Object Oriented Programming

Part 1

OOP Paradigm

- It is a way of thinking / approaching a problem.
- It is a way of looking at something
- It is a style of computer programming (a way of building the structure and elements of computer programs).
- It influences the programming language and other things built on top of the language, such as libraries, frameworks, and common styles and patterns of programming.

Types of Programming Paradigms

- Imperative Paradigm
    1. Procedure Oriented Programming
    2. Object Oriented Programming
- Declarative Paradigm
    1. Functional Programming
    2. Logical Programming

Multi -Paradigm Language

- It is the language that supports more than one paradigm
- Example : C++ , C#, …
    - Supports Procedural and Object Oriented Programming

Advantages of OOP

1. Improved software-development productivity.
2. Improved software maintainability.
    - Since the design is modular, part of the system can be updated in case of issues without a need to make large-scale changes.
3. Faster development
    - Reuse enables faster development. OOP languages come with rich libraries of objects
4. Lower cost of development
    - reuse of software also lowers the cost
5. Higher-quality software
    - Faster development of software and lower cost of development allows more time and resources to be used in the verification of the software.

Disadvantages of OOP

1. Difficult initial learning process
   - The thought process involved in OOP may not be natural for some people, and it can take time to get used to it.
2. Larger program size
   - More lines of code than procedural programming
3. Slower programs
   - Object-oriented programs are slower than procedure based programs
4. Not suitable for all types of problems

Basis of OOP

- View the problem world as a set of objects and communication between the objects.
- Entire world around as can be thought as a set of objects.
- Objects have characteristics and they can perform some functions.
- Similar objects may be grouped together.

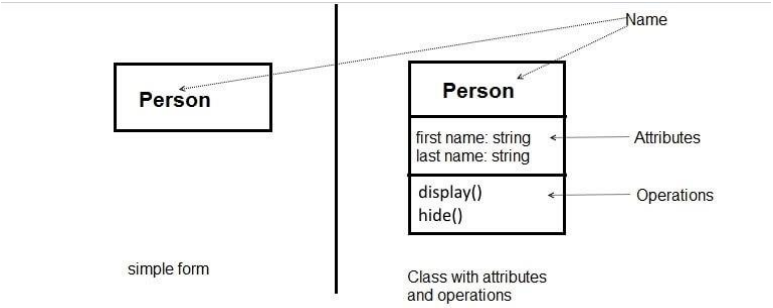
Class

- Depicts the set of similar objects.
- It distinguishes one type of object from another type.
- It is a User defined data type.
- It comprised of attributes and functions.
  - Attributes defines the properties of the object and
  - Functions describes "What an object is capable of doing"
- It is like a blueprint of a house and it indicates how the data and functions are used by the objects.
- It is like an object constructor for creating objects.
- It is the main building block of an application that is based on OOP.

Object

- In programming perspective, It is an instance of a class.
- It is created from a class.

Representation of the Class concept by UML



Terminologies

Class name	Attributes	Methods	Class	Object
	Data	Actions	Pattern	Instance
	Data Members	Member Functions	Blueprint	Example
	Variables	Functions	Object	
	Properties	Behaviors	Design	
	States	Characteristics	Template	
	Nouns	Functionalities	Data Structure	
		Verbs	Description	
		Operations		

Unified Modeling Language (UML)

What is UML ?

- It is a general-purpose modeling language used to visualize the system.
- It is a graphical language that is standard to the software industry for :
  - Specifying, visualizing, constructing, and documenting the artifacts of the software systems
  - Business modeling.

Benefits of UML

- It simplifies complex software design.
- It reduces thousands of words of explanation in a few graphical diagrams that may reduce time consumption to understand.
- It makes communication more clearly and more real.
- It helps to acquire the entire system in a view.
- It becomes very much easy for the software programmer to implement the actual demand once they have a clear picture of the problem.

Types of UML diagrams

Structural UML diagrams

1. Class diagram
2. Package diagram
3. Object diagram
4. Component diagram
5. Composite structure diagram
6. Deployment diagram

Behavioral UML diagrams

1. Activity diagram
2. Sequence diagram
3. Use Case diagram
4. State diagram
5. Communication diagram
6. Interaction overview diagram
7. Timing diagram

UML Diagrams Drawing Tools / Diagramming Tools (online/offline)

1. Visio
2. EdrawMax
3. LucidChart
4. Gleek.io
5. Diagrams.net
6. Cacoo
7. Gliffy
8. ……

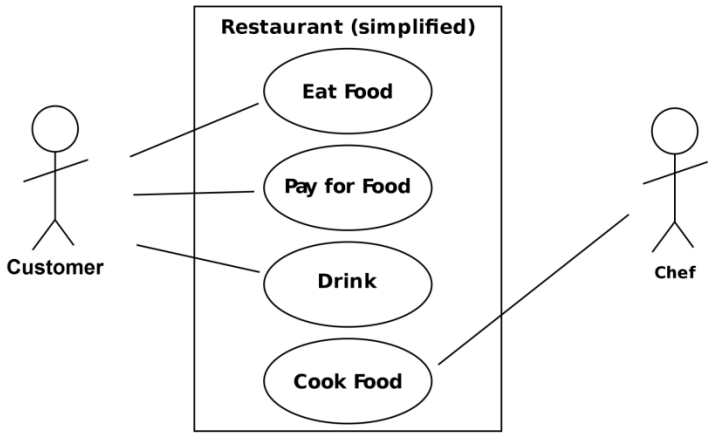
Use Case Diagram

- It determines the requirements of the system.
- It describes what the system should do.
- It provides a basis for testing to ensure that the system works as intended.

Use Case Diagrams elements

1. Use Cases
   A use case describes a function that a system performs to achieve the user's goal.
2. Actors / Stakeholders
   An actor represents a role of a user that interacts with the system that you are modeling. The user can be a human user, an organization, a machine, or another external system.
3. Relationships
   A relationship is a connection between model elements.

Use Case Diagram (Sample)



Class diagram

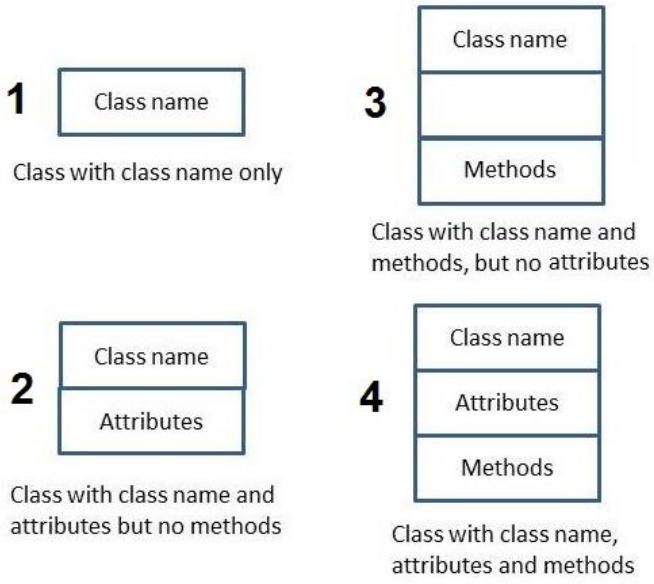
- It is a main building block of every object-oriented modeling.
- It can be used to show the classes, relationships, interface, association, and collaboration.
- It has an appropriate structure to represent the classes, inheritance, relationships, and everything that OOP have in their context.
- It describes various kinds of objects and the static relationship between them.
- It used for describing structure and behavior of use cases.
- It provides a conceptual model of the system in terms of entities and their relationships.
- It used for capturing the system requirements and showing the system stakeholders.
- Detailed class diagrams are used for developers.

- Each class is represented by a rectangle subdivided into three compartments Name, Attributes, and Operations

Class Diagram Syntax

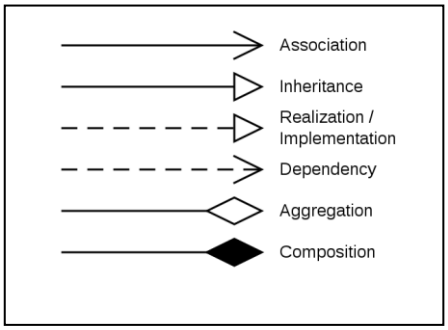
- There are three types of modifiers that are used to decide the visibility of attributes and operations :
  + is used for public visibility (for everyone)
  # is used for protected visibility (for friend and derived)
  – is used for private visibility (for only me)

Various representations of the Class Icons in Class Diagram



Class Diagram Relationships

- Classes are interrelated to each other by different types of logical connections.
- Types of logical connections that are possible in UML :
  1. Association
  2. Directed Association
  3. Reflexive Association
  4. Multiplicity
  5. Aggregation
  6. Composition
  7. Inheritance/Generalization
  8. Realization



Class Diagram Relationships (Association)

- It is a broad term that encompasses any logical connection or relationship between classes.
- It represented by a line.

Class Diagram Relationships (Directed Association)

- Refers to a directional relationship.
- Represented by a line with an arrowhead.
- The arrowhead depicts a container-contained directional flow.

Class Diagram Relationships (Reflexive Association)

- This occurs when a class may have multiple functions or responsibilities.
- For example, a staff member working in an airport may be a pilot, aviation engineer, a ticket dispatcher, a guard, or a maintenance crew member.
- If the maintenance crew member is managed by the aviation engineer there could be a managed by relationship in two instances of the same class.

Class Diagram Relationships (Multiplicity)

- It is the active logical association when the cardinality of a class in relation to another is being depicted.
- For example, one commercial airplane may contain zero to many passengers.
- The notation 0..* in the diagram means “zero to many”.

Class Diagram Relationships (Aggregation)

- It refers to the formation of a particular class as a result of one class being aggregated or built as a collection.
- For example, the class “library” is made up of one or more books, among other materials.
- In aggregation, the contained classes are not strongly dependent on the lifecycle of the container. In the same example, books will remain so even when the library is dissolved.
- To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

Class Diagram Relationships (Composition)

- It is very similar to the aggregation relationship. with the only difference being its key purpose of emphasizing the dependence of the contained class to the life cycle of the container class.
- That is, the contained class will be obliterated when the container class is destroyed.

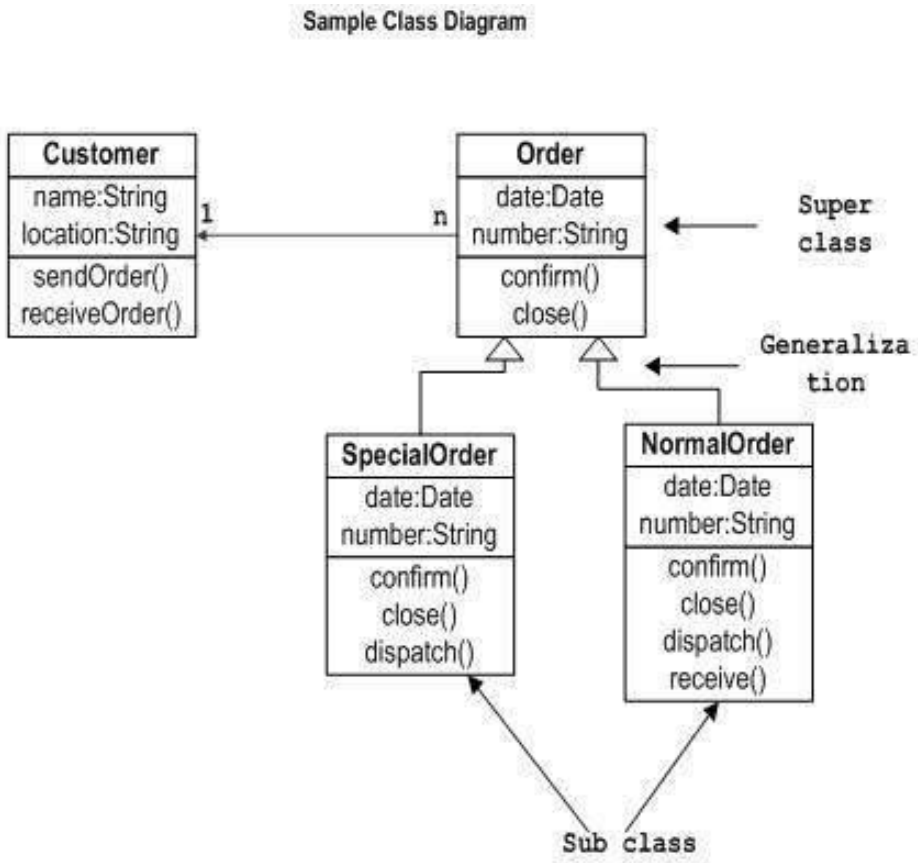
- For example, a shoulder bag's side pocket will also cease to exist once the shoulder bag is destroyed.

Class Diagram Relationships (Inheritance / Generalization)

- It refers to a type of relationship where the child class is a specific type of the parent class.
- Represented by a solid line from the child class to the parent class is with an unfilled arrowhead.

Class Diagram Relationships (Realization / Implementation)

- It denotes the implementation of the functionality defined in one class by another class.
- Represented by a broken line with an unfilled solid arrowhead is drawn from the class that defines the functionality to the class that implements the function.
- In the example, the printing preferences that are set using the printer setup interface are being implemented by the printer.



Class Diagram Example: Order System