A UML Class Diagram Tutorial

The UML Class diagram is a graphical notation used to construct and visualize object oriented systems.

A UML class diagram is made up of:

- A set of classes and
- A set of relationships between classes

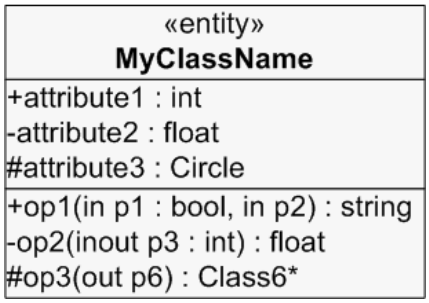
What is a class?

A class in an object oriented system provides a crisp abstraction of a well defined set of responsibilities.

A class consists of three parts: (Refer to the figure on the right)

- **Class Name:**
  - The name of the class appears in the first partition.
- **Class Attributes**:
  - Attributes are shown in the second partition.
  - The attribute type is shown after the colon.
  - Attributes map onto member variables (data members) in code.
- **Class Operations (Methods):**
  - Operations are shown in the third partition. They are services the class provides.
  - The return type of a method is shown after the colon at the end of the method signature.
  - The return type of method parameters are shown after the colon following the parameter name.
  - Operations map onto class methods in code

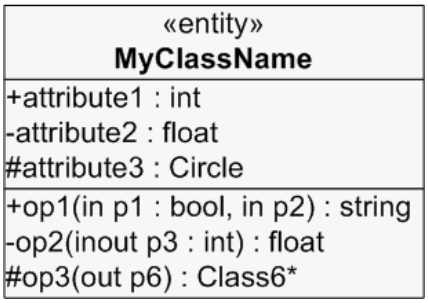
The graphical representation of a Class



We can observe the following for *MyClassName*

- *MyClassName* has 3 attributes and 3 operations
- Parameter *p3* of *op2* is of type *int*
- op2 returns a *float*
- op3 returns a pointer (denoted by a *) to *Class6*

Visibility and Access for attributes and operations of a class



The +, - and # symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.

+ denotes **public** attributes or operations
- denotes **private** attributes or operations
# denotes **protected** attributes or operations

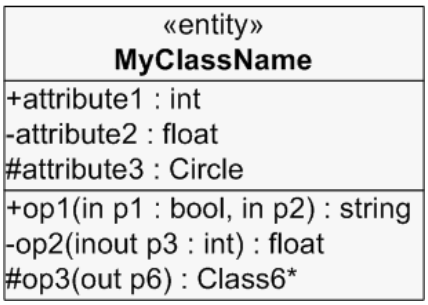
We can observe that

- *attribute1* and *op1* of *MyClassName* are public
- *attribute3* and *op3* are protected.
- *attribute2* and *op2* are private.

Access for each of these visibility types is shown below for members of different classes.

Access	**public (+)**	**private (-)**	**protected (#)**
Members of the same class	yes	yes	yes
Members of derived classes	yes	no	yes
Members of any other class	yes	no	no

Operation (Method) Parameter Directionality



Each parameter in an operation (method) may be denoted as *in*, *out* or *inout* which specifies its direction with respect to the caller. This directionality is shown before the parameter name.

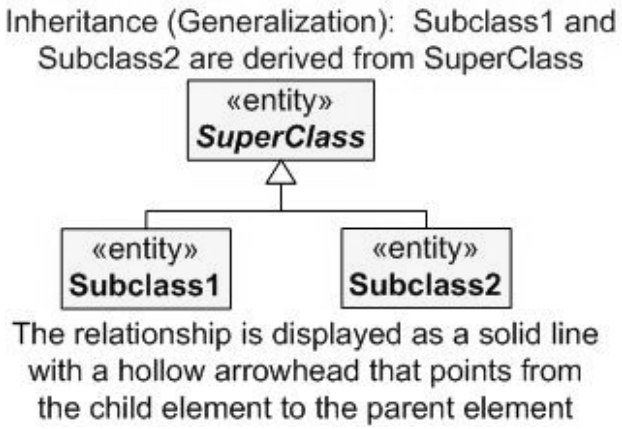
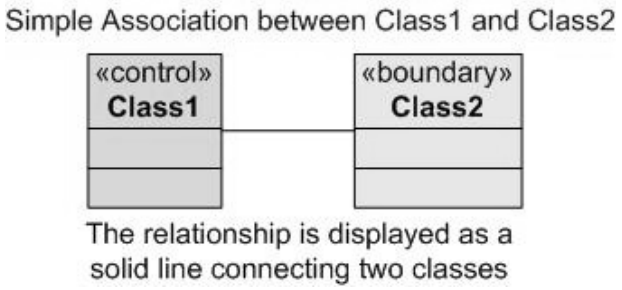
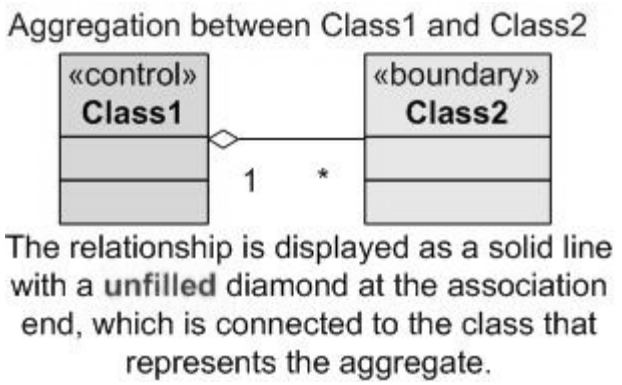
These are briefly explained below:

Parameter direction	Description
in	states that *p1* and *p2* are passed to *op1* by the caller. They are both *in* parameters.
inout	states that *p3* is passed to *op2* by the caller and is then possibly modified by *op2* and is passed back out. *p3* is an *inout* parameter.
out	states that *p6* is not set by the caller but is modified by *op3* and is passed back out. *p6* is an *out* parameter.

Relationships between classes

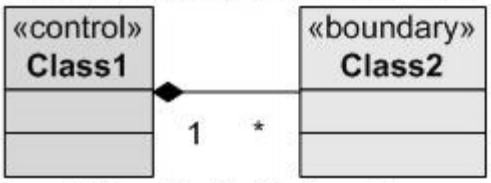
A class may be involved in one or more relationships with other classes.

A relationship can be one of the following types: (Refer to the figure on the right for the graphical representation of relationships)

Relationship Type	Graphical Representation
Inheritance (or Generalization): • Represents an "**is-a"** relationship. • An abstract class name is shown in italics. • *SubClass1* and *SubClass2* are specializations of *SuperClass*.	
Association • **Simple association**: o A structural link between two peer classes. o There is an association between *Class1* and *Class2*	
• **Aggregation**: A special type of association. It represents a **"part-of"** relationship. o *Class2* is part of *Class1*. o Many instances (denoted by the *) of *Class2* can be associated with *Class1*. o Objects of *Class1* and *Class2* have separate lifetimes.	

- **Composition**: A special type of aggregation where parts are destroyed when the whole is destroyed.
    - Objects of *Class2* live and die with *Class1*.
    - *Class2* cannot stand by itself.

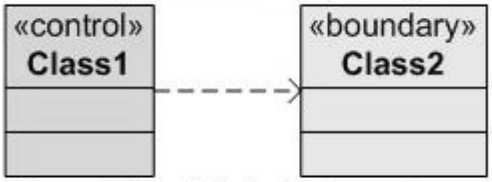
Composition between Class1 and Class2



The relationship is displayed as a solid line with a filled diamond at the association end, which is connected to the class that represents the whole, or composite.

- **Dependency:**
    - Exists between two classes if changes to the definition of one may cause changes to the other (but not the other way around).
    - *Class1* depends on *Class2*

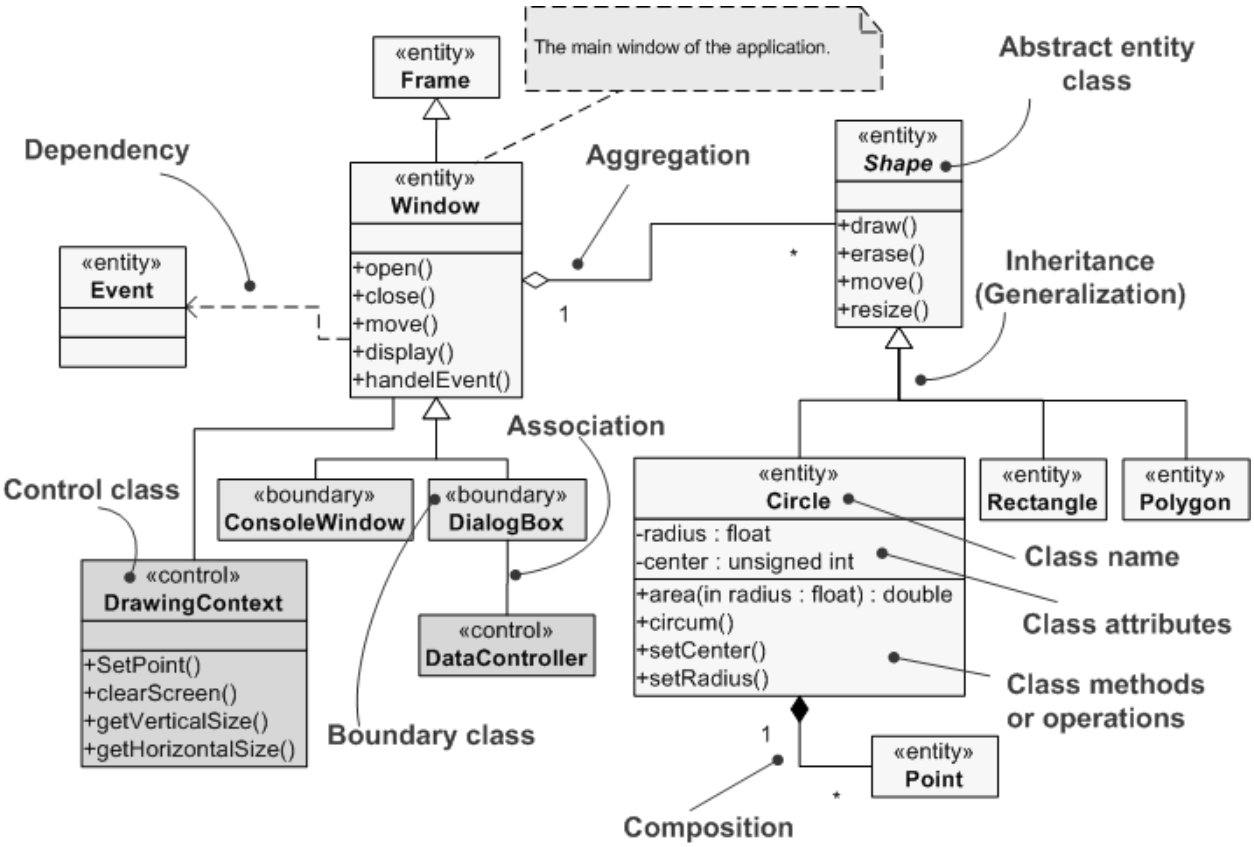
Dependency between Class1 and Class2



The relationship is displayed as a dashed line with an open arrow.

A class diagram may also have notes attached to classes or relationships. Notes are shown in grey.

An Example

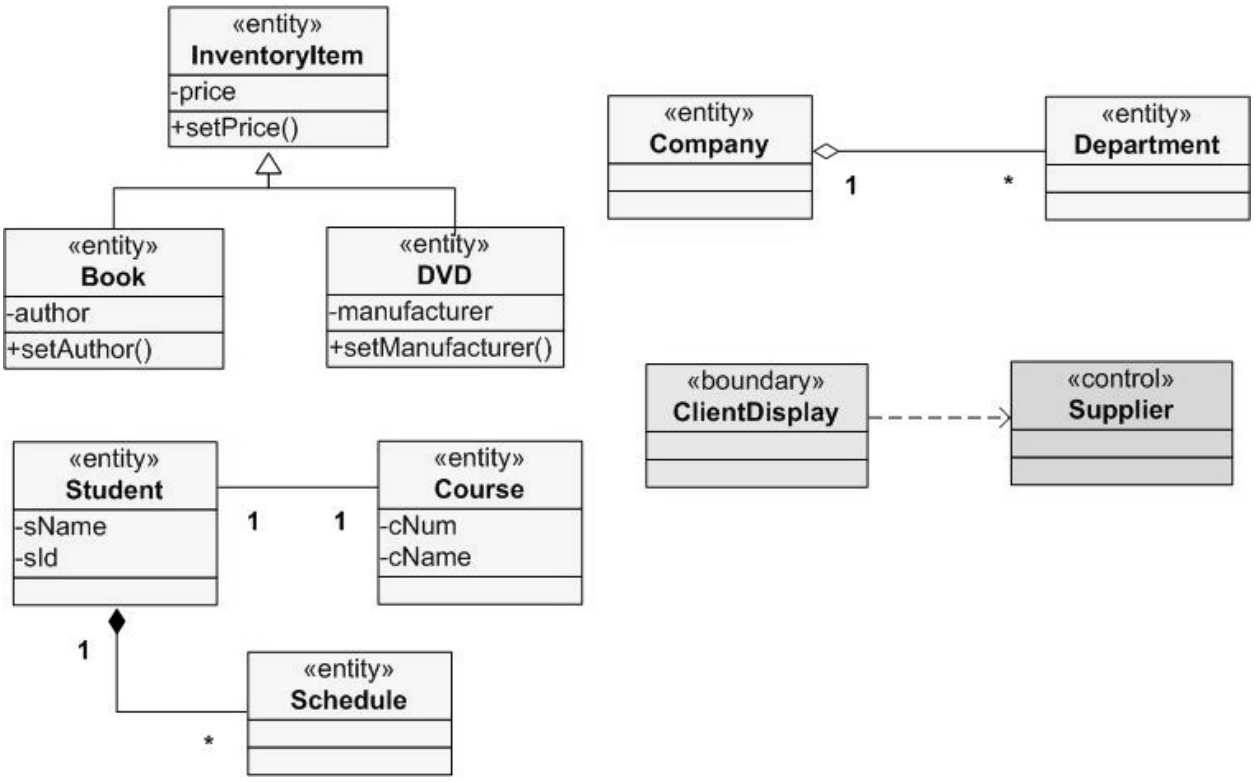


The following can be observed from the above diagram:

1. *Shape* is an abstract class. It is shown in Italics.
2. *Shape* is a superclass. *Circle*, *Rectangle* and *Polygon* are derived from *Shape*. In other words, a *Circle* ***is-a*** *Shape*. This is a generalization / inheritance relationship.
3. There is an association between *DialogBox* and *DataController*.
4. *Shape* is ***part-of*** *Window*. This is an aggregation relationship. *Shape* can exist without *Window*.
5. *Point* is ***part-of*** *Circle*. This is a composition relationship. *Point* cannot exist without a *Circle*.
6. *Window* is dependent on *Event*. However, *Event* is not dependent on *Window*.
7. The attributes of *Circle* are *radius* and *center*. This is an entity class.
8. The method names of *Circle* are *area(), circum(), setCenter()* and *setRadius()*.
9. The parameter *radius* in *Circle* is an *in* parameter of type *float*.
10. The method *area*() of class *Circle* returns a value of type *double*.
11. The attributes and method names of *Rectangle* are hidden. Some other classes in the diagram also have their attributes and method names hidden.

Exercise 1

Consider the following UML Class Diagram snippets. Choose the type of relationship between classes given below.



1. InventoryItem and Book
    a. Dependency
    b. Generalization(Inheritance)
    c. Association
    d. Composition
    e. Aggregation
    f. No relationship exists
2. Book and DVD
    a. Dependency
    b. Generalization(Inheritance)
    c. Association
    d. Composition
    e. Aggregation
    f. No relationship exists
3. Student and Course
    a. Dependency
    b. Generalization(Inheritance)
    c. Association
    d. Composition
    e. Aggregation
    f. No relationship exists

4. Student and Schedule
    a. Dependency
    b. Generalization(Inheritance)
    c. Association
    d. Composition
    e. Aggregation
    f. No relationship exists
5. ClientDisplay and Supplier
    a. Dependency
    b. Generalization(Inheritance)
    c. Association
    d. Composition
    e. Aggregation
    f. No relationship exists
6. Department and Company
    a. Dependency
    b. Generalization(Inheritance)
    c. Association
    d. Composition
    e. Aggregation
    f. No relationship exists
7. InventoryItem and DVD
    a. Dependency
    b. Generalization(Inheritance)
    c. Association
    d. Composition
    e. Aggregation
    f. No relationship exists

Exercise 2: True or False?	True/False
You can use a dependency relationship to represent precedence, where one model element must precede another.	
An aggregation is a special type of association in which objects are assembled or configured together to create a more complex object.	
An association represents a non-structural relationship that connects two classes.	
A composition association relationship connects a Student class with a Schedule class, which means that if you remove the student, the schedule is also removed.	
Because child classes in generalizations inherit the attributes, operations, and relationships of the parent, you must only define for the child the attributes, operations, or relationships that are distinct from the parent.	