

CHAPTER 7

Class and Object Diagrams

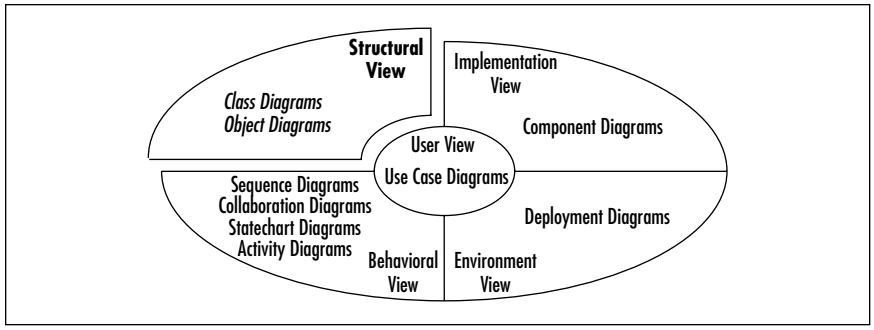


Figure 7-1: Structural Model View—Class Diagrams and Object Diagrams

Class diagrams (Figure 7-1) render the structural view of a system. These diagrams describe the static structure of a system, or how it is structured rather than how it behaves. These diagrams contain classes and associations. They may be constructed using the following technique:

1. Model classes.
2. Model associations between classes.
3. Refine and elaborate as required.

Class diagrams

- May be constructed in conjunction with object diagrams. Object diagrams may be used to explore different configurations of objects, and then combined or generalized into class diagrams. This results in a single class diagram that describes many possible object diagrams.
- May also be called *static structural diagrams* to distinguish them from behavioral diagrams.

- Contain classifiers. Classifiers
  - Are constructs that describe structural features and behavioral features. Structural features include attributes, whereas behavioral features include operations and methods.
  - May participate in associations.
  - Have many specific forms that are designated using stereotype keywords.

  Classes are the most common classifiers. The discussion of classes applies to the other element kinds as semantically appropriate. A type-instance correspondence exists between classifiers and instances, classes and objects, associations and links, and data types and values.
- May contain objects. A class diagram with objects and no classes is an object diagram.
- May have the various expressions (attributes, operations, etc.) they utilize be expressed using pseudocode or another language.

Object diagrams (Figure 7-1) render the structural view of a system at a particular time during its life. These diagrams describe a particular instantiation of a class diagram. These diagrams contain object and links. They may be constructed using the following technique:

1. Model objects.
2. Model links between objects.
3. Refine and elaborate as required.

Object diagrams

- May be considered special cases of class diagrams. That is, class diagrams with only objects and links are object diagrams; such diagrams do not have classes or associations.
- May be constructed in conjunction with class diagrams. Object diagrams may be used to explore different configurations of objects, and then combined or generalized into class diagrams. This results in a single class diagram that describes many possible object diagrams.
- May have the various expressions (attributes, operations, etc.) they utilize be expressed using pseudocode or another language.

Classes

Classes (Figure 7-2) are descriptions of a set of objects with common structural features, behavioral features, relationships, and semantics. They are used to model a set of concepts or entities with common characteristics. Classes

- Are denoted as solid-outline rectangles with compartments.
- Must have a name compartment. This contains a class's name. Class names are typically centered, in boldface, and begin with an uppercase letter. If they are abstract classes, they are in italics.
- May have an attribute compartment named "attributes" (Chapter 6). This contains a class's attributes.

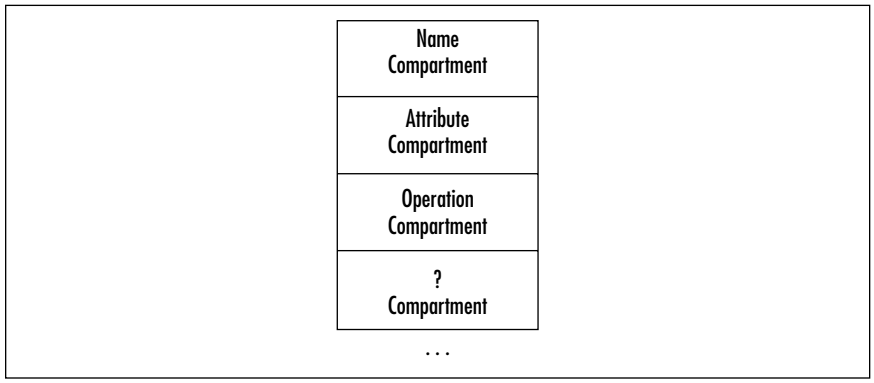


Figure 7-2: A Class

- May have an operation compartment named "operations" (Chapter 6). This contains a class's operations and methods.
- May have other compartments, which may be used for other model properties (business rules, responsibilities, variations, events, exceptions, etc.).
- May be referenced by using pathnames. This indicates that the class is referenced or used in this context rather than being defined.
- May have interfaces to specify collections of operations provided to its environment. All operations must have associated methods.
- May have generalization relationships with other classes.
- May be active. Active classes
  - Own a thread of control. A *thread* is a path of execution or control. A *process* is the physical construct that houses one or more threads. A thread executes through a process. Multiprocessing involves multiple processes. Multithreading involves multiple threads per process.
  - Are denoted by a heavy border.
  - May initiate control activity. That is, they may request services from other objects and classes.
- May be passive. Passive classes
  - React to the requests of other classes. They primarily hold data and provide services to other classes.
  - Are denoted by a normal border.
  - May not initiate control activity unless they are passed control from an active object requesting service.
- May represent signals used in statechart diagrams (Chapter 11) and activity diagrams (Chapter 12). Signals
  - Must be stereotyped using the "signal" keyword.
  - Must have their parameters shown in the attribute compartment.

– Must not have any operations.
– Must not have any relationships to other classes, but may have generalization relationships to other signal classes.

Attributes

Attributes (Figure 7-3) are descriptions of structural or static features of classifiers. They are used to model information associated with an entity; this is what an entity knows.

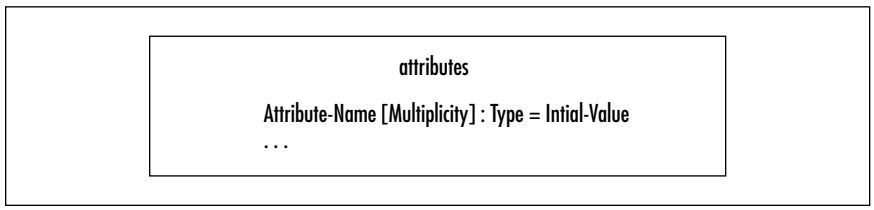


Figure 7-3: Attribute Compartment of a Class

Attributes

- Are denoted as text strings within an attribute compartment.
- Are named slots that describe the types of values that an instance of a classifier may hold.
- May be shown by composition. A class is composed of its attributes.
- Are typically left justified, in plain type, and begin with a lowercase letter.
- Must have a name or identifier string that represents the name of the attribute. Names must be unique within the containing class.
- May have square brackets containing a multiplicity expression that indicates the number of attributes per instance of the class. An attribute without this expression must hold exactly one value.
- May have a colon followed by a type expression that indicates the types of values an attribute may have.
- May have an equal sign followed by an initial value that is used to initialize attributes of newly created objects.
- May have class scope, as denoted by underlining.

Operations

Operations (Figure 7-4) are descriptions of behavioral or dynamic features of classifiers. They are used to model services or operations associated with an entity; this is what an entity may do. Operations

- Are denoted as text strings within an operation compartment.
- Are named services that may be requested of instances of a classifier.
- Are implemented by methods.

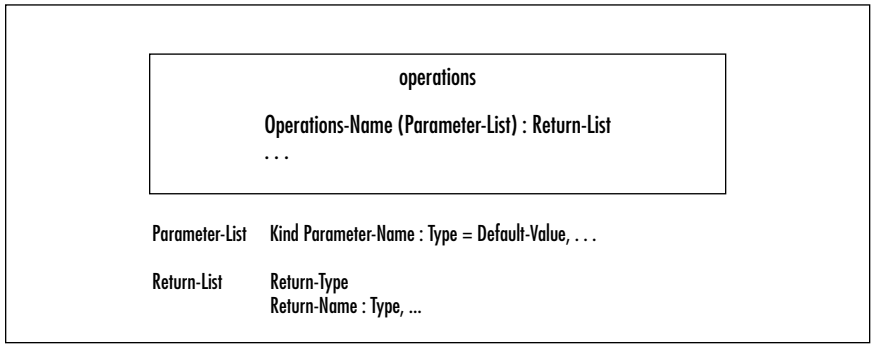


Figure 7-4: Operation Compartment of a Class

- Are typically left justified, in plain type, and begin with a lowercase letter. If they are abstract operations, they are in italics.
- Must have a name or identifier string that represents the name of the operation. Names must be unique within the containing class.
- May have parentheses containing a comma-separated parameter list that indicates the formal parameters passed to a method. When the parameter list is suppressed, the colon and return list must also be suppressed. Parameters
  - May have a kind specified. This value may be "in," indicating that the parameter is only passed into the operation; "out," indicating that the parameter is only returned from the operation; or "inout," indicating that the parameter may be passed into and returned from the operation. The default is "in."
  - Must have a name or identifier string that represents the name of the parameter. Names must be unique within the parameter list.
  - May have a colon followed by a type expression that indicates the types of values a parameter may have.
  - May have an equal sign followed by a default value that is used to set the value for unspecified parameters when the method is invoked.
- May have a return list consisting of a single return type expression.
- May have a return list consisting of a comma-separated list of formal parameters passed back from a method. This indicates that an operation returns multiple values. Return parameters
  - Must have a name or identifier string that represents the name of the parameter. Names must be unique within the parameter list.
  - May have a colon followed by a type expression that indicates the types of values a parameter may have.
- May have class scope, as denoted by underlining.
- May be specified or elaborated (implemented) in notes (using plain text) or other behavioral view diagrams attached to operations.

- May have extension points specifying where additional behavior may be inserted by subclasses.
- May be stereotyped using the "signal" keyword to indicate that objects of a class accept and respond to a given signal that triggers the operation. The response of an object to a signal is shown with a state machine (Chapter 11).

Types

Types (Figure 7-5) are classes that define a collection of operations without methods. They are used to model a collection of operations applicable to different classes of objects. Types

- Are denoted as stereotyped class rectangles.
- Are equivalent to abstract classes with no methods and only abstract operations.
- Must be stereotyped using the "type" keyword.
- Must not contain any methods.
- Are implemented by implementation classes.
- May have generalization relationships to other types.

Implementation Classes

Implementation classes (Figure 7-5) are classes that define the realization or implementation of other classes.

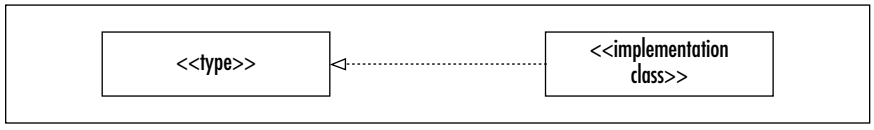


Figure 7-5: Type and Implementation Classes

Implementation classes

- Are denoted as stereotyped class rectangles.
- Must be stereotyped using the "implementation class" keyword.
- May implement other classes via a realizes generalization relationship from the implementation class to the implemented class. The implementation class inherits operations but not structure (attributes or associations).
- May have generalization relationships to other implementation classes. These are generalization relationships between implementation classes rather than generalization relationships between an implementation class and a nonimplementation class.

Interfaces

Interfaces (Figure 7-6) are classes that define a set of externally accessible operations without methods. They are used to model a collection of operations defining a service that may be offered by different classes.

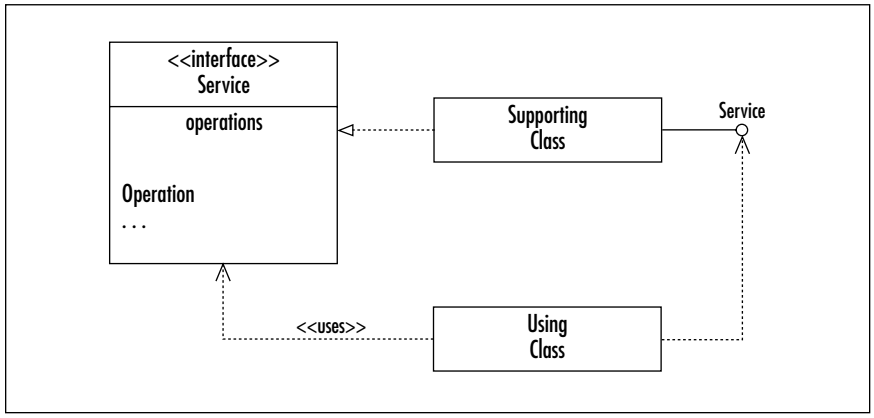


Figure 7-6: Interfaces

Interfaces

- Are denoted as stereotyped class rectangles, or as small circles with the name of the interface attached.
- Are equivalent to abstract classes with no attributes and no methods and only abstract public operations.
- Must be stereotyped using the "interface" keyword.
- Must not have an attribute compartment. The attribute compartment is omitted because it is always empty.
- Must not have associations.
- Must have an operation compartment named "operations." This contains an interface's operations.
- Must have only public operations.
- May have generalization relationships with other interfaces.
- May be supported by other classes via a realizes generalization relationship from the supporting class to the interface class. The supporting class must offer all the operations defined in the interface class, and may offer new operations, but does not have to support any of the data structures of the interface class.
- May be used by other classes via a dependency stereotyped using the "uses" keyword.

Templates

Templates (Figure 7-7) are constructs defining a family of classes having a common form. The differences between classes in the family are represented by parameters, allowing a single template to instantiate many classes. Specific classes within this family are defined by specifying actual values that are bound to formal parameters. In the template, the formal parameters are unbound or unspecified.

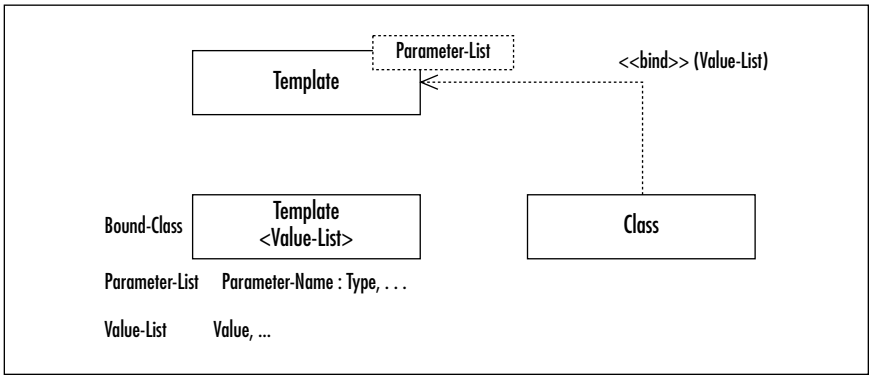


Figure 7-7: Template

Templates

- Are denoted as class rectangles with a dashed rectangle superimposed on the upper right corner.
- Must not be instantiated.
- Must not be superclasses within generalization relationships.
- May be subclasses of ordinary classes within generalization relationships.
- Must have a name or identifier string that represents the template name.
- May use their formal parameters in their definition.
- Must have a comma-separated parameter list that indicates the formal parameters used to create a class. Formal parameters
  - Must have a name or identifier string that represents the name of the parameter. Names must be unique within the parameter list.
  - May have a colon followed by a type expression that indicates the types of values a parameter may have. If a type expression is not specified, the argument for the parameter must resolve to a classifier.
- Must be bound to actual values using a comma-separated value list in order to be used. Values must match the order, number, and types specified in the parameter list of the template.
- May have bound classes. Bound classes
  - Are bound by referencing their template and specifying a value list.
  - Must not extend the template; that is, they cannot add attributes or operation.
  - May be extended via generalization relationships.
  - May be bound via a dependency association stereotyped using the “bind” keyword and specifying a value list.

  Binding a template involves duplicating the model fragment or template, replacing formal parameters with argument values, and making the result part of the effective model.

- May be applied to packages and any class of modeling elements.

Data Types

Data types are classes that define data values. They are used to model simple values. Data types

- Are represented as strings on diagrams.
- Have unique values that do not have identity. That is, occurrence of the same value cannot be differentiated. For example, the value "1" stored in a variable *X* is no different than the value "1" stored in a variable *Y*.
- Must have operations only.

Objects

Objects (Figure 7-8) are instances of classes. They are used to model particular entities.

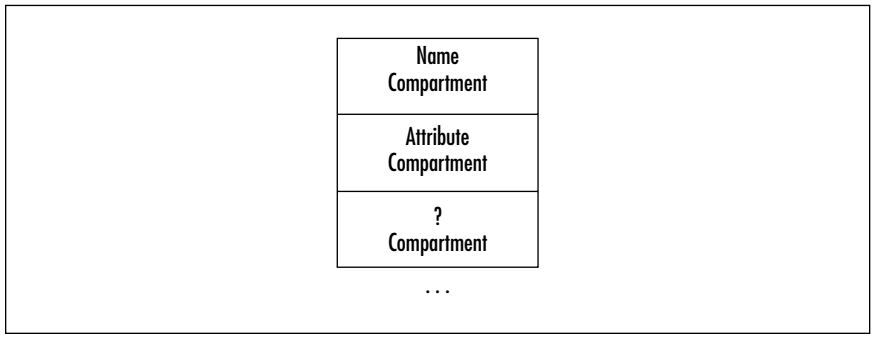


Figure 7-8: Object

Objects

- Are denoted as solid-outline rectangles with compartments.
- Are unique and have identity. That is, every object may be differentiated from every other object, even if the objects have the same attribute values.
- Are defined by their classes.
- Must have a name compartment (Figure 7-9).
- May have a name or identifier string that represents the name of the object. The name must be underlined. When omitted, the colon remains and the object is considered an anonymous object.
- May have a colon followed by a comma-separated class list that indicates the types and class of an object. Objects
  - May have multiple types (roles), which may change dynamically.
  - May have only one implementation class, which may not change.
- May have a comma-separated state list within square brackets that indicates the concurrent states of the object.

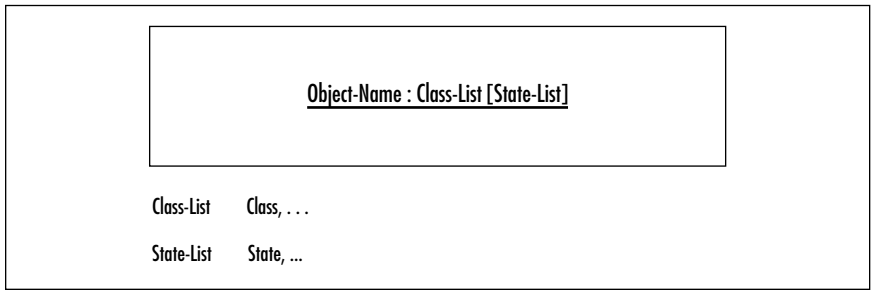


Figure 7-9: Name Compartment of an Object

- May have an attribute compartment named "attributes." This compartment consists of an object's attribute values. Only attributes of interest in the current context (or the current diagram) need be shown.
- May have other compartments as defined by their classes, but may not have an "operations" compartment.
- May be active, that is, instances of active classes.
- May be passive, that is, instances of passive classes.

Attributes

Attributes (Figure 7-10) are an object's attribute values.

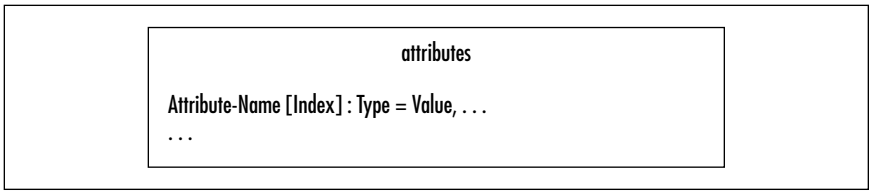


Figure 7-10: Attribute Compartment of an Object

Attributes

- Are denoted as text strings within an attribute compartment.
- May be suppressed if their values are not of interest in a diagram.
- May be shown by composition. An object is composed of its attributes.
- Must have a name or identifier string that represents the name of the attribute.
- May have an index within square brackets that indicates the specific attribute if the multiplicity of the attribute (as defined in its class) is more than one.
- May have a colon followed by a type expression that indicates the types of values an attribute may have. This type expression must match the type expression specified in the object's class.
- May have an equal sign followed by a literal value that represents the value of the attribute for the object. This may be represented as a comma-separated

list of values that indicates the different values that the attribute has had over a period of time.

- May have class scope, as denoted by underlining.

Associations

Associations (Figure 7-11) are descriptions of a set of links with common structural features, behavioral features, relationships, and semantics. They are used to model a set of relationships that relate two or more other concepts or entities where the relationships have common characteristics.

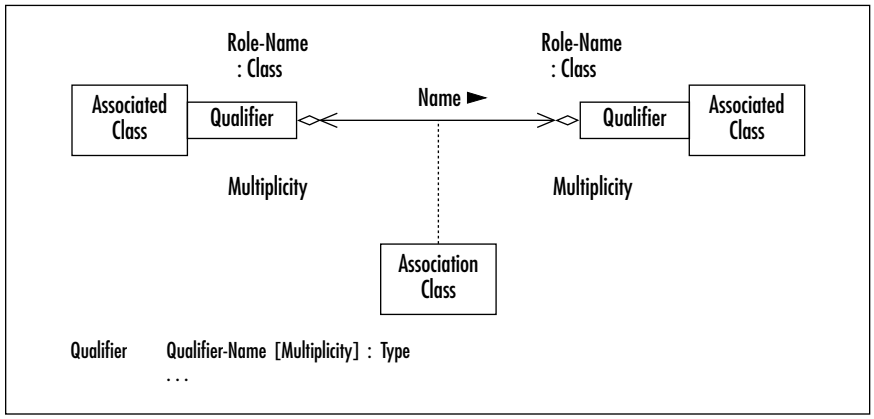


Figure 7-11: Binary Association

Associations

- Are denoted as lines or paths connecting two or more class symbols.
- Have association ends that connect association paths to two or more other classes. Association ends may have properties (Chapter 15) enclosed in curly brackets near the associated class.
- Must relate other classes, not themselves.
- May be reflexive (or recursive). That is, they may associate the same class to itself or associate two objects of the same class.
- May have a name or identifier string that represents the name of the association.
- May have a name-direction arrow. This is a small solid triangle attached to the name, where the point of the triangle indicates the direction in which to read the name of the association. When the small solid triangle is omitted, associations are read from left to right and top to bottom.
- May be association classes. Association classes
  - Are denoted as class symbols attached by dashed lines to associations.
  - Are associations with class properties or classes with association properties.

  - Define a set of characteristics that belong to the relationship. The characteristics are not owned by any of the classes they relate.
  - Are attached to their association paths by a dashed line.
  - May be suppressed, but the association path may not be suppressed.

  Association class names

  - May be placed on association paths and in association class symbols. They must be the same when placed in both places.
  - May be placed on association paths and omitted from association class symbols. This is often used for association classes with attributes and no operations or other associations. This emphasizes an association's "associative nature," or its role in relating other classes.
  - May be placed in association class symbols and omitted from association paths. This is often used for association classes that have operations or other associations. This emphasizes an association's "class nature," or its role as a class.

- May have a role name that indicates the role or specific behavior of an associated class as it participates in the association. If a visibility symbol is specified for the role name, it represents the visibility of the association traversing in the direction toward the role name.
- May have a class name (near the role name) that indicates the behavior expected of an associated class by the other related classes. It specifies an interface or type indicating the behavior required of the associated class in order to enable the association. The associated class must support the interface, and the interface is required for the class to participate in the association. When omitted, the association allows full access to the associated class.

  The use of role names and interface or type names may be used in creating collaborations (Chapter 10). These collaborations include the association as an association role, and include the role name as the class role name and the class name of the interface or type as the class of the class role within the collaboration.
- May have a multiplicity expression that indicates the number of potential instances involved in the association when the other associated classes are fixed, that is, the number of instances of a given class associated with the other classes involved in the association. If the multiplicity is omitted, the model is incomplete.
- May have a navigation arrow that indicates that navigability is supported toward an associated class attached to the arrow. This is indicated by placing an arrow on the association path. One of the following presentation options is possible:
  - All arrows are shown. The absence of an arrow indicates navigability is not supported.
  - No arrows are shown. No inferences can be drawn about navigability. This is similar to any situation where information is suppressed in a diagram.

  - Arrows for navigability in both directions are suppressed, and only arrows for one-way navigability are shown.

- May have an aggregation indicator, represented as a diamond. Diamonds
  - May be hollow to indicate the weak, or shared, form of aggregation. Associated class objects may belong to multiple aggregates and change owners over time. They are not deleted if one of their owners is deleted.
  - May be filled to indicate composition, or the strong form of aggregation. Associated class objects must belong to only one component, and are deleted if the composite is deleted.
  - Must not be attached to both ends of an association.
- May have a qualifier, that is, an attribute or list of attributes of an association whose values partition or divide a set of objects (called the *targets*) associated with a given object (called the *source*). The qualifier is attached to the source class of an association and determines how objects on the target side of an association are partitioned and identified. Qualifiers
  - Are denoted as small rectangles attached to the end of an association path between the final path segment and the symbol of the class to which they connect.
  - Must have a name or identifier string that represents the name of the attributes of the association.
  - May have a multiplicity expression within square brackets that indicates the number of attributes per instance of the association. The default multiplicity is one.
  - May have a colon followed by a type expression that indicates the types of values an attribute may have.
  - Reduce the effective multiplicity of an association. That is, if the multiplicity of a class involved in an association is more than one, a qualifier specifies a set of values or attributes that will return a subset of the objects of the class.

  An object of a source class together with a qualifier value uniquely selects a partition of the target class. Every object of the target class may be a constituent of exactly one partition.

  Multiplicity attached to the target class in an association denotes the possible cardinalities of the set of target objects selected by pairing a source object and a qualifier value. The following multiplicity values are common:
  - "0..1" indicates that a unique qualifier value may or may not select a target object; that is, not every possible qualifier value has to select a target object.
  - "1" indicates that a unique qualifier value selects a unique target object; that is, every possible qualifier value has to select a target object.
  - "*" indicates that a unique qualifier value selects a set of target objects; that is, every possible qualifier value is an index or key that partitions the

target objects into subsets. The qualifier is similar to a key that may be used to index a subset of the target objects.

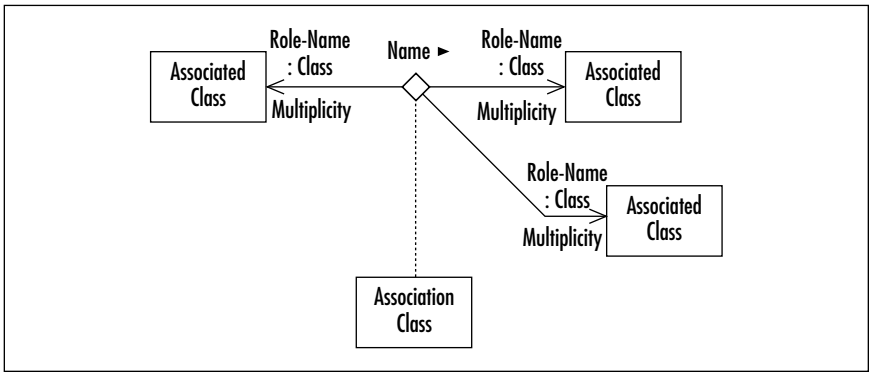


Figure 7-12: N*-ary Association*

- Among three or more classes, are shown as a diamond with paths from each corner or side (Figure 7-12). Such an association
  - Must not involve aggregation or qualifiers.
  - May have a single class appear more than once or on multiple paths.

Generalizations

Generalizations (Figure 7-13) are associations between more general elements and more specific elements in which the more specific elements are fully consistent with the more general elements and inherit and may add information to the characteristics provided by the more general elements. Two elements are fully consistent if they are the same type of elements. For example, two implementation classes are consistent since they both are classes of implementation constructs, and two actor classes are consistent since they both model actors. Generalizations are used to model taxonomic relationships between concepts.

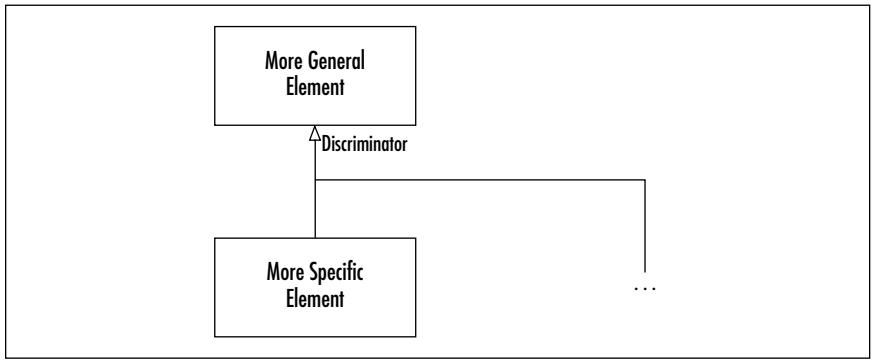


Figure 7-13: Generalization

Generalizations

- Are denoted as solid paths with a large hollow triangle pointing at the more general element.
- More specific elements
  - Specialize more general elements.
  - Receive all the characteristics (attributes, operations, methods, and associations) of the more general elements via the mechanism of inheritance.
  - May add their own characteristics (attributes, operations, methods, and associations).
  - May override inherited methods.
  - May be substituted for their more general elements.
- Must not be circular; that is, an element cannot have a generalization relationship to itself.
- May have discriminators. Discriminators
  - Specify the basis for the specialization. They define a named dimension of specialization through which more general elements are partitioned into groups of more specific elements.
  - Must be role names or attributes of more general elements whose values enumerate the partitions. The actual type of the attribute or role name that is used as a discriminator is known as the *powertype*.
- May have an ellipsis (...) in place of a more specific element to indicate that additional more specific elements exist in the model but are not shown on the diagram.

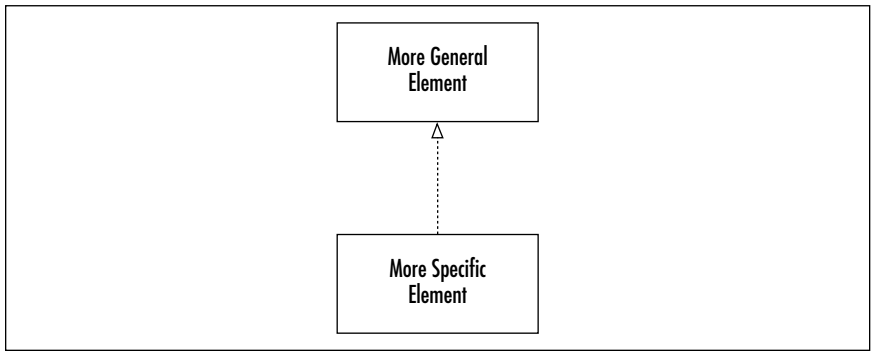


Figure 7-14: Realizes Relationship (Generalization)

- May be realizes relationships (Figure 7-14).
  - If the more general element is an implementation class, it implies inheritance of operations but not of structure (attributes and associations).
  - If the more general element is an interface, the more specific element supports the operations of the interface.

Dependencies

Dependencies (Figure 7-15) are associations between a collection of dependent client elements and another collection of independent supplier elements. The client elements require knowledge of and the presence of the supplier elements. Dependencies are used to model dependencies among model elements where changes in the independent supplier elements may require changes to the dependent client elements.

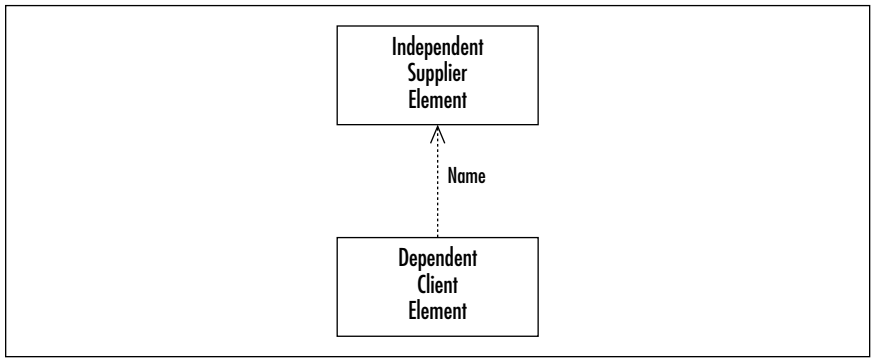


Figure 7-15: Dependency

Dependencies

- Are denoted as dashed arrows with the arrowhead pointing at the independent element.
- Relate elements of a model.
- Don't require instances to associate; rather, they exist among model elements.
- Have elements that exist at the same level of abstraction or detail. That is, the dependent client element and the independent supplier element are described at the same level of detail.
- May be stereotyped using the "refinement" keyword to depict different semantic levels of abstraction. The refinement may be described in a note attached to the dependency.
- May serve as a container for a group of dependencies from parts of a client element to parts of a supplier element.
- May be stereotyped using a keyword to express specific dependencies.
- May have a name.
- May have the arrowhead suppressed when used to attach notes or constraint strings to model elements. The direction is clear: the note or constraint string is the source of the arrow and the model element is the target of the arrow.

Links

Links (Figure 7-16) are instances of associations. They are used to model relationships that relate two or more other instances of concepts or entities.

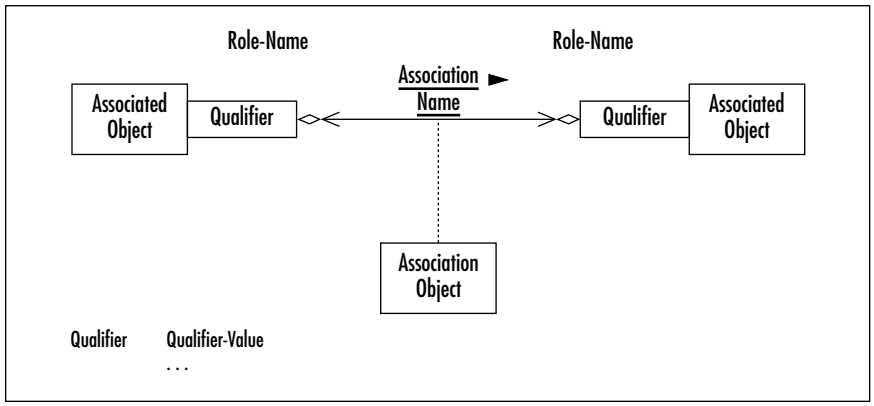


Figure 7-16: Binary Link

Links

- Are denoted as paths between objects.
- Have link ends that connect link paths to two or more other objects. Link ends may have properties (Chapter 15) enclosed in curly brackets near the objects they link.
- Are unique and have identity. That is, every link may be differentiated from every other link, even if the links are between the same objects.
- Are defined by their associations.
- Establish relationships between instances and are similar to objects. They are lists of references to related objects.
- May be reflexive. That is, they can associate two objects of the same class.
- Must not have a name or identifier string. They take their identity from the objects they relate.
- May have their association name rendered. The name must be underlined.
- May have a name-direction arrow. This must match the link's association name-direction arrow.
- May be instances or objects of association classes.
- May have a role name that indicates the role of an associated class. This must match the link's association role name.
- Must not have a class name following the role name, like associations have. Instead, the link's association indicates the behavior of the associated object.
- Must not have a multiplicity expression because they relate instances.
- May have a navigation arrow that indicates navigability is supported toward the associated objects attached to the arrow. This must match the link's association navigation arrow.
- May have an aggregation or composition indicator.

- May have a qualifier value or list of qualifier values, as expressed by the link's association.

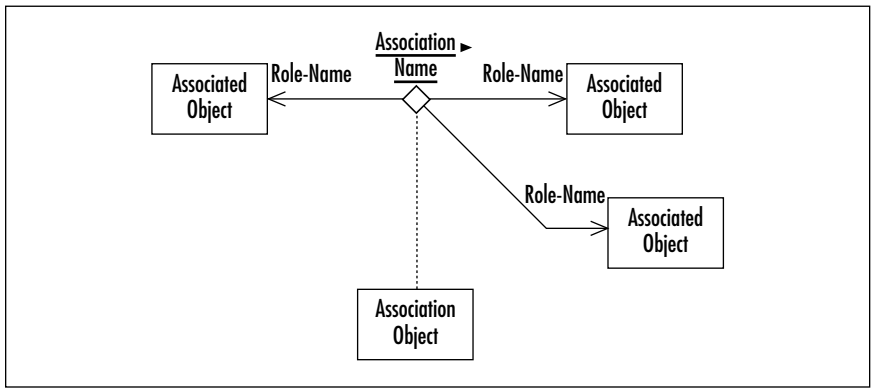


Figure 7-17: N*-ary Link*

- Among three or more classes are shown as a diamond with paths from each corner or side (Figure 7-17). Such an association
  - Must not involve aggregation or qualifiers.
  - May have a single object appear more than once or on multiple paths.

Compositions

Compositions (Figure 7-18) involve composite classes or classes involved in composite aggregations. Compositions are used to model parts that exist or live and die with their associated owner.

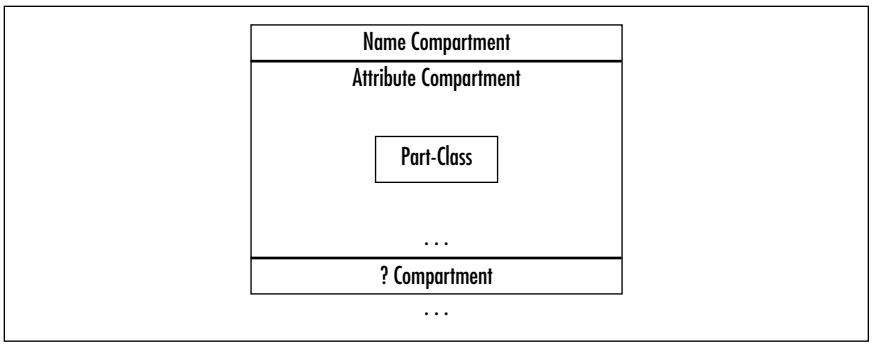


Figure 7-18: Composite Class

Compositions

- May be depicted in generic form using classes and associations.
- May be depicted in instance form using objects and links.
- Must have composite classes. Composite classes

  - Must have a name compartment.
  - Must have an attribute compartment depicting the parts of the composite. Attributes are effectively composition relationships between a class and the classes of its attributes.

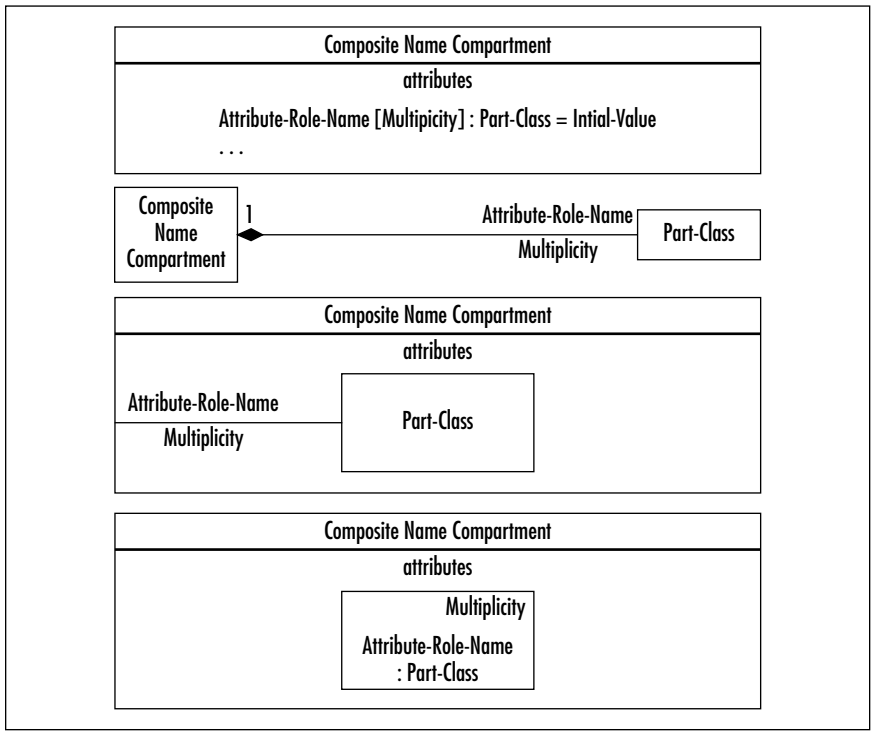


Figure 7-19: Composite Aggregation Association

- May have composite aggregation associations (Figure 7-19). Composite aggregation associations
  - Must only link objects belonging to the same container object or composite.
  - May be shown as a line ending in a filled diamond (as a composition association) or by using one of two different graphically nested forms.
  - Have parts that may have a multiplicity; it defaults to many.
  - Have Role names of the parts, which correspond to attributes of the composite.
- May have other associations.
  - If an association is drawn entirely within the border of a composite, the association is considered to be part of the composite; any objects on such a link must be from the same composite.

– If an association is drawn such that its path crosses the border of the composite, the association is not considered to be part of the composite; any objects on such a link may be from the same or a different composite.