Chapter IV

Understanding the Basics: Object oriented concepts

Compiled Yilkal B.

Chapter Outline

- OO concepts from structured point of view
- Abstraction,
- Encapsulation and information hiding
- inheritance
- Association
- Aggregation
- Collaboration
- Persistence
- Coupling
- Cohesion
- polymorphism
- Interfaces
- components
- Patterns

Object-Oriented Analysis

- ***Object–Oriented Analysis (OOA)*** *is the procedure of identifying software engineering requirements and developing software specifications in terms of a software system's object model, which encompasses of* ***interacting objects.***
- *The main difference between object-oriented analysis and other forms of analysis is that in* ***object-oriented approach****, requirements are organized around objects, which integrate both data and functions.*
- *They are modelled after real-world objects that the system interacts with.*
- *In traditional analysis methodologies, the two aspects functions and data - are considered separately.*
- *Grady Booch has defined OOA as, "****Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain****".*

Cntd…

The primary tasks in object-oriented analysis (OOA) are:

- *Identifying objects*
- ***Organizing the objects*** *by creating object model diagram*
- *Defining the internals of the objects, or* ***object attributes***
- *Defining the behavior of the objects, i.e., object actions*
- *Describing how the objects interact*

The common models used in OOA are ***use cases*** *and* ***object models.***

Object-Oriented Design

- ***Object–Oriented Design (OOD)*** *involves implementation of the conceptual model produced during object-oriented analysis.*
- *In OOD, concepts in the analysis model, which are* ***Technology independent****, are mapped onto implementing classes, constraints are identified and* ***interfaces*** *are designed,* ***resulting in a model for the solution domain****, i.e., a detailed description of how the system is to be built on concrete technologies.*
- *The implementation details generally include:*
  - *Restructuring the class data (if necessary),*
  - *Implementation of methods, i.e., internal data structures and algorithms,*
  - *Implementation of control, and*
  - *Implementation of associations.*

Cntd…

- *Object-oriented analysis and design (OOAD) is a software engineering approach that models a system as a group of interacting objects.*
- *Each object represents some **entity of interest** in the system being modeled, and is **characterized** by **its class**, its state (data elements), and its **behavior.***
- *Various models can be created to show the static structure, dynamic behavior, and **run-time deployment** of these collaborating objects.*
- *There are a number of different notations for representing these models, one such **model is Unified Modeling Language (UML).***

Object-Oriented Programming

- ***Object-oriented programming (OOP)*** *is a programming paradigm based upon objects (having both data and methods) that* ***aims to incorporate the advantages of modularity*** *and* ***reusability.***
- ***Objects,*** *which are usually instances of classes, are used to interact with one another to design applications and computer programs.*
- ***The important features of object oriented programming are:***
  - *Bottom up approach in program design*
  - *Programs organized around objects, grouped in classes*
  - *Focus on data with methods to operate upon object's data*
  - *Interaction between objects through functions*
  - *Reusability of design through creation of new classes by adding features to existing*
  - *Classes Some examples of object-oriented programming languages are C++, Java, C#, Perl, Python, Ruby, and PHP.*

The object model

Object oriented development *offers a different model from the traditional software development approach, which is based on functions and procedures.*

- *An Object-Oriented environment, software is a collection of discrete objects that encapsulate their data and the functionality to model real world "Objects".*
- *Object are defined,* ***it will perform their desired functions and seal them off in our mind like black boxes.***
- *The object- Oriented life cycle encourages a* ***view of the world as a system of cooperative and collaborating agents.***
- *An objective orientation producers system that are easier evolve, move flexible more robust, and more reusable than a top-down structure approach.*
- *An object orientation* ***allows working at a higher level of abstraction.***
- *It provides a seamless transition among different phases of software development.*
- ***It encourages good development practices.***
- *It promotes* ***reusability.***

Cntd…

- The object model ***visualizes the elements*** in a software application in terms of objects.
- The basic concepts and terminologies of object–oriented systems.

Objects and Classes

- The concepts of objects and classes are intrinsically linked with each other and form the foundation of object–oriented paradigm.
- ***Object*** An object is a real-world element in an object–oriented environment that may have a physical or a conceptual existence. Each object has:
  - Identity that distinguishes it from other objects in the system.
  - ***State*** that determines the characteristic properties of an object as well as the values of the properties that the object holds.
  - ***Behavior*** that represents externally visible activities performed by an object in terms of changes in its state.

Class

- ***A class*** *represents a collection of objects having same characteristic properties that exhibit common behavior.*
- *It gives the* ***blueprint*** *or description of the objects that can be created from it.*
- *Creation of an object as a member of a class is called instantiation. Thus, object is an instance of a class.*
- ***The constituents of a class are:***

  - *A set of attributes for the objects that are to be instantiated from the class.*
  - *Generally, different objects of a class have some difference in the values of the attributes.* ***Attributes*** *are often referred as* ***class data****.*
  - *A set of operations that represent the behavior of the objects of the class.* ***Operations*** *are also referred as* ***functions or methods****.*

Cntd…

- *E.g., Let us consider a simple class,* ***Circle,*** *that represents the geometrical figure circle in a two– dimensional space. The attributes of this class can be identified as follows:*
  - *x–coord, to denote x–coordinate of the center*
  - *y–coord, to denote y–coordinate of the center a, to denote the radius of the circle*
- *Some of its* ***operations can be defined as follows:***
  - *findArea(), method to calculate area*
  - *findCircumference(), method to calculate circumference*
  - *scale(), method to increase or decrease the radius*
- *During* ***instantiation,*** *values are assigned for at least some of the attributes. If we create* ***an object my_circle****, we can assign values like x-coord : 2, y-coord : 3, and a :4 to depict its state.*

Encapsulation and Data Hiding

Encapsulation

- *It 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 protecting an object's data is called* ***data hiding*** *or* ***information hiding****.*

cntd…

- *Example*
- *In the class Circle, data hiding can be incorporated by making attributes **invisible from outside** the class and adding two more methods to the class for accessing class data, namely:*
  - *setValues(), method to assign values to x-coord, y-coord, and a*
  - *getValues(), method to retrieve values of x-coord, y-coord, and a*
- *Here the **private** data of the object my_circle cannot be accessed directly by any method that is not encapsulated within the class Circle.*
- *It should instead be accessed through the **methods setValues()** and **getValues()**.*

Message Passing

- *Any application requires a number of objects interacting in a harmonious manner.*
- *Objects in a system may communicate with each other using message passing. Suppose a system has two objects: obj1 and obj2.*
- *The object obj1 sends a message to object obj2, if obj1 wants obj2 to execute one of its methods.*
- ***The features of message passing are:***
  - *Message passing between two objects is generally* ***unidirectional.***
  - *Message passing* ***enables all interactions between objects.***
  - *Message passing essentially involves invoking class methods.*
  - *Objects in* ***different processes*** *can be involved in message passing.*

Inheritance

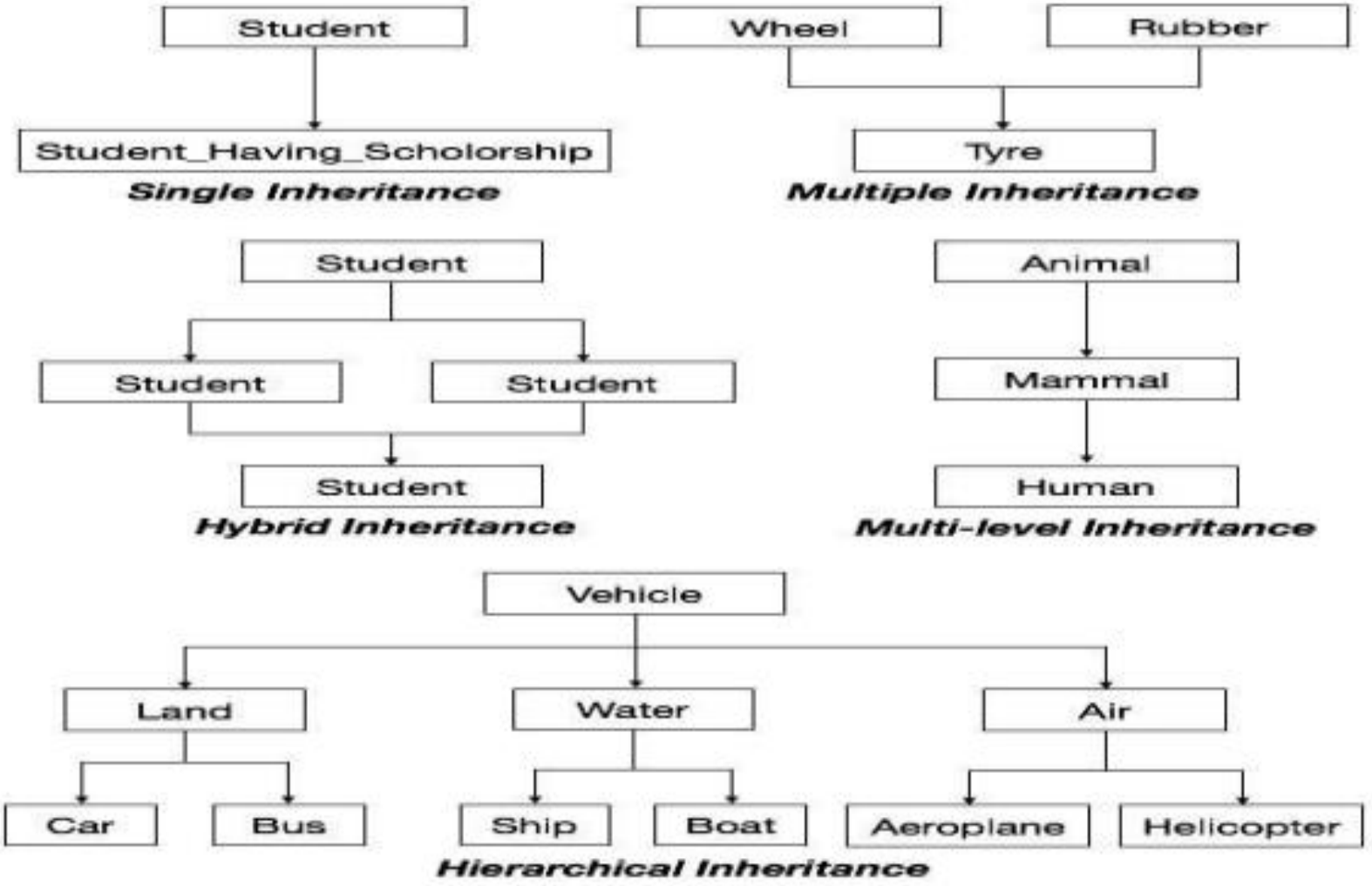
- ***Inheritance*** *is the mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities.*
- *The existing classes are called the* ***base classes/parent***
- ***classes/super-classes****, and the* ***new classes*** *are called the* ***derived classes/child 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.*
- *E.g., From a class Mammal, a number of classes can be derived such as Human, Cat, Dog, Cow, etc.*

Types of inheritance: *The following are the types inheritance*

- ***Single Inheritance*** *: A subclass derives from a* ***single super-class****.*
- ***Multiple Inheritance :*** *A subclass derives from* ***more than one*** *super-classes.*
- ***Multilevel Inheritance*** *: A subclass derives from a super-class which in turn is derived from another class and so on.*
- ***Hierarchical Inheritance*** *: A class has a number of subclasses each of which may have subsequent subclasses, continuing for a number of levels, so as to form a* ***tree structure****.*
- ***Hybrid Inheritance :*** *A combination of multiple and multilevel inheritance. So, as to form a* ***lattice structure.***

Cntd…

The following figure depicts the examples of different types of inheritance.



Generalization and Specialization

- *Generalization and specialization represent a hierarchy of relationships between classes, where subclasses inherit from super-classes.*

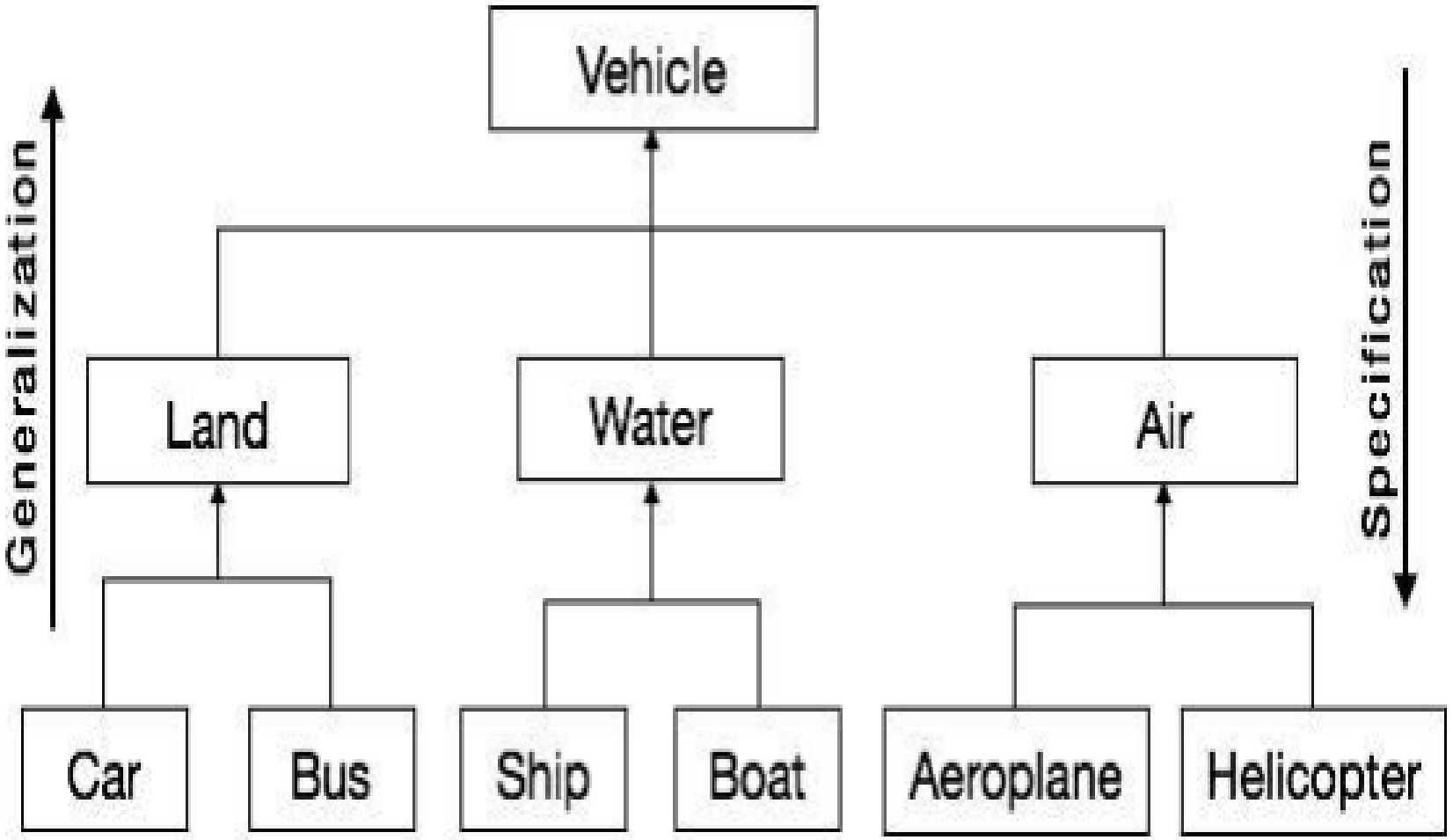
Generalization: *In the generalization process, the common characteristics of classes are combined to form a class in a higher level of hierarchy, i.e.,* ***subclasses are combined to form a generalized superclass.***

- *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".*

Specialization: *Specialization is the reverse process of generalization.*

- *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.***

Cntd…


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

Links and Association

Link: *it represents a* ***connection*** *through* ***which an object collaborates with other objects.***

- *It is a physical or conceptual connection between objects.*
- *Through a link, one object may invoke the methods or navigate through another object.*
- *A link depicts the relationship between two or more objects.*

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.*

Cntd…

- ***Degree of an Association*** : *Degree of an association denotes the number of classes involved in a connection. Degree may be unary, binary, or ternary.*
  - *A **unary** relationship connects objects of the same class.*
  - *A **binary** relationship connects objects of two classes.*
  - *A **ternary** relationship connects objects of three or more classes*

Cardinality Ratios of Associations

- *Cardinality of a binary association denotes the number of instances participating in an*
- *association. There are three types of **cardinality ratios**, namely:*
  - ***One–to–One :*** *A single object of class A is associated with a single object of class B.*
  - ***One–to–Many :*** *A single object of class A is associated with many objects of class B.*
  - ***Many–to–Many*** *: An object of class A may be associated with many objects of class B and conversely an object of class B may be associated with many objects of class A*

Aggregation or Composition

- ***Aggregation or composition*** *is a relationship among classes by which Aggregation or Composition 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.***
- *E.g., In the relationship, "a car has–a motor", car is the whole object or the aggregate, and the motor is a "part–of" the car.*
- *Aggregation may denote:*
  - ***Physical containment:*** *Example, a computer is composed of monitor, CPU, mouse, keyboard, and so on.*
  - ***Conceptual containment:*** *Example, shareholder has–a share.*

Benefits of Object Model

- *Now that we have gone through the core concepts pertaining to object orientation, it would be worthwhile to note the advantages that this model has to offer.*
- ***The benefits of using the object model are:***
  1. *It helps in **faster** development of software.*
  2. *It **is easy to maintain**. Suppose a module develops an error, then a programmer can fix that particular module, while the other parts of the software are still up and running.*
  3. *It **supports relatively** hassle -free upgrades.*
  4. *It enables reuse of objects, designs, and functions.*
  5. *It **reduces development risks,** particularly in integration of complex systems..*

Principles of Object-Oriented Systems

- The conceptual framework of object–oriented systems is **based upon the object model**.
- **There are two categories** of elements in an object-oriented system:

I. Major Elements:

By major, it is meant that if a model **does not have** any one of these elements, it stops to be object oriented.

The four major elements are:

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

II. Minor Elements:

- By minor, it is meant that these elements are useful, but not crucial part of the object model.
- The three minor elements are:
  - Typing
  - Concurrency
  - Persistence

Abstraction

- *Abstraction means to focus on the essential features of an element or object in OOP, **ignoring its extraneous properties**.*
- *The essential features are relative to the context in which the object is being used.*
- ***Grady Booch has defined abstraction as follows:*** *"An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects, relative to the perspective of the viewer."*
- ***E.g.,*** *When a class Student is designed, the attributes enrolment_number, name, course, and address are included while characteristics.*
- *like pulse rate and size of shoe are eliminated, since they are irrelevant in the perspective of the educational institution.*

Cntd…

- ***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.***

Modularity: *is the process of decomposing a problem (program) into a set of modules so as to* ***reduce the overall complexity*** *of the problem.*

Booch has defined modularity as: *"Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules."*

- *Modularity is essentially linked with encapsulation.*
- *Modularity can be visualized as a way of mapping encapsulated abstractions into real, physical modules having high cohesion within the modules and their inter–module interaction or coupling is low.*

Abstraction vs Encapsulation

Abstraction	Encapsulation
Abstraction in OOP solves the issues at the design level.	Encapsulation solves it implementation level.
Abstraction in Programming is about **hiding unwanted details** while showing most essential information.	Encapsulation means binding the code and data into a single unit.
Data Abstraction in Java allows focusing on what the information object must contain	Encapsulation means hiding the internal details or mechanics of how an object does something for security reasons.

Cntd…

Hierarchy

- In Grady Booch's words, "**Hierarchy is the ranking or ordering of abstraction**".
- Through hierarchy, a system can be made up of **interrelated subsystems**, which can have their **own subsystems** and so on **until the smallest level components are reached.**
- It uses the principle of "**divide and conquer**".
- Hierarchy allows code reusability.
- The two types of hierarchies in OOA are:

I. IS–A" hierarchy: It defines the hierarchical relationship in inheritance, whereby from a **super-class**, a number of **subclasses may** be derived which may again have **subclasses and so** on.

Cntd...

- *For example, if we develop a class Rose from a class Flower, we can say that a rose "is–a" flower.*

II. "PART–OF" hierarchy : *It defines the hierarchical relationship in aggregation by which a class may be composed of other classes.*

- *For example, a computer system is composed of hardware and software. It can be said that a software is a "part–of" computer.*

Persistence: ***An object occupies a memory space and exists for a particular period of time.***

- *In traditional programming, the lifespan of an object was typically the lifespan of the execution of the program that created it.*
- *In files or databases, the* ***object lifespan is longer than the duration of the process*** *creating the object.*
- *This property by which an object continues to exist even after its creator stops to exist is known* ***as persistence.***

Coupling

- *coupling is the degree of interdependence between software modules; a measure of how closely connected two routines or modules are.*
- *the strength of the relationships between modules.*

I. Procedural programming

- *A module here refers to a subroutine of any kind.*
- ***Content coupling (high):*** *Content coupling is said to occur when one module uses the code of another module, for instance a branch. This violates information hiding: a basic software design concept.*
- ***Common coupling:*** *it is said to occur when **several modules have access to the same global data**. But it can lead to uncontrolled error propagation and unforeseen side-effects when changes are made.*

Cntd…

- ***External coupling:*** *it occurs when two modules share an externally imposed data format, communication protocol, or device interface.*
- ***Control coupling:*** ***it is one module controlling the flow of another****, by passing it information on what to do (e.g., passing a what-to-do flag).*
- ***Stamp coupling (data-structured coupling):*** *it occurs when modules share a composite data structure and use only parts of it, possibly different parts (e.g., passing a whole record to a function that needs only one field of it).*
- ***Data coupling:*** *it occurs when modules* ***share data through****, for e.g., parameters.*

Each datum is an elementary piece, and these are the only data shared (e.g., passing an integer to a function that computes a square root).

Cntd…

II. Object-oriented programming

- ***Subclass coupling:*** *Describes the relationship between a child and its parent. The child is connected to its parent, but the parent is not connected to the child.*
- ***Temporal coupling:*** *It is when two actions are bundled together into one module just because they happen to occur at the same time.*
- ***Dynamic coupling:*** *The goal of this type of coupling is to provide a run-time evaluation of a software system.*
- ***Semantic coupling:*** *This kind of coupling considers the* ***conceptual similarities between software entities using****, for example, comments and identifiers and relying on techniques such as latent semantic indexing (LSI).*

Cntd…

- ***Logical coupling:*** *(or evolutionary coupling or change coupling) activities the release history of a software system to find change patterns among modules or classes: e.g., entities that are likely to be changed together or sequences of changes (a change in a class A is always followed by a change in a class B).*

- ***Disadvantages of tight coupling:***

Tightly coupled systems has the following disadvantages:

- *A change in one module usually forces a ripple effect of changes in other modules.*
- *Assembly of modules **might require more effort** and/or time due to the increased inter-module dependency.*
- *A particular module **might be harder to reuse** and/or test because dependent modules must be included.*

Cohesion

- ***Cohesion*** *refers to the degree to which the elements* ***inside a module belong*** *together.*
- *In one sense, it is a measure of the strength of relationship between t****he methods and data of a class*** *and some unifying purpose or concept served by that class.*
- *Cohesion is an ordinal type of measurement and is usually described as "****high cohesion****" or "****low cohesion****".*
- *Modules with* ***high cohesion*** *tend to be* ***preferable,*** *because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability.*
- *In contrast,* ***low cohesion is*** *associated with undesirable traits such as being difficult to maintain, test, reuse, or even understand.*

Cntd…

- ***Cohesion*** *is often contrasted with coupling, a different concept.*
- ***High cohesion*** *often correlates with* ***loose coupling****, and* ***vice versa****.*
- *"Coupling" describes the* ***relationships between modules****, and "cohesion" describes the* ***relationships within them****.*
- *cohesion can be ranked on a scale of the weakest (****least desirable****) to the strongest (most desirable) as follows.*
- ***Coincidental cohesion*** *[elements are in the same module for no particular reason]*
- ***Logical cohesion*** *[elements perform logically related tasks]*
- ***Temporal cohesion*** *[elements must be used at approximately the same time]*
- ***Communication cohesion*** *[elements share I/O]*

Cntd…

- ***Sequential cohesion*** *[elements must be used in a particular order]*
- ***Functional cohesion*** *[elements cooperate to carry out a single function]*
- ***Data cohesion*** *[elements cooperate to present an interface to a hidden data structure*
- *One can often estimate the degree of cohesion within a module by writing a brief statement of the module's purpose.*

The following tests are suggested :

- *If the sentence that describes the purpose of the module is a* ***compound sentence*** *containing a* ***comma or more than one verb****, the module is probably performing more than one function; therefore, it probably has* ***sequential or communicational binding*** *[or even less: temporal, logical, or coincidental]*

Cntd…

- *If the sentence contains words relating to **time,** such as "first," "next," "then," "after," "when," or "start," the module probably has **sequential or temporal binding**. An example is "**Wait for the instant teller customer to insert a card**, then prompt for the personal identification number."*
- *If the predicate of the sentence **does not contain a single, specific object** following the verb, the module is probably **logically bound.** For example, "Edit all data" has logical binding; "Edit source data" may have functional binding.*
- *If the sentence contains words such as "**Initialize**" or "**Clean up,**" the module probably has **temporal binding**.*

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.*
- *Polymorphism is an OOP concept that refers* ***to the ability of a variable, function, or object to take on multiple forms.***
- *In a programming language exhibiting polymorphism,* ***class objects belonging to the same hierarchical tree*** *(inherited from a common parent class) may have functions with the same name, but with different behaviors.*

cntd…

- *Types of polymorphism:*
  - ***Compile time polymorphism***

    Example: Method overloading
  - ***Runtime polymorphism***

    Example: Method overriding

Advantages of polymorphism:

- *It helps programmers reuse code and classes once written, tested, and implemented.*
- *A single variable name can be used to store variables of multiple data types (float, double, long, int, etc).*
- *It helps compose powerful, complex abstractions from simpler ones.*

Cntd…

Interfaces

- All data should be hidden within a class.
- make all data attributes private
- provide public methods (accessor methods) to get and set the data values

e.g. Grade information is usually confidential, hence it should be kept private to the student. Access to the grade information should be done through interfaces, such as setGrade() and getGrde()

Components: Components are **self contained entities that provide service to other components or actors.**

- The **deployment diagram** focuses on the allocation of components to nodes and provides a high-level view of each component.

Pattern (reading Assignment).

End of chapter Three

Any Question?