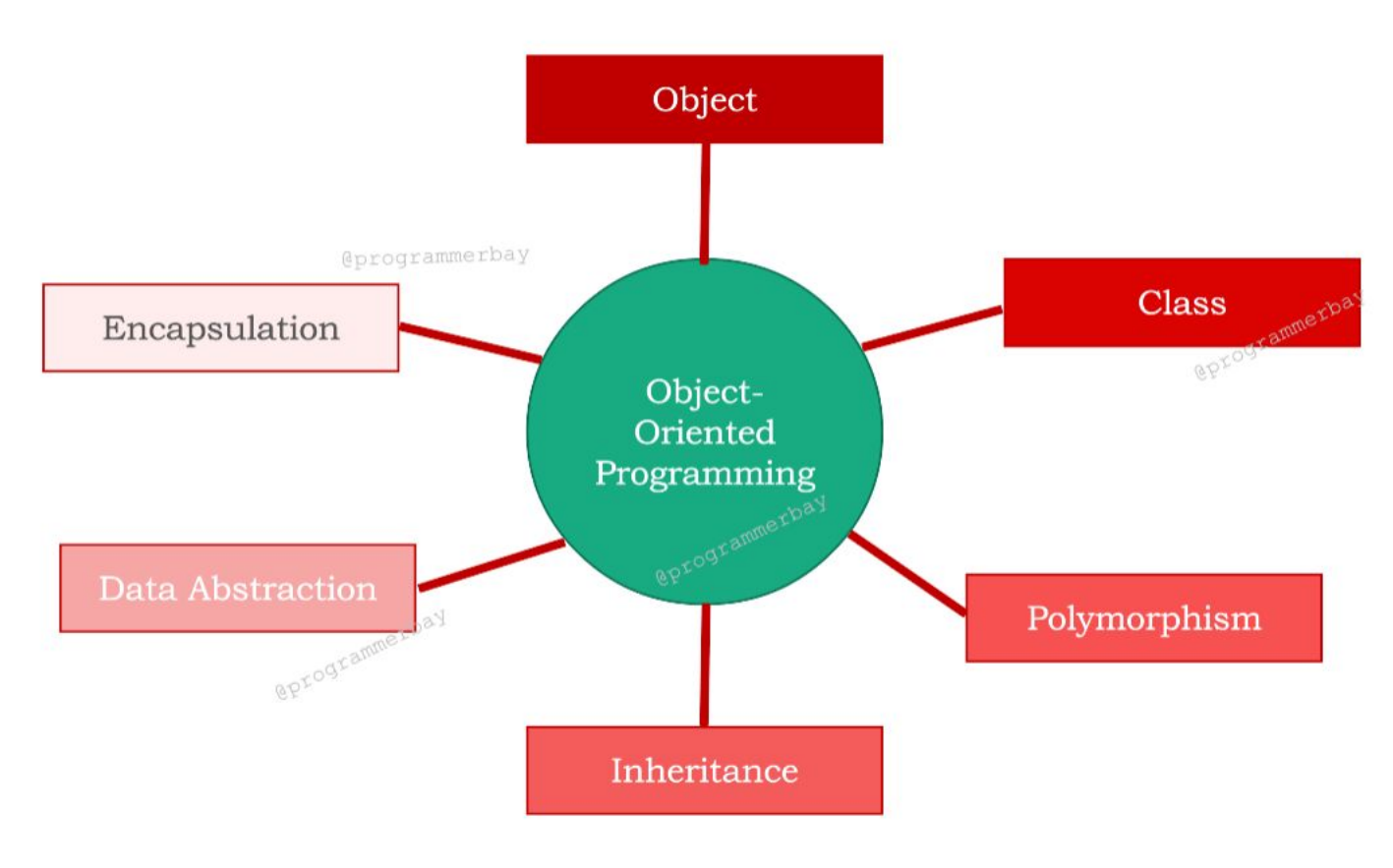
Unit 6. Introduction to Object-Oriented Development

Basic Characteristics of Object-Oriented Systems; Object-Oriented System Analysis and Design (OOSAD); Introduction to Unified Modeling Language, Structural (Class, Object, Deployment, and Component Diagram) and Behavioral (Use Case, Activity, Sequence, and State) Diagrams

Object-Oriented Programming

- Object-Oriented Programming or OOPs is a programming paradigm that revolves around the concept of object, which contains properties and methods. It combines a group of related attributes and behaviour within a single unit named class, that enhances program design structure and code readability.
- Further, it also resolves drawbacks of Procedural programming i.e code complexity, unusable code.

Characteristics of OOP



- Objects
  - Objects are like a signature of a class. An object is an instance of a class without an object, no memory is allocated to a class's data members or member function. With an object of a class, we can access the data members and member functions that can be accessed (as per the private, public, protected accessibility scope).
- Classes:
  - A class is the user-defined data type and the main building block of object-oriented programming. It is an identifiable entity that can have some descriptive properties. It is a user-defined data type that holds data members and member functions in a single unit. It is like a blueprint of an object.
  - For example, consider an entity "Laptop" , what attributes, you can think of? RAM, OS, memory, manufacturer name, model name and so on.

Inheritance:

- The ability to inherit the properties of one class to another, or inherit the properties from a base class to an inherited class is known as the concept of Inheritance. With the help of inheritance, we can use the data members and member functions of a class to another.
  - It enables a class to acquire or get the properties from another class
  - It makes code reusable
  - It makes easier to add new features or methods to a class
  - It provides an overriding feature which allows a child class to have a specific implementation of a method defined in the parent class
  - A class that is inherited by other classes is termed as super class or parent class or base class, whereas a class that extends another class is termed as sub-class or child class
- Features of Inheritance :

  • Code Reusability • Ease to add new feature • Overriding

- Polymorphism is best defined in one statement i.e., "Polymorphism means one name many forms".
- It can be best explained with an example with a comparison to C language; in C language there was one limitation that we can not use the already used function name, but C++ provides us a new feature of Polymorphism, by which we can use the same function name again and again with different signatures.

- **Data Abstraction** means hiding the background details and provide only essential details. one of the most common examples regarding this feature is Television, in television we control it with help of a remote and, know its external or the features to be used most whereas we don't know the internal working of Television.
- **Encapsulation** means binding up data under a single unit. A class is one of the main implementations of this concept. It can be viewed as a wrapper that restricts or prevents properties and methods from outside access. With the help of access modifiers such as private, protected, and public keywords, one can control the access of data members and member functions

- Dynamic Binding
  - Dynamic Binding which is also known as Late binding or run-time binding, is a process of executing the part of the code at runtime. Sometimes we need to access some part at the runtime if we need then we can use this approach. We use virtual functions to achieve Dynamic Binding.
  - Method Overriding is a perfect example of dynamic binding as in overriding both parent and child classes have same method and in this case the type of the object determines which method is to be executed. The type of object is determined at the run time so this is known as dynamic binding.
- Message Passing:
  - In this, objects pass the message to each other in order to contact each other. Like when we want 2 or multiple objects to contact each other it is possible with the OOP.

Object-Oriented Analysis And Design (OOAD)

- 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
- It's a structured method for analyzing, designing a system by applying the object-orientated concepts, and developing a set of graphical system models during the development life cycle of the software.

Object-Oriented Analysis

- In the object-oriented analysis, we …
  - **Elicit** requirements: Define what does the software need to do, and what's the problem the software trying to solve.
  - **Specify** requirements: Describe the requirements, usually, using use cases (and scenarios) or user stories.
  - **Conceptual** model: Identify the important objects, refine them, and define their relationships and behavior and draw them in a simple diagram.

Object-Oriented Design

- The analysis phase identifies the objects, their relationship, and behavior using the conceptual model (an **abstract** definition for the objects).
- While in design phase, we describe these objects (by creating class diagram from conceptual diagram — usually mapping conceptual model to class diagram), their attributes, behavior, and interactions.

- In the object-oriented design, we …
  - **Describe the classes** and their relationships using class diagram.
  - **Describe the interaction** between the objects using sequence diagram.
  - **Apply** software design principles and design patterns.
  - A class diagram gives a visual representation of the classes you need. And here is where you get to be really specific about object-oriented principles like inheritance and polymorphism.

- We may develop a model to represent the system from different perspectives.
  - **External**, where you model the context or the environment of the system.
  - **Interaction**, where you model the interaction between components of a system, or between a system and other systems.
  - **Structural**, where you model the organization of the system, or the structure of the data being processed by the system.
  - **Behavioral**, where you model the dynamic behavior of the system and how it respond to events.

Unified Modeling Language (UML)

- UML is a general purpose, developmental, modeling language in the field of software engineering that intended to provide a standard way to visualize the design of system.
- UML was motivated by the desire to standardize the disparate notational systems and approaches to software design.
- UML offers a way to visualize a system's architectural blueprints in a diagram, including : activities, components, How system run ?, How components or entities interact with each other ? and so on.

Unified Modeling Language (UML)

- The unified modeling language become the standard modeling language for object-oriented modeling. It has many diagrams, however, the most diagrams that are commonly used are:
  - **Use case diagram**: It shows the interaction between a system and it's environment (users or systems) within a particular situation.
  - **Class diagram**: It shows the different objects, their relationship, their behaviors, and attributes.

Unified Modeling Language (UML)

- **Sequence diagram**: It shows the interactions between the different objects in the system, and between actors and the objects in a system.
- **State machine diagram**: It shows how the system respond to external and internal events.
- **Activity diagram**: It shows the flow of the data between the processes in the system.