Class Diagram:

UML diagrams are divided into two types, i.e. structural and behavioral diagrams. Class diagram also falls in the category of the structural UML diagrams. It shows the structure of the system by use of classes, their attributes, methods and relationships among objects of the system. In modeling of the software product class diagram is the main building block. Further, class diagram can be used in detailed modeling by translating the models into coding.

For better understanding of class diagrams, consider the classes as town or cities and relationship between classes are like routes between these places. That route will navigate from class to class, traversing the relationship between those classes. Moreover, if you have to represent the any particular functionality / functional requirement through interaction or collaboration between classes then consider a specific route across that map.

Class diagram can be used in different phases of the software development life cycle depending on the level of details. In the analysis phase, classes represent a problem domain and of primary interest. While in design phase, classes represent the solution model by structure of classes and relationships between them. However, in development phase, implementable solution will be added, and relationship structure between the classes can be revised to reflect implementation consideration. So we can say that class diagrams are used throughout the software life cycle.

Purpose:

The purpose of the class diagram is to model the static view of an application. The class diagrams are the only diagrams which can be directly mapped with object oriented languages and thus widely used at the time of construction. Object Management Group describes the static structure diagram, but the notation specification points out it as a class diagram as it is shorter and well established. Class diagram shows behavioral and data management responsibilities of each class and thus how this responsibility passes across the model.

A Class diagram is also a UML diagram like use case or activity diagram, but it is a bit different as it gives coding hint instead of showing the sequential flow of the application. So from the code development or programming point of view class diagram is most important. The purposes of class diagram are as follows:

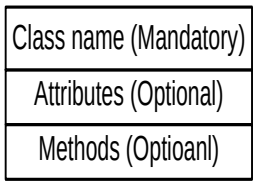
- It is used to analyze and to design the static view of an application.
- Class diagram can describe responsibilities of a system.
- Document the classes that organize a system or subsystems.
- Base for component and deployment diagrams.
- It is used to describe the association, generalization, composition and aggregation relationship among class.

- It is also used to show the features of the classes, i.e. the attribute and methods/operations of each class.
- It can show interface supported by a given class.
- It can be used to show the individual object instance within the class structure.
- It can also document that how the classes of a particular system interact with the existing class libraries.
- Class diagram can be used throughout the life cycle of software product. Such as form the specification of the classes in the problem defining stage to implementation model for a proposed system, to show the structure of that system.
- Class diagram can also be used in forward and reverse engineering process.

Representation of Class Diagram:

In the diagram, classes are represented with boxes which contain three parts:

- The top part contains the name of the class. It is printed in bold and centered, and the first letter is capitalized. The class name is mandatory.
- The middle part contains the attributes of the class which is optional. They are left-aligned and the first letter is lowercase.
- The bottom part contains the methods the class can execute which is also optional. They are also left-aligned and the first letter is lowercase.



In the design of a system, a number of classes are identified and grouped together in a class diagram which helps to determine the static relations between those objects. With detailed modelling, the classes of the conceptual design are often split into a number of subclasses.

In order to further describe the behavior of systems, these class diagrams can be complemented by a state diagram or UML state machine.

Components of class diagram:

A *class* is a description of a set of objects that share the same attributes, operations, relationships, and semantics. The components of a class are name, attribute and methods.

Name: The name of the class is the only required tag in the graphical representation of a class. It always appears in the topmost section.

Attribute: An *attribute* is a named property of a class that describes the object being modeled. In the class diagram, attributes appear in the second compartment just below the name-compartment. Attributes are usually listed in the form:

attributeName : Type

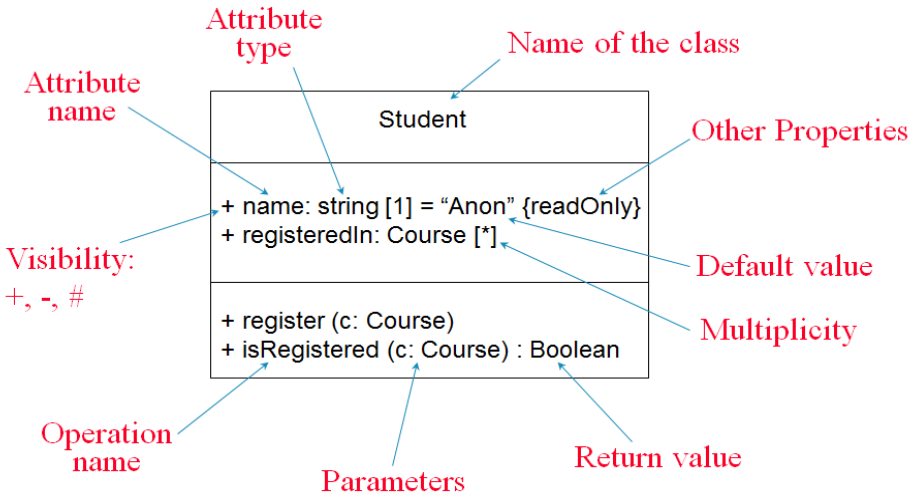
A *derived* attribute is one that can be assumed from other attributes, but doesn't actually exist. For example, a Person's age can be computed from his birth date. A derived attribute is designated by a preceding '/' as in:

/ age : Date

Attributes can be:

- + public
- # protected
- - private
- / derived

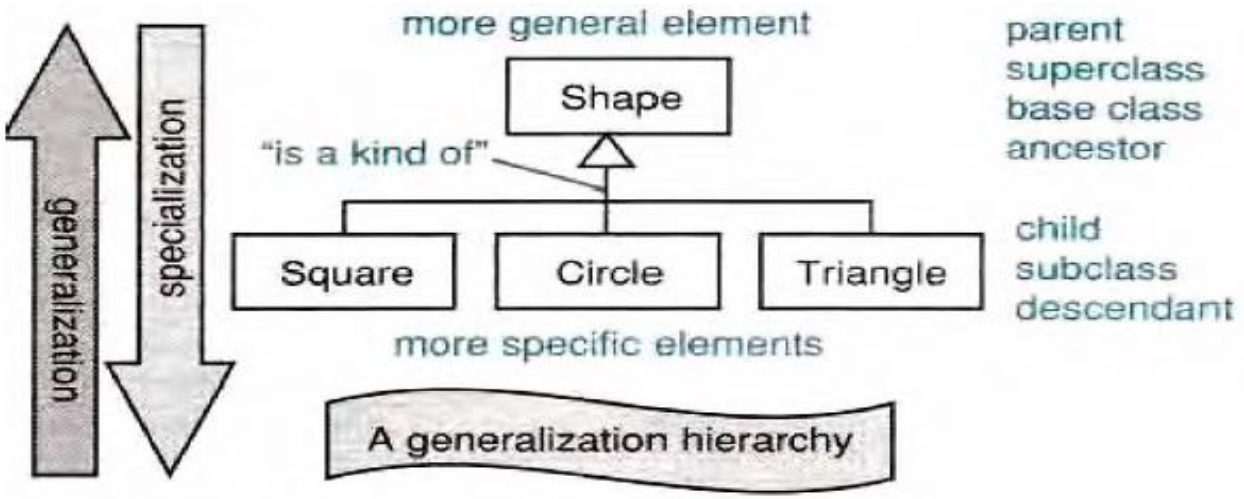
Methods/operations: *Operations* describe the class behavior and appear in the third compartment. You can specify an operation by stating its signature: listing the name, type, and default value of all parameters, and, in the case of functions, a return type. Complete representation of class components is as follows:



Relationship in class diagram:

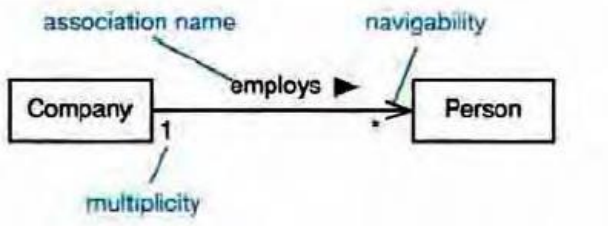
- **Generalization**

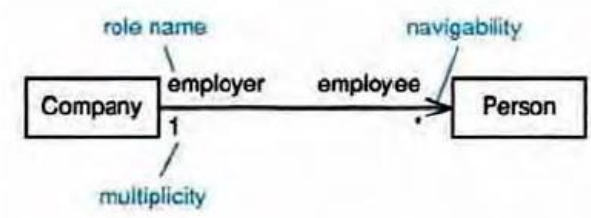
A *generalization* connects a subclass to its superclass. It denotes an inheritance of attributes and behavior from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass. UML permits a class to inherit from multiple superclasses, although some programming languages (*e.g.,* Java) do not permit multiple inheritance. Example of generalization is as follows:



- **Association**

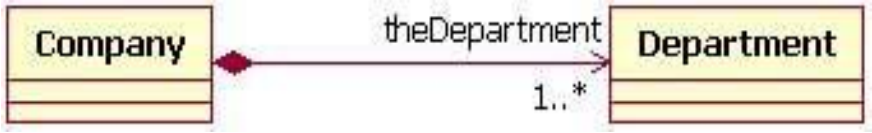
If two classes in a model need to communicate with each other, there must be a link between them.An *association* denotes that link. These links are instances of the association just as the objects are the instance of the class. Association can have name, role name, multiplicity, and navigability. For example,





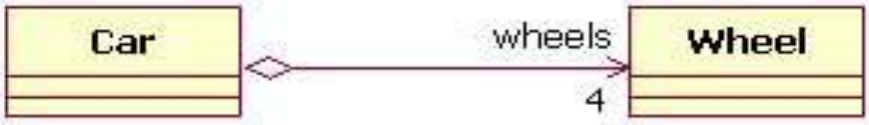
- **Composition**

Composition is the strongest form of association. In this relationship components have only one owner and they cannot exist independently. For example, a company has many departments and a department can't exist independently.



- **Aggregation**

Aggregation is also a form of association. In this relationship "a part of" concept is considered, but it is not essential. Components of a product can exist independently. For example, a wheel is a part of the car, but it can exist independently.



How to draw a class diagram?

Class diagrams are the most popular UML diagrams used for the construction of software applications. So it is very important to learn the drawing procedure of a class diagram. Class diagrams have a lot of properties to consider while drawing, but here the diagram will be considered from a top level view.

However, the class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represents the whole system. The following points should be remembered while drawing a class diagram:

- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility i.e. attributes and methods of each class should be clearly identified.
- For each class minimum number of properties should be specified. Because unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram. Because at the end of the drawing it should be understandable to the developer/coder.

So, in order to draw a class diagram you have to follow these steps:

- Identify element of the system

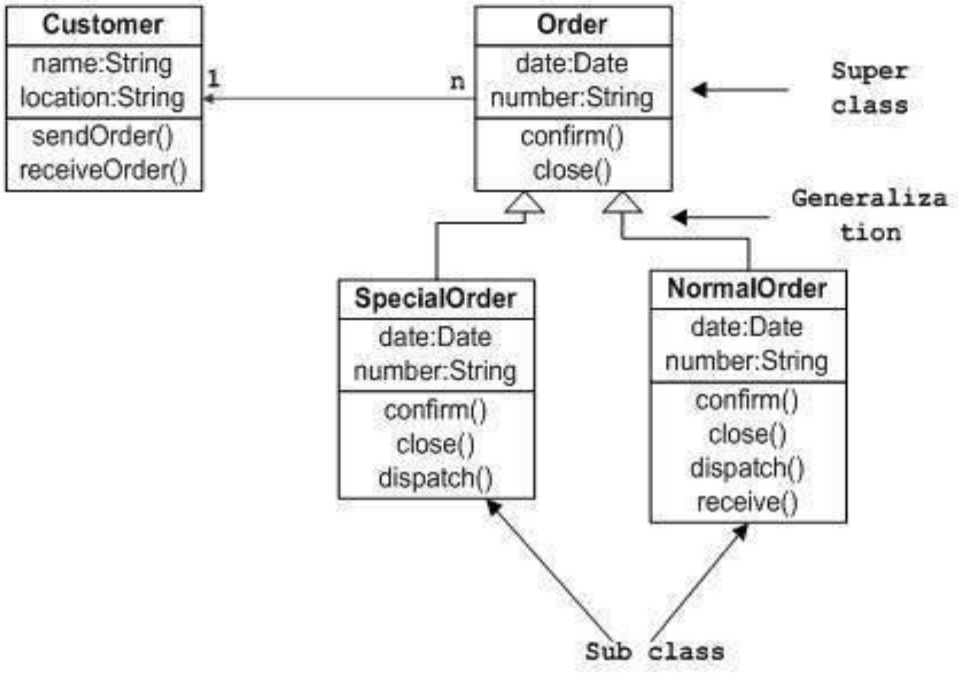
- Find a relationship between them
- Make classes for elements
- Identity functions/ methods of the classes
- Connect classes on basis of relationship

Taking an example of an Order System of an application, following are the steps for making class diagram. So it describes a particular aspect of the entire application.

- First of all *Order* and *Customer* are identified as the two elements of the system and they have a *one to many* relationship because a customer can have multiple orders.
- We would keep *Order* class is an abstract class and it has two concrete classes (inheritance relationship) *SpecialOrder* and *NormalOrder*.
- The two inherited classes have all the properties as the *Order* class. In addition, they have additional functions like *dispatch ()* and *receive ()*.

So the following class diagram has been drawn considering all the points mentioned above.

Sample Class Diagram



Usage of Class Diagram:

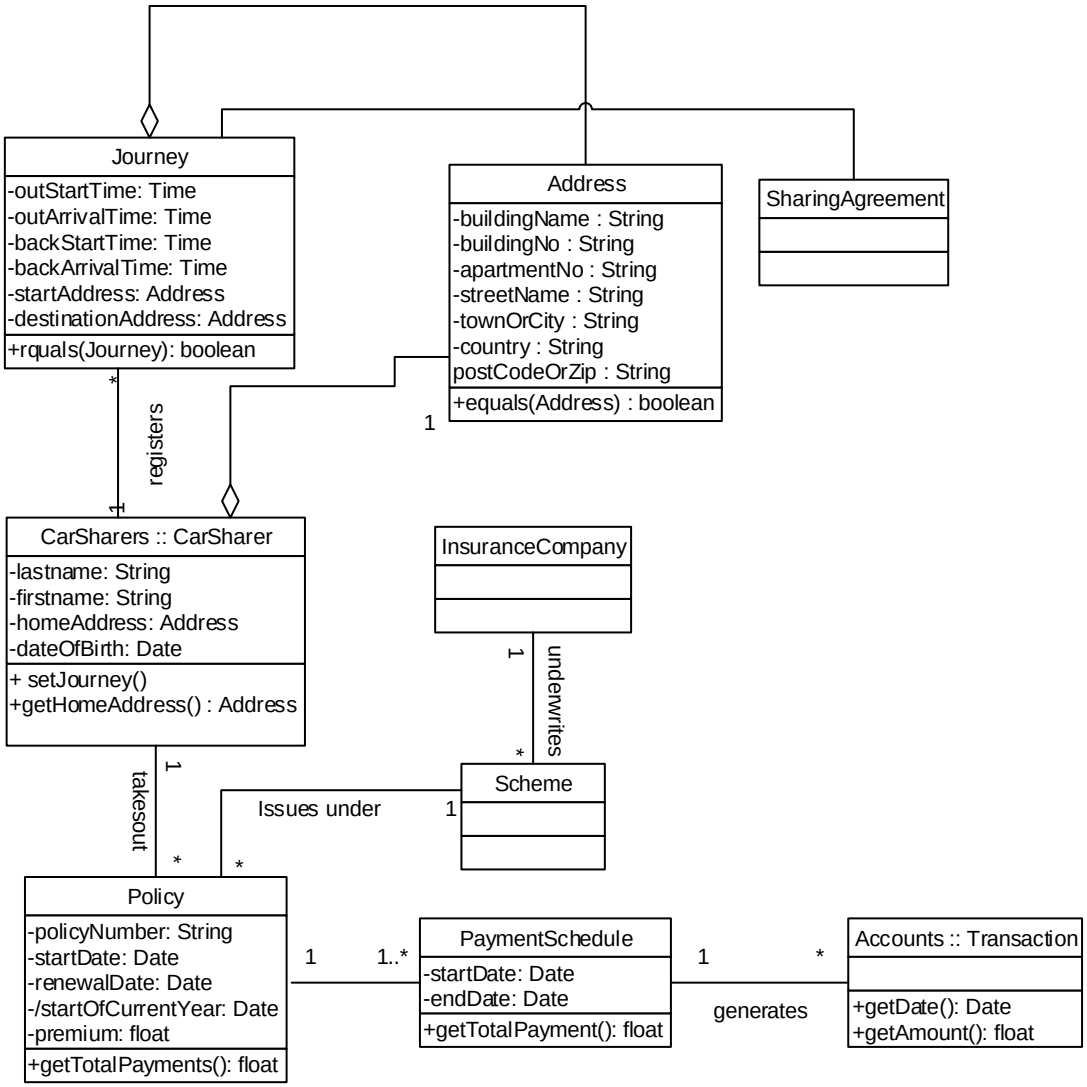
A Class diagram is considered as the foundation for component and deployment diagrams. Class diagrams are not only used to visualize the static view of the system, but they are also used to construct the executable code for forward and reverse engineering of any system. Generally UML diagrams are not directly mapped with any object oriented programming languages, but the class diagram is an exception.

Class diagram clearly shows the mapping with object oriented languages like Java, C++ etc. So from the practical experience class diagram is generally used for construction purpose.

So in a brief, 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.

Class diagram of CarMatch case study:


Journey
-outStartTime: Time
-outArrivalTime: Time
-backStartTime: Time
-backArrivalTime: Time
-startAddress: Address
-destinationAddress: Address
+rquals(Journey): boolean
Address
-buildingName : String
-buildingNo : String
-apartmentNo : String
-streetName : String
-townOrCity : String
-country : String
postCodeOrZip : String
+equals(Address) : boolean
SharingAgreement
registers
CarSharers :: CarSharer
-lastname: String
-firstname: String
-homeAddress: Address
-dateOfBirth: Date
+ setJourney()
+getHomeAddress() : Address
InsuranceCompany
underwrites
takesout
Scheme
Issues under
Policy
-policyNumber: String
-startDate: Date
-renewalDate: Date
-/startOfCurrentYear: Date
-premium: float
+getTotalPayments(): float
PaymentSchedule
-startDate: Date
-endDate: Date
+getTotalPayment(): float
generates
Accounts :: Transaction
+getDate(): Date
+getAmount(): float

References and Recommended links:

http://www.slideshare.net/erant/uml-class-diagram?next_slideshow=1 *****

http://creately.com/diagram-type/objects/class-diagram

http://creately.com/diagram-type/article/relationships-that-exist-between-classes

http://creately.com/diagram-type/objects/class-diagram

http://www.slideshare.net/gdup/class-diagrams?related=1

https://en.wikipedia.org/wiki/Class_diagram

http://www.sparxsystems.com/resources/uml2_tutorial/uml2_classdiagram.html