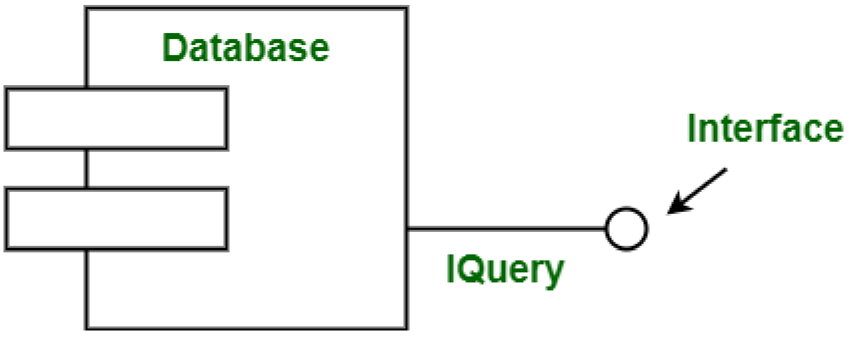
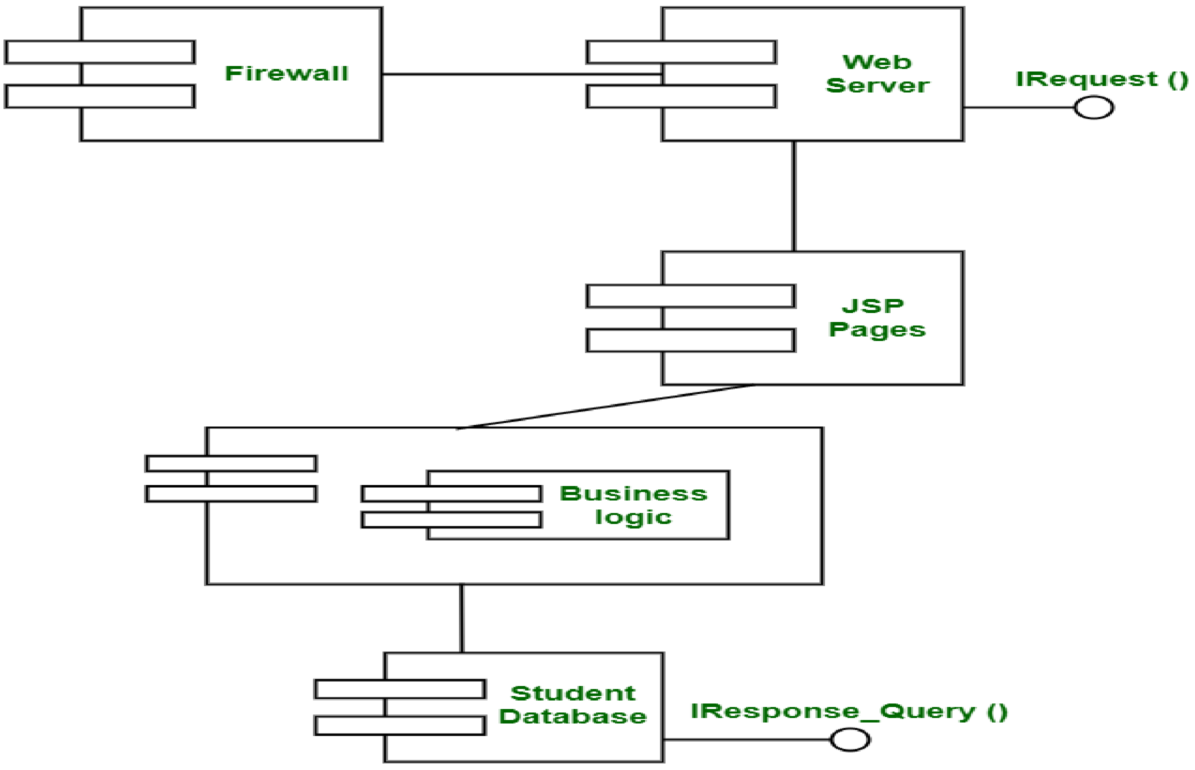
A component diagram is used to break down a large object-oriented system into the smaller components, so as to make them more manageable. It models the physical view of a system such as executables, files, libraries, etc. that resides within the node.

It visualizes the relationships as well as the organization between the components present in the system. It helps in forming an executable system. A component is a single unit of the system, which is replaceable and executable. The implementation details of a component are hidden, and it necessitates an interface to execute a function. It is like a black box whose behaviour is explained by the provided and required interfaces.



‘On-line Course Registration’ system



Advantages :

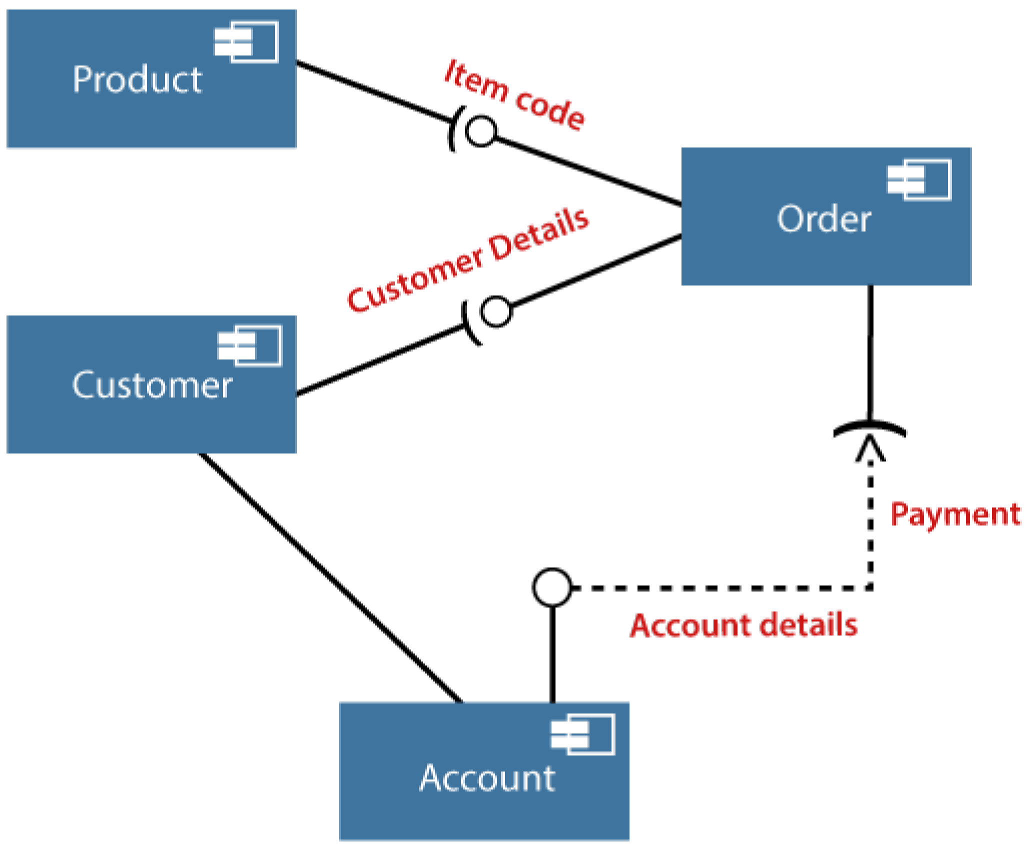
- Component diagrams are very simple, standardized, and very easy to understand.
- It is also useful in representing implementation of system.

- These are very useful when you want to make a design of some device that contains an input-output socket.
- Use of reusable components also helps in reducing overall development cost.
- It is very easy to modify and update implementation without causing any other side effects.

Disadvantages :

- They cannot be used for designing Software like web pages, applications, etc.
- It also requires sponsoring equipment and actuators for each and every component.

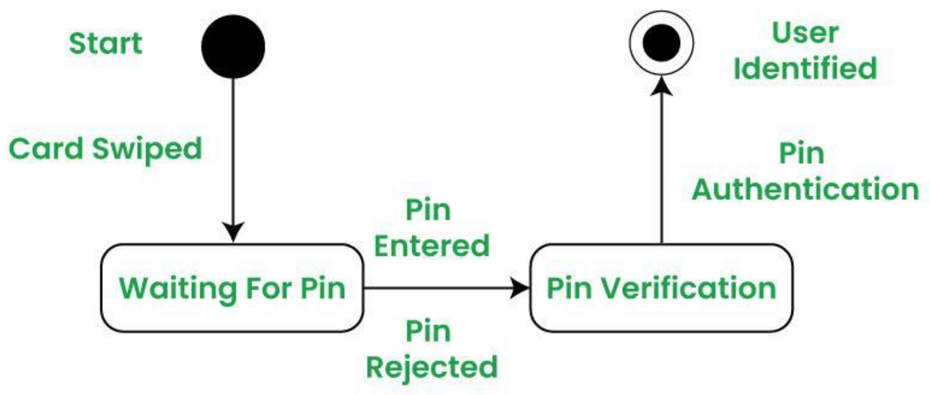
Online shopping system



State Machine Diagram

- A State Machine Diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioral diagram and it represents the behavior using finite state transitions. So simply, a state machine diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli.
- We can say that each and every class has a state but we don't model every class using State Machine diagrams.
- We prefer to model the states with three or more states.

A State Machine Diagram for user verification

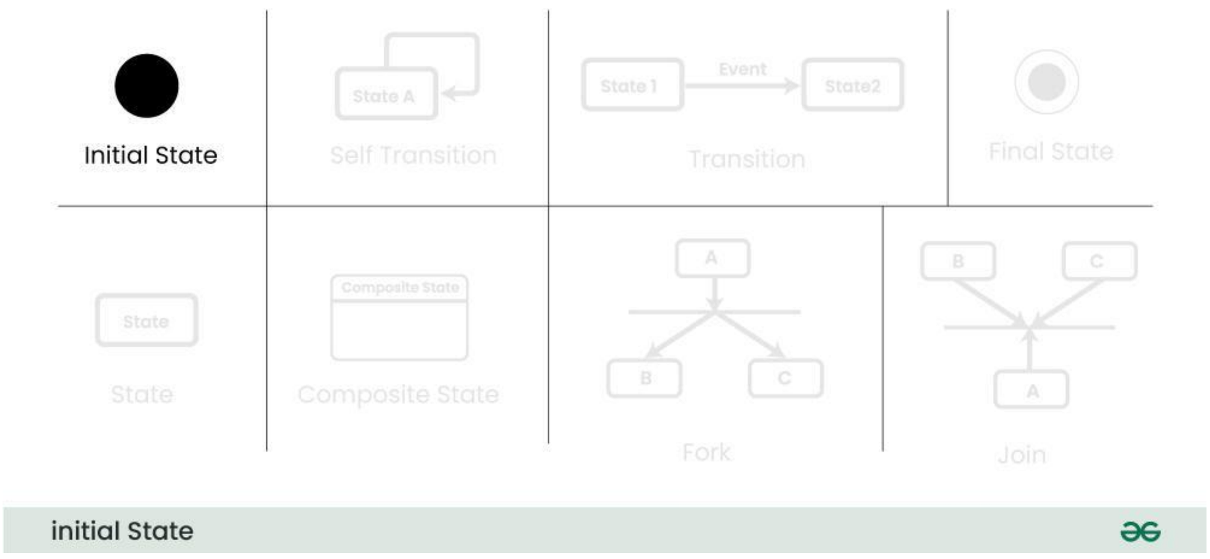


State Machine Diagrams | Unified Modeling Language (UML)

Basic components and notations of a State Machine diagram

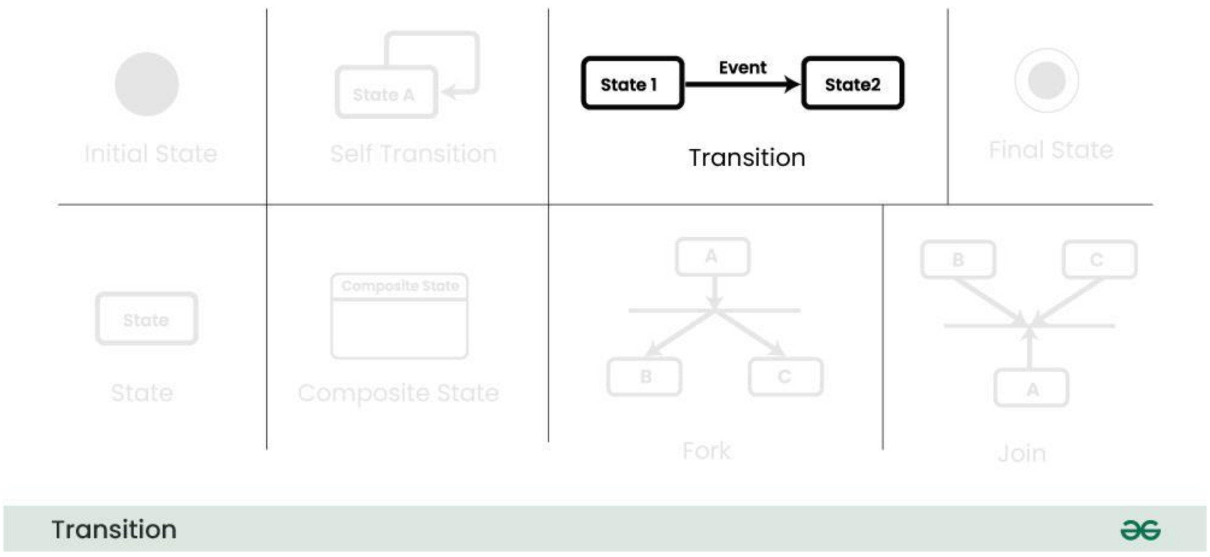
2.1. Initial state

We use a black filled circle represent the initial state of a System or a Class.



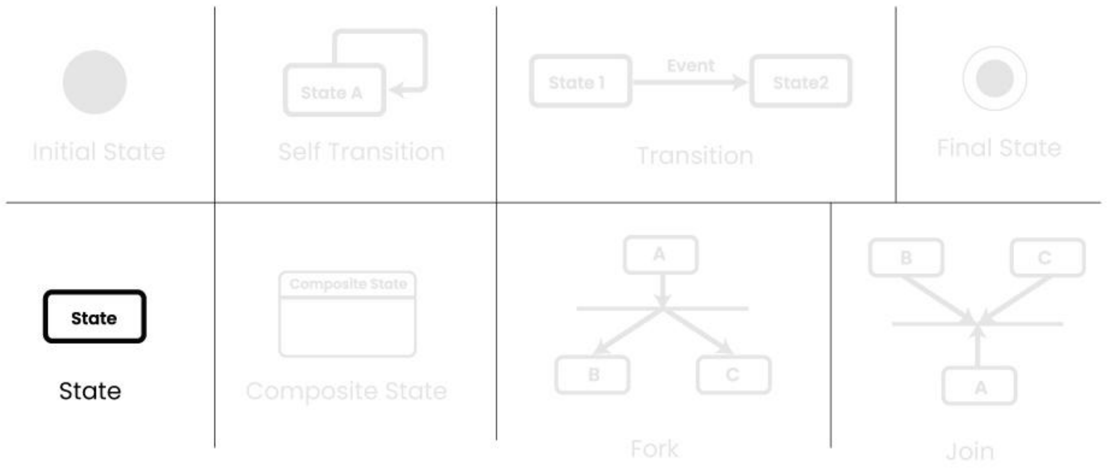
2.2. Transition

We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.



2.3. State

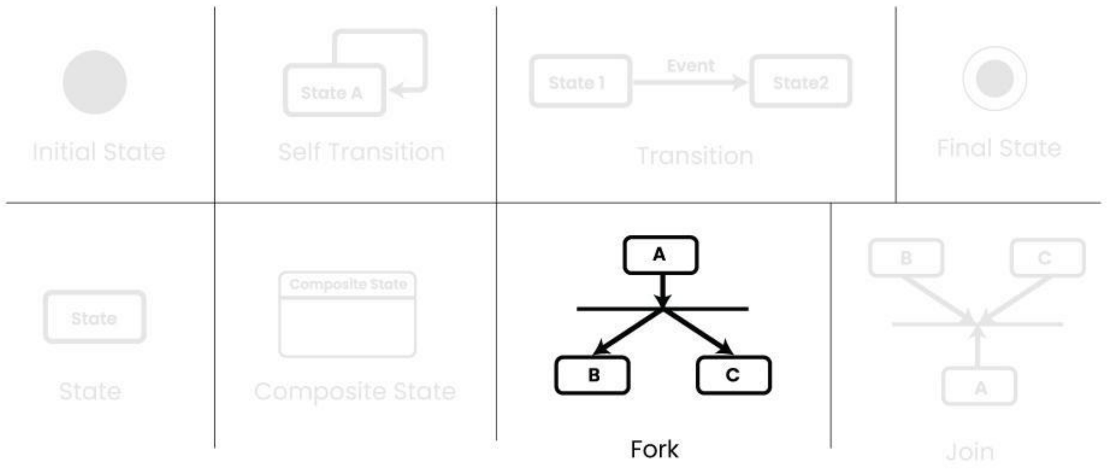
We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.



State

2.4. Fork

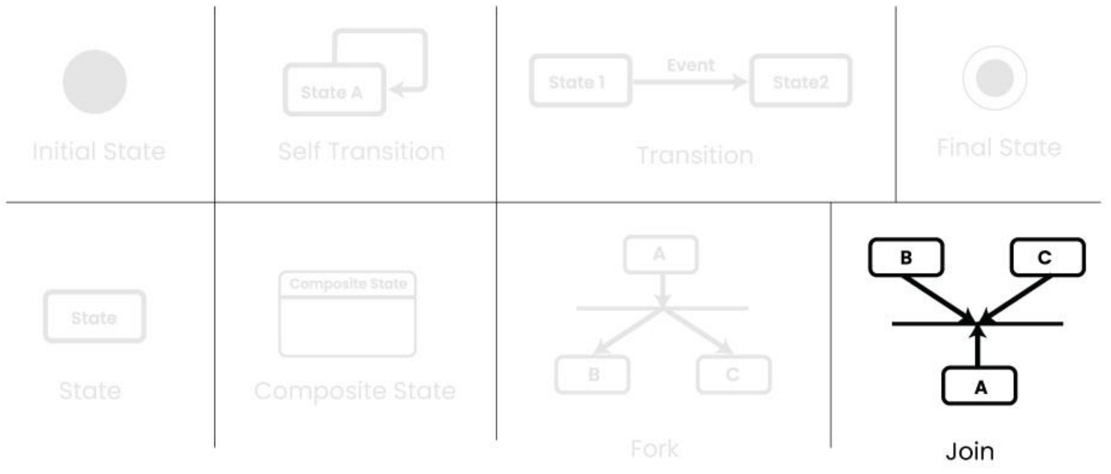
We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent state and outgoing arrows towards the newly created states. We use the fork notation to represent a state splitting into two or more concurrent states.



Fork

2.5. Join

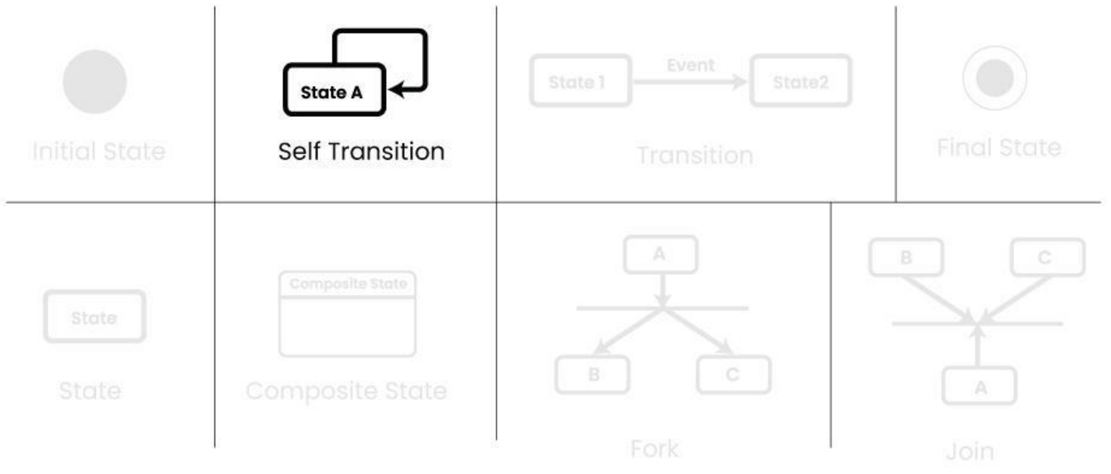
We use a rounded solid rectangular bar to represent a Join notation with incoming arrows from the joining states and outgoing arrow towards the common goal state. We use the join notation when two or more states concurrently converge into one on the occurrence of an event or events.



Join

2.6. Self-transition

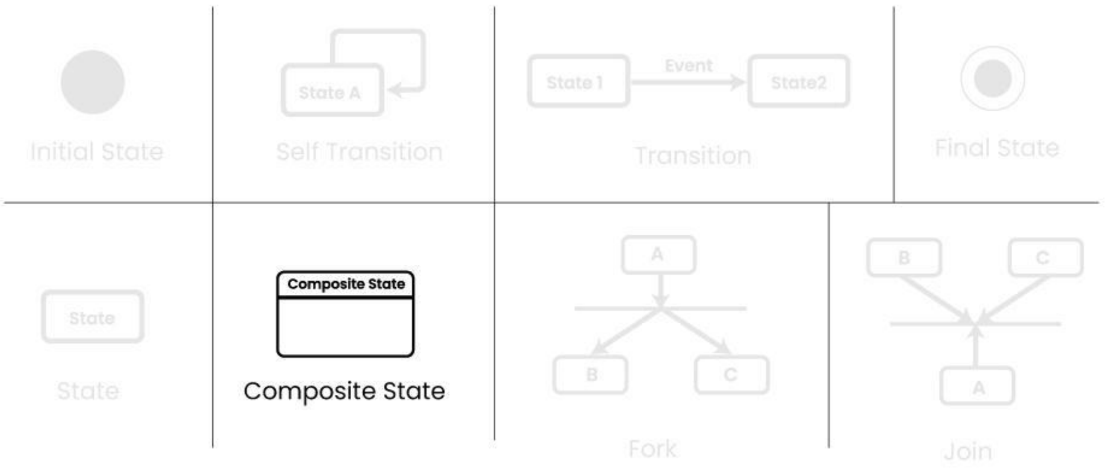
We use a solid arrow pointing back to the state itself to represent a self-transition. There might be scenarios when the state of the object does not change upon the occurrence of an event. We use self-transitions to represent such cases.



Self transition

2.7. Composite state

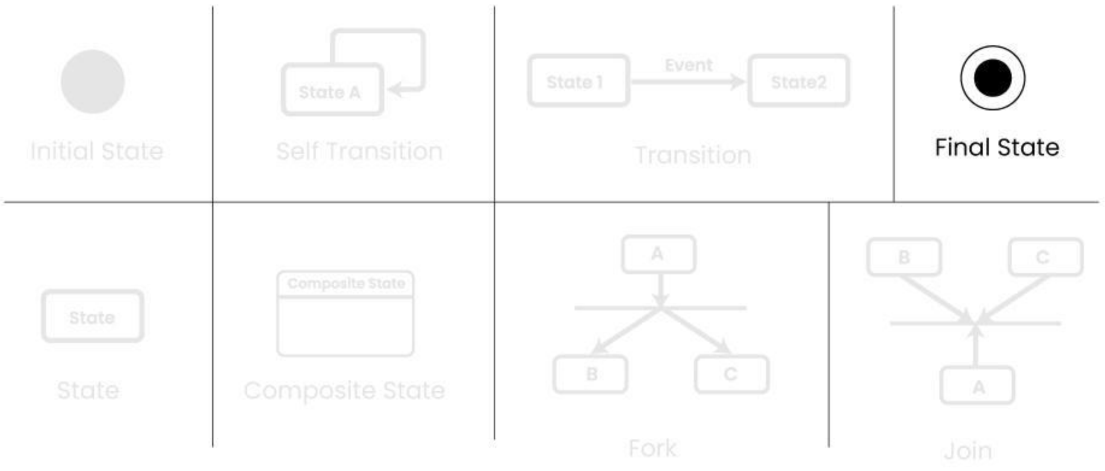
We use a rounded rectangle to represent a composite state also. We represent a state with internal activities using a composite state.



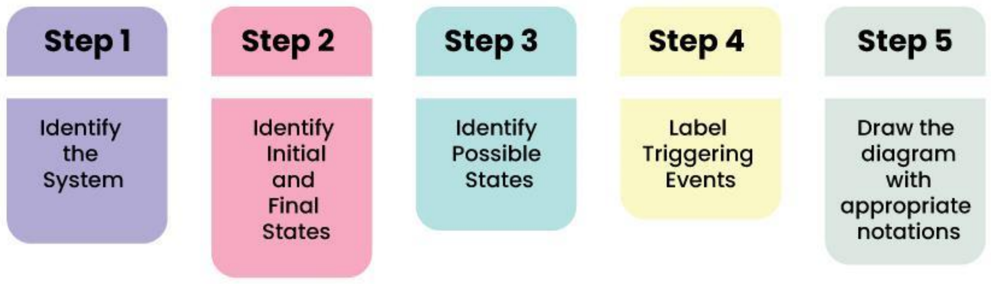
Composite State

2.8. Final State

We use a filled circle within a circle notation to represent the final state in a state machine diagram.

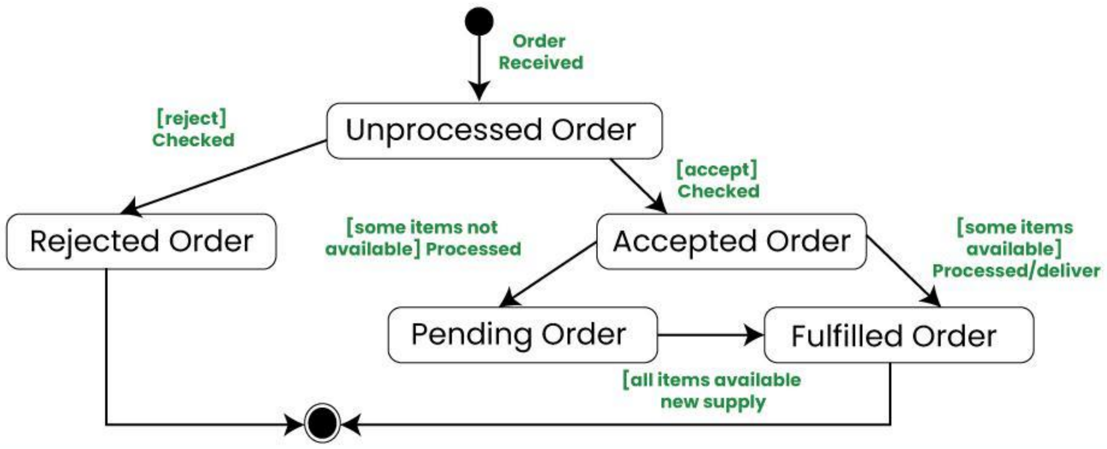


Final State



State Machine Diagrams | Unified Modeling Language (UML)

State machine diagram for an online order



State Machine Diagrams | Unified Modeling Language (UML)