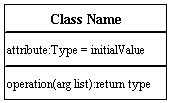
**Class Diagram**

Class diagrams are the backbone of almost every object-oriented method including UML. They describe the static structure of a system.

**Basic Class Diagram Symbols and Notations**

Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes.

Illustrate classes with rectangles divided into compartments. Place the name of the class in the first partition (centered, bolded, and capitalized), list the attributes in the second partition, and  
Write operations into the third.



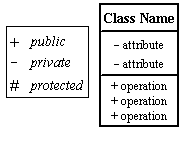
**Active Class**

Active classes initiate and control the flow of activity, while passive classes store data and serve other classes. Illustrate active classes with a thicker border.



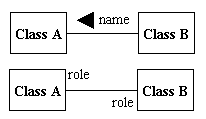
**Visibility**

Use visibility markers to signify who can access the information contained within a class. Private visibility hides information from anything outside the class partition. Public visibility allows all other classes to view the marked information. Protected visibility allows child classes to access information they inherited from a parent class. [.](http://www.smartdraw.com/resources/tutorials/Text-and-Tables)



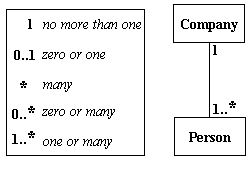
**Associations**

Associations represent static relationships between classes. Place association names above, on, or below the association line. Use a filled arrow to indicate the direction of the relationship. Place roles near the end of an association. Roles represent the way the two classes see each other.  
***Note:*** It's uncommon to name both the association and the class roles.



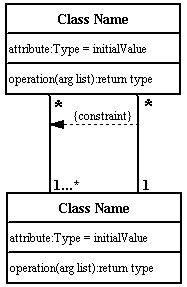
**Multiplicity (Cardinality)**

Place multiplicity notations near the ends of an association. These symbols indicate the number of instances of one class linked to one instance of the other class. For example, one company will have one or more employees, but each employee works for one company only.



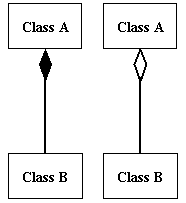
**Constraint**

Place constraints inside curly braces {}.

http://wc1.smartdraw.com/resources/tutorials/images/uml_constraint.gif*Simple Constraint* 

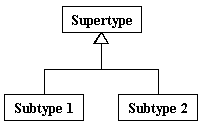
**Composition and Aggregation**

Composition is a special type of aggregation that denotes a strong ownership between Class A, the whole, and Class B, its part. Illustrate **composition** with a filled diamond. Use a hollow diamond to represent a simple **aggregation** relationship, in which the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other. The diamond end in both a composition and aggregation relationship points toward the "whole" class or the aggregate



**Generalization**

Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another. For example, Honda is a type of car. So the class Honda would have a generalization relationship with the class car.



In real life coding examples, the difference between inheritance and aggregation can be confusing. If you have an aggregation relationship, the aggregate (the whole) can access only the PUBLIC functions of the part class. On the other hand, inheritance allows the inheriting class to access both the PUBLIC and PROTECTED functions of the super class.



Fig: Admin and user class Diagram.