Topic 4: object oriented fundamental and analysis

❖Some Terminologies

- **Object**

• Real world entity that may have physical or conceptual existence.
• Each object consists of data and a set of functions.
• Each object has **identity, behavior and state**.
• Identity is the instance ID of an object that distinguishes it from other objects in the system.

- **Class**

• Description of a set of objects that share the same characteristic properties and exhibit common behavior.
• Object can be created as a member of a class by instantiation.

```
class Dog:
 def __init__(self, name):
      self.name = name  #state
 def bark(self): # Behavior
     print(f"{self.name} is barking.")
  dog1 = Dog("Buddy") # Creating an object (identity)
  dog2 = Dog("Max")

print(id(dog1))  # Unique identity of dog1
print(id(dog2))  # Unique identity of dog2

  dog1.bark() # Behavior
  dog2.bark()
```

- **Method**

- Method is a mean by which objects can manipulate data.
- The operations supported by an object are called its methods

- **Messages**

- A message is a method call from one object to another.

- **Abstraction**

- Handle complexity.
- Extracting essential properties relevant to a particular purpose and omitting the unnecessary details.
- It helps in code reuse.

- **Encapsulation and data hiding**

- Binding both attributes and methods together within a class.

- Data hiding means giving data access only by its class methods and prevented from direct access outside.
- Encapsulation ensures data hiding.
- It focuses upon the implementation that gives the behavior of an object.
- Data can be accessed from outer object through message passing only.

- **Inheritance**

- It is the process of defining new classes out of existing classes by extending and refining its capabilities.

- **Polymorphism**

- Allows using operations in different ways, depending upon the instance they are operating upon.
- Different objects have common external interface but differ in internal structures.

❖ Defining Models

- “Model” refers to an abstraction of a system that captures its essential characteristics, structure, behavior, and relationships among its components.
- Models are used to understand, describe, and communicate about the system being developed.
- Serve as blueprints for designing and implementing the system.

Types of models

1. **Use Case Model**: Describes how users interact with the system, focusing on their goals and actions.
2. **Class Model**: Illustrates the structure of the system by identifying classes, their attributes, methods, and relationships.

3. **Behavioral Model**: Depicts the system's dynamic behavior over time through diagrams like state, activity, sequence, and collaboration diagrams.
4. **Component Model**: Describes how software components are organized and interact within the system.
5. **Deployment Model**: Specifies how software components are deployed on hardware infrastructure, including servers, nodes, and networks.
6. **Interaction Model:** focuses on illustrating the interactions among various components within the system.

❖ Requirement Process

- Requirement process is a systematic approach to find, document, organize and track the needs of the users and response on changing requirements of a system.
- Requirements are the aspects that the system must conform.
- A requirement is a statement describes either

➢As aspect what the proposed system must do.

➢A constraints on the system development.

Requirement Types

1. Functional Requirement

- describes the behavior of the system.
- It includes user tasks that the system needs to support.
- It is phrased as actions.

2. Non-Functional Requirement

- It describes the properties of the system.
- It is phrases as constraints or negative assertions.

Requirement Elicitation Methods

1. Questionnaire
2. Task Analysis
3. Scenario
4. Case study

❖ Use case

- It is a specification of a set of actions performed by a system which yields an observable result.
- It represents what the actors want your system to do for them.
- Each use case is a complete course of events in the system from a user perspective.

Use Case diagram

- Use case diagram is a representation of a user's interaction with the system that shows relationship between the user and the different use cases in which the user is involved.

- It helps to identify, clarify and organize the system requirements.
- It describes the behavior of the target system from an external point of view.
- A use case diagram consists of following components:
  1. Actor
  2. Use case
  3. Relationship

Actor

- Actors are the entities that interface with the system.
- Actors are external to the system.
- They may be people, external hardware or other subjects

Use case

- Specification of a set of actions performed by a system which yields an observable result.
- It represents what the actors want your system to do for them.

- Each use case is a complete course of events in the system from a user perspective.

Relationships

➢ **<< include >> relationship**

- A use case may contain functionality of another use case.
- It implies that the behavior of the included use case is inserted into the behavior of the including use case.
- It is expressed as a dotted line labelled << include >> beginning at base use case and ending with an arrow pointing to included use case.

➢ **<< extend >> relationship**

- Certain use case may be performed as part of another use case.
- It is optional.
- The base use case can complete without the extended use case.

- It changes the behavior of base use case.
- It implies that the behavior of a use case may be extended by the behavior of another use case.

➢ **Association**

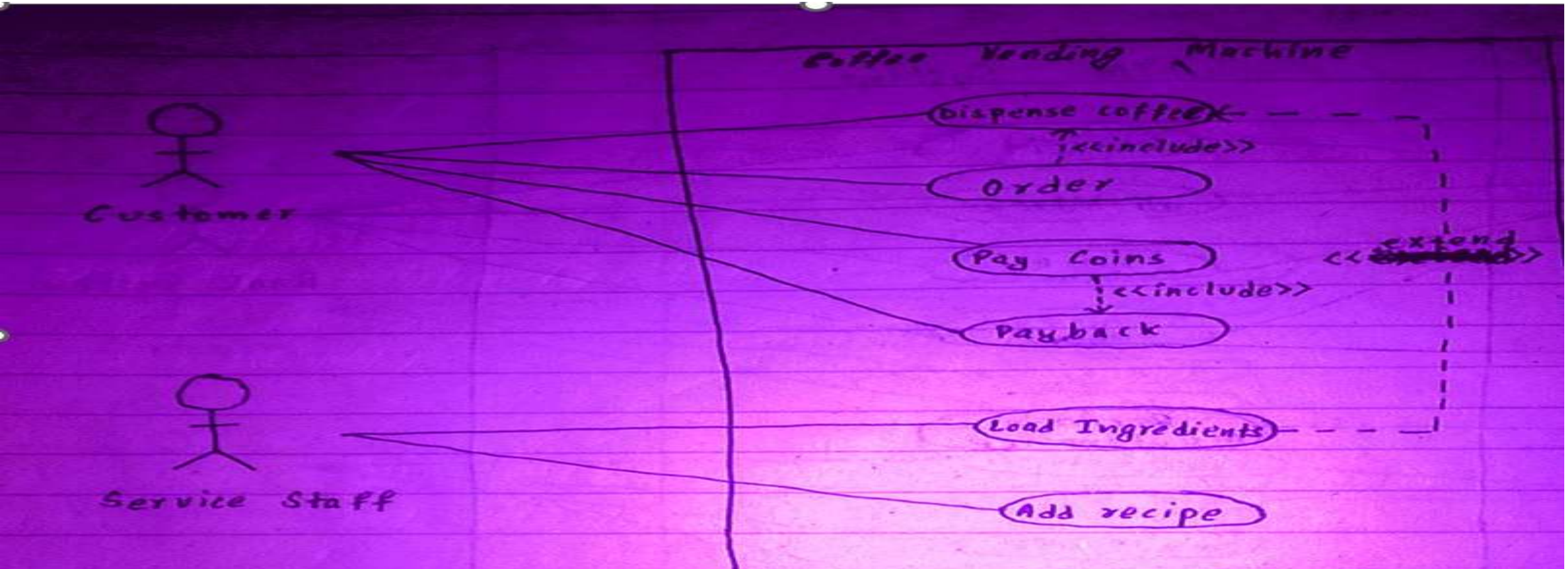
- It indicates the communication between an actor and a use case.
- It is represented by a solid line.

➢**Generalization**

- It is the relationship between general use case and a special use case.

Use case Example

A coffee vending machine dispenses coffee to customers. Customers order coffee by selecting a recipe from a set of recipes. Customers pay using coins. Change is given back if any to the customer. The service staff loads ingredients into machine. The service staff can also add a recipe by indicating name of coffee, units of coffee powder, milk, sugar, water and chocolate to be added as well as the cost of coffee.



❖ Object oriented development cycle

- The object-oriented development life cycle goes through following stages:
  1. Analysis
  2. Design
  a) System design
  b) Object design
  3. Implementation and Testing
- The most successful approach for object-oriented software development is Rational Unified Process (RUP). It is an approach that combines iterative, risk driven development into a well documented process description.
- The input to a process is the needs, process is the set of activities to reach goal and output is the software product.

The phases involved in RUP are as follows:

1. Inception

- The requirements are gathered.
- Feasibility study and scope of the project are determined.
- Actors and their interactions are analyzed.

2. Elaboration

- Project plan is developed.
- Risk assessment is performed.
- Non-functional requirements are elaborated.
- Software architecture is described.
- Use case model is completed.

3. Construction

- All the components are developed and integrated.
- All features are tested.
- In each iteration, refactoring is done.
- Stable product should be released.

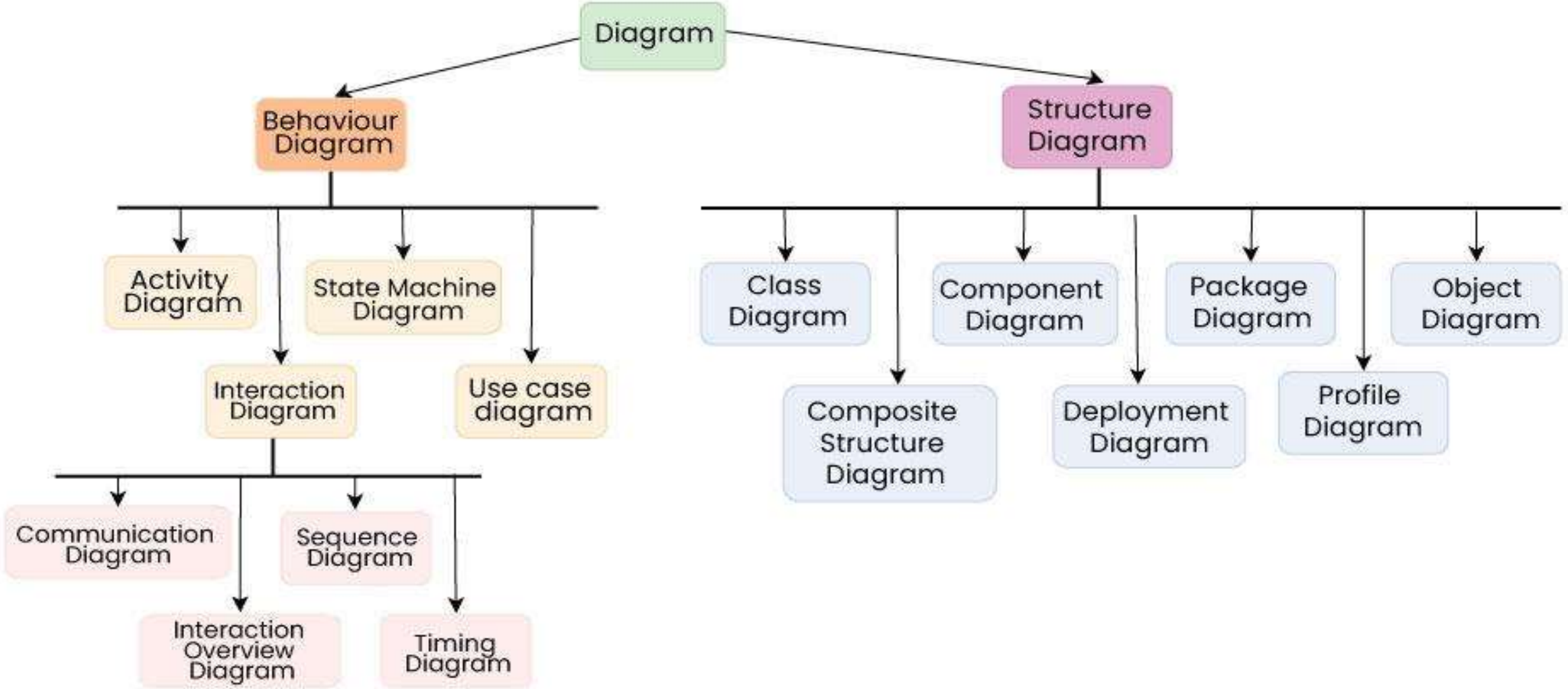
4. Transition

- Software product is launched to user.
- Deployment baseline should be complete.
- Final product should be released.

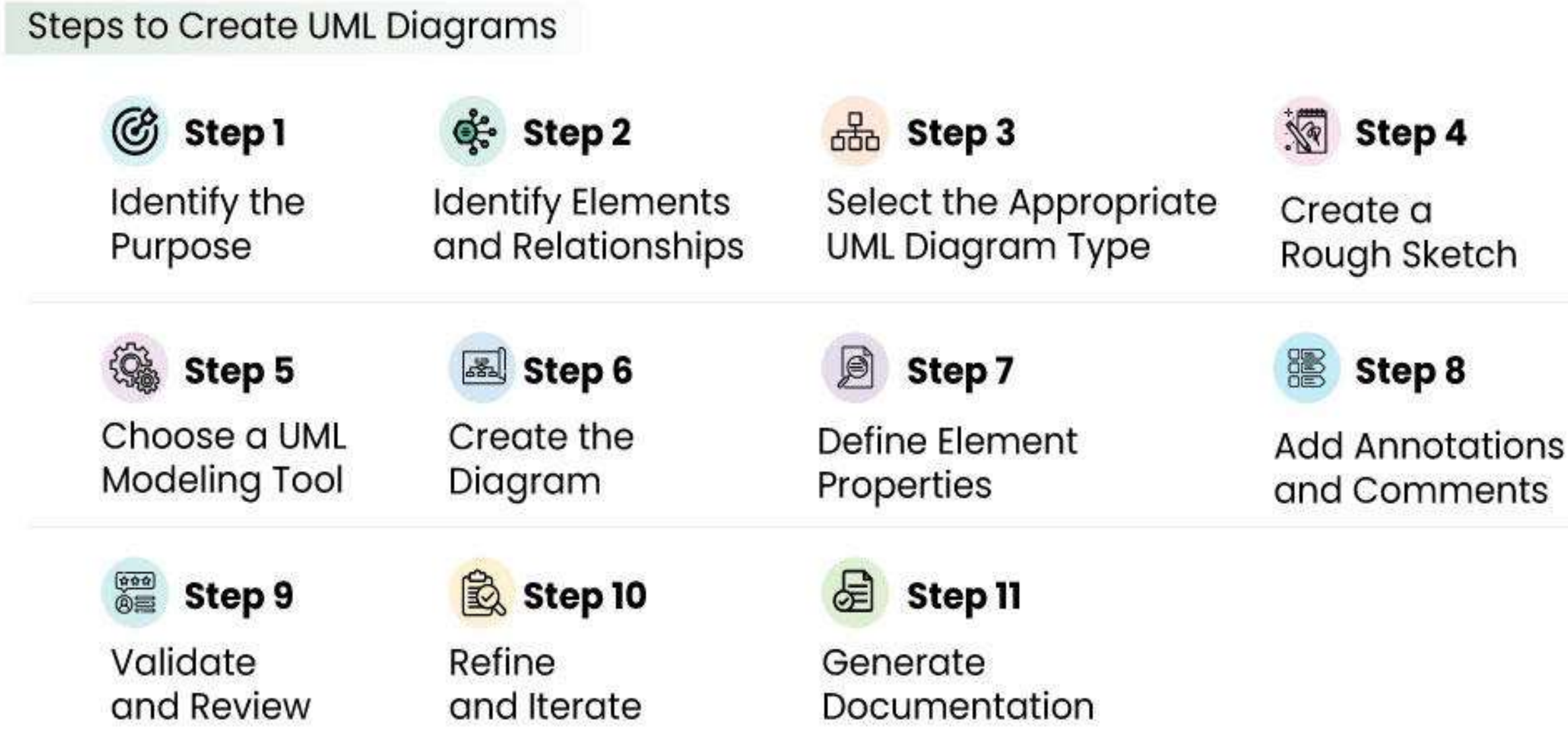
❖ Unified Modelling Language

- General-purpose, graphical modeling language.
- used to specify, visualize, construct, and document the artifacts (major elements) of the software system.
- UML is **not a programming language;** it is rather a visual language.
- UML diagrams to portray the **behavior and structure** of a system.
- Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them

Different types of UML



Steps to create UML Diagrams



UML view

1. User View

• It defines the functionalities provided by the system to the users.

• It includes use case diagram.

2. Structural View

• It defines the structure of the system and is also called static models.

• It includes class diagram, component diagram, object diagram, profile diagram, deployment diagram and package diagram.

3. Behavioral View

• It captures how objects interact with each other.

• It shows the time dependent or dynamic model of the system.

• It includes activity diagram, interaction diagram, sequence diagram and state machine diagram.

4. Implementation View

• It captures the components of the system and their dependencies.

• It includes component diagram.

5. Deployment View

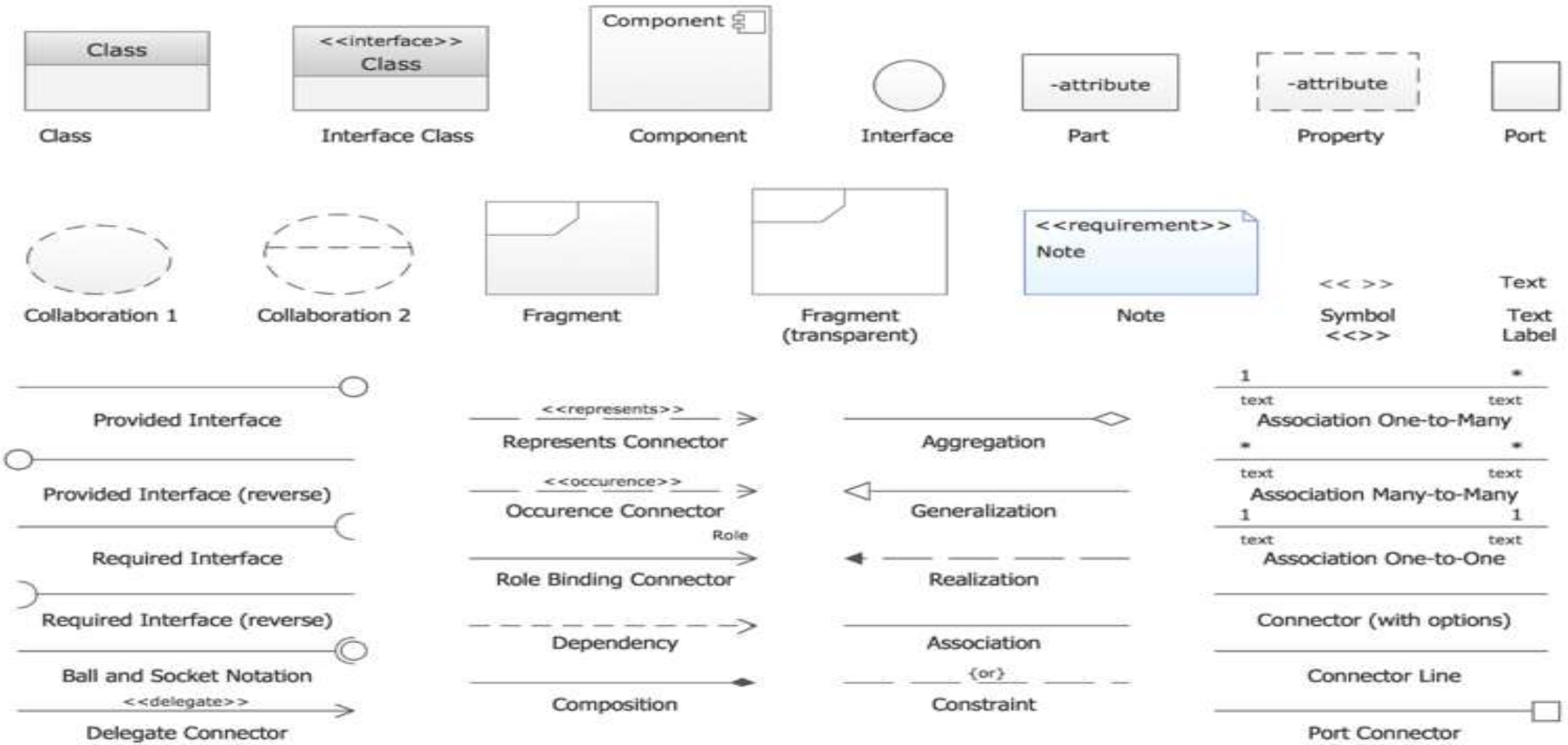
• It captures how components are deployed into the system.

• It includes package diagram and deployment diagram.

Relationships in UML

1. Dependency (change in one thing affect semantic of dependent things)
2. Association (describe links between objects)
3. Generalization (objects of specialized element substitutable for objects of generalized element)
4. Realization (two classifiers where one specifies contract and other guarantees to carry out the contract)

Notations in UML



❖ Building conceptual Model

❑*Domain Modelling*

- Domain model is the important conceptual model that illustrates the noteworthy concepts in a domain.
- It represents the context in which the system must operate.
- A domain model in UML is illustrated with a set of **class diagrams omitting the operations.**
- It shows the following concepts:
  1. Domain classes
  2. Associations between domain class
  3. Attributes of domain class
- It visualizes and relates concepts of the domain.

❖ Adding Associations and attributes

- Association is the relationship between classes.
- The ends of association may contain multiplicity.
- Multiplicity refers to the numerical relationship between instances of the class.
- Associations should be named with verb phrase in a readable and meaningful way.
- Association name starts with a capital letter.
- Each end of an association is called role.
- Two classes can also have multiple associations.
- Attributes is a logical data value of an object.
- Attributes are shown in second compartment of class box.
- Attribute name is compulsory.
- Type and other information are optional.

❖ Representation of System Behavior

- System behavior describes what a system does.
- It hides the implementational details of how system performs.
- It provides the dynamic model of the system.
- A system behavior is depicted as a black box.
- It must show the reaction of system with external events, timer events and faults with a time frame embedded within it.
- System behavior can be represented by:

1. Use Case
2. System Sequence Diagram (SSD)
3. Operation Contracts

System Sequence Diagram

- Actor generates events by requesting something to the system.
- The request event initiates an operation in the system.
- Ordering of events should follow their order in the scenario.
- SSD can be constructed from use case as:

a) Draw system as black box on right side.
b) For each actor, draw stick figure and lifeline.
c) For each events that each actor generates, draw message.

Operation Contracts

- It gives detailed representation of system behavior.
- Contract describes outcome of executing system operation in terms of state changes to domain objects.
- It is a document containing:

1. Operation
2. Cross Reference
3. Pre-conditions
4. Post conditions

THANK YOU