



# CS-3004

## SOFTWARE ANALYSIS AND DESIGN

RUBAB JAFFAR

[RUBAB.JAFFAR@NU.EDU.PK](mailto:RUBAB.JAFFAR@NU.EDU.PK)

**Activity diagram**  
**Lecture # 19, 20, 21**  
**22, 23, 24**

# TODAY'S OUTLINE

- Paper review(Mid exam)
- Discuss and understand activity diagrams
- Understand the elements of activity diagrams
  - Activity
  - Transition
  - Synch. Bar
  - Decision Diamond
  - Start & Stop Markers
  - Fork & join
  - Swim lanes
  - Advance features in activity diagram
    - Object node
    - Structured activity
    - Expansion regions

# WHAT IS AN ACTIVITY?

- Two definitions
  - An activity is some task that needs to be done, whether by a human or a computer
  - An activity is a method in a class
- Activity arrangement
  - Sequential – one activity is followed by another
  - SDA ■ Parallel – two or more sets of activities are performed concurrently, and order is irrelevant

# ACTIVITY DIAGRAM DEFINITION

- Activity diagrams represent the dynamics of the system.
- They show the workflow of a system.
- They show
  - The flow of control from activity to activity in the system,
  - What activities can be done in parallel.
  - Alternate paths through the flow.
- Purpose
  - Model business workflows
  - Model operations

# ACTIVITY DIAGRAM

- Describes how activities are coordinated.
- Is particularly useful when you know that an operation has to achieve a number of different things, and you want to model what the essential dependencies between them are, before you decide in what order to do them.
- Records the dependencies between activities, such as which things can happen in parallel and what must be finished before something else can start.

# ACTIVITY DIAGRAM

- Activity Diagrams are like Flow Charts, but Flow Charts are usually limited to sequential activities while Activity Diagrams can show parallel activities as well
- Activity Diagrams are useful for describing complicated methods
- Activity Diagrams are useful for describing use cases, since, after all, a use case is an interaction, which is a form of activity

# ACTIVITY DIAGRAM SYMBOLS AND NOTATIONS

- An activity diagram has following individual elements

- Start Marker



- A black circle is the standard notation for an initial state before an activity takes place. It can either stand alone or you can use a note to further elucidate the starting point.

- Stop Marker



- The black circle that looks like a selected radio button is the UML symbol for the end state of an activity. As shown in two examples above, notes can also be used to explain an end state.

# ACTIVITY DIAGRAM SYMBOLS AND NOTATIONS


- Activity:

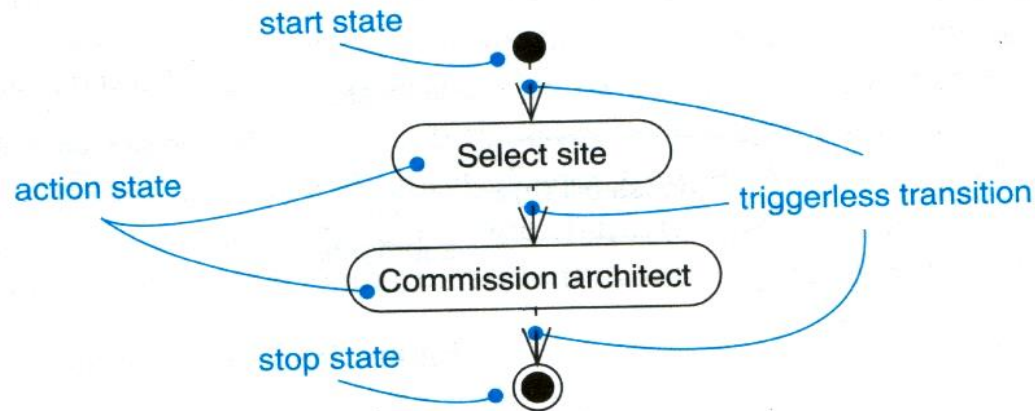


- The activity symbols are the basic building blocks of an activity diagram and usually have a short description of the activity they represent.
- Each activity can be followed by another activity (sequencing).



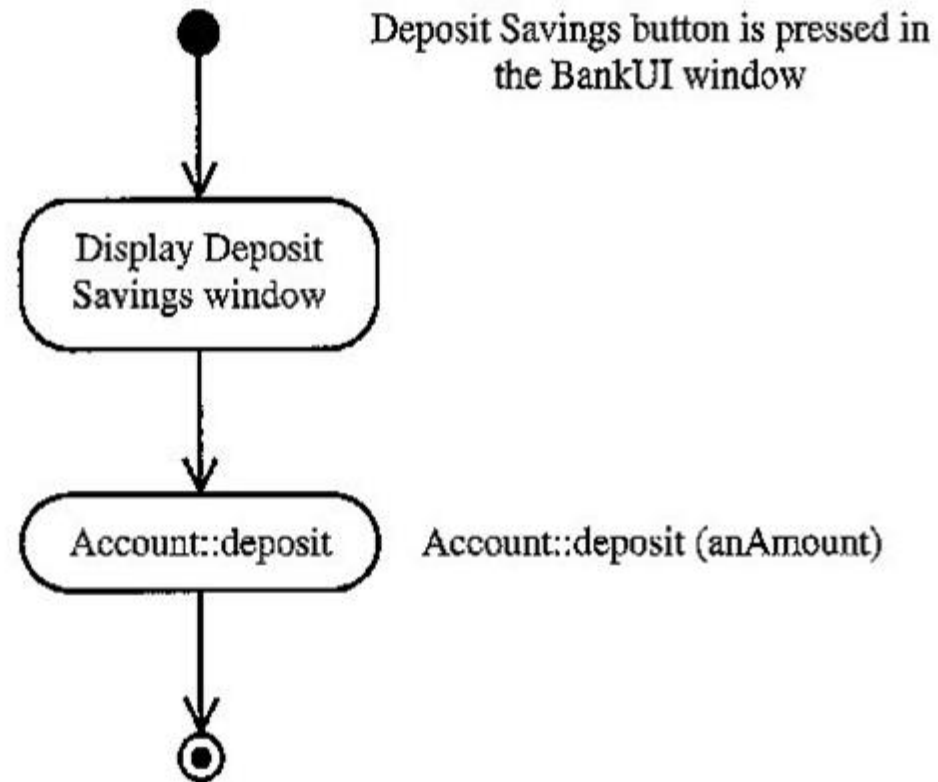
# ACTIVITY DIAGRAM SYMBOLS AND NOTATIONS

- Transition: 
- When the action or activity of a state completes, flow of control passes immediately to the next action or activity state
- Arrows represent the direction flow. The arrow points in the direction of progressing activities.



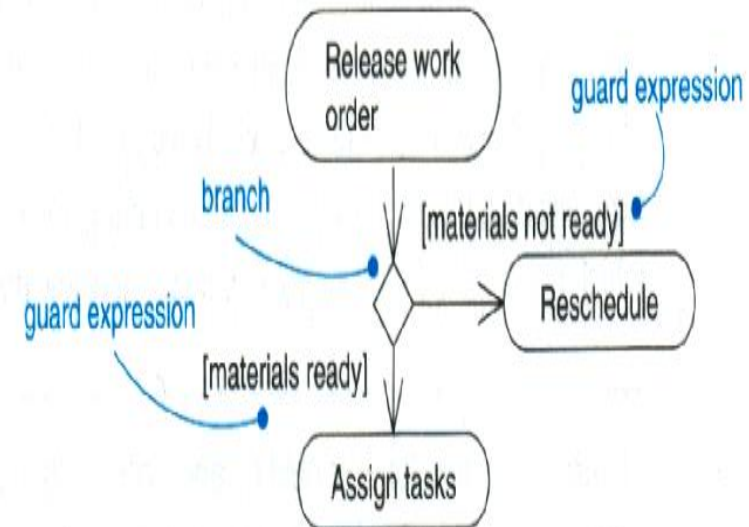
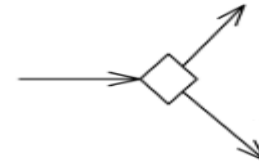
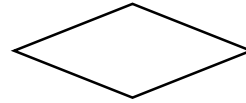
# ACTIVITY DIAGRAM: EXAMPLE (ACTIVITY AND TRANSITION)

Activity diagram for processing a deposit to a savings account.



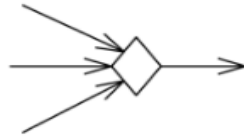
# ACTIVITY DIAGRAM SYMBOLS AND NOTATIONS

- Decision node / Branching
- Diamond.
  - Each trigger coming from it has a guard.
  - A branch specifies alternate paths taken based on some Boolean expression
  - A branch may have one incoming transition and two or more outgoing ones

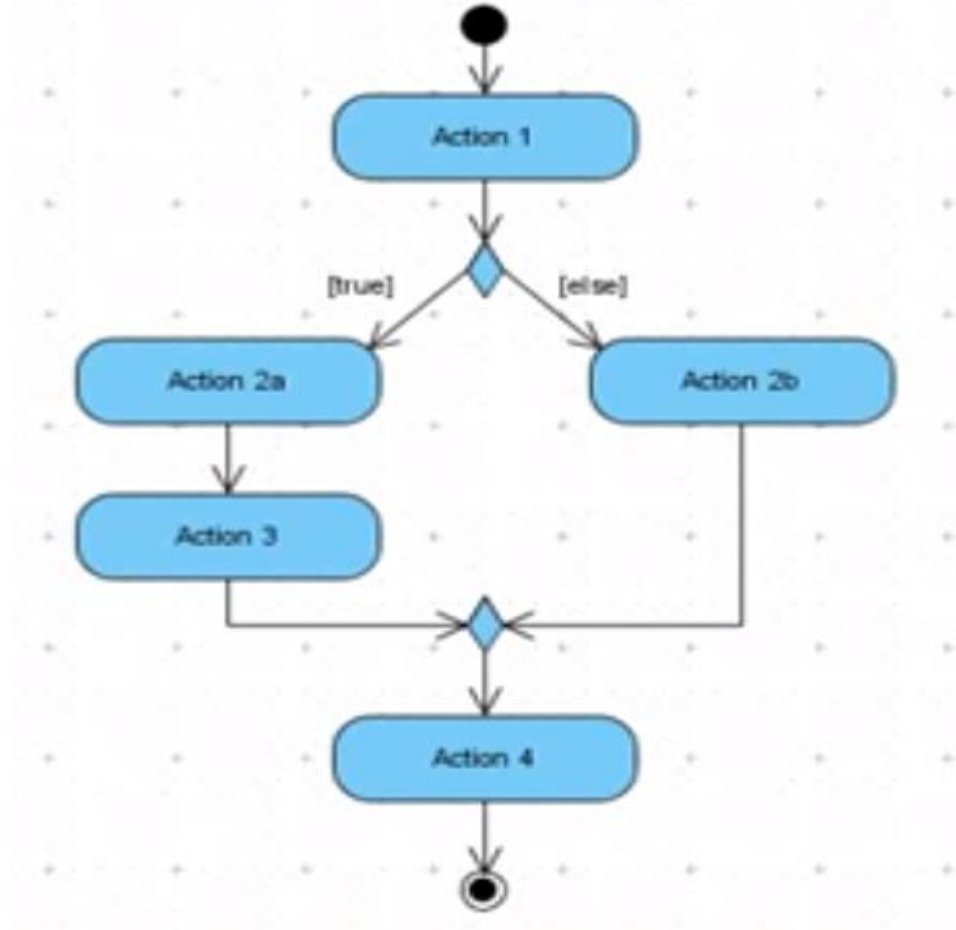


# ACTIVITY DIAGRAM SYMBOLS AND NOTATIONS

- Merge node :



- Merge node is a control node that brings together multiple incoming alternate flows to accept single outgoing flow. There is no joining of tokens. Merge should not be used to synchronize concurrent flows.



# ACTIVITY DIAGRAM SYMBOLS AND NOTATIONS

## ■ [Condition]

- Condition text is placed next to a decision marker to let you know under what condition an activity flow should split off in that direction.

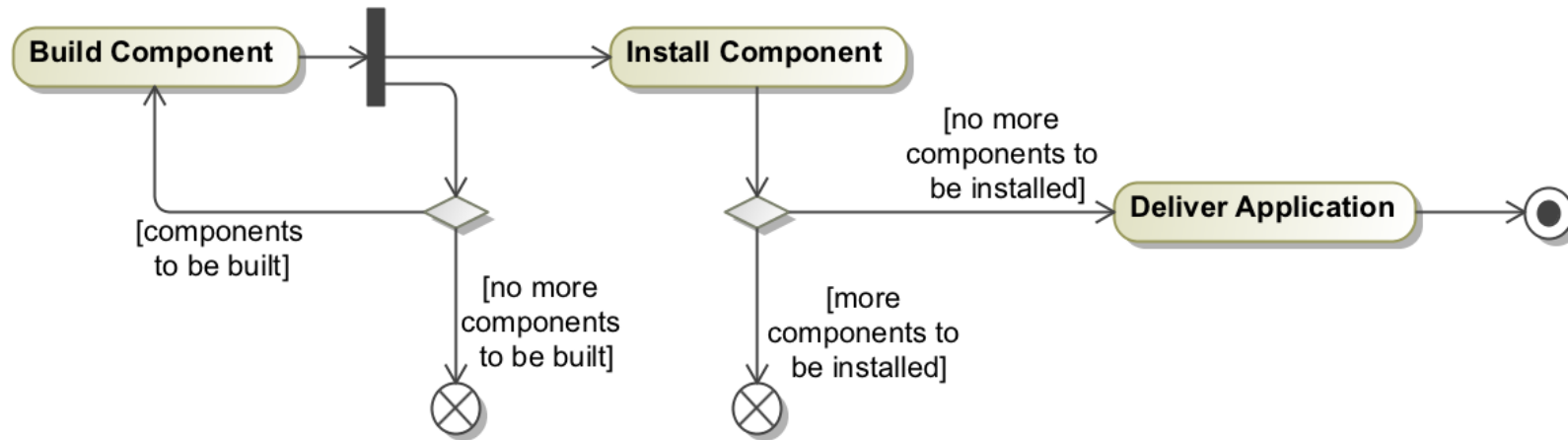
## ■ final flow

- The final flow marker shows the ending point for a process in a flow. Flow final node is a control final node that terminates a flow. It destroys all tokens that arrive at it but has no effect on other flows in the activity. The difference between a final flow node and the end state node is that the latter represents the end of all flows in an activity.

## ■ note

- The shape used for notes.

# FLOW FINAL NODE



# ACTIVITY DIAGRAM: EXAMPLE (BRANCH AND MERGE)

- We have to develop an activity diagram for HR department to manage the employee records.
- If the employee is a new employee, his/ her record is created. And if the employee is an existing employee, his record can be updated. Record is deleted if the employee is terminated employee.
- Final record is updated after any modification.

# ACTIVITY DIAGRAM SYMBOLS AND NOTATIONS

- Synchronisation bar:

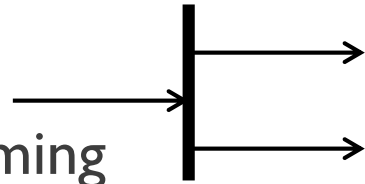


- All triggers from this attach to activities that can occur in parallel, with no specific sequence, or concurrently.
- The next synchronisation bar closes the concurrency.
- Use a synchronization bar to specify **the forking** and **joining** of parallel flows of control.
- A synchronization bar is rendered as a thick horizontal or vertical line



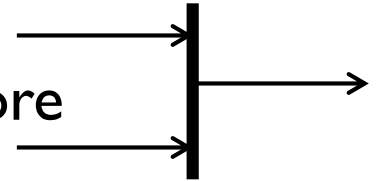
# FORK AND JOIN

- A fork may have one incoming transitions and two or more outgoing transitions



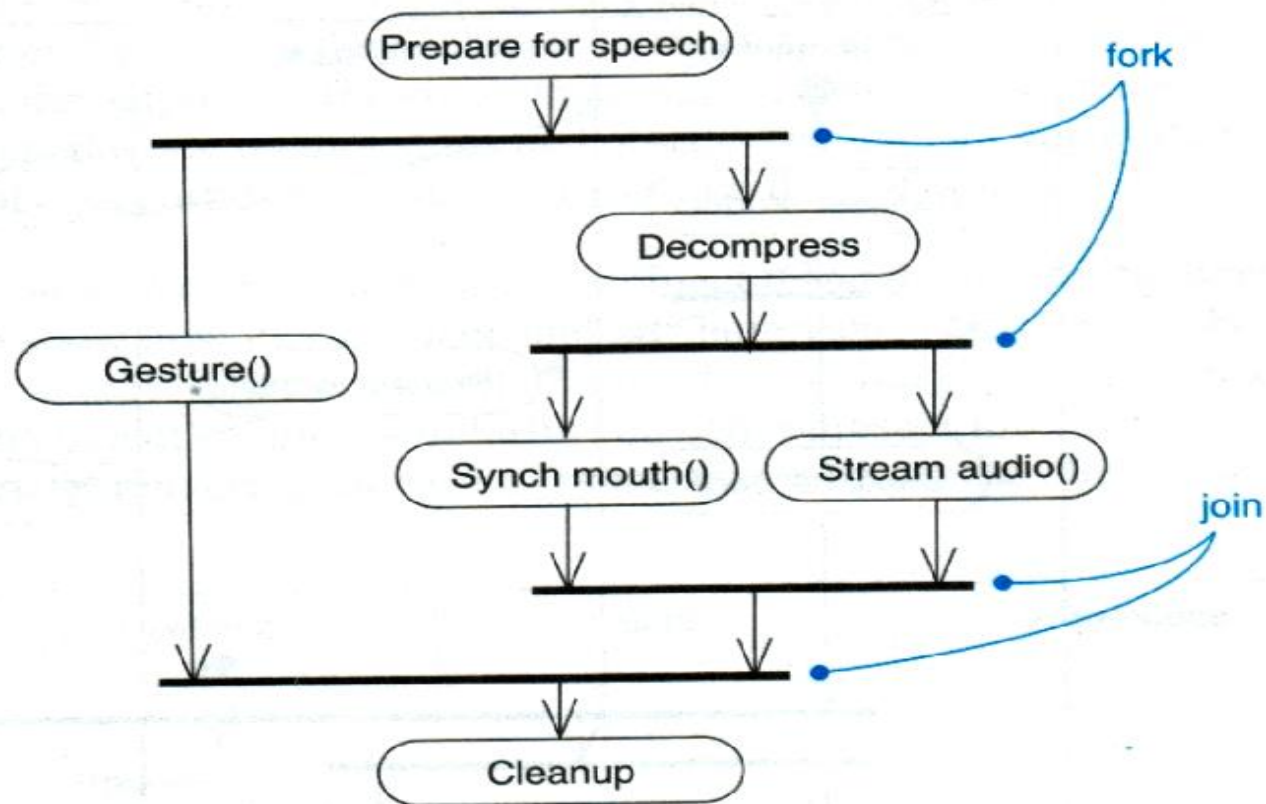
- each transition represents an independent flow of control
- conceptually, the activities of each of outgoing transitions are concurrent

- A join may have two or more incoming transitions and one outgoing transition

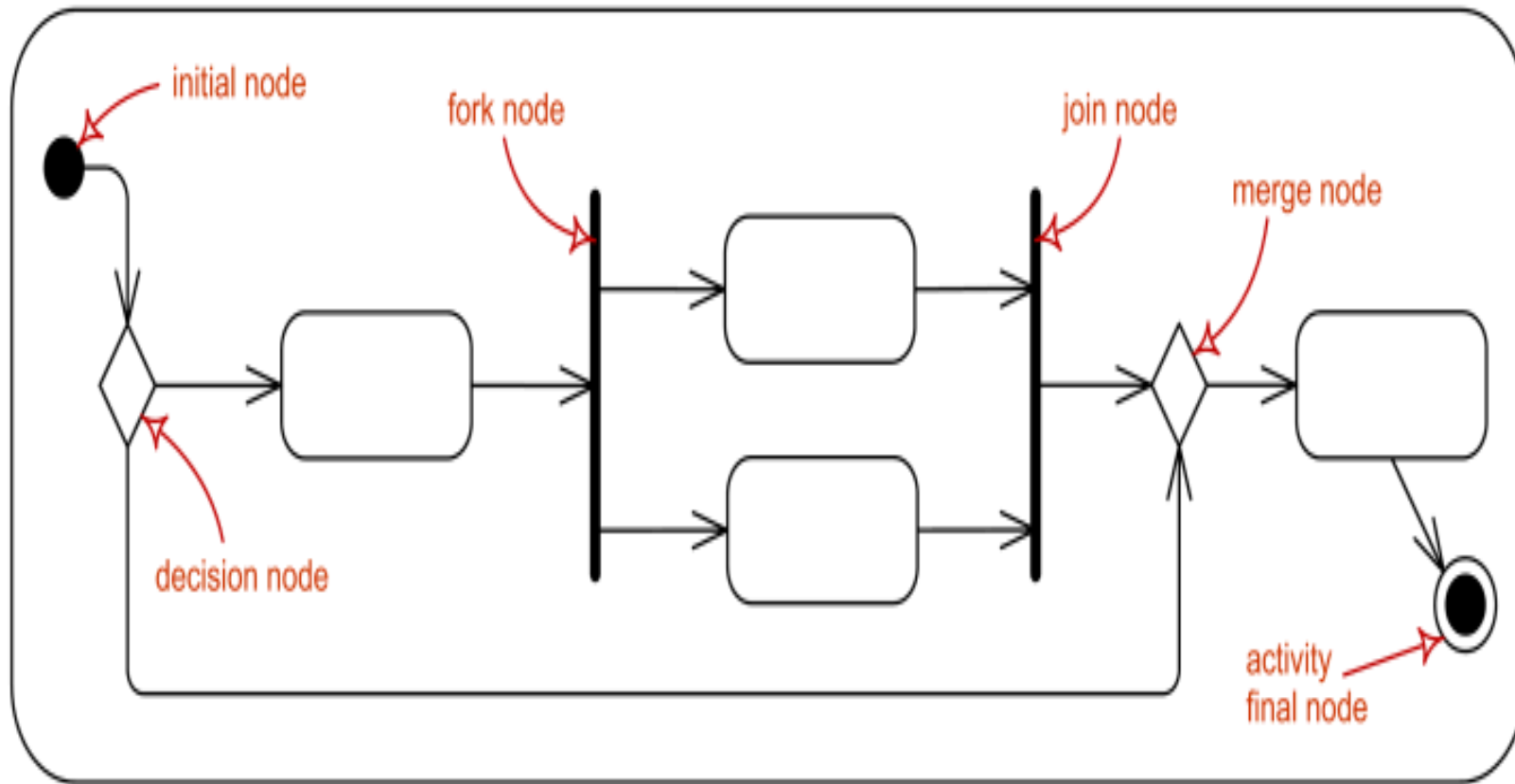


- above the join, the activities associated with each of these paths continues in parallel
- at the join, the concurrent flows synchronize and flow of control continues on below the join

# ACTIVITY DIAGRAM: EXAMPLE (FORK AND JOIN)



# BRANCH & MERGE, FORK & JOIN



# ACTIVITY DIAGRAM: EXAMPLE (FORK AND JOIN)

- We have to develop an activity diagram for order management system.
- When the order is received, order is filled and shipped after filling. *While we are doing this, invoice is sent for receiving the payment.*
- *After shipping the order and receiving the payment, order is closed.*

# SWIMLANES

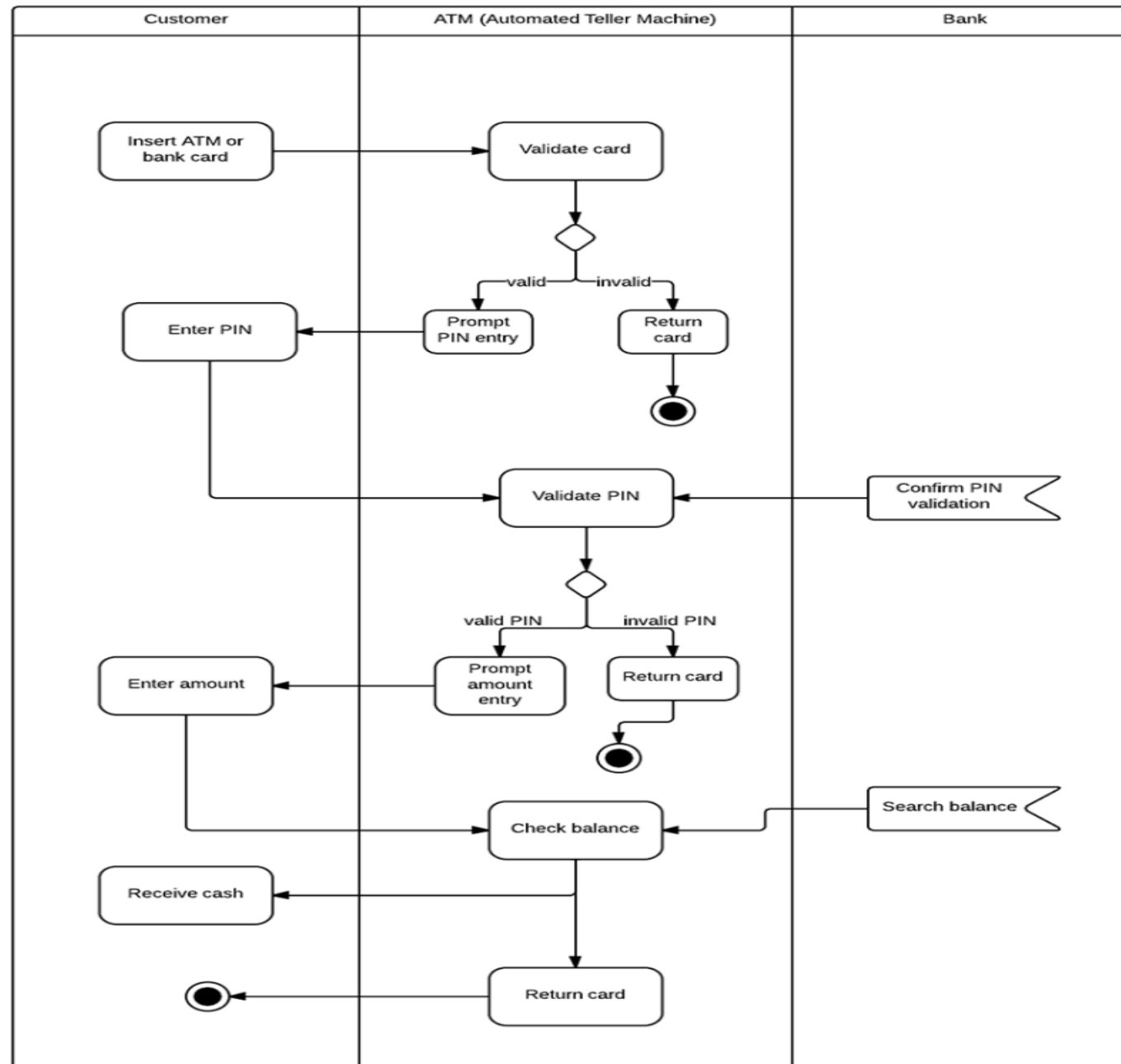
- Arrange activity diagrams into **vertical zones** separated by dashed lines.
- Each zone represents the **responsibilities** of a particular class or department.
- A swimlane specifies a **locus of activities**
- To partition the activity states on an activity diagram into groups
  - each group representing the business organization responsible for those activities
  - each group is called a swimlane
- Swimlane visually distinguishes **job sharing and responsibilities** for sub-processes of a business process. Swim lanes may be arranged either horizontally or vertically.

## SWIMLANES (2)

- Each swimlane has a name unique within its diagram
- Each swimlane may represent some real-world entity
- Each swimlane may be implemented by one or more classes
- Every activity belongs to exactly one swimlane, but transitions may cross lanes

## EXERCISE

- A customer wants to draw money from his bank account. He enters his card into an ATM (automated teller machine). The ATM machine accept the card and prompts for PIN number. The customer enters his PIN. The ATM (internally) retrieves the bank account number from the card. The ATM encrypts the PIN and the account number and sends it over to the bank. The bank verifies the encrypted Account and PIN number. Card is returned incase of invalid pin number. If the PIN number is correct, the ATM displays “Enter amount”, draws money from the bank account and prints the receipt.





# IMPLEMENTATION OF NESTED BRANCHING

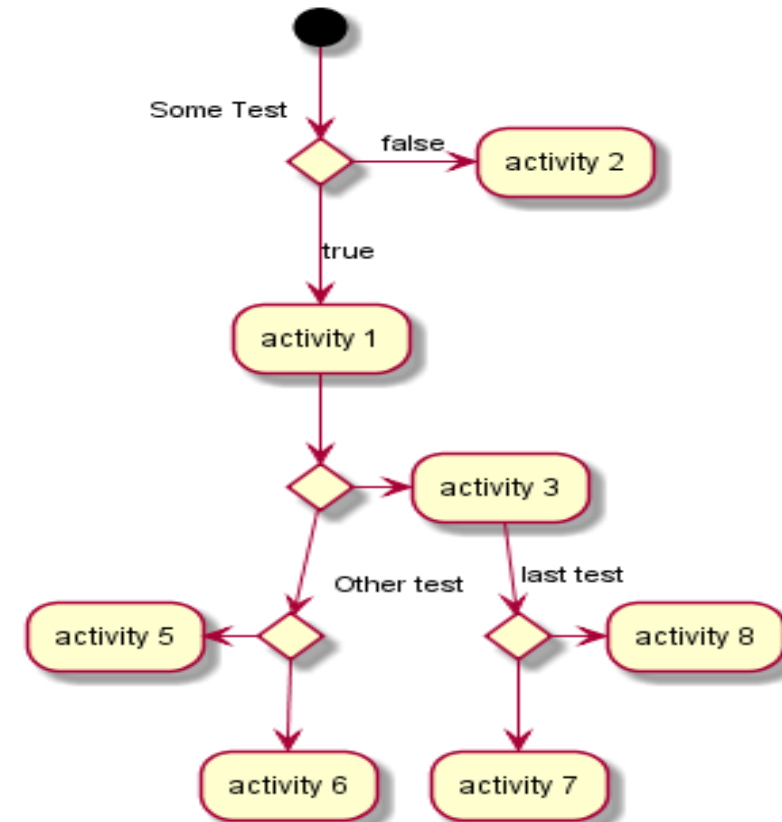
```
(*) --> if "Some Test" then  
-->[true] "activity 1"
```

```
if "" then  
-> "activity 3" as a3  
else  
if "Other test" then  
--> "activity 5"  
else  
--> "activity 6"  
endif  
endif
```

```
else  
->[false] "activity 2"
```

```
endif
```

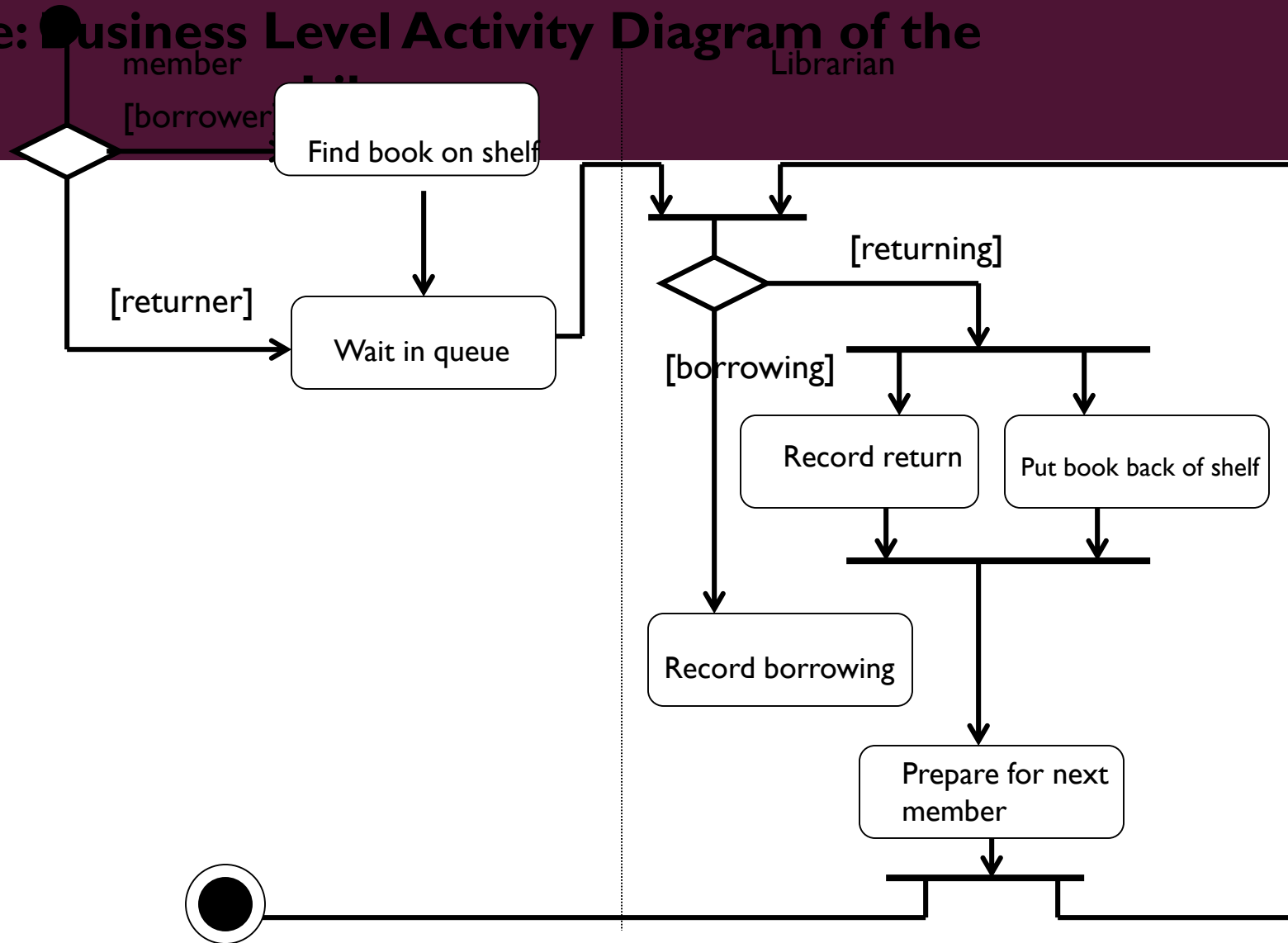
```
a3 --> if "last test" then  
--> "activity 7"  
else  
-> "activity 8"  
endif
```



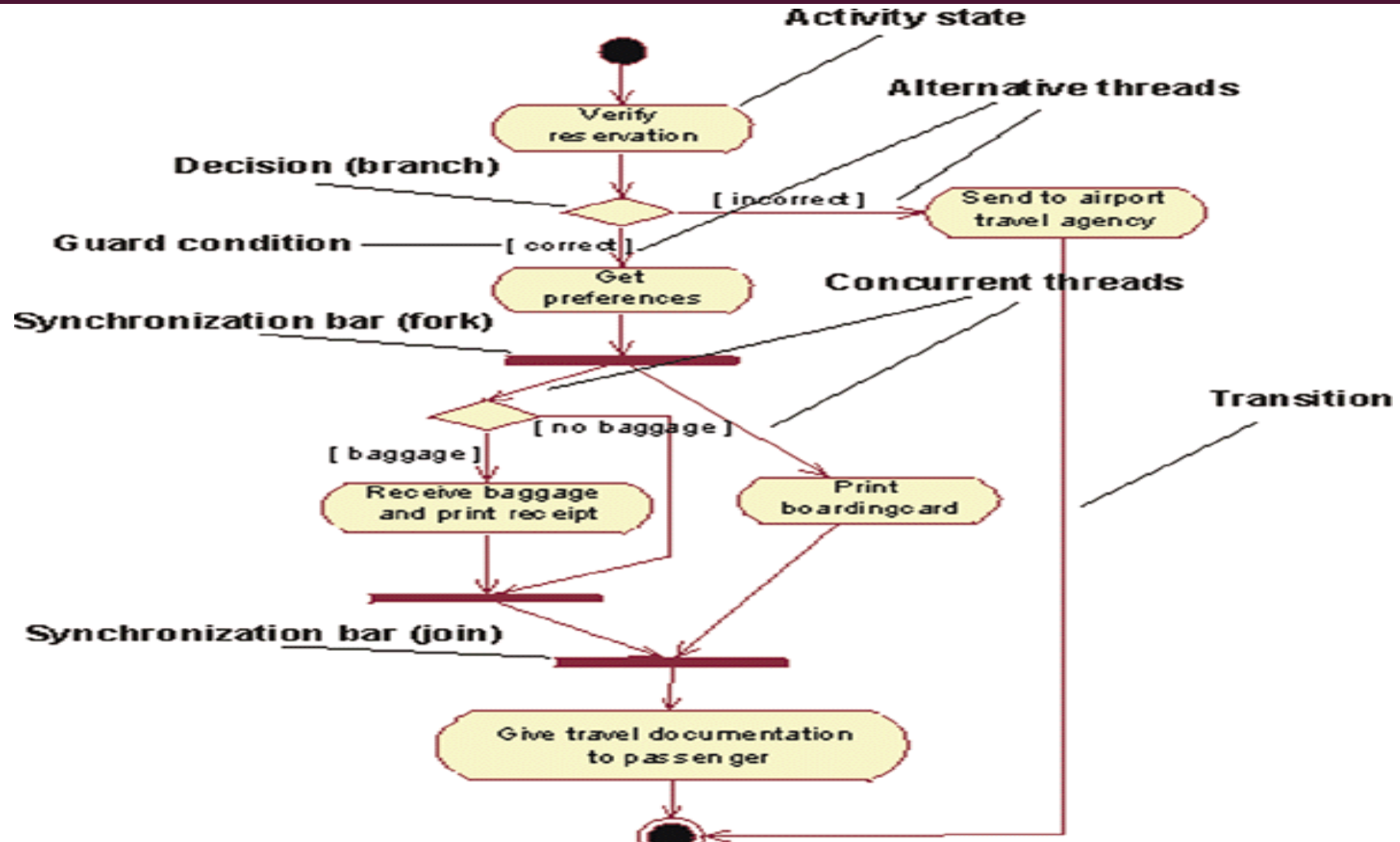
## EXERCISE

- A person visits a beverage shop where he/she can order soft drink or coffee. If he selects the coffee, then the coffee preparation process starts by putting the coffee beans in the filter, adding water to the reservoir and arranging cups. Filter after having the coffee beans in it is placed in the machine. Machine is turned on and coffee brews. When the machine's light turn off, coffee is poured into the cups and served for drinking.
- In case of soft drink selection, customer is asked for the soft drink brand. Customer is served with he drink if the desired drink is available in the shop.

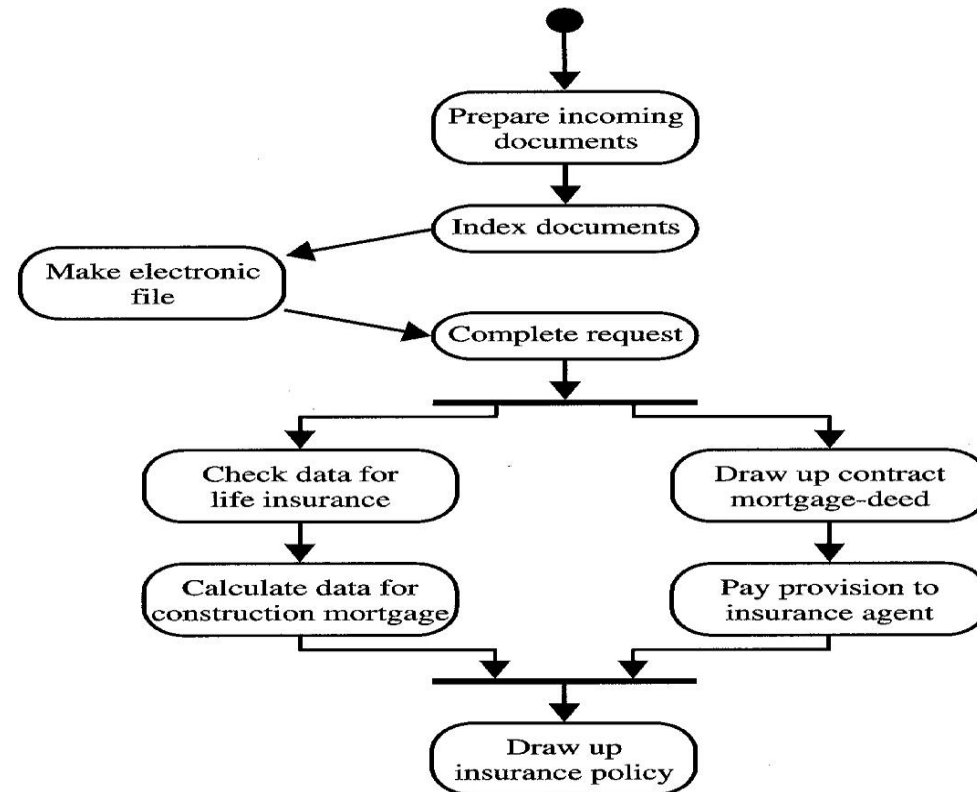
# Example: Business Level Activity Diagram of the



# EXAMPLE: BUSINESS LEVEL ACTIVITY DIAGRAM OF THE CHECK-IN PROCESS



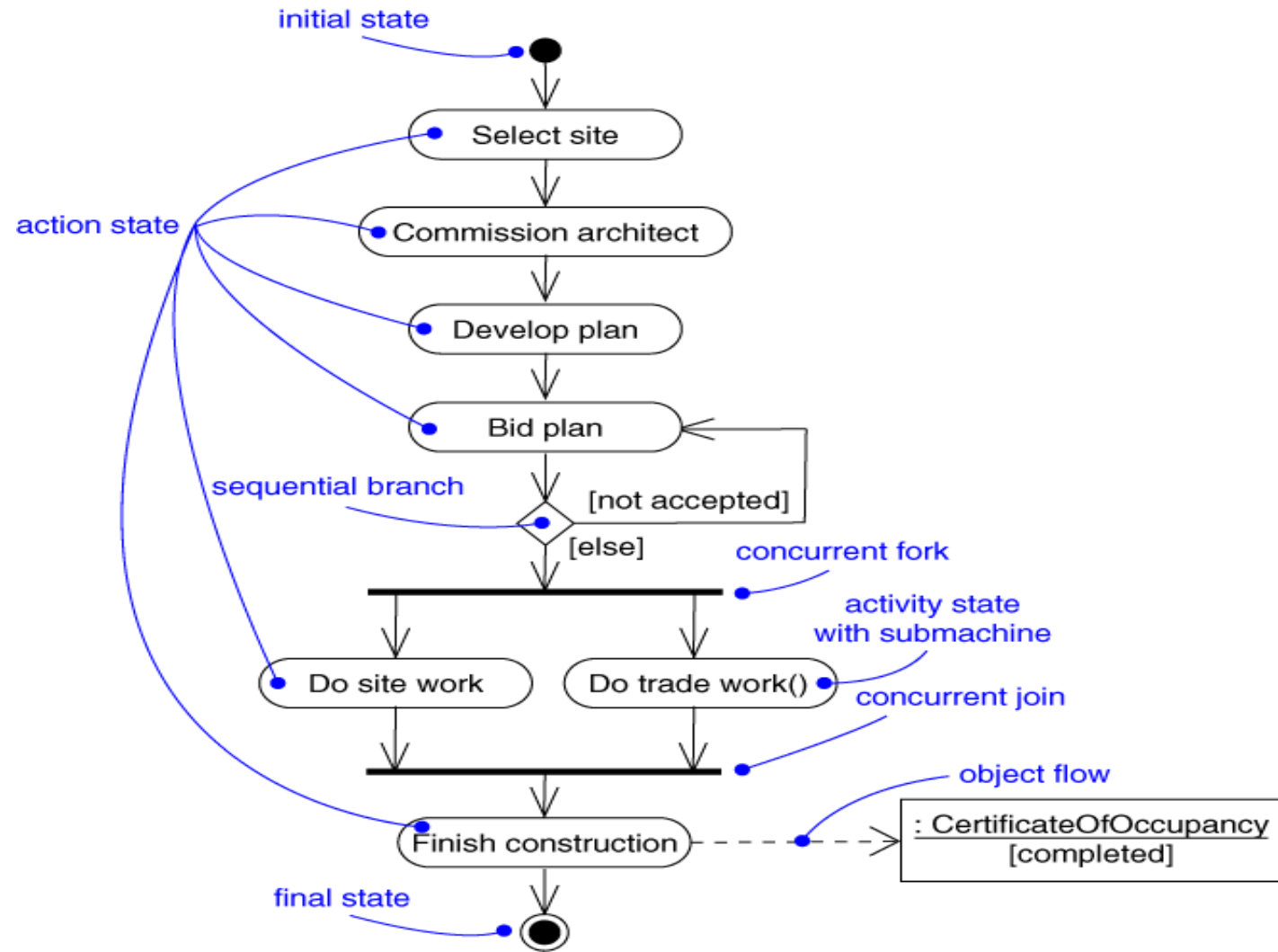
# ACTIVITY DIAGRAM: EXAMPLE (3)



**FIGURE 5-20**

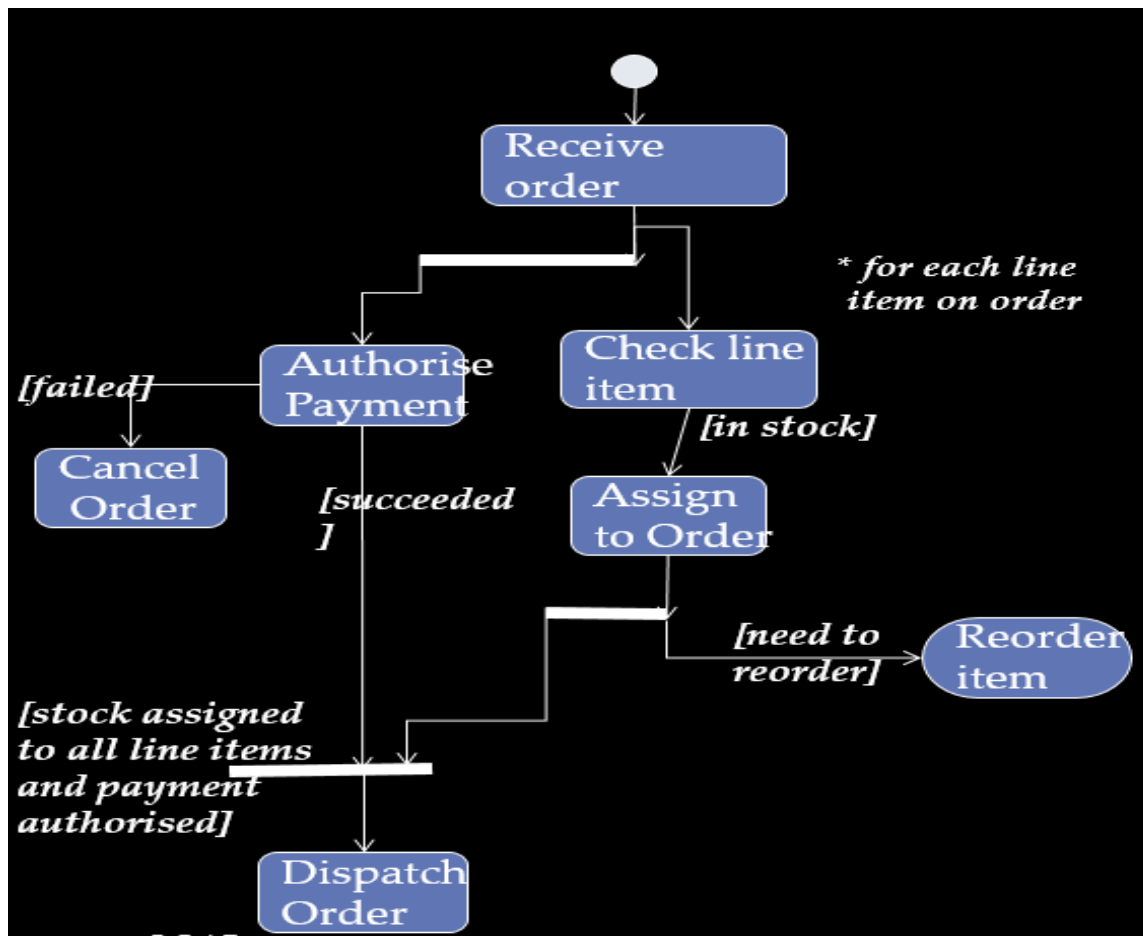
An activity diagram for processing mortgage requests (Loan: Processing Mortgage Request).

# ACTIVITY DIAGRAM: EXAMPLE (4)



# USE CASE FOR ORDER PROCESSING (GENERIC SYSTEM)

- *“When we receive an order, we check each line item on the order to see if we have the goods in stock. If we do, we assign the goods to the order. If this assignment sends the quantity of those goods inn stock below the reorder level, we reorder the goods. While we are doing this, we check to see if the payment is OK. If the payment is OK and we have the goods in stock, we dispatch the order. If the payment is OK but we don’t have the goods, we leave the order waiting. If the payment isn’t OK, we cancel the order.”*

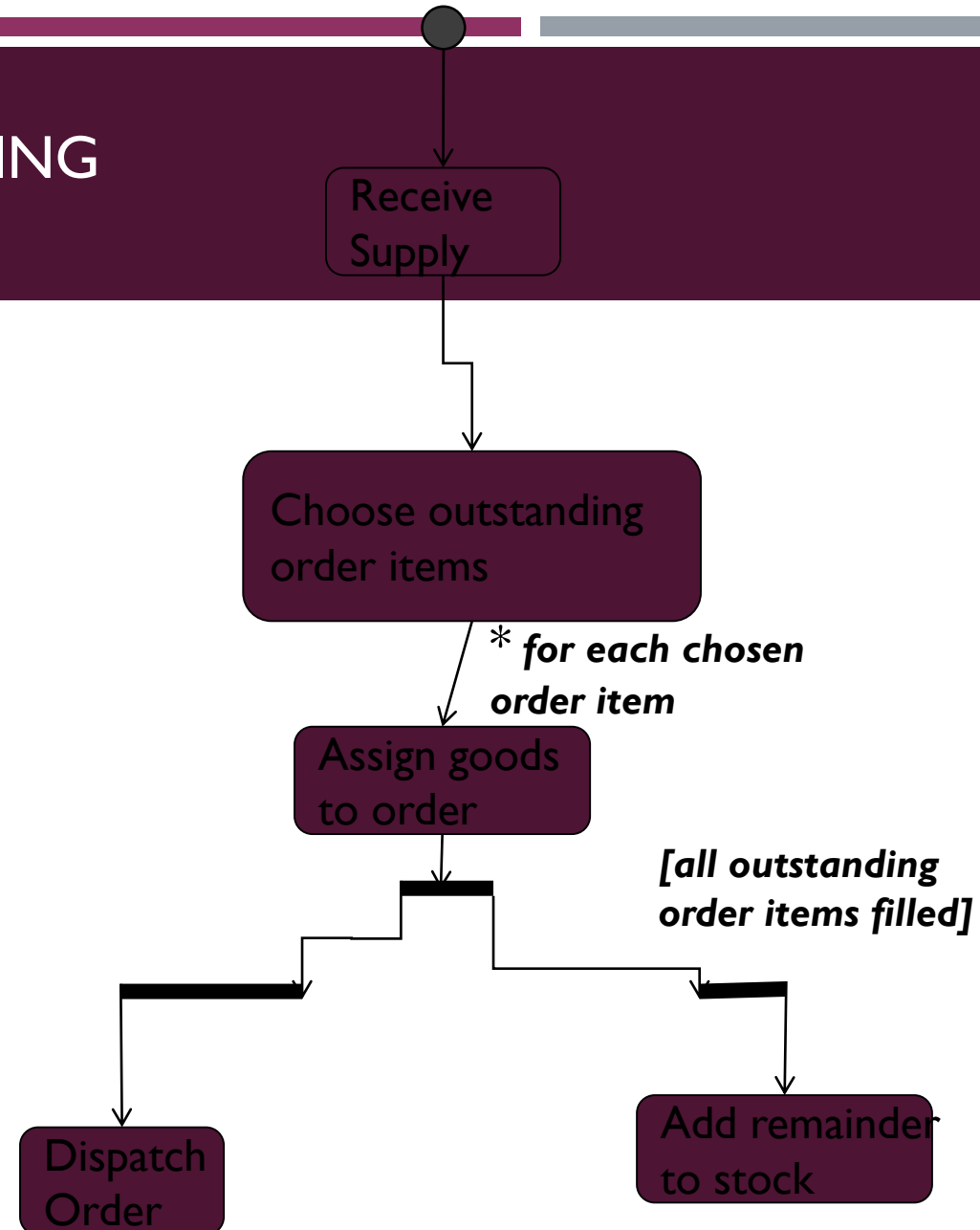




## RECEIVING SUPPLY

- *“When a supply delivery comes in, we look at the outstanding orders and decide which ones we can fill from this incoming supply. We then assign each of these to its appropriate orders. Doing this may release those orders for dispatching. We put the remaining goods into stock.”*

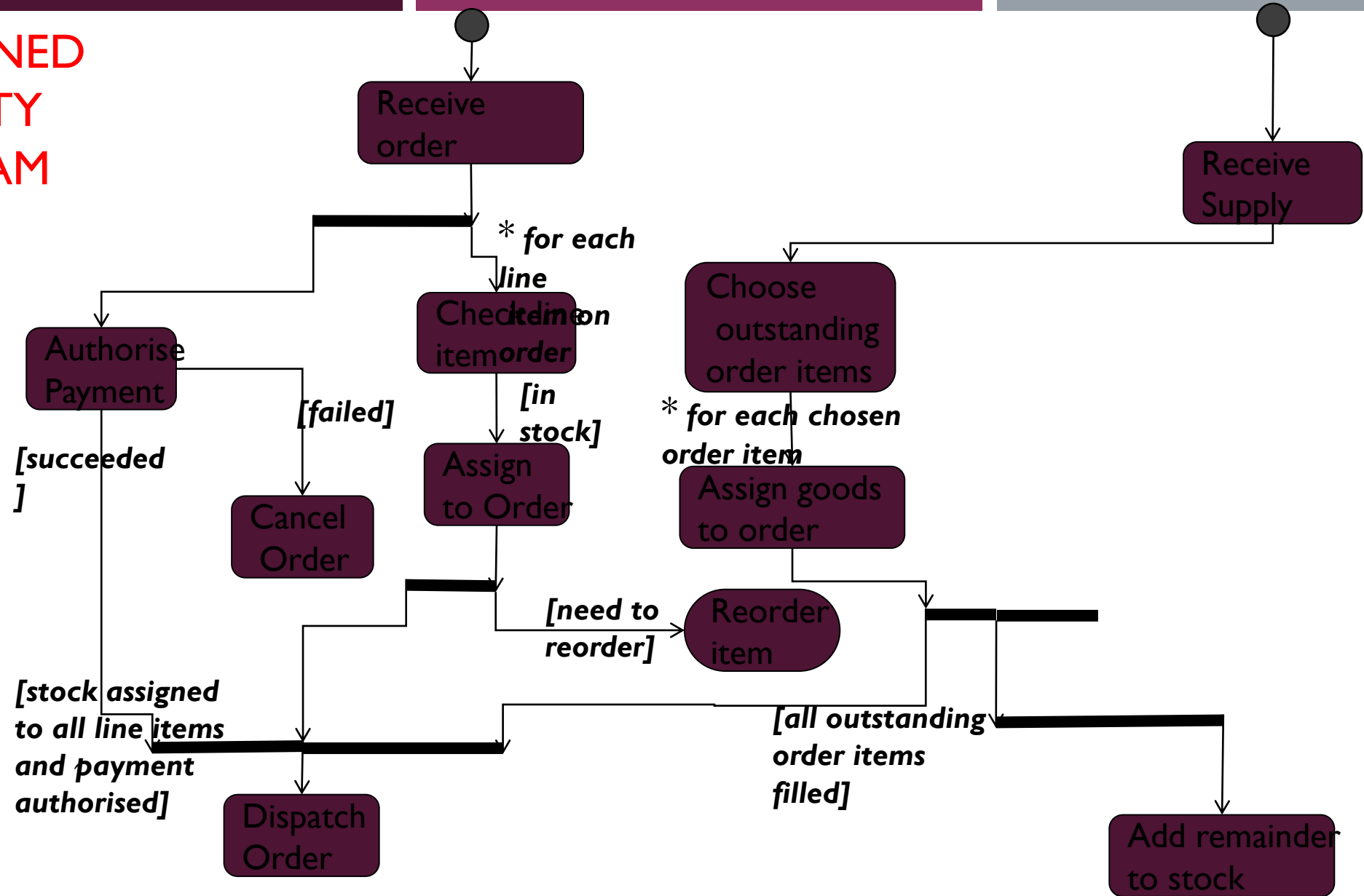
# ACTIVITY DIAGRAM FOR RECEIVING SUPPLY



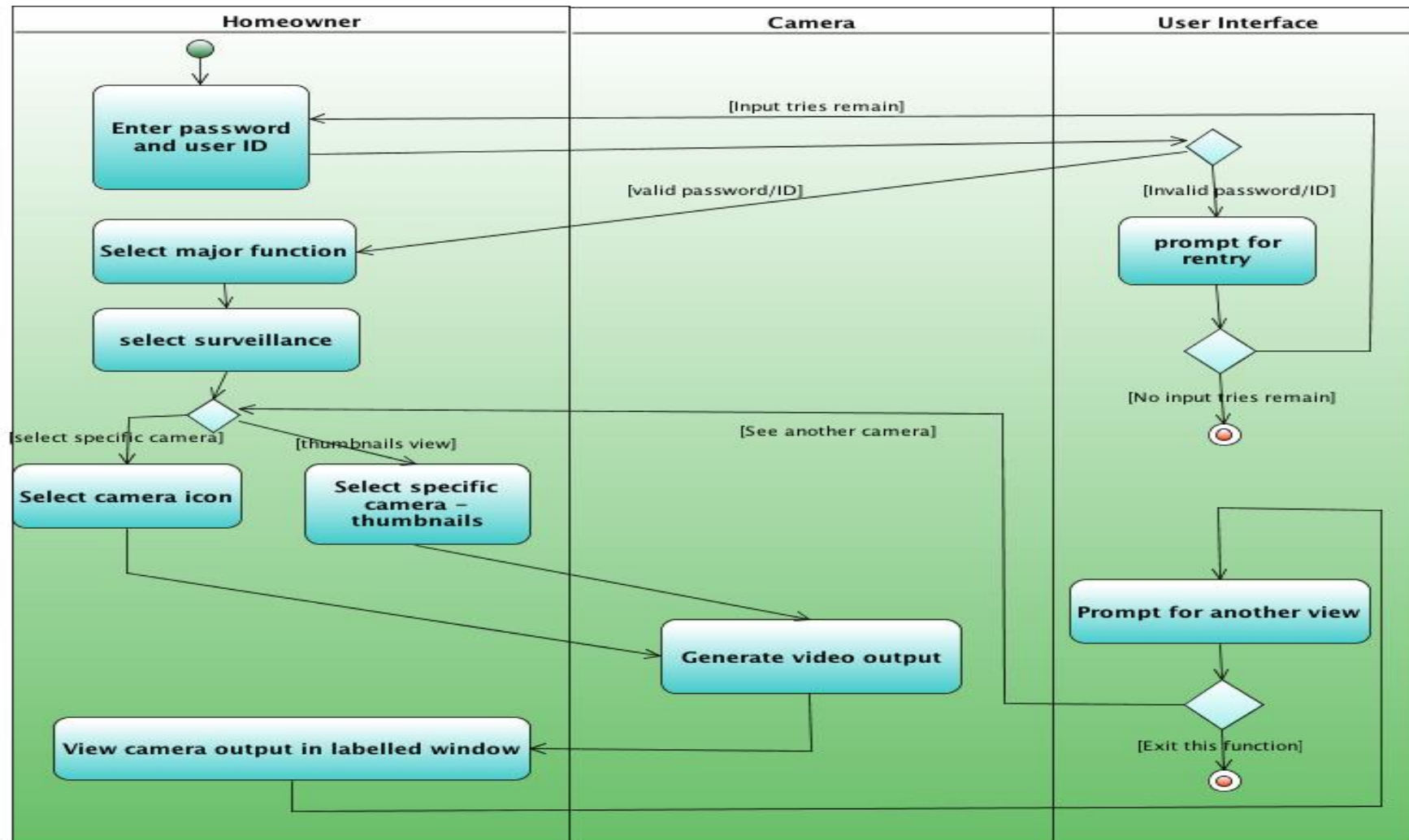
# COMBINING USE CASES

- The end point of an activity diagram is the point at which all triggered activities have been run and there are no more left to do.
- Dead ends (e.g. Reorder item) can occur.
- Sometimes dead ends meet up with other use cases (e.g. Check line item).

# COMBINED ACTIVITY DIAGRAM



# SWIMLANE DIAGRAM



# HOMEWORK TASK

- Create a swimlane diagram to apply for and get a job when you graduate.

# OBJECT AND OBJECT FLOW

ObjectNode

## ■ Object Node:

- An object node is an activity node that indicates an instance of a particular classifier. Object nodes can be used in a variety of ways, depending on where objects are flowing from and to.
- Objects in the UML language hold information - state - that is passed between actions. Objects may represent class instances:



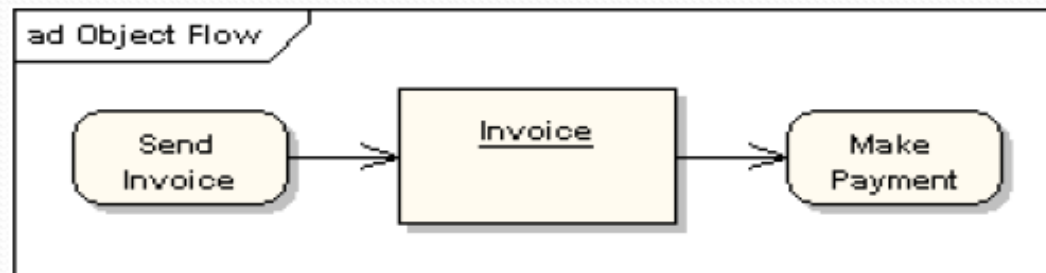
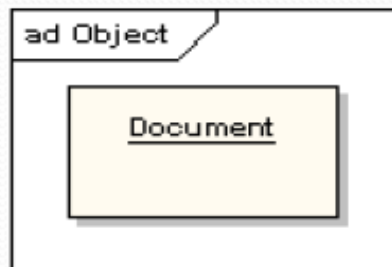
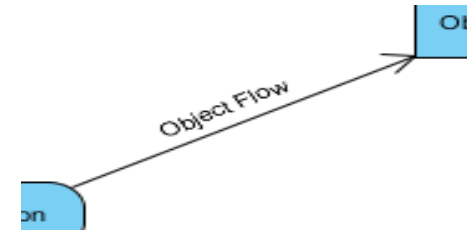
- Objects usually change state between actions:



# OBJECT AND OBJECT FLOW

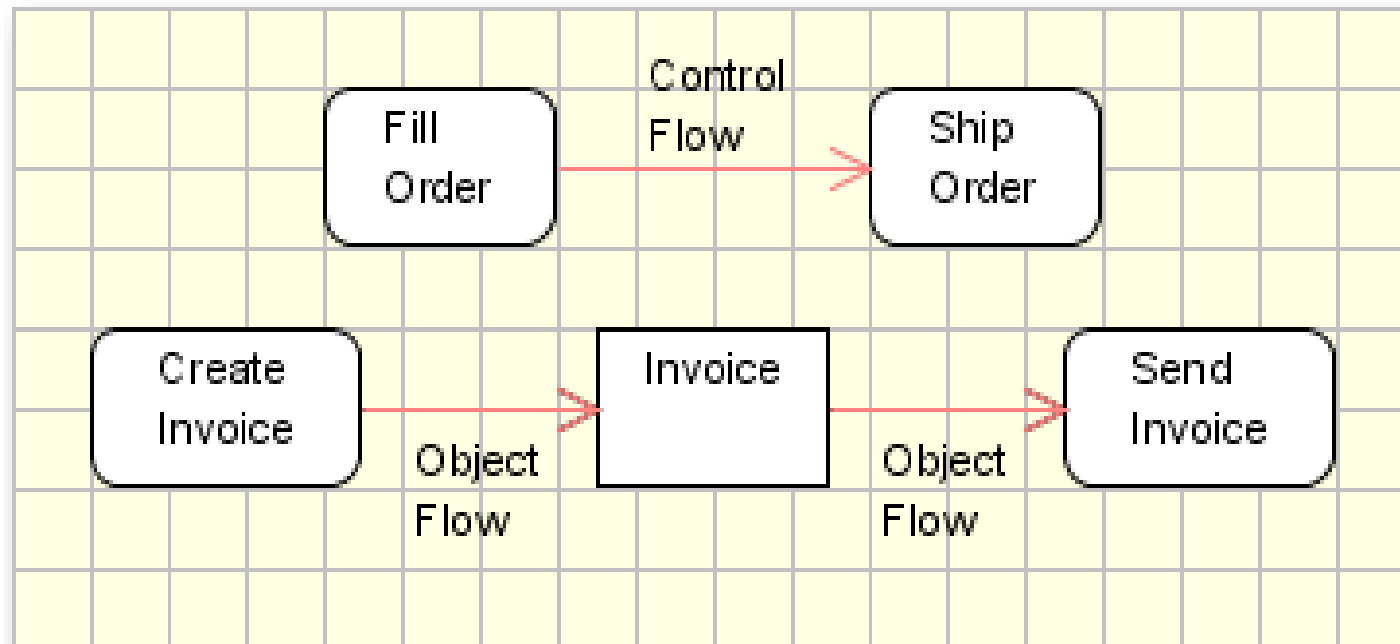
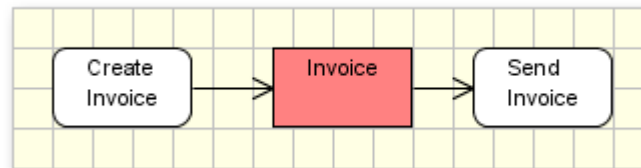
## ■ Object Flow:

- There is no data flow in activity diagram. Object flow plays role of data flow as well.
- An object flow is an activity edge that can have objects or data passing along it.
- An object flow is shown as a connector with an arrowhead denoting the direction the object is being passed.





# OBJECT FLOW AND CONTROL FLOW

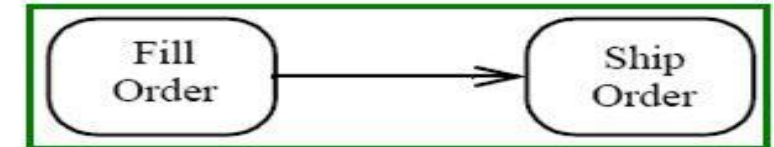
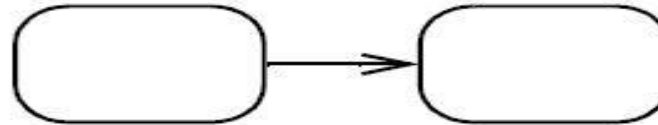




## Activity edges (2)

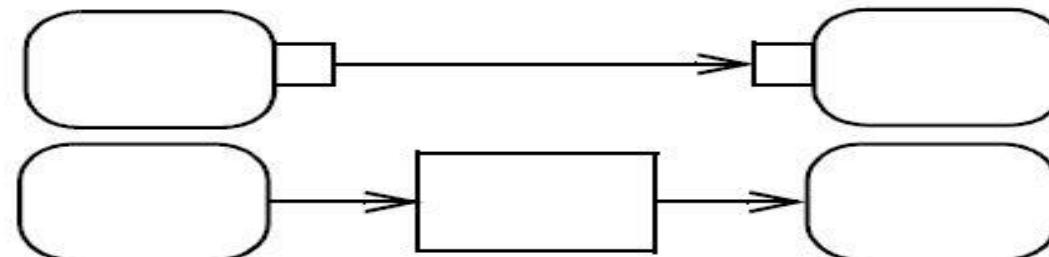
- Two kinds of edges:
  - Control flow edge** - is an edge which starts an activity node after the completion of the previous one by passing a control token

→  
**Notation without  
activity nodes**

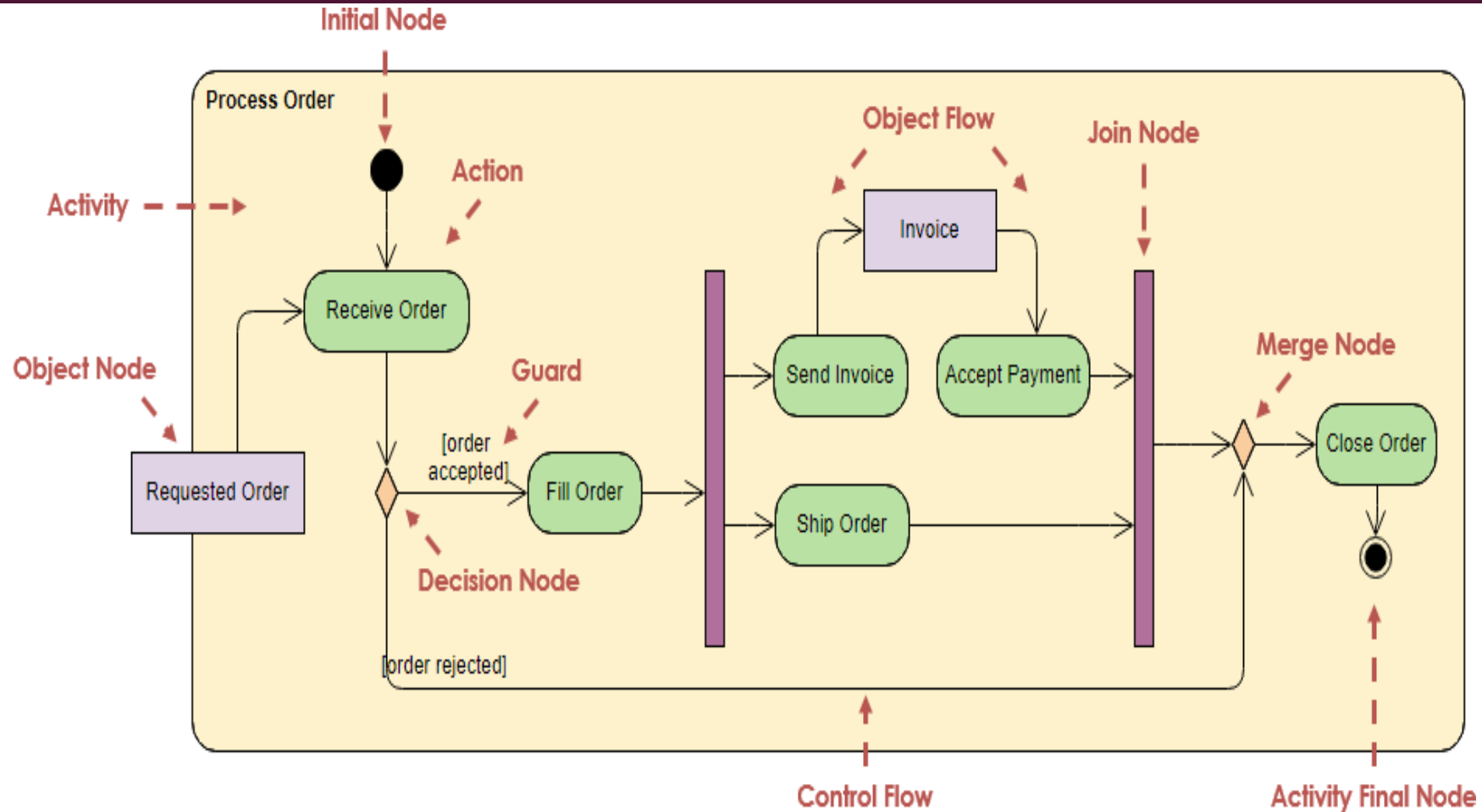


- Object flow edge** - models the flow of values to or from object nodes by passing object or data tokens

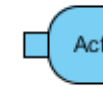
→  
**Notation without  
activity nodes**



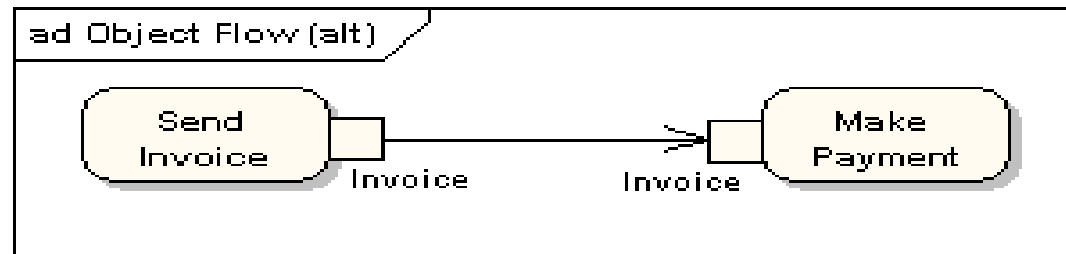
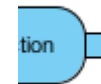
# EXAMPLE OBJECT FLOW AND CONTROL FLOW



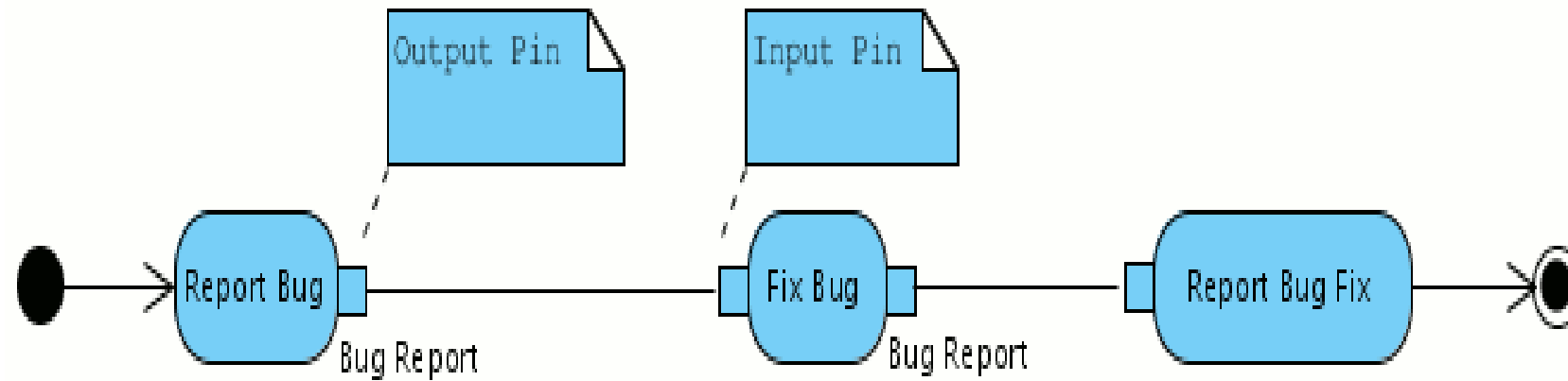
# INPUT AND OUTPUT PINS



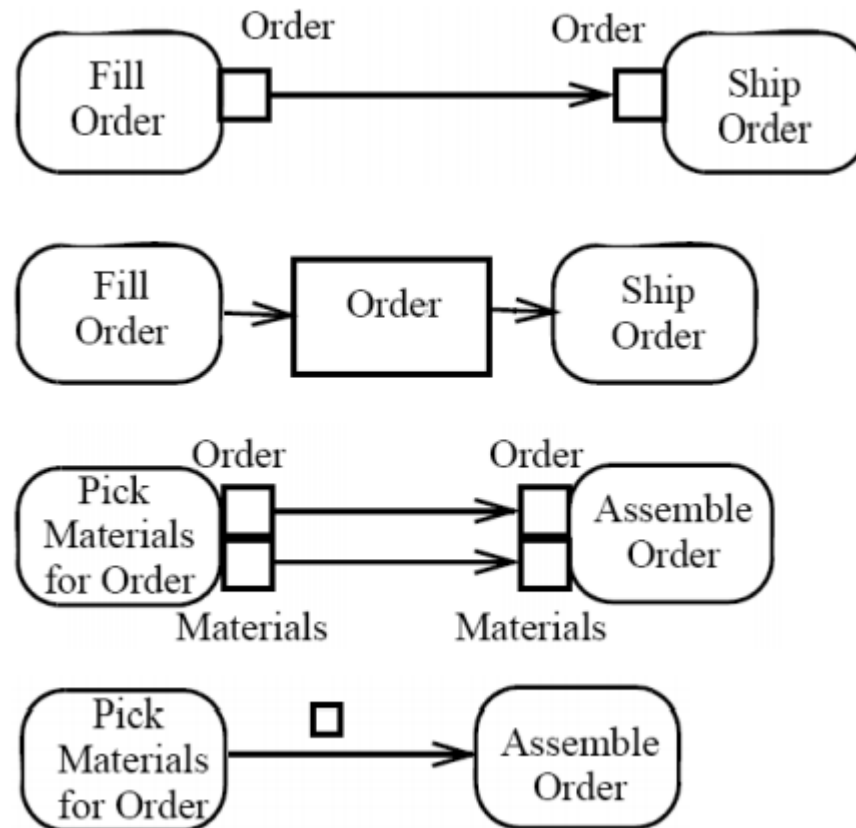
- An object flow must have an object on at least one of its ends. A shorthand notation for the above diagram would be to use input and output pins
- Input pins are object nodes that receive values from other actions through object flows
- Output pins are object nodes that deliver values to other actions through object flows.



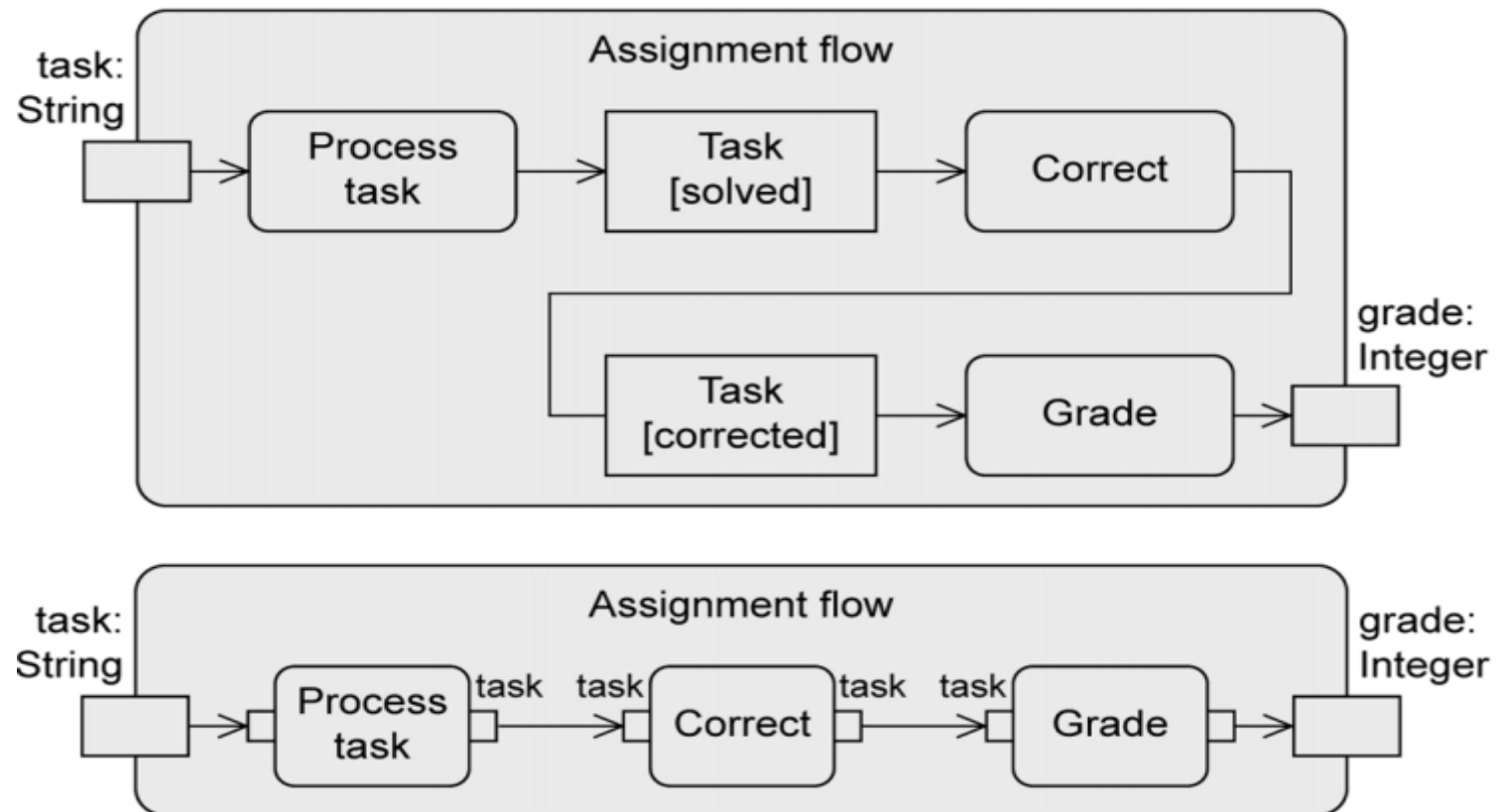
# INPUT AND OUTPUT PINS



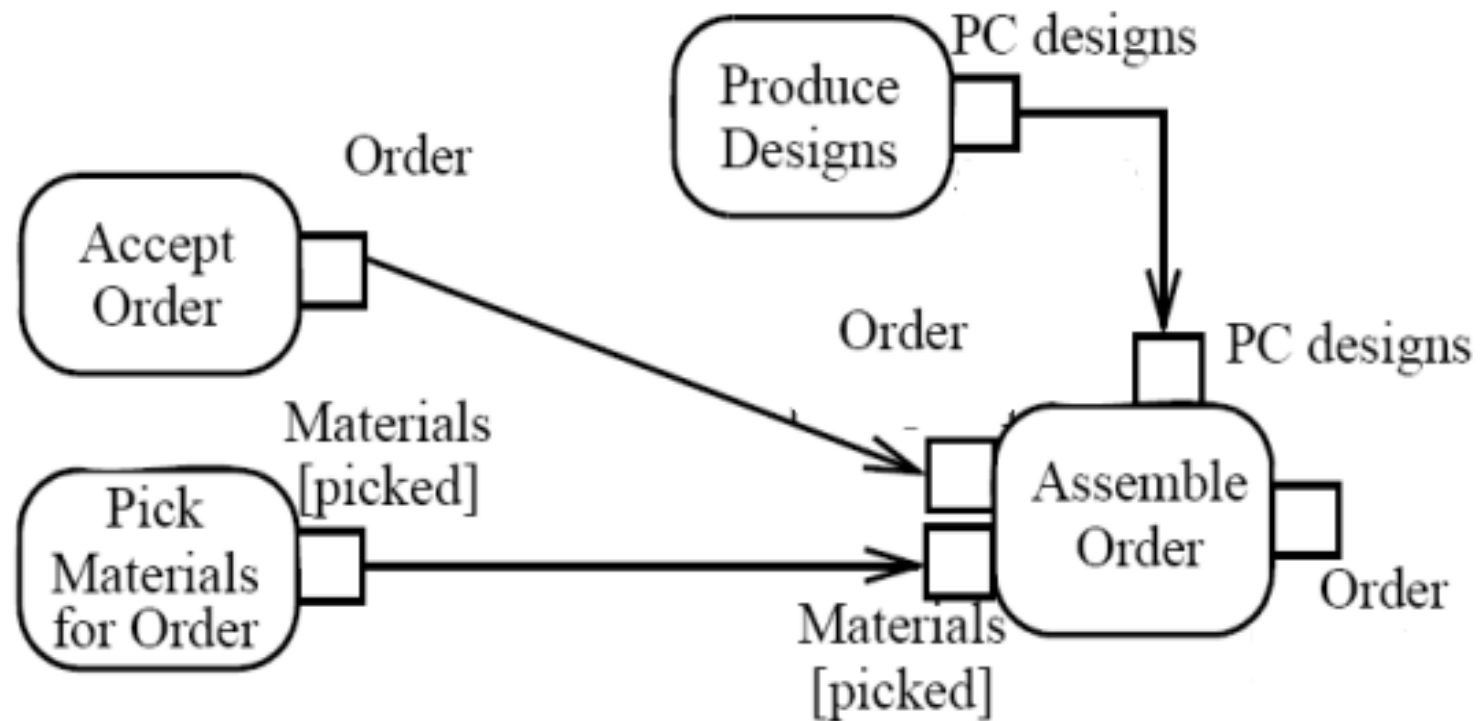
# INPUT AND OUTPUT PINS-REPRESENTATIONS



# EXAMPLE: ASSIGNMENT FLOW



# INPUT AND OUTPUT PINS-EXAMPLE

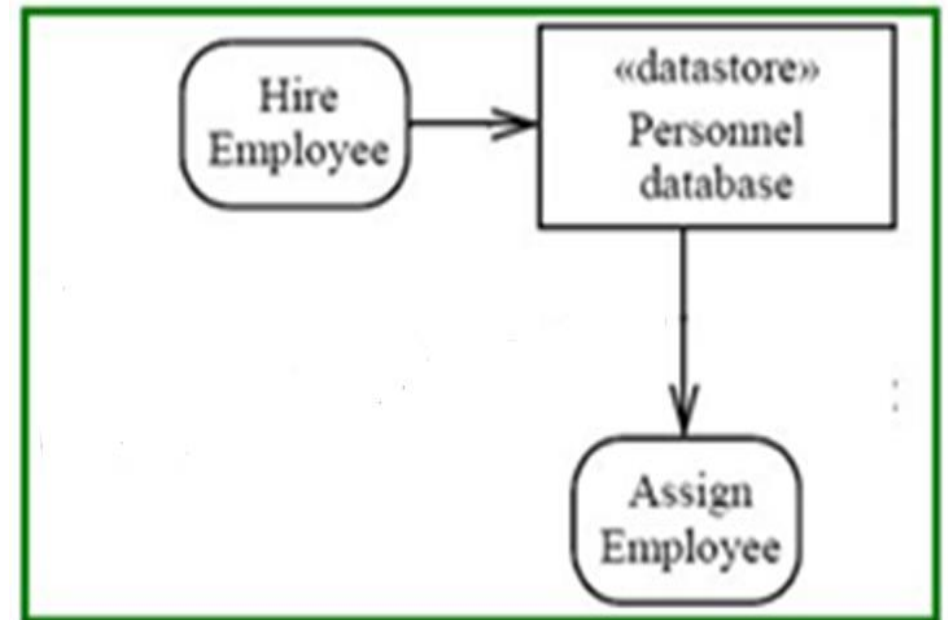




# DATA STORE NODE

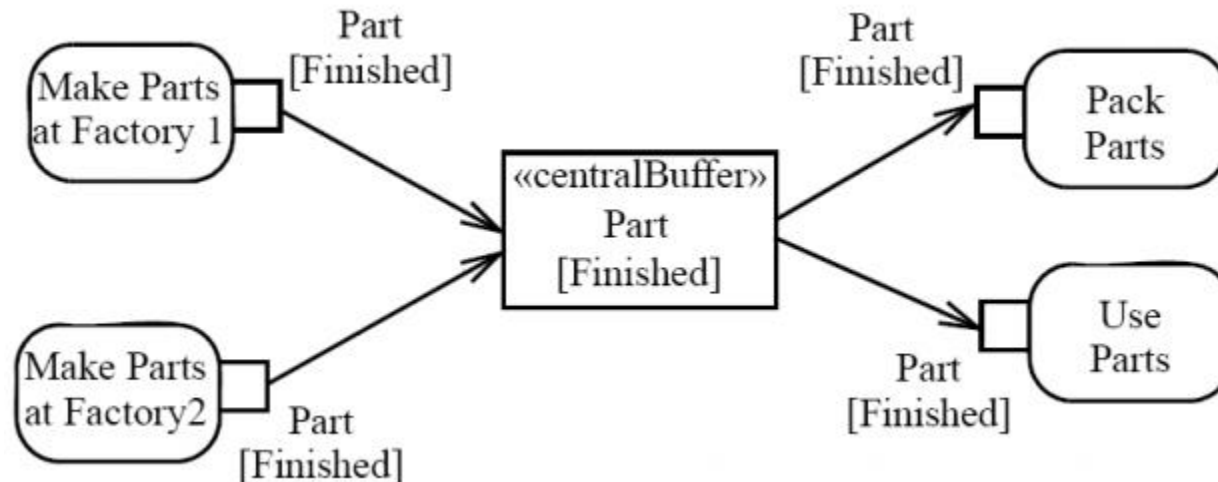
<<datastore>>  
DataStoreNode

- A data store is shown as an object with the «datastore» keyword.



# OBJECT NODE: CENTRAL BUFFER

- For saving and passing on object tokens
- Transient memory
- Accepts incoming object tokens from object nodes and passes them on to other object nodes.
- When an object token is read from the central buffer, it is deleted from the central buffer and cannot be consumed again



# STRUCTURED ACTIVITIES

- *Structured Activity* elements are used in Activity diagrams. A Structured Activity is an activity node that can have subordinate nodes as an independent *Activity Group*.
- Structured activities provide mechanisms to show the strategy for managing the synchronization issues in a real-time system.
- The activities that take place inside a basic structured activity are shown with the keyword <<structured>>.
- When tokens flow into the structured activity, that activity takes only one token at a time, waiting for the activity to complete in the protected region before the next token enters. A structured activity can have multiple exception handlers.

# REGION

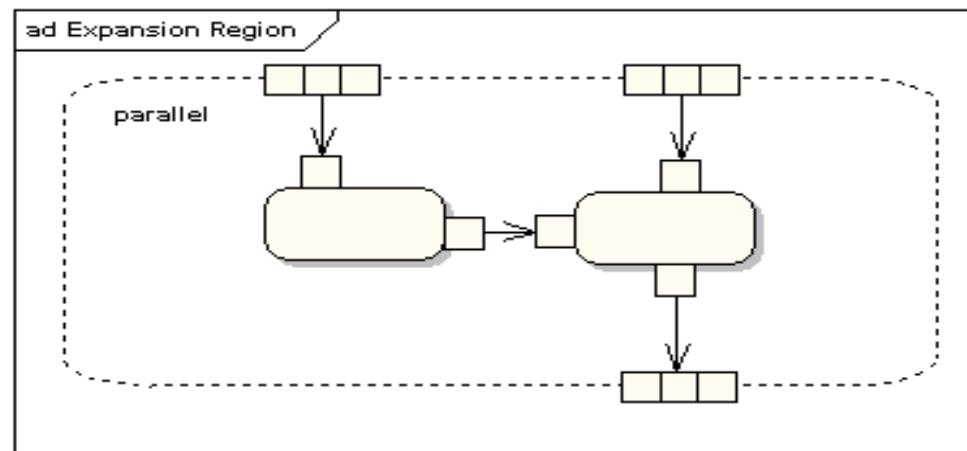
- Expansion Regions
- Interruptible Activity Regions.

# EXPANSION REGIONS

- An Expansion Region surrounds a process to be imposed multiple times on the incoming data, once for every element in the input collection.
- An expansion region is a structured activity region that executes multiple times for collection items.
- Input and output expansion nodes are drawn as a group of three / four boxes representing a multiple selection of items.

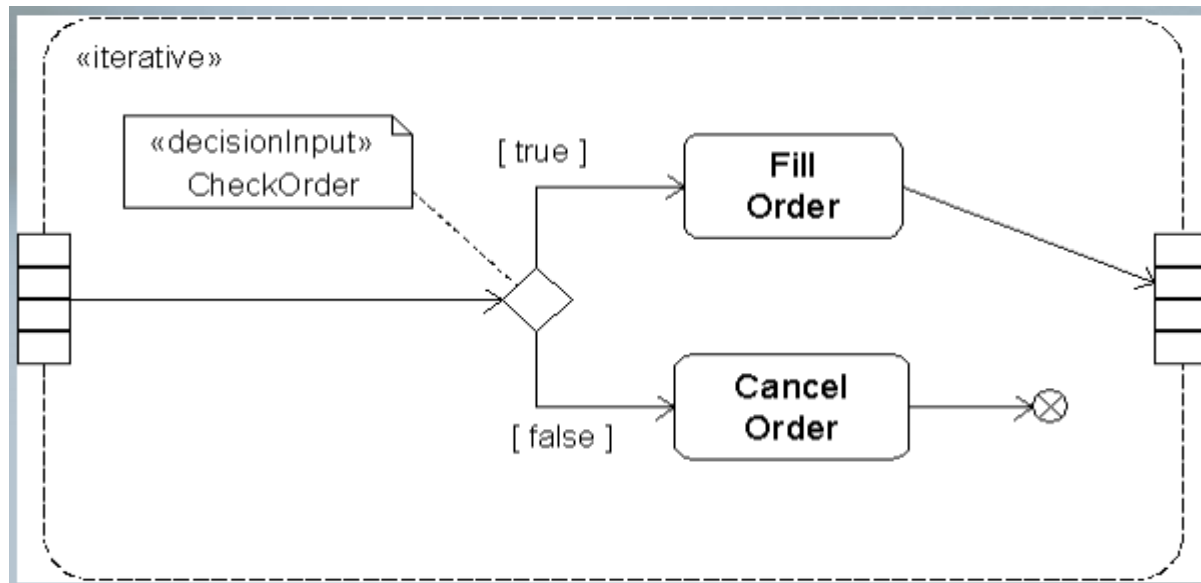
# MODES OF MULTIPLE EXECUTIONS

- Expansion Region's multiple executions can be specified as type parallel, iterative, or stream.
- The keyword iterative, parallel, or stream is shown in the top left corner of the region
- **Parallel** reflects that the elements in the incoming collections can be processed at the same time or overlapping.

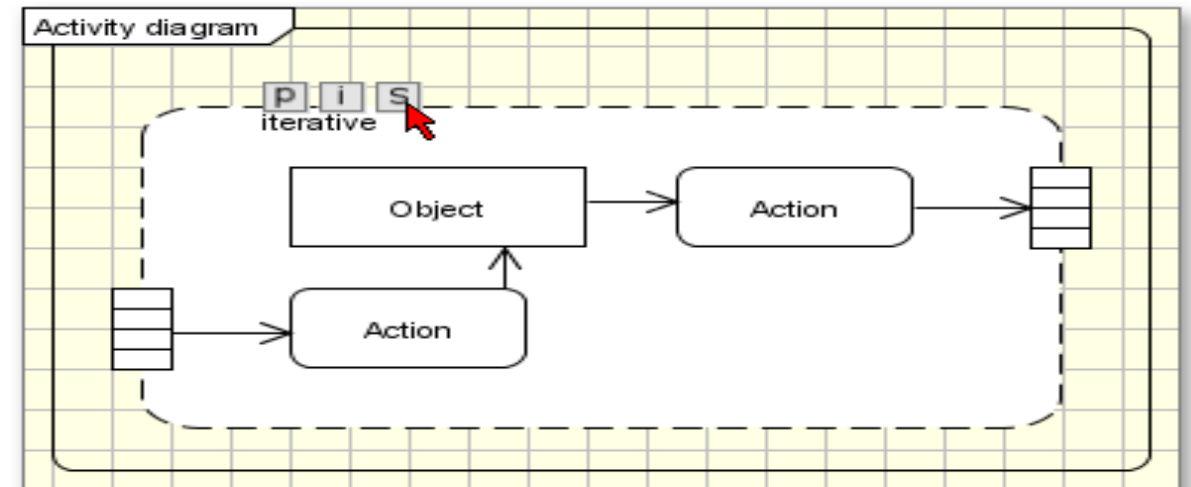


# MODES OF MULTIPLE EXECUTIONS

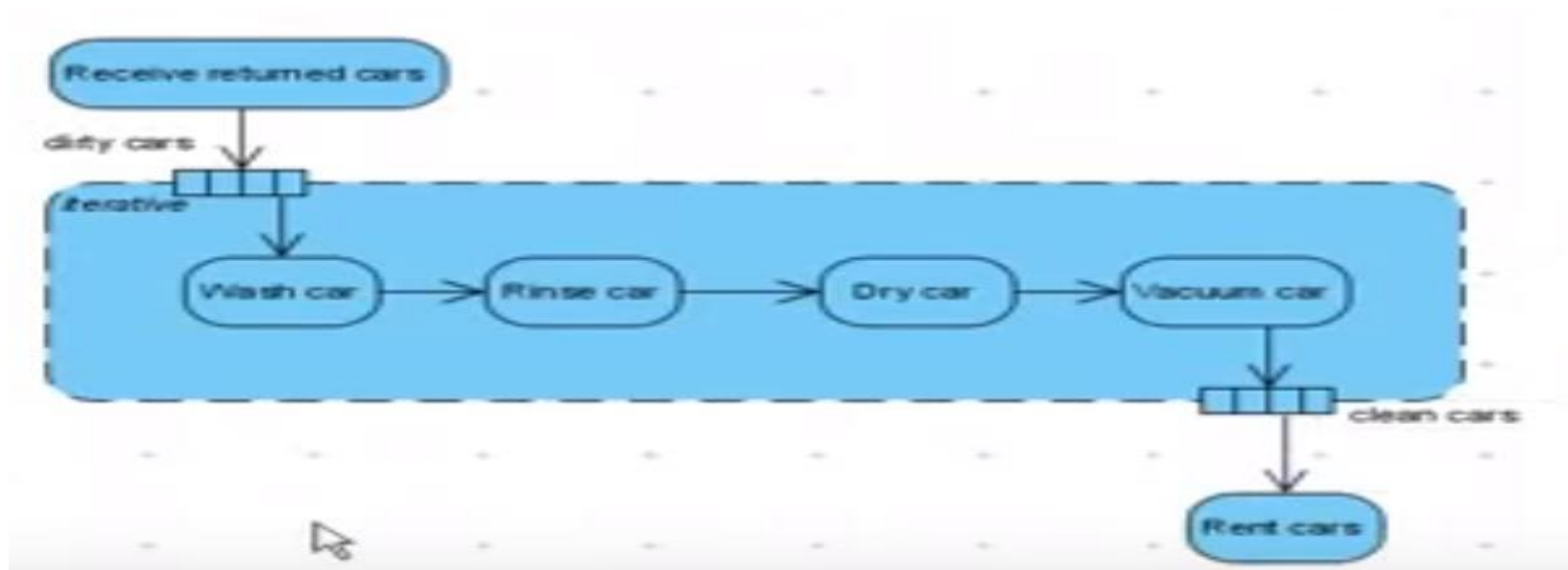
- An **iterative** concurrency type specifies that execution must occur sequentially.



SDA

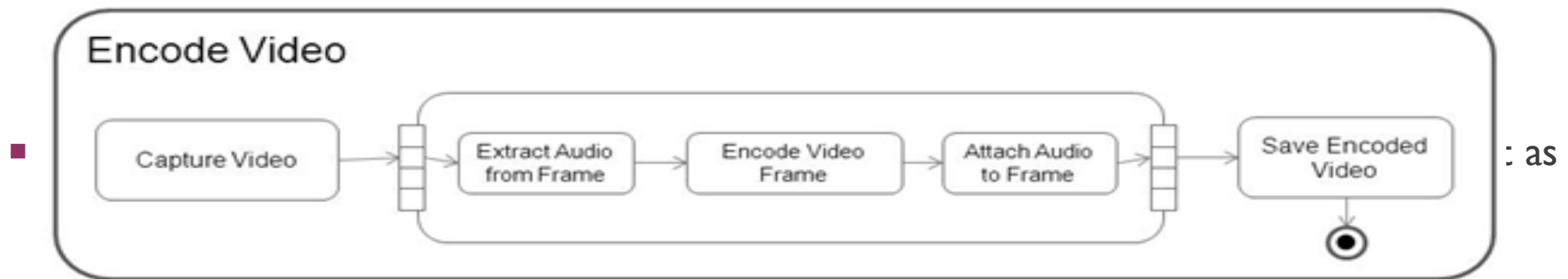


# ITERATIVE EXAMPLES





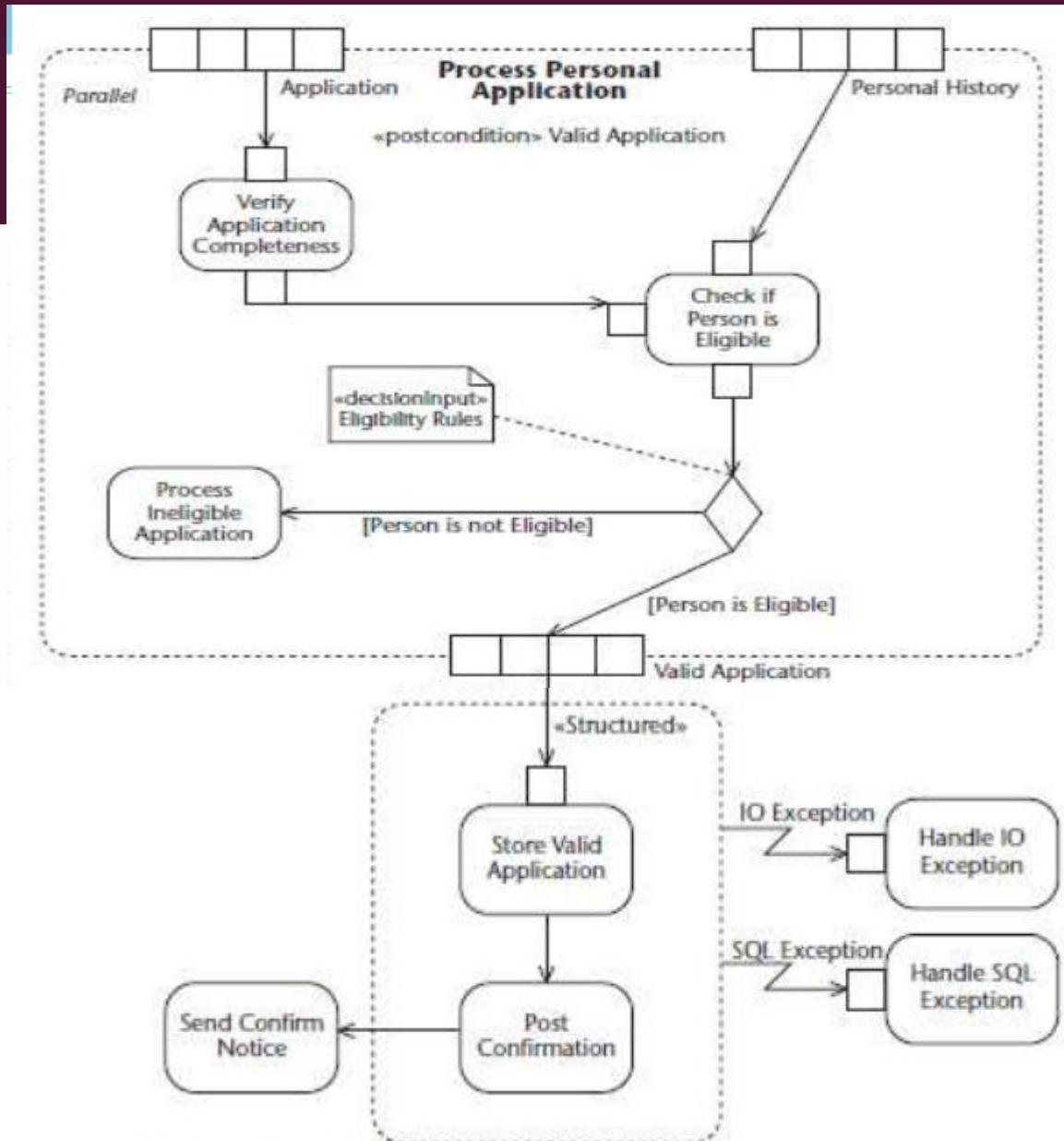
# MODES OF MULTIPLE EXECUTIONS



The activity region in this diagram is labeled with the keyword "stream," meaning there is only one instance of the activity region and it processes input data as it's available. As soon as data moves from the first action, it can begin processing the next available data.

# EXPANSION REGION:APPLICATION-PROCESSING

- The organization must accept an electronic application, review the eligibility of the application, store the application, and then confirm receipt of a valid application. The top section on the diagram shows an expansion region, a type of structured activity that handles the input and output of collections with multiple executions.
- The collection as two sets of four boxes on the top and one at the bottom showing the application, the personal information, and the verified applications.
- The italicized word in the upper-left corner shows that this expansion region allows multiple executions to the actions to occur in parallel, or concurrently.
- The action executions do not have to follow any ordering on the entering collections. You can also use iterative to show the region only allows one set of actions to execute at a time or streaming to show execution in the order of the collection.
- The system also relies on information from a database about the person.
- When combined with the eligibility rules for the application, shown on the diagram as a <>, the organization filters out ineligible applications and sends the application on for storage.



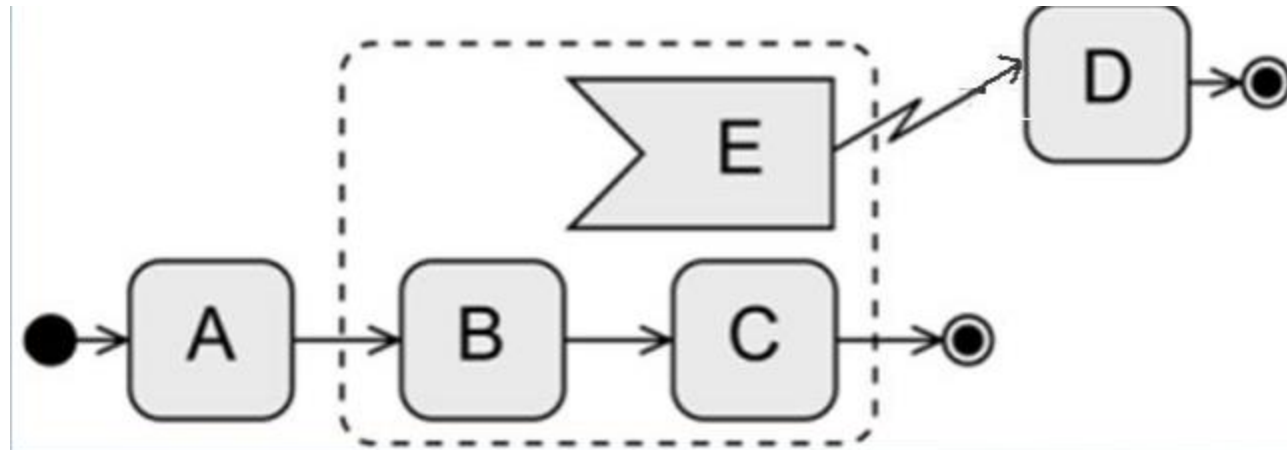
1.8 Managing applications for monetary grants.

# INTERRUPTIBLE ACTIVITY REGIONS

- An Interruptible Activity Region surrounds a group of Activity elements, all affected by certain interrupts in such a way that all tokens passing within the region are terminated when the interruption(s) be raised.
- An interruptible activity region surrounds a group of actions that can be interrupted.

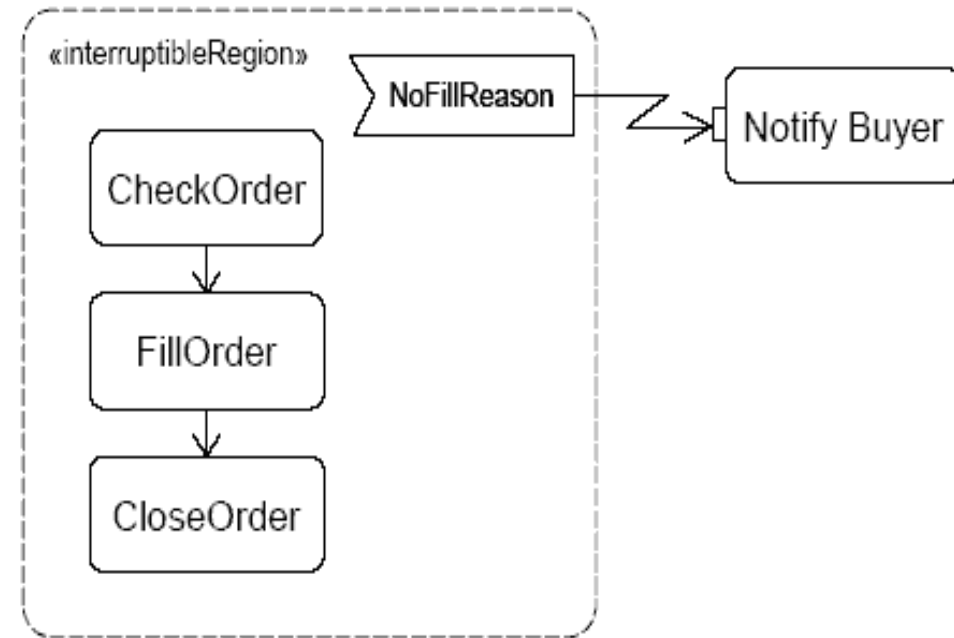
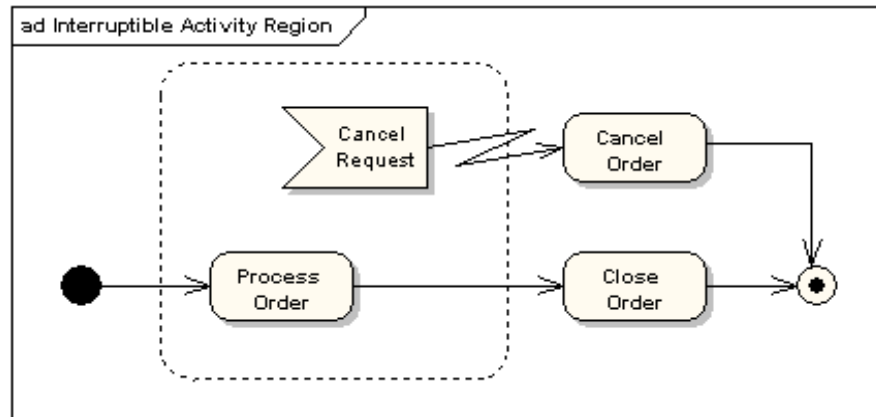
# INTERRUPTIBLE ACTIVITY REGIONS

- if E occurs while B or C are executed, Exception handling is activated
- All control tokens within the dashed rectangle (= within B and C) are deleted
- D is activated and executed No “jumping back” to the regular execution!

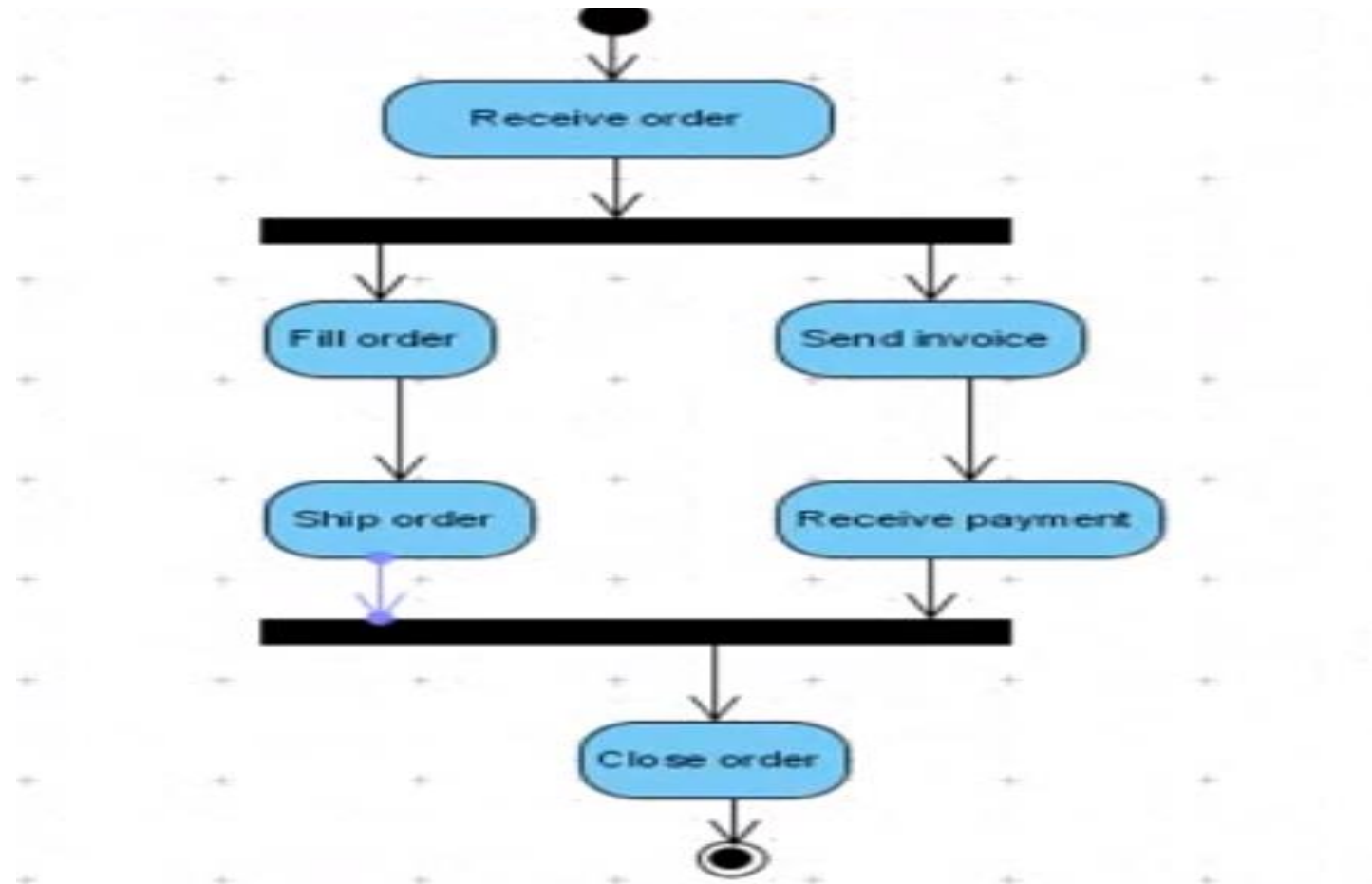


# INTERRUPTIBLE ACTIVITY REGIONS

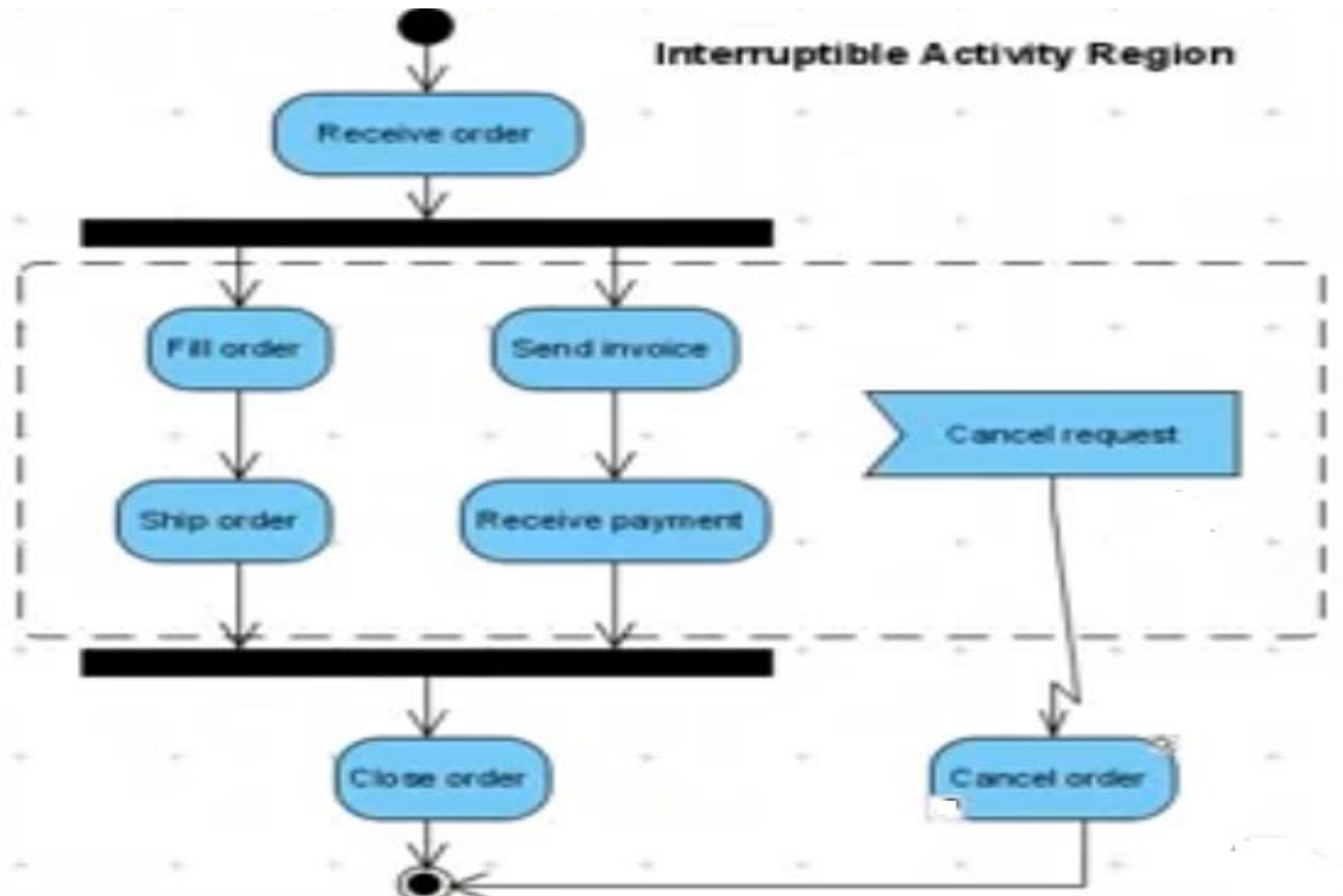
## EXAMPLES



# ACTIVITY DIAGRAM: EXAMPLE

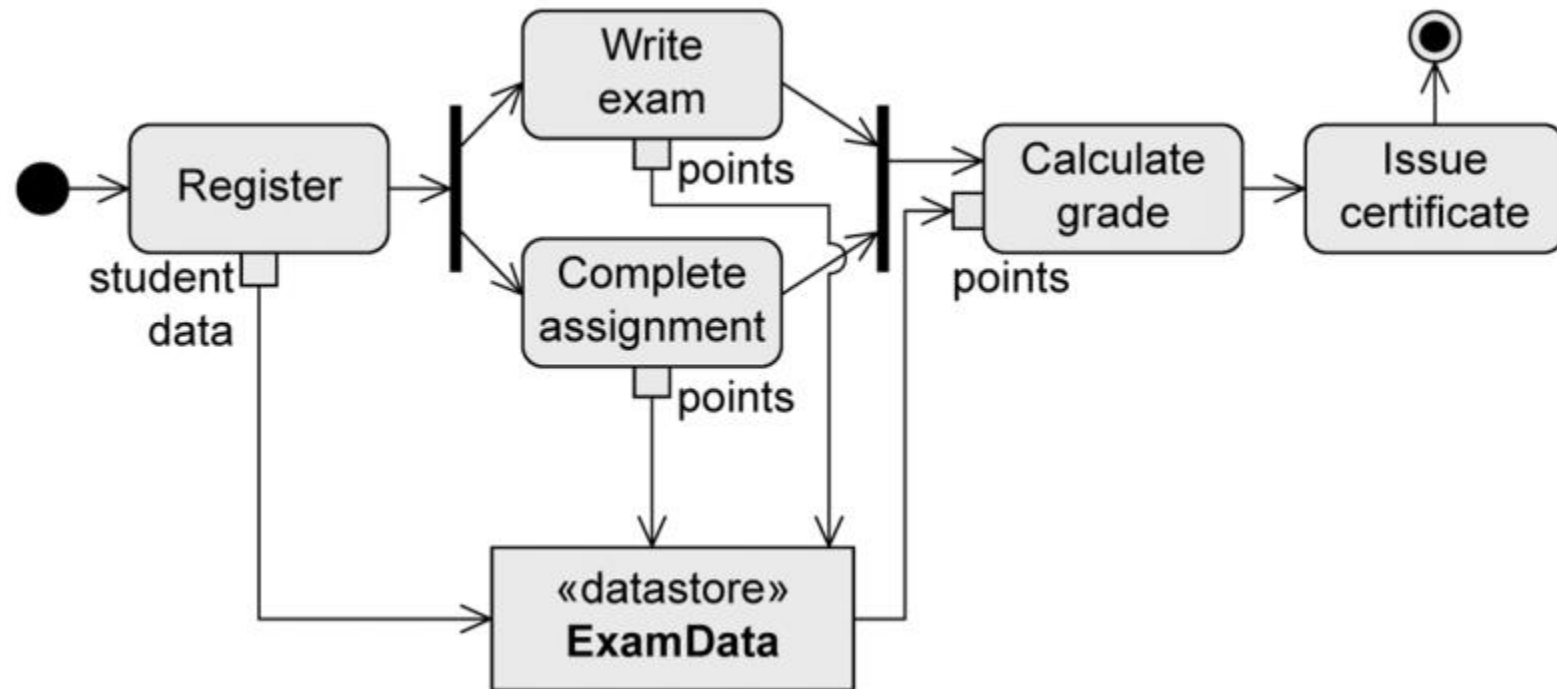


# INTERRUPTIBLE ACTIVITY REGION

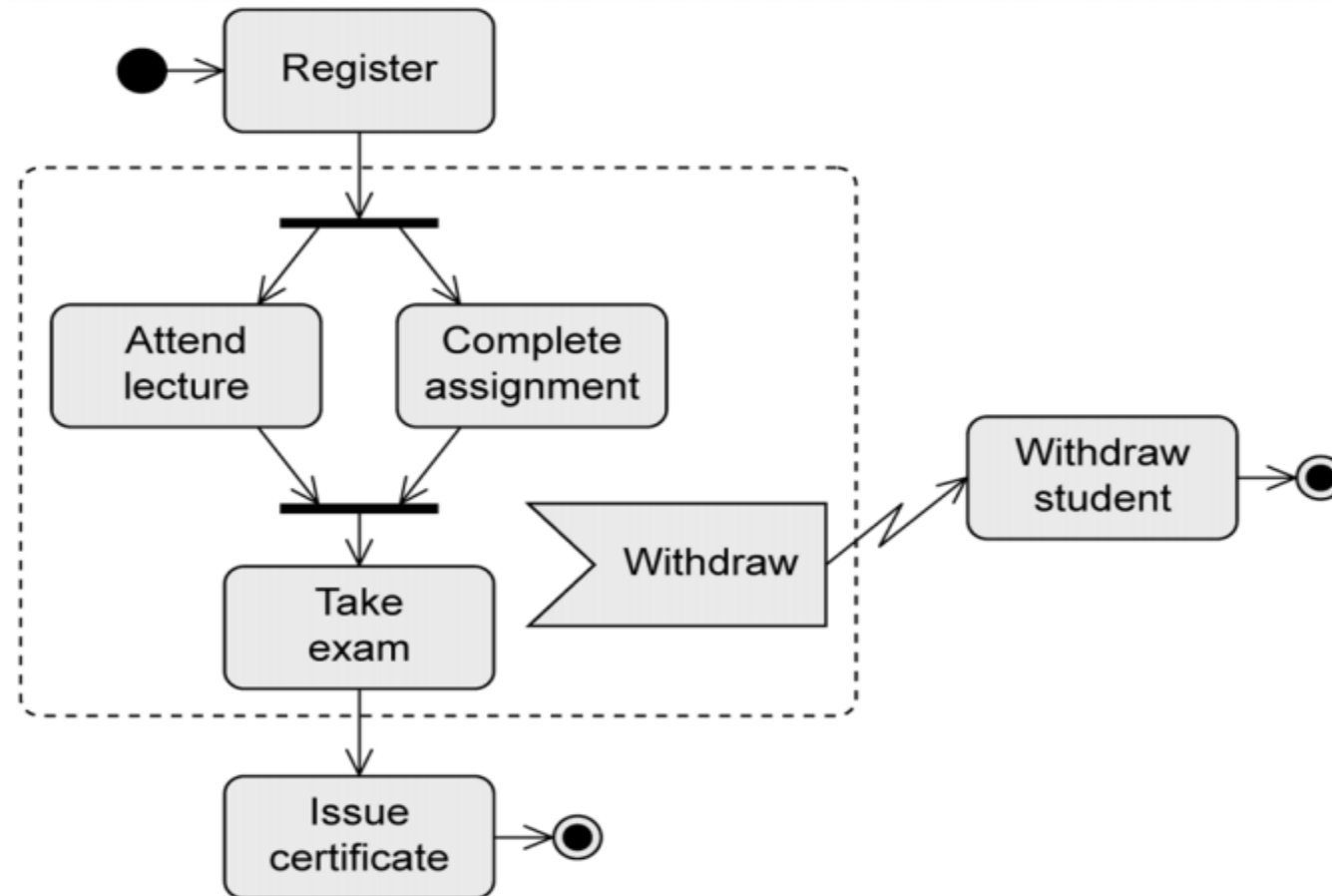




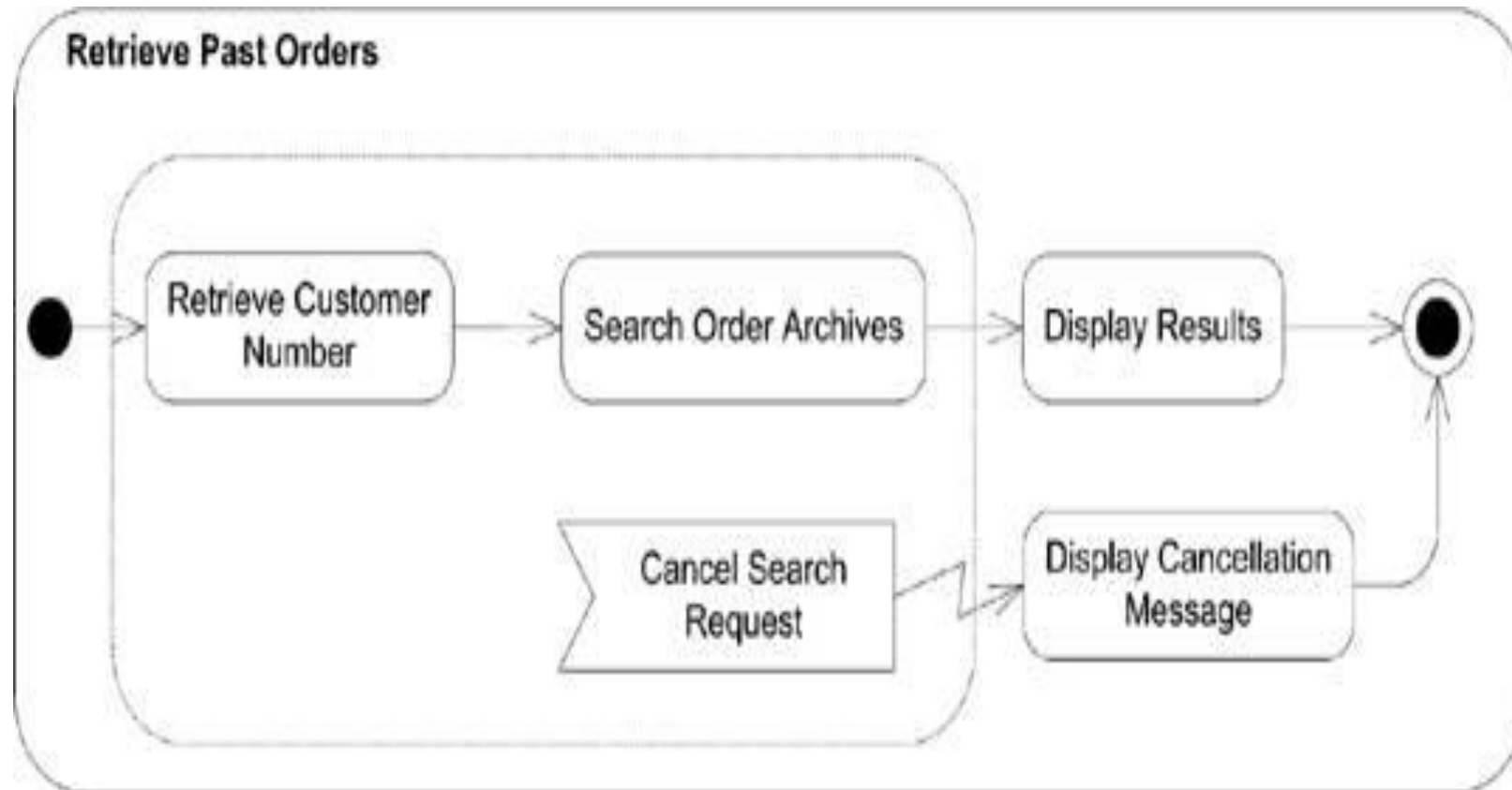
# EXAMPLE: REGISTER AND WITHDRAW



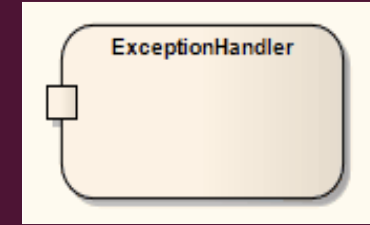
# EXAMPLE: REGISTER AND WITHDRAW



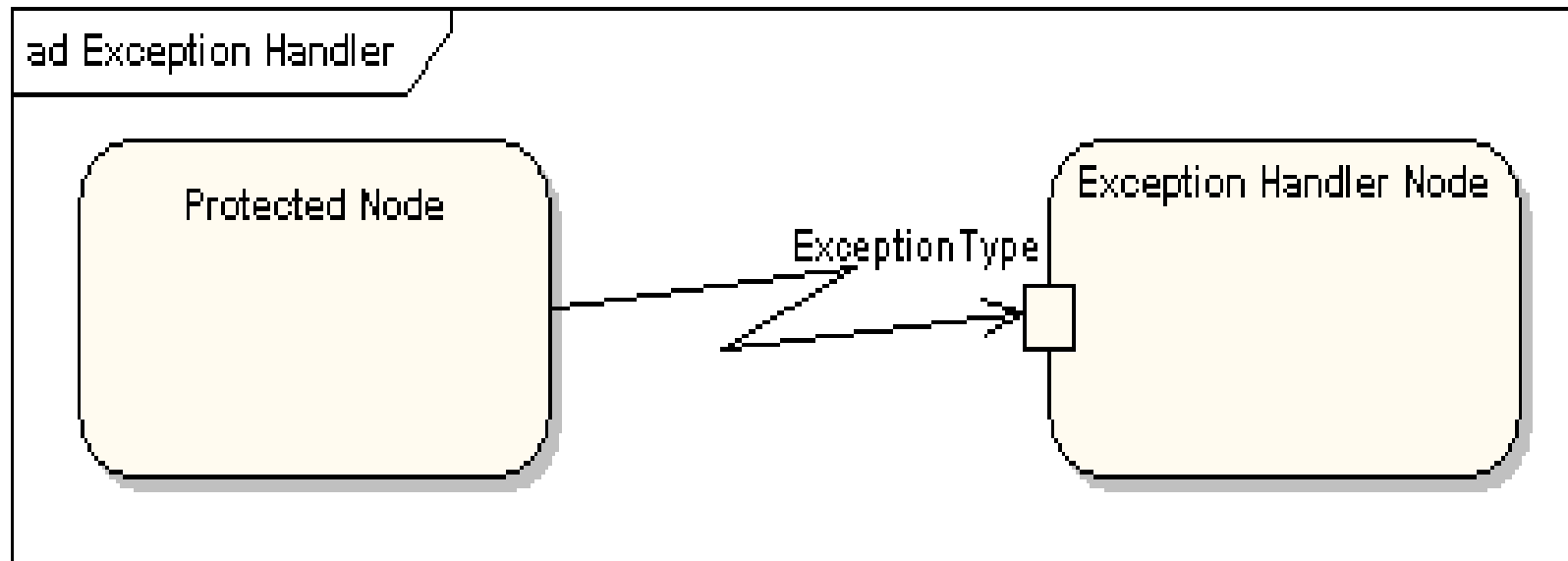
# INTERRUPTIBLE ACTIVITY REGION- EXAMPLE



# EXCEPTION HANDLER

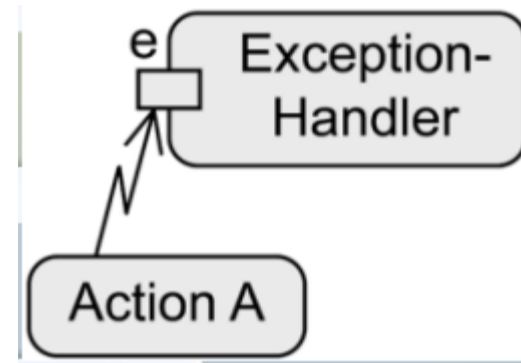


- An exception handler is an element that specifies a body to execute in case the specified exception occurs during the execution of the protected node.
- Exception Handlers can be modeled on activity diagrams as in the example below.



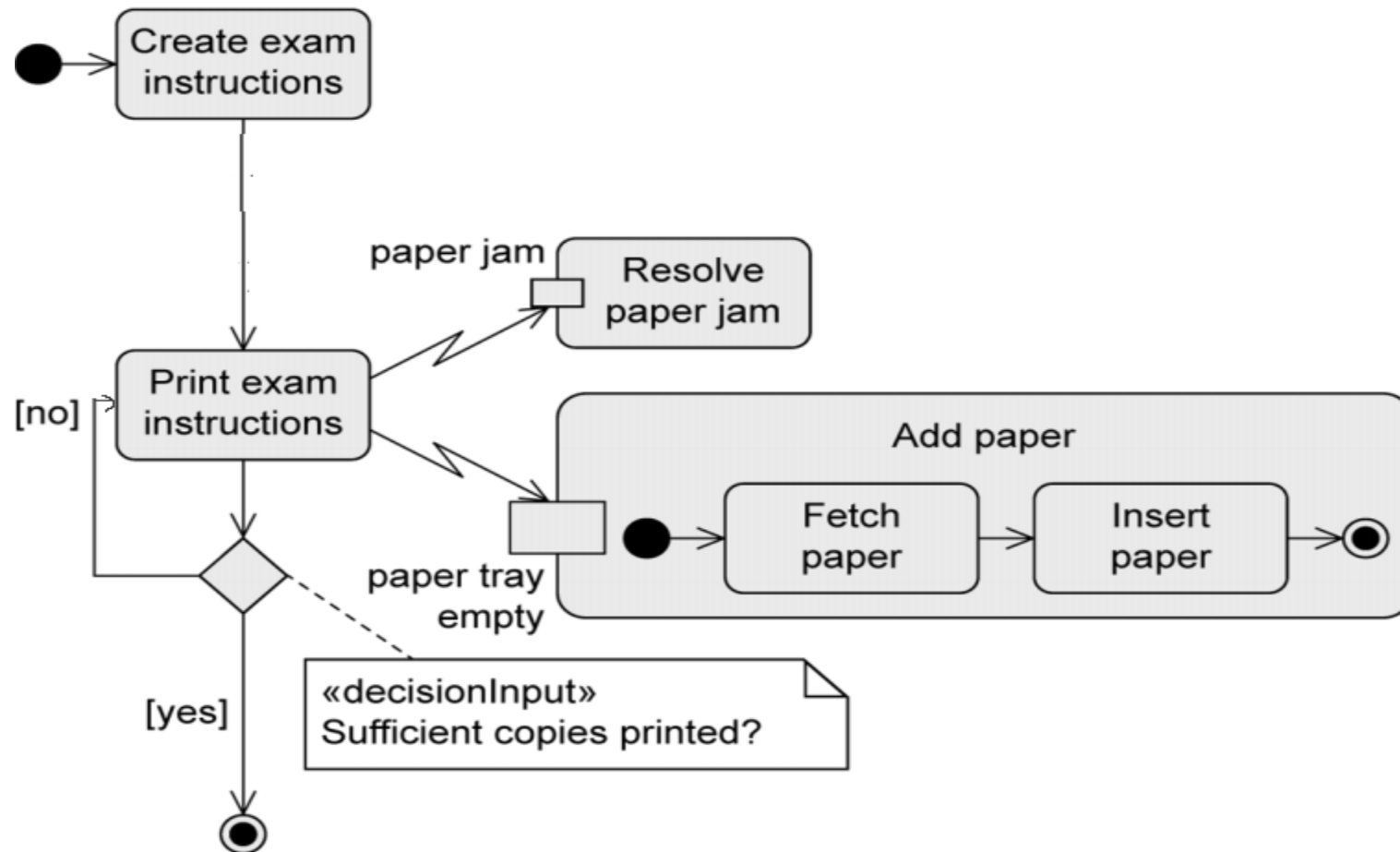
# EXCEPTION HANDLING

- If the error  $e$  occurs
- All tokens in Action A are deleted
- The exception handler is activated
- The exception handler is executed instead of Action A
- Execution then continues regularly



## EXAMPLE: EXCEPTION HANDLER

- You have been asked to develop a system that creates the exam instructions. These exam instructions are printed after the creation. As soon as the exam instruction copies are printed your system will be terminated.
- During printing activity, system may face the exception of paper Jam or paper tray is empty. System should be capable of handling both the exceptions. If the paper are not available, your system will fetch the paper and will insert the paper into the printer.

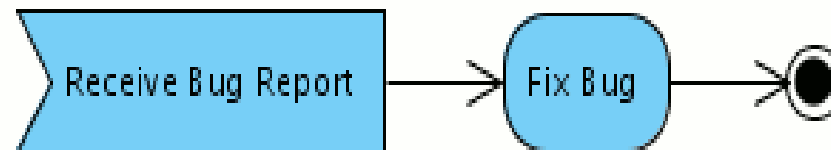


# ACCEPT EVENT ACTION

- AcceptEventAction is an action that waits for the occurrence of an event meeting specified condition.
- A receive signal "wakes up" an action that is "sleeping".



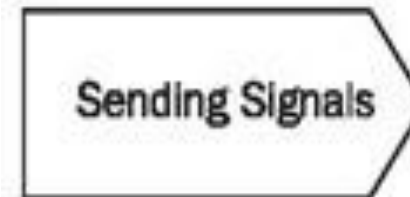
- An activity may also start with a **receive signal node**:





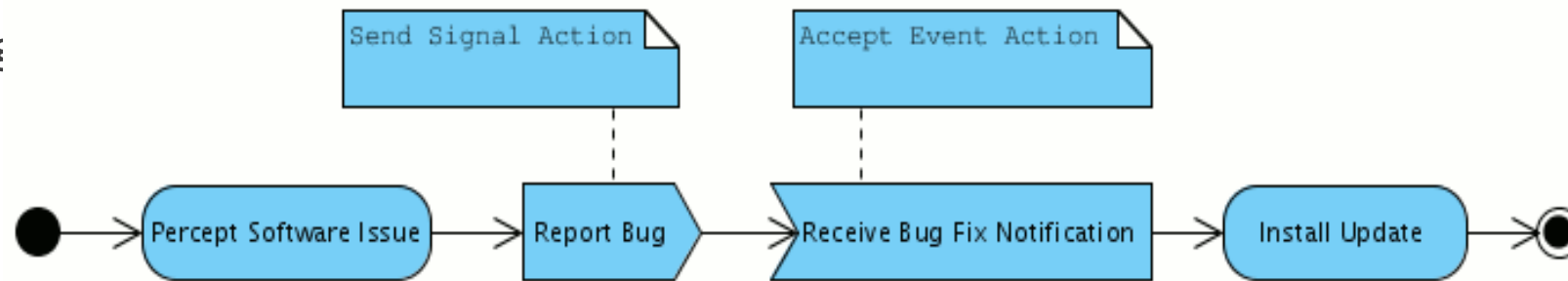
# SEND SIGNAL ACTION

- The sending of signals means that a signal is sent to a receiving activity: The receiving activity accepts the signal with the action accepting a signal and can respond accordingly, meaning, according to the flow that comes from this node in the activity diagram.
- *A send signal is sent to an external participant.*



# INTERACTING WITH EXTERNAL PARTICIPANTS

- In the following



- Note that the activity flow gets interrupted - gets into a wait state - until the bug fix notification is received. (If there was no receive signal action, however, the flow would just continue after executing the send signal action.)

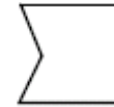
# ACCEPTEVENTACTION

- Waits for the occurrence of an event meeting specified conditions

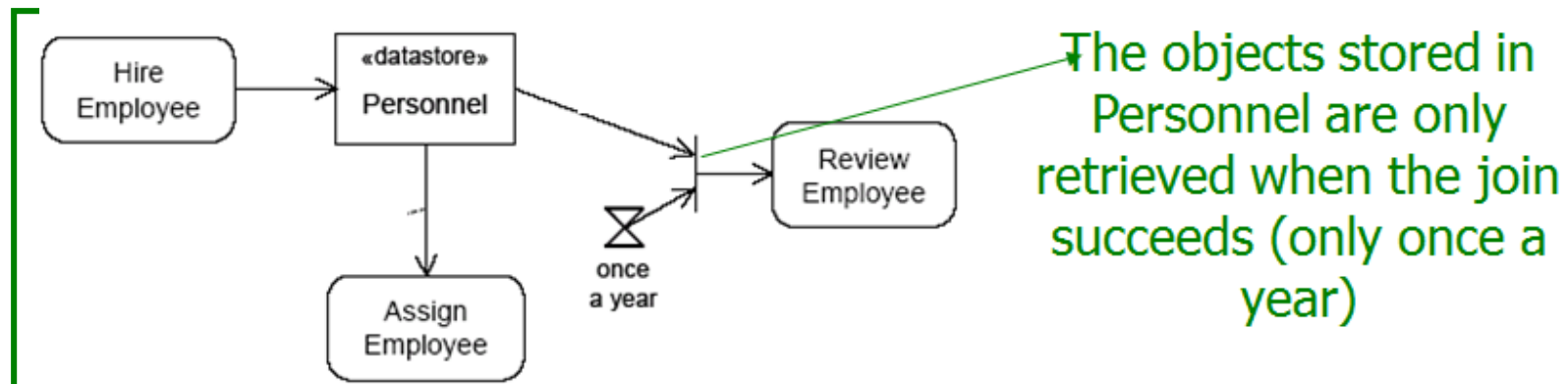
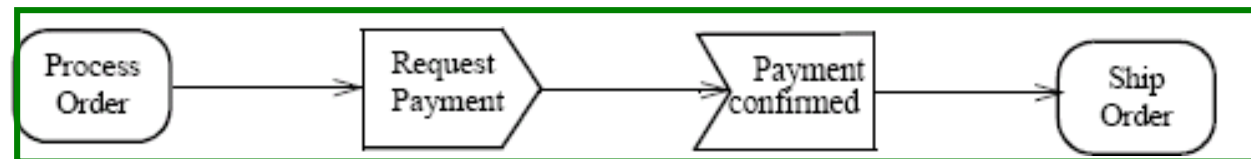
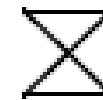
- Two kinds of AcceptEventAction:

- **Accept event action** – accepts signal events generated by a SendSignalAction
- **Wait time action** – accepts time events

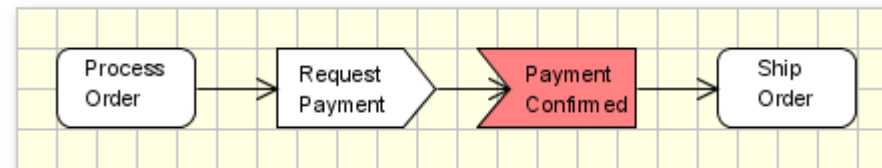
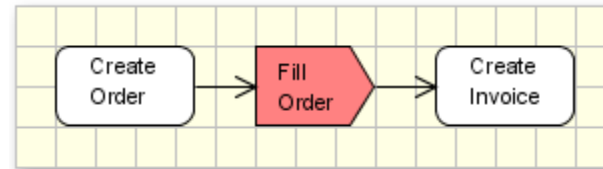
**Accept event action**



**Wait time action**

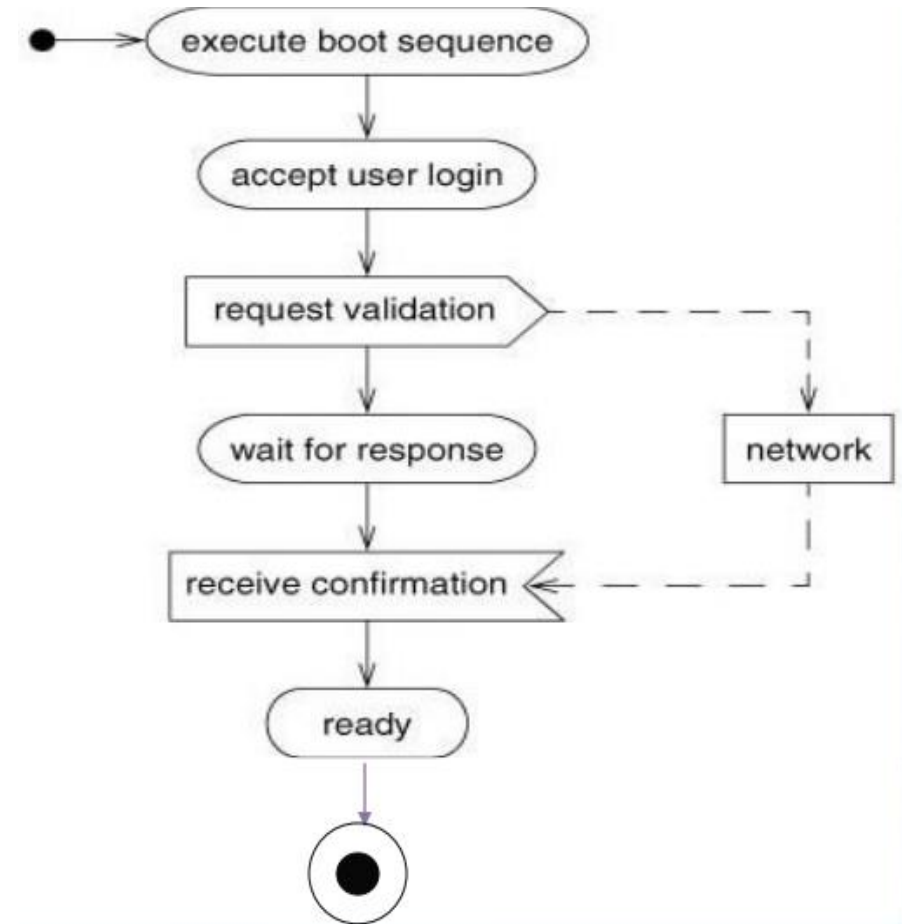


# SEND SIGNAL & ACCEPT EVENT ACTION



# SEND AND ACCEPT SIGNAL ACTIVITY

- Consider a workstation that is turned on. It goes through a boot sequence and then accepts the user login. After entering the name and password, the workstation queries the network to validate the user. Upon validation, the workstation then finishes its startup process and ready to use.

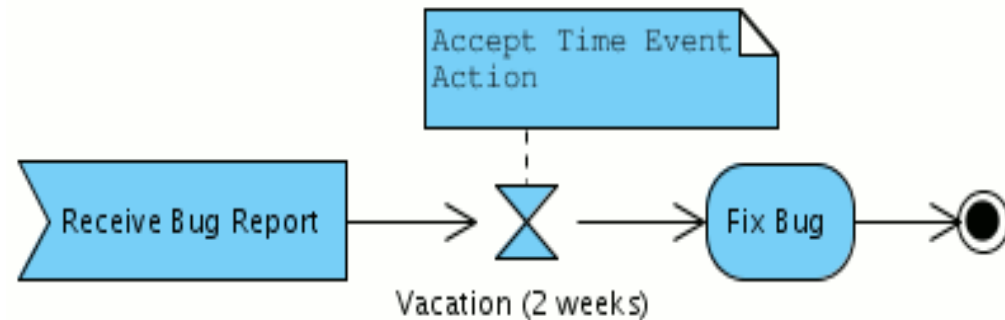


# ACCEPT TIME EVENT ACTION

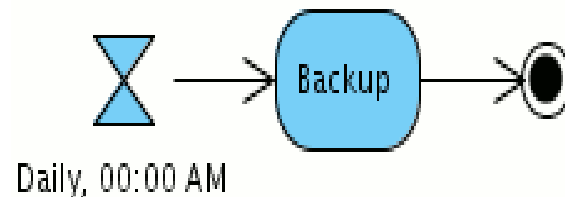


- If the occurrence is a time event occurrence, the result value contains the time at which the occurrence transpired. Such an action is informally called a wait time action.

- A **time event models a wait period:**

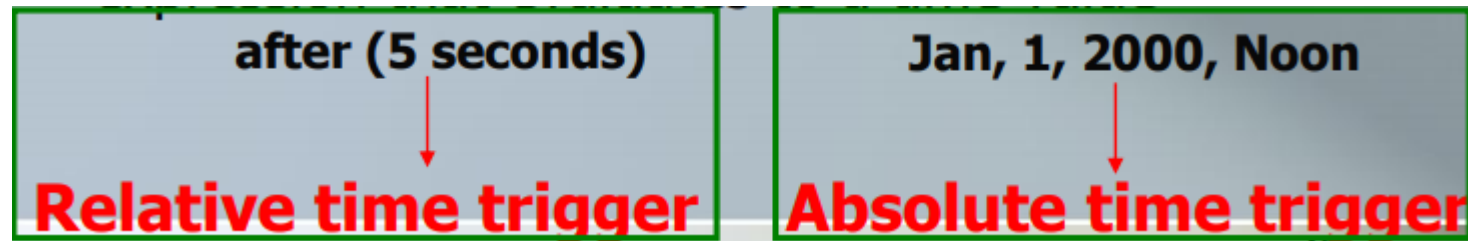


- An activity starting with a time event is launched periodically:

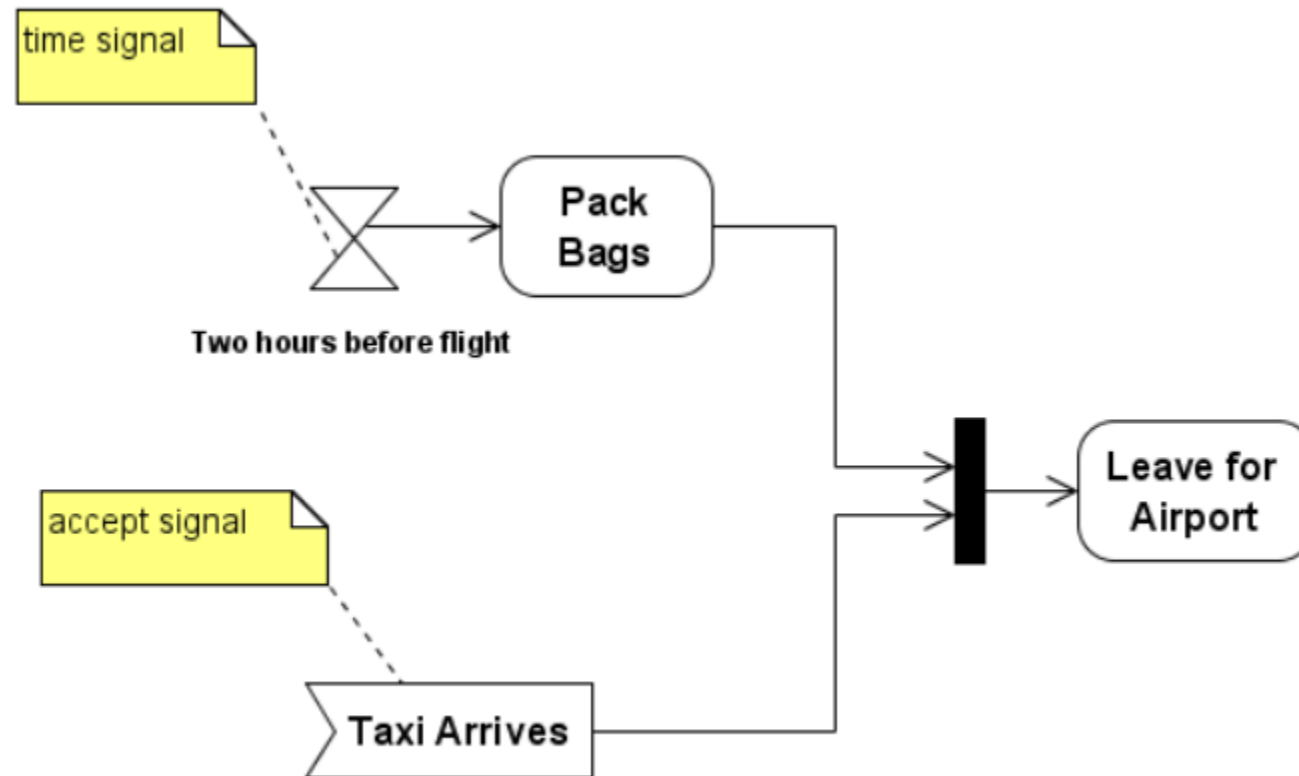


# TIME TRIGGER FORMS

- A Time trigger is a trigger that specifies when a time event will be generated
- This time may be relative or absolute
- Relative time trigger: is specified with the keyword 'after' followed by an expression that evaluates to a time value
- Absolute time trigger: is specified as an expression that evaluates to a time value



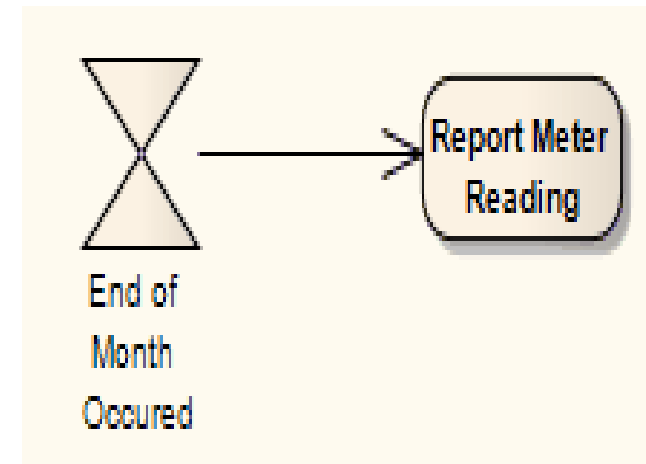
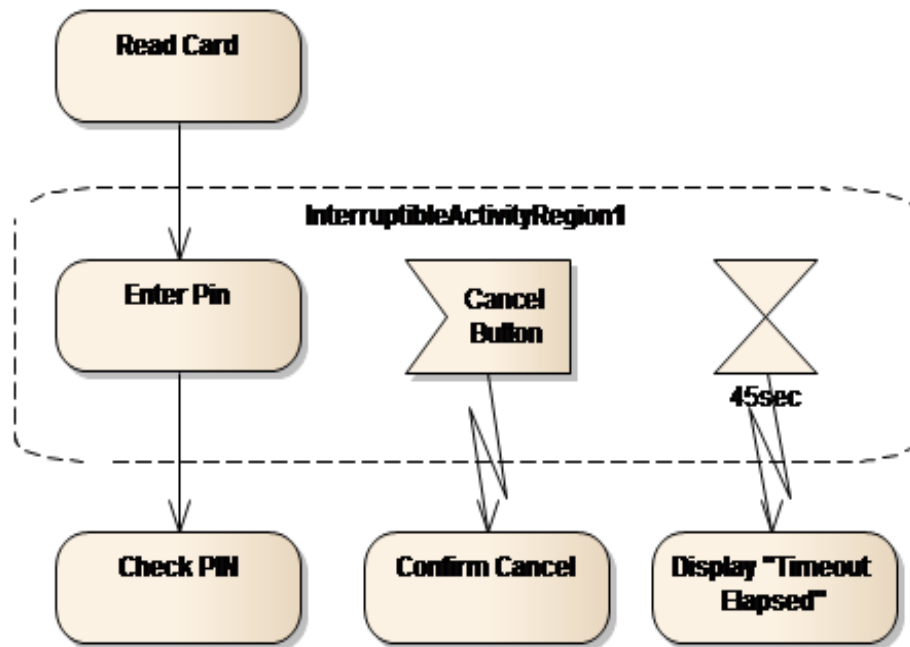
# TIME EVENT- EXAMPLE





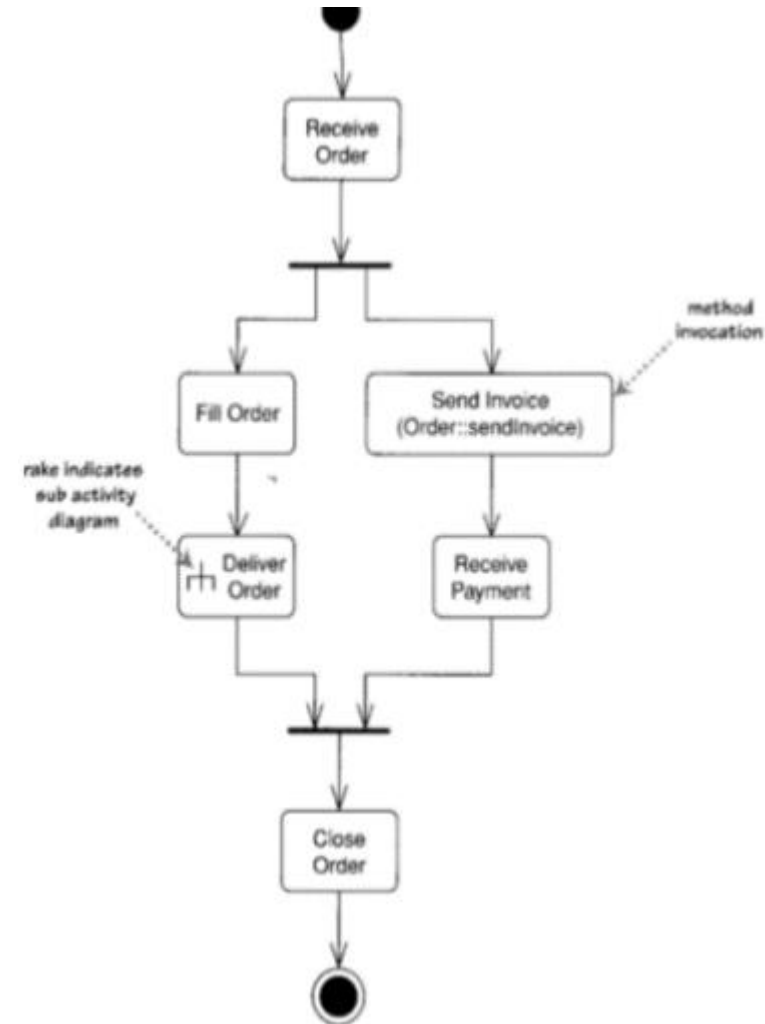
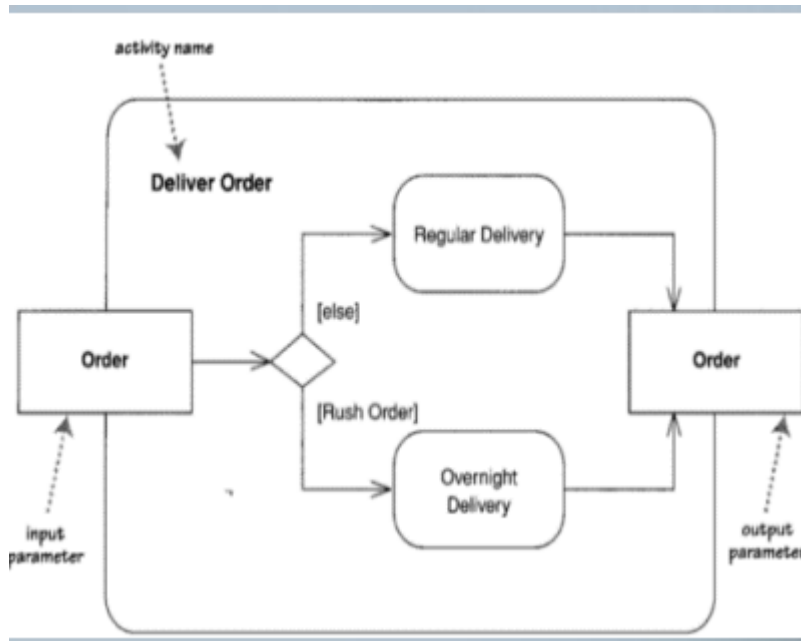
# TIME EVENT- EXAMPLES

- A typical **example** of a **time event** is triggering reminders after the deadline for payment has passed.
- Another example is timeout.

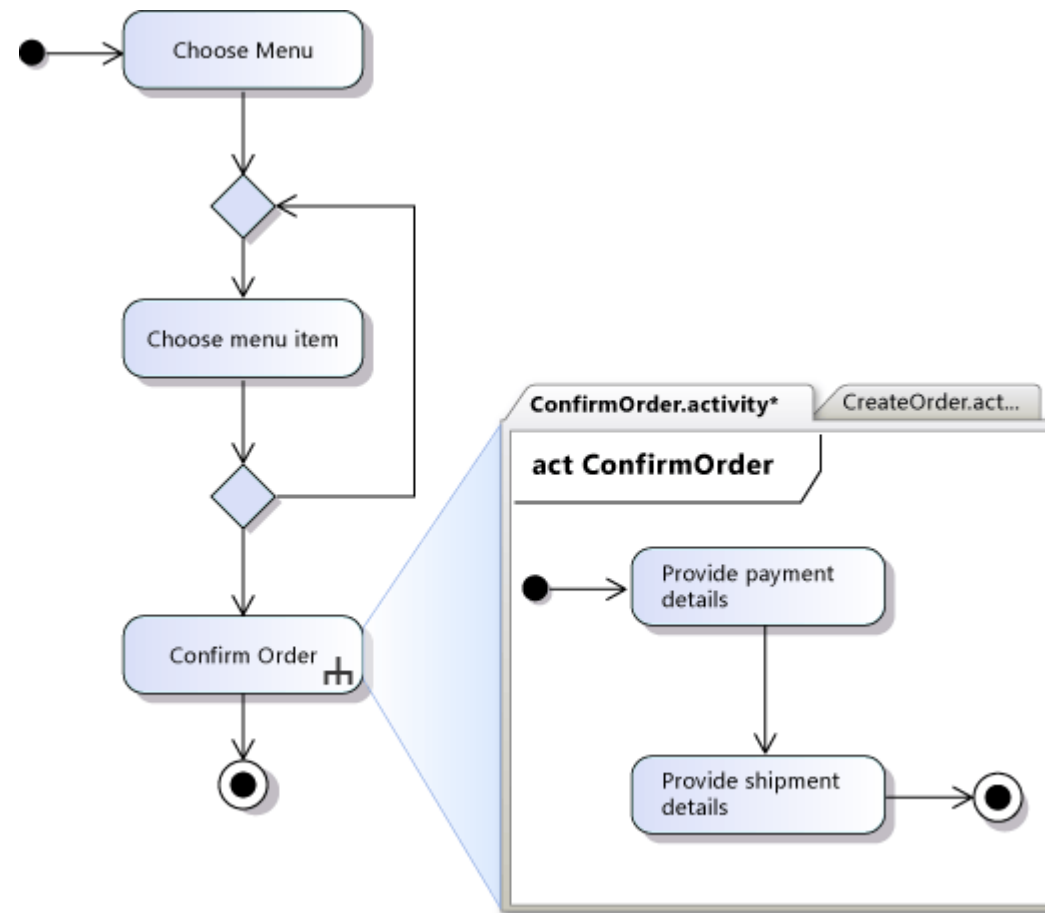


# HOW TO REPRESENT SUB- ACTIVITY DIAGRAM

In a UML Activity Diagram, an Action representing a Sub-activity diagram can be signaled with a "trident" icon,

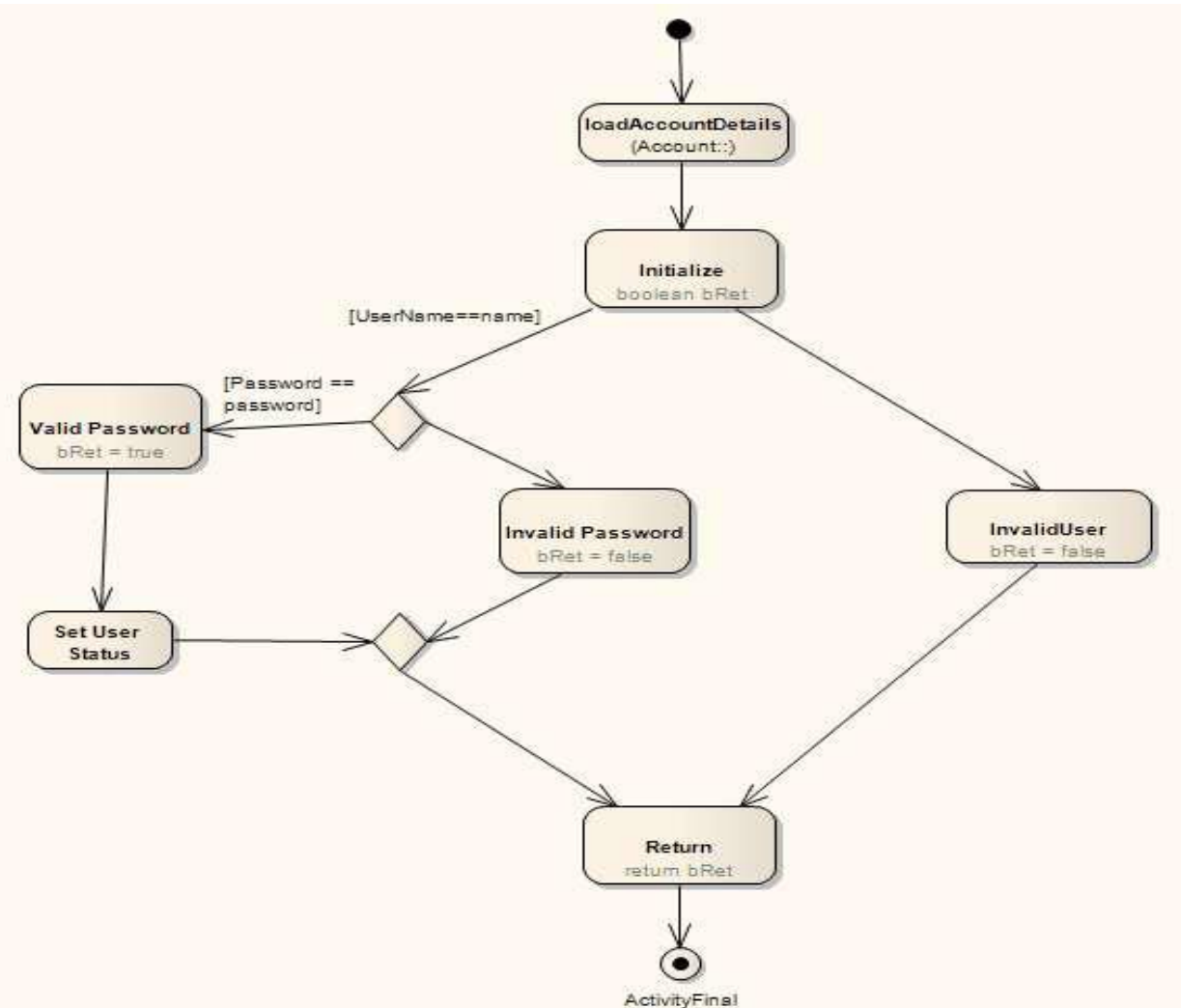


## ANOTHER EXAMPLE



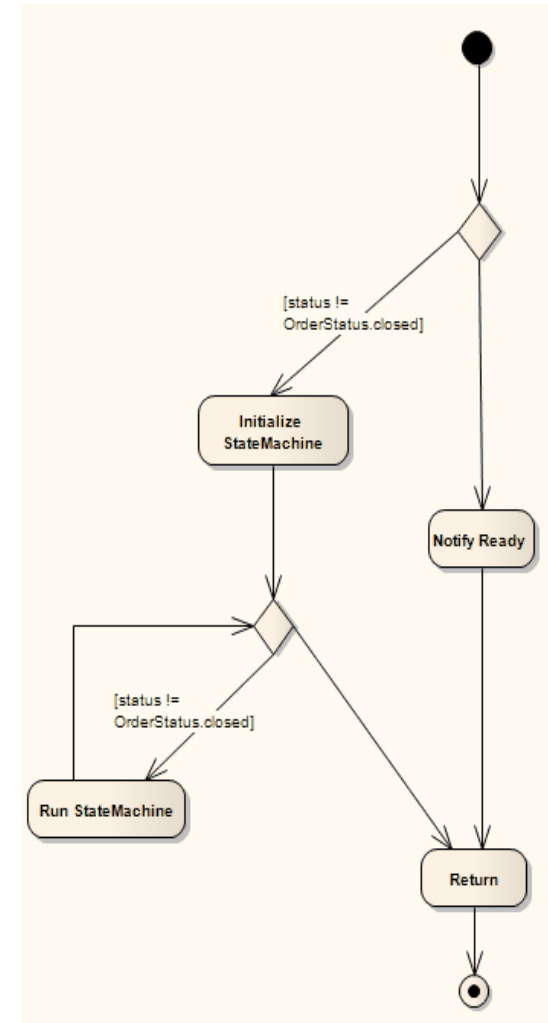
# CONDITIONAL STATEMENTS

```
public boolean doValidateUser(String Password,String UserName)
{
    loadAccountDetails();
    boolean bRet;
    if (Username==name)
    {
        if (Password == password)
        {
            bRet = true;
            bValidUser = true;
        }
        else
        {
            bRet = false;
        }
    }
    else
    {
        bRet = false;
    }
    return bRet;
}
```






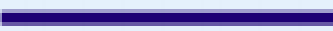
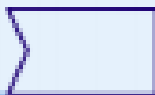
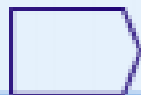




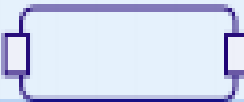
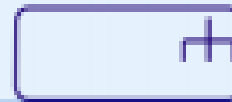



# LOOPS

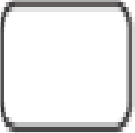




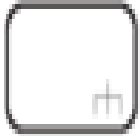

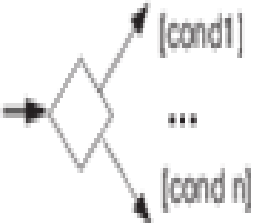
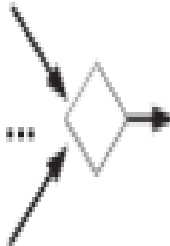
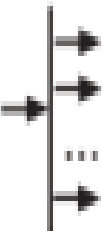
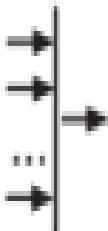
```
public void doCheckForOutstandingOrders()  
{  
    if (status != closed)  
    {  
        initializeStateMachine();  
        while (status != closed)  
        {  
            runStateMachine();  
        }  
    }  
    else  
    {  
        Notify ready;  
    }  
    return;  
}
```



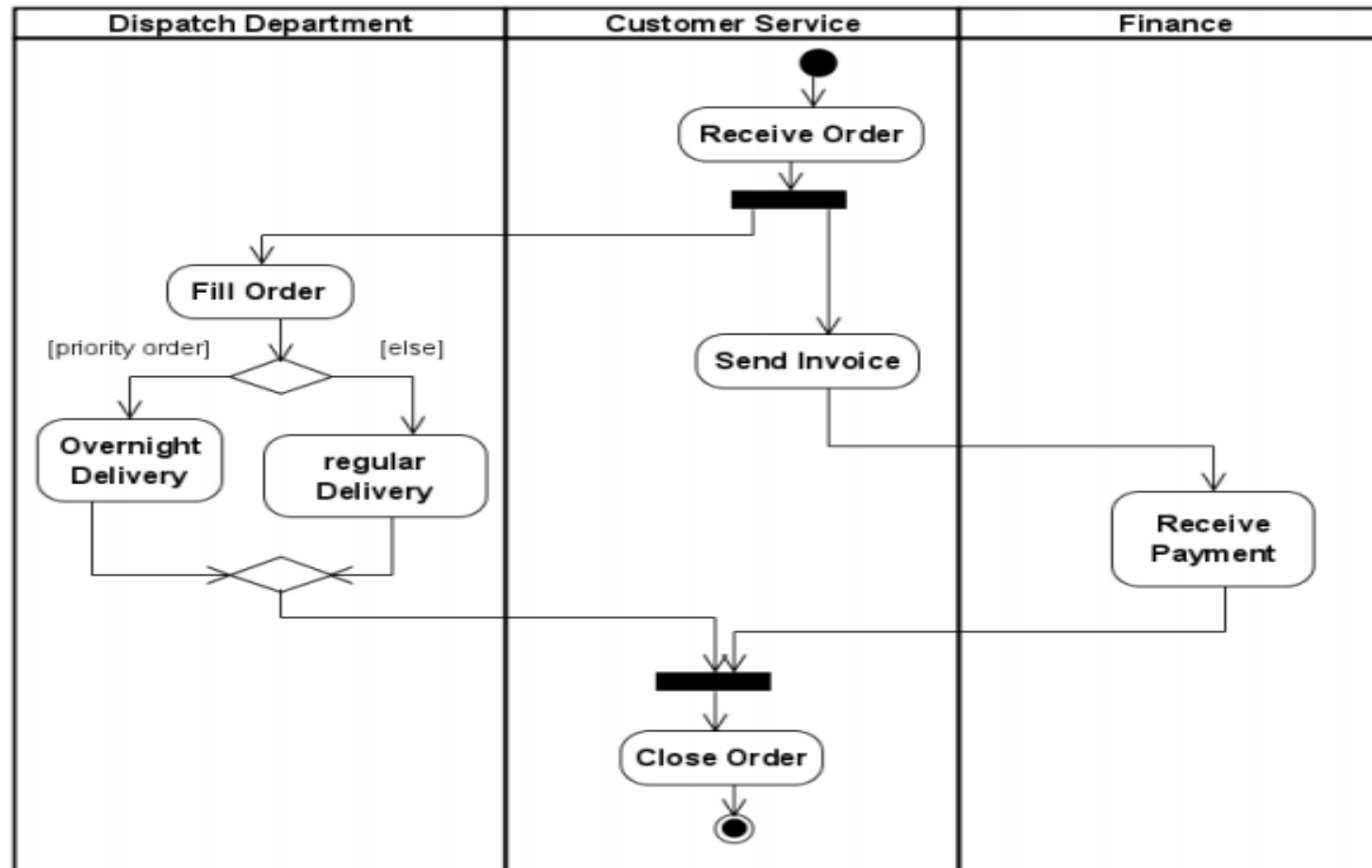
## SOME MORE NOTATIONS

	Graphical representation	Description
Action		action with three inputs
Control flow	  	start / stop markers
		decision, merge
		fork / join
Signals	  	incoming (accept), outgoing (send), time-based
Interrupts	  	interruptible activity region, interrupting edge
Subactivity	 	activity with input/output parameters, activity invocation
Collection		expansion region

## SOME MORE NOTATIONS

Action/Activity 	AcceptEvent 	InitialNode 	ActivityFinal 	FlowFinal 
CallBehaviorAction 	SendSignal 	Decision 	Merge 	Fork  Join 
a) Actions		b) Control Nodes		

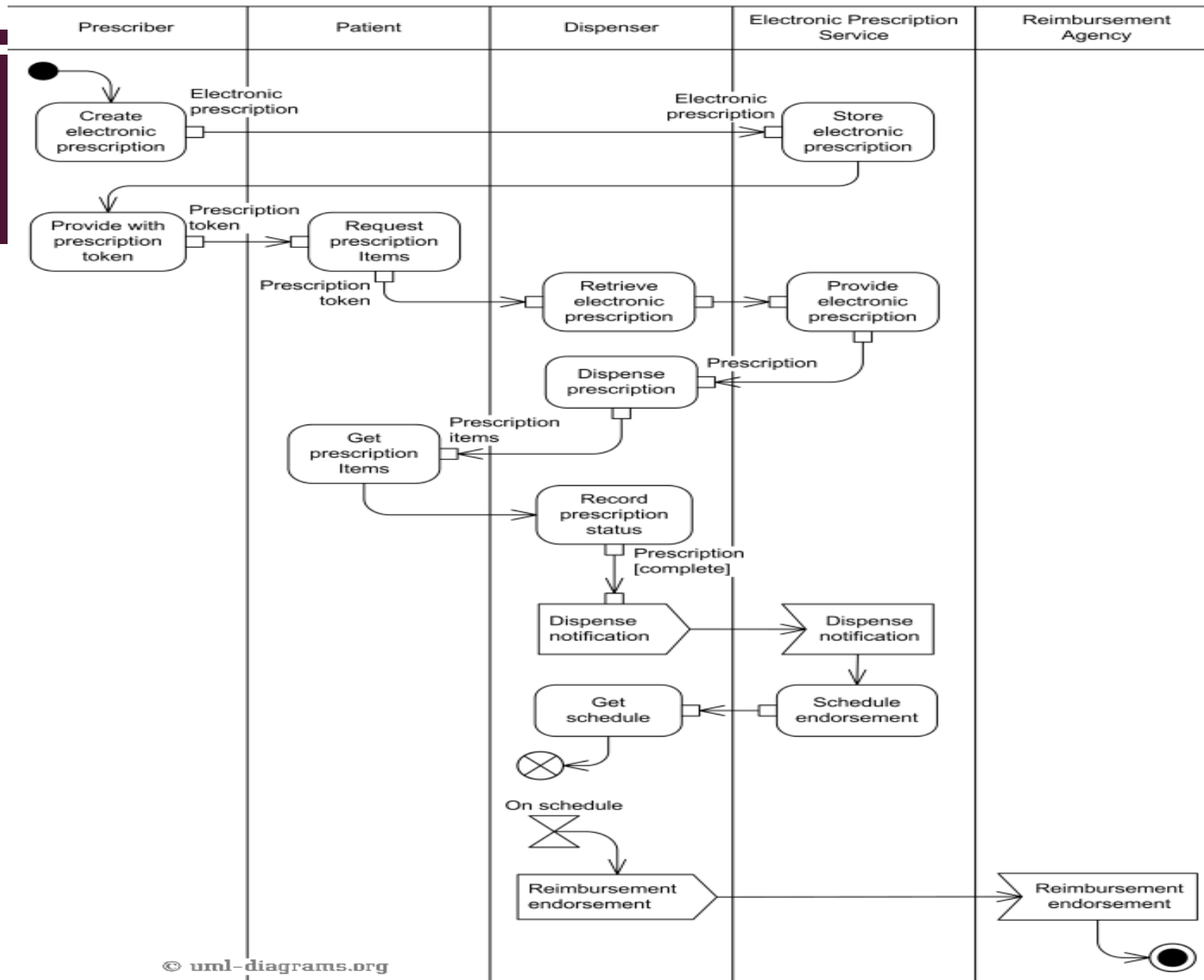
# EXAMPLE





## ELECTRONIC PRESCRIPTIONS ACTIVITY DIAGRAM-EXAMPLE

- This is an example of activity diagram for electronic prescriptions. The Electronic Prescription Service (EPS) enables prescribers - such as general practitioners (GPs) and practice nurses - to send prescriptions electronically to a dispenser (such as a pharmacy) of the patient's choice.
- Prescriber logs onto the clinical system using their Smartcard and passcode, chooses medication or medical appliance for the patient, adds prescribing endorsements where required, and applies electronic signature to authorize the electronic prescription. Electronic prescription is transmitted to the EPS which store it and a token is provided to the prescriber . Prescription token is printed where required. Authorized person hands prescription token to patient where necessary.
- When patient request the prescribed items, Dispenser retrieves electronic prescriptions and save a copy of electronic prescription in the EPS. Dispenser dispense the prescription and prescription items are issued to the patient.
- Dispenser should record the status of each of the prescription items. If dispensing process is complete, dispenser should send dispense notification to the Electronic Prescription Service. Upon receiving this message, the EPS will issue a schedule as to when to submit the electronic reimbursement endorsement message. Dispenser gets this scheduling information.
- To support the reimbursement, claim process, the EPS will allow dispensers to electronically submit reimbursement endorsement messages to the reimbursement agency for the dispensed electronic prescriptions so that the reimbursement agency can make a payment. The messages are sent according to the reimbursement agency scheduling.





That is all