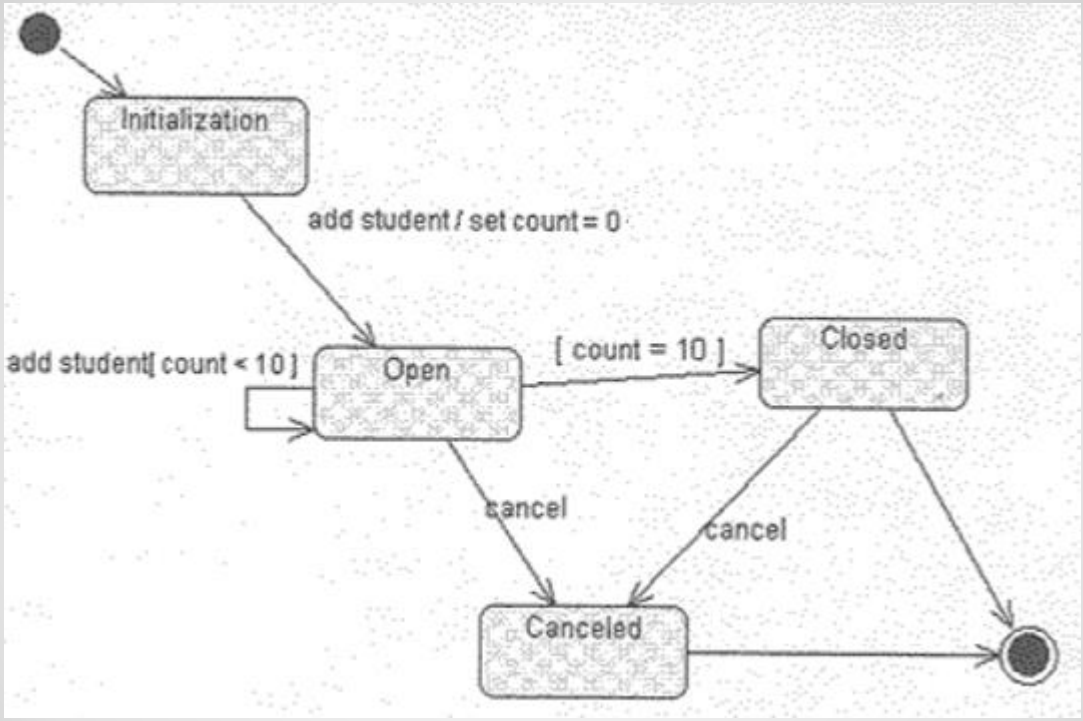
State Machines

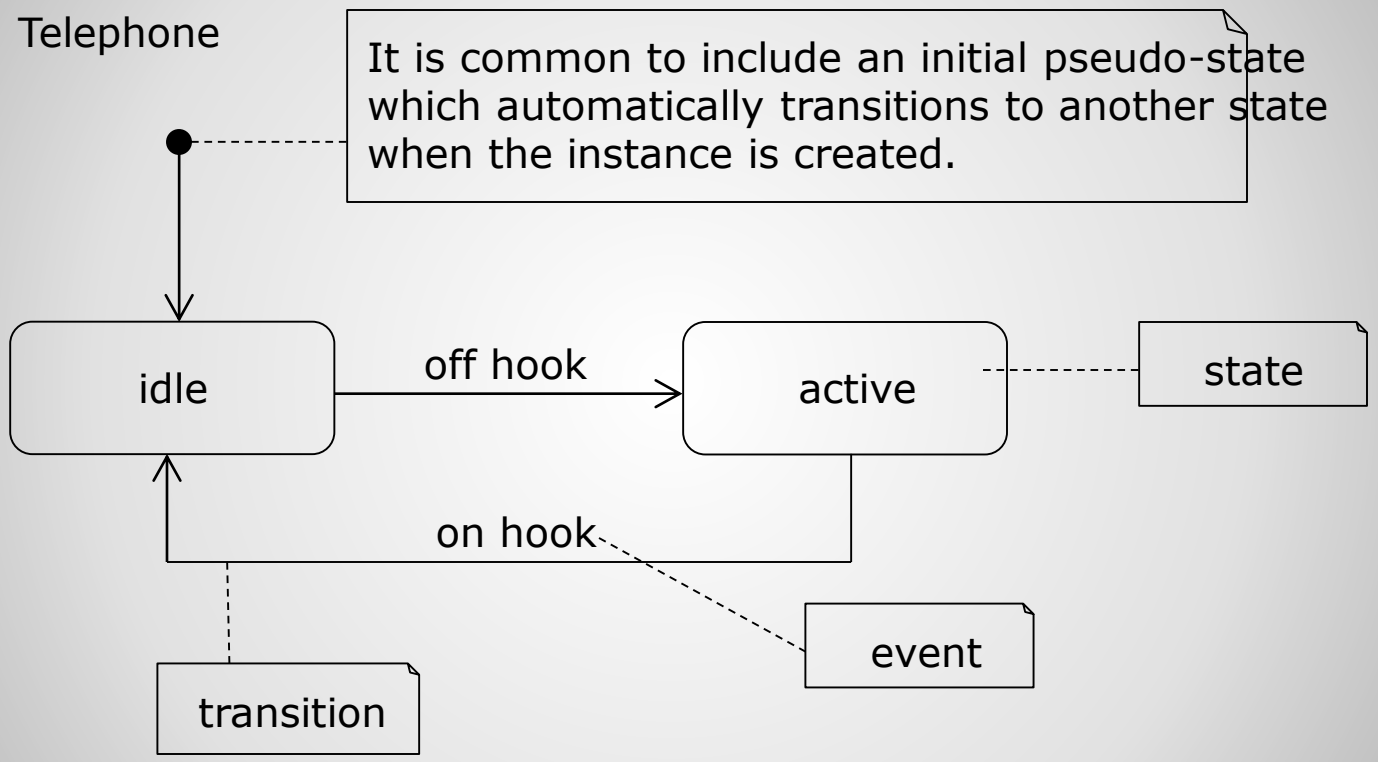
Models the behavior of an object

- A CourseOffering object may be open (able to add a student), cancelled (no longer offered) or closed(maximum number of students already assigned to the CourseOffering object). The state depends upon the number of students assigned to the particular CourseOffering object. The state can add student less then 10 and if students are equals to 10 than the course offering state will be closed. When the state is cancelled then the Course Roaster instance will be deleted.


Initialization
add student / set count = 0
add student[count < 10]
Open
[count = 10]
Closed
cancel
cancel
Canceled

Statechart Diagrams –role

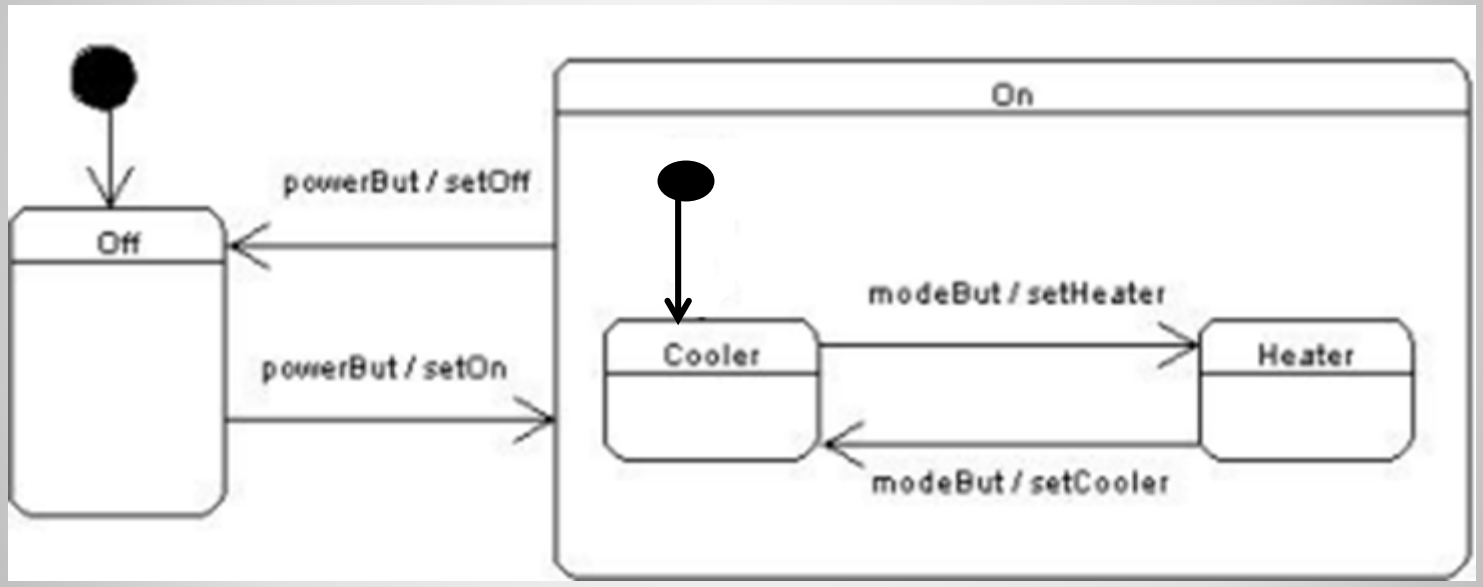
- **Statechart diagram**, also called **state machine**, is one of the five UML diagrams used to model dynamic nature of a system. It defines different states of an object during its lifetime and these states are changed by events.
- Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered
- Statechart diagrams are useful to model reactive systems. Reactive system can be defined as a system that responds to external or internal events.
- A state diagram (also state transition diagram) illustrates the events and the states of things: use cases, people, transactions, objects, etc.



State Machine Diagrams

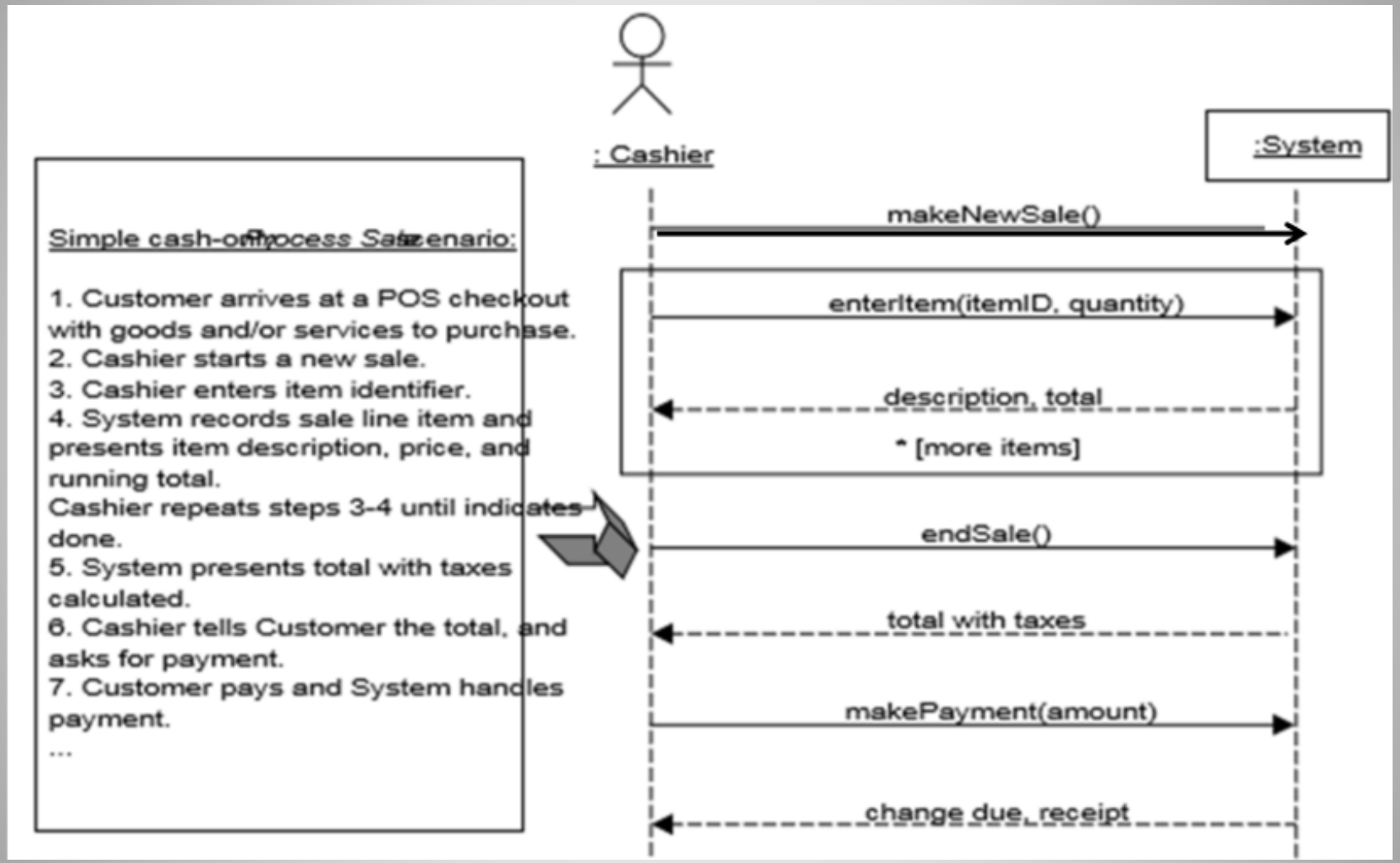
```
class AirCon { // context class
  public static final int off  = 1;
  public static final int on = 2;
  public static final int cooler = 3;
  public static final int heater  = 4;
  public int state;   // state variable
  public int on_subState;
AirCon() {  //constructor
  state = off;
  on_subState = cooler;
 }
public void modeBut() {  // event method
  switch (state) {
        case off :
              break;
        case cooler :
              setHeater;    // action
              // exit actions
              on_subState = Heater;
              state = on_subState;
              // entry actions
              break;
        case heater :
              setCooler;   // action
              // exit actions
              on_subState = Cooler;
              state = on_subState;
              // entry actions
              break;
        default :
              break;
  }
```

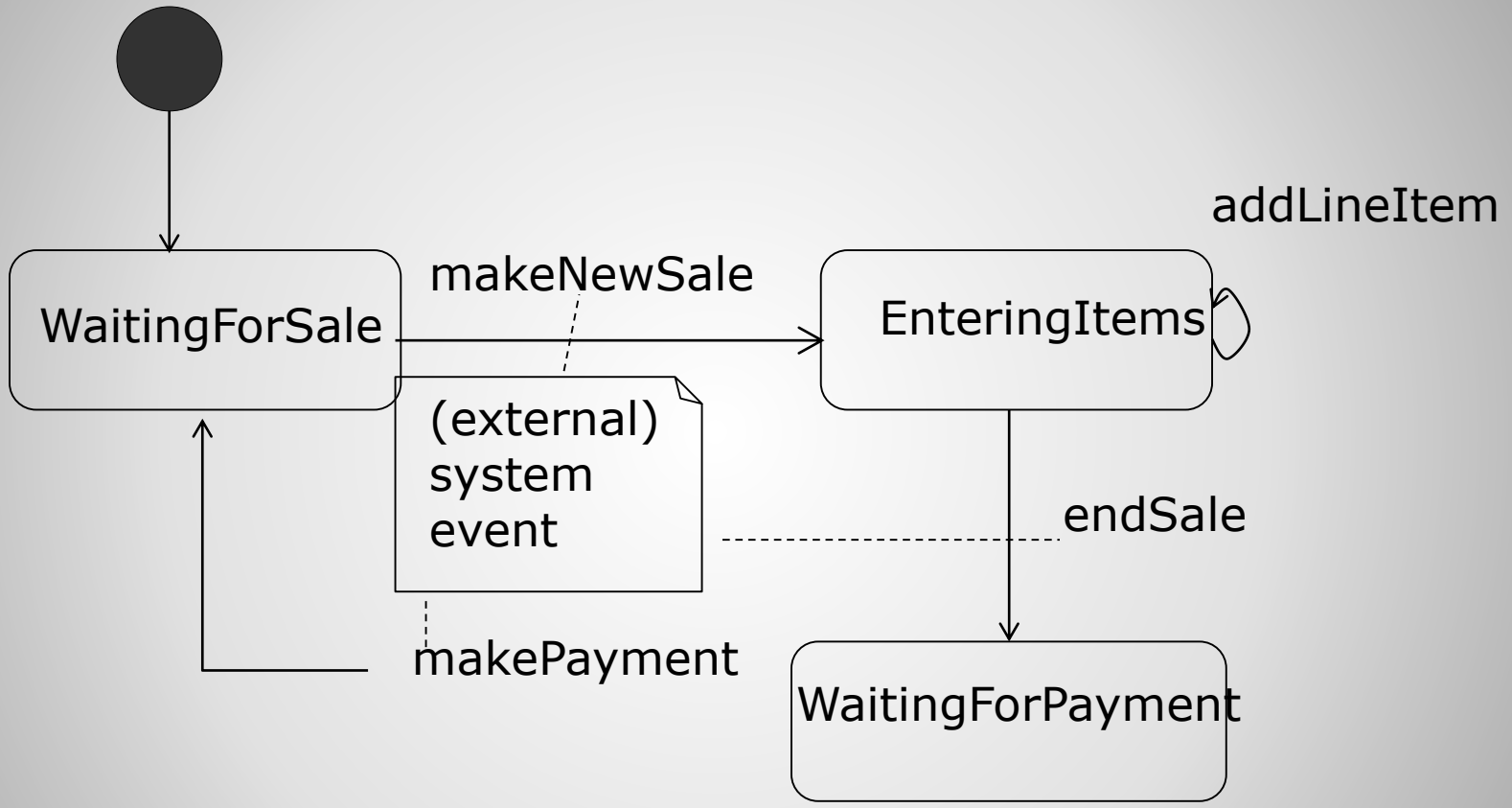
```
public void powerBut() {  // event method
  switch (state) {
        case off :
              setOn;   // action
              // exit actions
              state = on_subState;
              // entry actions
              break;
        case cooler :
              setOff;   // action
              // exit actions
              state = off;
              // entry actions
              break;
        case heater :
              setOff;    // action
              // exit actions
              state = off;
              // entry actions
              break;
        default :
              break;
  }
.....
}
```


On
powerBut / setOff
Off
modeBut / setHeater
Cooler
Heater
powerBut / setOn
modeBut / setCooler

State Machine Diagrams

- A state diagram may be applied to a variety of UML elements, including:
  - classes (conceptual or software)
  - use cases
- Since an entire system can be represented by a class, a statechart diagram may be used to represent it.
- Any UP element (Domain Model, Design model, etc.) may have deploy state diagrams to model its dynamic behavior in response to events.


Simple cash-only Process Sale scenario:
1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
...
: Cashier
:System
makeNewSale()
enterItem(itemID, quantity)
description, total
* [more items]
endSale()
total with taxes
makePayment(amount)
change due, receipt


addLineItem
WaitingForSale
makeNewSale
EnteringItems
(external)
system
event
endSale
makePayment
WaitingForPayment

State-independent and State Dependent Objects

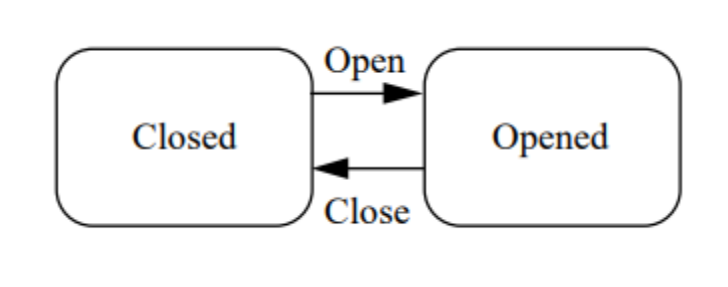
- An entity is considered to be *state-independent* if it responds in the same manner to all events.
- An entity is considered to be state-dependent if it responds differently to events.
- State diagrams are created for state-dependent entities with complex behavior.

- Statecharts are typically used in conjunction with interaction diagrams (usually sequence diagrams)
  - A statechart describes all events (and states and transitions for a single object)
  - A sequence diagram describes the events for a single interaction across all objects involved

Classes that Benefit from Statechart Diagrams

Example

- If a Bank Account was Closed and it saw an Open event, it would end up in the Opened state - If the account was Opened and it saw a Close event it would end up in the Closed state.



Classes that Benefit from Statechart Diagrams

Common State-dependent Classes

- Use cases
- Systems
- Windows
- Transaction. Dependent on its current state within the overall life-cycle.

- External event: caused by something outside a system boundary.
- Internal event: caused by something inside the system boundary.
- Temporal event: caused by the occurrence of a specific date and time or passage of time.

Illustrating External and Internal Events

- **State** - is a circumstance in which an object is situated in its life cycle, when it meets a certain condition, performs an operation or waits for an event to occur.
- **Transition** is a relationship between two states, indicating that the object being in the first state will perform certain actions and move to the other state, whenever a specific event occurs and certain conditions are met.
- **Initial state** is the initiation of the statechart diagram
- **Final state** is the termination of the statechart diagram

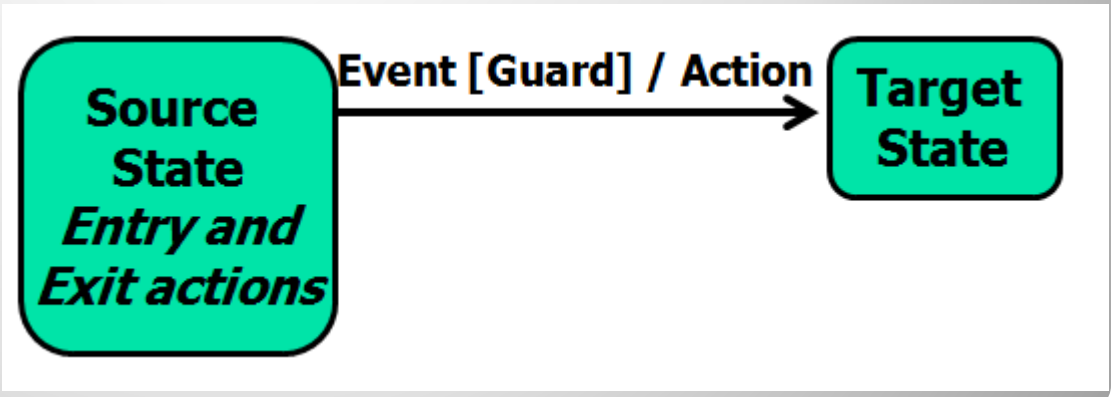
Statechart Diagrams

States

- State is a condition or situation during the life of an object within which it performs some activity, or waits for some events .
- State may also label activities, e.g., do/check item
- Examples of object states are:
  - The invoice (object) is paid (state).
  - The car (object) is standing still (state)
  - The engine (object) is running (state)
  - Jim (object) is playing the role of a salesman (state)

- States in state chart diagrams represent a *set of those value combinations,* in which an object *behaves the same in response to* events.
- Therefore, not every modification of an attribute leads to a new state.

- Show what the dependencies are between the state of an object and its reactions to messages or other events
- A solid arrow represents the path between different states of an object.
- Label the transition with the event that triggered it and the action that results from it.
- A state can have a transition that points back to itself.

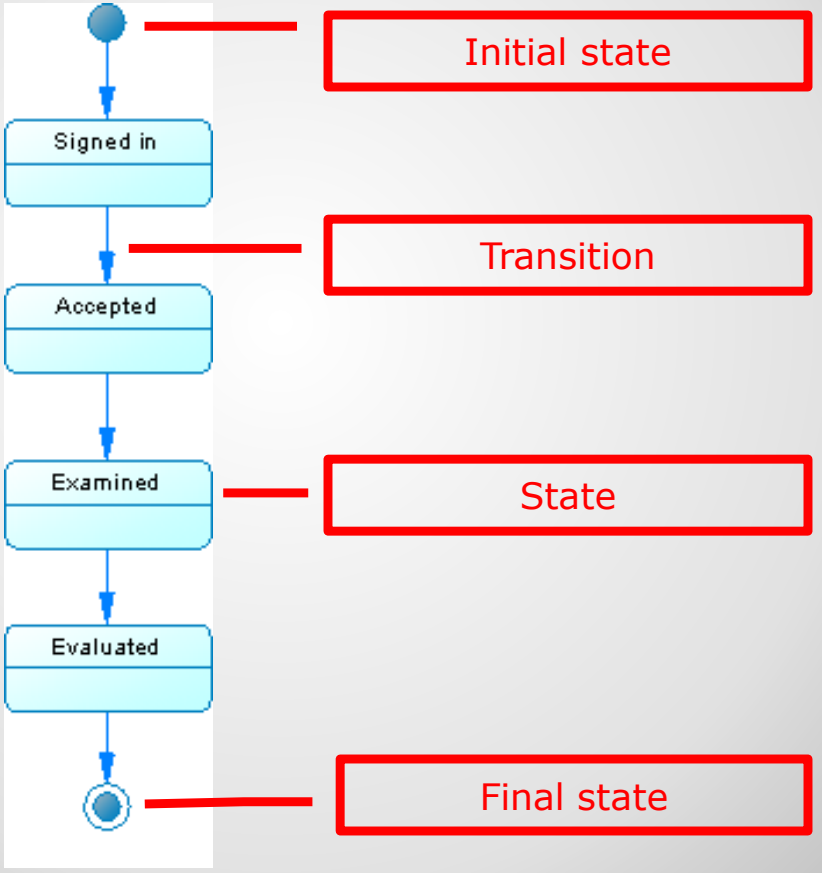


Transition

Events

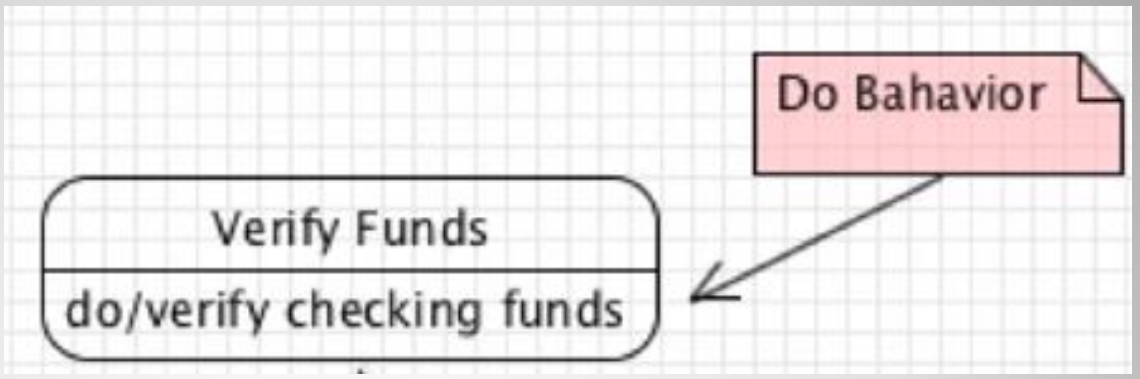
- An object transitions (changes) from one state to another state when something happens, which is called an event; for example, someone pays an invoice, starts driving the car,
- An event is an instant in time that may be significant to the behavior of the objects in a class - Events can have associated arguments
- Events are written simply as text strings

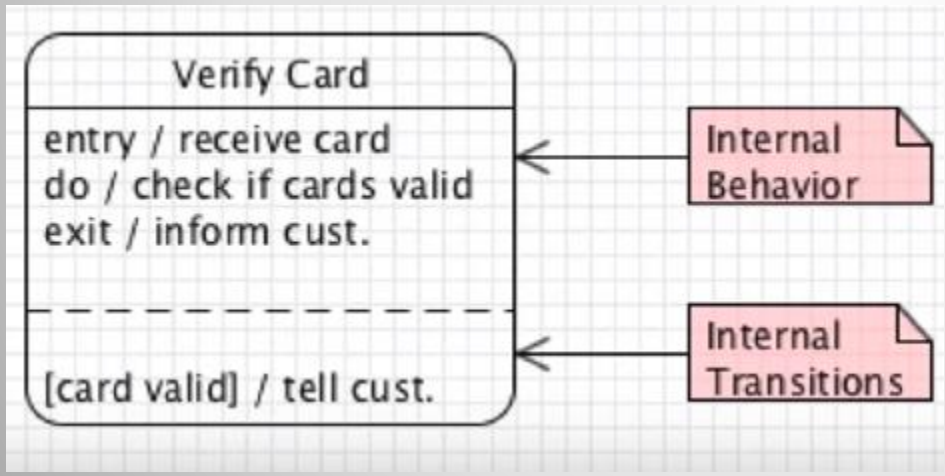
Statechart Diagrams –basic concepts



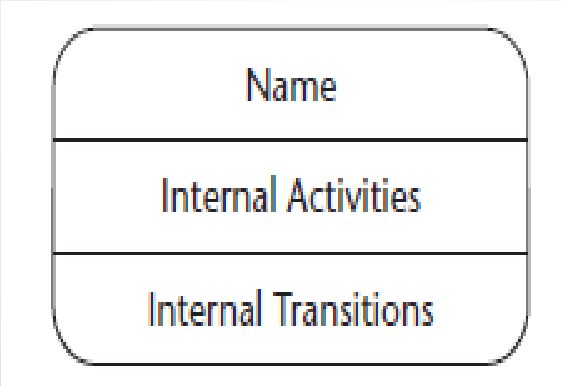
State Compartments

- A state may contain three kinds of compartments.
- The first compartment shows the name of the state, for example, idle, paid, and moving.
- The second compartment is the optional **activity compartment**, which lists behavior in response to events. You can define your own event, such as selecting a Help button, as well as the activity to respond to that event.
- Three standard events names are reserved in UML: entry, exit, and do.
- The entry event can be used to specify actions on the entry of a state; for example, assigning an attribute or sending a message.
- The exit event can be used to specify actions on exit from a state.
- The do event can be used to specify an action performed while in the state; for example, sending a message, waiting, or calculating.


Do Bahavior
Verify Funds
do/verify checking funds


Verify Card
entry / receive card
do / check if cards valid
exit / inform cust.
[card valid] / tell cust.
Internal Behavior
Internal Transitions

- The third compartment is the optional internal transition compartment. This compartment contains a list of internal transitions. A transition can be listed more than once if it has different guard conditions.

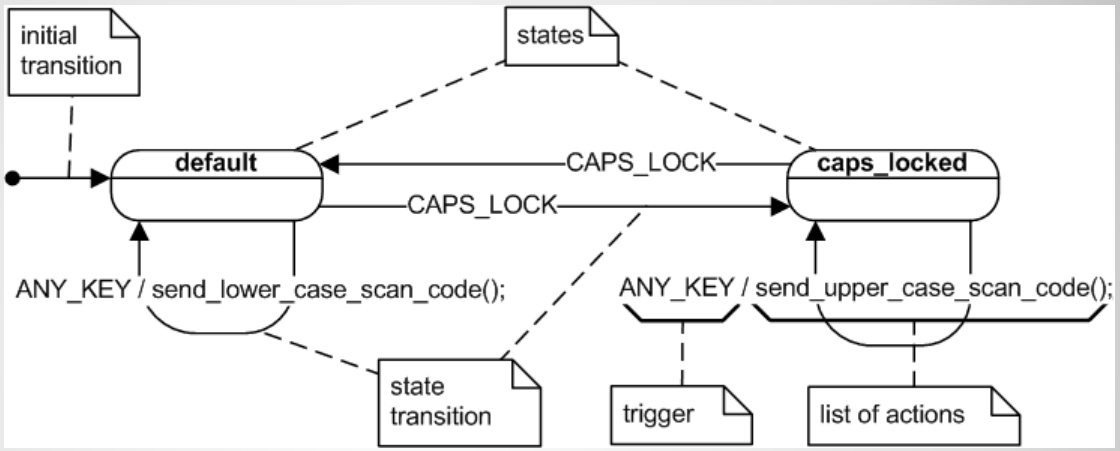


State Compartments

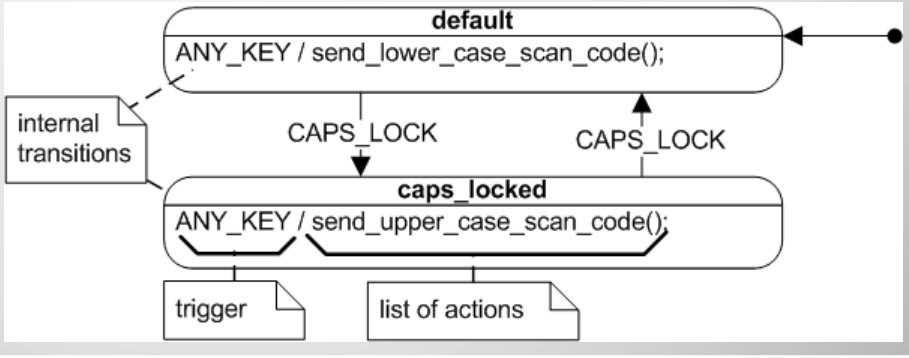
- An event causes only some internal actions to execute but does not lead to a change of state (state transition). In this case, all actions executed comprise the **internal transition**.

Internal transitions

Self-transition



Internal transition

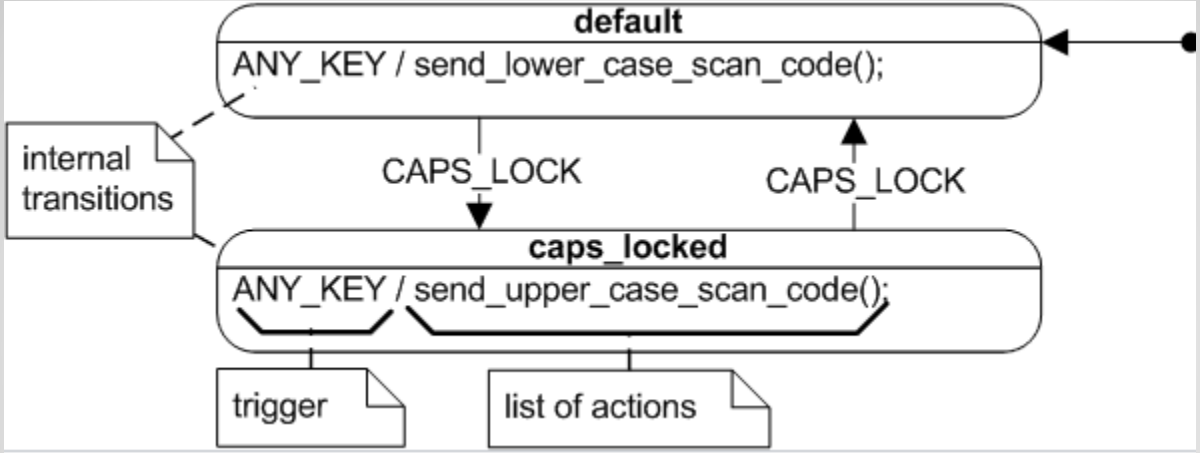


Types of Events

- UML has four types of events, also referenced as triggers because the event triggers behavior that depends on how you build the system.
- A condition becoming true. This is shown as a guard-condition on a state transition.
- Receipt of an explicit signal from another object. The signal is itself an object. This type of event is called a message.
- Receipt of a call on an operation by another object (or by the object itself). This type of event is also called a message.
- Passage of a designated period of time
  - **time event** – event, which occurs after a certain period of time (e.g. *after* (3 hrs.)),
  - **status change event** – event, which occurs at a certain point of time (e.g. *when* (date = 1 July)

Guard Condition

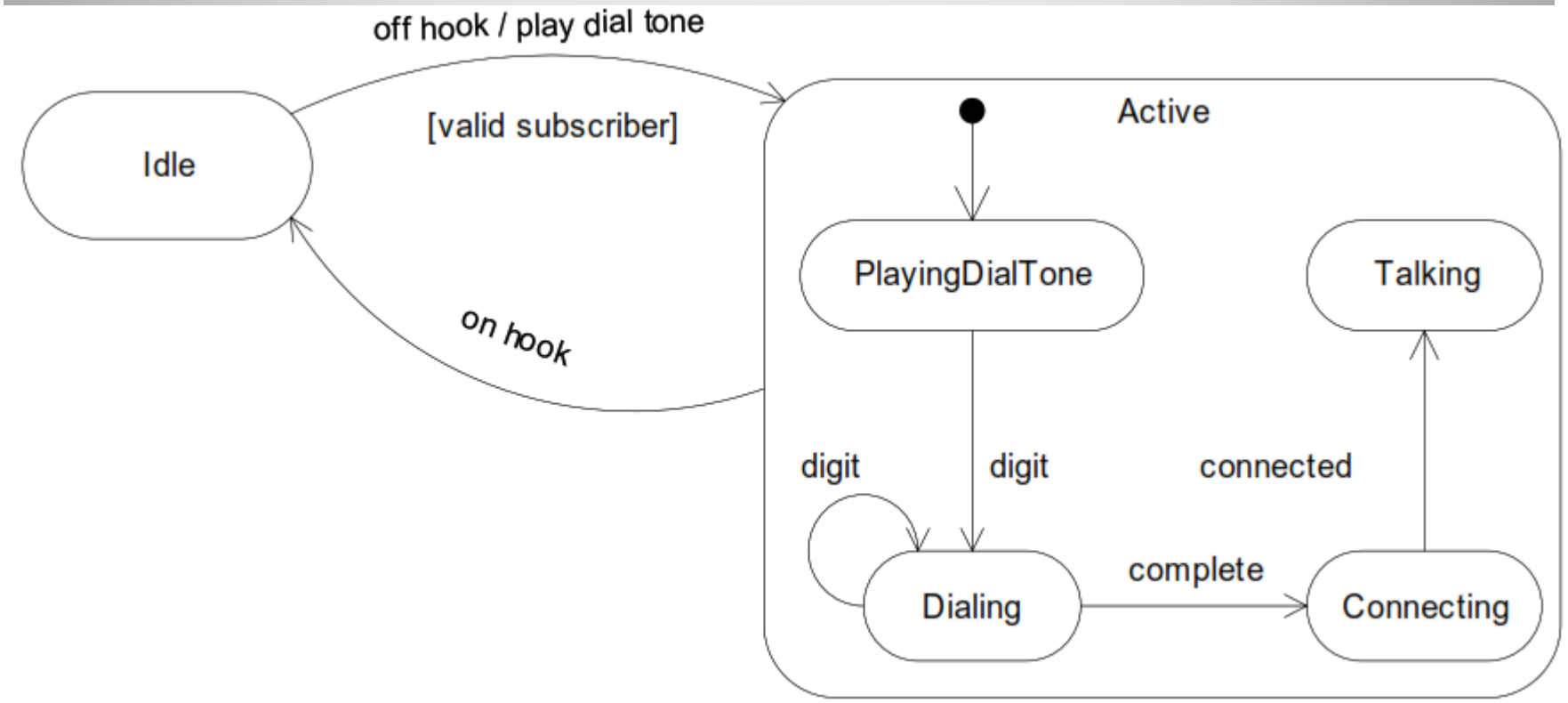
- Guard-condition is a Boolean expression placed on a state transition. If the guard-condition is combined with an event-signature, the event must occur, *and the guard-condition must be true for the transition to fire.*
- *If only a guardcondition* is attached to a state transition, the transition fires when the condition becomes true Examples of state transitions with a guard-condition are as follows:
  - [t = 15sec]
  - [number of invoices > n]
  - withdrawal (amount) [balance >= amount]



Internal transitions

Composite State

- A state can be refined hierarchically by composite states.
- Generally, **composite state** is defined as **state** that has substates (nested **states**). Substates could be sequential (disjoint) or concurrent (orthogonal).
- A state allows nesting to contain substates. A substate inherits the transitions of its superstate (the enclosing state).
  - Within the *Active* state, and no matter what substate the object is in, if the *on hook* event occurs, a transition to the *idle* state occurs.


off hook / play dial tone
[valid subscriber]
Idle
on hook
Active
PlayingDialTone
Talking
digit
digit
connected
Dialing
complete
Connecting

- **Orthogonal region** – concept used for showing states and transitions within in a composite state, which are activated concurrently. Transitions can cross composite states boundaries. All orthogonal regions must be realized in order for the state to be completed.
- **Pseudo-state** is an abstract category of statechart diagram modeling, which allows for organizing complex transitions by the use of fork node, join node, junction, decision node, termination, etc.

Statechart Diagrams

Concurrency

- Concurrency on a state machine diagram can be expressed by an orthogonal state (a composite state with multiple regions).
- If an entering transition terminates on the edge of the orthogonal state, then all of its regions are entered.
- When exiting from an orthogonal state, each of its regions is exited

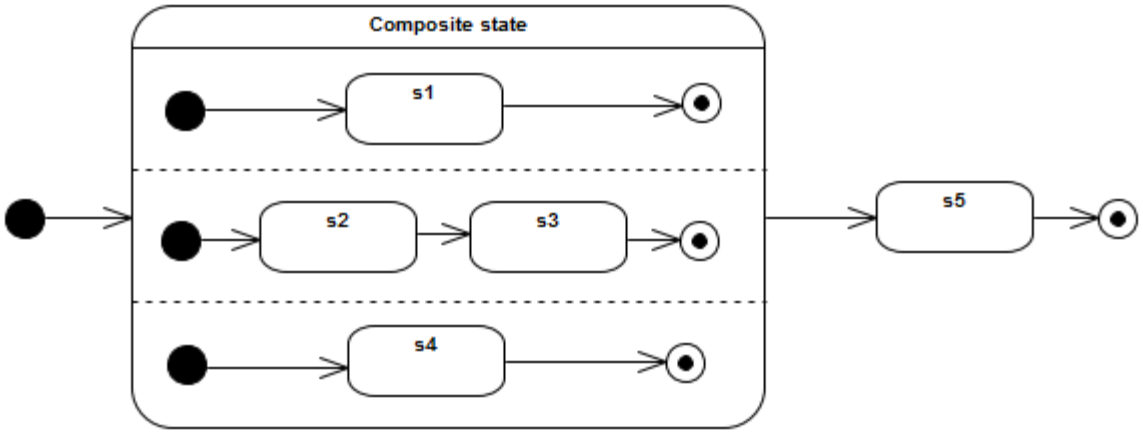
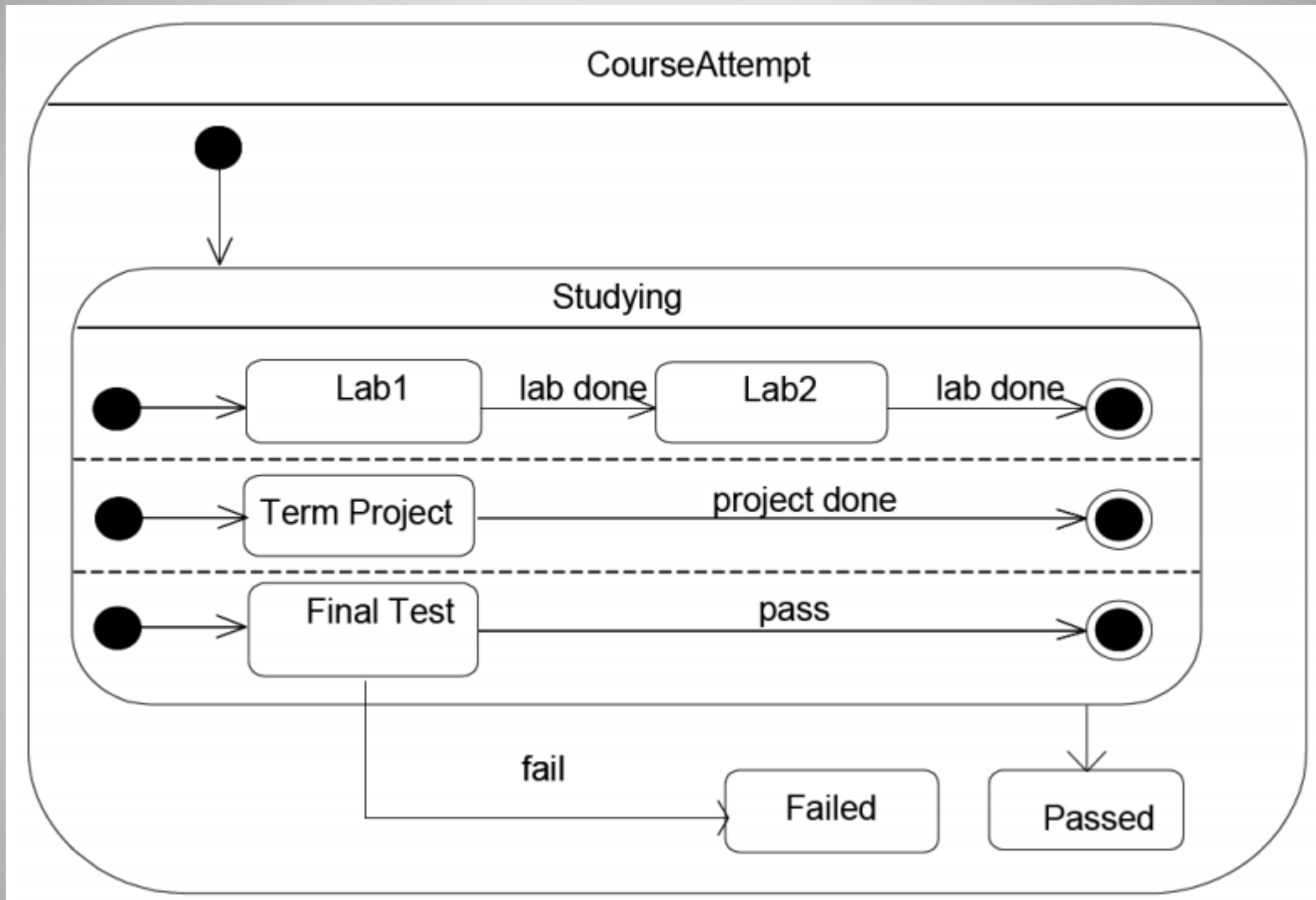


Figure 8. Orthogonal state


CourseAttempt
Studying
Lab1
lab done
Lab2
lab done
Term Project
project done
Final Test
pass
fail
Failed
Passed

Concurrency

- Concurrency can be shown explicitly using fork and join pseudostates.
- A fork is represented by a bar with one or more outgoing arrows terminating on orthogonal regions (i.e. states in different regions);
- a join merges one or more transitions.
- concurrent substates specify two or more state machines that execute in parallel in the context of the enclosing object
- Execution of these concurrent substates continues in parallel. These substates waits for each other to finish to joins back into one flow

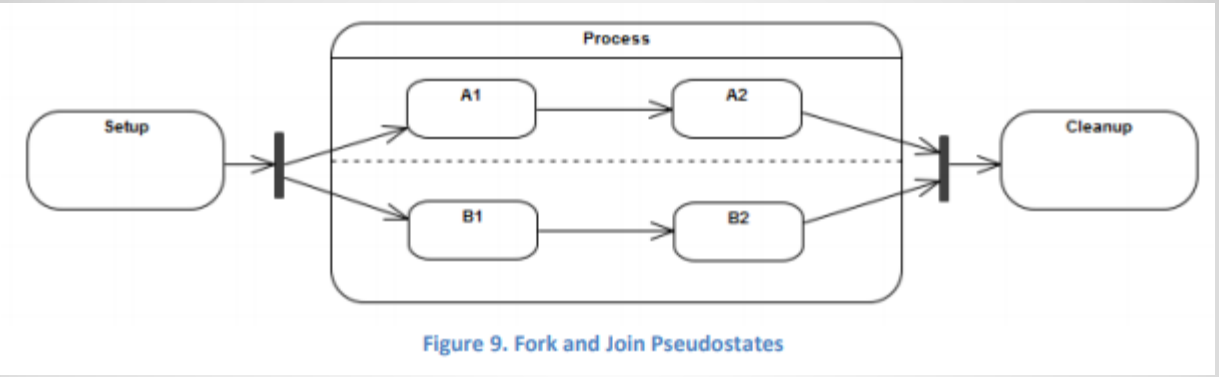


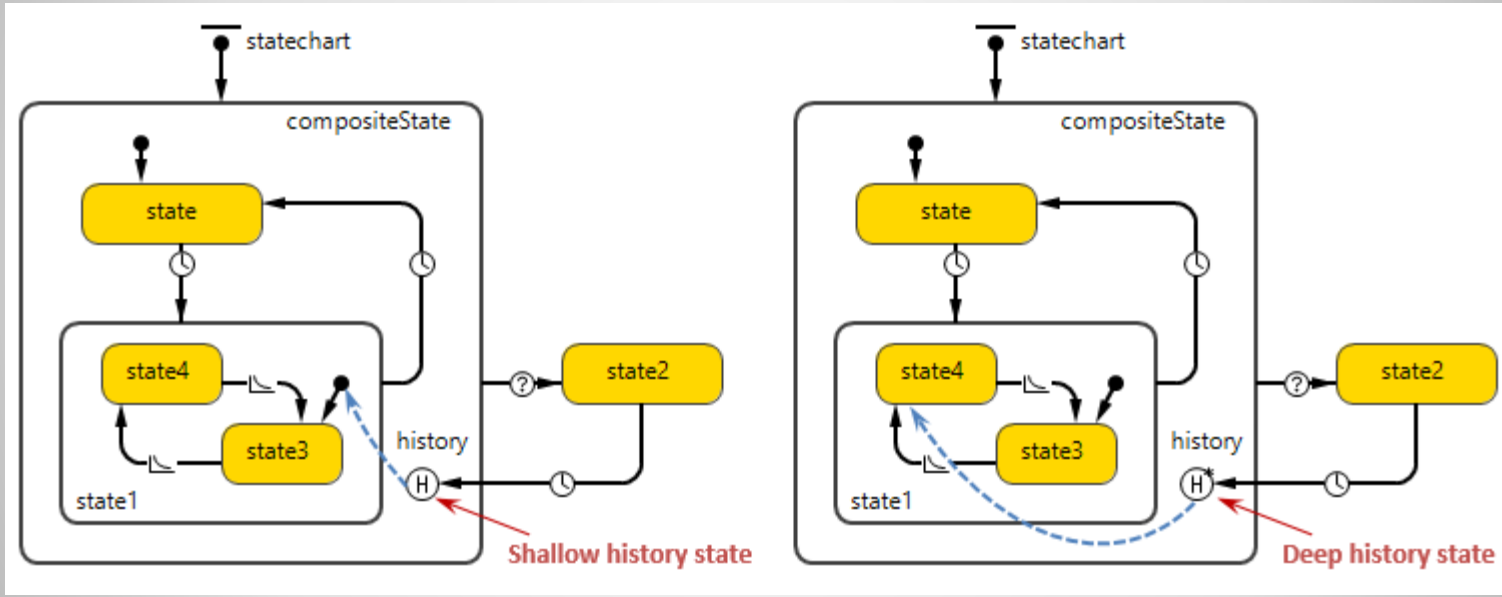
Figure 9. Fork and Join Pseudostates

Pseudo states

- Initial
- Junction
- Terminate
- Shallow history
- Deep history
- Entry
- Exit
- Choice
- Join
- Fork
- Final

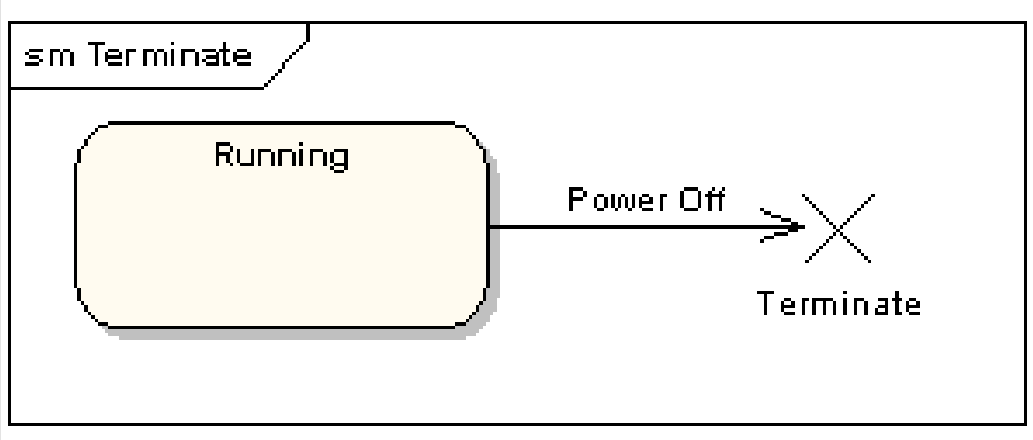
History

- **Shallow History**: *state* is a reference to the most recently visited state on the same hierarchy level within the composite state.
- **Deep History:** *state* is a reference to the most recently visited simple state within the composite state.


statechart
compositeState
state
state4
state3
state1
history
H
state2
Shallow history state
statechart
compositeState
state
state4
state3
state1
history
H*
state2
Deep history state

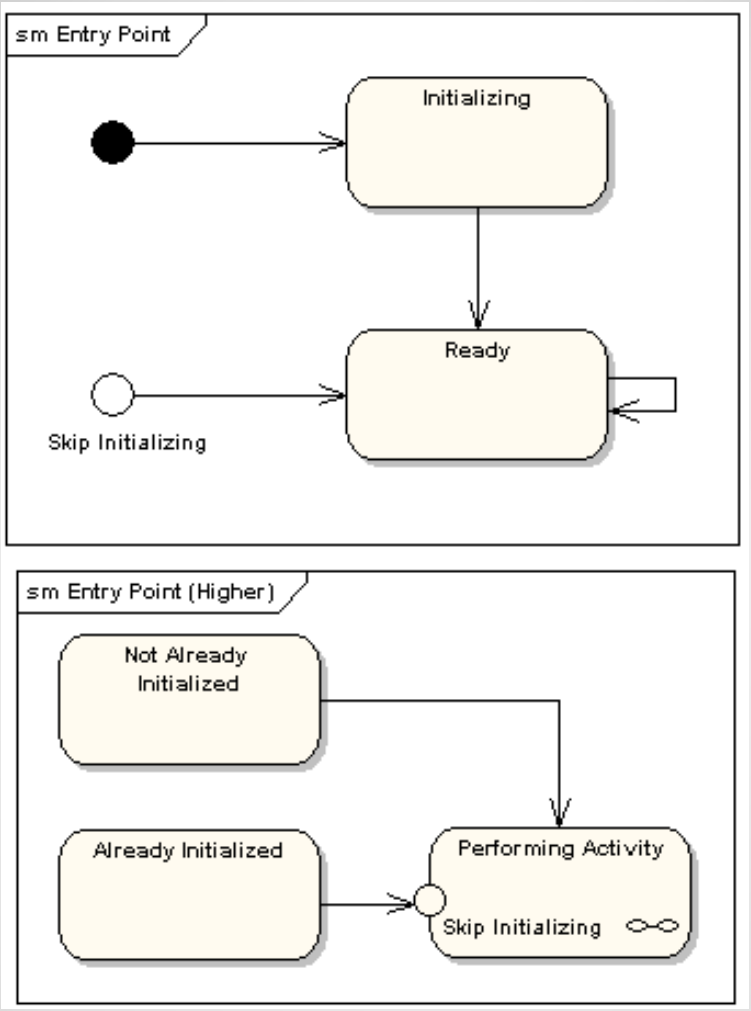
Terminate Pseudo state

- **Terminate pseudostate** implies that the execution of this state machine by means of its context object is terminated.
- The state machine does not exit any states nor does it perform any exit actions other than those associated with the transition leading to the terminate pseudostate.


sm Terminate
Running
Power Off
Terminate

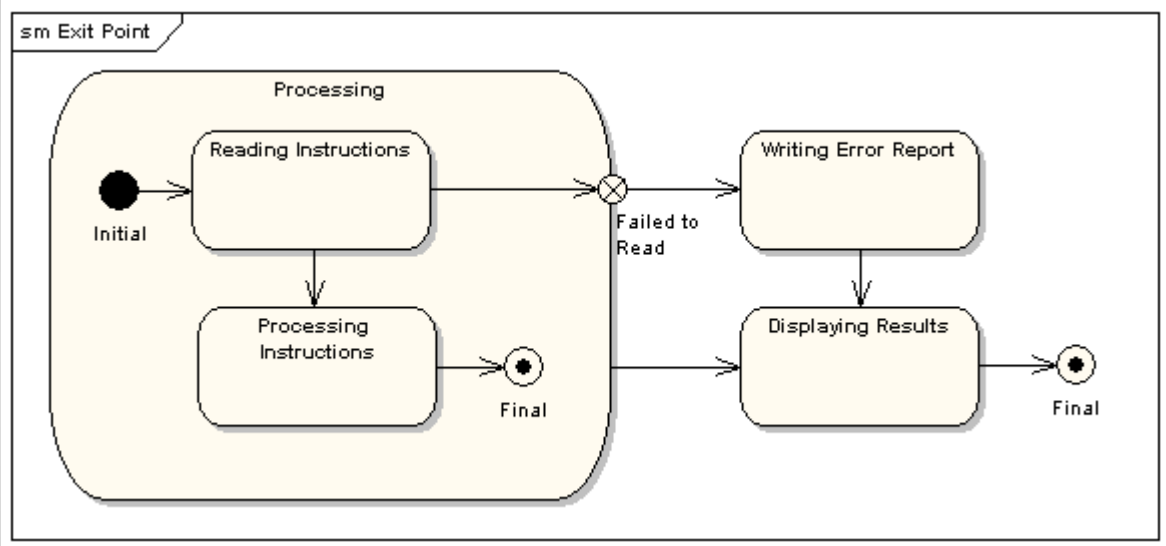
- Placed on the boundary or in a region of a composite state with a single outgoing transition to a substate. Used when there are multiple ways to enter a state and there is no single default substate for the transition to target.

Entry Pseudo state


sm Entry Point
Initializing
Ready
Skip Initializing
sm Entry Point (Higher)
Not Already Initialized
Already Initialized
Performing Activity
Skip Initializing

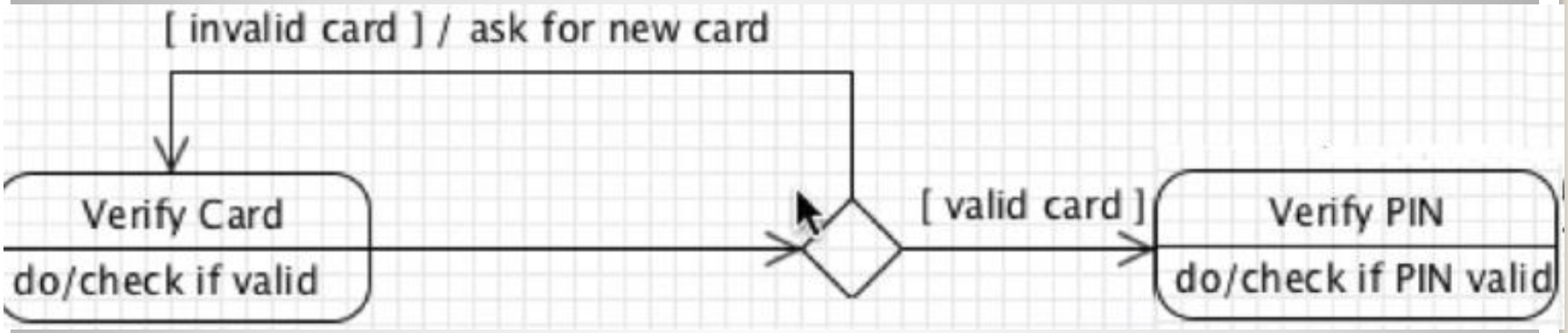
- In a similar manner to entry points, it is possible to have named alternative exit points. The state executes after the main processing state depends on which route is used to transition out of the state.

Exit Pseudo state


sm Exit Point
Processing
Reading Instructions
Initial
Processing
Instructions
Final
Failed to
Read
Writing Error Report
Displaying Results
Final

Choice Pseudo state

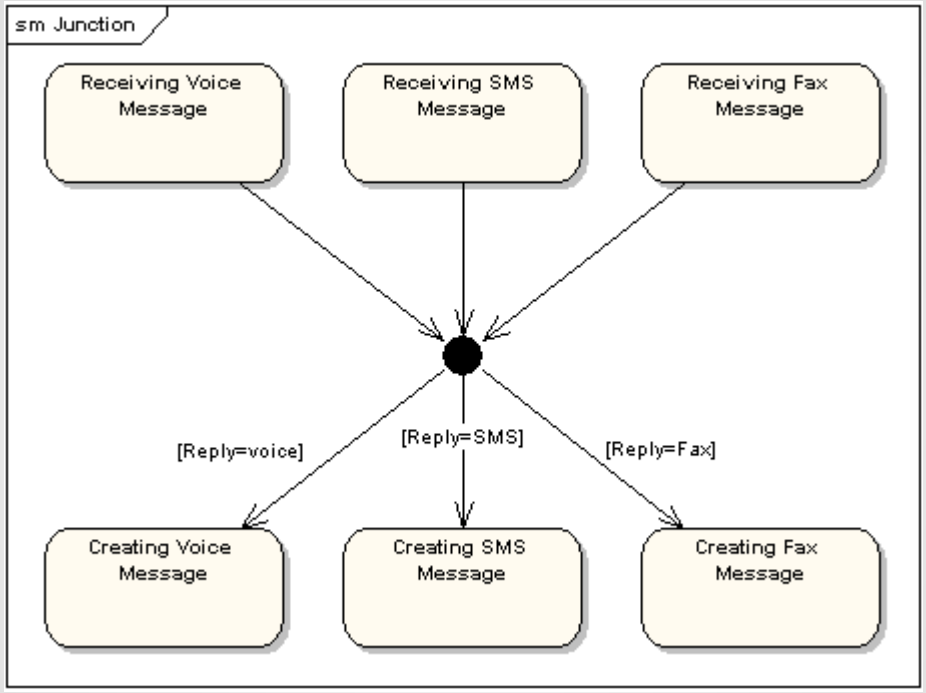
- **Choice pseudostate** realizes a dynamic conditional branch.
- It evaluates the guards of the triggers of its outgoing transitions to select only one outgoing transition.
- The decision on which path to take may be a function of the results of prior actions performed in the same run-to-completion step.
- Dynamic choices should be distinguished from static junction branch points.



Choice Pseudo state

Junction Pseudo state

- **Junction pseudostate** are used to chain together multiple transitions. They are used to construct compound transition paths between states. For example, a junction can be used to converge multiple incoming transitions into a single outgoing transition representing a shared transition path (this is known as a merge).
- Conversely, they can be used to split an incoming transition into multiple outgoing transition segments with different guard conditions. This realizes a static conditional branch.


sm Junction
Receiving Voice Message
Receiving SMS Message
Receiving Fax Message
[Reply=voice]
[Reply=SMS]
[Reply=Fax]
Creating Voice Message
Creating SMS Message
Creating Fax Message

Loan Request

- The statechart starts with accepted the loan application, If the loan passes the pre loan requirement, the loan is accepted otherwise loan is rejected. After the evaluation of pre loan approved requirement loan again can be approved or not approved. If the loan is approved, next, loan is closed after getting signature from the client and dispensing the cash. If the client doesn't sign the document within 14 days the loan is again rejected otherwise loan payments installments are further processed.

- The dark castle stored valuables in a safe. To show the key hole on the vault, must use the torch as a medium of information. This applies when the door is in the locked state. If the holes are already visible, the key can be entered to unlock the vault. As security procedures, these cabinets can be open if the torch attached. But if the torch they will not installed, it will produce monsters.

- We wish to model a basic oven with a door and a « start » button
- Once a meal has been put in the oven, the user pushes the « start » button. Once pushed, the meal is heated for 30 seconds. Then, the oven stops and plays a bell sound
- Alternatively, the user may open the door while the oven is heating. In that case, the heating process stops and the 30s delay is set to 0

Exercise: microwave oven

Exercise: Calculator

- Draw a statechart of a simple calculator
  - The interface of the calculator is composed of 10 buttons with digits, and 4 buttons with the basic operations (+, -, *, /)
  - The button “C” resets the display
  - The button “=” displays the result
  - Buttons “On” and “OFF”: try to guess.