OBJECT ORIENTED ANALYSIS & DESIGN
DATA STRUCTURES & ALGORITHMS

Unified Modelling Language

Modelling

Why Modeling ?

- A model is a simplification at some level of abstraction
- We build models to better understand the systems we are developing.
- To help us visualize to specify structure or behavior
- To provide template for building system to document decisions we have made

Principles of Modeling:

- The models we choose have a profound influence on the solution we provide
- Every model may be expressed at different levels of abstraction
- The best models are connected to reality

Unified Modelling Language

- **“A picture is worth a thousand words.” - old saying**
- **“A language provides a vocabulary and the rules for combining words for the purpose of communication”**
- **A modeling language is a language whose vocabulary and rules focus on the conceptual and physical representation of a system. A modeling language such as the UML is thus a standard language for software blueprints.”**
- **Brief History:** It was developed in 1990s as an amalgamation of several techniques, prominently OOAD technique by Grady Booch, OMT (Object Modeling Technique) by James Rumbaugh, and OOSE (Object Oriented Software Engineering) by Ivar Jacobson.
- UML attempted to standardize semantic models, syntactic notations, and diagrams of OOAD.

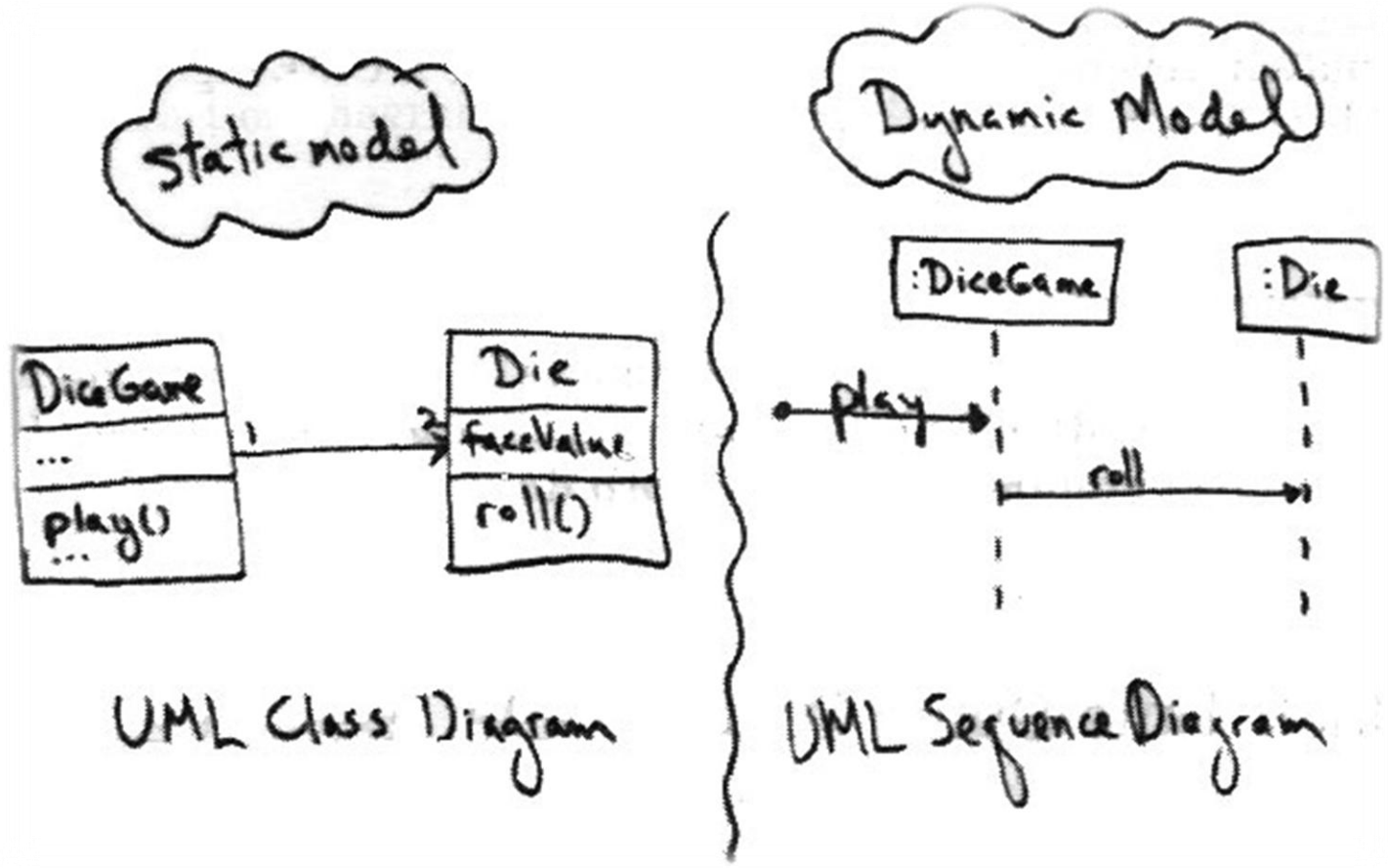
Static Models

- Static models, such as UML class diagrams, help design the definition of packages, class names, attributes, and method signatures (but not method bodies).

Dynamic Models

- Dynamic models, such as UML interaction diagrams (sequence diagrams or communication diagrams), help design the logic, the behavior of the code or the method bodies.
- They tend to be the more interesting, difficult, important diagrams to create.

Static and Dynamic Modelling



UML Diagrams

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 its environment (users or systems) within a particular situation.

Class diagram: It shows the different objects, their relationship, their behaviors, and attributes.

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.

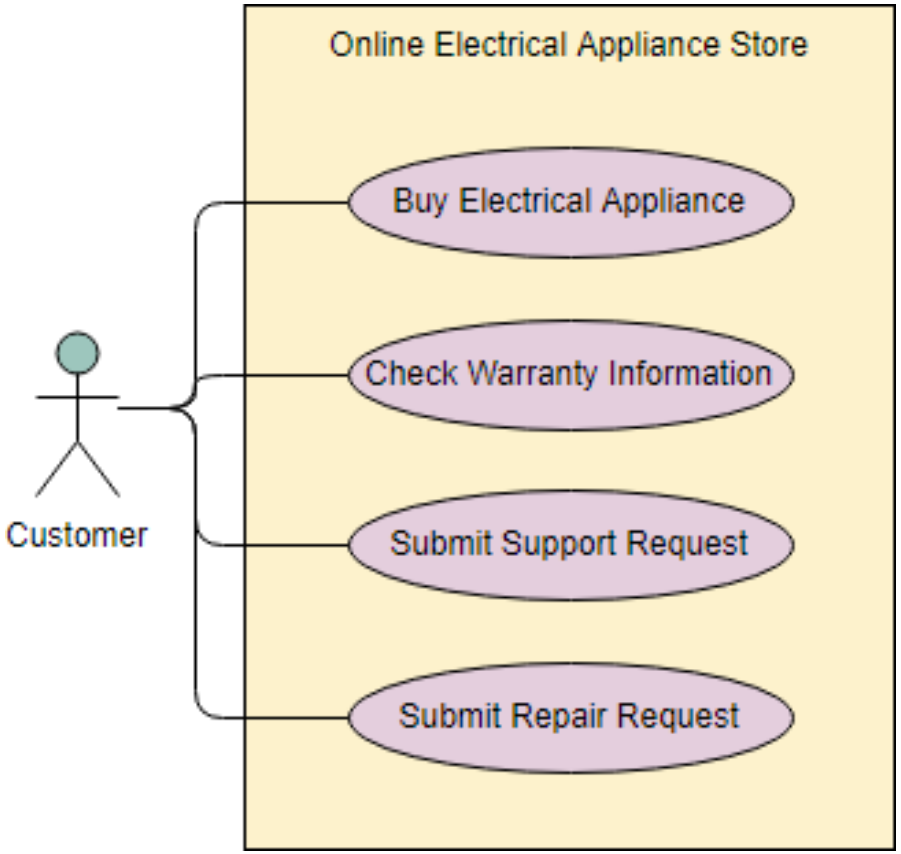
List of Tools :

https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools

Use Case Diagram

A use case describes how a user uses a system to accomplish a particular goal.

A use case diagram consists of the system, the related use cases and actors and relates these to each other to visualize: what is being described? (system), who is using the system? (actors) and what do the actors want to achieve? (use cases), thus, use cases help ensure that the correct system is developed by capturing the requirements from the user's point of view.



Use Case Diagram

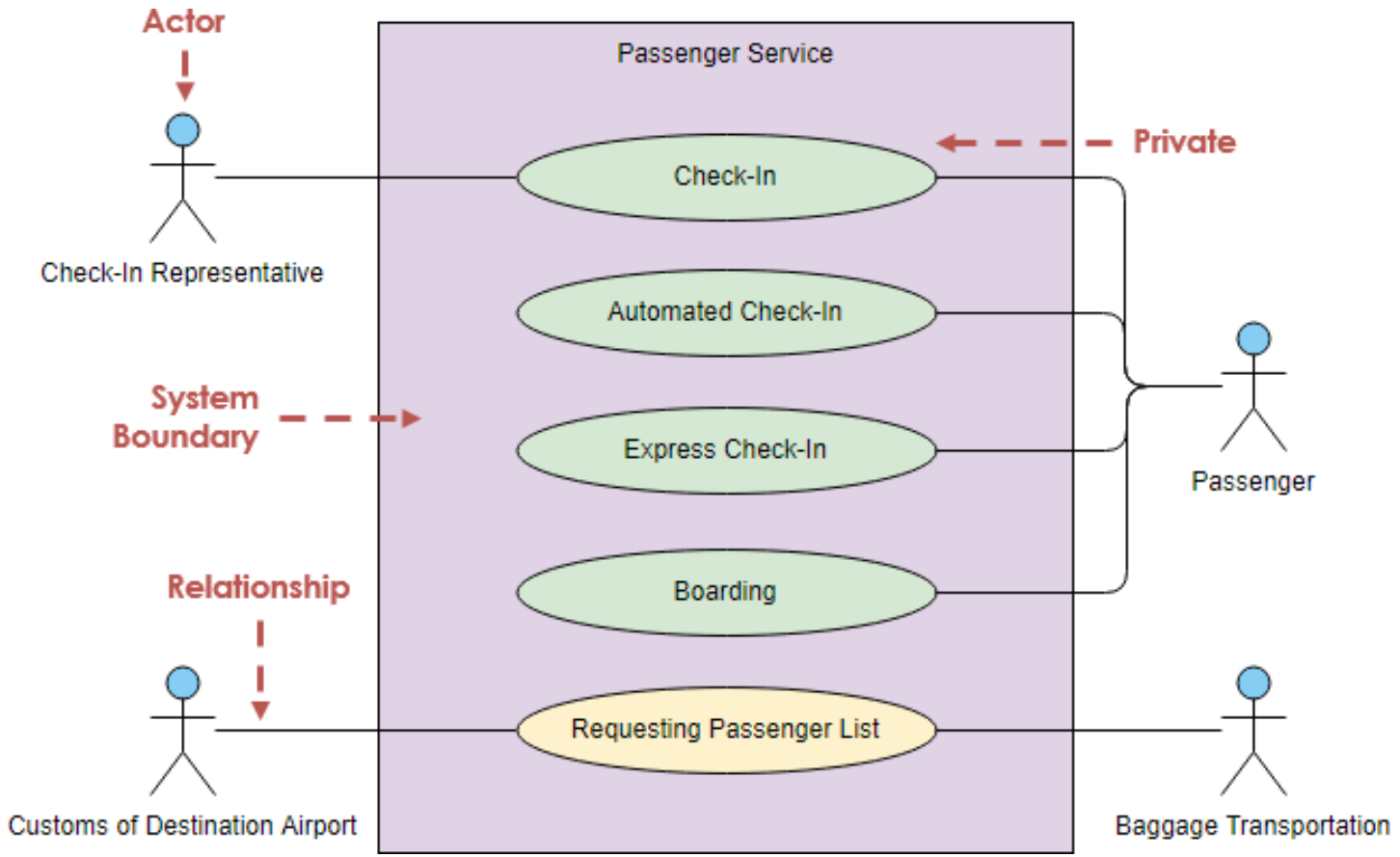
Actors are usually individuals involved with the system defined according to their roles. The actor can be a human or other external system.

A use case describes how actors uses a system to accomplish a particular goal. Use cases are typically initiated by a user to fulfill goals describing the activities and variants involved in attaining the goal.

The relationships between and among the actors and the use cases.

The system boundary defines the system of interest in relation to the world around it.

Use Case Diagram (Contd.)



Class Diagrams

A class diagram describes the structure of an object-oriented system by showing the classes in that system and the relationships between the classes. A class diagram also shows constraints, and attributes of classes.

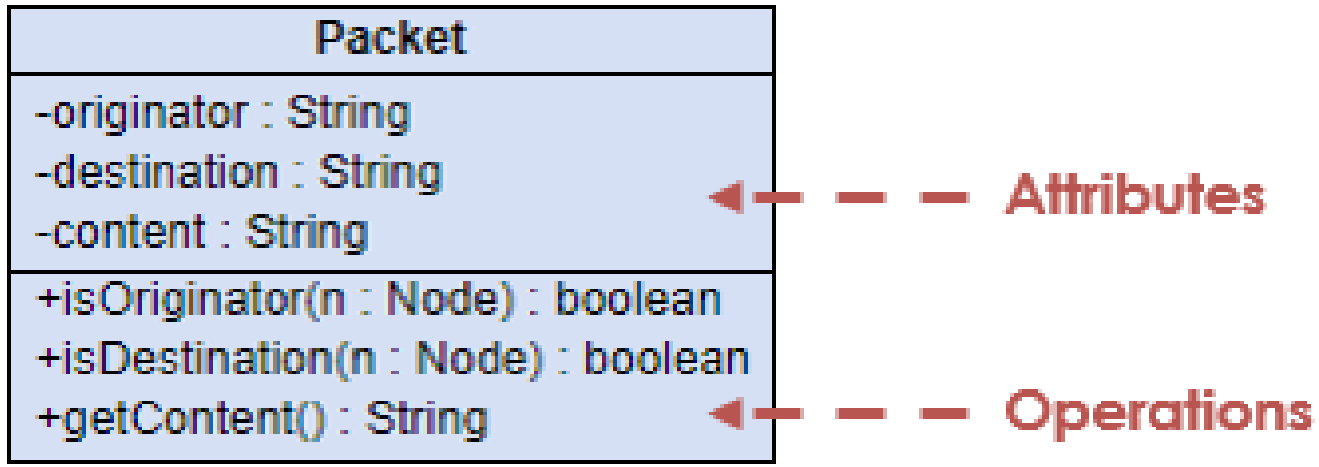
Attribute

The attribute section of a class lists each of the class's attributes on a separate line. The attribute section is optional, but when used it contains each attribute of the class displayed in a list format. The line uses this format: name : attribute type (e.g. cardNumber : Integer).

Operation

The operations are documented in the bottom compartment of the class diagram's rectangle, which also is optional. Like the attributes, the operations of a class are displayed in a list format, with each operation on its own line. Operations are documented using this notation: name (parameter list) : type of value returned (e.g. calculateTax (Country, State) : Currency).

Class Diagrams



Color Coding

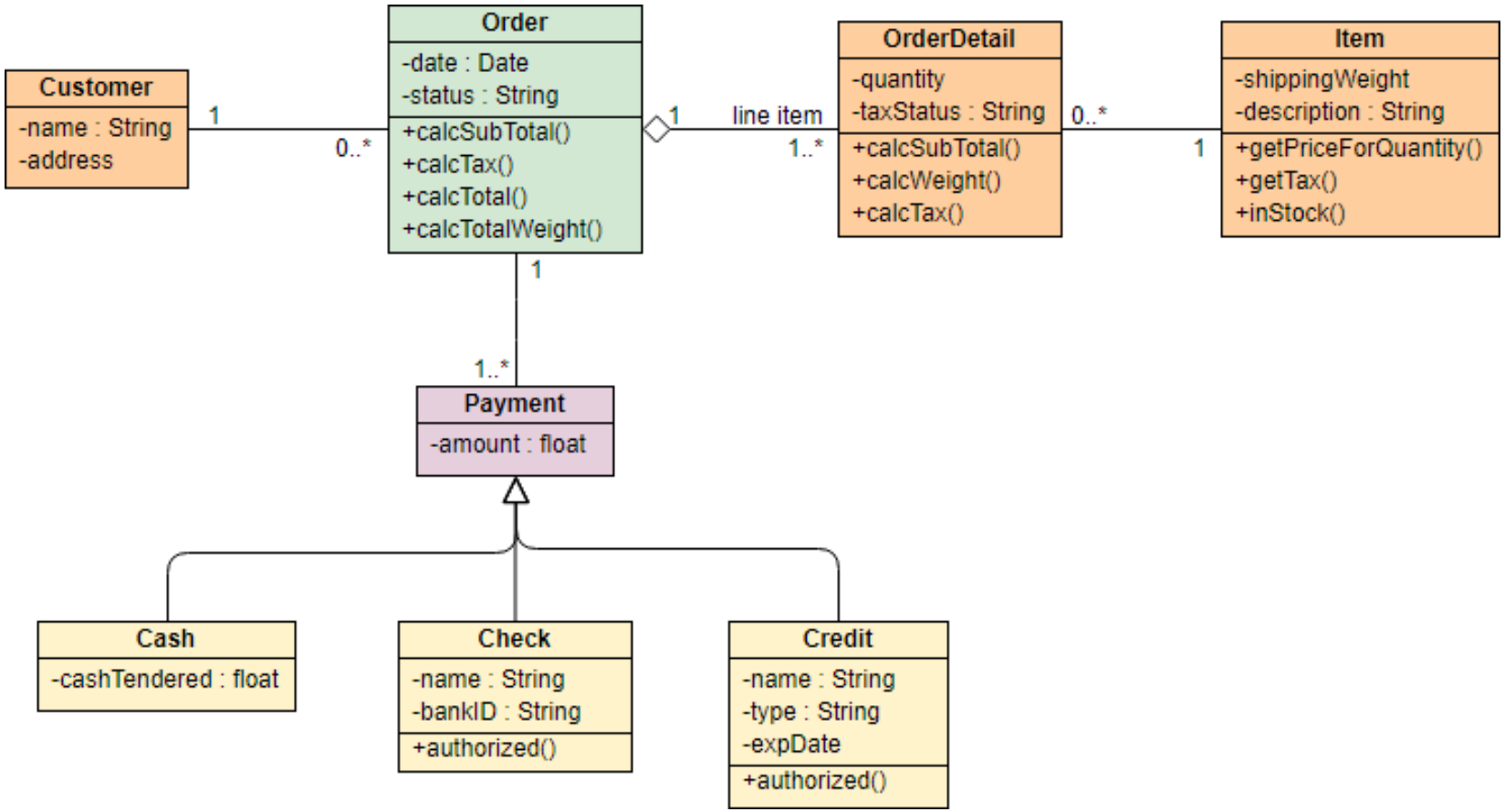
- Pink (Moment-interval) — Represents a moment or interval of time. e.g: a sale, a video rental, an account registration
- Yellow (Role) — A way of participating in the above activity. e.g: Cashier, Customer, Manager
- Blue (Description) — A catalog like description which classifies objects. e.g: product catalog
- Green (Party, Place or Thing) — Something that is uniquely identifiable. e.g: Person, Store, Product

Common Operators

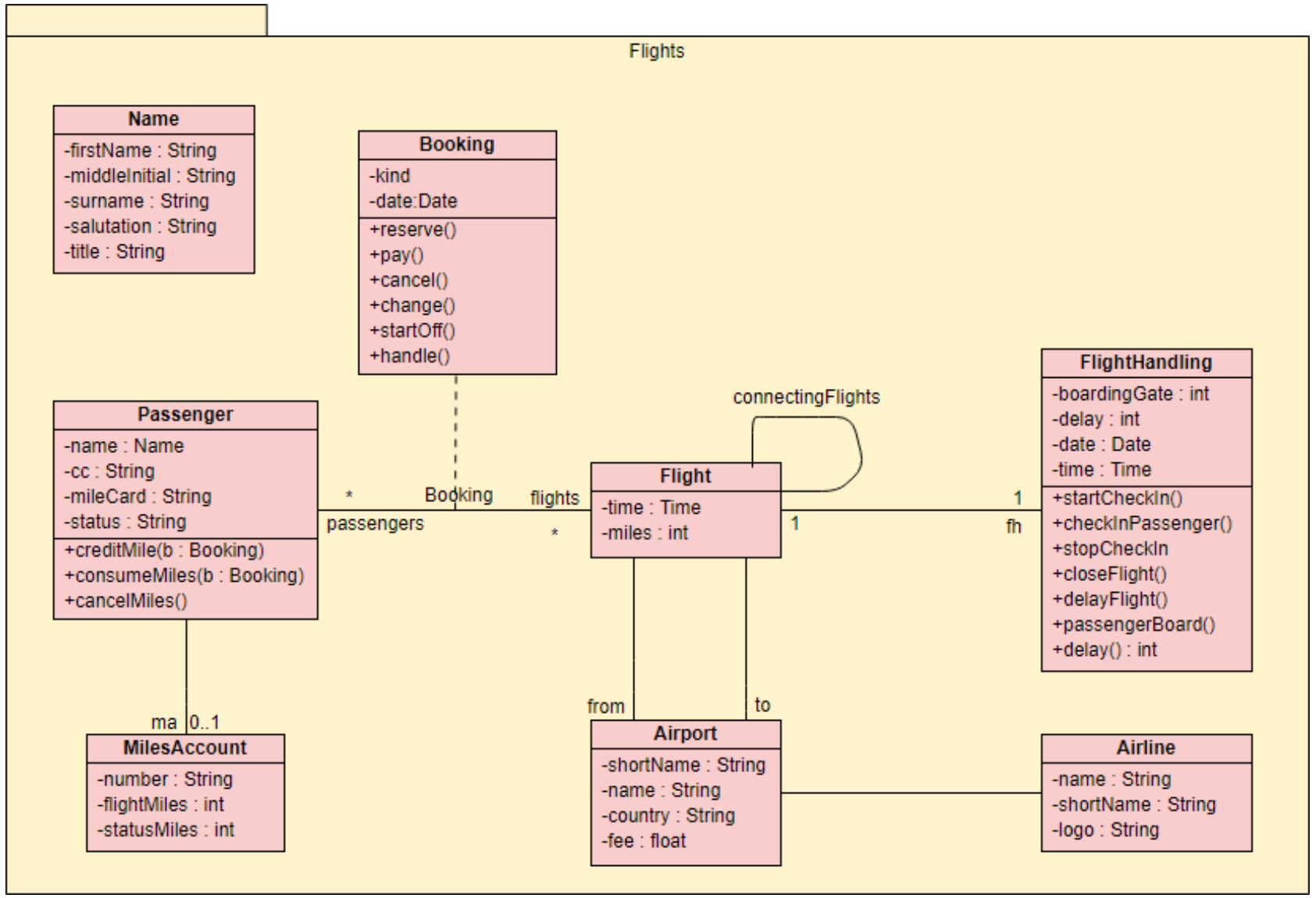
Table 4.1. Common Operators for Interaction Frames

Operator	Meaning
`alt`	Alternative multiple fragments: only the one whose condition is true will execute
`opt`	Optional; the fragment executes only if the supplied condition is true. Equivalent to an `alt` with only one trace
`par`	Parallel; each fragment is run in parallel.
`loop`	Loop; the fragment may execute multiple times, and the guard indicates the basis of iteration
`region`	Critical region; the fragment can have only one thread executing it at once.
`neg`	Negative; the fragment shows an invalid interaction.
`ref`	Reference; refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.
`sd`	Sequence diagram; used to surround an entire sequence diagram, if you wish.

Class Diagrams – Case Study



Class Diagrams – Case Study



UML Objects

1. Identify
2. Refine
3. Draw

1. Identify Objects

- What we do is to start collecting our use cases, user stories, and any other written requirements together.
- Now, we are going to identify the most important parts of our software; the most important things, or objects.
- Objects will be in form of nouns. Those are the candidate objects, some of them will be actual objects in the system, and the rest won't, as you'll see later.

Use Case Scenario: Customer confirms items in shopping cart. Customer provides payment and address to process sale. System Validates payment and responds by confirming order, and provides order number that customer can use to check on order status. System will send customer a copy of order details by email.

2. Refine Objects

After underlying on your candidate objects, you start refining them, you start choosing your actual objects that will be in the system. So, to do that

- Remove any duplicates. We may find same objects with different names, but they actually mean the same thing.
- You may need to combine some objects, or, even splitting them into some other objects.
- You may identify an attribute as an object instead.
- You may identify a behavior as an object instead.

An attribute is a property or characteristic of the object. For example, when we say “A car is red”, red here is a property for the car, which is the actual object.

A behavior is something an object can do (responsibility). For example, when we say, “A bird can fly”, fly here is a behavior, while the bird is the actual object.

2. Refine Objects (Contd.)

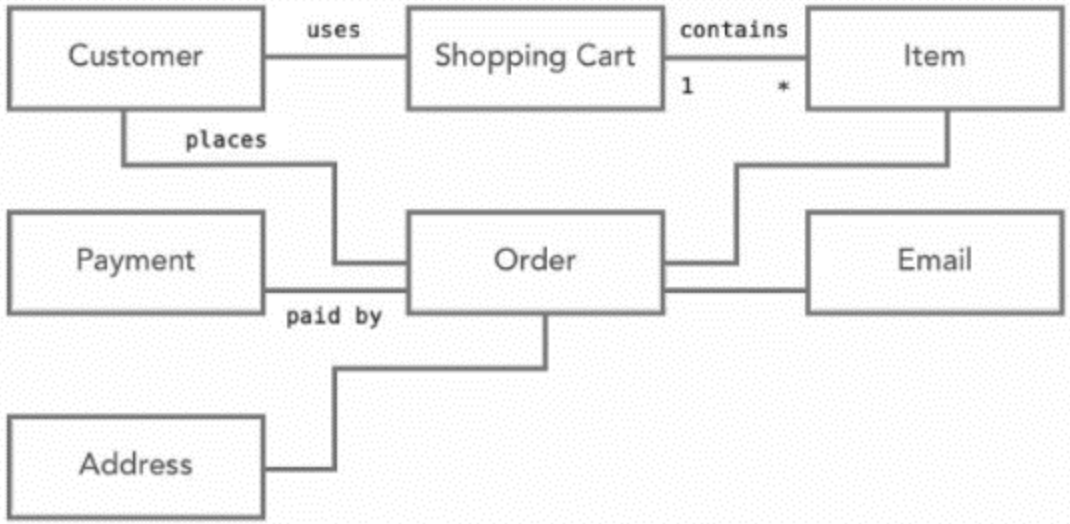
Customer	Order
Item	~~Order Number~~
Shopping Cart	~~Order Status~~
Payment	~~Order Details~~
Address	Email
~~Sale~~	~~System~~

3. Draw Objects

- What you need to do now is using your pencil and paper, just draw the conceptual model by box all objects.
- There are some tools you may use, but for now, a pencil, and piece of paper are more than enough.

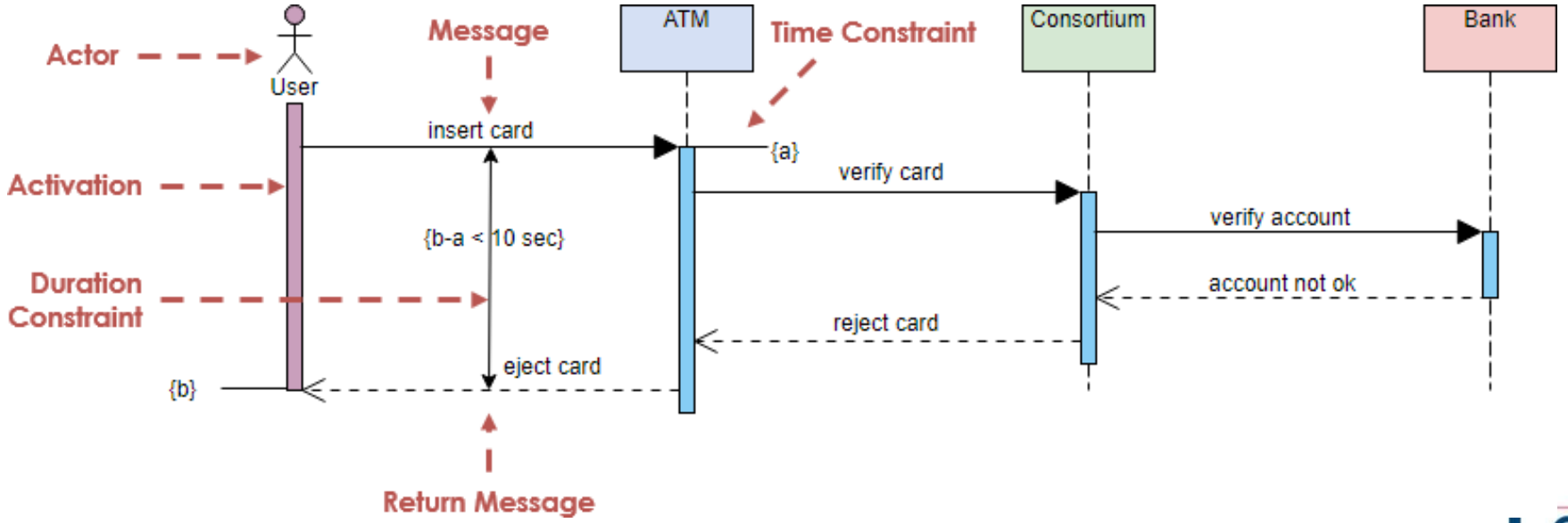
4. Identify Relationships

- You start indicate the relationships between your objects.
- It’s obvious that these objects will interact with each other. For example, a customer can place an order, a student can enroll in a course, an admin can update a post, and so on.
- Drawing a line between objects and writing the relationship verbs is



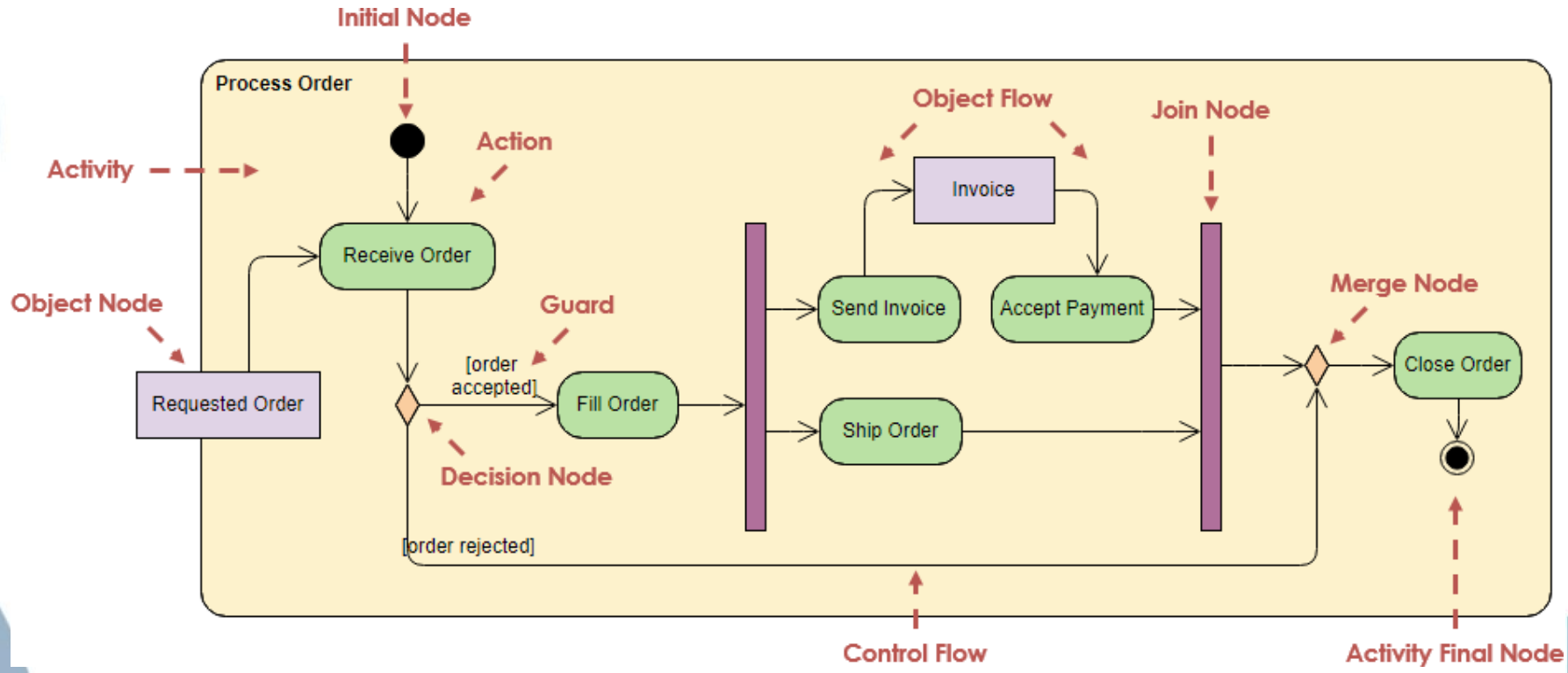
Sequence Diagrams

A sequence diagram describes an interaction among a set of objects participated in a collaboration (or scenario), arranged in a chronological order; it shows the objects participating in the interaction by their "lifelines" and the messages that they send to each other.



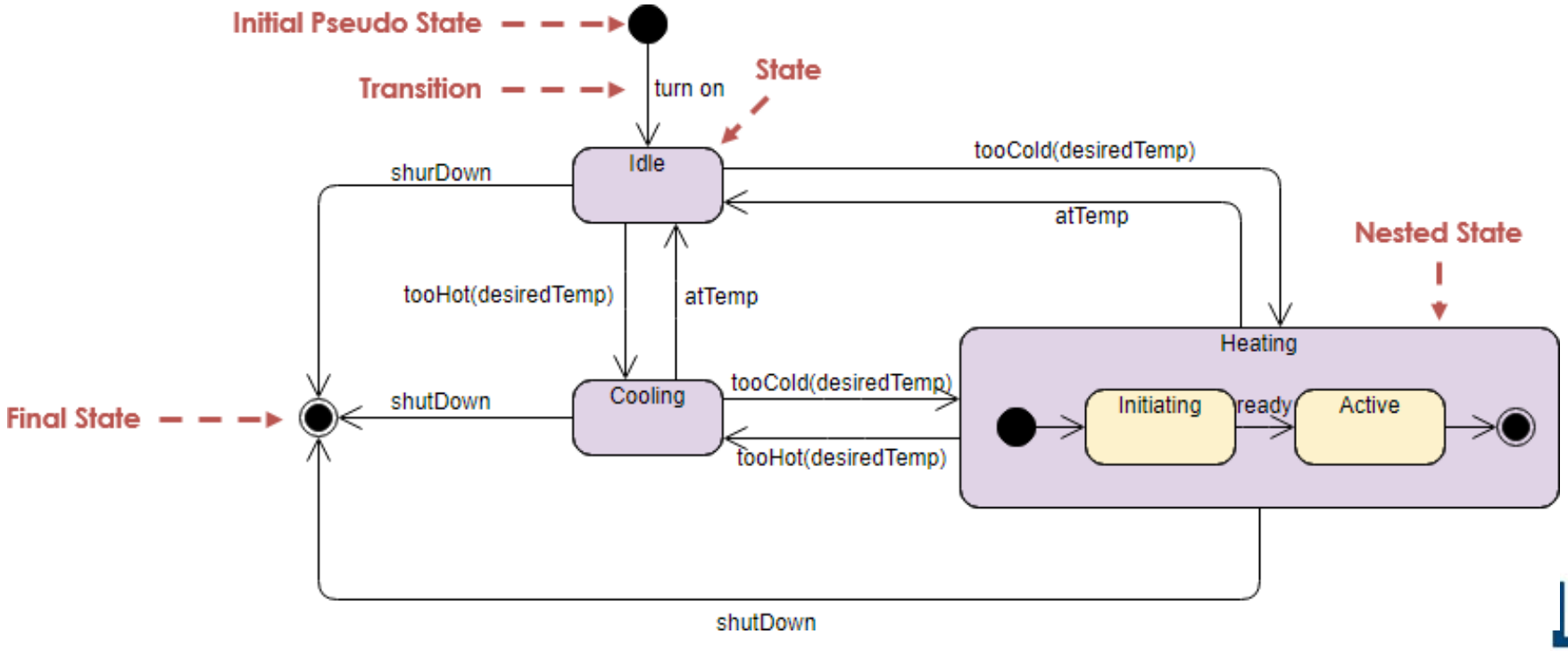
Activity Diagram

Activity Diagram is similar to a business workflow diagram or simply a flowchart with much richer semantics. It describes the system activities, or the person who does the activity, and the sequential flow of these activities. The activity diagram is one of the UML diagrams associated with object-oriented approach, through it can be used in any other software development paradigm



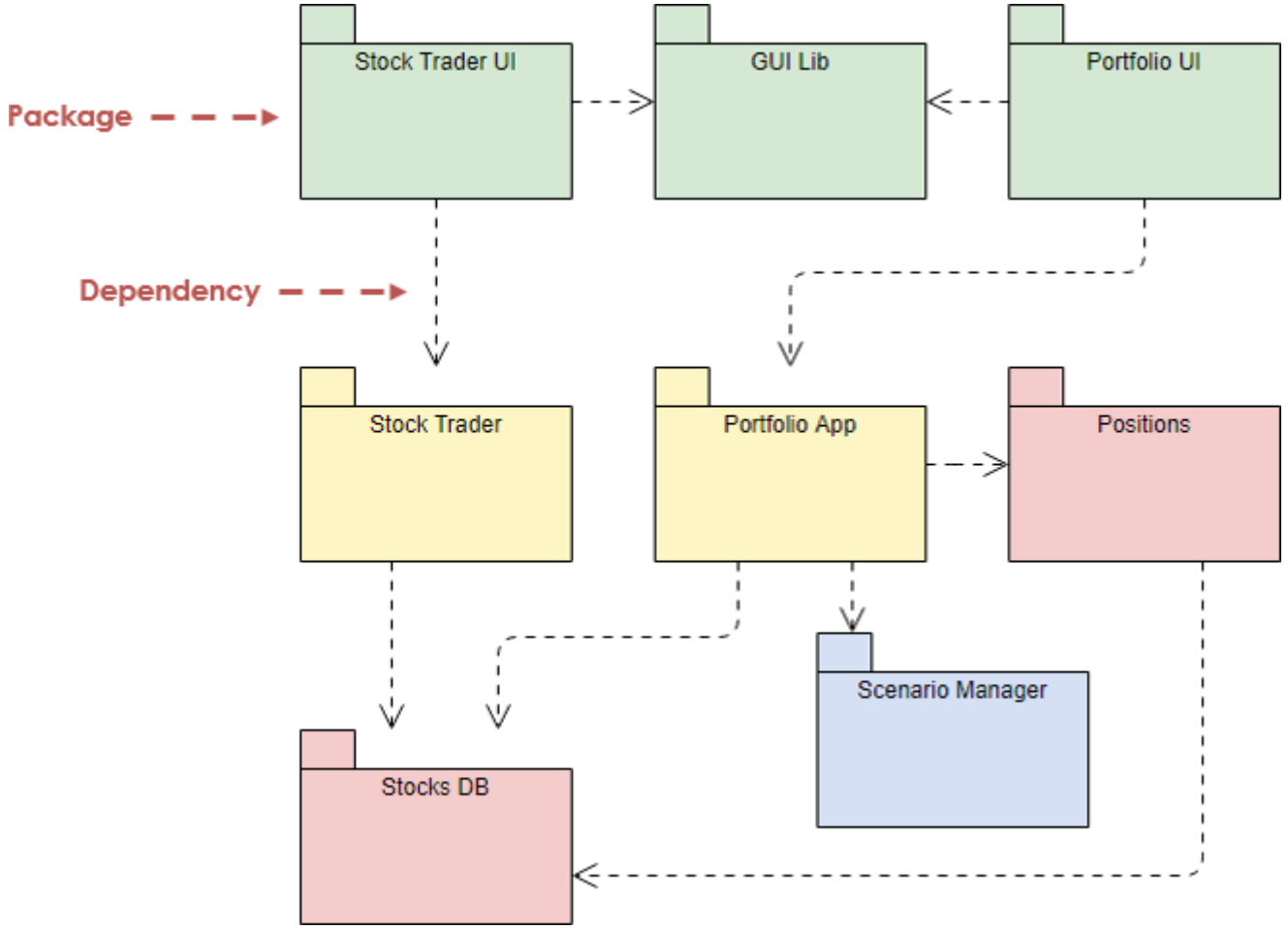
State Machine Diagram

A state machine diagram is used to model the dynamic behavior of individual class objects, use cases, and entire systems. In other words, when a state machine created where the object it is attached to, that object becomes the owner of the state machine, for example, the object to be attached by the state machine could be a class, use case or even the entire system.

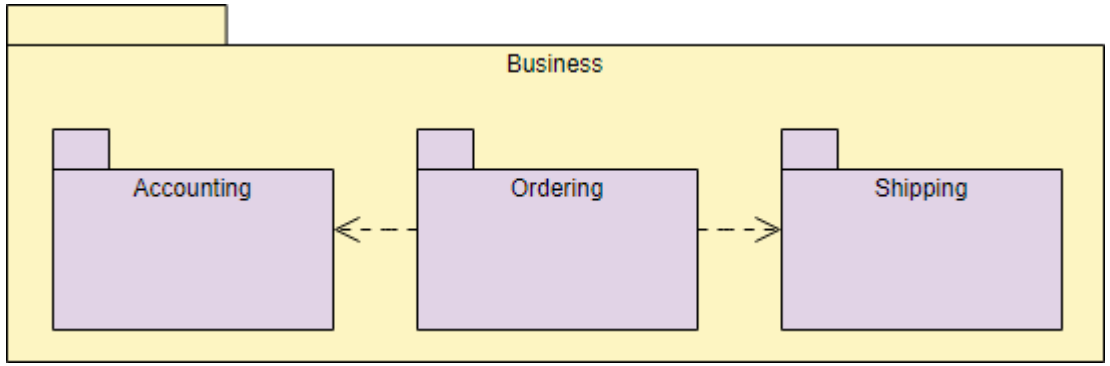


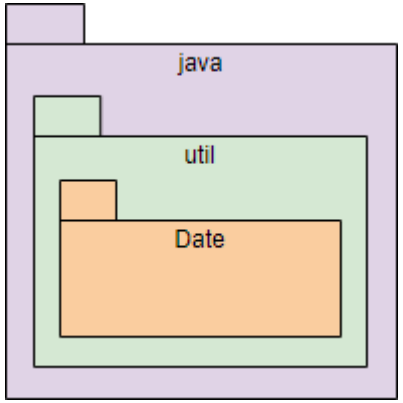
Packages

Package diagram shows the arrangement and organization of model elements in middle to large scale project that can be used to show both structure and dependencies between sub-systems or modules.



Packages (Contd.)





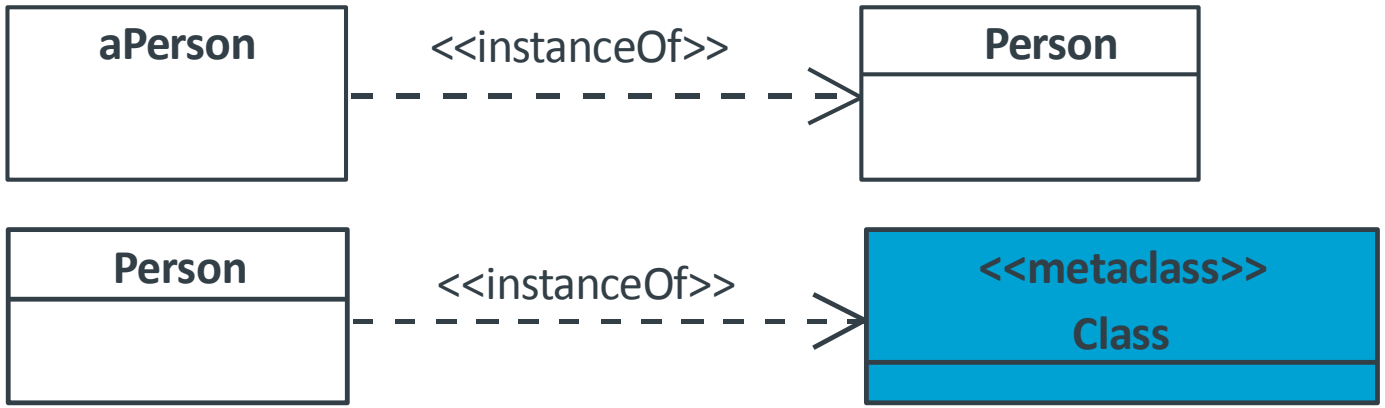
UML Extensibility

UML defines three extensibility mechanisms to allow modelers to add extensions without having to modify the underlying modeling language.

- Stereotypes
- Constraints
- Tagged Values

UML Meta Model

Relationship between model and meta-model:



The meta class Class is a model for the class Person
Since Person is a model (for the instance aPerson), Class is a meta model (model for models)

Thank You!