SOOAD

UNIT 3

OBJECT ORIENTED ANALYSIS & DESIGN

UNIT -3 OOAD

- Introduction
- Object-Oriented Modelling
- Object-Oriented Approach
- The Constituents of OOAD
- Pillars of Object-Oriented Analysis and Design
- The Language of OOAD – Unified Modelling Language

Introduction

- Object-oriented analysis and design (OOAD) is a technical approach for analyzing and designing an application, system, or business by applying object-oriented programming, as well as using visual modeling throughout the software development process to guide stakeholder communication and product quality.
- OOAD in modern software engineering is typically conducted in an iterative and incremental way.
- The outputs of OOAD activities are analysis models (for OOA) and design models (for OOD) respectively.
- The intention is for these to be continuously refined and evolved, driven by key factors like risks and business value.

Introduction

- The first object–oriented language was Simula (Simulation of real systems) that was developed in 1960 by researchers at the Norwegian Computing Center.
- In 1970, Alan Kay and his research group at Xerox PARK created a personal computer named Dynabook and the first pure object-oriented programming language (OOPL) - Smalltalk, for programming the Dynabook.
- In the 1980s, Grady Booch published a paper titled Object Oriented Design that mainly presented a design for the programming language, Ada. In the ensuing editions, he extended his ideas to a complete object–oriented design method.
- In the 1990s, Coad incorporated behavioral ideas to object-oriented methods.
- The other significant innovations were Object Modeling Techniques (OMT) by James Rum Baugh and Object-Oriented Software Engineering (OOSE) by Ivar Jacobson.

OBJECT ORIENTED MODELLING

Object Oriented Modelling

- Object Oriented Analysis & Design (OOAD) encompasses on
  - Objects
  - Analysis
  - Design

- OOAD is a software engineering approach that models a system as group of interacting objects.

- Each object represents some entity of interest in the system being modelled and characterized by its
  - class
  - state
  - behaiviour

OOAD

What is OOAD?

INTELLIGROUP

- **Object-oriented analysis and design** (OOAD) is a software engineering approach that models a system as a group of interacting objects .
- **Analysis** — understanding, finding and describing concepts in the problem domain.
- **Design** — understanding and defining software solution/objects that *represent* the analysis concepts and will eventually be implemented in code.
- **OOAD** — Analysis is object-oriented and design is object-oriented. A software development approach that emphasizes a logical solution based on objects.

Object Oriented Modelling

- Various models are used to show the static structure, dynamic behaviour and run-time deployment of these collaborating objects.
- Different Models for different phases are:
  - Analysis Model
  - Architecture Model
  - Component Model

Object Oriented Modelling

- Analysis Model:
  - Model of existing system
  - The user's requirement
  - a high-level understanding of a possible solution to those requirements.

Object Oriented Modelling

- Architecture Model:
  - Evolving model
  - structure of the solution to the requirements defined in analysis model
  - Primary focus is on architecture:
    - Components
    - interfaces and
    - the structure of the solution,
    - the deployment of structure across nodes and
    - trade-offs and decisions that lead up to that structure

Object Oriented Modelling

- Component (Design) Model:
  - Number of models
  - one per component shows internal structure of the pieces of the architecture model.
  - Detailed class structure of its component
  - Attributes, operations, dependencies, and the behaviour of its classes.

Why to do Modelling?

- Can make model changes in development model
- To catch costly bugs early
- Eary detection and correction can save a lot on the cost and schedule of a bug fix.

OOA vs. OOD

- Object Oriented Analysis (OOA) applies object-modelling techniques to analyze the functional requirements for a system
- Object Oriented Design (OOD) elaborates the analysis models to produce implementation specifications.

OOA focuses on what the system does whereas

OOD focuses on how the system does it.

OBJECT ORIENTED APPROACH

Object Oriented Approach

First step of OO methodology is concerned with understanding the domain and modelling the real world application for problem statement formulation.

OOAD Stages:

- Analysis stage
- Design stage
- Implementation stage

Object Oriented Approach

- Analysis stage - produces SRS (Software/ System Requirement Specification)
  - Consists of:
    - Business process diagrams
    - Use-case diagrams
    - Class and object diagram

Object Oriented Approach

- Design Phase – Database Design is arrived from analysis stage.
  - Involves
    - Sequence diagram
    - Collaboration diagram
    - Activity diaram
    - State-chart diaram
- Implementation Phase – clear modular structure for programs where implementation details are hidden
  - Produces
    - Component diagram
    - Deployment diagram

Object Oriented Programming

- OOP is primarily concerned with programming language and software implementation issues.
- OOP makes it easy to maintain and modify existing code
- OOP provides a good framework for code libraries
- Code can be easily adopted and modified by a programmer.

Object Orientation

- Object Oriented (OO) means organize software as a collection of discrete objects.
- It has both data structure and behaviour.
- Map natuarally to real-world objects.
- Supports abstraction at object level.
- Development can proceed at the object level
- Designing, coding, testing and maintaining the system much simpler.
- Moving from one phase to another does not require different styles and methodologies
- Reduce complexity and makes clear system development
- Nice syntatic mechanism for achieving some classic aspects of well-designed code.

Object Orientation Analysis

- Examines problem domain
- Producing a conceptual model of the information that exists in the area being examined.
- Do not consider implementation problems.
- Analysis is done before design
- Sources of analysis:
  - written statement
  - formal document
  - Interviews with stakeholders

Object Orientation Analysis

- System is divided into multiple domains, representing different business, technological and analyzed seperately.

Result of OOA is a description of what the system is functionally required to do, in the form of a conceptual model.

A set of USE-CASES, one or more UML class diagrams, and a number of interaction diagrams.

OOA is the process of defining the problems in terms of real world objects.

Object Orientation Analysis

- OOA is the process of defining the problem in terms of objects:
  - real world objects with which system must interact, and candidate software objects used to explore various solutions aalternatives.
- One can define all of the real-world objects in terms of their classes, attributes and operations.

Object Orientation Design

- OOA means defining the problem and OOD is the process of defining the solution.
- Defining the ways in which the system is prepared as per analysis phase.
- "Object Oriented Design (OOD) is the process of defining the components, interfaces, objects, classes, attributes and operations that satisfies funtional requirements."
- Start with candidate objects defined during analysis, but can add much more rigour to their definitions.
- Then we add or change objects as needed to refine a solution.

Object Orientation Design

- Two Scales of Design:
  - *Achitectural design* - defining components of system
  - *Component design* – defining classes and interfaces
  - OOD transforms the conceptual model to take account of constraints imposed by the chosen architecture and any non functional constraints, like transaction throughput, response time, run-time platform, developement environment, or programming language.
  - The concepts in the analysis model are mapped onto implementation classes and interfaces.
  - The result is a model of the solution domain, a detailed description of how the system is to be built.

THE CONSTITUENTS OF OOAD

The Constituents Of OOAD

- Objects
- Classes
- Links
- Association
- Generalization
- Specialization
- Aggregation
- Composition

Object

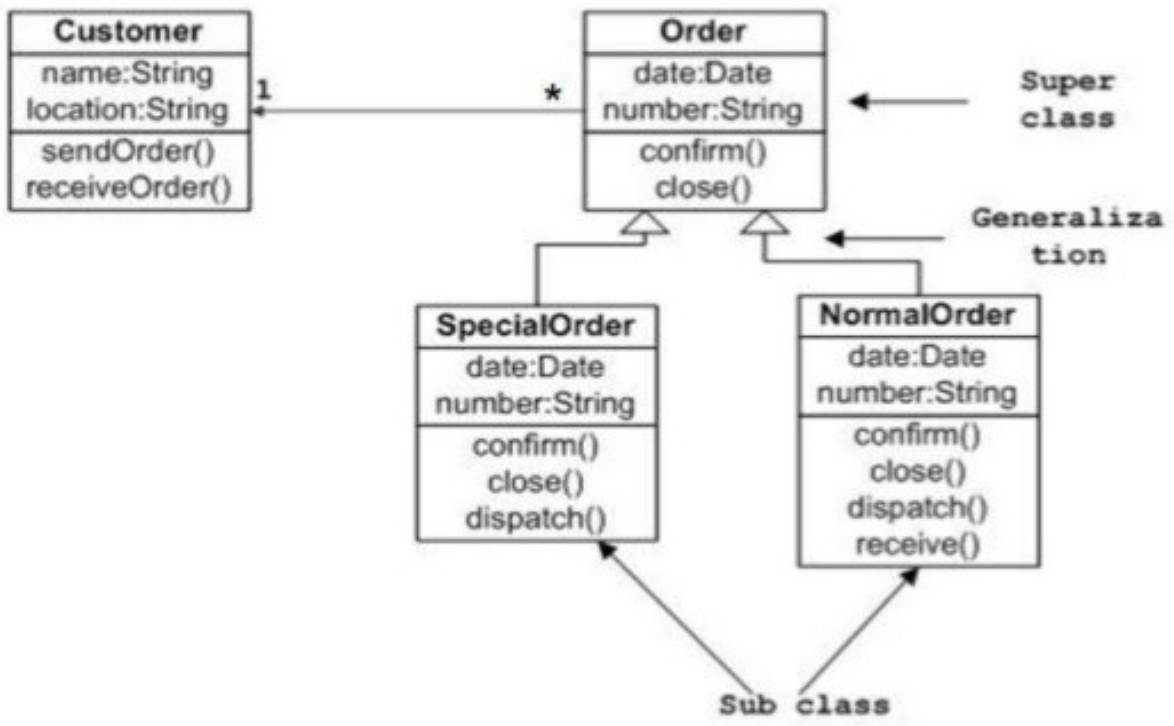
- An object is the foundation of an OOP.
- Basic run-time entities in an OO system.
- Problems are identified in terms of objects.
- Object interact with each other by sending messages.
- Without knowing the details of their data or code.
- Objects improves program reliability, simplify software maintenance and management of libraries.
- Object is identified by its identity that distingushes it from other objects and its behaviour.

Class

- Class is a blue print or factory that defines the abstract characteristics of an object including its attributes, fields or properties and its abstract behaviour likes its methods, operations or features.
- Collection of objects of similar types.
- Attributes: state that describes an object.
- Operations (Methods): is a behaviour that an object can perform.

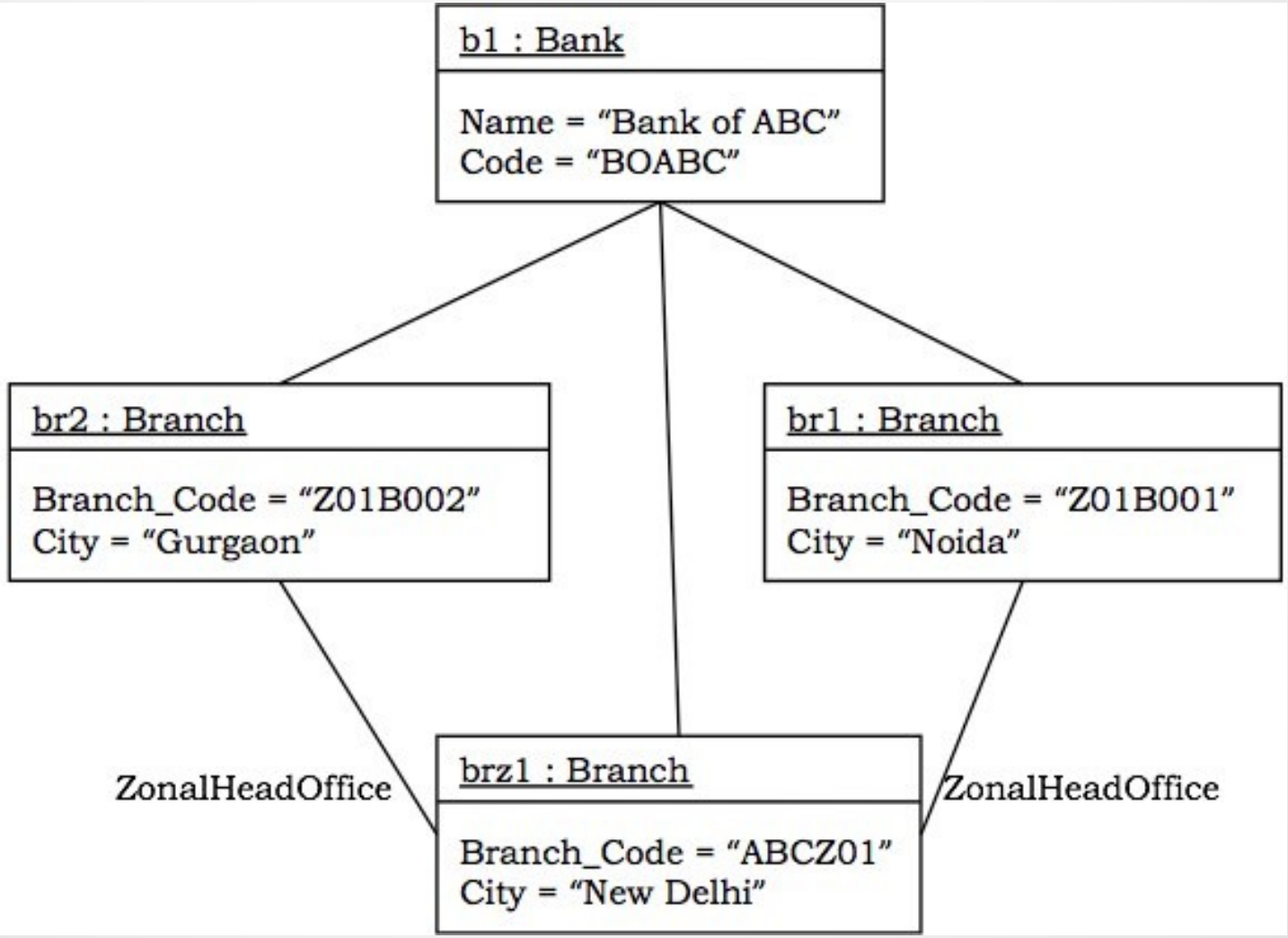
Class

Class Diagrams



A sample class diagram

Class


b1 : Bank
Name = "Bank of ABC"
Code = "BOABC"
br2 : Branch
Branch_Code = "Z01B002"
City = "Gurgaon"
br1 : Branch
Branch_Code = "Z01B001"
City = "Noida"
ZonalHeadOffice
brz1 : Branch
Branch_Code = "ABCZ01"
City = "New Delhi"
ZonalHeadOffice

Links

- A link is a relationship among instance of classes (objects).
- To represent relationship between two objects.
- A link represents a connection through which an object collaborates with other objects.
- It as “a physical or conceptual connection between objects”.
- A link depicts the relationship between two or more objects.

Association

- Association is used to represent the relationship between the two classes.
- Example: A Student and a Faculty are having an association.
- Association is a group of links having common structure and common behavior.
- Association depicts the relationship between objects of one or more classes.
- A link can be defined as an instance of an association.

Links & Association

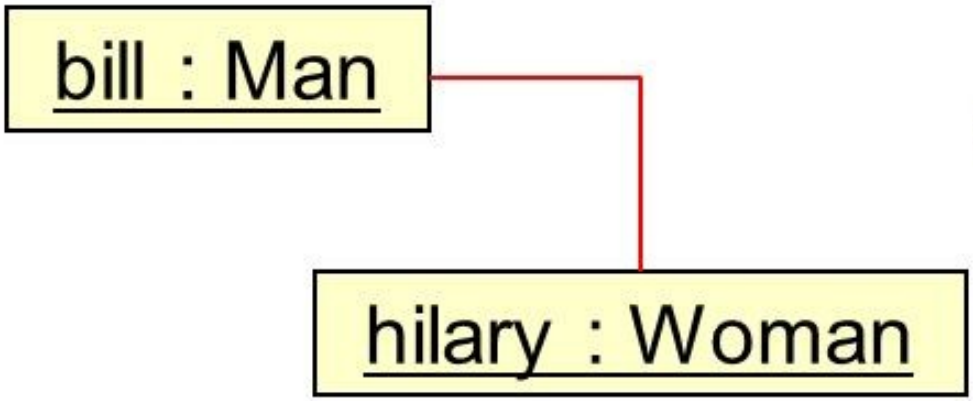
LINK AND ASSOCIATION

- links and association are the means for building the relationship among the objects and classes.
- Links and association , both are quite same feature but links establishing among the objects (instance) and association establishing among the class.
- Finally link is related to objects whereas association is related to classes

Links & Association

- **Association**

- **Link**
  - Instance of an association

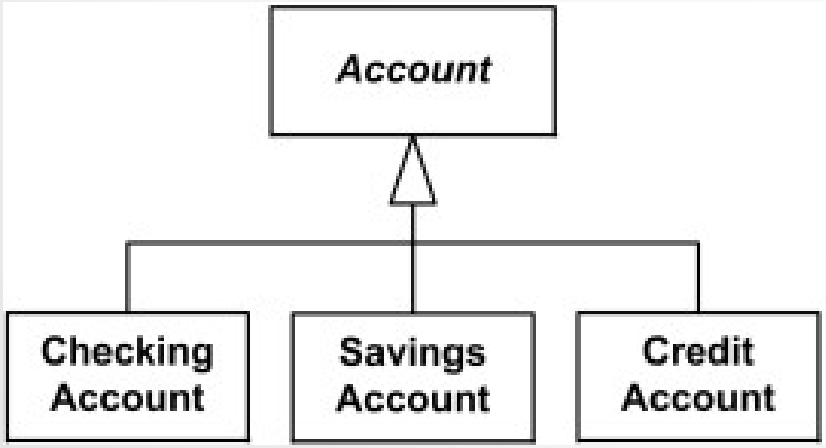


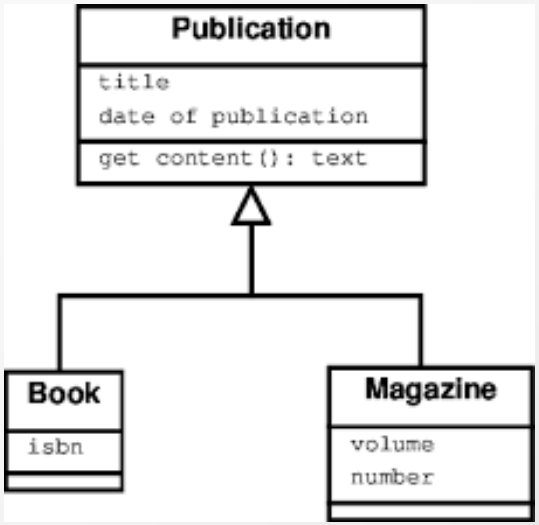
Generalization

- Act of identifying and categorizing similar objects into classes.
- In the generalization process, the common characteristics of classes are combined to form a class in a higher level of hierarchy, i.e., subclasses are combined to form a generalized super-class.
- It represents an "is – a – kind – of" relationship.
- For example, "car is a kind of land vehicle", or "ship is a kind of water vehicle".

Generalization

- Generalization is the process of extracting shared characteristics from two or more classes, and combining them into a generalized superclass.
- Shared characteristics can be attributes, associations, or methods.

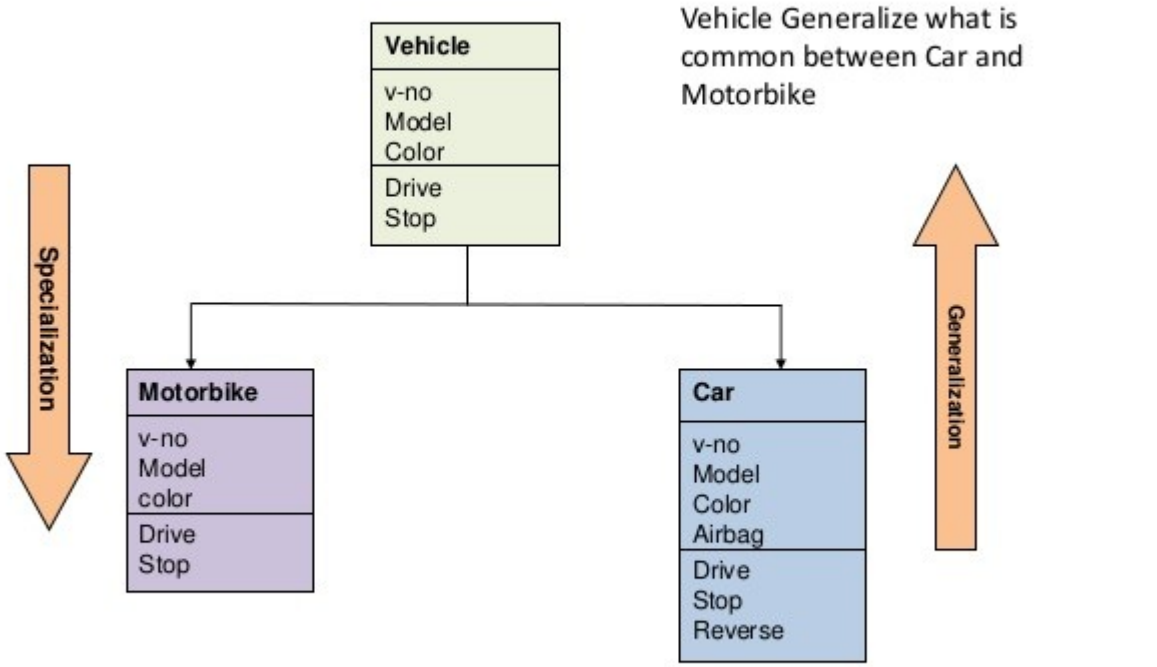




Specialization

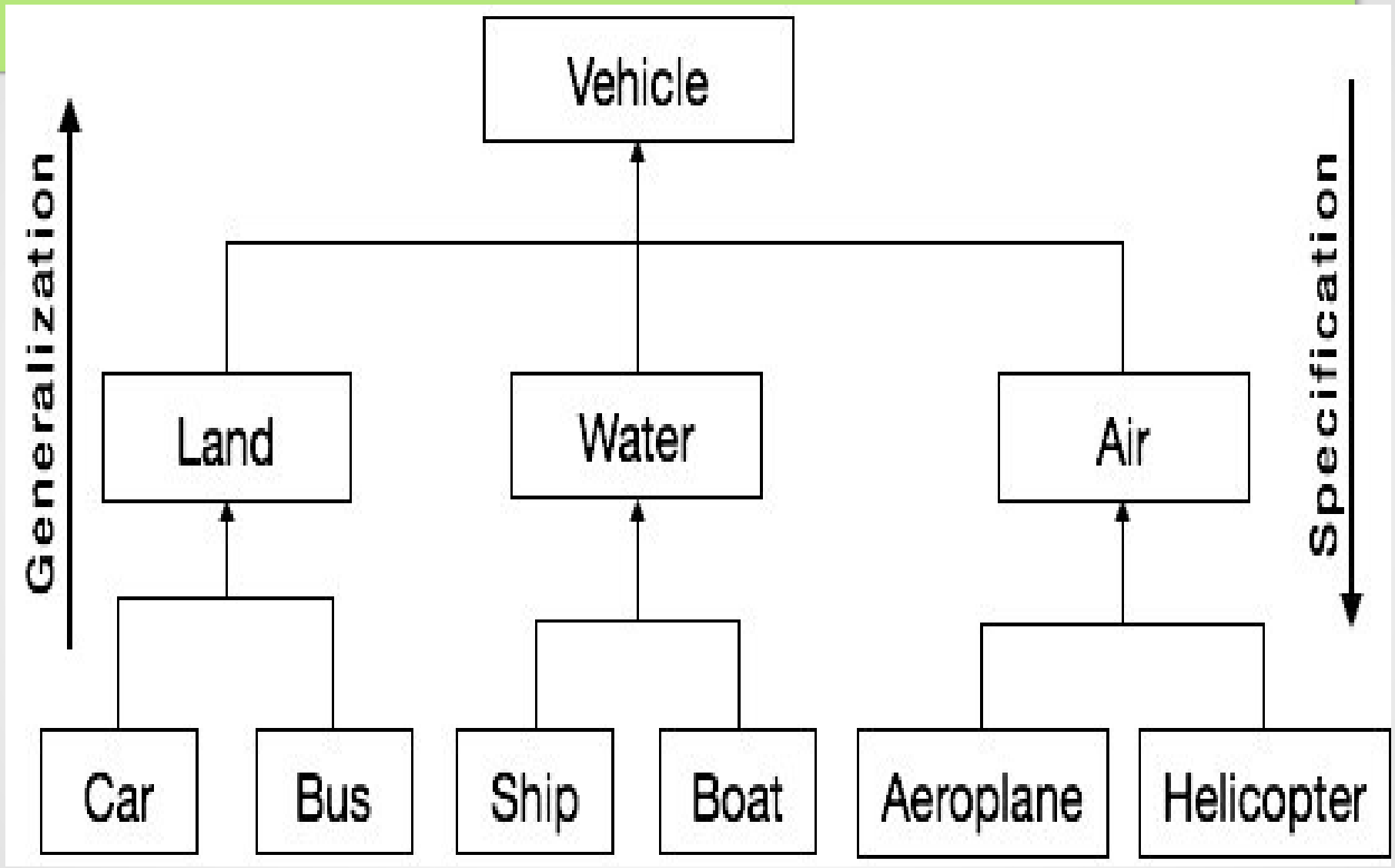
- In contrast to generalization, specialization means creating new subclasses from an existing class.
- If it turns out that certain attributes, associations, or methods only apply to some of the objects of the class, a subclass can be created.
- Here, the distinguishing features of groups of objects are used to form specialized classes from existing classes.
- It can be said that the subclasses are the specialized versions of the super-class.

Generalization / Specialization



Generalization & Specialization

- *Generalization* is the process of identifying common features among classes leading to *superclasses*
- *Specialization* is the process of creating more specialized *subclasses* from an existing class


Vehicle
Generalization
Specification
Land
Water
Air
Car
Bus
Ship
Boat
Aeroplane
Helicopter

Aggregation or Composition

- Aggregation or composition is a relationship among classes by which a class can be made up of any combination of objects of other classes.
- It allows objects to be placed directly within the body of other classes.
- Aggregation is referred as a “part–of” or “has–a” relationship, with the ability to navigate from the whole to its parts.
- An aggregate object is an object that is composed of one or more other objects.

Aggregation or Composition

- Aggregation is a special case of association.
- A directional association between objects.
- When an object ‘has-a’ another object, then you have got an aggregation between them.
- Direction between them specified which object contains the other object.
- Aggregation is also called a “Has-a” relationship.

Composition

- Composition is a special case of aggregation.
- In a more specific manner, a restricted aggregation is called composition.
- When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition.
- Example: A class contains students. A student cannot exist without a class. There exists composition between class and studen

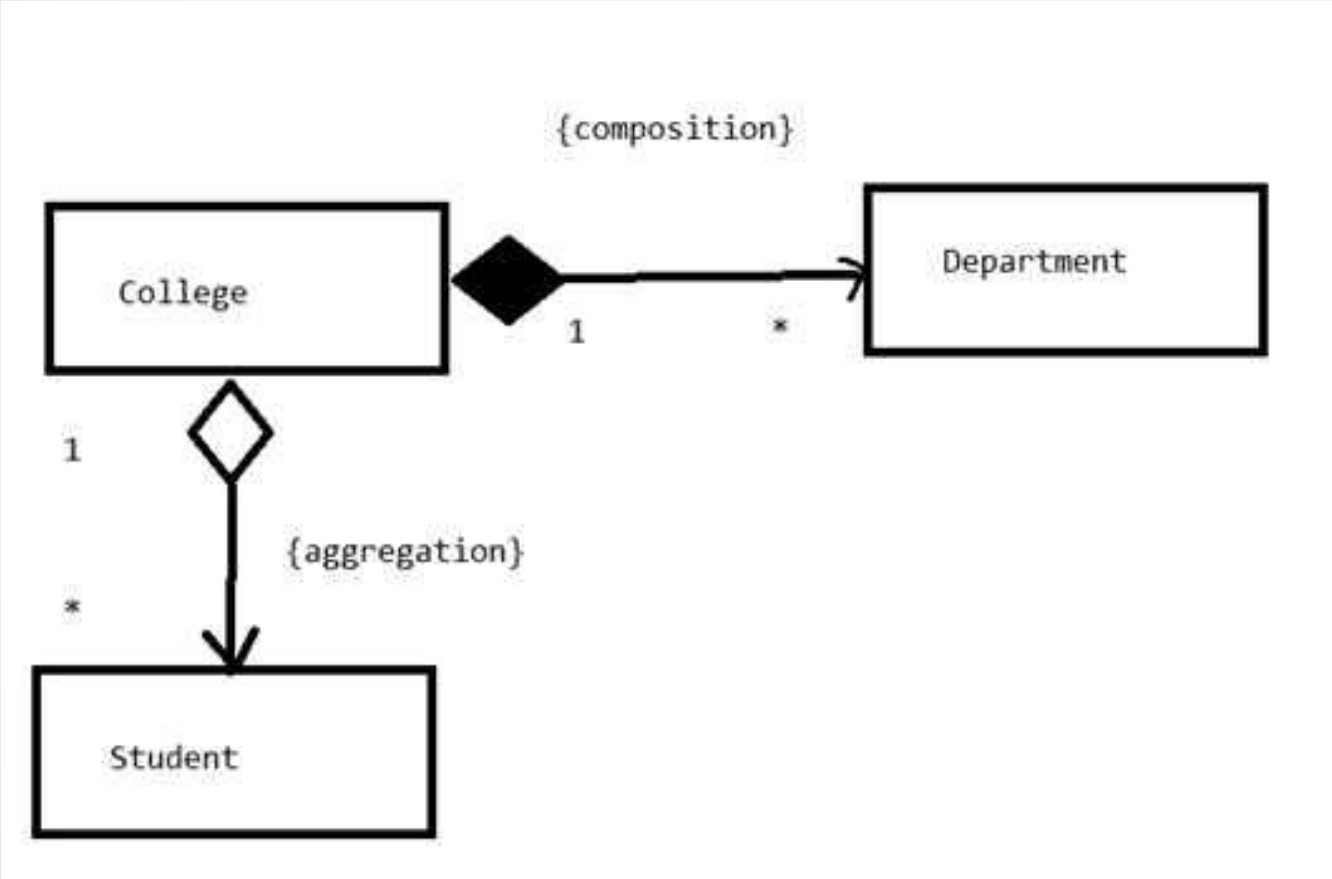
Difference between aggregation and composition

- Composition is more restrictive. When there is a composition between two objects, the composed object cannot exist without the other object. This restriction is not there in aggregation.
- Though one object can contain the other object, there is no condition that the composed object must exist.
- The existence of the composed object is entirely optional.
- In both aggregation and composition, direction is must. The direction specifies, which object contains the other object.

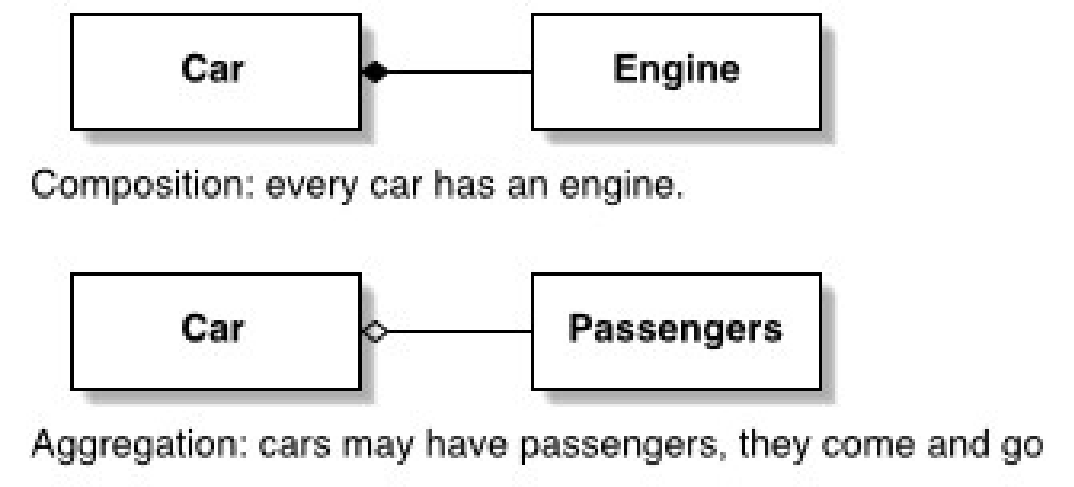
Difference between aggregation and composition

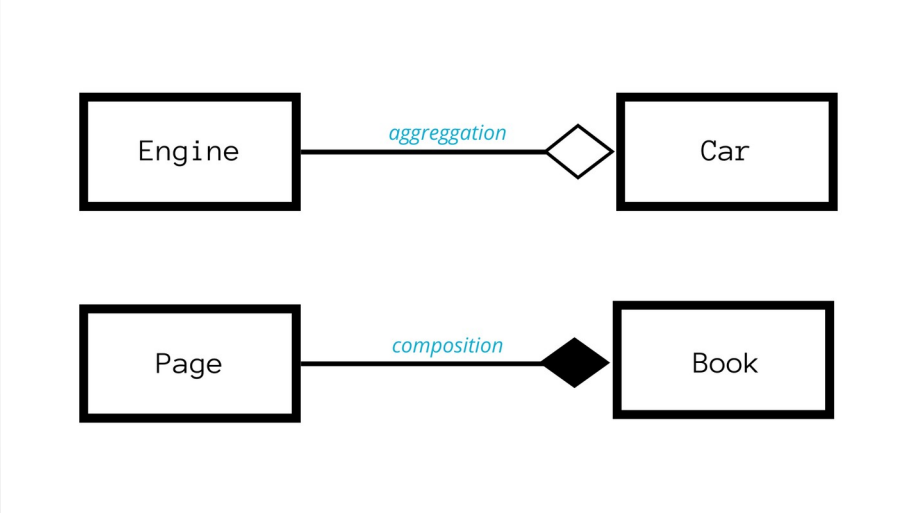
- Example:
  - A Library contains students and books.
- Relationship between library and student is aggregation.
- Relationship between library and book is composition.
- A student can exist without a library and therefore it is aggregation.
- A book cannot exist without a library and therefore its a composition.

Difference between aggregation and composition



Difference between aggregation and composition





PILLARS OF OOAD

PLLARS OF OOAD

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Coupling
- Cohesion
- Components
- Interface

Abstraction

- Abstraction is specifying the framework and hiding the implementation level information.
- Concreteness will be built on top of the abstraction.
- It gives you a blueprint to follow to while implementing the details. Abstraction reduces the complexity by hiding low level details.
- Act of representing essential features without including the background details or explanations.

Encapsulation

- Like a black box.
- Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. It permits the elements of the class to be accessed from outside only through the interface provided by the class.
- Data Hiding
-
- Typically, a class is designed such that its data (attributes) can be accessed only by its class methods and insulated from direct outside access. This process of insulating an object's data is called data hiding or information hiding.

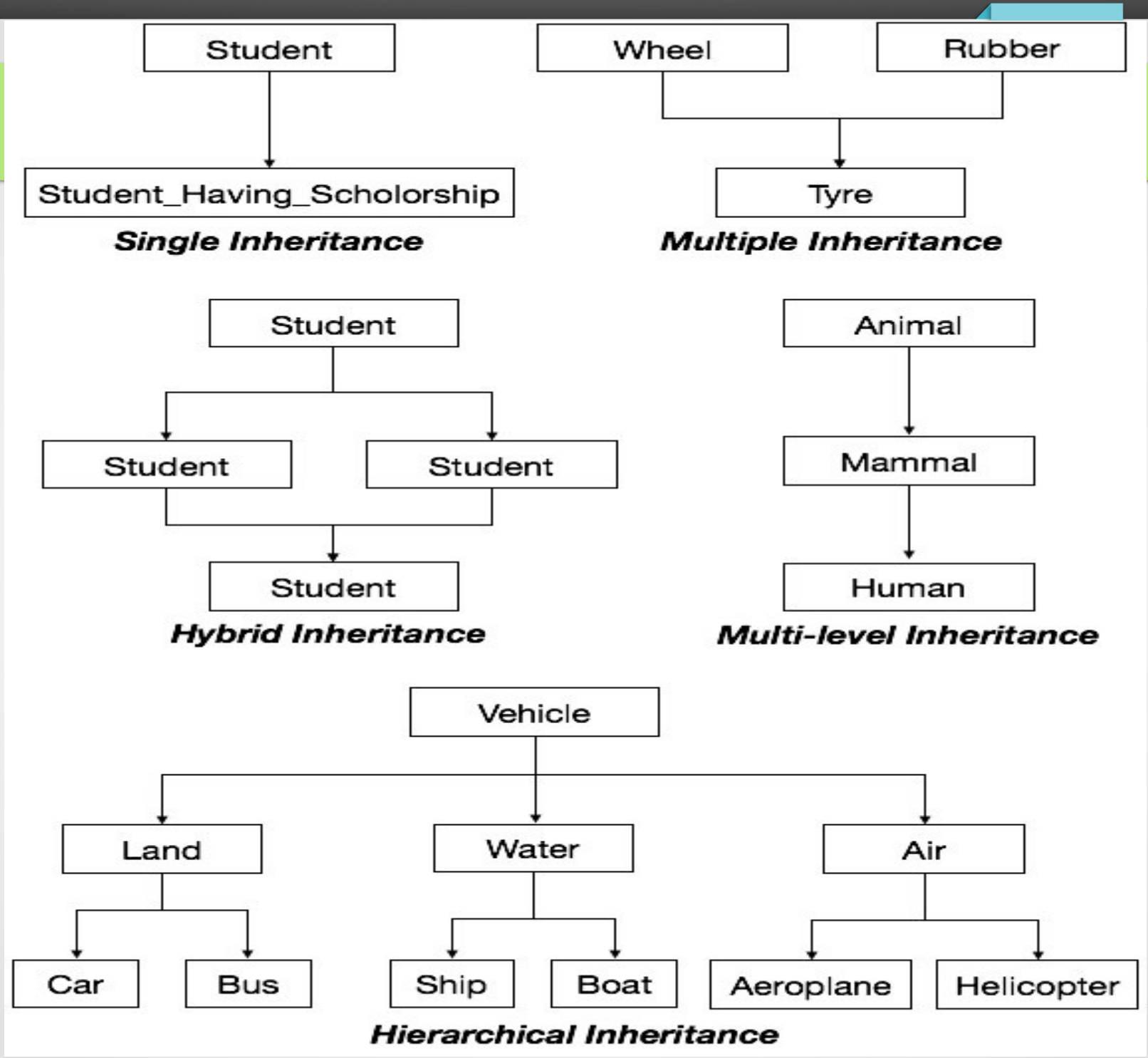
- Encapsulation means as much as shielding. Each OO object has a shield around it. Objects can't 'see' each other. They can exchange things though, as if they are interconnected through a hatch.
- coffee protocol, an example of encapsulation Customer, waiter and kitchen are three shielded objects in the 'cup of coffee' example. Customer and kitchen do not know each other. The waiter is the intermediary between those two. Objects can't see each other in an Object Oriented world. The 'hatch' enables them to communicate and exchange coffee and money.
- Encapsulation keeps computer systems flexible. The business process can change easily. The customer does not care about the coffee brew process. Even the waiter does not care. This allows the kitchen to be reconstructed, is only the 'hatch' remains the same. It is even possible to change the entire business process. Suppose the waiter will brew coffee himself. The customer won't notice any difference.
- Encapsulation enables OO experts to build flexible systems. Systems that can extend as your business extends. Every module of the system can change independantly, no impact to the other modules.

Inheritance

- Inheritance is the mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities.
- The existing classes are called the base classes/parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses.
- The subclass can inherit or derive the attributes and methods of the super-class(es) provided that the super-class allows so.
- Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines an “is – a” relationship.

Polymorphism

- Polymorphism is originally a Greek word that means the ability to take multiple forms. In object-oriented paradigm, polymorphism implies using operations in different ways, depending upon the instance they are operating upon.
- Polymorphism allows objects with different internal structures to have a common external interface. Polymorphism is particularly effective while implementing inheritance.


Student
Student_Having_Scholorship
Single Inheritance
Wheel
Rubber
Tyre
Multiple Inheritance
Student
Student
Student
Student
Hybrid Inheritance
Animal
Mammal
Human
Multi-level Inheritance
Vehicle
Land
Water
Air
Car
Bus
Ship
Boat
Aeroplane
Helicopter
Hierarchical Inheritance

coupling

- Refers to the ways, in which and degrees, to which one part of the system relies on the details of another part.
- The tighter the coupling, the more changes in one part of the system will ripple throughout the system.
- With loose coupling, the interfaces between subsystems are well defined and restricted.
- What lies beyond those interfaces can change without any changes needed in the client subsystems.
- Oop supports loose coupling by allowing us to define and publish a class's method without publishing how those methods are carried out.
-

Cohesion

- Refers to the degree in which elements within a subsystem form a single, unified concept, with no excess elements.
- Where there is high cohesion, there is easier comprehension and thus more reliable code.
- Oop supports strong cohesion by allowing one to design classes in which the data and the fuctions that operate on them are tightly bound together.

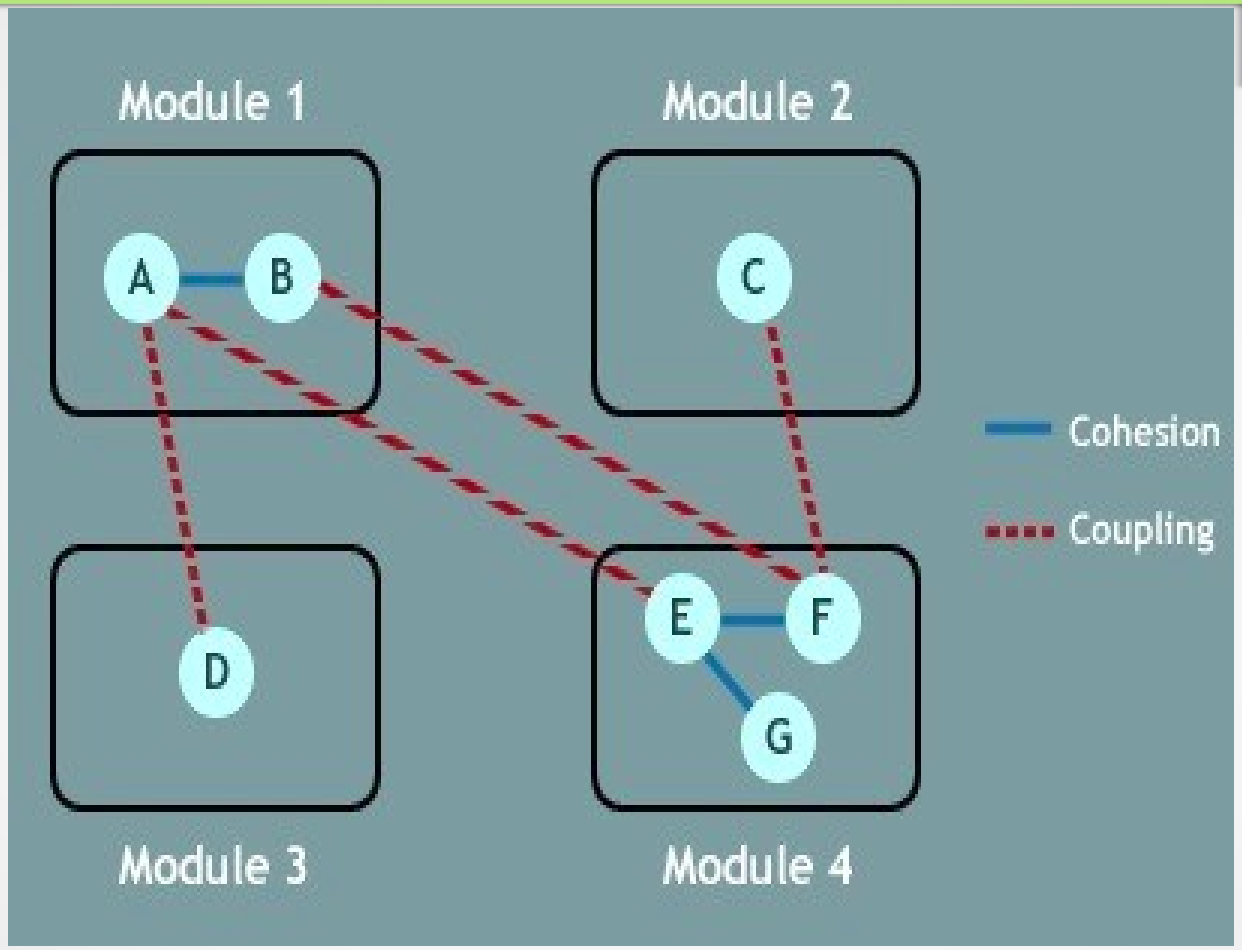
coupling

- Cohesion and Coupling deal with the quality of an OO design. Generally, good OO design should be loosely coupled and highly cohesive.
- Coupling is the degree to which one class knows about another class.
- Let us consider two classes class A and class B. If class A knows class B through its interface only i.e it interacts with class B through its API then class A and class B are said to be loosely coupled.

- If on the other hand class A apart from interacting class B by means of its interface also interacts through the non-interface stuff of class B then they are said to be tightly coupled. Suppose the developer changes the class B‘s non-interface part i.e non API stuff then in case of loose coupling class A does not breakdown but tight coupling causes the class A to break.
- So its always a good OO design principle to use loose coupling between the classes i.e all interactions between the objects in OO system should use the APIs. An aspect of good class and API design is that classes should be well encapsulated.

Cohesion

- Cohesion is used to indicate the degree to which a class has a single, well-focused purpose. Coupling is all about how classes interact with each other, on the other hand cohesion focuses on how single class is designed. Higher the cohesiveness of the class, better is the OO design.


Module 1
Module 2
A
B
C
D
E
F
G
Module 3
Module 4
Cohesion
Coupling

Cohesion is the indication of the relationship within module.

Coupling is the indication of the relationships between modules.

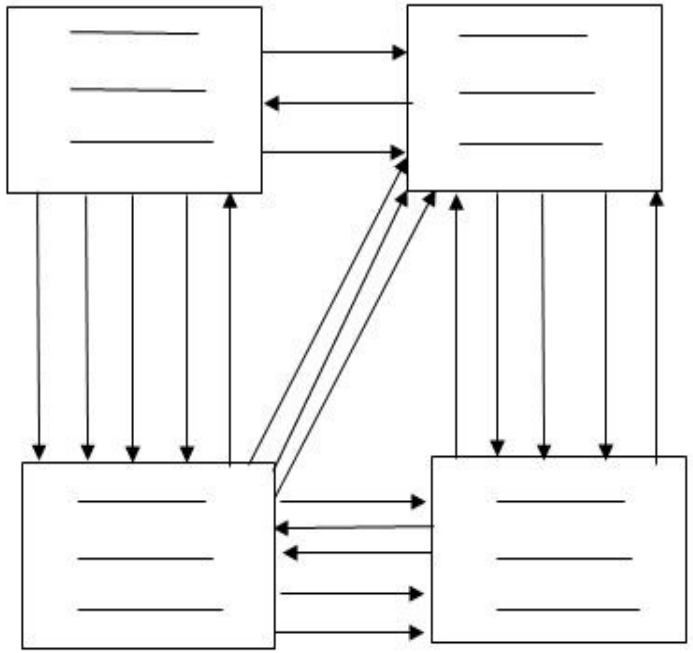
Cohesion shows the module's relative functional strength.

Coupling shows the relative independence among the modules.

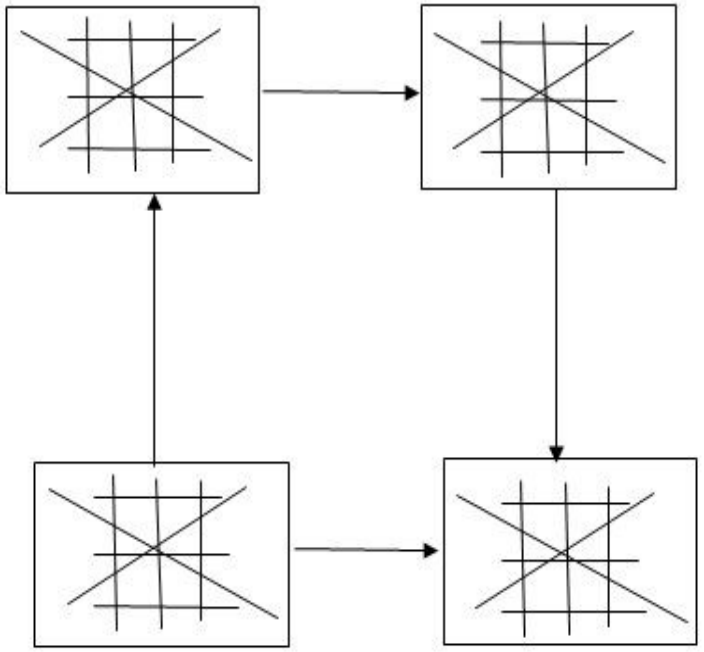
Cohesion is a degree (quality) to which a component / module focuses on the single thing.

Coupling is a degree to which a component / module is connected to the other modules.

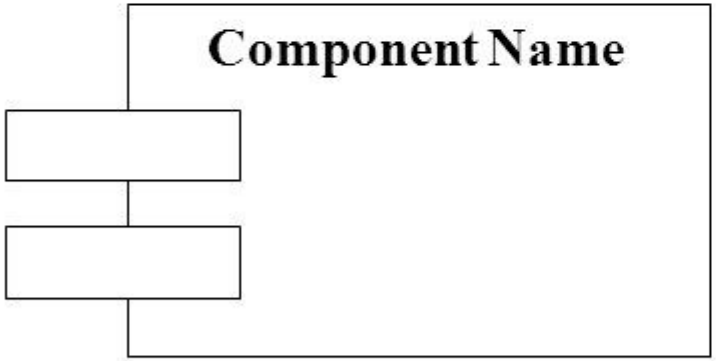
- While designing you should strive for high cohesion i.e. a cohesive component/ module focus on a single task (i.e., single-mindedness) with little interaction with other modules of the system.
- While designing you should strive for low coupling i.e. dependency between modules should be less.
- Cohesion is the kind of natural extension of data hiding for example, class having all members visible with a package having default visibility.
- Making private fields, private methods and non public classes provides loose coupling.
- Cohesion is Intra – Module Concept.
- Coupling is Inter -Module Concept.

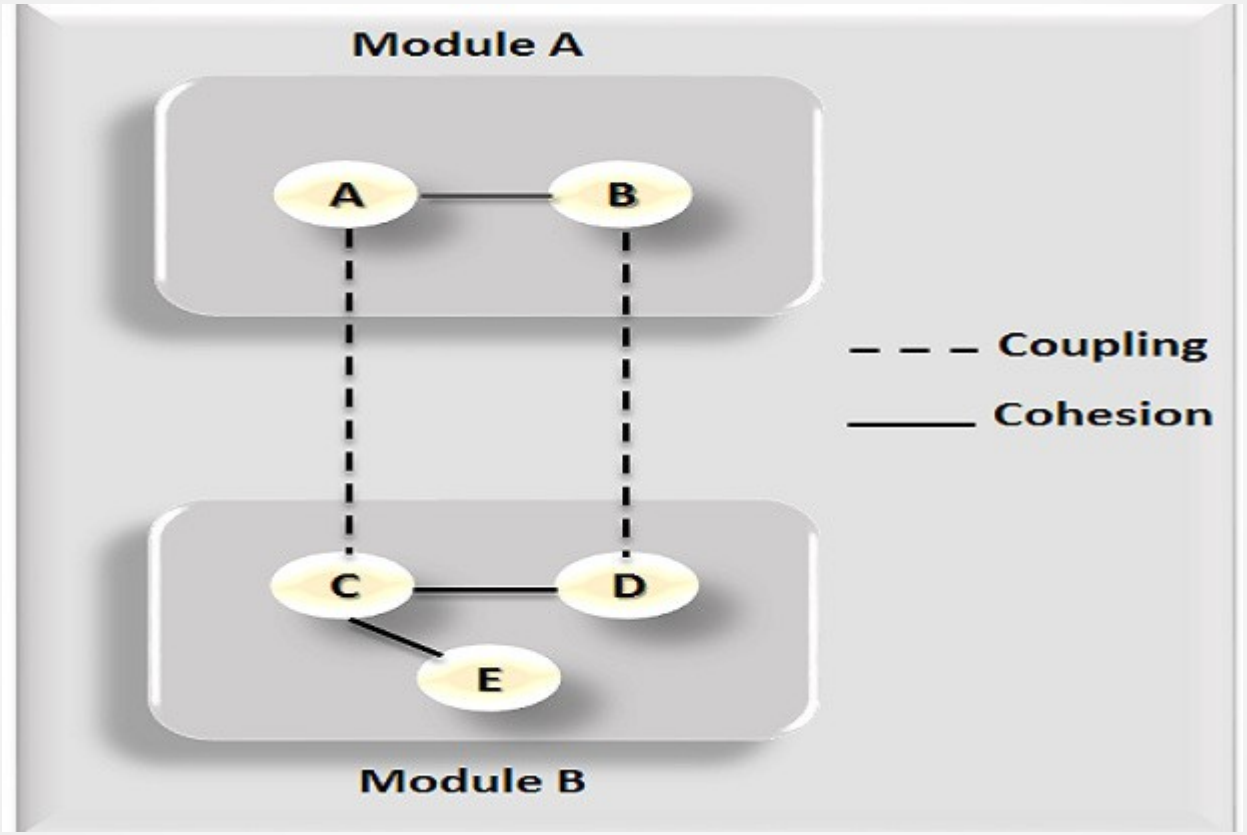


High Coupling
Low Cohesion



Low Coupling
High Cohesion




Module A
A
B
C
D
E
Module B
Coupling
Cohesion

components

- In programming and engineering disciplines, a component is an identifiable part of a larger program or construction.
- Usually, a component provides a particular function or group of related functions. In programming design, a system is divided into components that in turn are made up of modules.
- Component test means testing all related modules that form a component as a group to make sure they work together.
- Component is a collection of related classes that together provide a larger set of services.
- Component in a system might include applications, libraries, Activex controls, javaBean daemons and services.
- In the .net environment, most of the projects will require component development

Interfaces

- An interface is a definition of a set of services provided by a component or by a class.
- This alllows further encapsulation:the auther of a component can pubish just the interfaces to the component, completely hiding any implementation details.

  Interface is a collection of methods of a class or component. It specifies the set of services that may be provided by the class or component.

THE LANGUAGE OF OOAD-UML & BPMN

- The Unified Modeling Language is a graphical language for visualizing, specifying, constructing & documenting the artifacts of a software intensive system.
- Offers a standard way to write a system's blueprints, including conceptual things like business processes and system functions as well as concrete thing such as programming language statements, database schemas, and reusable software components.
- Provides a comprehensive notation for communicating the requirements, behavoiur, architecture, and realization of an oo design.
- Provides a sway to create and document a model of a system.

`

- Purpose is communcication; to be specific, it is to provide a comprehensive notation for communcating the requirements, architecture, implementation, deployment and states of a system.
- Everything is decribed in terms of obejcts: the actions that objects take, the relationships between the objects, the deployment of the objects, and the way the states of objects change in response to extrenal events.
- Static diagram: shows the structure of the system
- Dynamic diagram: shows the behavior of the system.

UML Diagrams

- Class Diagram: a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.
- Object Diagram: An object diagram in the Unified Modeling Language (UML), is a diagram that shows a complete or partial view of the structure of a modeled system at a specific time. Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams. Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment.

- Use-case Diagram: A use case diagram is a graphic depiction of the interactions among the elements of a system. A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.
- Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.

- Sequence Diagram: A sequence diagram is an interaction diagram that shows how objects operate with one another and in what order. It is a construct of a message sequence chart.A sequence diagram is a kind of interaction diagram, because they describe how—and in what order—a group of objects works together. Sequence diagrams are a popular dynamic modeling solution in UML because they specifically focus on the "lifelines" of an object and how they communicate with other objects to perform a function before the lifeline ends.
- Collaboration Diagram: The UML Collaboration diagram is used to model how objects involved in a scenario interact, with each object instantiating a particular class in the system. Objects are connected by links, each link representing an instance of an association between the respective classes involved. The link shows messages sent between the objects, and the type of message passed (synchronous, asynchronous, simple, balking, and timeout).
- A collaboration diagram resembles a flowchart that portrays the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time.

- Statechart Diagram: The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a component/object of a system.
- A Statechart diagram describes a state machine. State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events
- Activity Diagram: Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.
- Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.
- The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc

- Component Diagram: Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Now the question is, what are these physical aspects? Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node.
- Component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems.
- Deployment Diagram: Deployment diagram is a structure diagram which shows architecture of the system as deployment (distribution) of software artifacts to deployment targets.
- Artifacts represent concrete elements in the physical world that are the result of a development process. Examples of artifacts are executable files, libraries, archives, database schemas, configuration files, etc.

BPNM

- The Business process Management Initiative has developed a standard Business Process Modeling Notation.
- BPMN 1.0 specification was relased to the public in May 2004.
- To provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those process.

- The BPMN creates a standardized bridge for the gap between the business process design and process implementation
- It is an agreement between multiple modelling tools vendors, who had their own notations, to use a single notation for the benefit of end-user's understanding and training..
- BPMN defines a Business Process Diagram, which is based on a flowcharting technique tailored for creating graphical objects, which are activities and the flow controls that define their oder of performance.