CLASS DIAGRAM

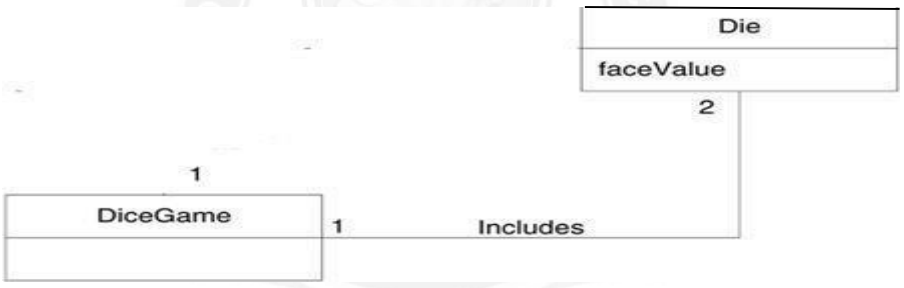
- The UML includes class diagrams to illustrate classes, interfaces, and their associations. They are used for static object modeling.
- Used for static object modeling . It is used to depict the classes within a model.
- It describes responsibilities of the system , it is used in forward and reverse engineering
- Keywords used along with class name are { abstract , interface, actor}

Definition: Design Class Diagram

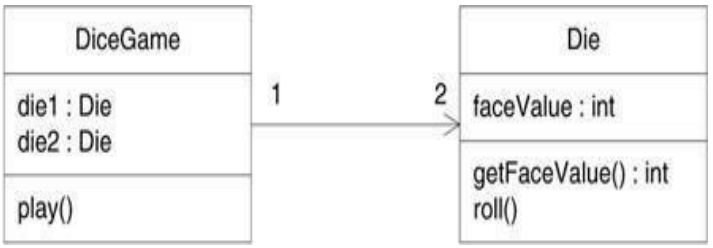
The class diagram can be used to visualize a domain model. we also need a unique term to clarify when the class diagram is used in a software or design perspective. A common modeling term for this purpose is design class diagram (DCD).

UML class diagrams in two perspectives

Domain Model

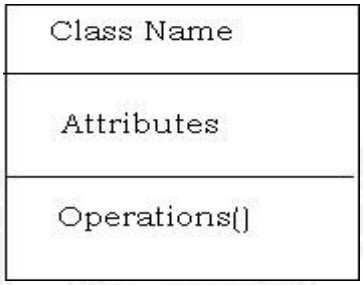


Design Model



Class Diagram Representation

Class is represented as rectangular box showing classname, attributes , operations.



The main elements of class are

1. ***Attributes***
2. ***Operations & Methods***
3. ***Relationship between classes***

1. Attributes (refer pg no 26)

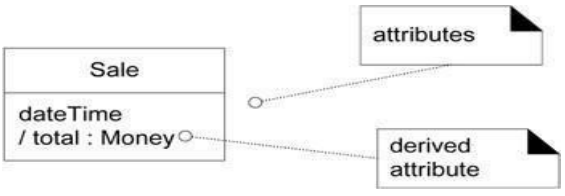
An **attribute** is a logical data value of an object. Attributes of a classifier also called structural properties in the UML. The full format of the attribute text notation is:

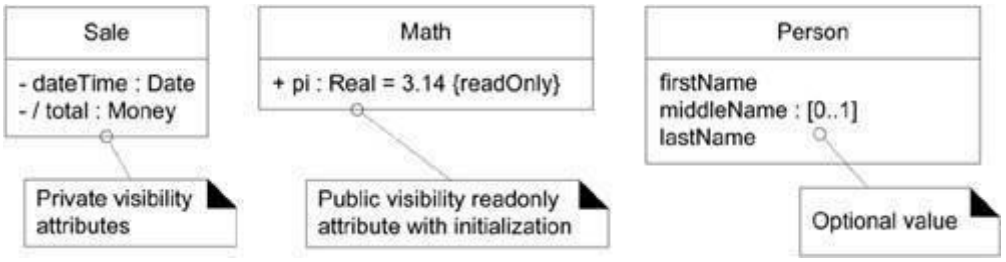
Syntax:

visibility name : type multiplicity = default {property-string}

visibility marks include + (public), - (private)

Examples:

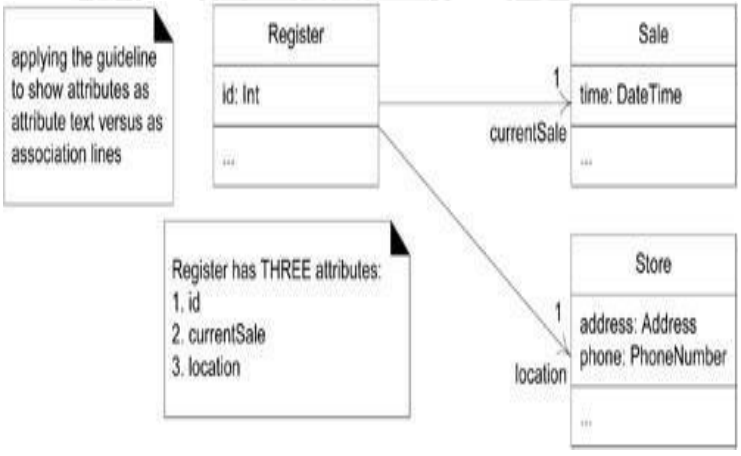




different formats

-classOrStaticAttribute:int
+ publicAttribute : string
- privateAttribute
assumedPrivateAttribute
isIntializedAttribute : Bool = true
aCollection:VeggieBurger [*]
attributeMayLegallyBeNull : String[0..1]
finalConstantattribute : int = 5 { readonly}
/derivedAttribute

Guideline: Use the attribute text notation for data type objects and the association line notation for others.



2. Operations and Methods

Operations : One of the compartments of the UML class box shows the signatures of operations . Assume the version that includes a return type. Operations are usually assumed public if no visibility is shown. both expressions are possible

An operation is not a method. A UML **operation** is a declaration, with a name, parameters, return type, exceptions list, and possibly a set of constraints of pre-and post-conditions. methods are implementations.

Syntax :

visibility name (parameter-list) : return-type {property-string}

Example::

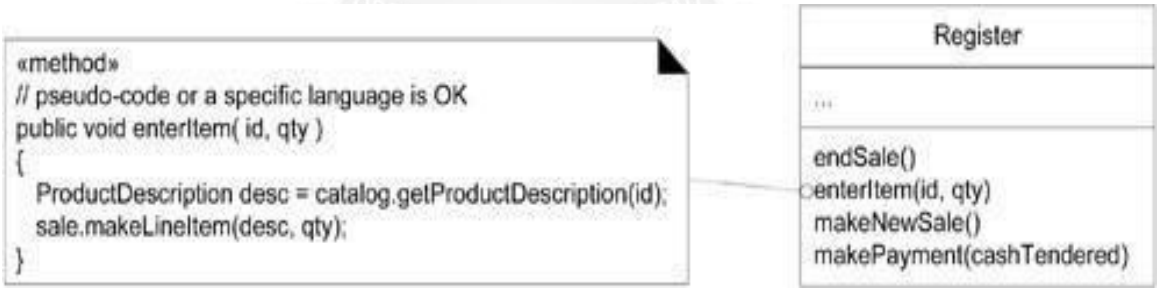
UML REPRESENTATION	+ getPlayer(name : String) : Player {exception IOException}
JAVA CODING	public Player getPlayer(String name) throws IOException

+ classOrStaticMethod()
+ publicMethod()
assumedPublicMethod()
- privateMethod()
protectedMethod()
~ packageVisibleMethod()
«constructor» SuperclassFoo(Long)
methodWithParms(parm1 : String, parm2 : Float)
methodReturnsSomething() : VeggieBurger
methodThrowsException() {exception IOException}
abstractMethod()
abstractMethod2() { abstract } // alternate
finalMethod() { leaf } // no override in subclass
synchronizedMethod() { guarded }

To Show Methods in Class Diagrams:

A UML **method** is the implementation of an operation. A method may be illustrated several ways, including:

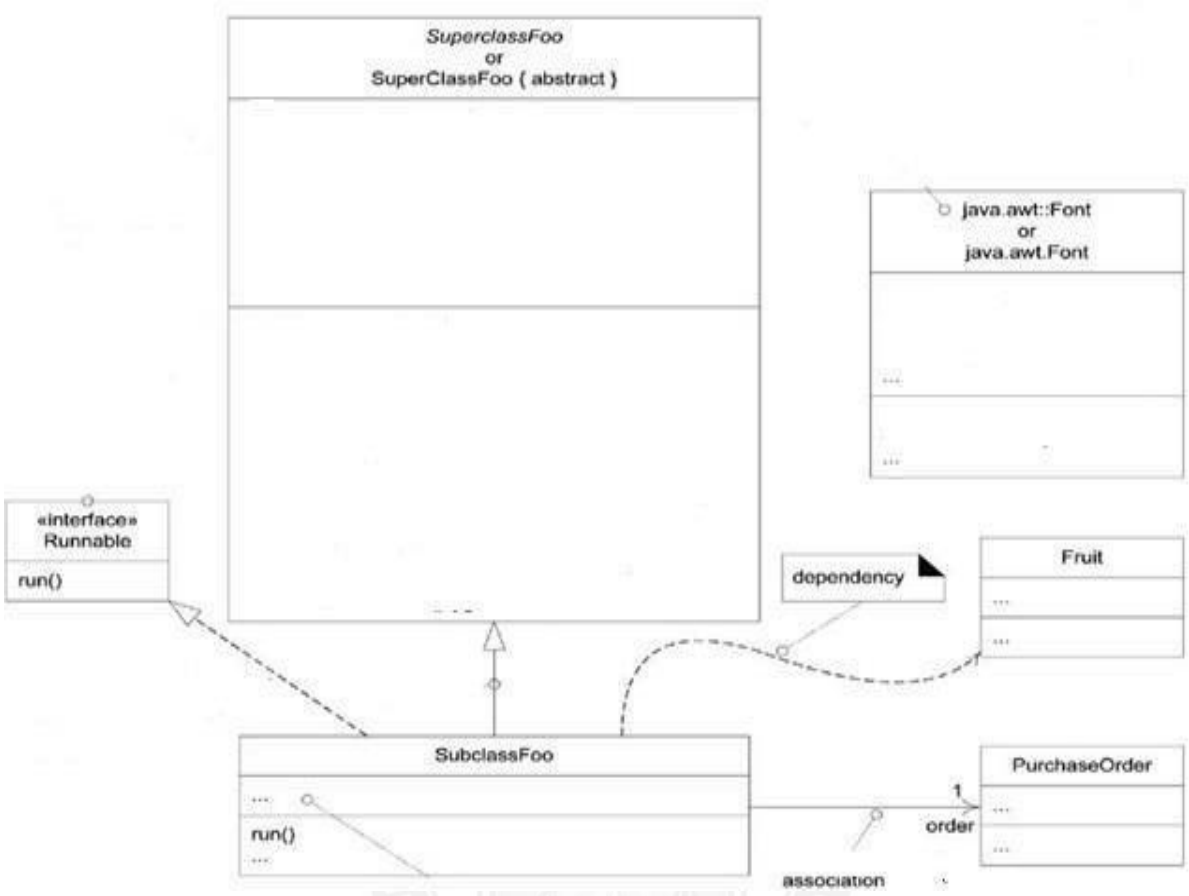
- in interaction diagrams, by the details and sequence of messages
- in class diagrams, with a UML note symbol stereotyped with «method»



3. Relationship between classes

There are different relationship exists between classes. They are

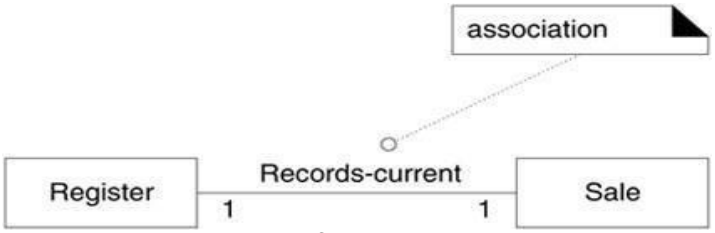
A. Association
B. Generalization & specialization
C. Composition and aggregation
D. Dependency
E. Interface realization



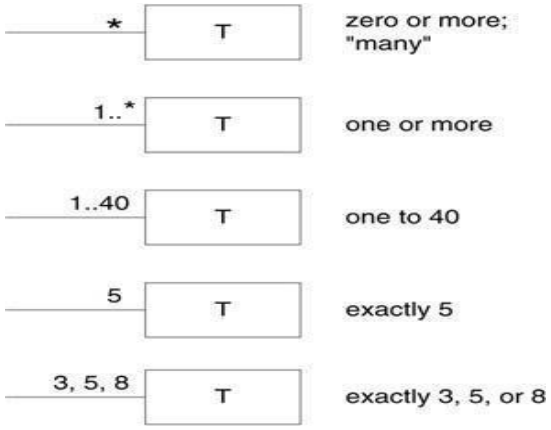
A) Association

An **association** is a relationship between classes. The semantic relationship between two or more classifiers that involve connections among their instances.

Example



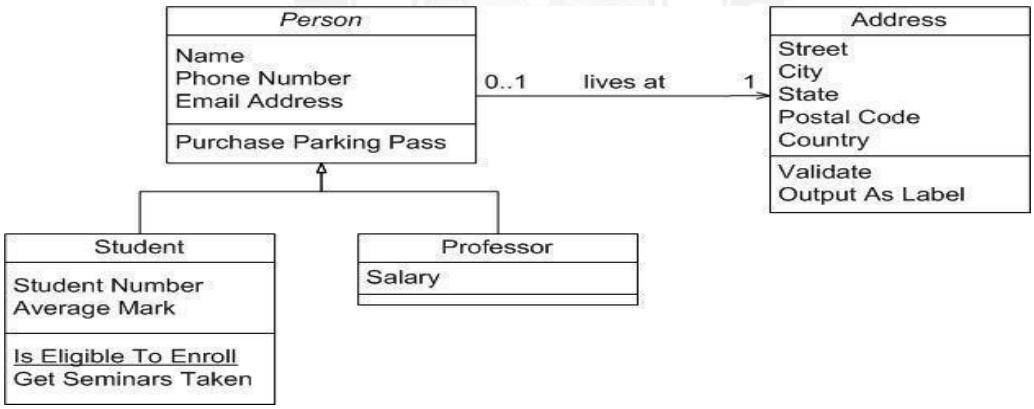
For example, a single instance of a class can be associated with "many" (zero or more, indicated by the *) Item instances.



B) Generalization & Specialization

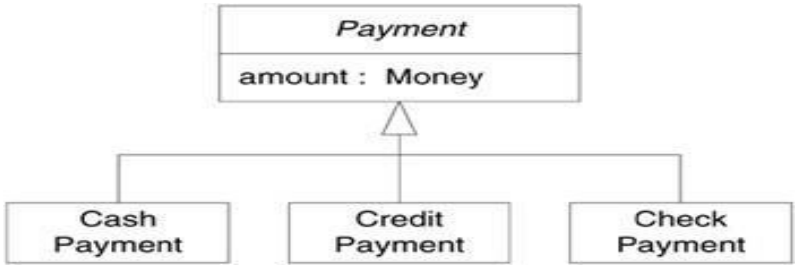
Generalization is the activity of identifying commonality among concepts and defining superclass (general concept) and subclass (specialized concept) relationships.

Ex1:



In the above example person is the generalized class and specialized classes are student and professor

Ex2:

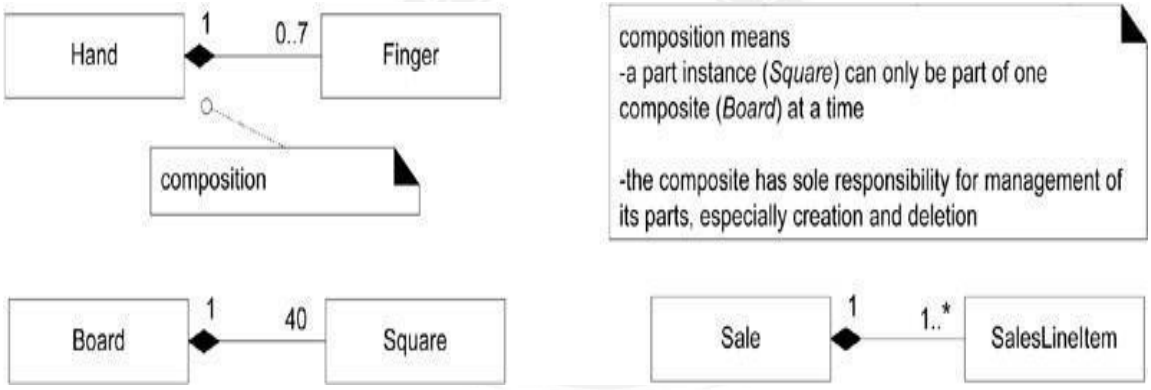


In the above example payment is the generalized class and specialized classes are cash payment , credit payment and check payment .

C) Composition and Aggregation

Composition, also known as composite aggregation, is a strong kind of whole-part aggregation and is useful to show in some models. A composition relationship implies that

1) an instance of the part (such as a Square) belongs to only one composite instance (such as one Board) at a time,
2) the part must always belong to a composite (no free-floating Fingers)
3) the composite is responsible for the creation and deletion of its parts either by itself creating/deleting the parts, or by collaborating with other objects.



Aggregation is a vague kind of association in the UML that loosely suggests whole-part relationships .Aggregation implies a relationship where the child can exist independently of the parent. Example: Class (parent) and Student (child). Delete the Class and the Students still exist.

For example, a Department class can have an aggregation relationship with a Company class, which indicates that the department is part of the company. Aggregations are closely related to compositions.

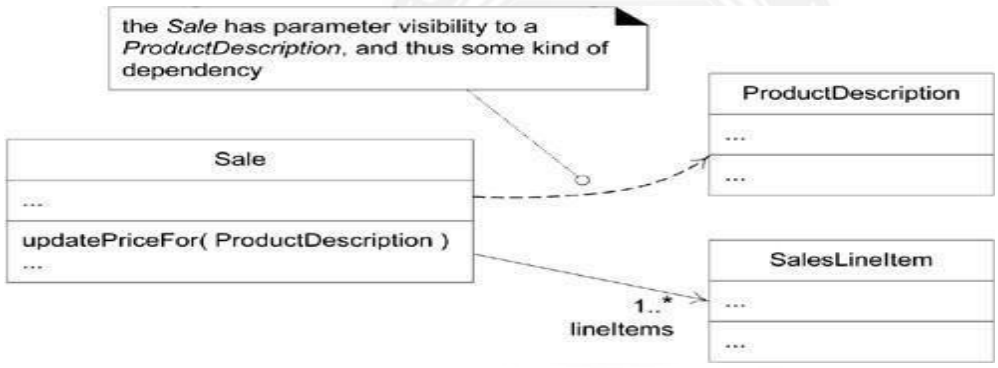
D) Dependency

A general dependency relationship indicates that a client element (of any kind, including classes, packages, use cases, and so on) has knowledge of another supplier element and that a change in the supplier could affect the client.

Dependency can be viewed as another version of **coupling**, a traditional term in software development when an element is coupled to or depends on another.

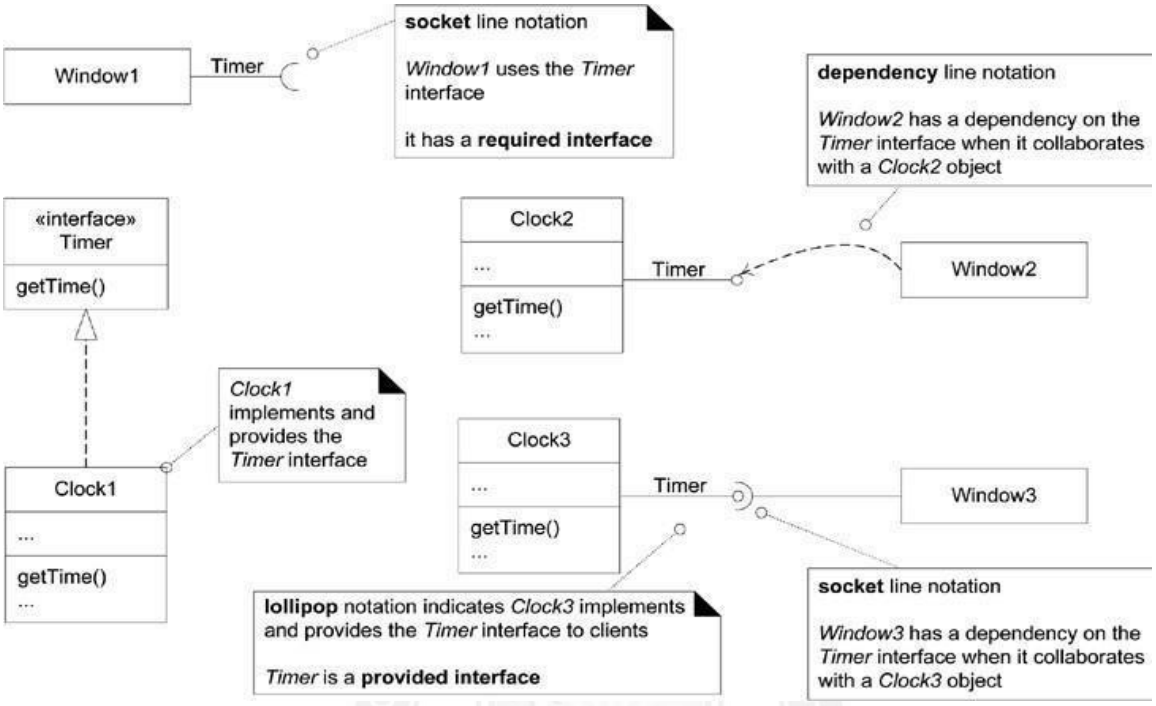
There are many kinds of dependency

- having an attribute of the supplier type
- sending a message to a supplier; the visibility to the supplier could be:
  - an attribute, a parameter variable, a local variable, a global variable, or class visibility (invoking static or class methods)
- receiving a parameter of the supplier type
- the supplier is a superclass or interface



E) Interface realization

The UML provides several ways to show **interface** implementation, providing an interface to clients, and interface dependency (a required interface). In the UML, interface implementation is formally called interface realization

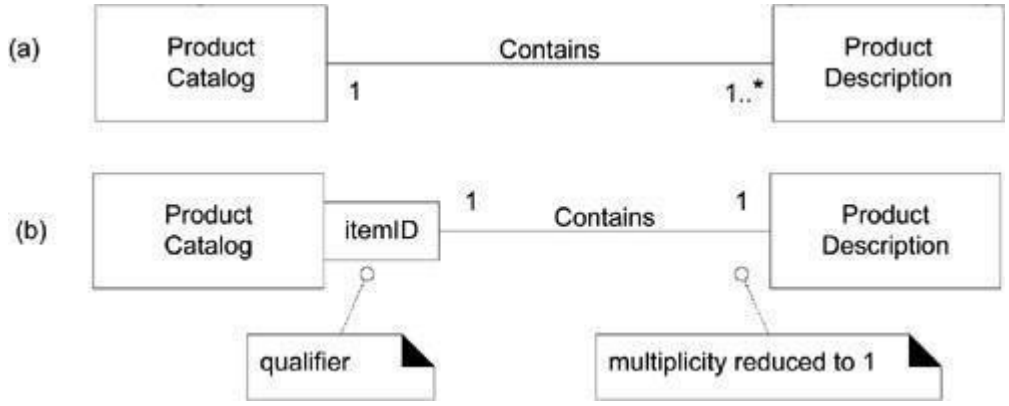


In the above example , Clock is the server program implementing Timer interface giving Timer as the provided interface, window is the client program with Timer

as required interface. The Timer interface contains the services provided by the server object.

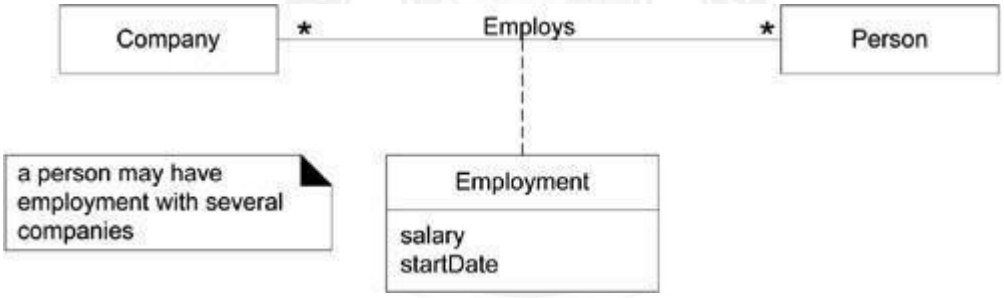
Qualified Association

A **qualified association** has a qualifier that is used to select an object (or objects) from a larger set of related objects, based upon the qualifier key



Association Class

An association class allows you treat an association itself as a class, and model it with attributes, operations, and other features. For example, if a Company employs many Persons, modeled with an Employs association, you can model the association itself as the Employment class, with attributes such as startDate.



WHEN TO USE CLASS DIAGRAMS

Class diagram is a static diagram and it is used to model the static view of a system. The static view describes the vocabulary of the system.

Class diagram is also considered as the foundation for component and deployment diagrams. Class diagrams are not only used to visualize the static view of the system but they are also used to construct the executable code for forward and reverse engineering of any system.

Class diagram clearly shows the mapping with object-oriented languages such as Java, C++, etc. From practical experience, class diagram is generally used for construction purpose.

Class Diagrams are used for −

- Describing the static view of the system.
- Showing the collaboration among the elements of the static view.
- Describing the functionalities performed by the system.
- Construction of software applications using object oriented languages.
- Forward and reverse engineering.

Ex: Order Processing System