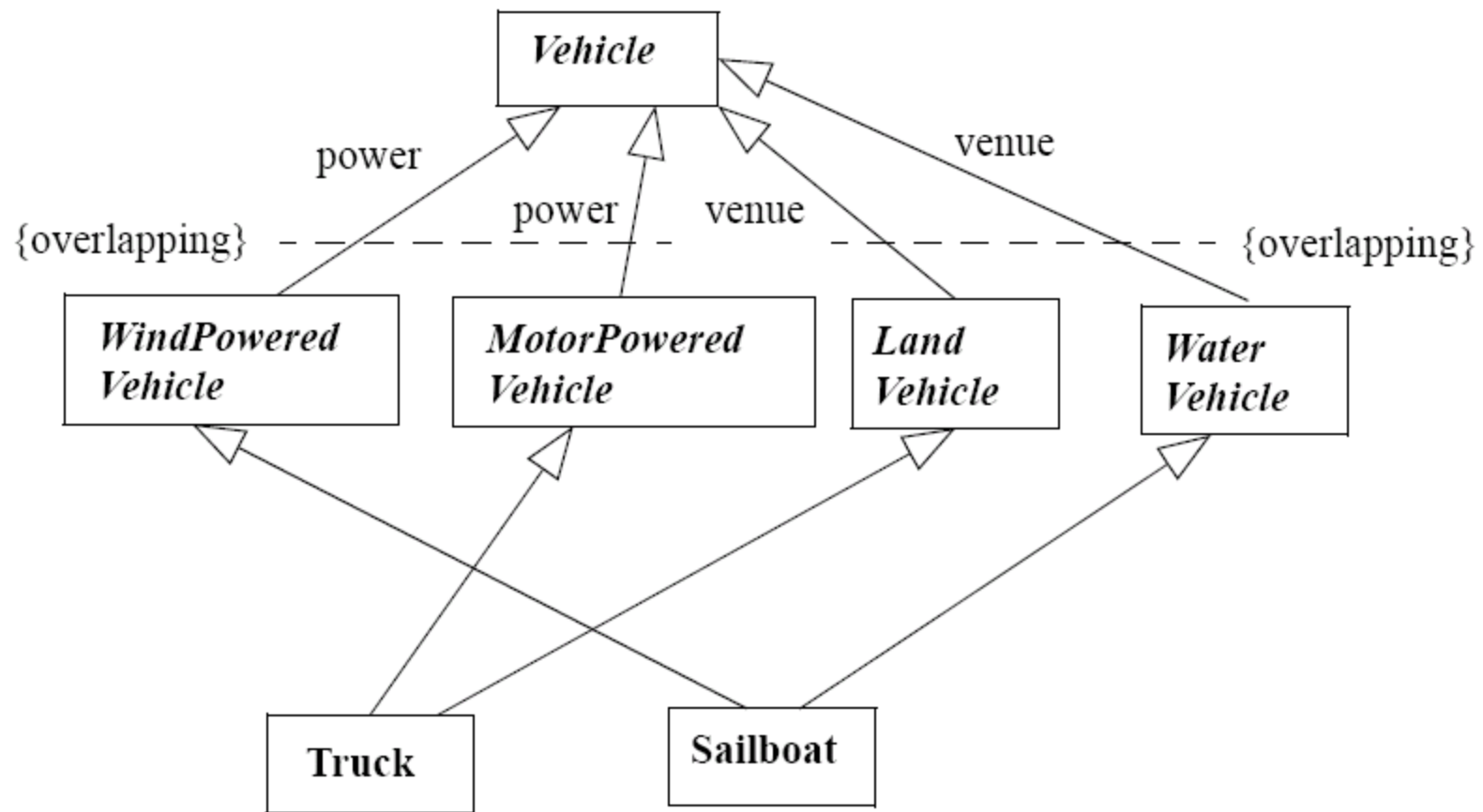
- Instances of an association
- Stereotypes
  - «parameter» - method parameter
  - «local» - local variable
  - «global» - global variable
  - «self» - message sent to self



# Generalization

- Liskov Substitutability Principle
  - *If for each object  $o_1$  of type  $S$  there is an object  $o_2$  of type  $T$  such that for all programs  $P$  defined in terms of  $T$ , the behavior of  $P$  is unchanged when  $o_1$  is substituted for  $o_2$ , then  $S$  is a subtype of  $T$*
- Discriminator (label)
- Constraints
  - {overlapping}
  - {disjoint}
  - {complete}
  - {incomplete}
- Multiple specialization allowed

# Generalization Example



# Dependency

- *A dependency indicates a semantic relationship between two model elements. ... It indicates a situation in which a change to the target element may require a change to the source element in the dependency. (UML v1.5 Specification)*

# Predefined Dependencies

<b>Keyword</b>	<b>Name</b>	<b>Description</b>
access	Access	The granting of permission for one package to reference the public elements owned by another package (subject to appropriate visibility)
bind	Binding	A binding of template parameters to actual values to create a nonparameterized element
derive	Derivation	A computable relationship between one element and another (one more than one of each)
import	Import	The granting of permission for one package to reference the public elements of another package, together with adding the names of the public elements of the supplier package to the client package
refine	Refinement	A historical or derivation connection between two elements with a mapping (not necessarily complete) between them
trace	Trace	A historical connection between two elements that represents the same concept at different levels of meaning
use	Usage	A situation in which one element requires the presence of another element for its correct implementation or functioning

# Kinds of Usage

- An instance of class A sends a message to an instance of class B
- An instance of class A creates an instance of class B
- An instance of class A has an attribute that is an instance of class B
- An instance of class A has a collection of instances of class B
- An instance of class A receives a message with a parameter that is an instance of class B

# Examples of Dependency

