Rules for Drawing UML Activity Diagram

1. **Define the Scope and Context**:
   - Clearly specify the process or workflow being modeled (e.g., "Student Registration Process").
   - Focus on a single use case, scenario, or business process to keep the diagram targeted.
2. **Identify Start and End Points**:
   - Mark the start of the process with a **filled circle** (initial node).
   - Mark the end of the process with a **filled circle inside a larger circle** (final node).
   - Ensure each diagram has at least one initial node and one or more final nodes.
3. **Represent Activities**:
   - Draw activities (tasks or actions) as **rounded rectangles** with descriptive, action-oriented names (e.g., "Enter Credentials," "Validate Input").
   - Ensure activities reflect specific steps in the workflow, aligned with functional requirements.
4. **Connect Activities with Transitions**:
   - Use **solid arrows** (control flows) to connect activities, showing the sequence of execution.
   - Label transitions with conditions or events if needed (e.g., "[valid input]" or "submit clicked").
5. **Model Decision Points**:
   - Represent decisions (branching points) as **diamonds** with multiple outgoing arrows.
   - Label each outgoing arrow with a guard condition in square brackets (e.g., "[login successful]" or "[invalid credentials]").
   - Ensure all decision paths lead to valid activities or merge points.
6. **Use Merge Points**:
   - Represent the convergence of multiple paths (e.g., after a decision) with a **diamond** (merge node) that has multiple incoming arrows and one outgoing arrow.
   - Ensure merge nodes reconnect alternative paths into the main flow.
7. **Incorporate Fork and Join for Concurrency**:

- Use a **thick horizontal bar** (fork) to split the flow into concurrent activities.
- Use a **thick horizontal bar** (join) to synchronize concurrent activities back into a single flow.
- Ensure all forked paths are joined before proceeding to the next activity.

8. **Include Swimlanes for Responsibility**:
   - Divide the diagram into **swimlanes** (vertical or horizontal partitions) to assign activities to specific actors, roles, or system components (e.g., "Student," "System").
   - Label each swimlane clearly with the responsible entity.
9. **Model Object Flows (Optional)**:
   - Represent data or objects passed between activities as **rectangles** with a name (e.g., "Registration Form").
   - Connect object flows to activities using **dashed arrows** to show input or output data.
   - Example: "Registration Form" as input to "Validate Form."
10. **Handle Exceptions and Interruptions**:
    - Model exceptions or interruptions (e.g., "System Error") using **lightning bolt arrows** leading to an exception handler activity or final node.
    - Alternatively, use a **note** to describe exception conditions (e.g., "Error if database unavailable").
11. **Maintain Simplicity**:
    - Focus on **key activities and flows**, avoiding excessive detail or low-level implementation steps.
12. **Ensure Consistency with Requirements**:
    - Align activities and flows with **functional requirements** from the requirements engineering process (e.g., services like "Process Registration").
    - Verify that actors or roles in swimlanes match stakeholders identified during elicitation.
    - Exclude non-functional requirements (e.g., performance) unless they directly influence the workflow.