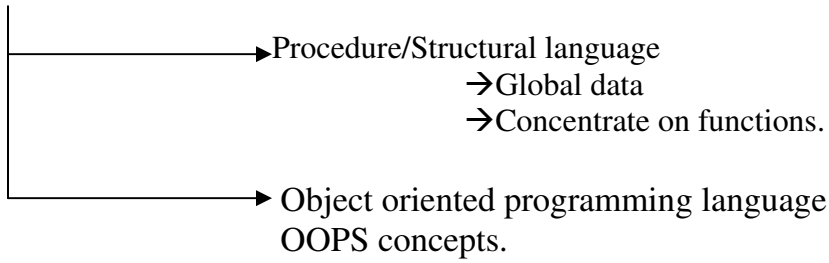
UNIT-I

Introduction:

The various trends in S/W development give the change in the languages. In earlier days S/W developers used Machine Languages, which deals with 0's and 1's [Binary Number]. S/W developers felt it was difficult to program using binary numbers. In later stage Assembly Language was used for a programming. Assembly Language uses mnemonics, which is better than binary language. Then high-level language was introduced. The human understandable English is used in the programming languages. Initial stages of high-level languages have the procedural /structural languages. Programmers concentrate more on functions rather than data. To overcome this object oriented programming languages was introduced. In OOProgramming the programmer concentrate or gives equal importance to functions and data. The advantages over procedure languages are OOPS concepts.

Machine Language→0,1
↓
Assembly Language→Mnemonics
↓
High Level Language→Human Understandable Language



The OOPS concepts are

→Data hiding
→Data encapsulation
→Data abstraction
→Inheritance
→Polymorphism
→Objects
→Class
→Dynamic binding
→Message passing.

The detailed view of oops concepts is discussed later.

OBJECT ORIENTATION:

Object oriented methods enable us to create sets of objects that work together synergistically to produce software that better module their problem domains than similar systems produced by traditional techniques. The system created using object oriented methods are easier to adapt changing requirements, easier to maintain, more robust, promote greater design. The reasons why object orientation works

→ High level of abstraction.
→Seamless transition among different phases of software development.
→Encourage of good programming techniques.
→Promotion of reusability.

High level of abstraction:

Top-down approach →It supports abstraction of the function level.

Objects oriented approach → It supports abstraction at the object level. The
object encapsulate both the data (attributes) and functions (methods), they work as a higher level of abstraction. The development can proceed at the object level, this makes designing, coding, testing, and maintaining the system much simpler.

Seamless transition among different phases of software development

Traditional Approach:

The software development using this approach requires different styles and methodologies for each step of the process. So moving from one phase to another requires more complex transistion.

Object-oriented approach:

We use the same language to talk about analysis, design, programming and database design. It returns the level of complexity and reboundary, which makes clearer and robust system development.

Encouragement of good programming techniques:

A class in an object-oriented system carefully delineates between its interface and the implementation of that interface. The attributes and methods are encapsulated within a class (or) held together tightly. The classes are grouped into subsystems but remain independent one class has no impact on other classes. Object oriented approach is not a magical one to promote perfect design (or) perfect code.

Raising the level of abstraction from function level to object level and focusing on the real-world aspects of the system, the object oriented method tends to

→ Promote clearer designs.
→ Makes implementation easier.
→ Provide overall better communication.

Promotion of Reusability:

Objects are reusable because they are modeled directly out of real world. The classes are designed generically with reuse. The object orientation adds inheritance, which is a powerful technique that allows classes to built from each other. The only differents and enhancements between the classes need to be designed and coded. All the previous functionality remains and can be reused without change.

OBJECT-ORIENTED SYSTEM DEVELOPMENT

Traditional Software Development:

The S/W development is based on function and procedures.

Object-oriented software development:

It is a way to develop software by building self-contained modules or objects that can be easily replaced, modified and reused. In an object-oriented environment, software is a collection of discrete objects that encapsulate their data as well as the functionality to model real-world objects. An object orientation yields important benefits to the practice of software construction. Each object has attributes (data) and methods (function). Objects are grouped into classes.

In object-oriented system, everything is an object and each object is responsible for itself.

For example:

→ Windows applications needs windows object that can open themselves on screen and either display something or accept input.

→ Windows object is responsible for things like opening, sizing, and closing itself.

→ When a windows display something, that something is an object. (ex) chart.

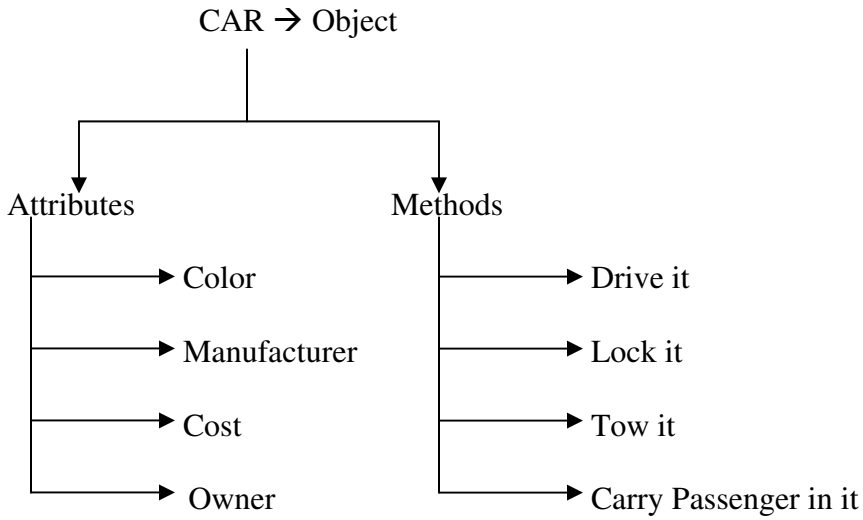
→ Chart object is responsible for maintaining its data and labels and even for drawing itself.

Review of objects:

The object-oriented system development makes software development easier and more natural by raising the level of abstraction to the point where applications can be implemented. The name object was chosen because "everyone knows what is an object is ". The real question is "what do objects have to do with system development" rather that "what is an object?"

Object:

A car is an object a real-world entity, identifiably separate from its surroundings. A car has a well-defined set of attributes in relation to other object.



Attributes:

→ Data of an object.
→ Properties of an object.

Methods:

→ Procedures of an object.
or
→ Behaviour of an object.

The term object was for formal utilized in the similar language. The term object means a combination or data and logic that represent some real-world entity.

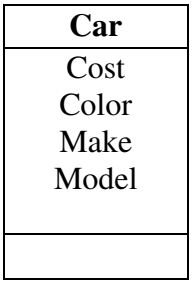
When developing an object oriented applications, two basic questions arise

→ What objects does the application need?
→What functionality should those objects have?

Programming in an object-oriented system consists of adding new kind of objects to the system and defining how they behave. The new object classes can be built from the objects supplied by the object-oriented system.

Object state and properties (Attributes):

Properties represent the state of an object. In an object oriented methods we want to refer to the description of these properties rather than how they are represented in a particular programming language.



Attributes of car object

We could represent each property in several ways in a programming languages.

For example:

Color →1. Can be declared as character to store sequence or character [ex: red, blue, ..]
2. Can declared as number to store the stock number of paint [ex: red paint, blue paint, ..]
3. Can be declared as image (or) video file to refer a full color video image.

The importance of this distinction is that an object abstract state can be independent of its physical representation.

Object Behaviour and Methods:

We can describe the set of things that an object can do on its own (or) we can do with it.

For example:

Consider an object car,
→ We can drive the car.
→ We can stop the car.

Each of the above statements is a description of the objects behaviour. The objects behaviour is described in methods or procedures. A method is a function or procedures that is defined in a class and typically can access to perform some operation. Behaviour denotes the collection of methods that abstractly describes what an object is capable of doing. The object which operates on the method is called receiver. Methods encapsulate the behaviour or the object, provide interface to the object and hide any of the internal structures and states maintained by the object. The procedures provide us the means to communicate with an object and access it properties.

For example:

An employee object knows how to compute salary. To compute an employee salary, all that is required is to send the compute payroll message to the employee object.

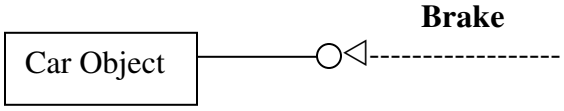
So the simplification of code simplifies application development and maintenance.

Objects Respond to Messages:

The capability of an object's is determined by the methods defined for it. To do an operation, a message is sent to an object. Objects represented to messages according to the methods defined in its class.

For example:

When we press on the brake pedal of a car, we send a stop message to the car object. The car object knows how to respond to the stop message since brake have been designed with specialized parts such as break pads and drums precisely respond to that message.



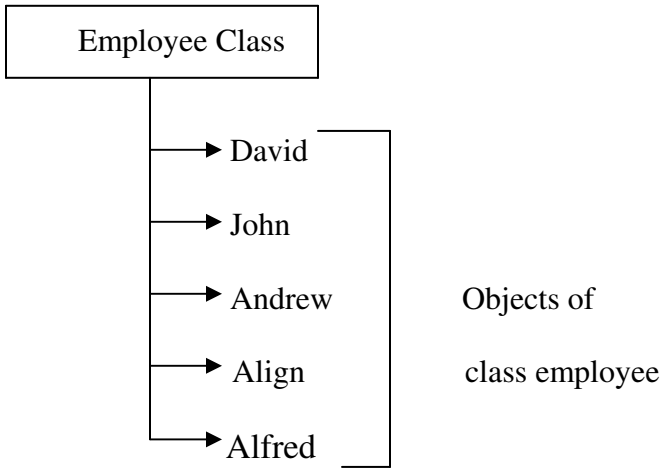
Different object can respond to the same message in different ways. The car, motorcycle and bicycle will all respond to a stop message, but the actual operations performed are object specific.

It is the receiver's responsibility to respond to a message in an appropriate manner. This gives the great deal or flexibility, since different object can respond to the same message in different ways. This is known as polymorphism.

Objects are grouped in classes:

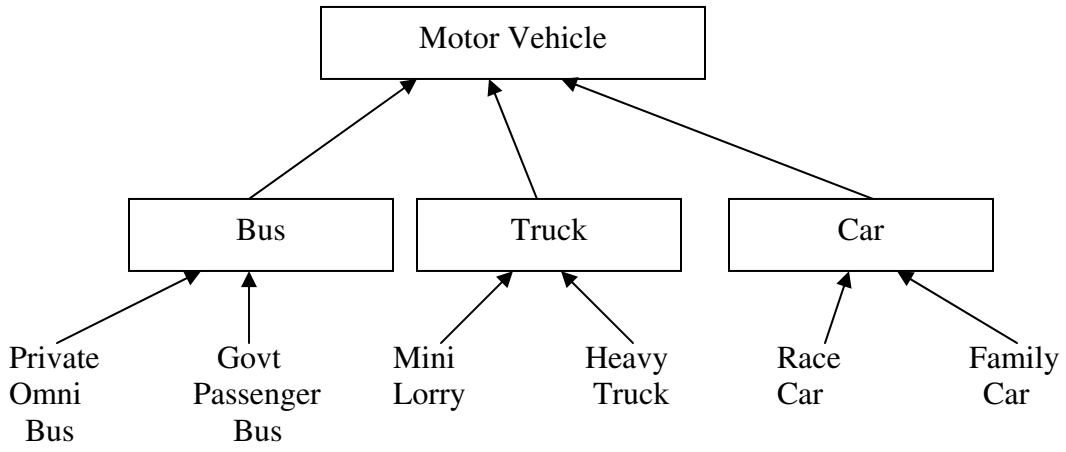
The classification of objects into various classes is based its properties (states) and behaviour (methods). Classes are used to distinguish are type of object from another. An object is an instance of structures, behaviour and inheritance for objects. The chief rules are the class is to define the properties and procedures and applicability to its instances.

For example:



Class Hierarchy:

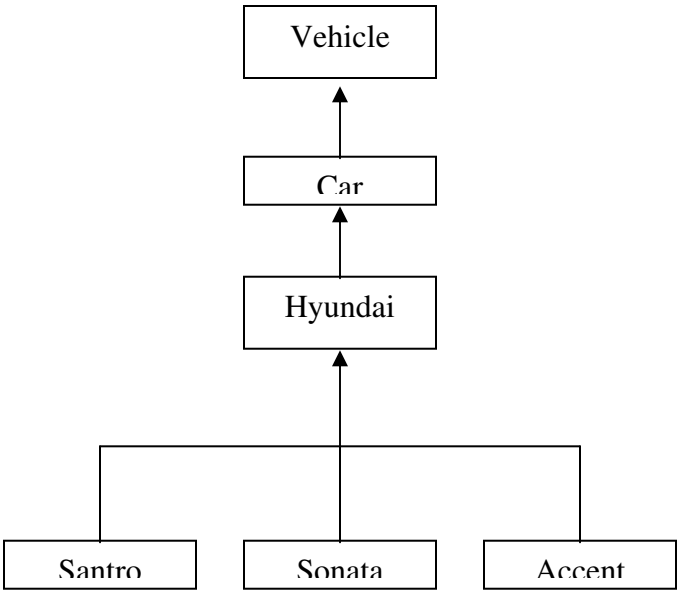
An object-oriented system organizes classes into a subclass super class hierarchy. The properties and behaviours are used as the basis for making distinctions between classes are at the top and more specific are at the bottom of the class hierarchy. The family car is the subclass of car. A subclass inherits all the properties and methods defined in its super class.



Super class/Subclass Hierarchy

Inheritance:

It is the property of object-oriented systems that allow objects to be built from other objects. Inheritance allows explicitly taking advantage of the commonality of objects when constructing new classes. Inheritance is a relationship between classes where one class is the parent class of another (derived) class. The derived class holds the properties and behaviour of base class in addition to the properties and behaviour of derived class.



Inheritance allows reusability.

Dynamic Inheritance:

Dynamic inheritance allows objects to change and evolve over time. Since base classes provide properties and attributes for objects, hanging base classes changes the properties and attributes of a class.

Example:

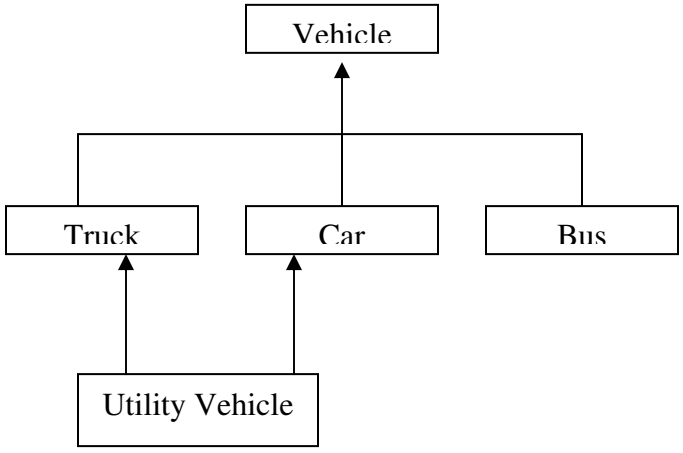
A window objects change to icon and basic again. When we double click the folder the contents will be displayed in a window and when close it, changes back to icon. It involves changing a base class between a windows class and icon class.

Multiple Inheritances:

Some object-oriented systems permit a class to inherit its state (attributes) and behaviour from more than one super class. This kind or inheritance is referred to as multiple inheritances.

For example:

Utility vehicle inherits the attributes from the Car and Truck classes.



Encapsulation and Information Hiding:

Information hiding is the principle of concealing the internal data and procedures of an object. In C++ , encapsulation protection mechanism with private, public and protected members.

In per-class protection:

Class methods can access any objects of that class and not just the receiver.

In per-object protection:

Methods can access only the receiver.

An important factor in achieving encapsulation is the design at different classes of objects that operate using a common protocol. This means that many objects will respond to the message using operations tailored to its class.

A car engine is an example of encapsulation. Although engines may differ in implementation, the interface between the driver and car is through a common protocol.

Polymorphism:

Poly → ”many”
Morph → “form”

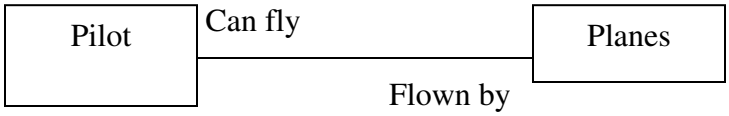
It means objects that can take on or assume many different forms. Polymorphism means that the same operations may behave differently on different classes. Booch defines polymorphism as the relationship of objects many different classes by some common super class. Polymorphism allows us to write generic, reusable code more easily, because we can specify general instructions and delegate the implementation detail to the objects involved.

Example:

In a pay roll system, manager, office worker and production worker objects all will respond to the compute payroll message, but the actual operations performed are object specific.

Object Relationship and associations:

Association represents the relationships between objects and classes. Associations are bi-directional. The directions implied by the name are the forward direction and the opposite is the inverse direction.



A pilot “can fly” planes. The inverse of can fly is “is flown by “. Plane “is flown by” pilot

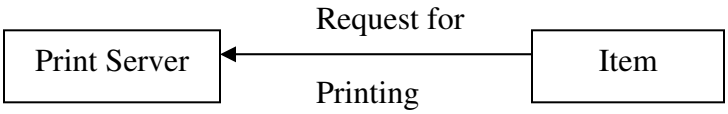
Cardinality:

It specifies how many instances of one class may relate to a single instance of an associated class. Cardinality constrains the number of related objects and often is described as being “one” or “many”.

Consumer-producer association:

A special form or association is a consumer-producer relationship, also known as a client-server association (or) a use relationship. It can be viewed as one-way interaction. One object requests the service or another object. The object that makes the request is the consumer or client and the object that receives the request and provides the service is the producer (or) server

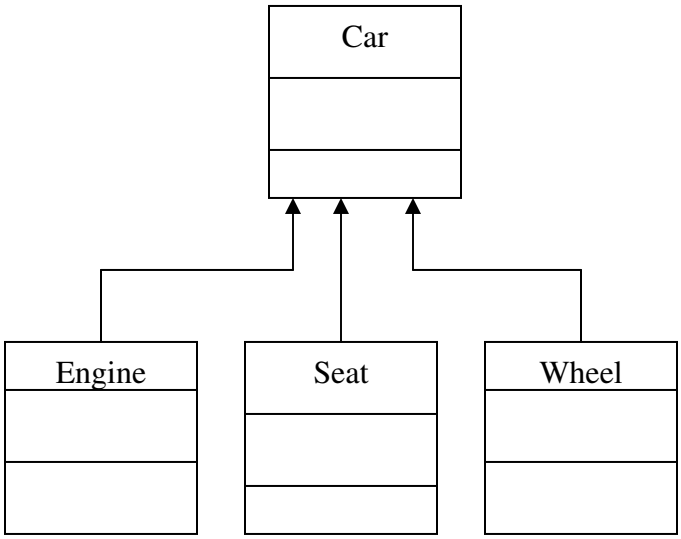
Example:



The consumer-producer association we have a print object that prints the consumer object. The print producer provides the ability to print other objects.

Aggregations:

All objects, except the most basic ones, are composed of and may contain other objects. Breaking down objects in to the objects from which they are composed is de composition. This is possible because an objects attributes need not be simple data fields, attributes can reference other objects. Since each object has an identity, one object can refer to other objects. This is known as aggregation. The car object is an aggregation of other objects such as engine, seat and wheel objects.



Static and Dynamic Binding:

Determining which function has to be involved at compile time is called static binding. Static binding optimized the calls. (Ex) function call.

The process of determining at run time which functions to involve is termed dynamic binding. Dynamic binding occurs when polymorphic call is issued. It allows some method invocation decision to be deferred until the information is known.

Example:

Cut operation in a edit submenu. It pass the cut operation to any object on the desktop, each or which handles the message in its own way.

Object Persistence:

Objects have a lifetime. They are explicitly created and can exist for a period of time that has been the duration of the process in which they were created. A file or database can provide support for objects having a longer lifeline, longer than the duration of the process for which they are created. This characteristic is called object persistence.

Meta-Classes:

In an object-oriented system every thing is an object, what about a class? Is a class an object?. Yes, a class is an object. So, If it is an object, it must belong to a class, such a class belong to a class called a meta-class (or) class or classes.

Object-Oriented Systems Development Life Cycle [OOSDLC]

Introduction:

The S/W development process that consists of Analysis, Design, implementation, testing and refinement is to transform users needs into a software solution that satisfies those needs. Some people view the s/w developing process as interesting but feel it has less importance in developing s/w. Ignoring the process and plunge into the implementation and programming phases of s/w development is much like the builder who would by pass the architect. The object oriented approach requires a more rigorous process to do things right. We have to spend more time on gathering requirements, developing a requirements model and an analysis model, then turning them into the design model. We should consult a prototype of some of the key system components shortly after the products are selected. It is used to understand how easy or difficult it will be to implement some of the features of the system.

Software Development process:

S/W development can be viewed as a process. The development itself is a process of change, retirement, transformation (or) addition to the existing product. It is possible to replace one sub process with a new one, as long as the new sub process has the same interface as the old one, to allow it to fit into the process as a whole. The object-oriented approach provides us a set of rules for describing inheritance and specialization in a consistent way when a sub process changes the behaviour of its parent process. The process can be divided into small, interacting phases know as sub processes. The sub processes must be defined in such a way that they are clearly spelled out, to allow each activity to be performed as independently of other sub processes as possible. Each sub process must have

→ A description in terms of how it works

→ Specification of the input required for the process

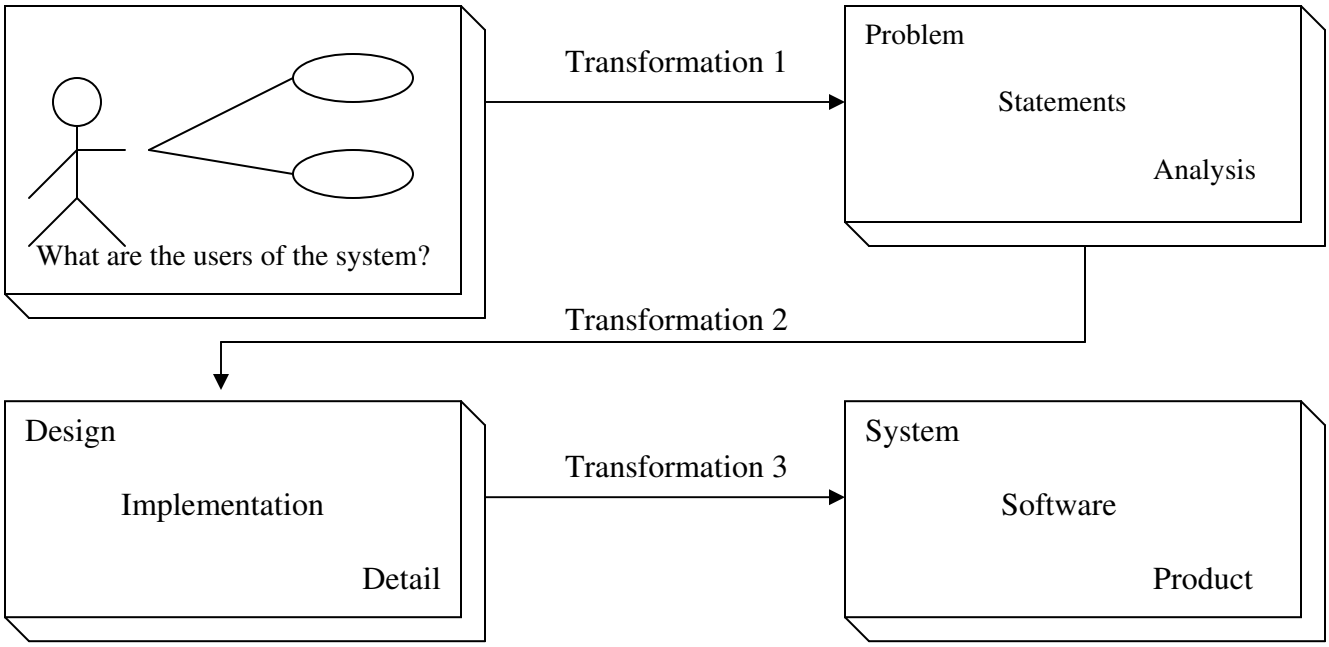
→ Specification of the output to be produced.

The software development process can be viewed as a series of transformations, where the output of one transformation becomes the input of the subsequent transformation.

Transformation 1 [Analysis]

Transformation 2 [Design]

Transformation 3 [Implementation]



Transformation 1 [Analysis]

It translates the users' needs into system requirements and responsibilities. The way they use can provide insight into user requirements.

Transformation 2 [Design]

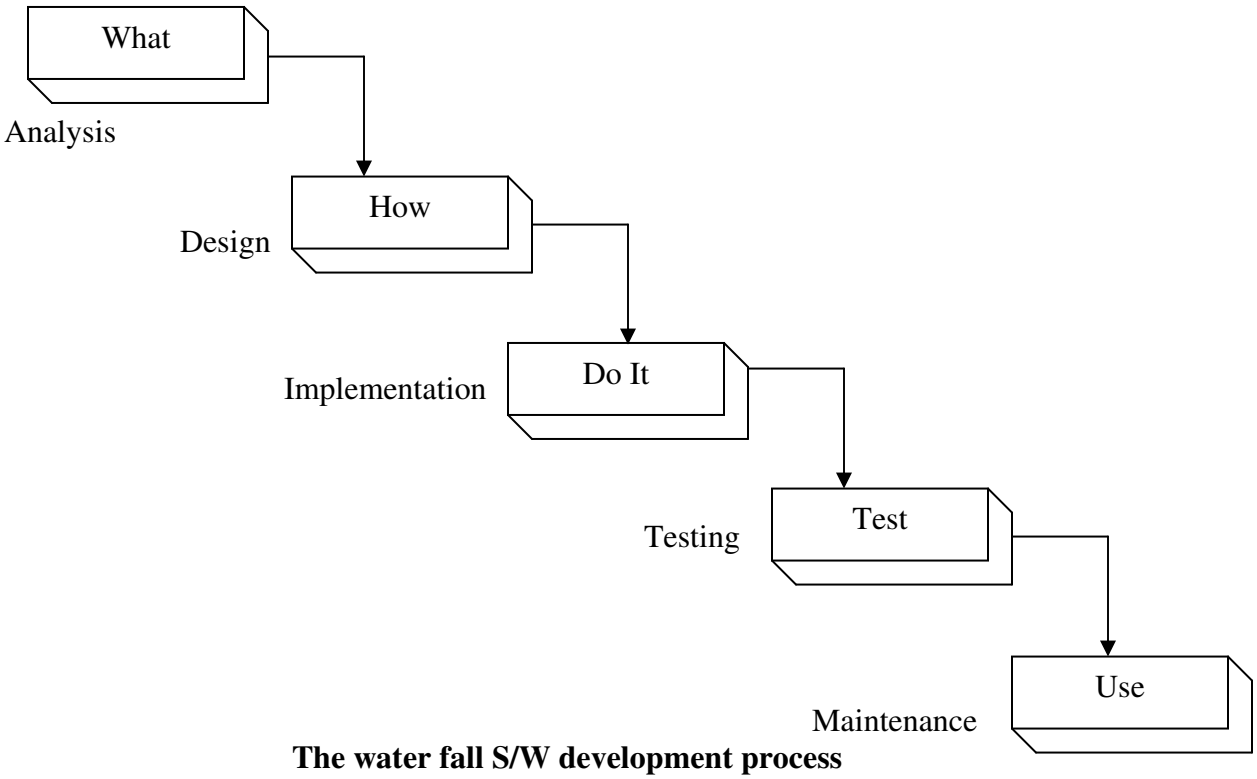
It begins with a problem statement and ends with a detailed design that can be transformed into an operational system. This transformation includes the bulk of the software development activity, including definition of how to build the software, its development and it's testing. It includes the design descriptions, the program and the testing materials.

Transformation 3 [Implementation]

It refines the detailed design into the system deployment that will satisfy the users needs. It represents embedding the software product within its operational environment.

The software development process is the waterfall approach which starts with deciding

→**what** is to be done (what is the problem)
→ **How to** accomplish them
→ Which we **do it**
→ **Test** the result to see it we have satisfied the users requirements
→ Finally we **use** what we have done



The water fall S/W development process

Building High-Quality Software

The software process transforms the users needs via the application domain to a software solution that satisfies those needs. High-Quality products must meet users needs and expectations. The quality of the product should be improved prior to delivery rather than correcting them after deliver.

To achieve high quality software we need to be able to answer the following question.

→ How do we determine when the system is ready for delivery?

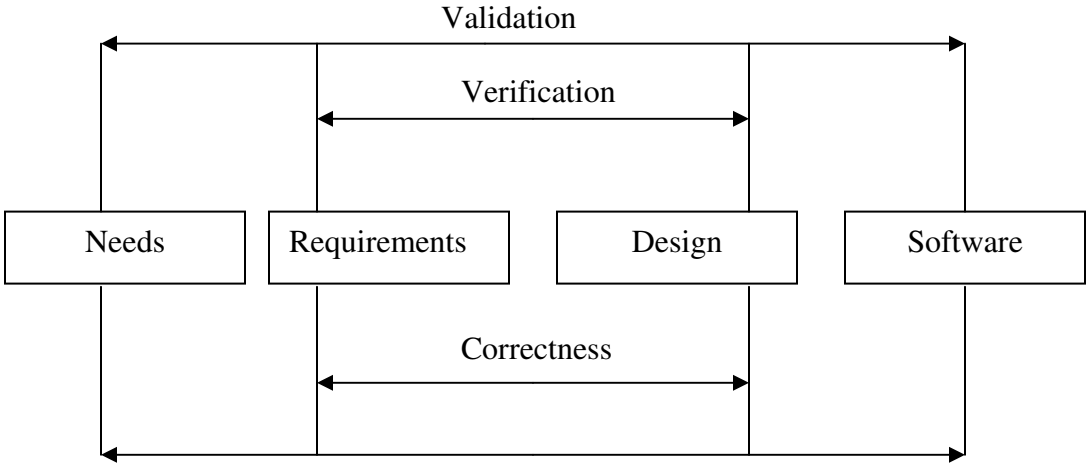
→ It is now operational system that satisfies uses needs?

→ It is correct and operating as we thought it should?

→ Does it pass an evaluation process?

There are different approaches for systems testing. Blum describes a means of system evaluation in terms of four quality measures,

→ Correspondence
→ Correctness
→ Verification and
→ Validation



* Correspondence measures how well the delivered system matches the needs of the operational environment as described in the original requirements statement.

* Validation is the task of predicting correspondence. True correspondence can be determined only after the system is in place

* Correctness measures the consistency of the product requirements with respect to the design specification

* Verification is the exercise of determining correctness.

Boehm observes that these quality measures, verification and validation is answering the following questions.

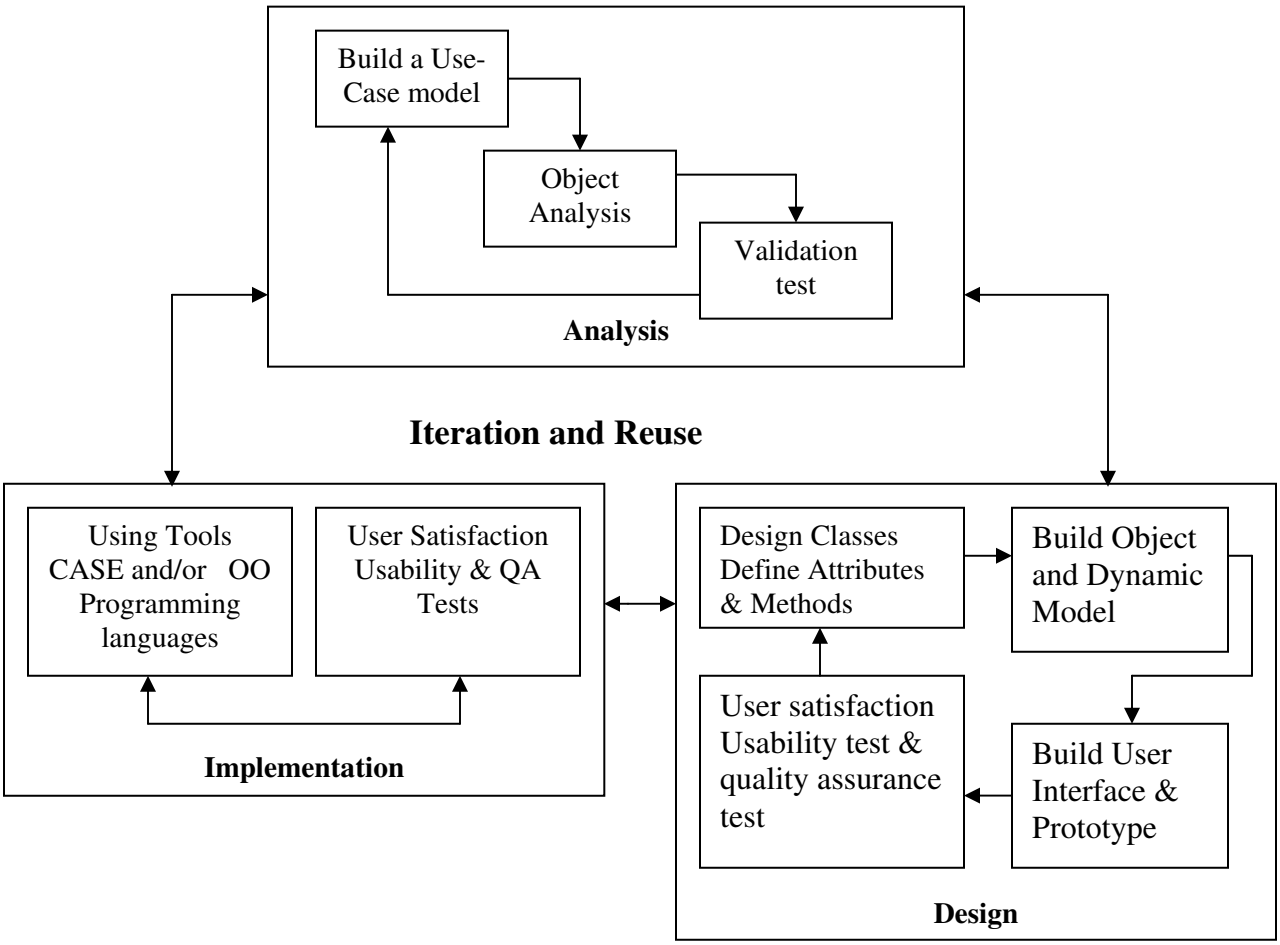
→ Verification- Am I building the product right?

→ Validation- Am I building the right product.

Object-Oriented Systems Development: **A use-case Driven Approach:**

The object-oriented S/W development Life Cycle (SDLC) consists of three macro process.

→ Object-Oriented Analysis
→ Object-oriented Design and
→ Object-oriented Implementation.



The use-case model can be employed throughout most activities of software development. The main advantage is that all design decisions can be traced back directly to user requirements.

→ Object-oriented Analysis – Use case driven
→ Object-oriented design
→ Prototyping
→ Component-based development
→ Incremental testing

Object-Oriented Analysis –Use-Case Driven:

The object-oriented analysis phase of S/W development is concerned with determing the system requirements and identifying classes and their relationship to other classes in the problem domain. To understand the system requirements we need to identify the users or the actors. Who are the actors and how do they use the system, scenarios are used to help analysis to understand the requirements. Ivar Jacobson came up with the concept of the use case, his name for scenario to describe user-computer system inter action. The object-oriented community has adopted use case to a remarkable degree.

Scenarios are a great way of examine who does what in the interactions among objects and what role they play. That is their inert relationship. This inter actions among the objects roles to achieve a given goal is called collaboration.

A use-case is a typical interaction between a user and a system that captures user goals & needs. Expressing these high-level processes and interactions it is referred to as use-case modeling. Once the use case model is better understood and developed we should start to identify classes and create their relationships.

The physical objects in the system also provide us important information an objects in the system. The objects could be individuals' organizations, machines, units of information; pictures (or) what ever else makes up the application and makes sense in the context of the real-world system.

For example: The object in the payroll system is as follows,

→ The employee, worker, supervisor, office admin.
→ The paycheck.
→ The product being made.
→ The process used to make the product.

The objects need to have meaning only within the context of the application domain.

Few guide lines to use in object-oriented design.
→ Reuse, rather than build, anew class, know the existing classes.
→ Design a large number of simple numbers of simple classes, rather than a small number of complex classes.
→ Design methods.
→ Critique what we have proposed. It possible go back and refine the classes.

Prototyping:

It is important to construct a prototype of some of the key system components shortly after the products are selected. A prototype is a version of a software product developed in the early stages of the product's life cycle for specific, experimental purposes. It enables to fully understand how easy or difficult it will be to implement some of the features of the system. It gives users a chance to comment on the usability and usefulness of the user interface design, it can define use cases and it makes use Case modeling much easier.

Prototyping was used as a "quick and dirty" way to test the design, user interface and so forth, something to be thrown away when the "industrial strength" version was developed. The rapid application development (RAD) refines the prototype into the final product.

Prototypes have been categorized in various ways. The following categorized are some of the commonly accepted prototypes.

- Horizontal prototype
- Vertical prototype
- Analysis prototype
- Domain prototype

Horizontal Prototype:

It is a simulation of the interface but contains no functionality. This has the advantages of being very quick to implement, providing a good overall feel of the system and allowing users to evaluate the interface on the basis of their normal, expected perception of the system.

Vertical Prototype:

It is a subset of the system features with complete functionality. The advantage of this method is that the few implemented functions can be tested in great depth. The prototypes are hybrid between horizontal and vertical, the major portions of the interface are established so the user can get the feel of the system and features having a high degree of risk are prototyped with much more functionality.

Analysis Prototype:

It is an aid for exploring the problem domain. This class of prototype is used to inform the user and demonstrate the proof of a concept. It is not used as the basis of development and is discarded when it has served its purpose.

Domain Prototype:

It is an aid for the incremental development of the ultimate software solution. It demonstrates the feasibility of the implementation and eventually will evolve into a deliverable product.

The typical time required to produce a prototype is anywhere from a few days to several weeks, depending on the type and function of prototype. The prototype makes the end users and management members to ascertain that the general structure of the prototype meets the requirements established for the overall design. The purpose of this review is

To demonstrate that the prototype has been developed according to the specification and that the final specification is appropriate.

To collect information about errors or other problems in the system, such as user interface problems that need to be addressed in the intermediate prototype stage

To give management and everyone connected with the project the first glimpse of what the technology can provide.

Prototyping is a useful exercise of almost any stage of the development. Prototyping should be done in parallel with the preparation of the functional specification. It also results in modification to the specification.

Implementation:

Software components are built and tested in-house, using a wide range of technologies. Computer aided software engineering (CASE) tools allow their users to rapidly develop information systems. The main goal of CASE technology is the automation of the entire information system's development life cycle process using a set of integrated software tools, such as modeling, methodology and automatic code generation. The code generated by CASE tools is only the skeleton of an application and a lot needs to be filled in by programming by hand.

Component-Based Development: (CBD)

CASE tools are the beginning of Component-Based Development. Component-Based Development is an industrialized approach to the software development process. Application development to assembly of prebuilt, pretested, reusable software components that operate with each other: The two basic ideas of using Component-Based development.

1. The application development can be improved significantly if applications can be assembled quickly from prefabricated software components.

2. An increasingly large collection of interpretable software components could be made available to developers in both general and specified catalogs.

A CBD developer can assemble components to construct a complete software system. The software components are the functional units of a program, building blocks offering a collection of reusable services. The object-Oriented concept addresses analysis, design and programming, where as component-Based development is concerned with the implementation and system integration aspects of software development.

Rapid Application Development (RAD):

RAD is a set of tools and techniques that can be used to build application footer than typically possible with traditional methods. The term is often conjunction with S/W prototyping. RAD encourages the incremental development approach of "grow, do not build" software.

Testing:
(Refer Software Engineering Book)

Design Patterns:

Design pattern is instructive information for that captures the essential structure and insight of a successful family of proven design solutions to a recurring problem that arises within a certain context.

Gang Of Four (GoF) [Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides] introduced the concept of design patterns.

Characteristics of Design Patterns:

1. *It solves the problem* – Design patterns are not just abstract representations of theoretical research. To be accepted as a pattern it should have some proves practical experiences.
2. *It's a proven concept* – Patterns must have a successful history.
3. *It describes a relationship* – Patterns do not specify a single class instead it specifies more than one classes and their relationship.
4. *It has a human component* - Good patterns make the job of the programmer easy and time saving.

Contents of Design Pattern:

- Name of the pattern is used to identify the pattern as well as an descriptive of the problem solution in general. Easy to remember and context related names makes remembering patterns easy.
- Context of the pattern describes when and where the pattern is applicable. It also describes the purpose of pattern and also the place where it is not applicable due to some specific conditions.
- Solution of the design pattern is describes how to build the appropriate design using this appropriate design.
- Consequences of design patterns describe the impact of choosing a particular design pattern in a system.

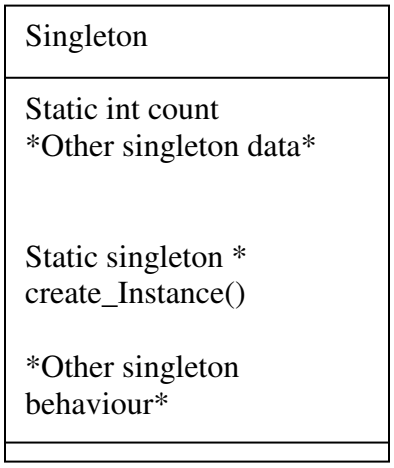
Pattern Template:

1. PATTERN NAME (good and relevant names make patterns easy to remember)
2. INTENT (Which problem does the pattern solve)
3. ALSO KNOWN AS(alias names given to the pattern)
4. APPLICABILITY(when should this pattern be applied)
5. STRUCTURE(Graphical representation of the Pattern (using UML))
6. PARTICIPANTS(classes and objects taking part in the pattern and their relationship)
7. COLLABORATORS (says how objects/actors interact to achieve the goal).
8. CONSEQUENCES (how does they solve the problem and what are the consequences if the problem is solved by this way.)
9. IMPLEMENTATION (Issues related with Implementation, language specific issues)
10. KNOWN USES (Examples of the same pattern used in real systems)
11. RELATED PATTERNS (Specify is there any similar patterns. Where are they used.)

The Singleton Design Pattern:

1. **Pattern Name –** Singleton
2. **Intent –** To ensure a class has only one instance a global point of access to it.
3. **Motivation –** Its common in software development where some component developers specify that more than one object of a Class alive make systems ambiguous.

4. **Applicability –** Singleton can be used where there must be exactly one object and it must be accessible to multiple clients/objects.
5. **Structure:**



6. **Participants –** Singleton class defines a Class function which can be accessed by the clients for creating instance.
7. **Collaborations –** Clients access a singleton object solely through instance operation.
8. **Consequences –** Controlled access to the single instance, Reduced name space, can be sub classed and more behaviors can be added, and can be modified for existence of more than one objects(BASED ON THE DOMAIN).

Implementation:

```
class singleton
	{
		private:
			static int no_of_obj;
			static singleton * pointer_instance;
			// other private data members and member functions

		public:
			static singleton * create_instance()
				{
					if (no_of_obj==0)
						{
							pointer_instance=new singleton;
							no_of_obj=1;
						}
					return pointer_instance;
				}
	}
```

//function f() uses the following statement to create a new instance:

```
singleton s=singleton::create_instance();
```

Generative Patterns: Patterns that suggest the way of finding the solution
Non Generative patterns: They do not suggest instead they give a passive solution.

Non Generative patterns cannot be used in all the situation.

Frameworks:

Frameworks are the way of delivering application development patterns to support/share best development practice during application development.

In general framework is a generic solution to a problem that can be applied to all levels of development. Design and Software frameworks and most popular where Design pattern helps on Design phase and software frameworks help in Component Based Development phase.

Framework groups a set of classes which are either concrete or abstract. This group can be sub classed in to a particular application and recomposing the necessary items.

a. Frameworks can be inserted in to a code where a design pattern cannot be inserted. To include a design pattern the implementation of the design pattern is used.
b. Design patterns are instructive information; hence they are less in space where Frameworks are large in size because they contain many design patterns.
c. Frameworks are more particular about the application domain where design patterns are less specified about the application domain.

Note:

Include the details of Microsoft .Net development framework.

UNIT –II

OBJECT ORIENTED METHODOLOGIES:

Overview of methodologies:

In 1980's, many methodologists were wondering how analysis and design methods and processes would fit into an object-oriented world. Object oriented methods suddenly had become very popular and it was apparent that the techniques to help people execute good analysis and design were just as important as the object-oriented methodologies, sometimes called second-generation object-oriented methods.

Many methodologies are available to choose from for system development. The methodology is based on modeling the business problem and implementing the differences lie primary in the documentation of information, modeling notations and languages. An application an be implemented in many ways to meet some requirements and provide the same functionality. Two people using the methodology may produce

applications designs that look radically different. This does not necessarily mean that one is right and one is wrong, just that they are different.

The various methodologies and their notations are developed by
→ Jim Rum Baugh
→ Grady Booch
→ Ivar Jacobson
These is the origin of the UML (Unified Modeling Language)

Each method has its own strengths.

Rum Baugh Method → Describing the object model or the static structure of the system.

Jacobson Method → Good for providing user driven analysis models.

Booch Method → Produces detailed object-oriented design methods.

Object Modeling Technique (OMT) or Rum Baugh ET AL's Object Modeling Technique*:*

The object modeling techniques (OMT) presented by Jim Ram Baugh and his counters describes a method for the analysis, design and implementation or system using an object-oriented technique. Object modeling technique (OMT) is a fast, intuitive approach for identifying and modeling all the objects all the objects making up a system. The information such as class attributes, methods, inheritance and association also can be expressed easily. The dynamic behavior of objects within a system is described in OMT dynamic model. This dynamic model specifies the detailed state transition and description. The relationships can be expressed in OMT's functional model.

OMT consists of four phases, which can be performing iteratively.

- **Analysis**: The results are objects and dynamic and functional models.
- **System design**: The results are a structure of the basic architecture of the system along with high-level strategy decisions.
- **Implementation**: The activity produces reusable, extensible and robust code.

OMT separates modeling into three different parts.

- **An object model:**

  Presented by the object model and the data dictionary.

- **An dynamic model:**

  Presented by the state diagrams and event flow diagrams.

- **Functional model:**

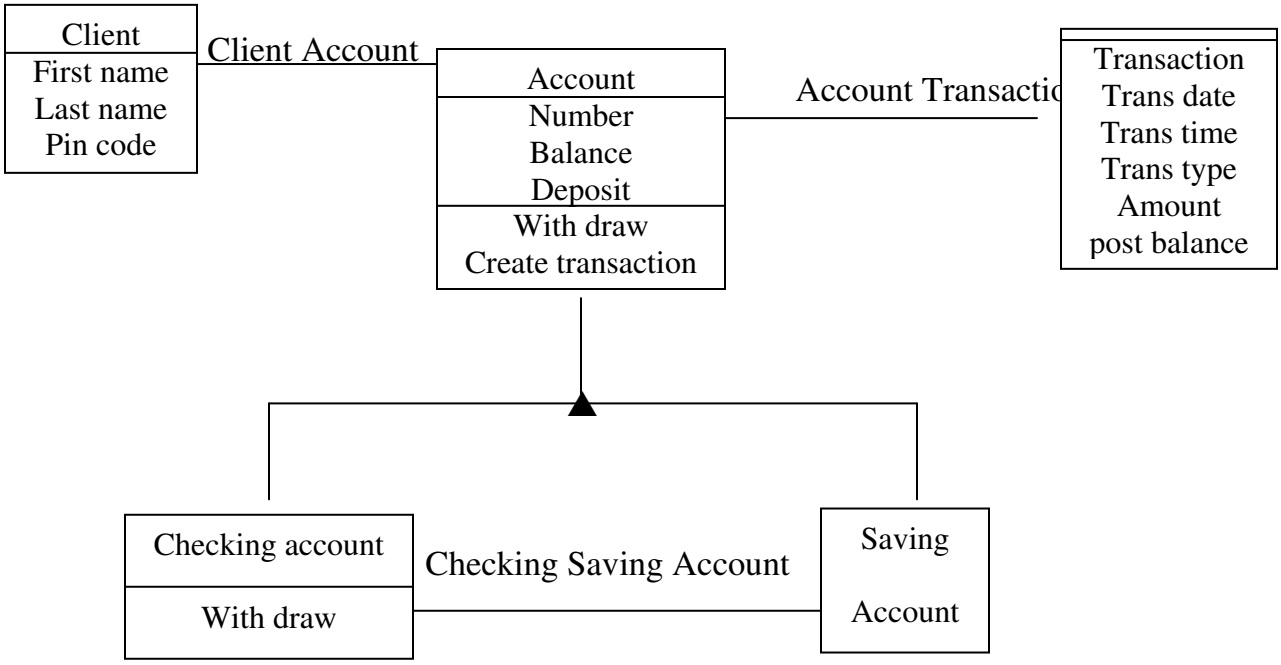
  Presented by data flow and constraints.

The Object model:

It describes the structure of objects in a system, their identity, relationships to other objects, attribute and operations. This model is graphically represented by an object diagram, which contains classes interconnected by association lines. The object diagram contains classes interconnected by association lines. The association lines establish relationships among the classes. The links from the objects or one class to the objects or another class.

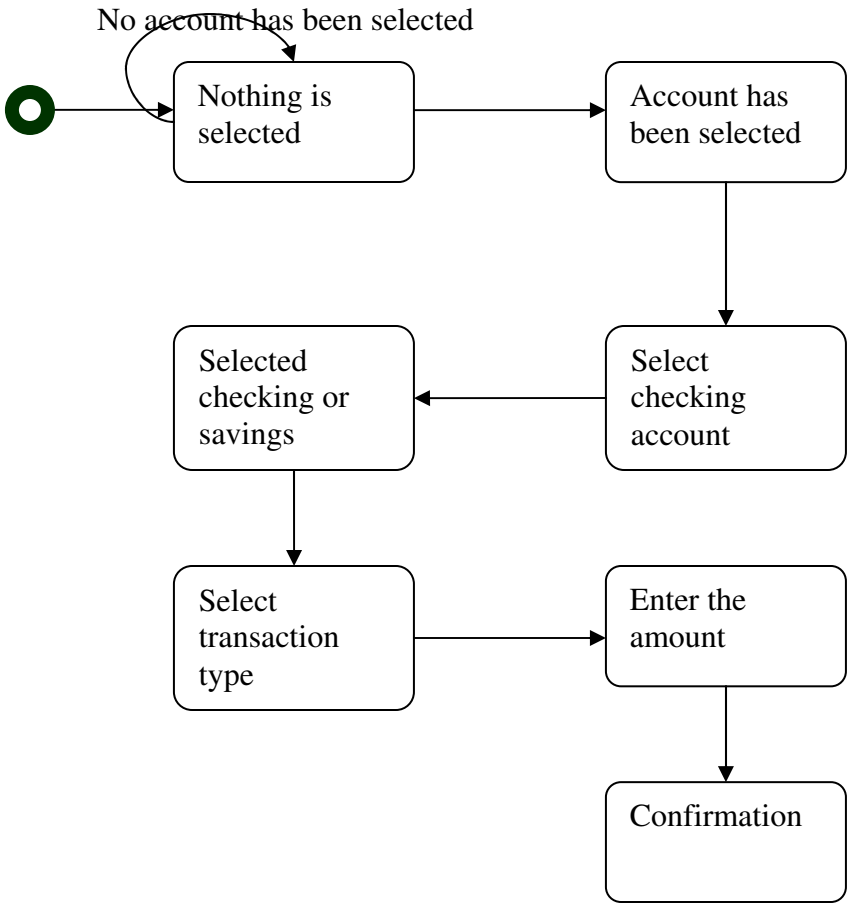
The **OMT** object model of a bank system

The object model of bank system. The boxes represent classes and the field triangle represents specialization.



The OMT dynamic model

OMT provides a detailed and comprehensive dynamic model. The state transition diagram is a network of states, transitions, events, and actions. Each state receives one or more events and the next state depends on the current state as well as the events.



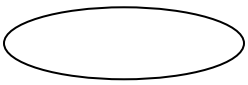
The OMT functional model

The OMT data flow diagram (DFD) shows the flow of data between different processes in a business. An OMT DFD provides a simple and intuitive method for describing business process with out focusing on the details of computer systems.

Data flow diagrams use four primary symbols:

- **Process:**

  The process is any function being performed. (ex: verify password or pin in the ATM)



- **Data flow**:

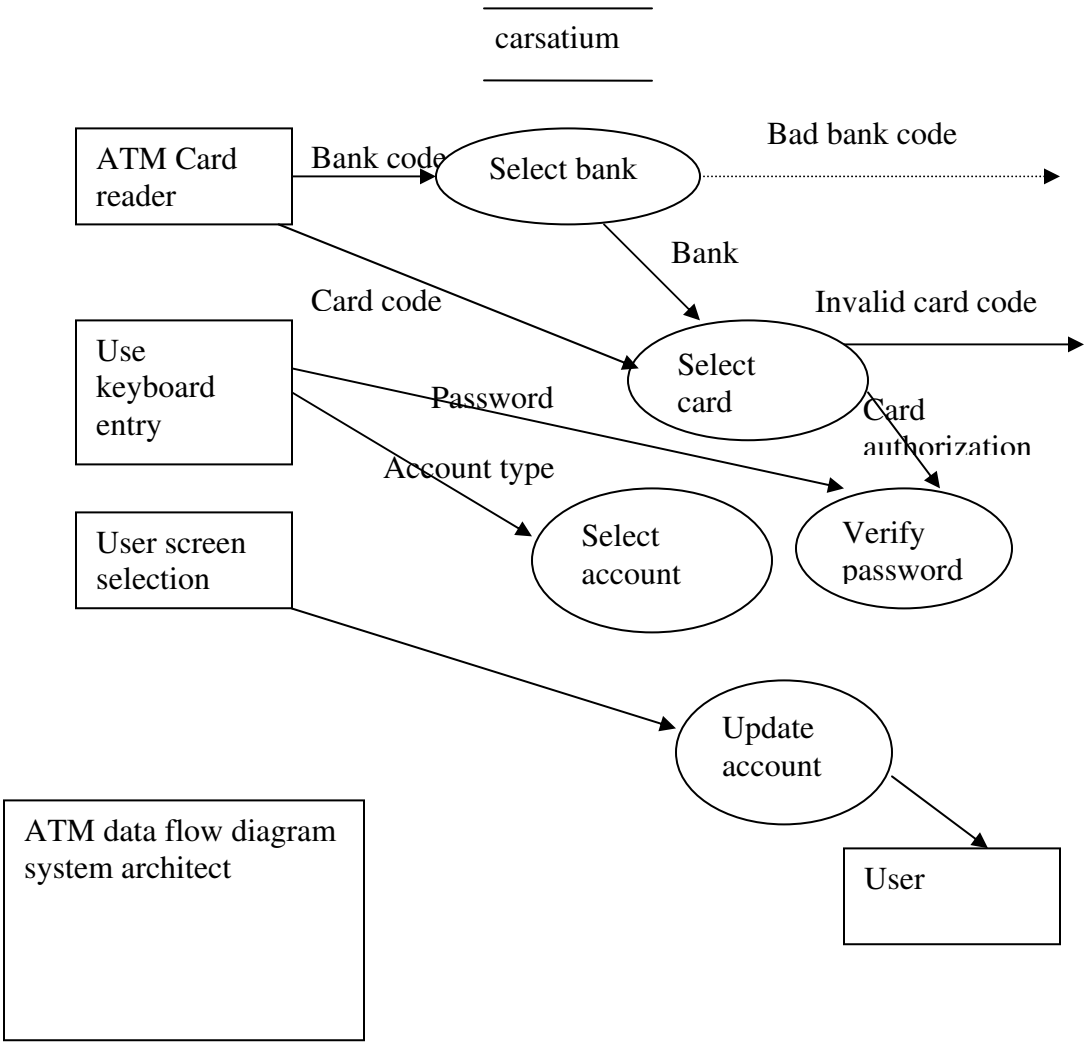
  The data flow shows the direction of data element movement.

- **Data store**:

  The data store is a location where data is stored. (ex: account in ATM)

- **External entity:**

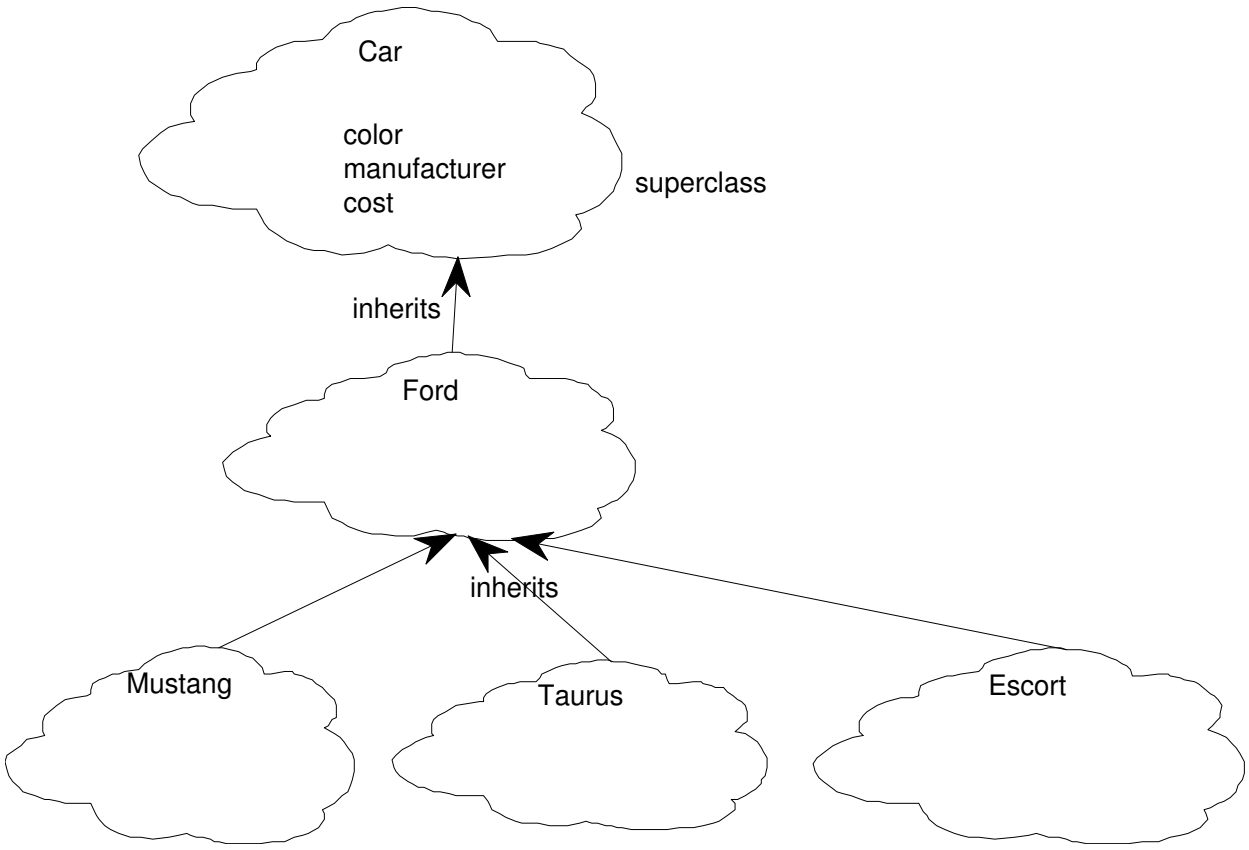
  The external entity is a source or destination of a data element. (ex: ATM card reader)



The Booch Methodologies:

The Booch Methodologies is a widely used object-oriented method. It helps to design our system using object paradigm. It covers the analysis and design phases of an object-oriented system. Booch uses large set of symbols. We will never use all these symbols and diagrams. We start with class and object diagrams in various steps.
The Booch method consists of the following diagrams.

→ Class Diagrams.
→ Object Diagrams.
→ State Transition Diagrams.
→ Module Diagrams
→ Process Diagrams
→ Interaction Diagrams.



The Macro Development Process:

The macro process serves as a controlling frame work for the micro process and can takes weeks (or) even months. The technical management of the system is interested less in the actual object-oriented design than in how well the project corresponds to the requirements set for it and whether it is produced on time.

→ The macro development process consists of the following steps.
→ Conceptualization.
→ Analysis and Development of the Architecture.

→ Design or Create the System Architecture.
→ Evolution or Implementation.

Conceptualization:

* Establish the core requirements of the system.

* Establish a set of goals and develop a problem to prove the concept.

Analysis and Development of the Model:

→ The class diagrams are used to describe the roles and responsibilities of objects to carry out in performing the desired behavior of the system.
→ The object diagrams describe the desired behavior of the system in terms of scenarios.
→ The interaction diagrams are also used to describe the behavior of the system in terms of scenarios.

Design or Create the System Architecture:

* The class diagram is used to decide what classes exist and how they relate to each other.

* The object diagrams decide what mechanisms are used to regulate how objects collaborate.

* The module diagram used to map out where each class and object should be declared.

* The module diagram is used to determine which processor to allocate a process. It also determines the schedule for multiple processes on each relevant processor.

Evolution or Implementation:

Produce a stream of software implementation after a successfully refining the system through many iterations.

Maintenance:

Make localized changes to the system to add new requirements and eliminate bugs.

The Micro Development Process:

Each macro development process has its own micro development process. The micro process is a description of the day to day activities by a single (or) small group of software developers.

The micro development process consists of the following steps.

* Identify classes and objects.
* Identify class and object semantics.
* Identify class and object relationships.
* Identify class and object interface and implementation.

The Jacobson Methodologies:

The Jacobson methodologies cover the entire life cycle and stress trace ability between the different phases, both forward and backward. This trace ability enables reuse of analysis and design work. The heart of their methodologies is the use-case concept, which evolved with objectory (Object Factory for Software Development)

→ Object-oriented Business Engineering (OOBE)

→ Object-Oriented Software Engineering (OOSE)

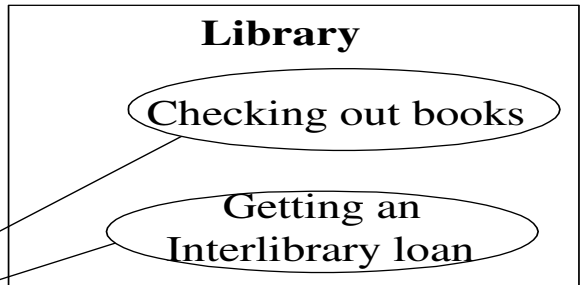
The Booch methodology prescribes a macro development and micro development process.

Use Cases

- Use cases are scenarios for understanding system requirements.
- A use case is an interaction between users and a system.
- The use-case model captures the goal of the user and the responsibility of the system to its users

The use case description must contain:

- How and when the use case begins and ends.
- The interaction between the use case and its actors, including when the interaction occurs and what is exchanged.
- How and when the use case will store data in the system.
- Exceptions to the flow of events.

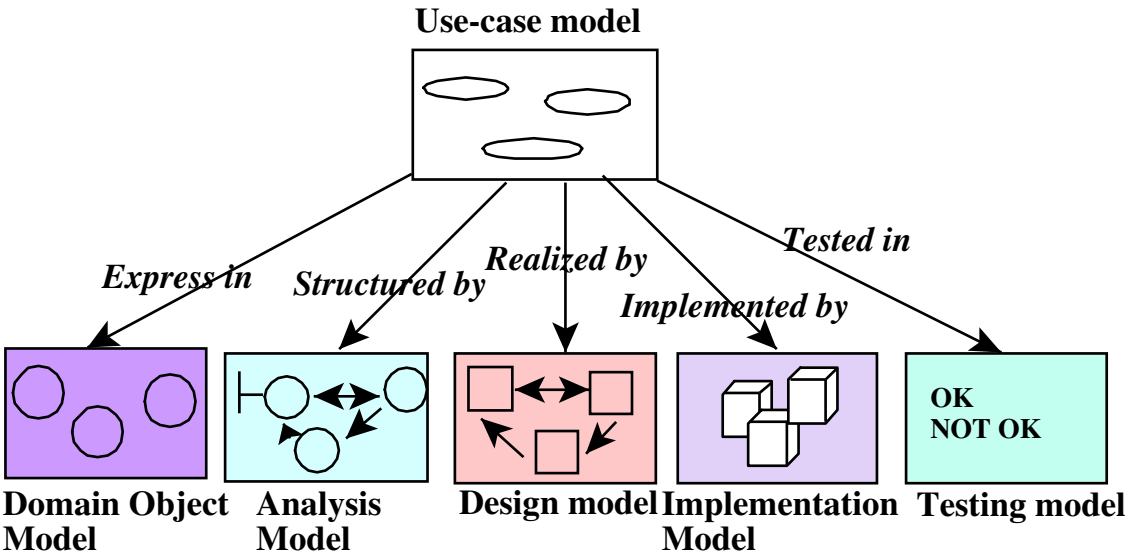


Object-Oriented Software Engineering: Objectory

Object-oriented software engineering (OOSE), also called Objectory, is a method of object-oriented development with the specific aim to fit the development of large, real-time systems.

Objectory is built around several different models:

- Use case model.
- Domain object model.
- Analysis object model.
- Implementation model.
- Test model.



Use case model.

The use case model defines the outside (actors) and inside (use case) of the system behavior.

Domain object model.

The objects of the real world are mapped into the domain object model.

Analysis object model.

The analysis object model presents how the source code should be carried out and written.

Implementation model.

The implementation model represents the implementation of the system.

Test model

The test model constitutes the test plans, specifications, and reports.

Object-Oriented Business Engineering (OOBE)

- Object-oriented business engineering (OOBE) is object modeling at the enterprise level.
- Use cases again are the central vehicle for modeling, providing traceability throughout the software engineering processes.

OOBE consists of:

- Analysis phase
- Design
- Implementation phases and
- Testing phase.

Analysis phase

The analysis phase defines the system to be built in terms of the problem-domain object model, the requirements model, and the analysis model. The analysis process is iterative but the requirements and analysis models should be stable before moving to subsequent models.

Design and Implementation phases

The implementation environment must be identified for the design model. The analysis objects are translated into design objects that fit the current implementation.

Testing phase.

There are several levels of testing and techniques. The levels include unit testing, integration testing, and system testing.

Patterns

A pattern is instructive information that captures the essential structure and insight of a successful family of proven solutions to a recurring problem that arises within a certain context and system of forces.

- The main idea behind using patterns is to provide documentation to help categorize and communicate about solutions to recurring problems.
- The pattern has a name to facilitate discussion and the information it represents.

A good pattern will do the following:

It solves a problem.
Patterns capture solutions, not just abstract principles or strategies.

It is a proven concept.
Patterns capture solutions with a track record, not theories or speculation.

The solution is not obvious.
The best patterns generate a solution to a problem indirectly—a necessary approach for the most difficult problems of design.

It describes a relationship.
Patterns do not just describe modules, but describe deeper system structures and mechanisms.

The pattern has a significant human component.
All software serves human comfort or quality of life; the best patterns explicitly appeal to aesthetics and utility.

Generative Patterns:
Patterns that suggest the way of finding the solution

Non Generative patterns:
They do not suggest instead they give a passive solution. Non Generative patterns cannot be used in the entire situation.

Patterns template

There are different pattern templates are available which will represent a pattern. It is generally agreed that a pattern should contain certain following components.

Name → A meaningful name.

Problem → A statement of the problem that describes its intent.

Context → The preconditions under which the problem and its solution seem to recur and for which the solution is desirable. This tells us the pattern's applicability.

Forces → constraints and conflicts with one another with the goals which we wish to achieve.

Solution → solution makes the pattern come alive.

Examples → sample implementation

Resulting context → describes the post conditions and side effects of the pattern.

Rationale → justifying explanation of steps or rules in the pattern. This tells how the pattern actually works, why it works and why it is good.

Related patterns. → The static and dynamic relationships between these patterns and others with in the same pattern language or system.

Known uses→ The known occurrences of the pattern and its application within existing systems.

Anti patterns

A pattern represents a best practice whereas an anti pattern represents worst practice or a lesson learned.

Anti patterns come in two varieties:

- Those describing a bad solution to a problem that resulted in a bad situation
- Those describing how to get out of a bad situation and how to proceed from there to a good solution.

Capturing Patterns

Guidelines for capturing patterns:

- Focus on practicability.
- Aggressive disregard of originality.
- Nonanonymous review.
- Writers' workshops instead of presentations.
- Careful editing.

Frameworks:

Frameworks are the way of delivering application development patterns to support/share best development practice during application development.

In general framework is a generic solution to a problem that can be applied to all levels of development. Design and Software frameworks and most popular where Design pattern helps on Design phase and software frameworks help in Component Based Development phase.

Framework groups a set of classes which are either concrete or abstract. This group can be sub classed in to a particular application and recomposing the necessary items.

d. Frameworks can be inserted in to a code where a design pattern cannot be inserted. To include a design pattern the implementation of the design pattern is used.

e. Design patterns are instructive information; hence they are less in space where Frameworks are large in size because they contain many design patterns.
f. Frameworks are more particular about the application domain where design patterns are less specified about the application domain.

Note:

Include the details of Microsoft .Net development framework.

Differences Between Design Patterns and Frameworks

- Design patterns are more abstract than frameworks.
- Design patterns are smaller architectural elements than frameworks.
- Design patterns are less specialized than frameworks.

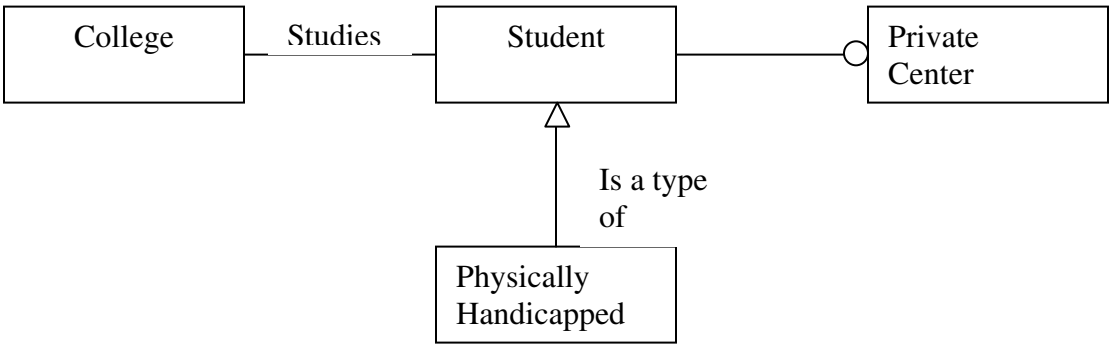
Object Oriented Methodologies

There are different methods for modeling object oriented systems. Each methodology can represent same model with varying documentation style, Modeling language and notations.

1. Rumbaugh's Object Modeling Technique

- The system can be modeled with the help of 3 different models
    1. Object Model
    2. Dynamic Model
    3. Functional Model
- These models are related to different phases as they are the outcome of each phase.
- In analysis phase less detailed [more abstract] representation of object, dynamic and functional model are used.
- In system design phase Architectural diagram is drawn that represents blocks are relations.
- In object design the models generated during analysis phase are refined.
- In implementation phase reusable robust code is generated from the design.

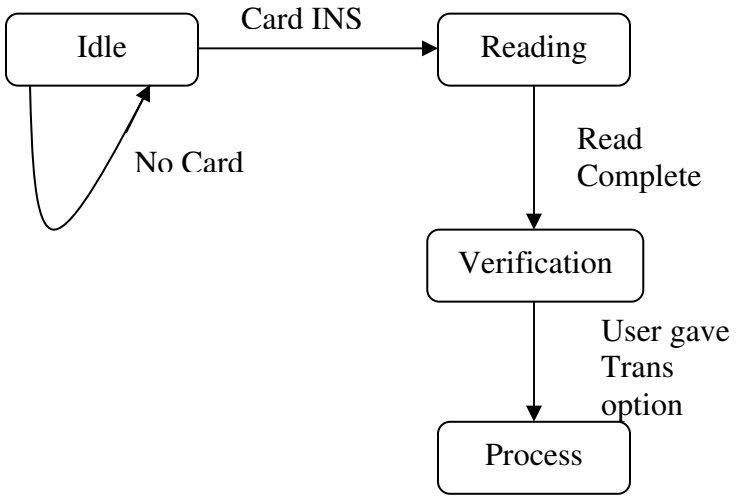
- **Object Model:**
    - It's an object diagram containing interrelated objects.
    - Objects are represented by object notation and it contains Name, Behavior and attributes.
    - Association lines represent the relationship between the objects.
        - One – to – One relationship is one in which an object uses only one object at the other end. [Represented by straight line].
        - One – to – Many relationship is one in which an object at one end uses many objects at the other end. [Represented by bubbled line].
        - Specialized relationship [Inheritance] – is one in which the one object is a type of other object. [Represented by filled triangle].



- In the above example there exist an one to one relationship between student and college as a student studies in a college at a time
- There exists one to much relationship between student and private computer center coz a student can courses in more than one private center.
- Physically handicapped student is a type of student [generalization].

- **Dynamic Model**

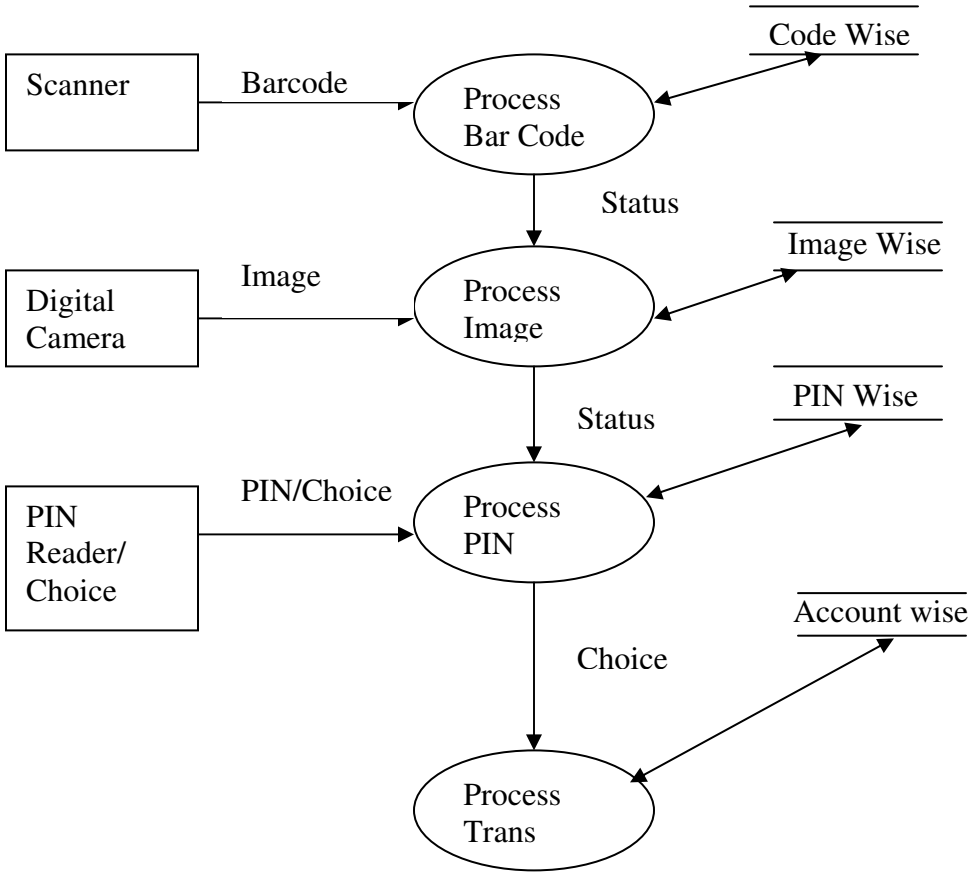
  - Represents a set of states possessed by the system.
  - Interconnected lines represent the transition between the states.
  - The system performs some activity when it is in a state.
  - One or more event may occur in a state and the system may undergo transition from one state to the other state based on the event.
  - State transition can be triggered by an event or completion of an activity.
  - Hence next state depends on the current state and event.



- The above diagram shows the different states of an ATM machine.

- The system remains in idle state until the user inserts card when user inserts card the system goes to Reading state and once it's completed the system goes to verification state and prompt for process choice option.
- The system goes to Process state when the user gives the option for transaction.

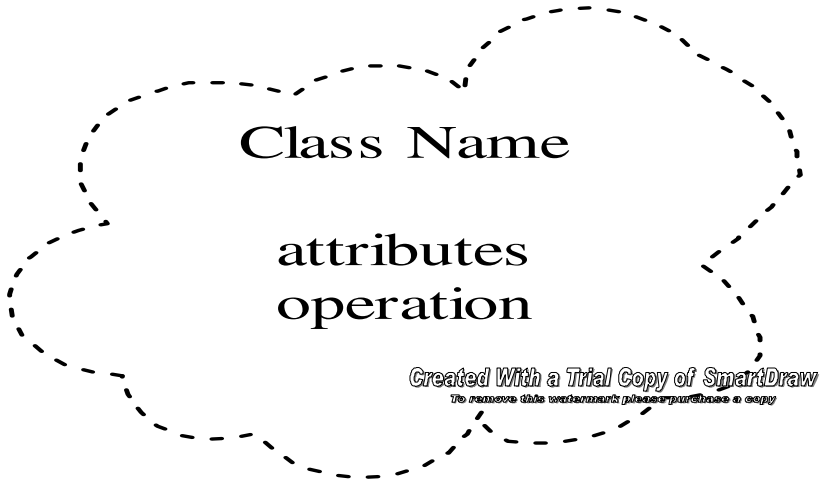
- **Functional Model [Data Flow Diagram]**
  - Represents flow of data between various functional blocks of the system.
  - A functional model may be an external device, process or a Data store.
  - Functional blocks are connected by labeled line that represents flow of data between various functional models.



- The above diagram represents the flow of data between various functional blocks.
- The external devices/entities are represented in a rectangular box
- The Process is represented in a oval shape which performs a particular functionality.
- The Data Store [Information Storage] is represented inside a parallel line.
- Labeled arrows represent direction of flow of data and the date itself.

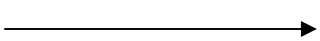
2. The Booch Methodology

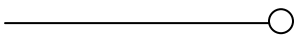
- Booch provides a technique for creating more informative model.
- This approach provides a large set of notations so that a complete model can be built during the analysis and design phase.
- Booch gave equal importance to process [Management Aspect] and diagrams [Technical].
- Diagrams introduced by Booch are
    1. Class Diagram
    2. State Transition Diagram
    3. Module Diagram
    4. Process Diagram
    5. Interaction Diagram
- Booch define process in terms of Macro Process and Micro Process.
- **Diagrams:**
    - **Class Diagram**
        - Represents a set of interrelated classes.
        - This diagram shows the existence of various classes and logical relation between them.

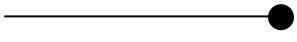


General Notation for a class

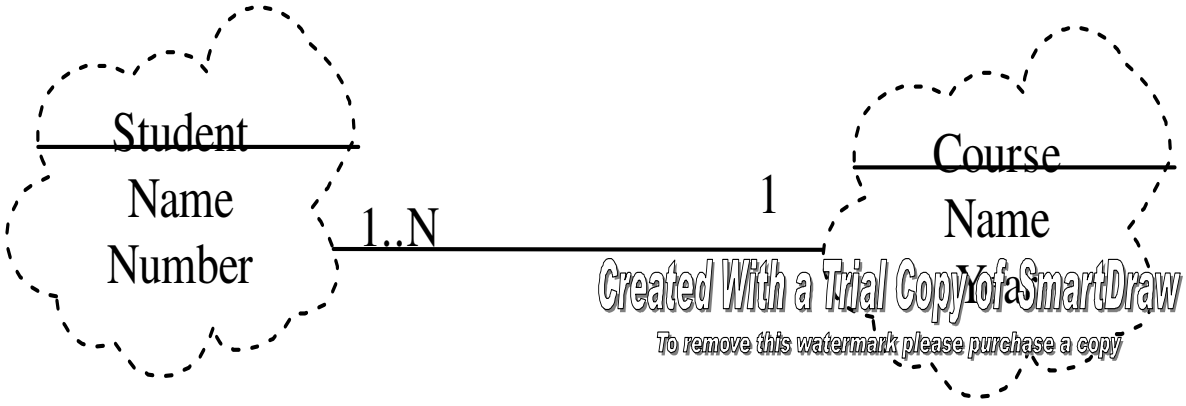
→ Association


→ Inheritance


→ Has


→ using

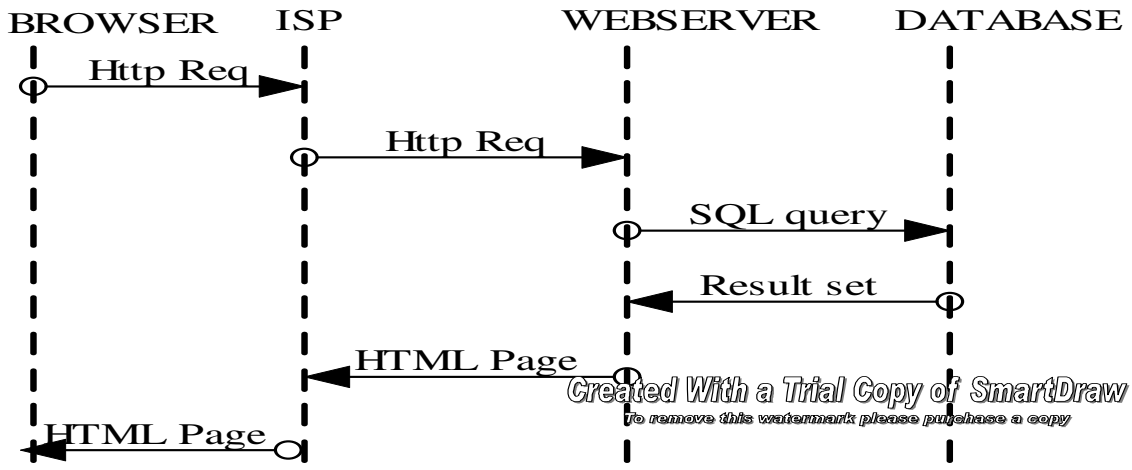
- Association can be quantified with the help of Cardinality.



- The above class diagram represents the association between student class and Course class. The cardinality says that each student can attend one course and each course can have any number of students.

- Note: For 16 marks question convert all the UML diagrams in to Booch diagrams.

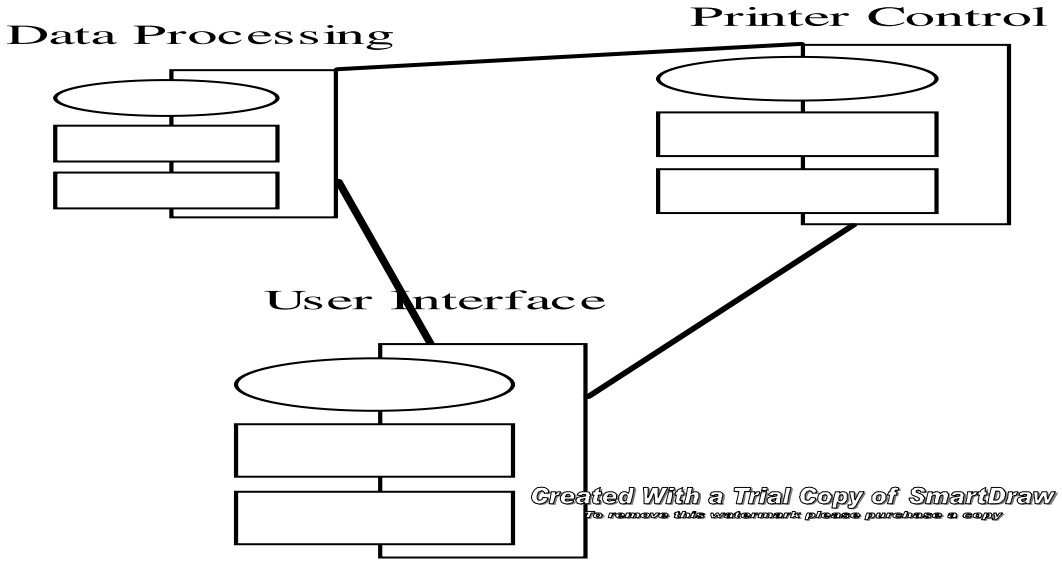
- **Interaction Diagram**
  - An interaction diagram is used to trace the execution scenario in the same context as an object diagram.
  - Interaction diagram includes objects involved in the sequence of communication.
  - The interaction diagram represents the sequence of message passing among related objects.



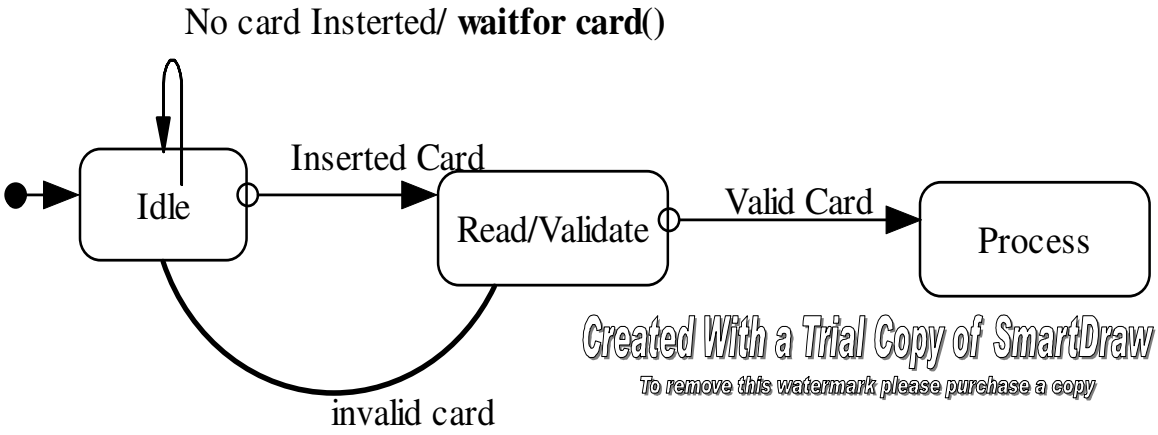
- The above interaction diagram represents the sequence of message passing between various objects.
- [Note: Explain the sequence]

- **Module Diagram**

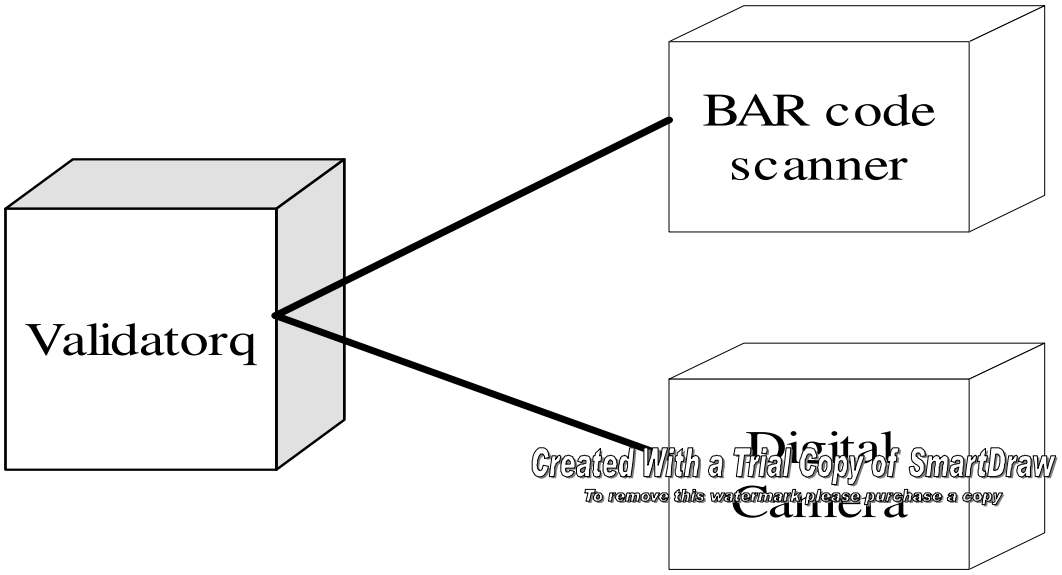
- Module diagram is used to show the allocation of classes and objects to modules in the physical design of the system.
- A single module diagram represents the view of module structure of the system.
- These diagrams are used to indicate the physical layering of the system during architectural design.
- Module diagram contains modules and dependencies.
- Dependencies are represented by straight line.



- **State Transition Diagram**
  - State transition diagrams represent the different states of the machine as whole or different states of an object.
  - The states are named and represented in a state icon.
  - Various transitions between states are represented by directed line with the event and result.
  - This labeled line says that when that particular event occurs the machine undergoes transition from one to another state.



- **Process Diagram**
  - Process Diagram is used to show the allocation of process to processors in the physical design of the system.
  - The process diagram is used to indicate the physical collection of processors and devices.
  - The processor is represented by cube with shaded sides and device is represented by a cube.

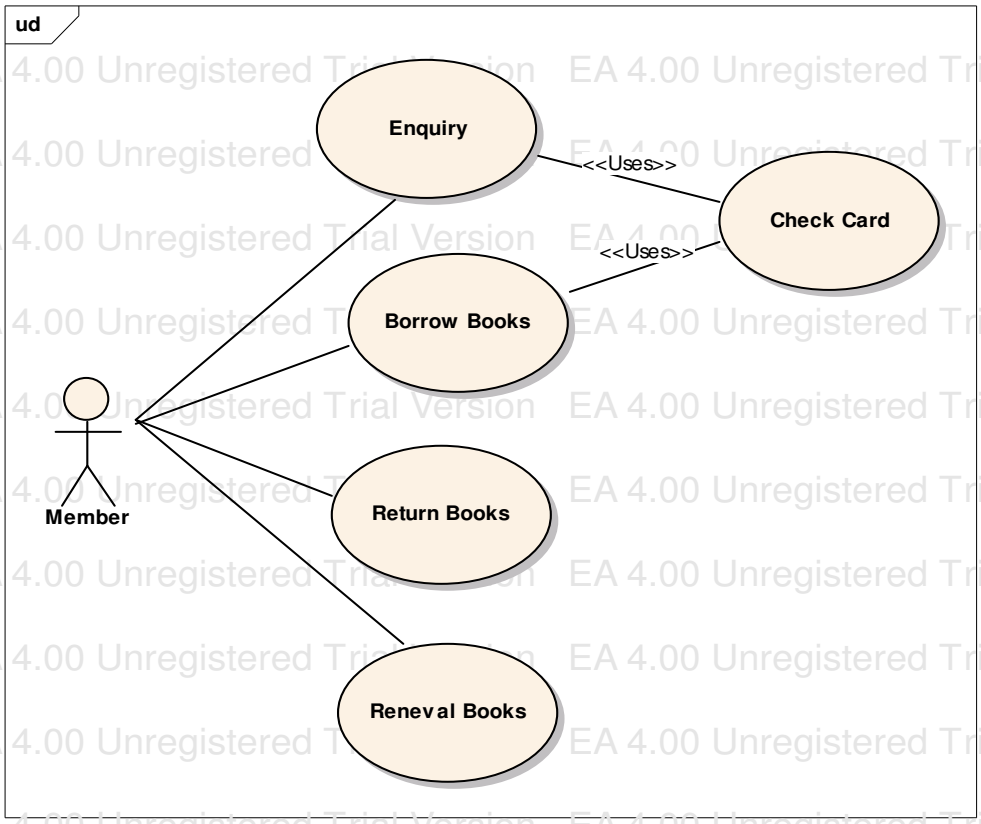


- The Process – Booch gave the concept of Macro process and Micro Process. It highlights the management aspect.
  - The Macro Development Process – Concerns about the management aspect of the system. The entire process is composed of a set of Macro Development Process. The set of Macro process are
    - Conceptualization – core requirement of the system
    - Analysis and Development of the Model – Trace the use of the software, actors, what the system should do.

        - Design and Create System Arch. – Set of interconnected classes with detailed representation of properties and behavior. Various modules that exist from OO decomposition are represented.
        - Evolution or Implementation – After successful iterations of previous steps the representation is implemented in programming languages.
        - Maintenance – It is carried out to make changes needed after the release because of new requirements.
    - The Micro Development Process – It represents the set of minute activities that belongs to a Macro Development Process. In detail the Micro development process represents the minute set of activities carried out by a programmer or group of programmers.
    - The steps involved are
        - Identify classes and objects
        - Identify class and semantics
        - Identify class and relationship
        - Identify class, object interfaces and implementation

2. The Jacobson Methodology

- Jacobson and team came with the concept of Object Oriented Software Engineering and Object Oriented Business Engineering which covers the entire project life cycle.
- Use Case is introduced by this team and later it is adopted in UML.
- Use Case diagram captures the complete requirements of the user and is used in almost all the phases of the software development.
- Use Case represents the interaction between the actor and the system.



- Use Case diagram contains a set of use case where a single use case represents a flow of events.
- There exist <<extends>> and <<uses>> relationships between the various use cases.
- If a use case extends a particular use case the derived case does some functionality more than the base use case.
- The relation between two use cases is said to be <<uses>> if one use case invokes the used one whenever needed.
- In the above example the Member (actor) may borrow books or return books in a library.
- In both the cases the card should be checked and redundancy can be eliminated by establishing uses relationship.
- An use case is said to be **concrete** if that particular case is initiated by the actor where a **abstract** use case is one which is not initiated by the actor.

- Object Oriented Business Engineering is a variant of Object Oriented Software Engineering.
- Various model specified by Jacobson are
    - Use Case Model – Needs of the user.
    - Domain Object Model – mapped real world objects.
    - Analysis Model – represents what should be done and how should be done to the customer to satisfy them
    - Implementation Model – represents the runtime representation of the system.
    - Test Model – represents the test plan, specifications and the report.

UML

(Unified Modeling Language)

Model → Represents an abstract of the system. It is build prior to the original system. It can be used to make a study on the system and also can be used to analyze the effect of changes on the system. Models are used in all disciplines of engineering.

Static Model → Represents the static structure of the system. Static models are stable and they don't change over time.

E.g. Class diagram.

Dynamic Model → Collection of diagrams that represents the behavior of the system over time. It shows the interaction between various objects over time.

E.g. Interaction Diagram.

A model includes

a) Model Elements – Fundamental modeling concepts and semantics.

b) Notation – Visual rendering of model elements.

c) Guidelines – Expression of usage.

Modeling:

Technique of creating models. It is also a good medium of communication between developers at various levels.

Advantages:

- Clarity – Visual representations are mode clear and informative than listed or written documents. Missed out details can be easily found out.
- Familiarity – Similar modeling language and techniques is followed by developers working in same domain.
- Maintenance – Changes can be made easily in visual systems and changes can be confirmed easily.
- Simplification – More complex structures can be represented in an abstract manner to deliver the conceptual idea.

Unified Modeling Language:

It's a language for modeling software systems. This language is used for specifying, visualizing, constructing and documenting software systems through out the development. (Mostly object oriented development).UML is used to model systems build through Unified Approach. Unified Approach combines the methodologies of Booch, Rumbaugh and Jacobson.

Models can be represented at different levels based on the abstraction and refinement. A complete model can be obtained only after continuous refinement of UML diagrams.

UML is composed of 9 graphical diagrams:

1) **Class Diagram**
2) **Use – Case Diagram**
3) **Behavior Diagram**
    a. **Interaction Diagram**
        i. **Sequence Diagram**
        ii. **Collaboration Diagram**
    b. **State Chart Diagram**
    c. **Activity Diagram**
4) **Implementation Diagram**
    a. **Component Diagram**
    b. **Deployment Diagram**

UML Class Diagram:

Class diagram represents the types of objects in the system and the various kinds of static relationships that exist between them. Class diagrams are used in object modeling where real world objects are mapped to logical objects in computer program. Notations and symbols used in Class diagrams are

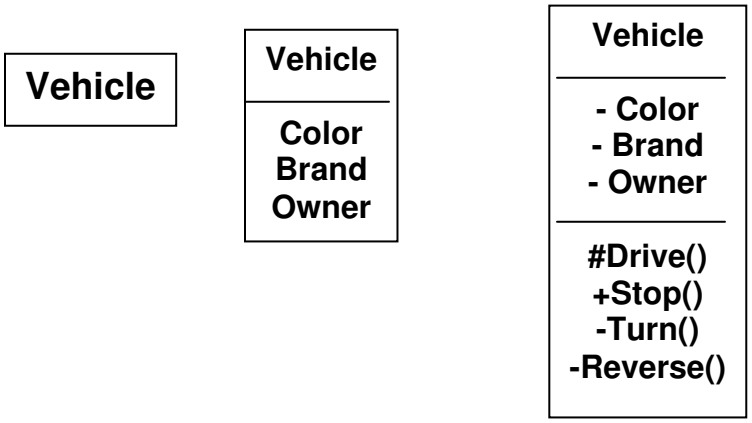
1) ***Class Notation***
2) ***Object Diagram***
3) ***Class Interface Notation***
4) ***Binary Association Notation and Association Role***
5) ***Qualifier***
6) ***Multiplicity***
7) ***OR Association***
8) ***Association Class***
9) ***N – ARY Association***
10) ***Aggregation and Composition***

11) Generalization

1) Class Notation:

Classes are represented in a rectangular box. The top box has the name, the middle one has the attributes/properties/data members and the lower one has the behavior/ member functions/ methods. A Box with a class name represents the most abstract representation of a class.

E.g.



All the above diagrams represent the same class in different levels of abstraction.

Visibility of members can be specified using -, + and # symbols.

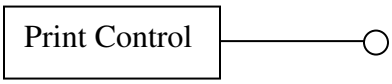
- \- indicates a private member
- \+ indicates a public member
- \# indicates a protected member.

2) Object Diagram:

Object diagram is an instance of a class diagram. It gives a detailed state of a system at a particular point of time. Class diagrams can contain objects and Object diagram cannot contain classes. Hence Class diagram with objects and no classes is called an object diagram.

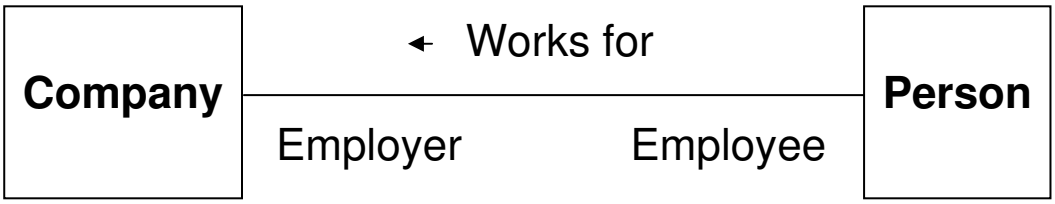
3) Class In

It represents the externally visible behavior of a class. Externally visible behaviors are public members. The notation is small circle with a line connected to a class.



4) Binary Association Notation and association role:

It represents the association between two classes represented by a straight line connecting 2 classes. Association has got a name written on the line and association role

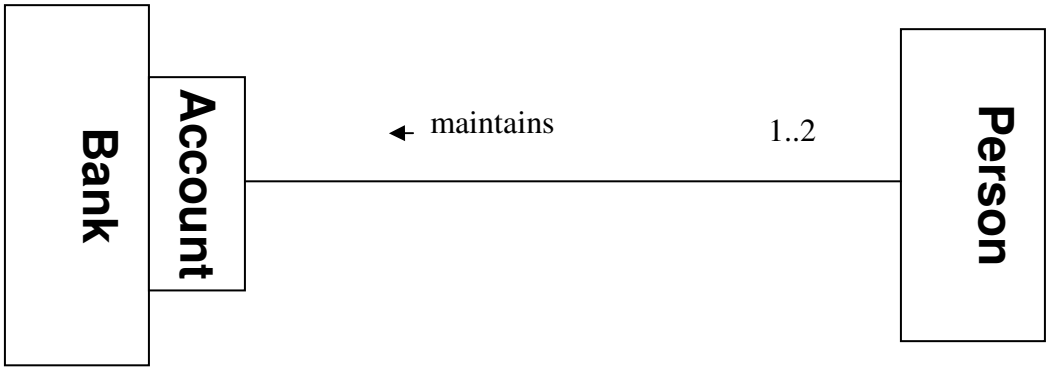


In the above diagram Works for is the association that exist between Person and Company. The arrow mark indicates the direction of association. i.e., The Person Works for the company.

Association Role: it's related to association. Each class that is a member of an association plays a role in the association called association role. E.g. The person plays the role of *EMPLOYEE* in the *WORKS FOR* association where a company plays the role of *EMPLOYER.*

5) Qualifier:

Qualifier is an attribute of an association. It makes the association more clear.



The qualifier here is the *account* and it defines that each instance of account is related to 1 or 2 person. Hence the account qualifies the association maintains.

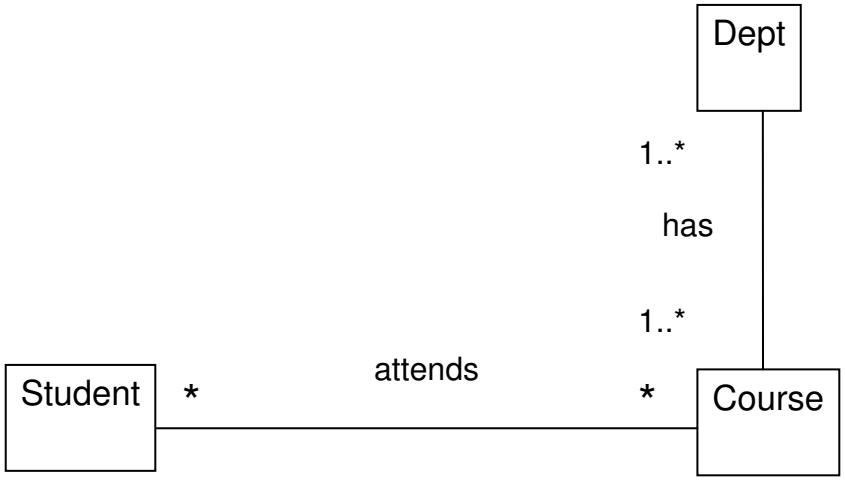
6) Multiplicity:

It gives the range of associated classes. It is specified of the form ***lower bound..upper bound*** or *integer.* Lower bound must be an integer where upper bound can be an integer or a *. * Denotes many.

When a multiplicity is stated at one end it states that each class at the other end can have relation with stated number of classes at the nearer end.

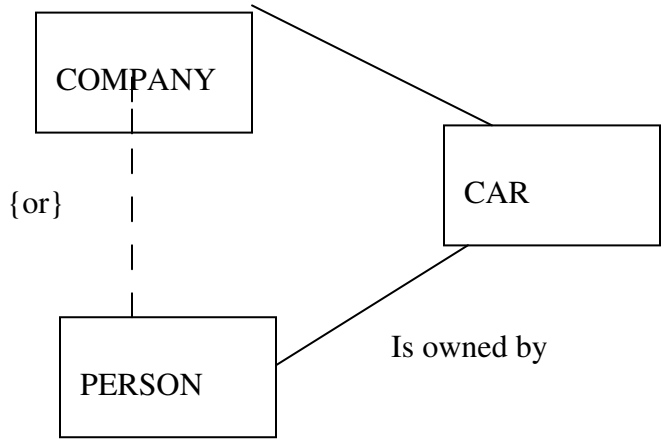
E.g:

The below diagrams says that each course can have any number of students OR each student may attend any number of courses.
Also each department has one or more courses and a course may belong to one or more departments.



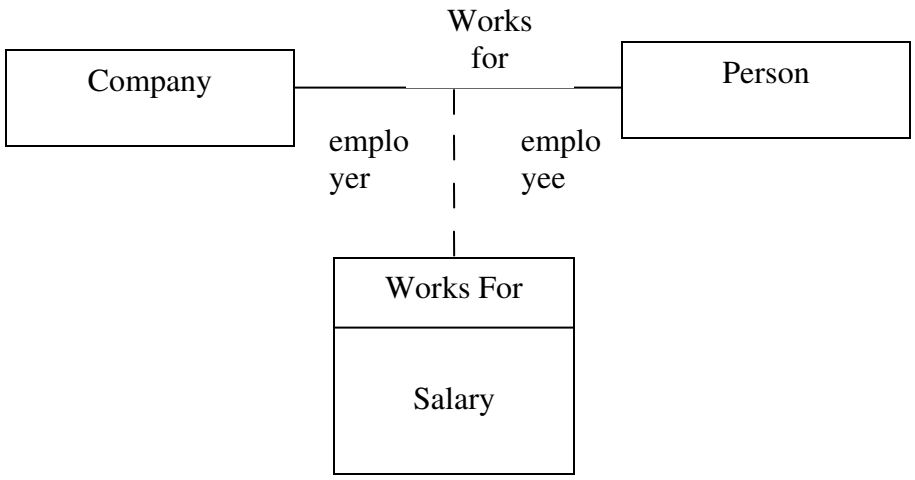
7) OR – Association:

It's a relation in which a class is associated to more than one class and only one association is instantiated at any instance of time for an object. It is represented by a dashed line connecting two associations. A constraint string can be used to label the OR association line.



8) Association Class:

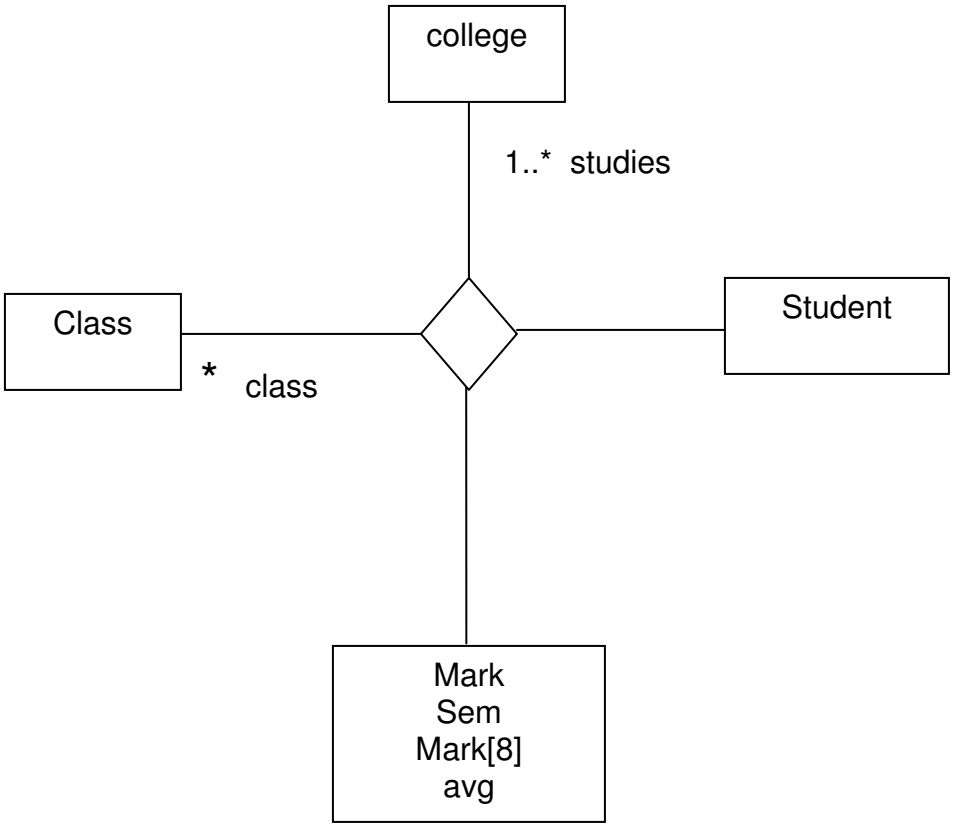
It's an association that has class properties. The association class is atta.ched to an association with a dashed line



Here the works for relation has got one attribute *salary*. Hence an association class is maintained.

9) N – Ary Association:

It's an association where more classes participate. They are connected by a big diamond and the name of the association is named near the line



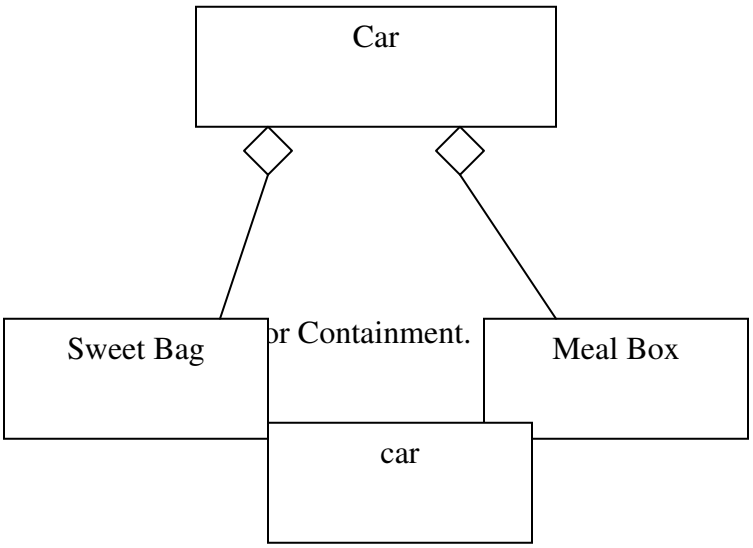
10) Aggregation and Composition:

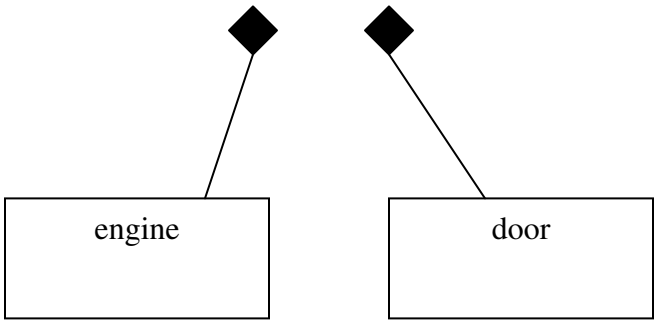
Aggregation is a 'part – of' association. Containment is a type of aggregation with weak ownership where composition is a part of relationship with strong owner ship.

For E.g

A Car consist of Engine, Door, light etc.. [Composition]
A Car contains a Bag [Containment]

Containment is represented by a line with hollow diamond arrow at the end where a Composition is represented by filled diamond at the end.





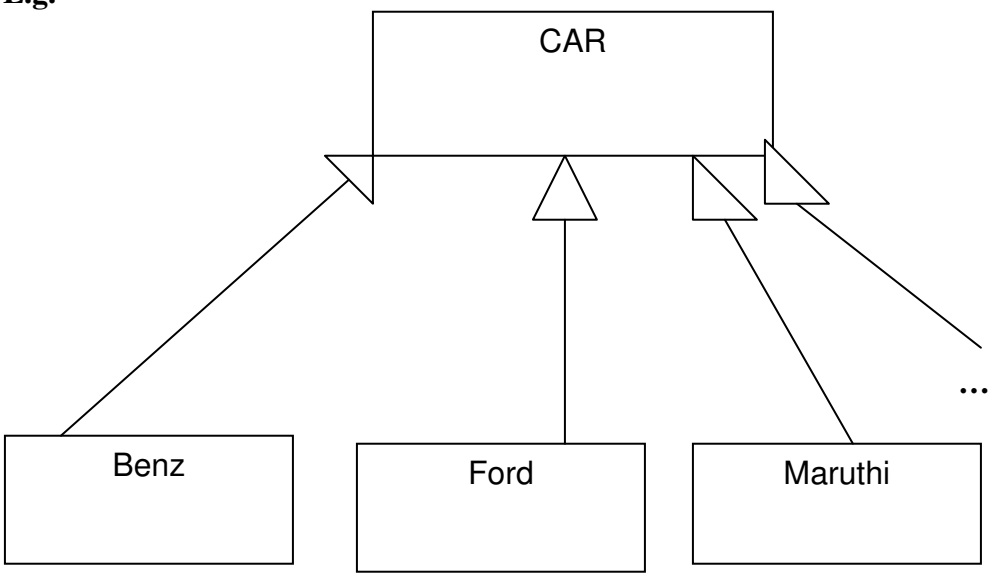
This is an example for Containment.

12) Generalization:

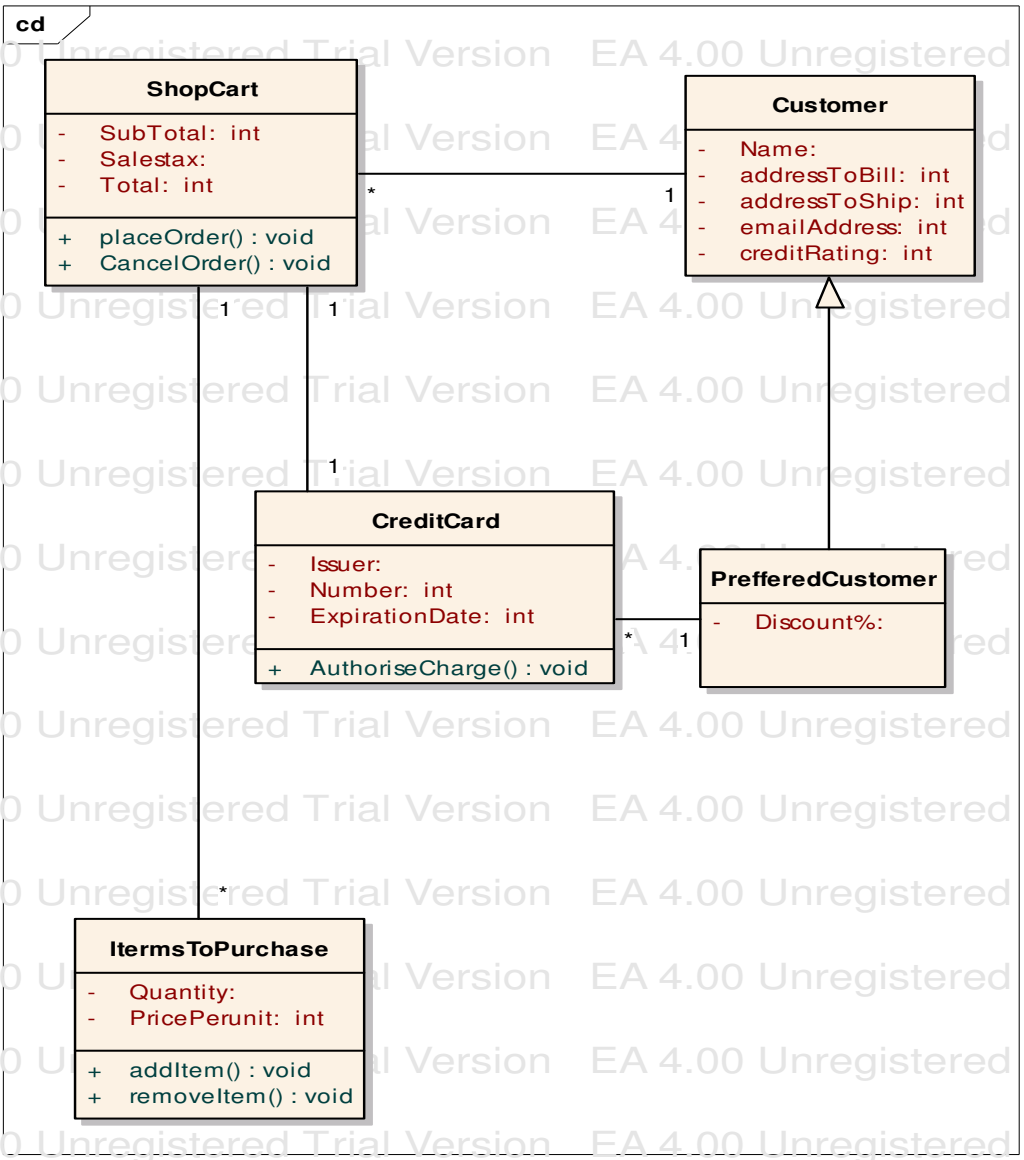
It is a relationship between more general and specific classes. It's represented by a directed line with a hollow arrow head. Some diagrams specify incomplete number of subclasses. It can be represented by ellipses.

… in the example diagram specifies that there are some more sub classes of CAR class are available and they are not mentioned.

E.g.



EXAMPLE FOR A CLASS DIAGRAM:



Note: like this study all the diagrams

UNIT-III

Object Analysis: Classification

Classification:

It's the technique of identifying the class of an object rather than individual objects. In other words it's the process of checking whether the object belong to particular category or not.

Classification Theory:

Many persons introduced many theories for classification.

1. **Booch**: Classification guides us in making good decisions for modularization using the property of sameness. The identified classes can be placed in same module or in different module based on the sameness. Sameness/ Similarity can be measured with coupling and cohesion. Cohesion is the measure of dependency between the classes/packages/components in a module where coupling is the measure of dependency between different modules. In real software development prefers weak coupling and strong cohesion.
2. **Martin and Odell**: Classification can be used for well understanding of concept [Building Blocks].These classes iteratively represent the refinement job during design. Classes also act as an index to various implementations.
3. **Tou and Gonzalez**: Explained classification based on physicophysiology i.e., the relation between the person and the system. When a person is introduced to a system the human intelligence may help him to identify a set of objects and classes which later can be refined.

Classification Approaches:

Many approaches have been introduced for identifying classes in a domain. The most used ones are

1. Noun Phrase Approach
2. Use – Case driven Approach
3. Common Class Approach
4. Class Responsibilities and Collaborators

1. Noun Phrase Approach:

The classes are identified from the *NOUN PHRASES* that exist in the requirements/ use case. The steps involved are

1. Examining the use case/ requirements.
2. Nouns in textual form are selected and considered to be the classes.
3. Plural classes are converted into singular classes.
4. Identified classes are grouped into 3 categories
    a. Irrelevant Classes – They are the unnecessary classes
    b. Relevant Classes – They are the necessary classes
    c. Fuzzy Classes - They are the classes where exist some uncertainty in their existence.
5. Identify candidate classes from above set of classes.

Guidelines for selecting candidate classes from Relevant, Irrelevant and Fuzzy set of classes.

1. Redundant Classes – Never keep two classes that represent similar information and behavior. If there exist more than one name for a similar class

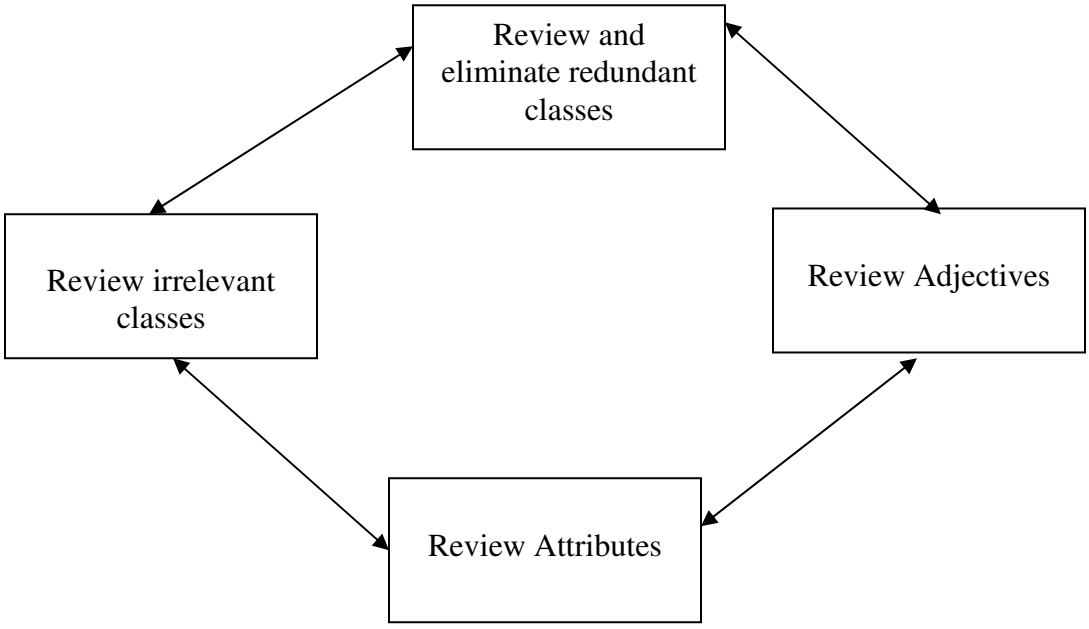
select the more relevant name. Try to use relevant names that are use by the user. ***(E.g. Class Account is better than Class Moneysaving)***

2. Adjective Classes – An adjective qualifies a noun (Class). Specification of adjective may make the object to behave in a different way or may be totally irrelevant. Naming a new class can be decided how far the adjective changes the behavior of the class. ***(E.g. The behavior of Current Account Holder differs from the behavior of Savings Account Holder. Hence they should be named as two different classes. In the other case the toper adjective doesn't make much change in the behavior of the student object. Hence this can be added as a state in the student class and no toper class is named)***

3. Attribute Class – Some classes just represent a particular property of some objects. They should not be made as a class instead they can be added as a property in the class. ***(E.g. No Minimum Balance, Credit limit are not advised to named as a class instead they should be included as an attribute in Account class).***

4. Irrelevant Class – These classes can be identified easily. When a class is identified and named the purpose and a description of the class is stated and those classes with no purpose are identified as irrelevant classes. ***(E.g .Class Fan identified in the domain of attendance management system is irrelevant when u model the system to be implemented. These type of classes can be scraped out.)***

The above steps are iterative and the guidelines can be used at any level of iteration. The cycles of iteration is continues until the identified classes are satisfied by the analyst/ designer.
The iterative Process can be represented as below



[Note: USE EXAMPLE FROM THE CASE STUDY]

2. Common Class Patterns Approach:

A set of classes that are common for all domains are listed and classes are identified based on that category. The set of class category is listed based on the previous knowledge (Past Experience).

The Class Patterns are

1. Concept Class
    - This category represents a set of classes that represent the whole business activity (Concept). The never contains peoples or events. These classes represent the entire concept in an abstract way. (E.g. SavingsBank Class)
2. Events Class
    - These are the category of classes that represent some event at a particular instance of time. Mostly they record some information. (E.g. Transaction Class)
3. Organization Class
    - These are the category of classes that represent a person, group of person, resources and facilities. Mostly a group of organization class has a defined mission (target or task). (E.g. CSEDEPT Class represents a group of employees who belongs to Dept of CSE).
4. People Class
    - This category contains the individuals who play some role in the system or in any association. These people carry out some functions that may be some triggers. People class can be viewed as a subcategory of Organization class. This category again contains 2 subsets
        i. People who use the system (E.g. Data Entry Operator who use the system for entering attendance may not be an employee of the college but a contract staff.)
        ii. People who do not use the system but they play some role in the system. (E.g. Lecturer, Students, Instructor etc)

5. Place Class
    - This category of classes represents physical locations which is needed to record some details or the place itself is recorded in detail. (E.g. Information about BLOCK1 where CSE dept functions).

6. Tangible things and Device Class

    - This category includes tangible objects and devices like sensors involved in the system.

[Note: USE EXAMPLE FROM THE CASE STUDY]

3. Use Case Driven Approach – Identifying Classes

Use case diagram/ Model represents different needs of the user and various actors involved in the domain boundary. Unified Approach recommends identification of objects with

the Use Case model as the base. Since the use case represents the user requirements the objects identified are also relevant and important to the domain.

The activities involved are

1. A particular scenario from an use case model is considered
2. Sequence of activities involved in that particular scenario is modeled.
3. Modeling the sequence diagram needs objects involved in the sequence.
4. Earlier iteration starts with minimum number of objects and it grows.
5. The process is repeated for all scenarios in the use case diagram.
6. The above steps are repeated for all the Use Case diagrams.

[Note: USE EXAMPLE FROM THE CASE STUDY, SPECIFY ATLEAST 2 LEVELS OF ITERATION FOR A SINGLE USE CASE SCENARIO]

3. Classes, Collaborators, Responsibilities – Classification (CRC)

Classes represents group of similar objects.
Responsibilities represent the attributes and methods (responsibilities of the class)
Collaborators represent other objects whose interaction is needed to fulfill the responsibilities of the class/object.

CRC Cards – They are 4" X 6" cards where all the information about the objects is written.

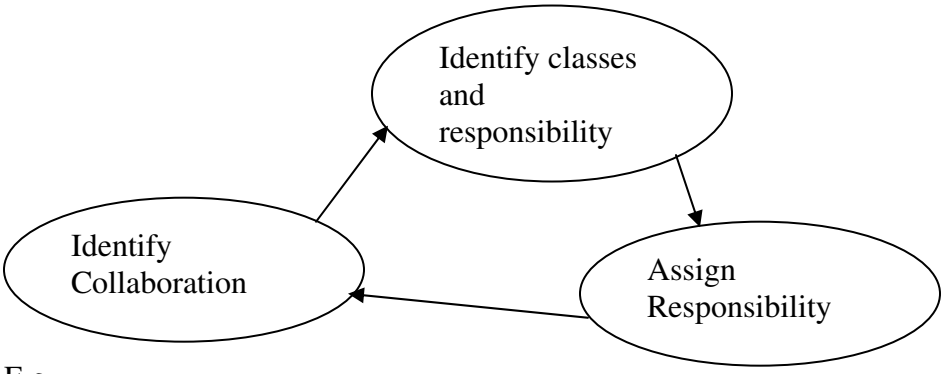
Class Name Responsibilities … ..	Collaborators

The above diagram represents the format of a CRC card. It contains the class name and responsibilities on the L.H.S compartment. Class name on the upper left most corner and responsibilities in bulleted format. Class Name identifies the class and Responsibilities represent the methods and attributes. Collaborators represent the other objects involved to fulfill the responsibility of the object.

This information on the card helps the designer to understand the responsibilities and collaborating classes.

The Process involves

- Identify classes.
- Identify responsibilities.
- Find out the collaborators and need.
- Create CRC card for each class identified.



E.g.

Account	
• Balance • number • withdraw() • deposit() • getBalance()	Current Account (Sub class) Saving Bank (Sub class) Transaction.

The above diagram shows an example of a CRC card representing Account Class.
Its responsibility is to store information like Balance, Number and to have behaviors like withdraw, deposit and getBalance.

Account class has 2 subclasses. They are Current Account and Savings Account. It also collaborates with Transaction class to fulfill the responsibilities.

Guidelines for Naming Classes:

- Use singular Class Name
- Use Comfortable/ relevant names
- Class Name should reflect the responsibility
- Capitalize class names or Capitalize first letter

Identifying Object Relations, Attributes and Methods.

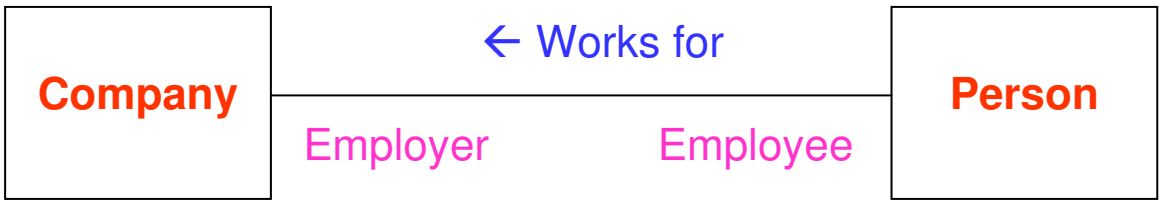
1. Association:-

Association represents a physical or conceptual connection between 2 or more objects. Association is represented by a straight line connecting objects.

Binary association exists between 2 objects/ classes.
Trinary association exists between 3 objects/ classes.
N- Ary association exists between N objects/ classes.

Association names make the association more informative. It is the label attached to the line representing the association.

Association role is the role played the objects involved in the association. It is attached to link representing the association.



The above diagram represents the association between the person and the company. The association says that Person object works for a company object.

Association Name: Works for

In this association Company plays the role of employer and person plays the role of employee.

Steps in identifying associations:

Associations are identified by analyzing the relationship among classes. The dependencies are found out by analyzing the responsibilities of the class.

Answers for the following questions can be used to identify the associations.

1. Is the class capable of doing all its responsibilities?
2. If not what does it need?
3. What are the other class needed to fulfill the requirements?

Some cases the associations are explicit where in other cases they are identified from general knowledge.

Common Association Patterns:

These patterns and associations are group of associations identified by good expertise persons and researchers.

For Example

- Location Pattern – associations of type ***next to, part of, contained in*** that represents association with respect to the physical location. For e.g. tyre is a part of car.
- Communication Pattern – association of type *talk to, order to* that represents associations with respect to communication. For e.g. driver turns the vehicle.

These patterns are maintained in the repository as groups and when a new association is identified it is placed in the relevant group.

Guidelines for eliminating unnecessary associations:

1) Remove implementation associations → separate those associations that represent associations related to implementation. Postpone these issues to later stages of design or initial stages of coding.

2) Eliminate higher order associations by decomposing them into set of binary associations. Higher order associations increase the complexity where binary relations reduce complexity by reducing the confusions and ambiguities.

3) Eliminate derived associations by representing them in simpler associations. For e.g. Grand Parent Association can be represented in terms of two parent relationship. Hence it is enough to deal the parent relationship and no grandparent relationship.

1. Super – Sub Class Relationship (Inheritance):-

Super – Sub class relationship known as generalization hierarchy. New classes can be built from other classes hence the effort for creating new classes gets reduced. The newly built class is called a derived class and the class from which the new class is built is called Base Class. This inheritance allows user to share the attributes and methods.

[NOTE: Give one example]

Guidelines for identifying Super – Subclass relationship

1) Top – Down: - Identify more generalized classes first analyze the purpose and importance of those classes. If necessary identify the specialized classes and represent them. If needed increase the number of levels of generalization/ inheritance.
2) Bottom – Up: - Identify classes and compare them for similar properties and methods. If generalization applies find a new class that can represent the similar classes.
3) Reusability: - Analyze the specialized classes and check weather similar properties/ behavior lie in same layer. If such members exist push them to top most level as possible.
4) Remove multiple inheritance if it creates any ambiguity in the design. Such inheritance

Object Oriented Analysis:

Analysis is a process of transforming a set of facts into a set of complete, unambiguous and consistent picture of requirements of the system must do to fulfill the user's requirement needs.

In this phase developer will analyze how the user will use the system and what should be done to satisfy the user. The analyst may user the following technique

1. Examination of existing system requirements

2. Interviews
3. Questionnaire
4. Observation

Analysis is a difficult process because of the following contents in the SRS

1. Fuzzy descriptions
2. Incomplete requirements
3. Unnecessary features

Business Object Analysis process:

In three tire architecture the business layer actually implements the logic that solves the requirement. Hence analysis done for objects in business layer.
The Object Oriented Analysis Process involves the following steps

1. ***Identify the actors who use the system***
    a. ***Who is using the system? OR***
    b. ***Who will be using the system?***
2. ***Develop the simple business process model using an Activity diagram***
3. ***Develop Use Case***
    a. ***What are the users doing with the system?***
    b. ***What are the users going to do with the system?***
4. ***Prepare Interaction diagram for classification***
5. ***Classification***
    a. ***Identify classes***
    b. ***Identify relationships***
    c. ***Identify attributes***
    d. ***Identify methods***
6. ***Iterate and refine***

1. Identifying Actors:

Actor represents the role of a user with respect to the system. An user may play more than one role. Analyst have to identify what are the roles played by the user and how they use. Actors can be identified using the following questions

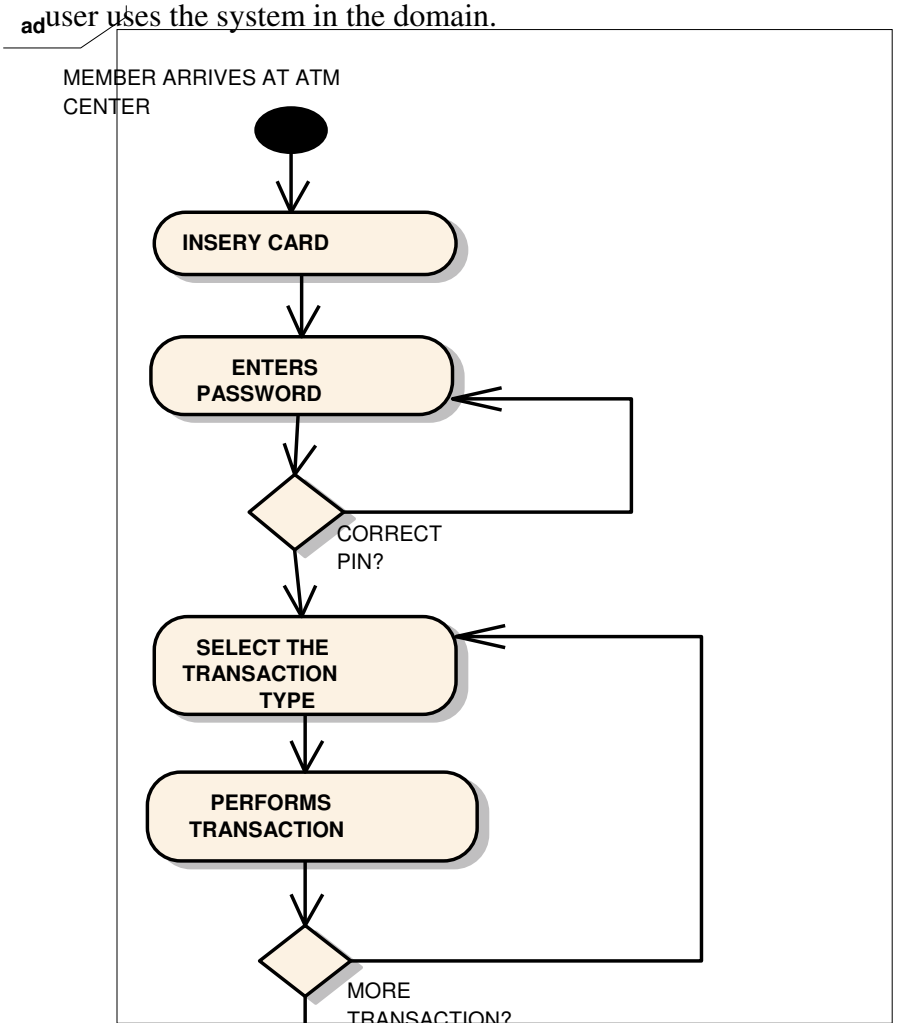
1. Who is using the system?
2. Who is affected by the system?
3. Who affects the system?
4. What are the external systems used to fulfill the task?
5. What problems does this application solve and for whom?
6. How users use the system?

Jacobson provides ***two – three*** rule in identifying actors. i.e. start with 2 or 3 classes (minimum) number of classes and refine on later iterations.

[Note: State the ATM System domain and specify the actors found from the Case Study]

2. Developing a Simple Business Process Model

The entire set of activities that takes place in the domain are represented with the help of a simple Business Process Model created with an UML activity diagram. This makes each member of the team very familiar with the domain and overall activities that takes place when a user uses the system in the domain.



The above diagram represents the Business Process Model for an ATM System. It lists out following activities done by the user in an ATM Center.

1. User enters the ATM Center and inserts the card
2. He Enters the PIN
3. If he enters an incorrect PIN machine ask for correct PIN
4. User selects the type of Transaction
5. System Performs the Transaction
6. User collects the cash, card and leaves the ATM Center.

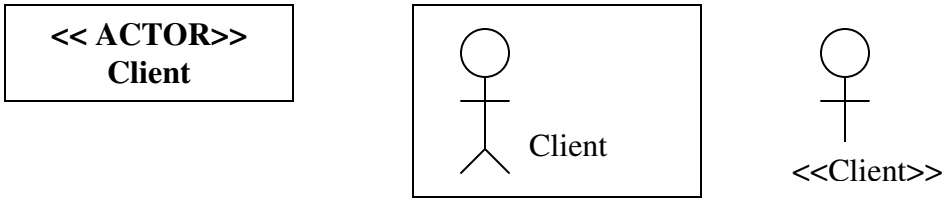
2. Develop use case model.

Use case diagram represents various requirements of the user. This use case model can be used in most of the phases.

It consists of

a. Actors involved
b. Various Scenarios (Use Cases)
c. Communication between actors and use cases
d. Relation between various use cases
    i. USES
    ii. Extends

a. Actors: Actors represents the role played by the user with respect to the system. An user may play more than one role. Actors are represented by any of the three ways in UML



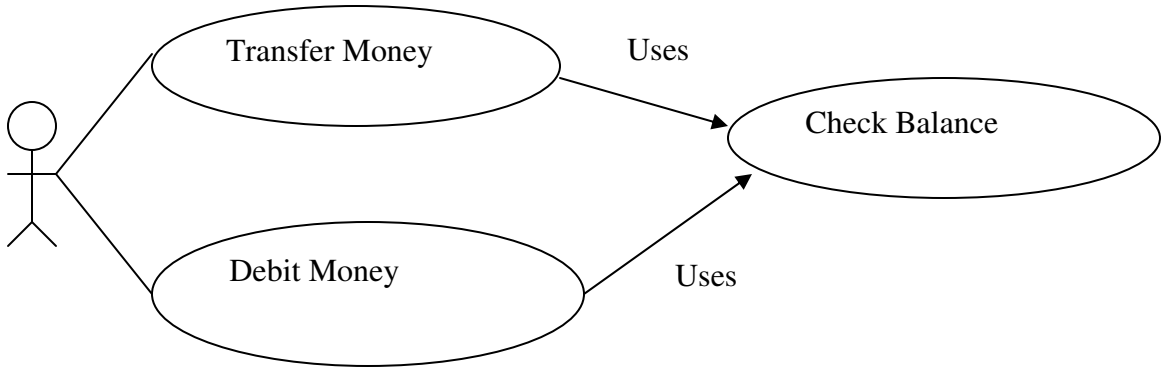
While dealing with the actors importance is given to the role than the people. An actor uses more than one use case.

b. Use Cases: Use Case represents the flow of events/ sequence of activities possible in the system. A use case can be executed by more than one actor. An use case is developed/ named by grouping a set of activities together.

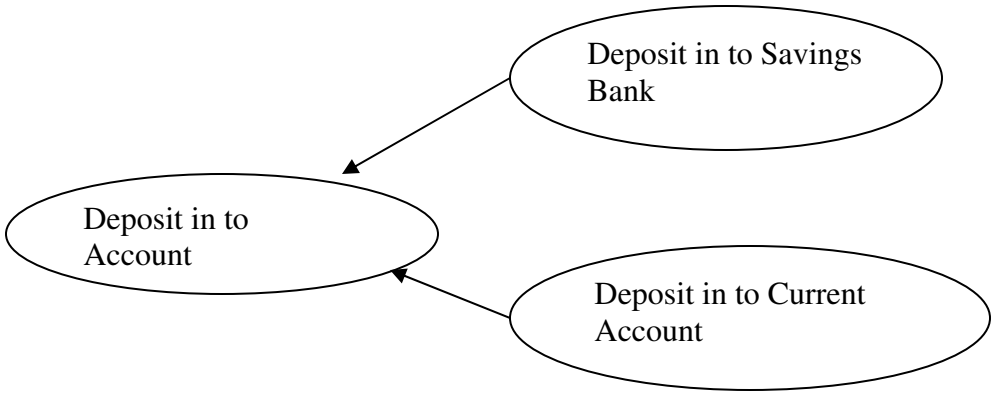
c. Communication between actors and use case. Communication between an actor and use case is represented by a straight line connecting the actor and the use case. The line represents that the actor/ user uses that particular use case. An actor may use more than one use case.

d. Relation between Use Case.

i) Uses – This relationship exist when there is a sub flow between use case. In order to avoid redundancy (created in all the places where there is a sub flow) sub flow is represented by a single Use Case and it can be used by any Use case. This is a way of sharing use cases.



ii) Extends – Extends relation exist between Use Cases if one use case is similar to the other use case but does some more operations. (Note the same relation exist between super – sub class). This relation helps the analyst and the designer to establish relationship between classes and packages that implement the use case.



Abstract and Concrete Use Case:

An Abstract Use Case is one not executed by the user and it is not complete. Abstract Use Case is used by other use case. They can be inherited.

A Concrete Use Case is one executed by the user and is complete.

Guidelines for developing Use Case:

1. Capture simple and normal Use Case first
2. Find out the mistakes and alternate ways of representing the work.

3. Find out the common operations among the use cases and represent them as a specialized use case.

Documentation:

Documentation is the effective way of communication between different developers/ team. This document reduces the gap between different phases/ team. The detailed representation of each work and product developed during various iterations of different phases are written. Document serves as a reference point for future reference.

Guidelines for Developing Effective Documentation:

1. Use common cover
2. Mind 80 – 20 rule while creating and referring documents.
3. Try only familiar terms in document.
4. Make document as short as possible.
5. Represent the document in an organized way.

UNIT-IV

Access Layer Design:

The need of access layer is to design/ create a set of classes that have rights and the way to communicate with the database or data storage place. It isolates following information from the business layer hence it gives service to the business layer.

1) Where data resides?

   Local, Local server, remote server etc.

2) How data resides?

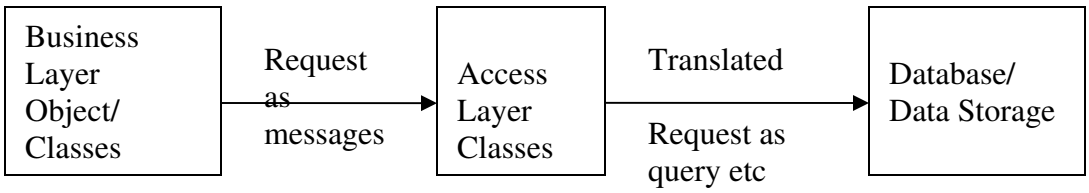
   In a database, in a file, DBMS, RDBMS, ORDBMS, Internet etc.

3) How to access the stored data?

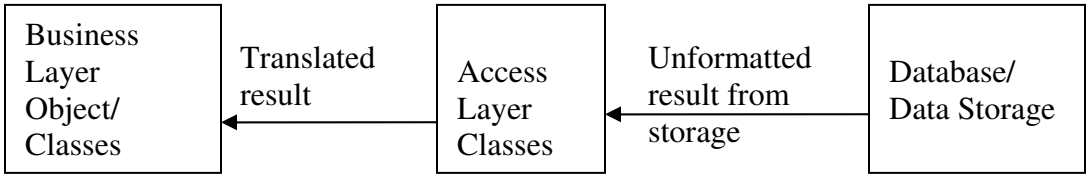
   SQL, stream, File stream, ORB (for DCOM/ EJB) etc.

Access layer provides 2 important service to business layer

Translate Request → The business layer is not aware of the protocol for accessing data as the internal details are known only to the access layer classes. So any request from the business layer for data cannot be transformed to storage as such. Such request are translated in to a form that storage managers can understand and then transformed.



Translate Result → The business layer objects/ classes cannot understand the data send as such from the database/ storage. But the access layer classes can understand the format of result data from the storage as well as the format the business layer can understand. Hence the access layer classes translate the result data in to a form so that business layer can understand.



Persistent Data → Persistent data is one which exists between executions. These data is to be stored permanently for future executions.

E.g. In a student class the name, no, address etc are persistent data.

Transient Data → Transient data is one that may not exist between executions. These data are need not to be stored in database for future execution.

E.g. In a student class the variables meant for temporary purpose like temp_tot etc are transient data

Access Layer Design Sub Process:

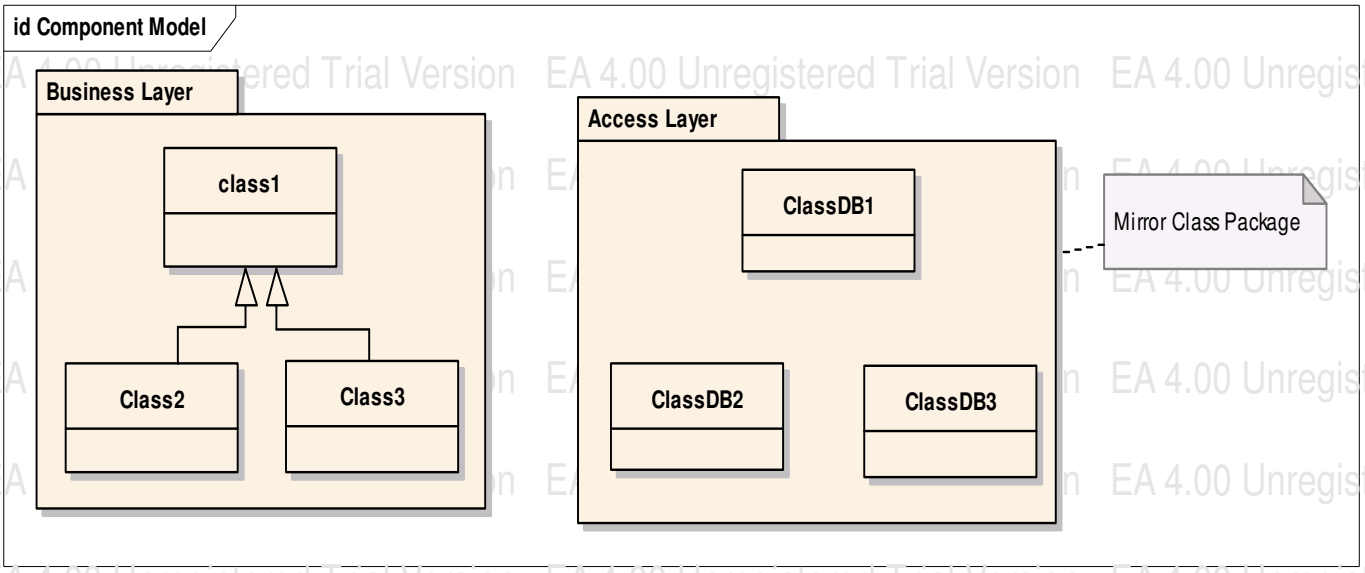
I. ***Design access layer***
   i. ***Create mirror class for all classes identified in business layer which contains persistent data.***
   ii. ***Identify access layer class relationship***
   iii. ***Simplify access layer classes and class relationship***
      1. ***Remove redundant classes***

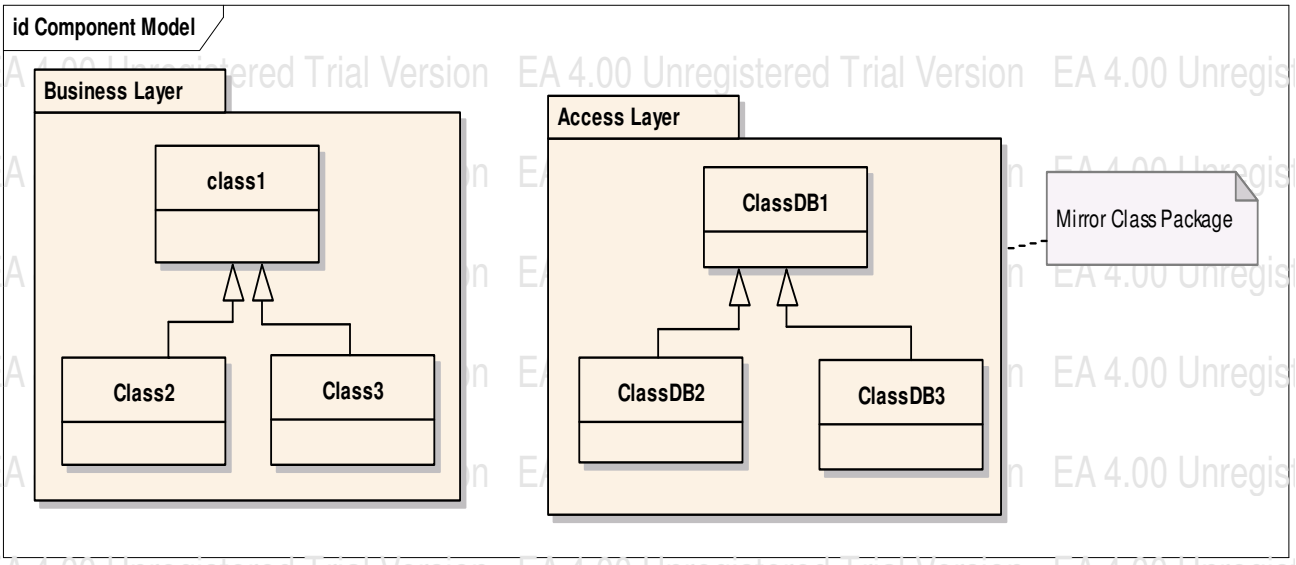
2. ***Singe method classes can be removed and added in another class.***

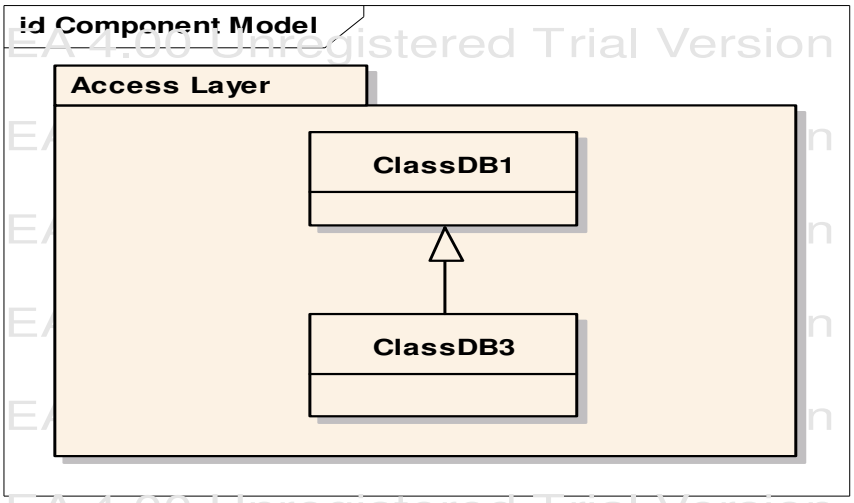
iv. ***Iterate and refine.***

[Note: Explain the process in detail]

The below diagram demonstrates the sub process.
First diagram represents the step mirroring.
The second diagram represents the step of identifying relationship between access layer classes.
The third diagram represents simplified class diagram.







Study of Object Storage Techniques:

DBMS – Database Management System is a set of programs that enables the creation and maintenance of collection of related data. The DBMS and associated programs access, manipulate, protect and manage the data.

Lifetime of objects/ data can be categorized as following

Transient:

1. Transient results to the evaluation of expression.
2. Variables involved in procedure activation
3. Global variables that are dynamically allocated

Persistent:

1. Data that exist between different executions of programs.
2. Data that exist between different versions of programs
3. Data that outlive a program.

Study of DBMS:

DBMS – Database Management System is a set of programs that enables the creation and maintenance of collection of related data. The DBMS and associated programs access, manipulate, protect and manage the data.
DBMS also contains the full definition of the data formats. It is called meta data or schema. Since the complexity and issues regarding the storage lies with in the DBMS it provides a generic storage management system.

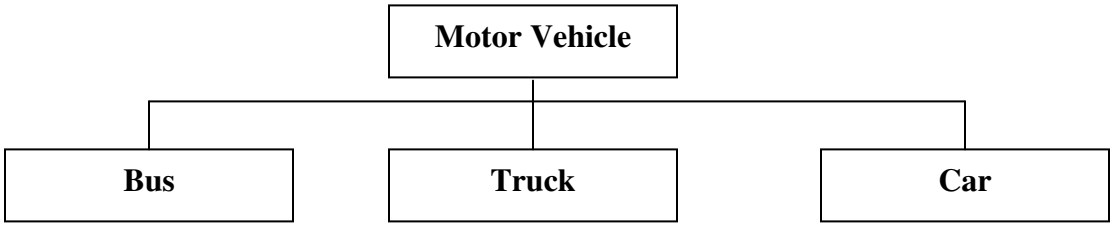
Database Views:

The low level storage details are isolated from the user and for better understanding the logical concepts are supplied to the user.
The various logical concepts are represented by different database views.

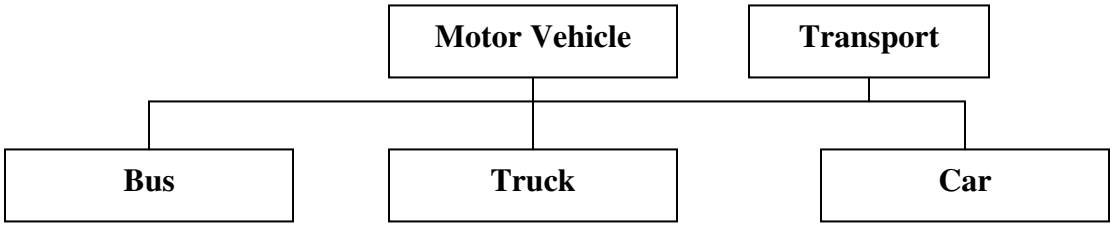
Database Models:

A Persistence – This refers to the life time of an object. Some objects outlive the programs. Persistent Objects are one whose life time is long and transient objects are those whose lifetime is very short.
database model is a collection of logical constructs used to represent the data structure and data relationships with in the database.
The conceptual model represents the logical nature of organization of data where a implementation model represents the real implementation details.

1. Hierarchical Model
   This model represents the data as a single rooted tree structure. Each node represents the data object and connection between various nodes represents the parent – child relationship. This relationship resembles the generalization relationship among objects. A parent node can have any number of child node where each child node shouldn't have more than one parent node.



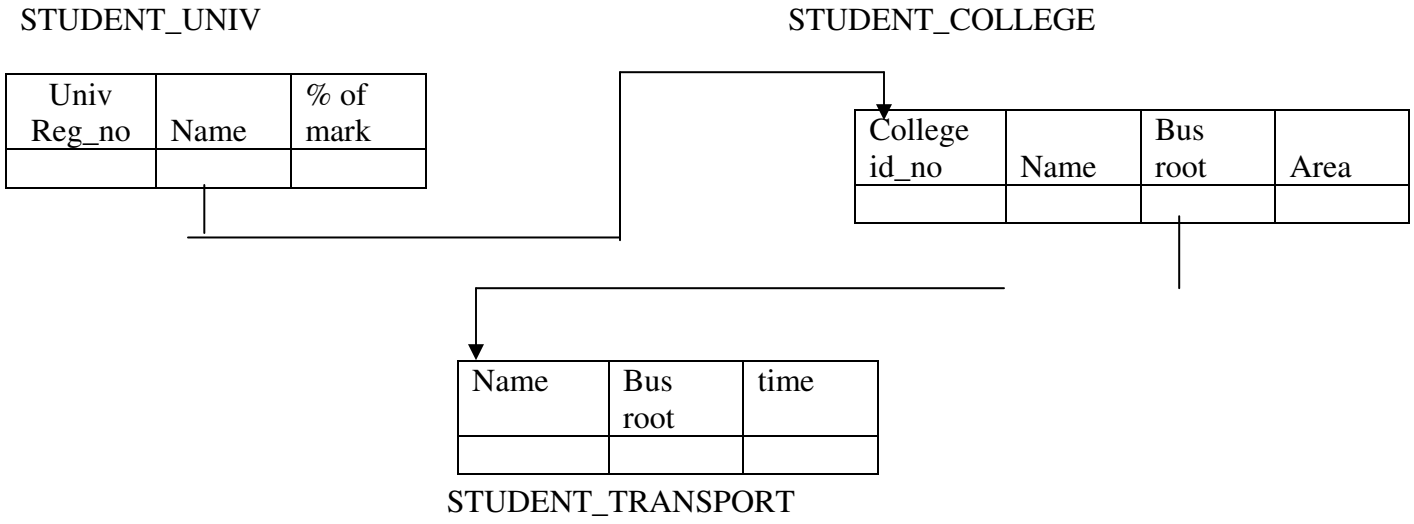
2. Network Model
   A network database model is similar to hierarchical model. Here in this model each parent can have any number of child nodes and each child node can have any number of parent nodes.



3. Relational Model

   This model defines 4 basic concepts. Table, Primary Key, Foreign Key and relation between tables.
   Table – It's a collection of records form the table. The Table is composed of various rows (tuples) and columns (attributes).
   A primary key is a combination of one or more attributes which is used to identify any tuple unambiguously. Primary never gets duplicated in a table.
   Foreign key is an attribute of a table that is a primary key of another table.

Relation between tables – The primary key of one table is the foreign key of another table. Also data can be searched with the combination of more then one table.
Because of these reasons the relational model is the most widely used model.



Database Interface:

The interface of a database includes Data Definition Language (DDL), Data Manipulation Language (DML) and a query.
There are two ways to establish relation with the database

1) By embedding SQL in a program that needs an interface. Since SQL (Structured Query Language) is one of widely accepted language usage of SQL in a program makes programmers feel easy.
2) Extending the programming language to manage data. Here the programmers have to know about the data models and implementation details.

DDL – Data Definition Language is the language used to describe the structure of
Objects (data) stored in a database and relation between them. This structure of information is called Database Schema. DDL is used to create tables in a database.

E.g

CREATE SCHEMA COLLEGE
CREATE DATABASE COLLEGE_DB
CREATE TABLE STUDENT_TRANSPORT (Name char (10) NOT NULL, Busroot number (2) NOT NULL, time TIME)

DML and Queries:

Data Manipulation Language is used for creating, changing and destroying data inside a table. SQL (Structured Query Language) is the standard language for making queries.

A query usually specifies

* The domains of the discourse over which to ask the query.
* The elements of general interest.
* The conditions are constraints that apply.
* The ordering, sorting, or grouping of elements and the constraints that apply to the ordering or grouping.

Traditional DML specifies what are the data desired and specifies how to retrieve the data. Object Oriented DML just specifies what data is desired and not how. While developing applications that uses SQL the mostly used way is to embed the SQL statements inside the program.

Transaction:

A transaction is a unit of change in which many individual modifications are aggregated into a single modification that occurs entirely or not at all. Thus all the changes inside the transactions are done fully or none at all.

A transaction is said to be commit if all the transactions made are successfully updated to the database and said to abort if all the changes made cannot be added to database.

Concurrent Transaction:

A transaction is said to be concurrent if it uses a database which is used by another transactions. Hence a database lock is used to avoid problems like “last updated”. When a transactions starts using a database it is locked and is not available to other transactions.

Distributed Database is one in which a portion of database lies of one node and other on another node.

Client Server Computing.

Client – Node that request for a service
Server – Node that services the request.

Client Server computing is the logical extension of modular programming. The fundamental concept behind the modular programming is decomposing the larger software in to smaller modules for easier development and maintainability.

Client Server computing is developed by extending this concept i.e, modules are allowed to execute in different nodes with different memory spaces. The module that needs and request the service is called a client and the module that gives the service is called a server.
The network operating system is the back bones of this client sever computing. It provides services such as routing, distribution, messages, filing and printing and network management. This Network Operating System (NOS) is called middleware.

Client Program:

- It sends a message to the server requesting a service (task done by server).
- Manages User Interface portion of the application.
- Performs validation of data input by the user.
- Performs business logic execution (in case of 2 tier).
- Manages local resources.
- Mostly client programs are GUI.

Server Program:

- Fulfills the task requested by the client.
- Executes database retrieval and updation as requested by the client.
- Manages data integrity and dispatches results to the client.
- Some cases a server performs file sharing as well as application services.
- Uses power full processors and huge storage devices.

File Server – Manages sharing of files or file records. Client sends a message to the file server requesting a file or file record. The File Server checks the integrity and availability of file/record.

Data Base Servers – Client pass the SQL query in the form of messages to the server in turn server performs the query and dispatches the result.

Transaction Servers – Client sends message to the server for a transaction (set of SQL statements) where the transaction succeeds or fails entirely.

Application Servers – Application servers need not to be database centric. They may Serve any of user needs such as sending mails, regulating download.

Characteristics of Client Server Computing:

1. A combination of client/ front end process that interacts with the user and server/ backend process that interacts with the shared resources.
2. The front end and back end task have different computing resource requirements.
3. The hardware platform and operating system need not be the same.
4. Client and Server communicate through standard well defined Application Program Interface (API).
5. They are scalable.

Distributed and cooperative processing

In Distributed Computing the applications and business logic are distributed across multiple processing platforms. It implies that the processing should be carried out in different process for the transaction to be completed. These processes may not run at same time. Proper synchronization mechanism is provided if needed.

Cooperative processing is a type of distributed computing where more then one process is to be completed for completing the entire transaction. These processes are executed concurrently on different machines and good synchronization and inter process mechanism is provided.

Distributes Object Computing offers more flexible way of distributed computing where mobile software components (objects) travel around the network and get executed in different platforms. They communicate with application wrappers and manage the resources they control. In this computing the entire system is made up of users ,objects and methods.

Various DOC standards are OMG's CORBA, OpenDoc, Microsoft ActiveX/ DCOM.

Object Relation Mapping.

In a relational database system the data are stored in the form of tables where each table contains a set of attributes/fields and tuple/rows. In an object oriented environment the counterpart of class is a table.
In the mapping the classes are mapped to table such a way that the persistent data members of classes will become the attributes. Each row in the table corresponds to an object.
The following mapping types are used in object oriented environment

1. Table – Class Mapping
2. Table – Multiple Class Mapping
3. Table – Inherited Class Mapping
4. Tables – Inherited Class Mapping

1. **Table Class Mapping**

It's a simple one – to – one mapping of a class to a table and properties of class are become the fields. Each row in the table represents an object and column represents a property of objects.

CAR TABLE

Cost	Color	Make	Model

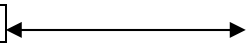
CAR
Cost Color Make Model

2. **Table – Multiple Classes Mapping**

Here a single table is mapped to more than one non inherited classes. So all the persistent properties of mapped classes represents the columns of the table. The column value that is not common for the mapped classes can be used to identify the instance.

In the below example the Employee Class and Customer Class are mapped to person table. Instances of employee class can be identified from the rows whose custID value is NULL. Also instances of Customer class can be identified from the rows whose empID is NULL.

Name	Address	custID	empID



Employee
Name Address empID

Mr. X	A1, a2	C123	NULL
Mr.Y	A2, A3	NULL	E123

Employee
Name Address empID

3. **Table Inherited Classes Mapping**

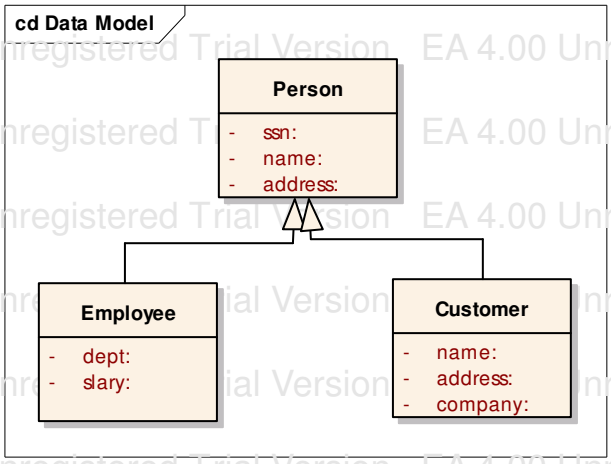
In this case a single table is mapped to more than one class which has a common super class. This mapping allows user to represent the instances of super class and subclasses in a single table.

In the given example the instances of Employee class can be identified from the rows whose wage and salary are NULL. The instances of Hourly Employee can be identified from the rows whose salary is NULL. The instances of Salaries Employee can be identified from the rows with Wage as NULL.

4. **Multiple Tables – Inherited Classes Mapping**.

This kind of mapping allows ***is a*** to be established among tables. In a relational database this is possible by using primary key and foreign key.

In the below example Employee and Customer inherits the properties of Person class. The Person table is mapped to Person class, Employee table is mapped to Employee class and Customer table is mapped to Customer Class. There exist is a relation between Employee, person and customer, person.

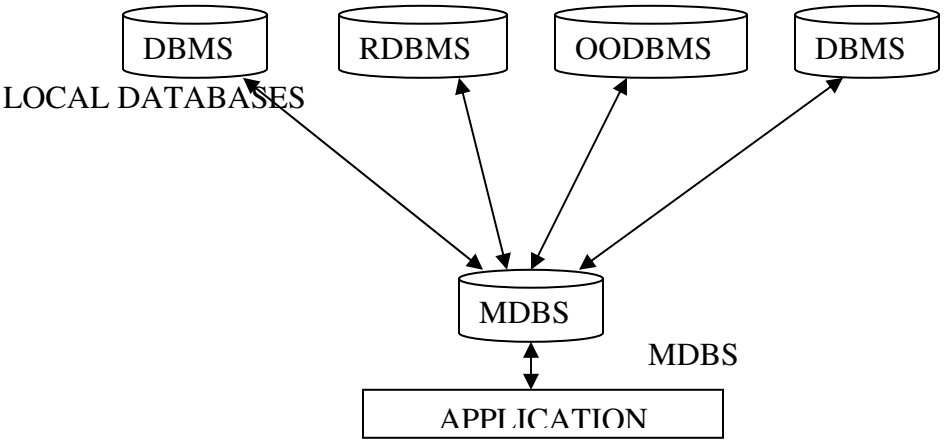


Name	Address	SSN

Name	Dept	SSN	Salary

Name	Address	Company

Study of Multi Database System and Open Database Connectivity

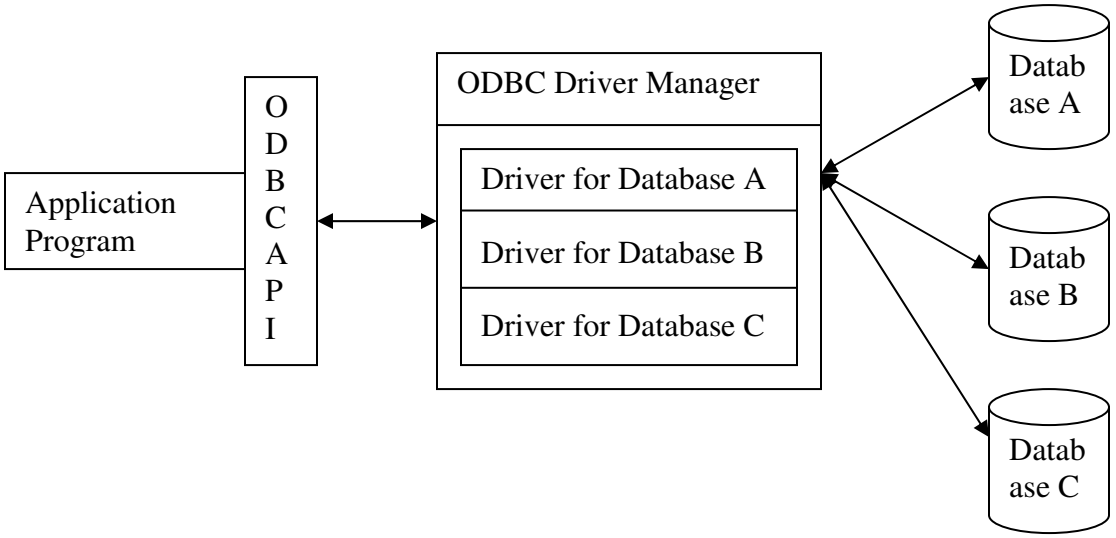


Multi database system is a heterogeneous data base system facilitate the integration of heterogeneous database systems and other information sources. Federated multi database systems are one that are unstructured or semi unstructured.

This multi database system provides single database illusion to the users. The user initiate a single transaction that in turn uses many heterogeneous databases. Hence the user performs updation and queries only to a single schema. This schema is called the global schema and it integrates schemata of local databases. Neutralization solves the schemata conflicts.

The query and updates given to global schema by the user is decomposed and dispatched to local databases. The local databases are managed by gateways as one gate way for each local database.

Open Database Connectivity (ODBC) is an API (Application Program Interface) that provides database access to application programs. The application programs can communicate with the database through function calls (message passing) regardless of the type and location of the database.



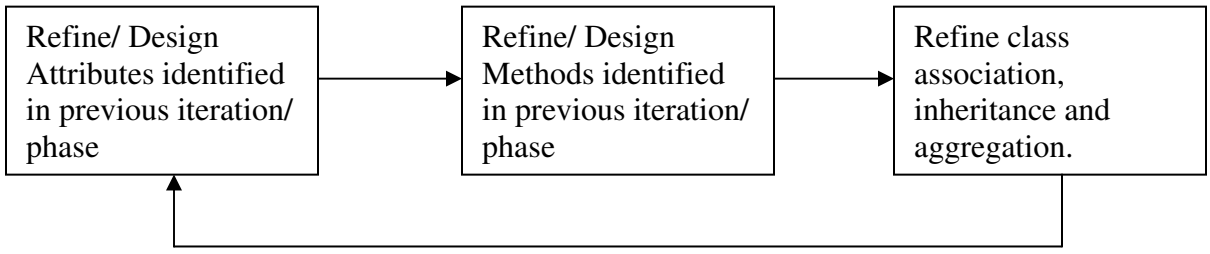
The above diagram shows the logical view of Virtual Database using ODBC. The application program uses ODBC API to communicate with the database. Application programs passes same messages to interface irrespective of the type and location of database. ODBC maintains a set of drivers necessary for communicating with the database. This reduces the database related complexities for a programmer.
The driver manager loads and unloads drivers, performs status checking, manages multiple connection and heterogeneous databases.

Design of Business Layer

Business layer involves lot of logic that is to be implemented in order to achieve the customer needs. Analysis is carried out for business layer objects. Hence the design for business layer has got a strong dependency with the model produced in the analysis phase.

The activities involved in Business Layer design are

2. ***Business Layer Class Design – apply design axioms for designing classes for business layer. Designing classes includes designing their attributes, methods and relationships.***
    I. ***Design/ Refine UML Class diagram developed in previous phase/ iteration.***
        i. ***Design/ Refine attributes (Use OCL)***
            1. ***Add left out attributes***
            2. ***Specify visibility, data type and initial value if any for attributes***
        ii. ***Design/ Refine Methods (Use OCL and UML Activity diagram)***
            1. ***Add left out methods***
            2. ***Specify visibility of the protocol (+, - ,#)***
            3. ***Specify the argument list and return type***
            4. ***Design the method body using UML Activity diagram and OCL.***
        iii. ***Refine association***
        iv. ***Refine Generalization and aggregation.***
        v. ***Iterate and refine.***



Refining Attributes:

Attributes represents the information maintained by each object. Complete list of attributes should be identified in order to maintain a complete set of information. Detailed information of attributes is not specified in analysis phase but in design phase.

OCL is used to represent the attribute details inside a class diagram/ notation.

Various types of attributes are

1) Single valued attributes – an attribute represents one value at a time.
2) Multi valued attributes – an attribute can store more than one value
3) Reference attributes – an attribute refers (alias) another instance.

OCL format for representing attributes:

The OCL specification for specifying attributes is

Visibility attribute name : type

OR

Visibility attribute name : type = initial value

E.g.

+ Name : String
    - represents a public attribute Name of type String

Name : String = "Hello"
    - represents a protected attribute Name of type String with initial value Hello

[Note: Use the example from the case study]

Designing/ Refining Methods:

Designing methods involves design of Method protocol and Method body. A method protocol defines the rule for message passing to this particular object where the method body gives the implementation details. The types of methods provided by class are

1) Constructor – Method that is responsible for creating objects/ Method invoked during instantiating.
2) Destructor – The method that destroys instances/ Method invoked when an object is freed from memory.
3) Conversion Method – Methods responsible for converting one form of date to other form.
4) Copy Method – Methods responsible for copying information.
5) Attribute Set – Method responsible for setting values in attributes
6) Attribute Ger – Method responsible for getting the values from an attribute
7) I/O Methods – Method responsible for getting and sending data from a device

8) Domain Specific – Those methods responsible for some functionality in a particular domain.

Designing Protocol:

Protocol gives the rule for message passing between objects. Protocol is the interface provided by the object. Based on the visibility of the protocol it can be classified into

1. Public Protocol
2. Private Protocol
3. Protected Protocol.

Private protocols specify messages that can be send only by the methods inside the class. They are visible only inside the class.

Protected protocols specify messages that can be send only by the methods inside the class. But they can be inherited by the subclass.

Public protocols specify messages that can be send by the methods with in the class as well as objects outside the class.

Protocol and Encapsulation leakage – If protocols aren't well designed unnecessary messages are made available outside the class results in encapsulation leakage.

Internal layer contains the private and protected protocols where an External layer contains public protocols.

OCL Specification for Protocol Design:

The specification is

Visibility protocol name (argument list) : return type

Where argument list is *arg1: type, arg2: type, arg3: type … argn: type*

E.g.

+ getName () : String

It's a public protocol named getName with no parameters and it returns a value of type String.

- setData (name : String, no : Integer) : Boolean

It is a private protocol that accepts 2 arguments one of type String and other of type Integer. It returns a value of type Boolean.

Designing Method body:

UML Activity diagram along with OCL specification can be used for representing the body of the method. Activity diagram representing the method body says how the work should be done.

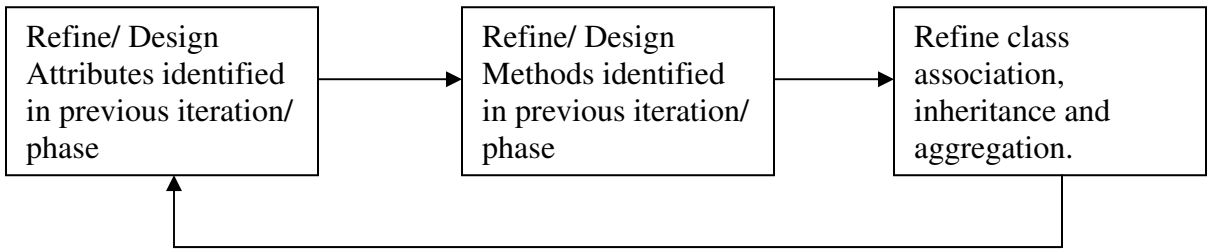
[Note: Use an activity diagram from case study]

Design of Business Layer

Business layer involves lot of logic that is to be implemented in order to achieve the customer needs. Analysis is carried out for business layer objects. Hence the design for business layer has got a strong dependency with the model produced in the analysis phase.

The activities involved in Business Layer design are

3. ***Business Layer Class Design – apply design axioms for designing classes for business layer. Designing classes includes designing their attributes, methods and relationships.***
    I. ***Design/ Refine UML Class diagram developed in previous phase/ iteration.***
        i. ***Design/ Refine attributes (Use OCL)***
            1. ***Add left out attributes***
            2. ***Specify visibility, data type and initial value if any for attributes***
        ii. ***Design/ Refine Methods (Use OCL and UML Activity diagram)***
            1. ***Add left out methods***
            2. ***Specify visibility of the protocol (+, - ,#)***
            3. ***Specify the argument list and return type***
            4. ***Design the method body using UML Activity diagram and OCL.***
        iii. ***Refine association***
        iv. ***Refine Generalization and aggregation.***
        v. ***Iterate and refine.***



Refining Attributes:

Attributes represents the information maintained by each object. Complete list of attributes should be identified in order to maintain a complete set of information. Detailed information of attributes is not specified in analysis phase but in design phase.

OCL is used to represent the attribute details inside a class diagram/ notation.

Various types of attributes are

4) Single valued attributes – an attribute represents one value at a time.
5) Multi valued attributes – an attribute can store more than one value
6) Reference attributes – an attribute refers (alias) another instance.

OCL format for representing attributes:

The OCL specification for specifying attributes is

Visibility attribute name : type

OR

Visibility attribute name : type = initial value

E.g.

+ Name : String
    - represents a public attribute Name of type String

Name : String = "Hello"
    - represents a protected attribute Name of type String with initial value Hello

[Note: Use the example from the case study]

Designing/ Refining Methods:

Designing methods involves design of Method protocol and Method body. A method protocol defines the rule for message passing to this particular object where the method body gives the implementation details. The types of methods provided by class are

9) Constructor – Method that is responsible for creating objects/ Method invoked during instantiating.
10) Destructor – The method that destroys instances/ Method invoked when an object is freed from memory.
11) Conversion Method – Methods responsible for converting one form of date to other form.
12) Copy Method – Methods responsible for copying information.
13) Attribute Set – Method responsible for setting values in attributes
14) Attribute Ger – Method responsible for getting the values from an attribute
15) I/O Methods – Method responsible for getting and sending data from a device

16) Domain Specific – Those methods responsible for some functionality in a particular domain.

Designing Protocol:

Protocol gives the rule for message passing between objects. Protocol is the interface provided by the object. Based on the visibility of the protocol it can be classified into

4. Public Protocol
5. Private Protocol
6. Protected Protocol.

Private protocols specify messages that can be send only by the methods inside the class. They are visible only inside the class.

Protected protocols specify messages that can be send only by the methods inside the class. But they can be inherited by the subclass.

Public protocols specify messages that can be send by the methods with in the class as well as objects outside the class.

Protocol and Encapsulation leakage – If protocols aren't well designed unnecessary messages are made available outside the class results in encapsulation leakage.

Internal layer contains the private and protected protocols where an External layer contains public protocols.

OCL Specification for Protocol Design:

The specification is

Visibility protocol name (argument list) : return type

Where argument list is *arg1: type, arg2: type, arg3: type … argn: type*

E.g.

+ getName () : String

It's a public protocol named getName with no parameters and it returns a value of type String.

- setData (name : String, no : Integer) : Boolean

It is a private protocol that accepts 2 arguments one of type String and other of type Integer. It returns a value of type Boolean.

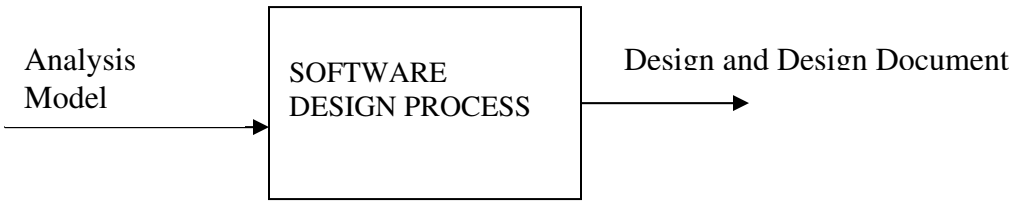
Designing Method body:

UML Activity diagram along with OCL specification can be used for representing the body of the method. Activity diagram representing the method body says how the work should be done.

[Note: Use an activity diagram from case study]

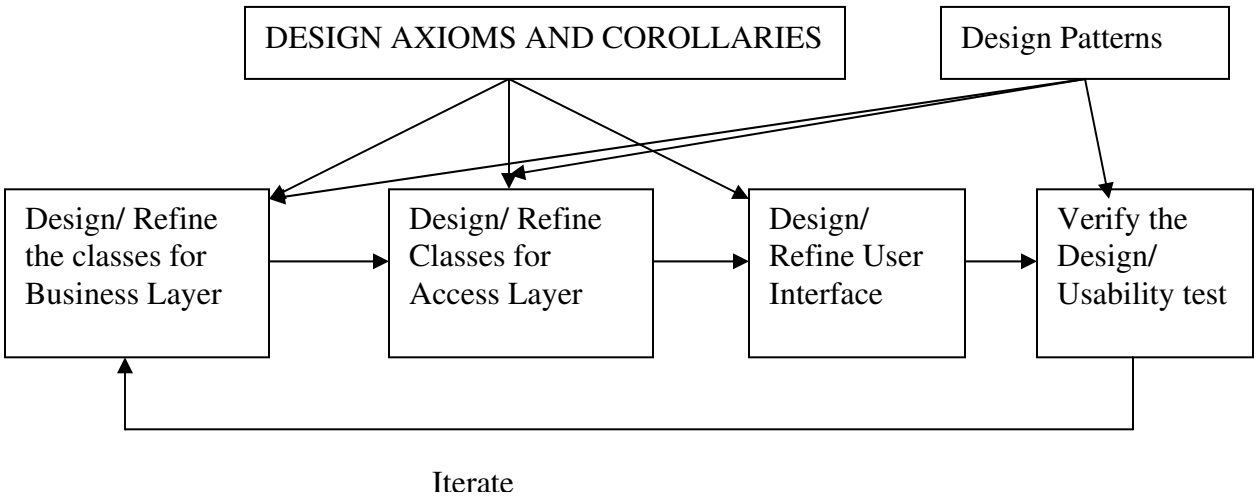
Object Oriented Design

Software Design represents the logic of the software system providing more dependency to the computer domain than physical/ user domain. Design actually deals with "LOGIC TO IMPLEMENT IN PROGRAM TO ACHIEVE THE SYSTEM GOAL"



I. SOFTWARE DESIGN PROCESS:

Software Design Process is the set of activities involved in developing a good and quality design. This is a sub process of Software Engineering Process.



The above diagram shows the different sub phases in software design process.
Unified Approach suggests 3-tired architecture. Since design has strongly dependency with implementation, the design should carried out for these layers separately.

4. ***Business Layer Class Design – apply design axioms for designing classes for business layer. Designing classes includes designing their attributes, methods and relationships.***
    I. ***Design/ Refine UML Class diagram developed in previous phase/ iteration.***

i. ***Design/ Refine attributes (Use OCL)***

1. ***Add left out attributes***
2. ***Specify visibility, data type and initial value if any for attributes***

ii. ***Design/ Refine Methods (Use OCL and UML Activity diagram)***

1. ***Add left out methods***
2. ***Specify visibility of the protocol (+, - ,#)***
3. ***Specify the argument list and return type***
4. ***Design the method body using UML Activity diagram and OCL.***

iii. ***Refine association***

iv. ***Refine Generalization and aggregation.***

v. ***Iterate and refine.***

II. ***Design access layer***

i. ***Create mirror class for all classes identified in business layer.***

ii. ***Identify access layer class relationship***

iii. ***Simplify access layer classes and class relationship***

1. ***Remove redundant classes***
2. ***Singe method classes can be removed and added in another class.***

iv. ***Iterate and refine.***

III. ***Design View Layer***

i. ***Design the macro level user interface – identify view layer objects***

ii. ***Design micro level user interface***

1. ***Design view layer objects by applying design axioms and corollaries.***
2. ***Build a prototype of view layer interface***

iii. ***Verify Usability and User Satisfaction***

iv. ***Iterate and refine***

5. ***Iterate and refine the above steps in necessary.***

[Note: Explanation is given for 3 sub process separately. Example is given in CASE study of ATM system.]

II. DESIGN AXIOMS AND COROLLARIES:

Axiom is a fundamental truth that has no exception or counter proof.
A corollary is one derived from axiom or another proves theorem.

These can be used in the software design for the following reasons

1. Making the design more informative and uniform.
2. Avoid unnecessary relationships and information.
3. Increase the quality.
4. Avoid unnecessary effort.

Axiom1 → The independence axiom → Maintain independence of components/ classes/ activities

Axiom2 → The information axiom → Minimize the information content of the design

Axiom1 → The independence axiom

It says that when we implement one requirement of an user it should not affect the other requirement or its implementation. I.e., each component should satisfy its requirements without affecting other requirements.

E.g.

Requirement1: Node1 should send multimedia files requested by the Node2.
Requirement2: Node1 one should take minimum time for sending due to heavy traffic.

Consider the component **C1** responsible for sending Multimedia files.

Choice 1: C1 reads the files and send the file header first and then the content in a byte stream. Here the component satisfies the first requirement where it fails to satisfy the second requirement.

Choice 2: C1 reads the files and compress the content and file header contains the file and compression information. Since the file size transferred is reduced this choice satisfies both requirements.

Axiom2 → The information axiom

It deals with the simplicity and less information content. The fact is less number of information makes a simple design, hence less complex. Minimizing complexity makes the design more enhanced. The best way to reduce the information content is usage of inheritance in design. Hence more information can be reused from existing classes/ components.

E.g.

Chioce1: (with out inheritance)

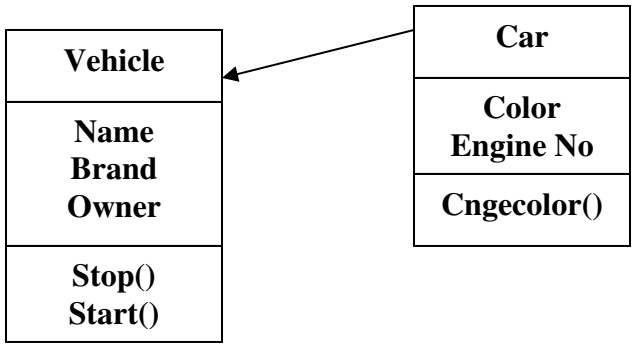
Vehicle
Name **Brand** **Owner**
Stop() **Start()**

Car
Name **Brand** **Owner** **Color** **Engine No**
Stop()

Class car maintains more information even though they are already maintained in vehicle class. Since car class maintains more information the design that contains the car class makes the design look more complex.

Vehicle → 3 properties and 2 methods.
Car → 5 properties and 3 methods.

Chioce2 (with inheritance)



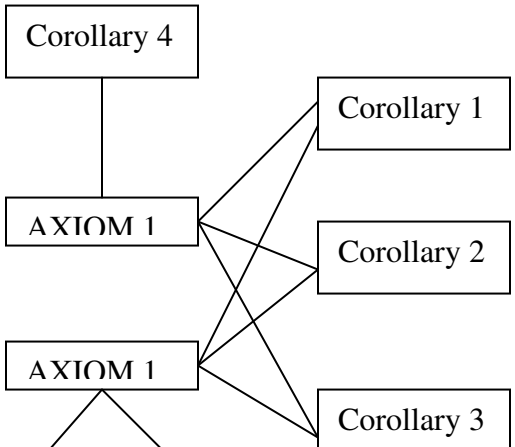
In this case the class car inherits some reusable methods and properties from vehicle class and hence it has to maintain 2 attributes and 1 method. Hence the class car looks simple.

Corollaries:

Corollaries are derived from design axioms(rules). These corollaries are suggestions to the designer to create a quality design.

They are

1. Corollary 1 → Uncoupled Design with less information content (from Axiom1 and 2)
2. Corollary 2 → Single purpose classes (from Axiom1 and 2)
3. Corollary 3 → large number of simple classes (from Axiom1 and 2)
4. Corollary 4 → Strong Maping (from Axiom 1)
5. Corollary 5 → Standardization (from Axiom 2)
6. Corollary 6 → Design with inheritance (from Axiom 2)



Corollary 1 → Uncoupled design with less information content.

This corollary explains the concept of dependency by cohesion and coupling.
Cohesion is the dependency among the classes inside a component
Coupling the measure of dependency between 2 components.
Designers prefer design with

1) Low coupling between components
2) High cohesion among the classes inside a component.

Hence by reducing the strength of coupling between classes/ components reduces the complexity of the design.

Coupling → It's the measure of association established between 2 objects/ components. Designers prefer weak coupling among components because effect of change in one component should have less impact on the other component. The degree (strength) of coupling is the function of

1. How complicated the connection is?
2. Whether the connection refers to object itself or something inside the referred object
3. What message/ data is being send and received.

Interaction coupling exist between 2 objects if there is a message passing between those 2 objects. The strength of interaction coupling depends on the complexity of messages passed between them.

Inheritance coupling exist between super and sub class. Inheritance coupling is desirable.
Types of coupling

1. Content Coupling (Very High)
2. Common Coupling (High)
3. Control Coupling (Medium)
4. Stamp Coupling (Low)
5. Data Coupling (Very low)

Cohesion → is the strength of dependency between classes with in a component. More cohesion reflects single purpose of the class. Designers prefer strong cohesion among contents of the component.

Corollary 2 → Single purpose

Each class should have a single well defined purpose and the aim of the class is to full fill that responsibility. If the class aims at implementing multi purpose subdivide the class in to smaller classes.

Corollary 3 → Large number of simpler classes for reusability

Complex classes are difficult to understand and hence for reuse needs more effort. Many times unnecessary members are reused as the super class was a complex one. The guideline says that "***The smaller are your classes, better your chances of reusing them in other projects. Large and complex classes are too specialized to be reused."***

Corollary 4 → Strong Mapping

The designer, analyst and programmer should maintain strong dependency among the products obtained during the different phases of SDLC. Hence a class identified during the analysis is designed in design phase and coded in implementation phase.
Designer should consider the programming language while creating design using technologies.

Corollary 5 → Standardization

The Designer/ Programmer should be aware of the existing classes/ components available in the standard library. This knowledge will help the designer to reuse the existing classes and design the newly needed classes/ components. A class/ pattern repository is maintained to store all the reusable classes and components. Even in most of the cases the repository is shared. Repository maintains the reusable components, description, commercial/ non commercial and usage.

Corollary 6 → Design with Inheritance

[Note: Include inheritance , importance and need of inheritance, types with one example]

OCL – Object Constraint Language:

It's a specification language used for representing properties of objects in an UML diagram. It is English like language. The rules and semantics of the UML diagrams can be represented using OCL. OCL specifications in UML diagrams make UML diagrams more clear and informative.
Sets, arithmetic expressions, Boolean expressions can be represented using OCL.

OCL Specifications:

1. Item. Selector

Selector → Used to get the value of the attribute.

Item → Entity to which the attribute belongs to.

E.g.

Stundent1. No = 30

Student1 is the Item and no is the selector.

2. Item. Selector [qualifier value]

Selector → Used to identify a set of similar values.

Item → Entity to which the attribute belongs to.

Qualifier→ specifies the particular value among the group.

E.g.

Student1. Mark [3]

Student1 is the Item, mark is the selector and 3 (qualifier) that represents 3rd mark

3. Boolean Expression

(Item1. Selector *Boolean operation* Item2.Selector)

E.g. S1. mark > S2. mark represents a Boolean value of true or false.

4. Set operation

Set → select (Boolean expression) is used to select a group of objects that

satisfies the Boolean expression.

Student → select (mark >40) selects a list of students who has mark greater than 40.

5. Attribute specification

The OCL specification for specifying attributes is

Visibility attribute name : type

OR

Visibility attribute name : type = initial value

E.g.

+ Name : String

- represents a public attribute Name of type String

Name : String = "Hello"

- represents a protected attribute Name of type String with initial value Hello

6. Protocol Design Specification

The specification is

Visibility protocol name (argument list) : return type

Where argument list is *arg1 : type, arg2 : type, arg3 : type … argn : type*

E.g.

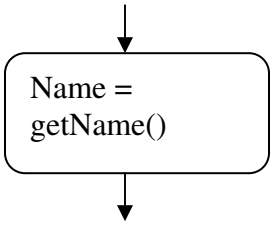
+ getName () : String

It's a public protocol named getName with no parameters and it returns a value of type String.

- setData (name : String, no : Integer) : Boolean

It is a private protocol that accepts 2 arguments one of type String and other of type Integer. It returns a value of type Boolean.

7. OCL in representing function call



Student:: + getName() : String

The activity diagram does not specify any details of getName where the OCL specification near the function call represents the clear idea about the method getName.

[Note: more examples should be added in each category]

Designing View Layer Classes:

View layer objects are more responsible for user interaction and these view layer objects have more relation with the user where business layer objects have less interaction with users. Another feature of view layer objects are they deal less with the logic. They help the users to complete their task in an easy manner.

The Major responsibilities of view layer objects are

1. Input – View Layer objects have to respond for user interaction. The user interface is designed to translate an action by the user (Eg. Clicking the button) in to a corresponding message.
2. Output - Displaying or printing information after processing.

View Layer Design Process:

1. **Macro Level UI Design Process**
   a. **Identify classes that interact with human actors**
   b. **A sequence/ collaboration diagram can be used to represent a clear picture of actor system interaction.**

c. **For every class identified determine if the class interacts with the human actor. If so**
    i. **Identify the view layer object for that class.**
    ii. **Define the relationship among view layer objects.**

2. **Micro Level UI Design Process**
    a. **Design of view layer objects by applying Design Axioms and Corollaries.**
    b. **Create prototype of the view layer interface.**
3. **Testing the usability and user satisfaction testing.**
4. **Iterate and refine the above steps.**

[Note: Explain the above process for an essay]

User Interface Design Rules:

UI Design Rule 1: Making the interface simple

For complex application if the user interface is simple it is easy for the users to learn new applications. Each User Interface class should have a well define single purpose. If a user cannot sit before a screen and find out what to do next without asking multiple questions, then it says your interface is not simple.

UI Design Rule 2: Making the Interface Transparent and Natural.

The user interface should be natural that users can anticipate what to do next by applying previous knowledge of doing things with out a computer. This rule says there should be a strong mapping and users view of doing things.

UI Design Rule 3: Allowing users to be in control of the Software.

The UI should make the users feel they are in control of the software and not the software controls the user. The user should play an active role and not a reactive role in the sense user should initiate the action and not the software.

Some ways to make put users in control are

1. Make the interface forgiving.
2. Make the interface visual.
3. Provide immediate feedback.
4. Avoid Modes.
5. Make the interface consistent.

Purpose of View Layer Interface - Guidelines

Unit – V

Software Quality Assurance Testing:

Bugging and Debugging - Related to Syntax Errors.
Testing – Related to Logical, Interface and communication errors.
Testing – for assuring quality. (In general)

Software Quality Assurance Testing – For Satisfying the Customer – Give more importance to testing the Business Logic and less importance to satisfaction of the user.

User Interface Testing – Usability and User Satisfaction – More importance is given to the easiness of the User Interface and less importance to the logic.

Quality Assurance Testing

- Error Based Testing
  - Methods are tested with some valid and invalid parameters
  - Boundary Tests
    - Methods are tested with boundary values of the parameters. The boundary values are
      - - ve value (if applicable)
      - Minimum Value – 1
      - Minimum Value
      - Medium Value
      - Maximum Value – 1
      - Maximum Value
      - Maximum Value + 1etc etc
- Testing Strategies
  - Black Box Testing.
  - White Box Testing.
  - Top down Approach.
  - Bottom up Approach.

    (Note: Refer SE notes/ Pressman book)
- Impact of OO on Testing
  - Impact of Inheritance

        - It is not necessary to test Inherited Methods because its already been verified in the Base class.
        - But if the inherited method is over ridded then the behavior may change and it is needed to be tested.
        - (Note: Explain with an example)
    - Reusability of Test Cases
        - Test Cases can be reused based up on the level of reusage
        - Overridden methods show a different behavior.
        - If the similarities in behavior exist test cases can be reused.
        - In some cases Inherited method accept same parameter as Base but different behavior.
        - In the above case a test case can be reused such a way that the expected o/p of the test case is changed and used.
    - Test Cases
        - Test cases represents various testing scenarios.
        - A good test case is one which has a high probability of detecting an undiscovered error.
        - A successful test case is one that can detect an undiscovered error.
        - Guidelines
            - Describe which feature the test attempt to cover
            - If scenario based then develop test case based on Use case
            - Specify what feature is going to be tested and what is the input/parameters and expectations
            - Test the normal usage of that object
            - Test abnormal reasonable usage
            - Test abnormal unreasonable usage
            - Test boundary conditions
            - Document the cases for next iteration
    - Test Plan
        - Objectives of the Test – Create the objectives and describe how to achieve them
        - Development of Test Case – Develop test data, both input and expected output
        - Test Analysis – Analysis of test case and documentation of test results.
    - Guidelines for developing Test Plans
        - Develop test plan based on the requirements generated by the user.
        - It should contain the schedule and list of required resources.
        - Determine the strategy (Black box, white box etc.) document what is to be done.
        - SCM (Software Configuration management or Change control) activities should be considered when ever a change is made due to a test result.
        - Keep the plan up to date.
        - Update documents when a mile stone is reached.
    - Continuous Testing
        - Since UA suggests iterative development continuous testing is advisable for efficient management of resource.
        - Testing is also carried out for each iteration of development.
- Myer's bug location and debugging principles.
    - Bug Locating Principles
        - Think

        - If you reach an impasse (deadlock) sleep on it.
        - If impasse still remains discuss the problems to some one else
    - Debugging Principles
        - Where there is one bug there is likely to be another
        - Fix the error and not the symptom
        - The probability of correctness of the solution decreases when the program size increases.
        - Beware that error correction should not create more number of errors.

Example Format
For example..fill out the categories for any object
Test plan

1. Project Name:
2. Team Name:
3. Test Team Details:
4. Shedule:
5. Object/ Class under testing:
6. List of Methods:
7. Test Cases for Each Method.
    a. Method1
        i. Test Case 1
        ii. Test Case 2
        iii. ..
        iv. Test Case N
    b. Method2
        i. Test Case 1
        ii. Test Case2
        iii. ..
        iv. Test Case N

Note: For user Satisfaction and usability testing refer text book.