Rules for Drawing UML State Machine Diagram

1. **Define the Scope and Context**:

   - Clearly specify the system, subsystem, or object being modeled (e.g., "Order Processing System" or "User Account").
   - Focus on a single entity or process to model its state changes in response to events.

2. **Identify States**:

   - Represent states as **rounded rectangles** with descriptive names (e.g., "Idle," "Processing," "Completed").
   - Use names that reflect the condition or status of the object at a specific point (e.g., "Logged In" or "Pending Approval").
   - Ensure states are mutually exclusive and collectively exhaustive for the modeled behavior.

3. **Mark Initial and Final States**:

   - Indicate the starting state with a **filled circle** (initial state) connected by an arrow to the first state.
   - Indicate the end state with a **filled circle inside a larger circle** (final state).
   - Ensure at least one initial state and optionally one or more final states.

4. **Model Transitions**:

   - Draw **solid arrows** between states to represent transitions triggered by events.
   - Label transitions with the format: `event [guard] / action` (e.g., "submitOrder [valid] / processOrder").
   - Include:
     - **Event**: The trigger causing the transition (e.g., "submitOrder").
     - **Guard** (optional): A condition in square brackets (e.g., "[valid]").
     - **Action** (optional): The operation executed during the transition (e.g., "processOrder").

5. **Use Composite States (Optional)**:

   - Represent complex states with **nested sub-states** inside a larger rounded rectangle (e.g., "Order Processing" containing "Validating" and "Packaging").
   - Label the composite state clearly and ensure sub-states have their own transitions.
   - Use an initial state within the composite state to indicate the entry point.

6. **Handle Entry and Exit Actions (Optional)**:

- Specify actions performed when entering or exiting a state using keywords: `entry / action` or `exit / action` inside the state rectangle.
- Example: In “Processing” state, `entry / startProcessing` or `exit / logCompletion`.

7. **Model Self-Transitions**:

   - Draw a **looping arrow** back to the same state for events that do not change the state but trigger an action (e.g., “refresh / updateDisplay”).
   - Label self-transitions with the event and action.

8. **Maintain Simplicity**:

   - Focus on **key states and transitions** relevant to the system’s behavior, avoiding excessive detail or low-level operations.

9. **Ensure Consistency with Requirements**:

   - Align states and transitions with **functional requirements** from the requirements engineering process (e.g., behaviors like “Process Order”).
   - Verify that events and states match scenarios or use cases identified during elicitation.
   - Exclude non-functional requirements (e.g., performance) unless they directly influence state behavior.