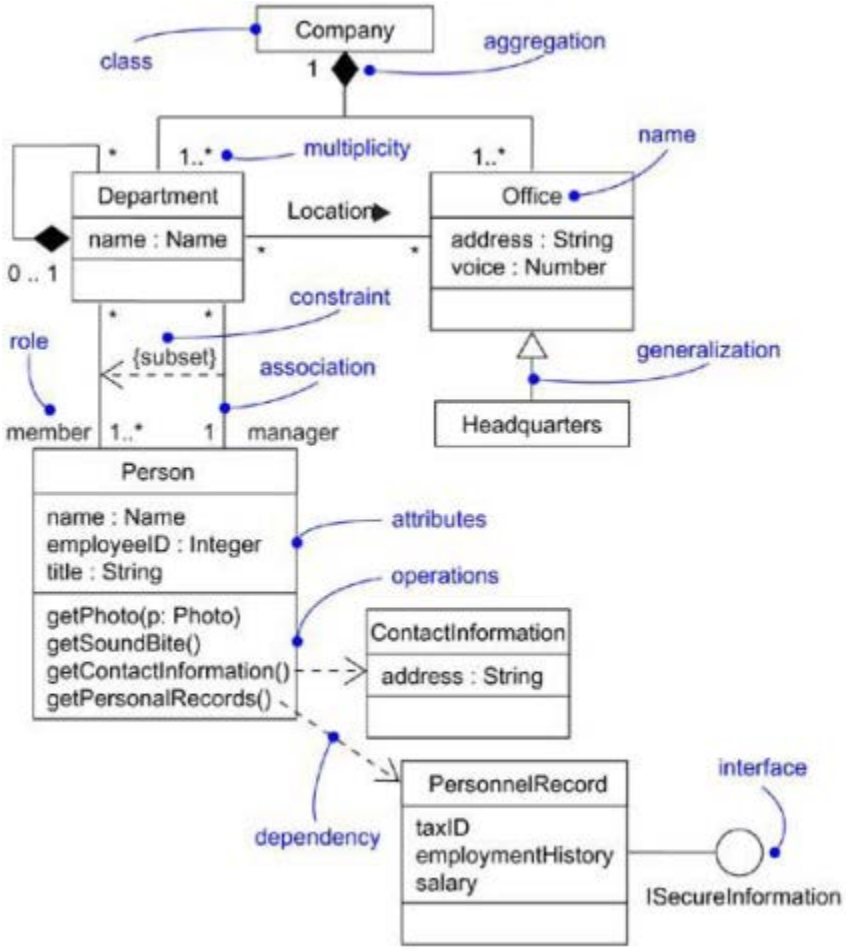
CLASS DIAGRAMS

A class diagram is a graphic presentation of the static view that shows a collection of declarative (static) model elements, such as Classes, Interfaces, Types and their contents and relationships. A class diagram may show a view of a package and may contain symbols for nested packages. A class diagram contains certain reified(even though not real but can be made real) behavioral elements, such as operations, but their dynamics are expressed in other diagrams, such as statechart diagrams and collaboration diagrams. A *sample class diagram* is shown below.



When you model the static design view of a system, you'll typically use class diagrams in one of three ways.

- To model the vocabulary of a system
- To model simple collaborations
- To model a logical database schema

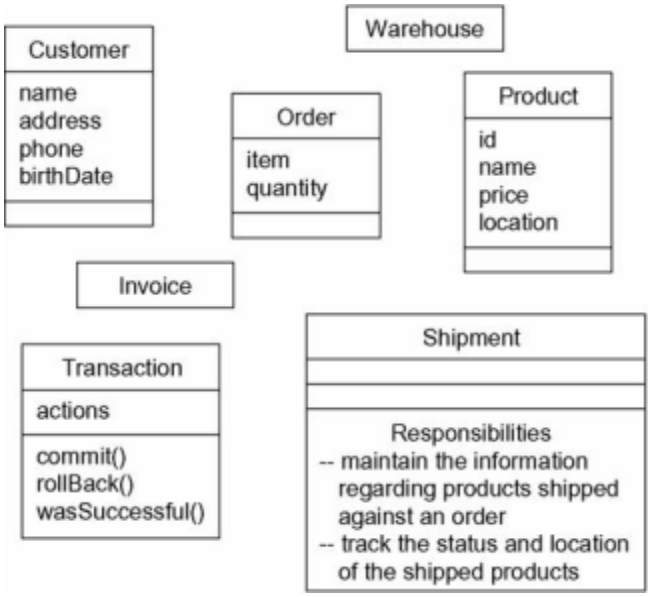
<u>To model the vocabulary of a system</u>

Modeling the vocabulary of a system involves making a decision about which abstractions are a part of the system under consideration and which fall outside its boundaries. You use class diagrams to specify these abstractions and their responsibilities.

Steps involved in modeling:

1. Identify those things that users or implementers use to describe the problem or solution
2. For each abstraction, identify a set of responsibilities
3. Provide the attributes and operations that are needed to carry out these responsibilities

A sample representation of vocabulary of a retail system is shown below,



<u>To model simple collaborations</u>

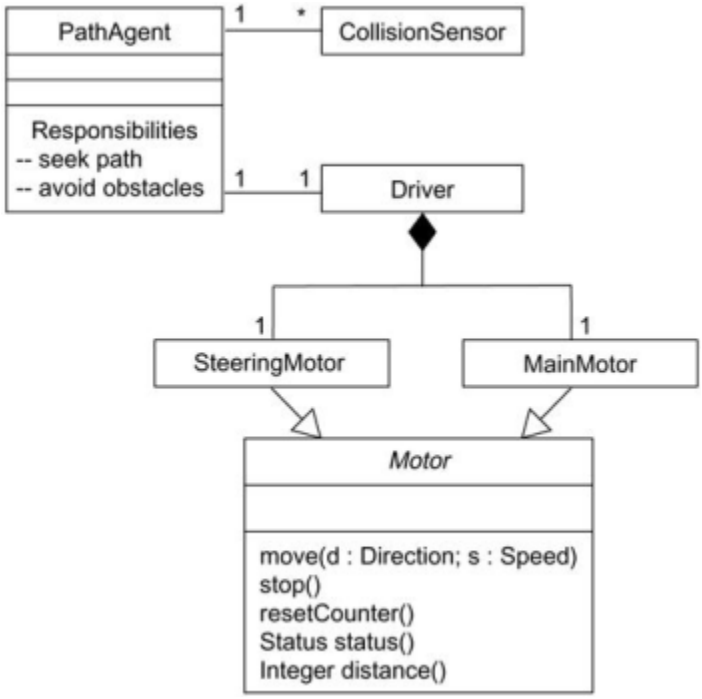
A collaboration is a society of classes, interfaces, and other elements that work together to provide some cooperative behavior that's bigger than the sum of all the elements. For example,when you're modeling the semantics of a transaction in a distributed system,

you can't just stare at a single class to understand what's going on. Rather, these semantics are carried out by a set of classes that work together. You use class diagrams to visualize and specify this set of classes and their relationships.

Steps involved in modeling:

1. Identify the mechanism you'd like to model. A mechanism represents some behavior or function of the part of the system you are modeling that results from the interaction of a society of classes, interfaces, and other things.
2. For each mechanism, identify the classes, interfaces, and other collaborations that participate in this collaboration. All these along with their relationships.
3. Use scenarios to walk through these things and in doing so you will find parts that were missing and parts that were wrong.
4. Be sure to populate these elements with their contents. For classes, start with responsibilities & later turn these into concrete attributes and operations.

An example of collaborations involved from the implementation of an autonomous robot is given below.



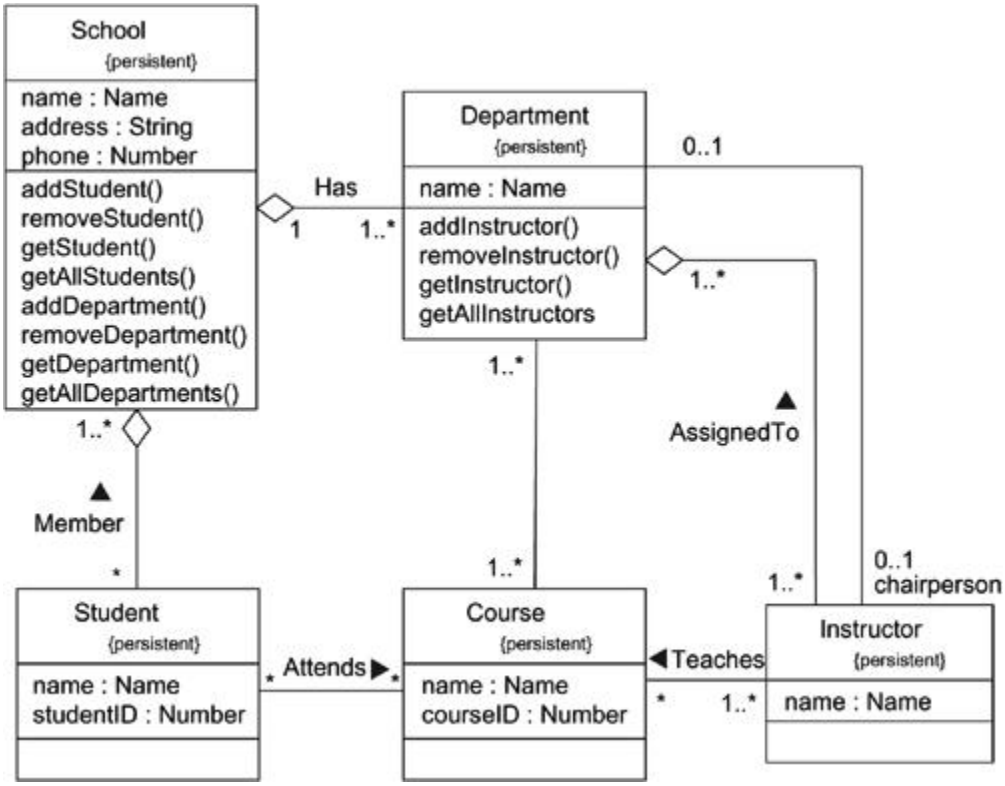
To model a logical database schema

You can think of a schema as the blueprint for the conceptual design of a database.

Here you'll want to store persistent information in a relational database or in an object-oriented database. You can model schemas for these databases using class diagrams. Steps involved in modeling:

1. Identify classes in model whose state must persist forever.
2. Create class diagram mark such classes as persistent
3. Expand the structural(static) details by their attributes and focus on associations and their cardinalities of these classes
4. For complex patterns use intermediate abstractions to simplify logical structure.
5. Expand operations that are important for data access and data integrity.
6. Where possible, use tools to help you transform your logical design into a physical design.

Diagram below shows logical database schema for a set of classes drawn from an information system for a school,



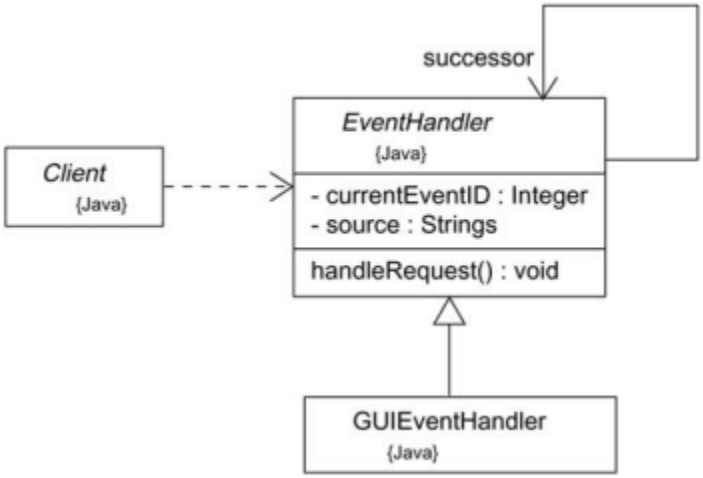
The primary product of a development team is software, not diagrams. The reason you create models is to predictably deliver at the right time the right software that satisfies the evolving goals of its users and the business.

Forward engineering is the process of transforming a model into code through a mapping to an implementation language. This results in a loss of information, because models written in the UML are semantically richer than any current object-oriented programming languages.

Steps involved for forward engineer a class diagram:

1. Identify the rules for mapping UML to your implementation language.
2. Depending on the semantics of the languages you choose, you may have to constrain your use of certain UML features. Eg:- UML permits you to model multiple inheritance, but Smalltalk permits only single inheritance.
3. Use tagged values to specify your target language.
4. Use tools to forward engineer your models.

An example of converting a class diagram into java is given below,



The corresponding java program,

```
public abstract class EventHandler {
EventHandler successor;
private Integer currentEventID;
private String source;
EventHandler() {}
public void handleRequest() {}
}
```

Reverse engineering is the process of transforming code into a model through a mapping from a specific implementation language. This results in flood of information, some of which is at a lower level of detail than you'll need to build useful models. Reverse engineering is incomplete because of the incapability of current object oriented language to express every ideas in its fullness .That is, there is a loss of information when forward engineering models into code, and so you can't completely recreate a model from code unless your tools encode information in the source comments that goes beyond the semantics of the implementation language.
Steps involved for reverse engineer a class diagram:

1. Identify the rule for mapping from implementation language to UML
2. Use tool to generate a new model or modify an existing one
3. Using your tool, create a class diagram by querying the model. For example, you might start with one or more classes, then expand the diagram by following specific relationships or other neighboring classes. Expose or hide details of the contents of this class diagram as necessary to communicate your intent.

Source : http://praveenthomasln.wordpress.com/2012/02/25/class-diagrams/