Software Design and Analysis

CS-3004

Activity Diagram

Lecture # 22, 23, 24
25, 27, 28 Oct

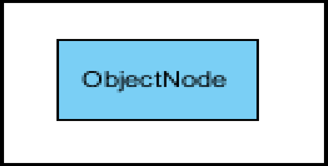
Rubab Jaffar
rubab.jaffar@nu.edu.pk

Today's Outline

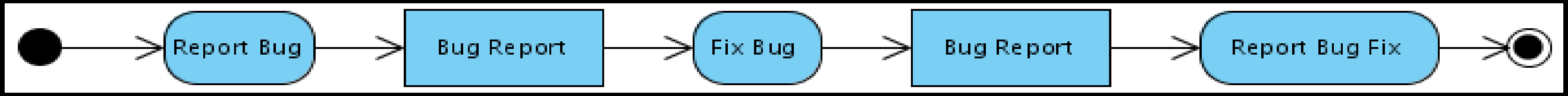
- Some more features in activity diagrams
- Object and object flow
- Input and Output pin
- Data Store
- Regions
  - Expansion region
  - Interruptible region
- Exception Handling
- Conditional statements
- Class Activity

Object and Object Flow

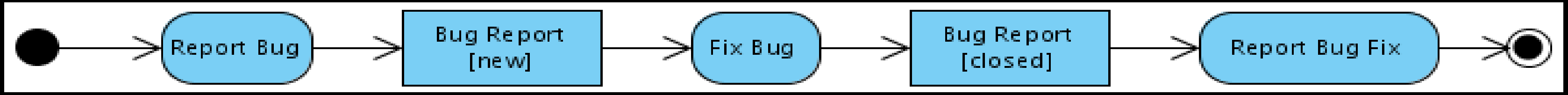
- Object Node:



- 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:

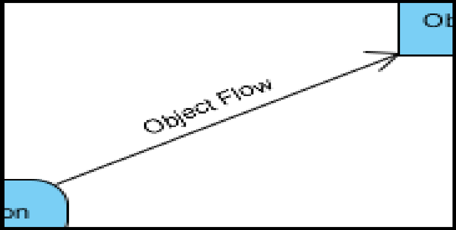


- Objects usually change state between actions:

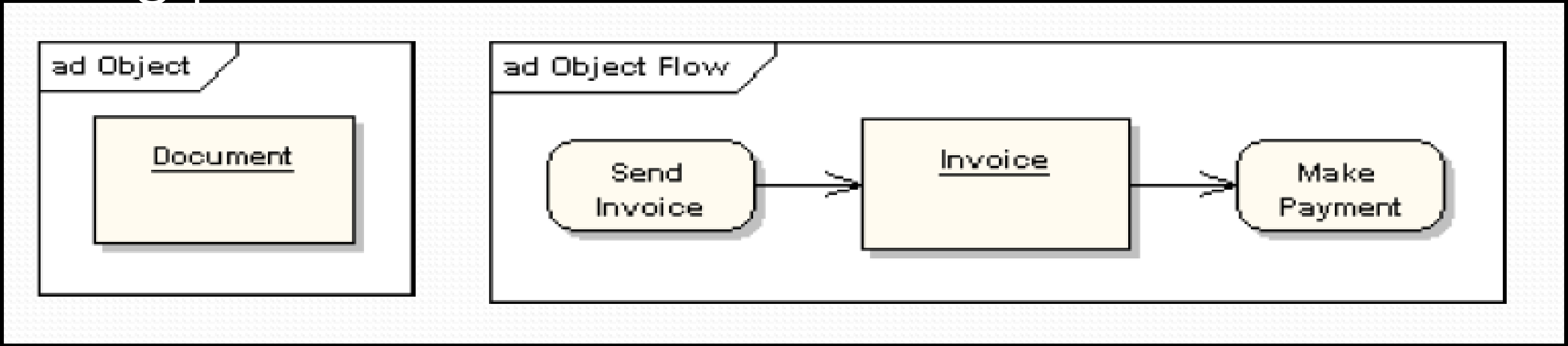


Object and Object Flow

- 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.

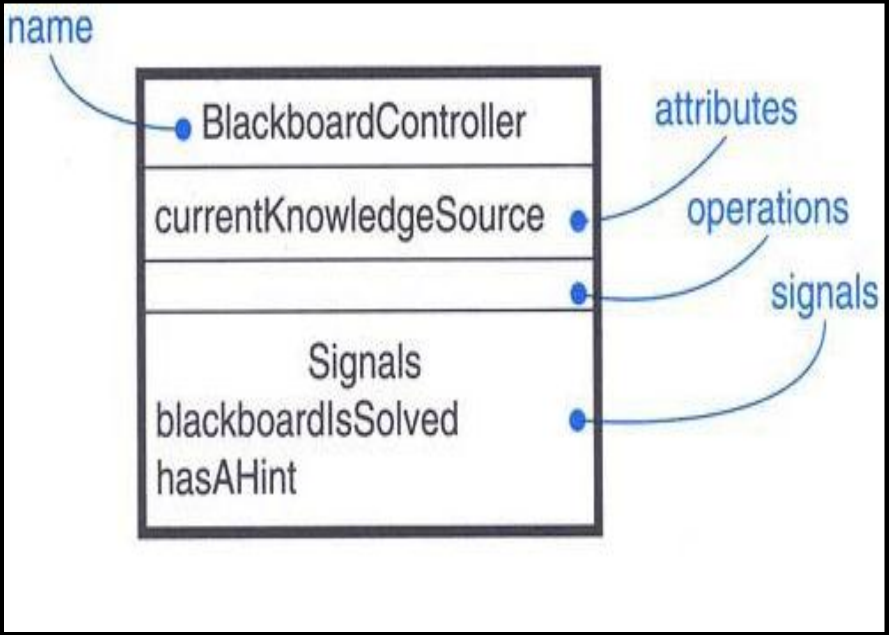


Active Objects and Active Classes

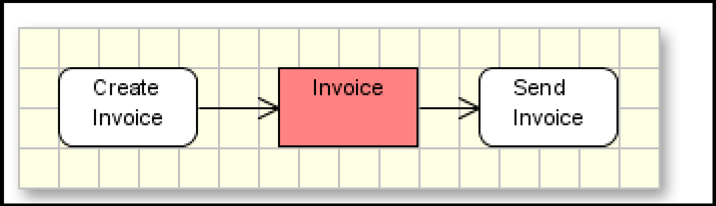
- Processes and Threads
- A process is a heavyweight flow that can execute concurrently with other processes.
- A thread is a lightweight flow that can execute concurrently with other threads within the same process.
- An active object is an object that owns a process or thread and can initiate control activity.
- An active class is a class whose instances are active objects.
- Graphically, an active class is rendered as a rectangle with thick lines. Processes and threads are rendered as stereotyped active classes.

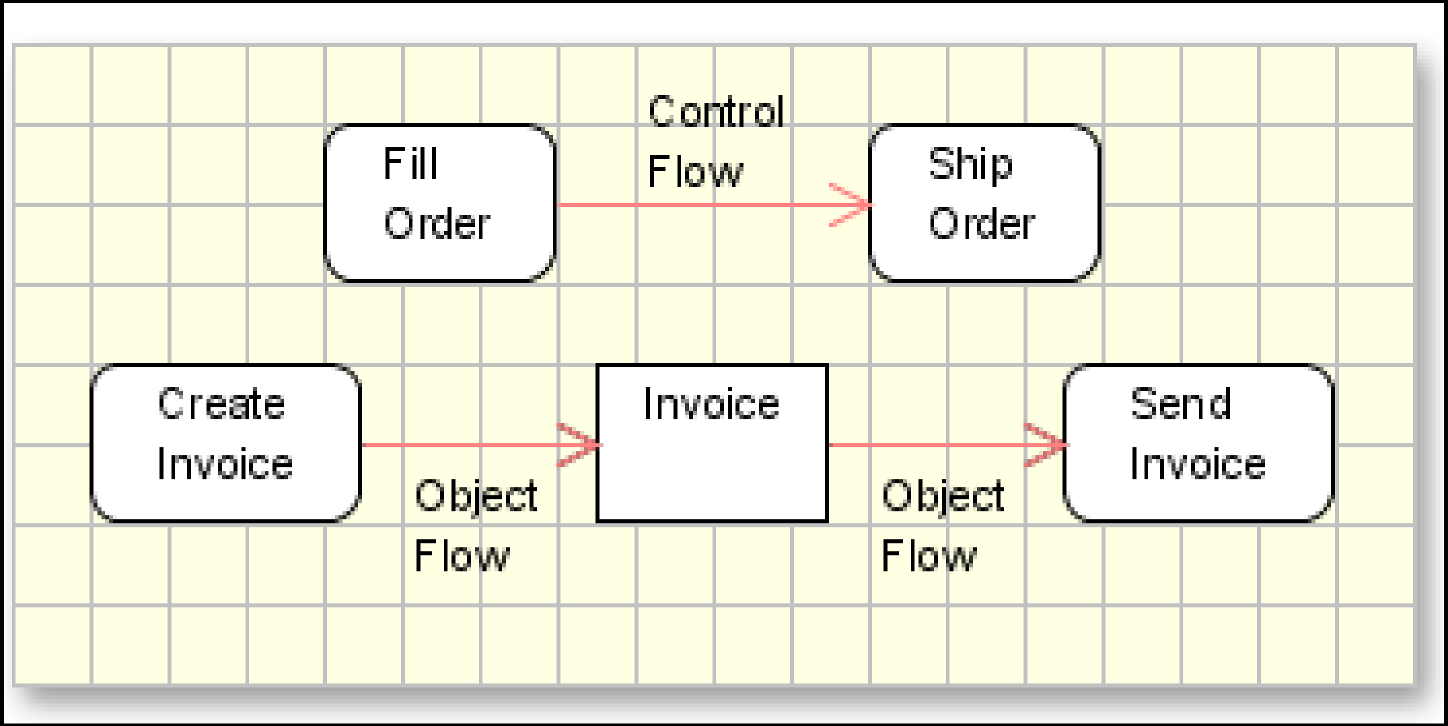
An Active Class

•Active classes are just classes which represents an independent flow of control
•Active classes share the same properties as all other classes.
•When an active object is created, the associated flow of control is started; when the active object is destroyed, the associated flow of control is terminated



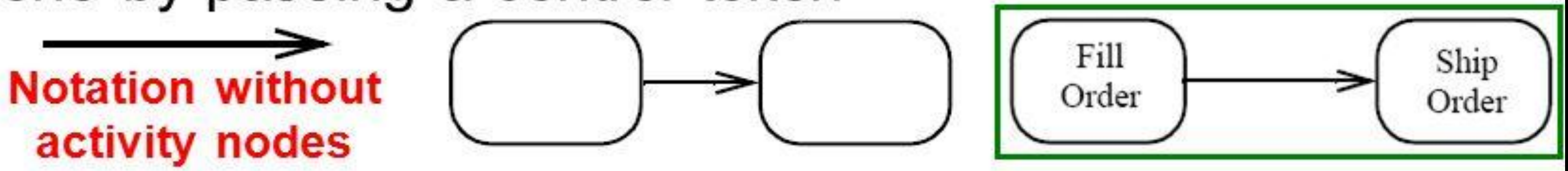
Object Flow and Control Flow



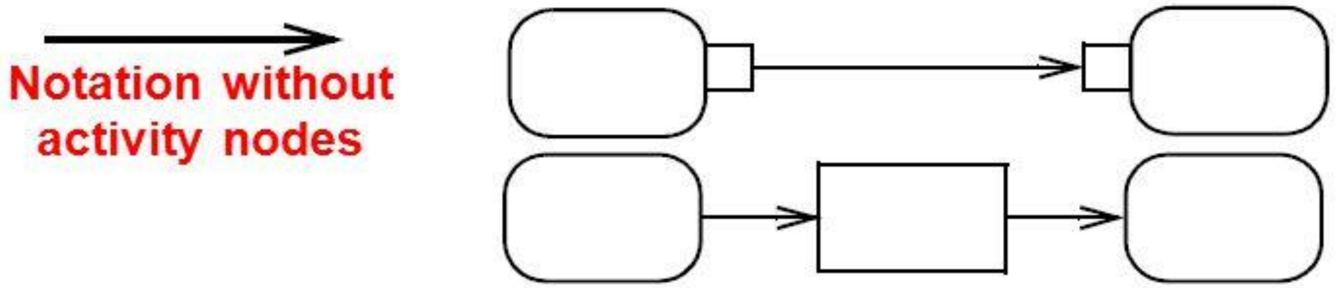


Activity edges (2)

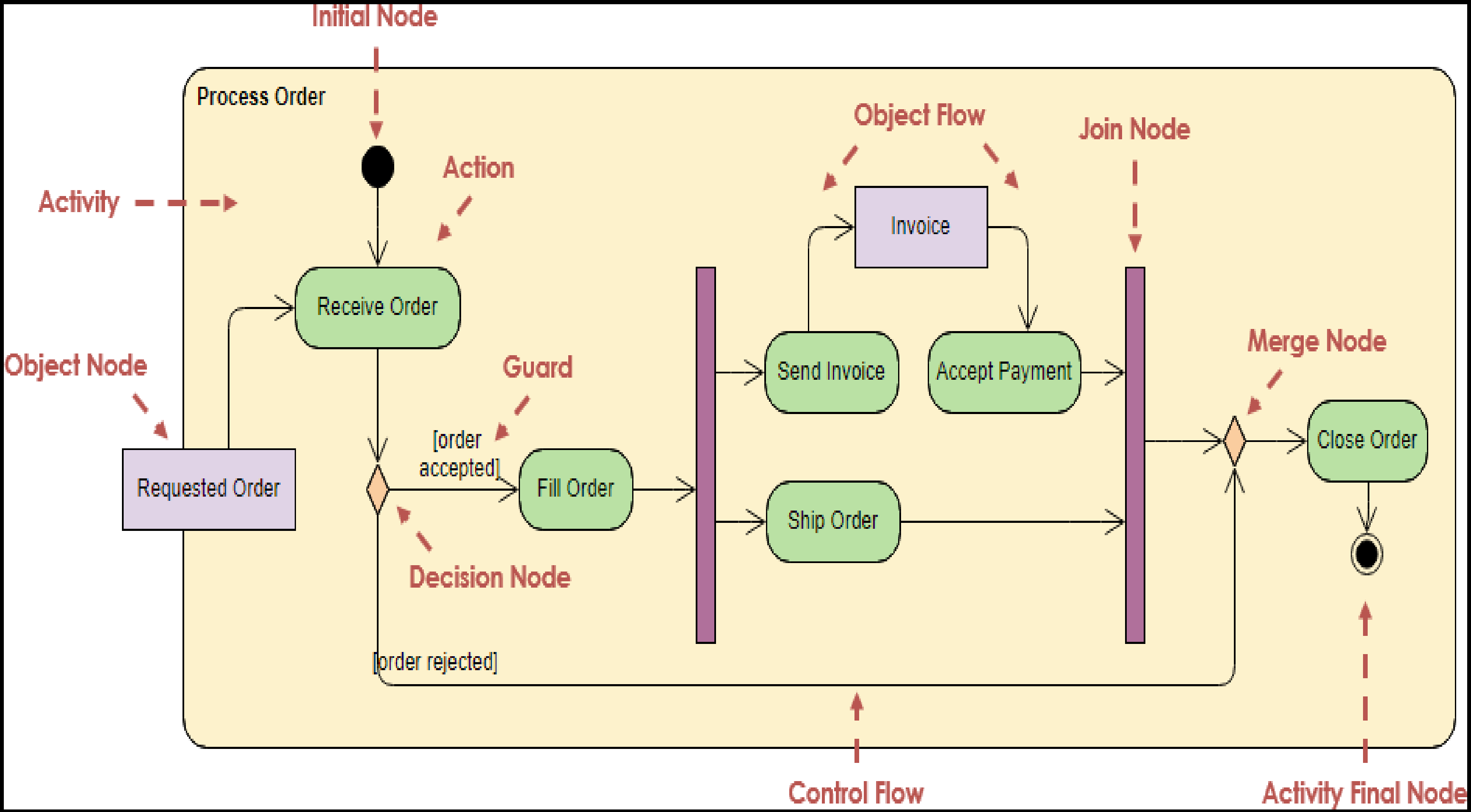
- Two kinds of edges:
  - **Control flow edge** - is an edge which starts an activity node after the completion of the previous one by passing a control token



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

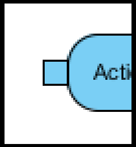


Example Object Flow and Control Flow

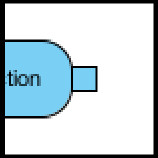


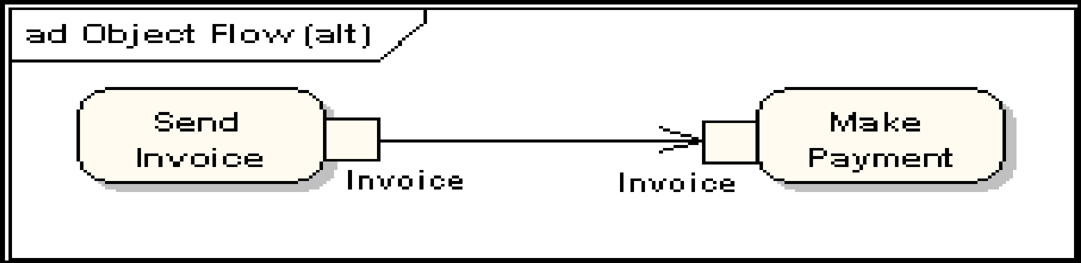
Input and Output Pins

- 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

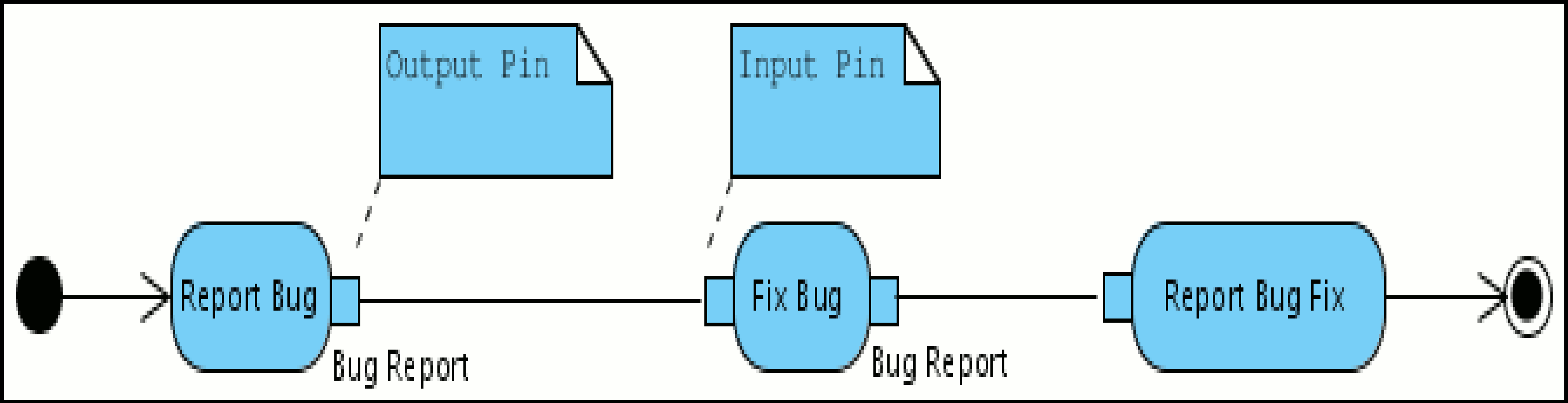


- Output pins are object nodes that deliver values to other actions through object flows.

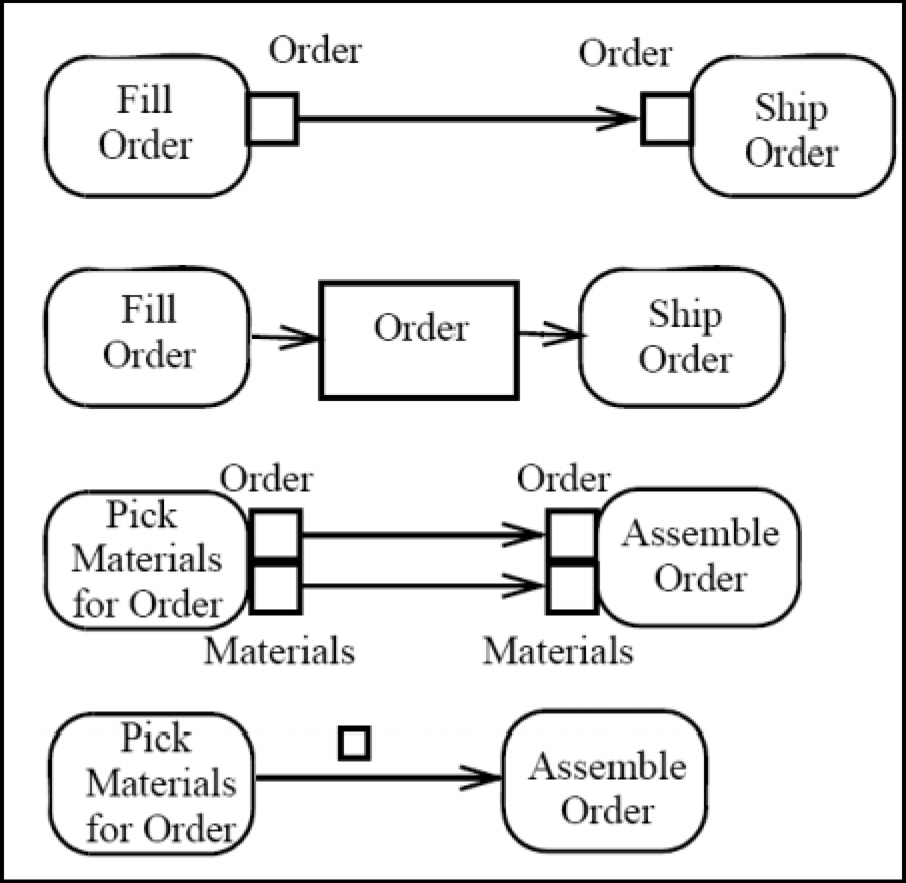




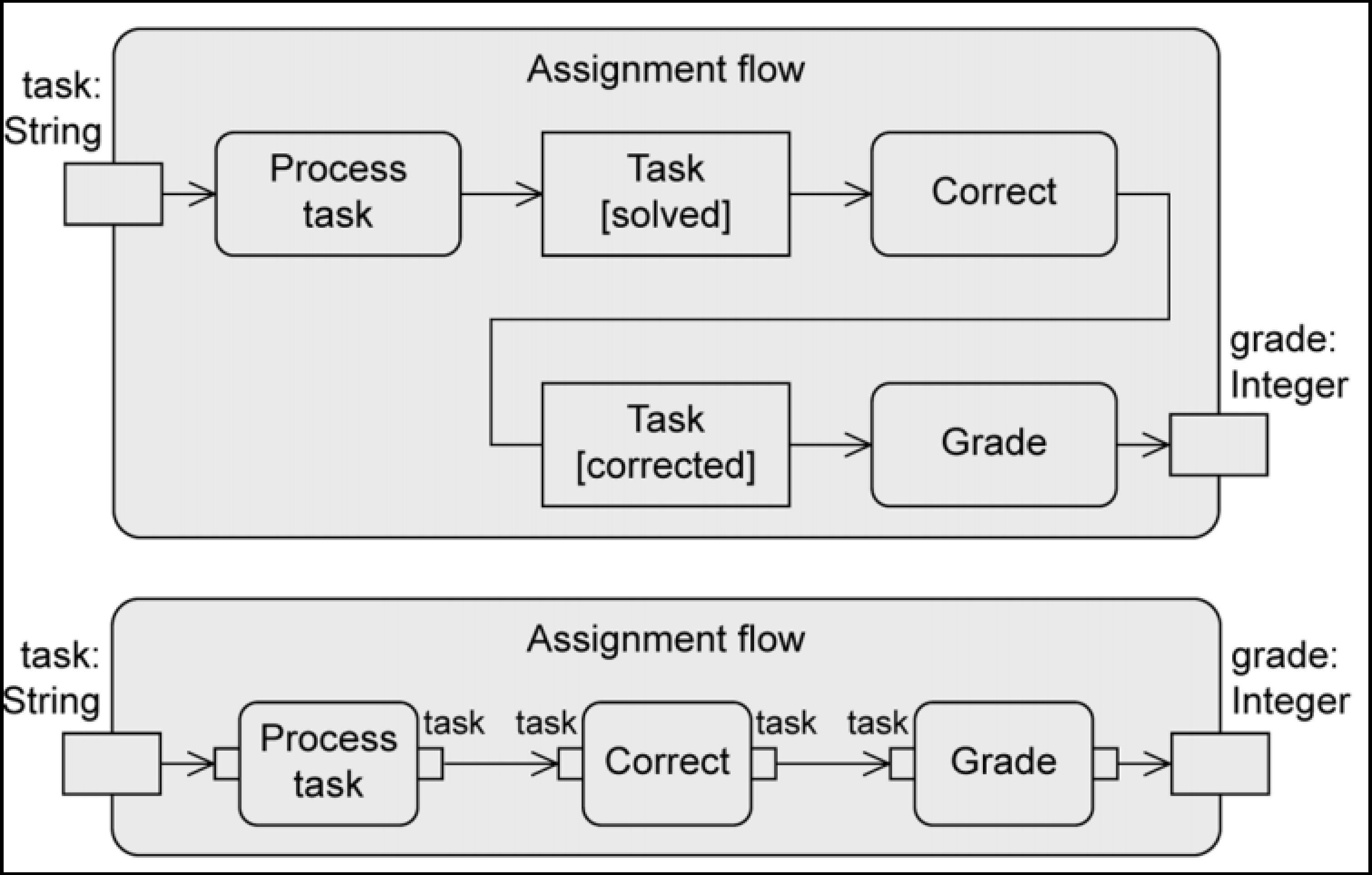
Input and Output Pins



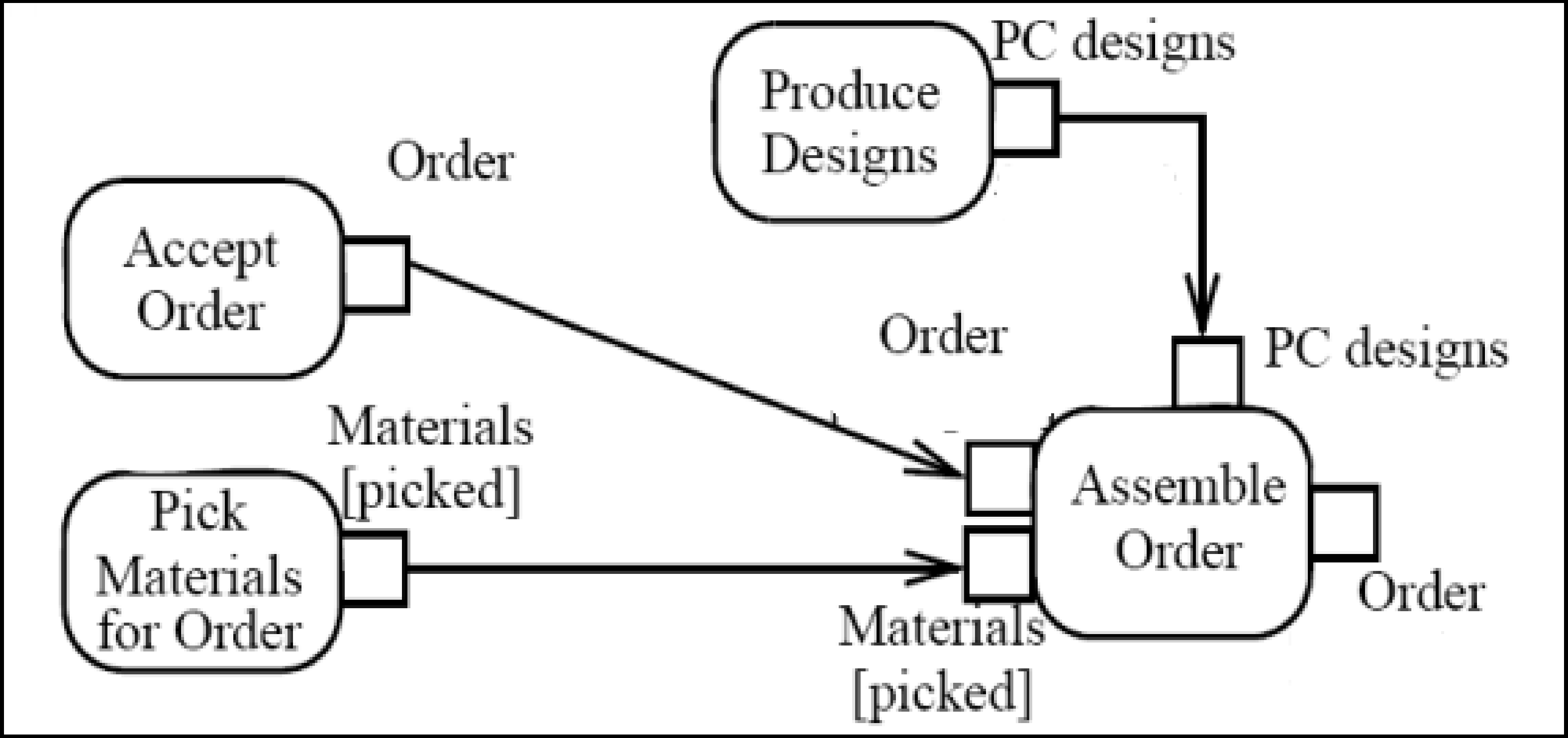
Input and Output Pins-Representations



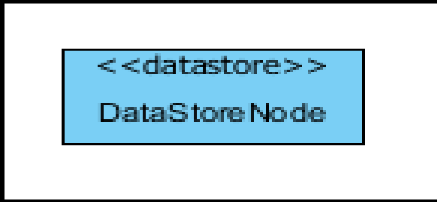
Example: Assignment Flow



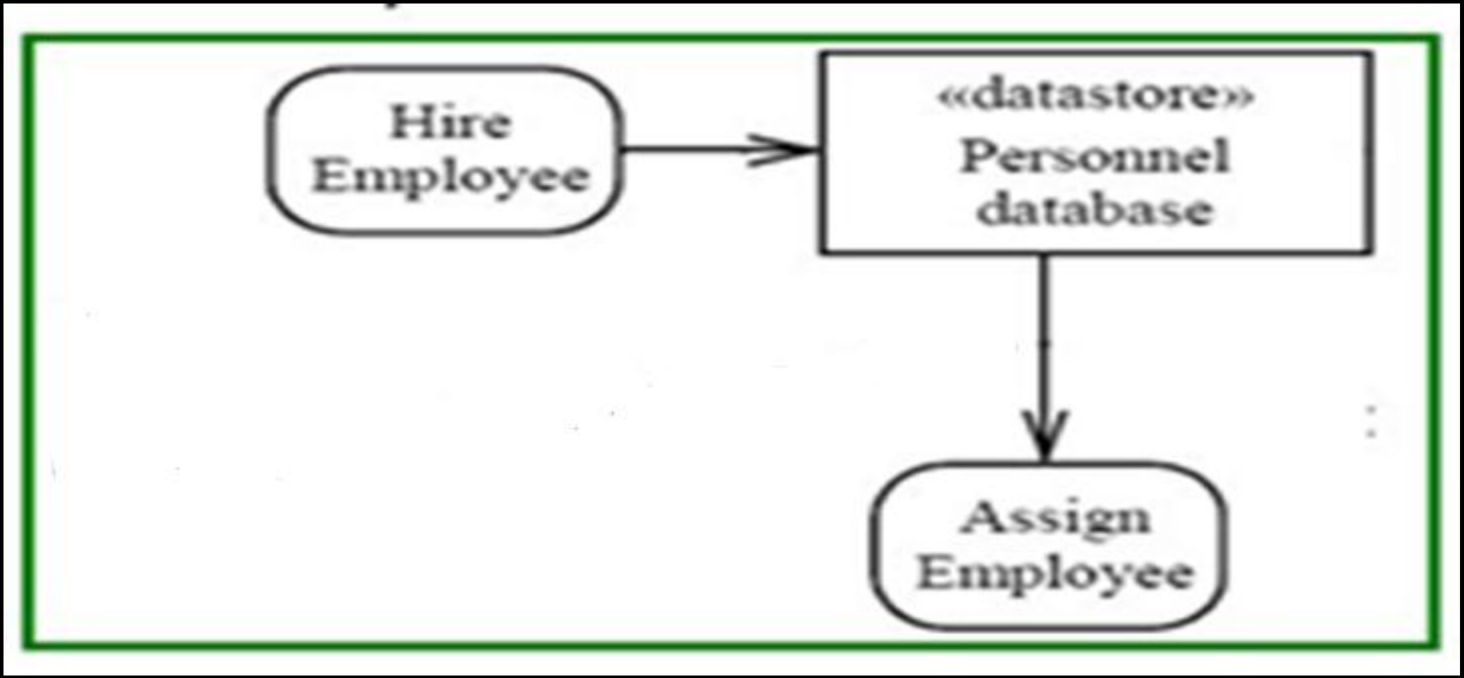
Input and Output Pins-Example



Data Store Node

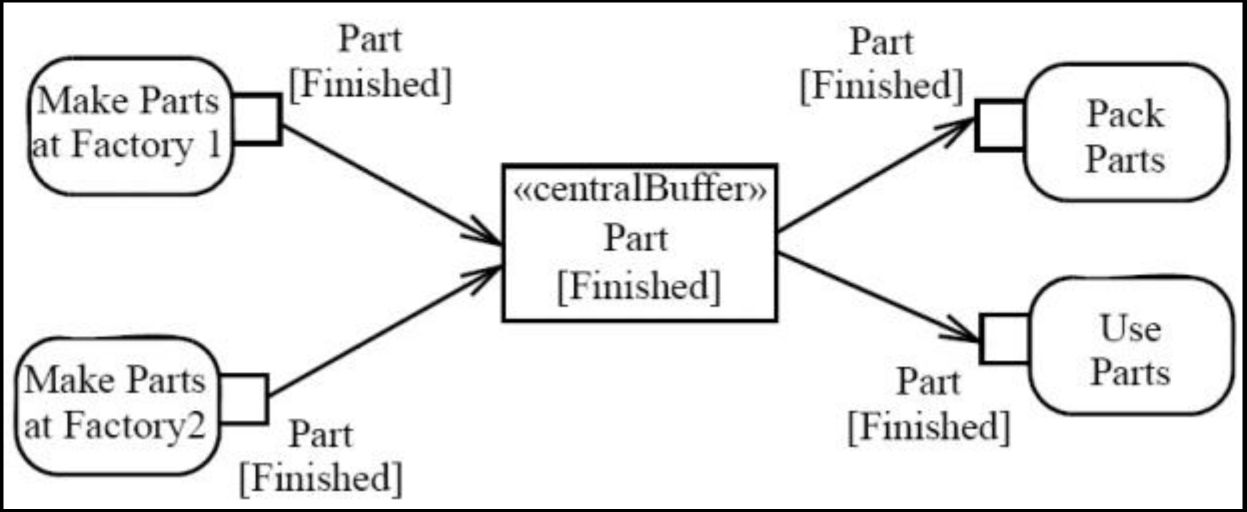


- A data store is shown as an object with the «datastore» keyword.



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

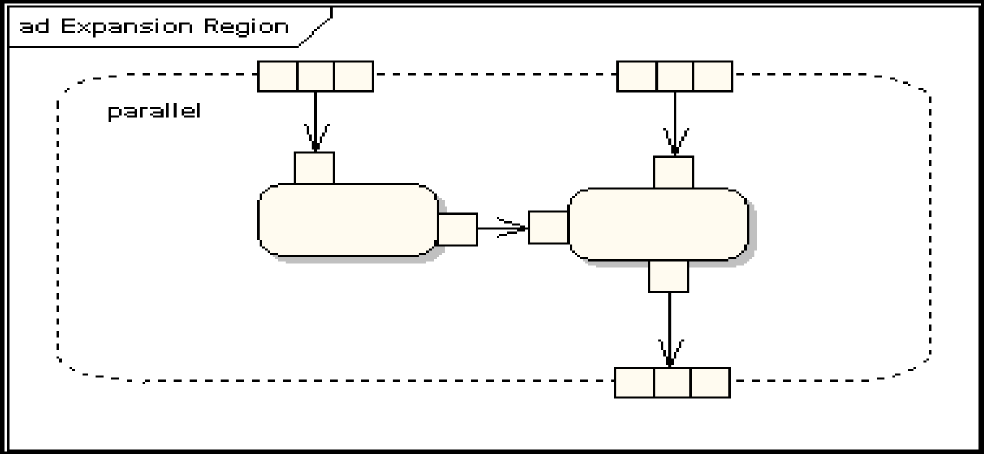
- Expansion Regions
- Interruptible Activity Regions.

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.

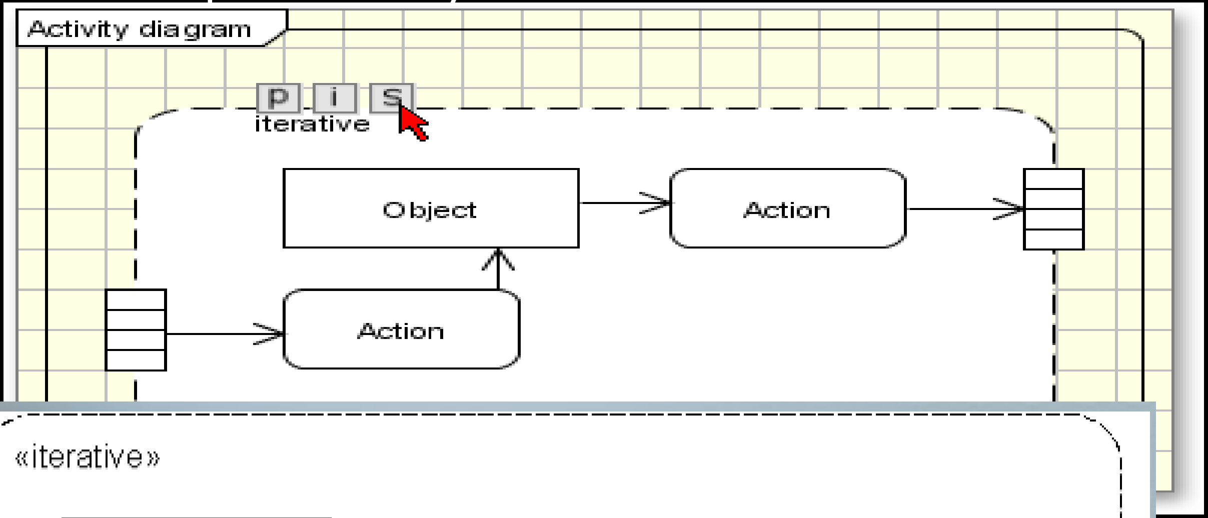
Modes of Multiple Executions

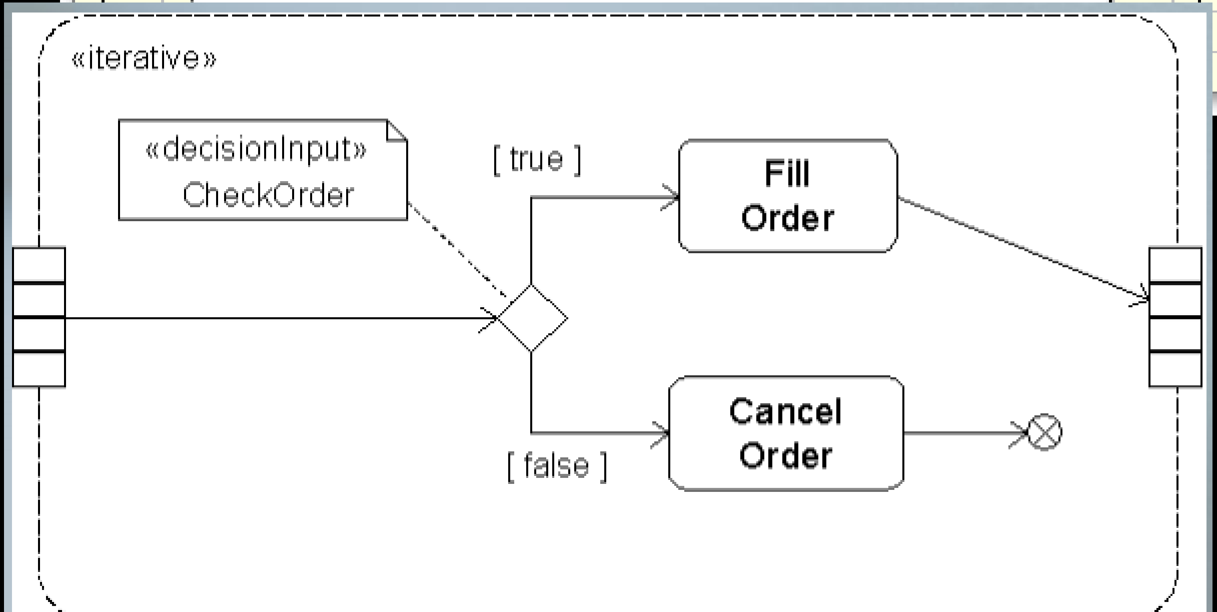
- Expansion Region's multiple executions can be specified as type parallel, iterative, or stream.
- The keyword iterative, parallel, or stream is shown in the top left corner of the region
- **Parallel** reflects that the elements in the incoming collections can be processed at the same time or overlapping.



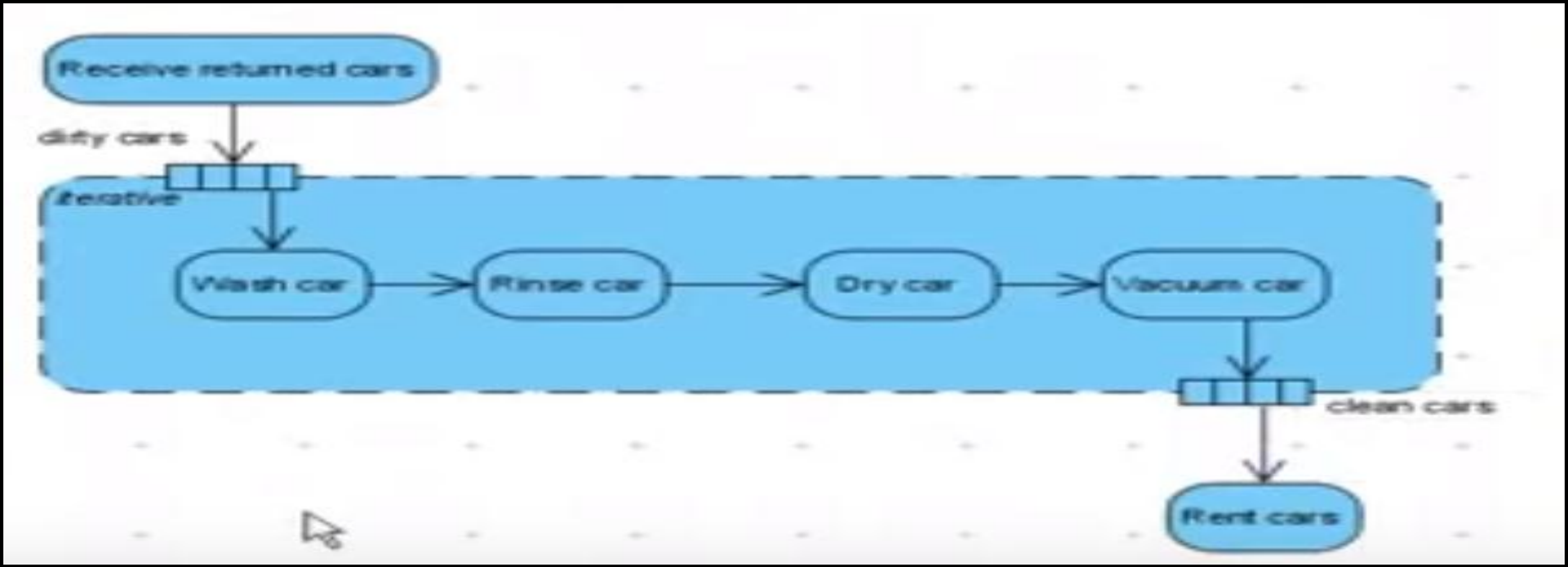
Modes of Multiple Executions

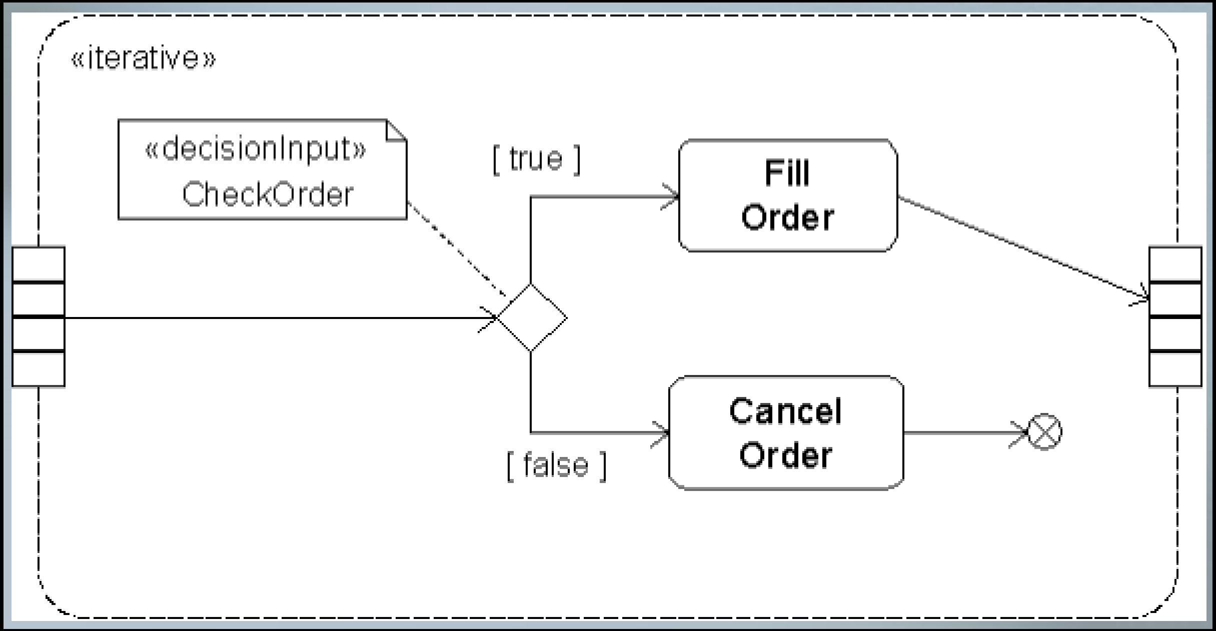
- An **iterative** concurrency type specifies that execution must occur sequentially.





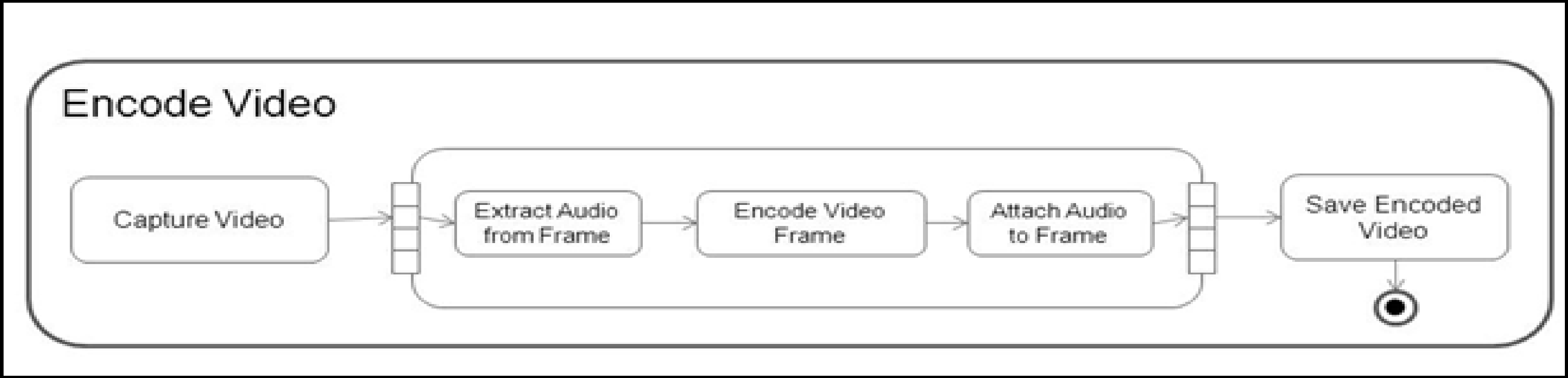
Iterative Examples

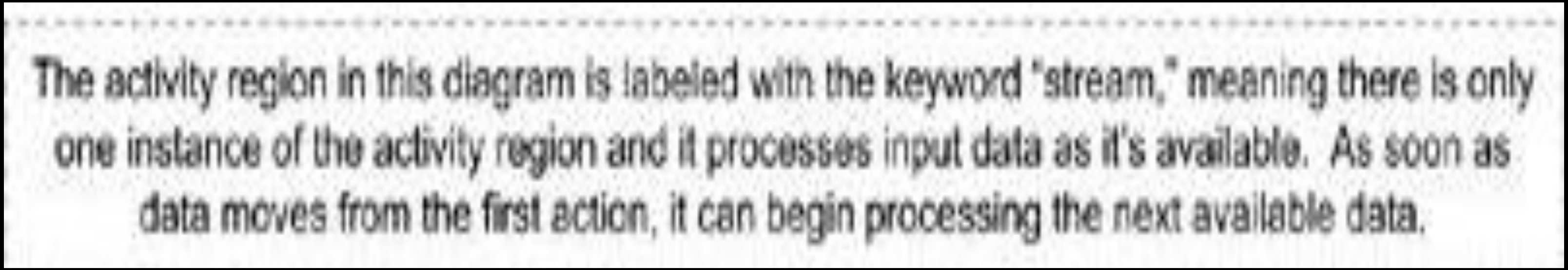




Modes of Multiple Executions

- A **stream-type** Expansion Region indicates that the input and output come in and exit as streams,



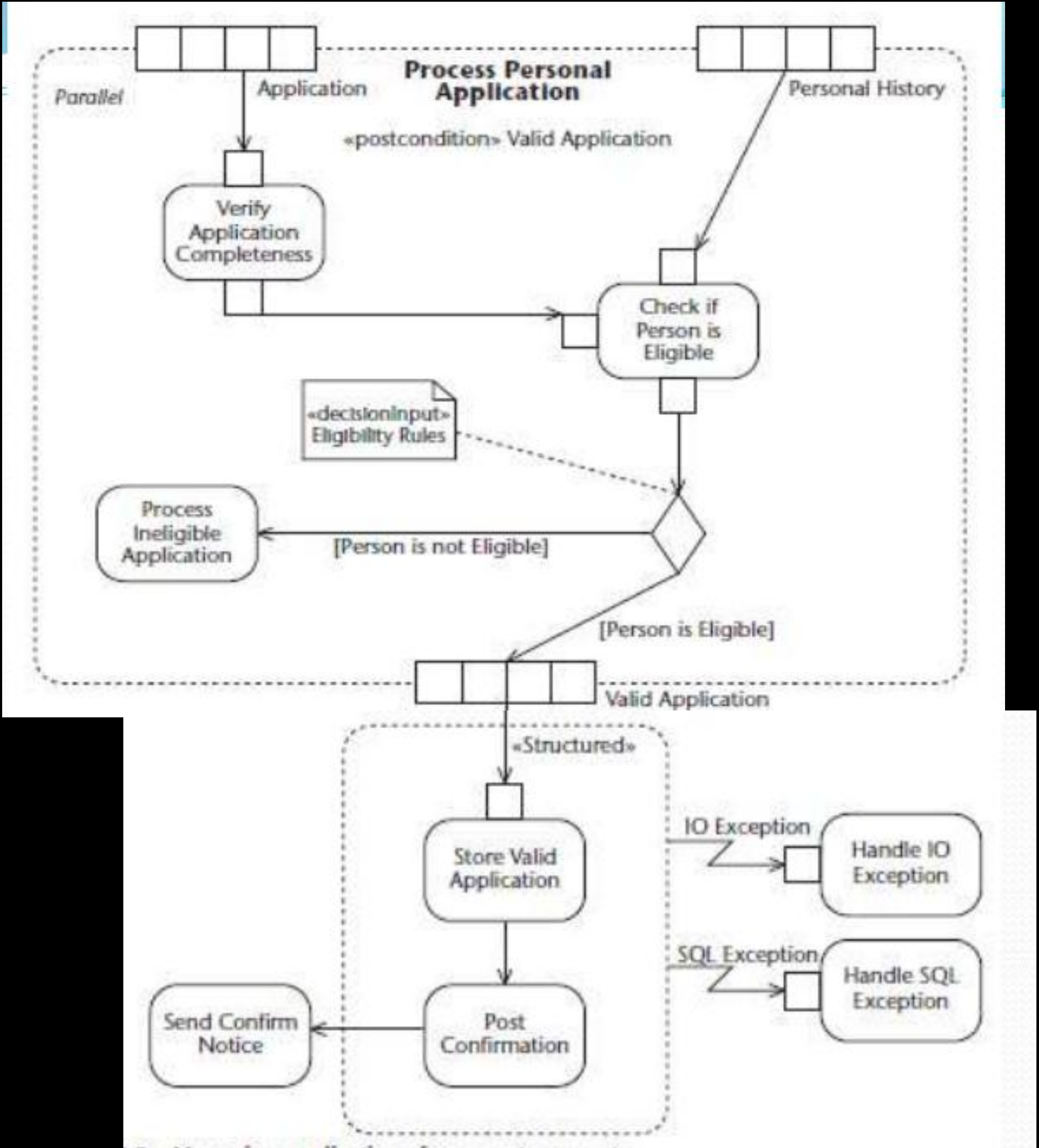


The activity region in this diagram is labeled with the keyword "stream," meaning there is only one instance of the activity region and it processes input data as it's available. As soon as data moves from the first action, it can begin processing the next available data.

Class Activity

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.**



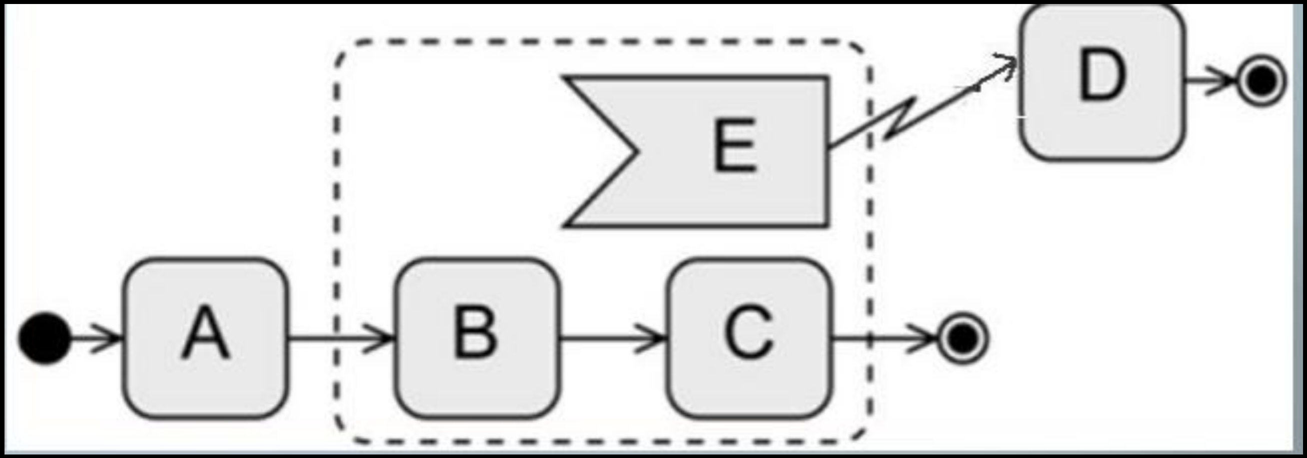
.8 Managing applications for monetary grants.

Interruptible Activity Regions

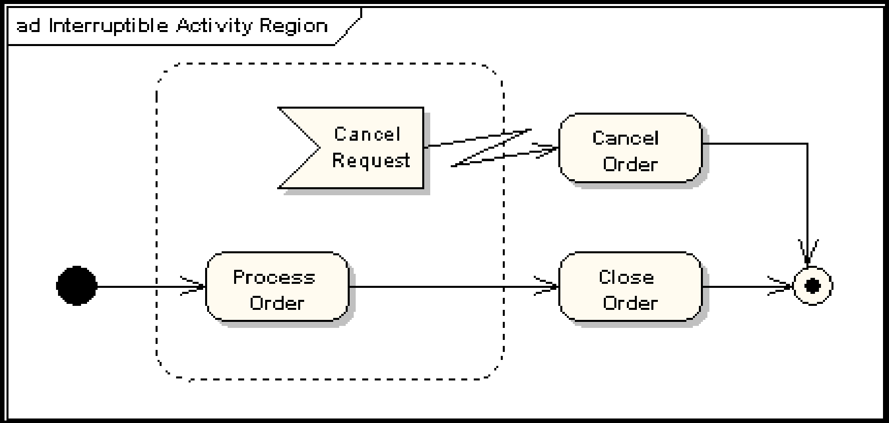
- An Interruptible Activity Region surrounds a group of Activity elements, all affected by certain interrupts in such a way that all tokens passing within the region are terminated when the interruption(s) be raised.
- An interruptible activity region surrounds a group of actions that can be interrupted.

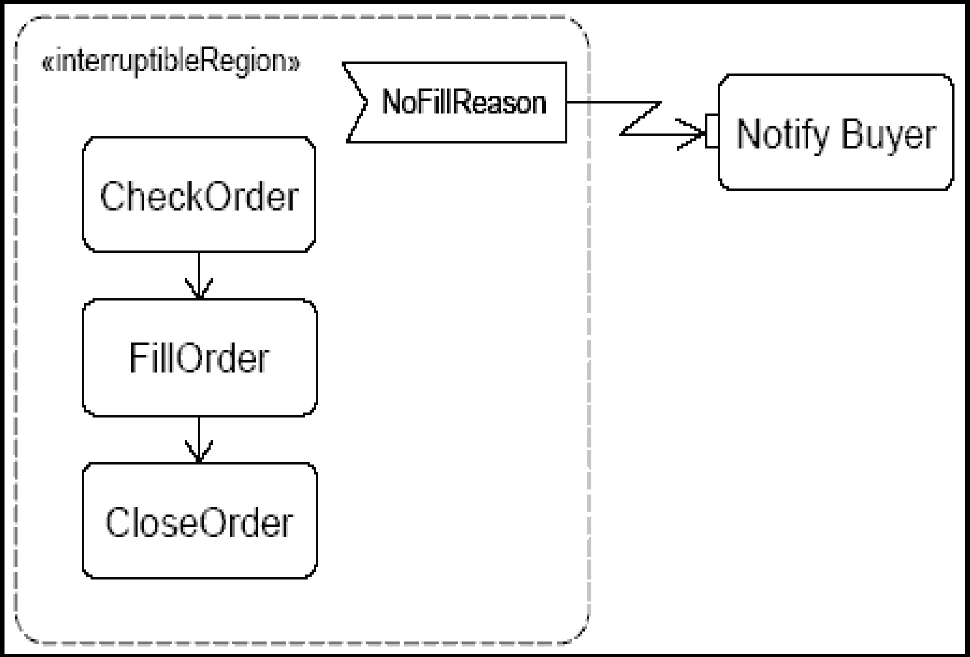
Interruptible Activity Regions

- 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!

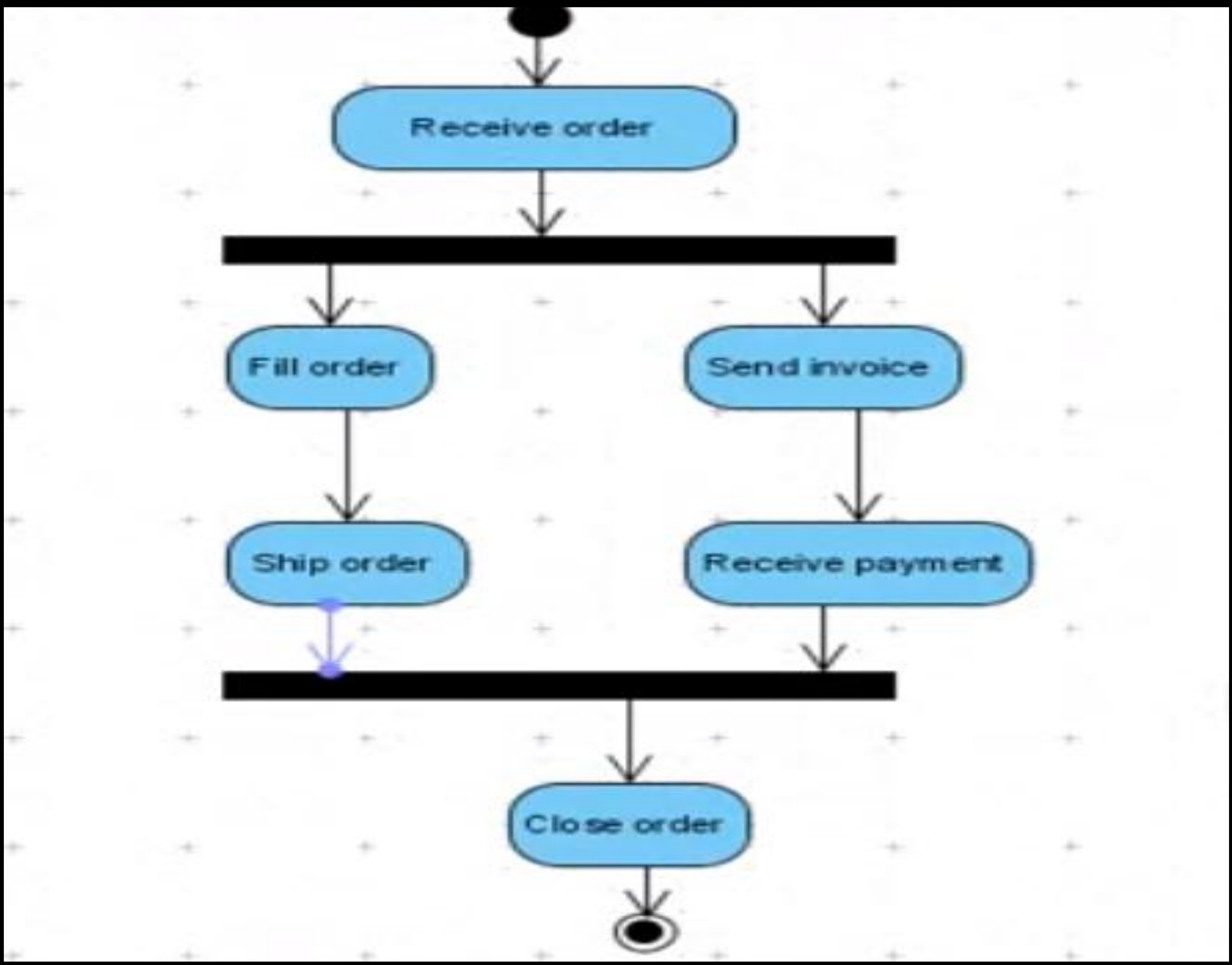


Interruptible Activity Regions Examples

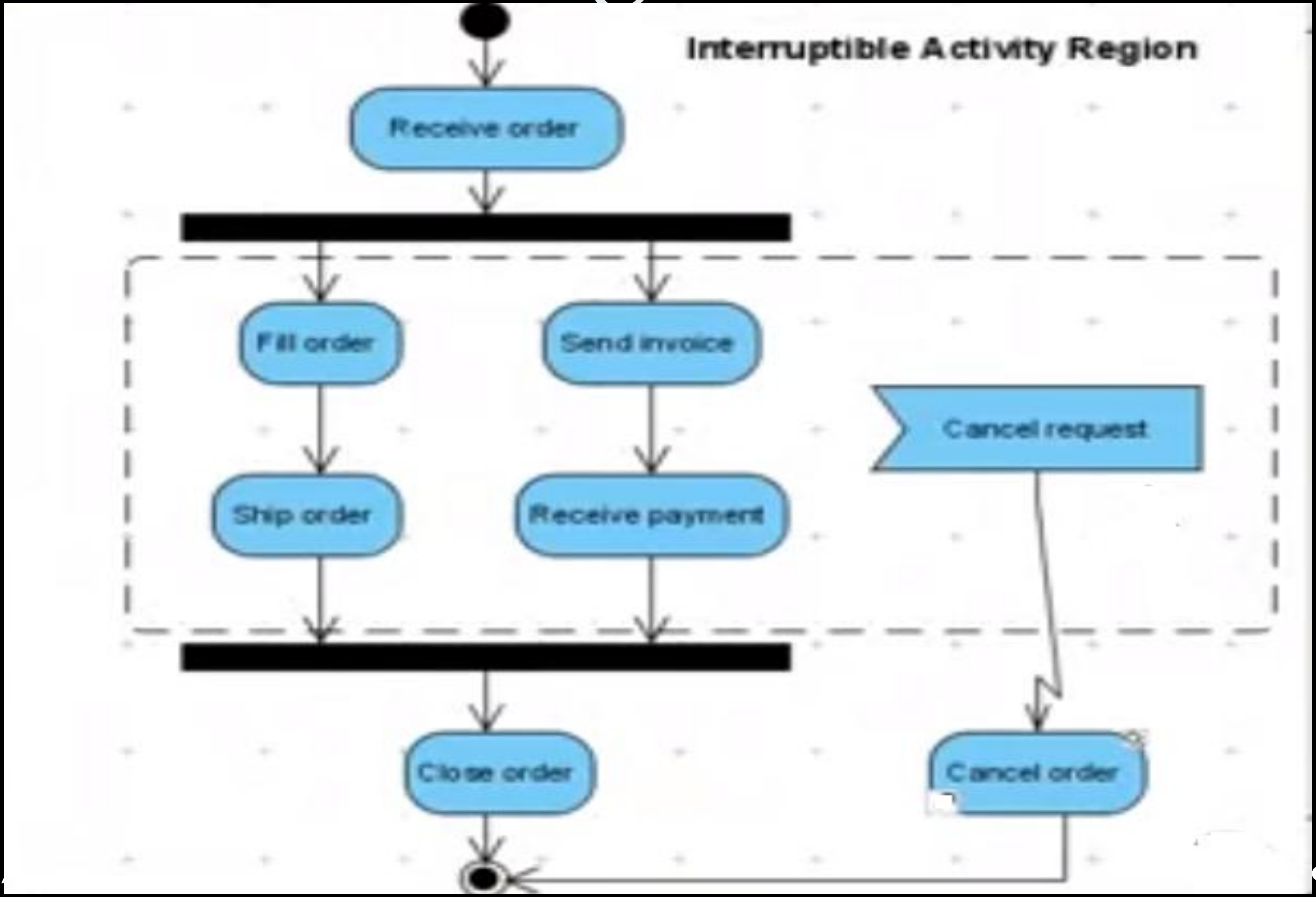




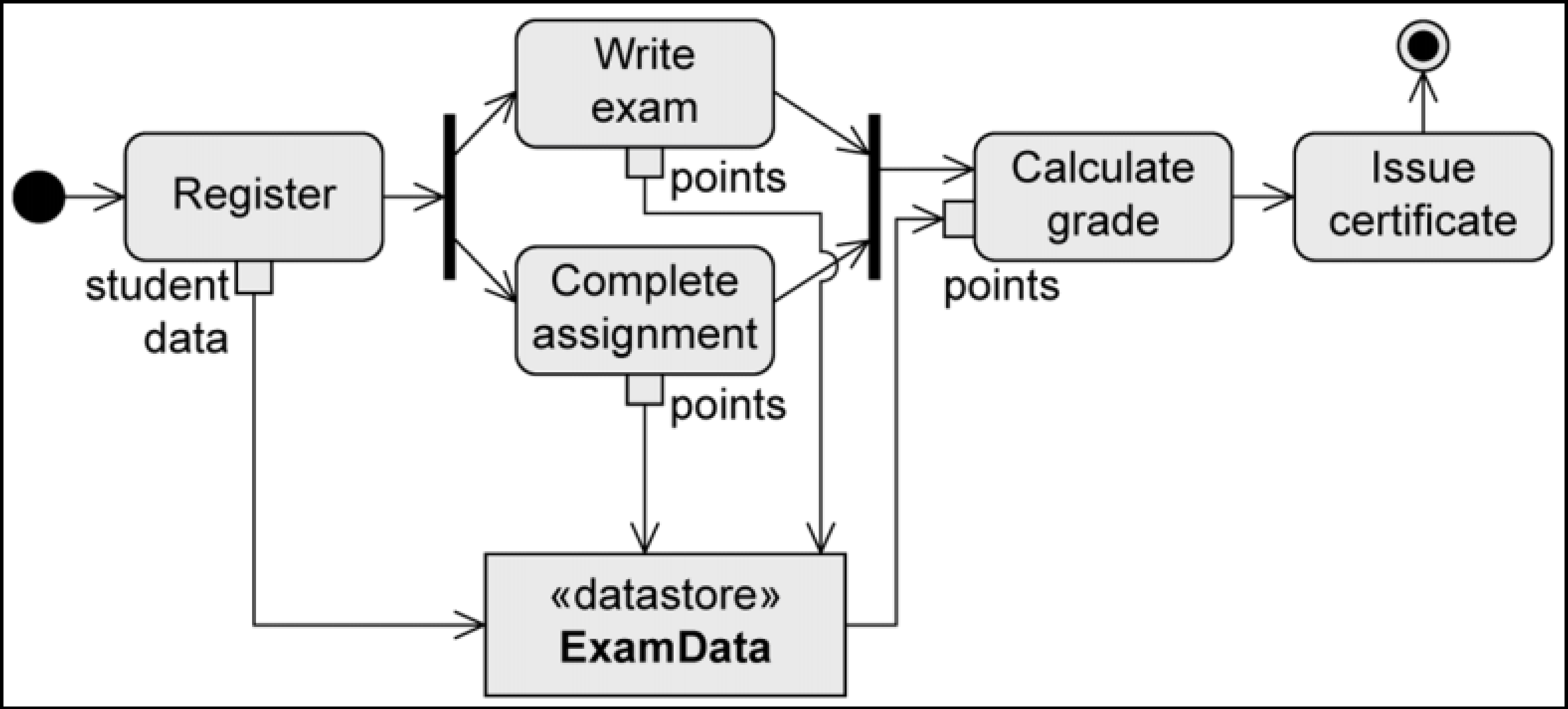
Activity Diagram: Example



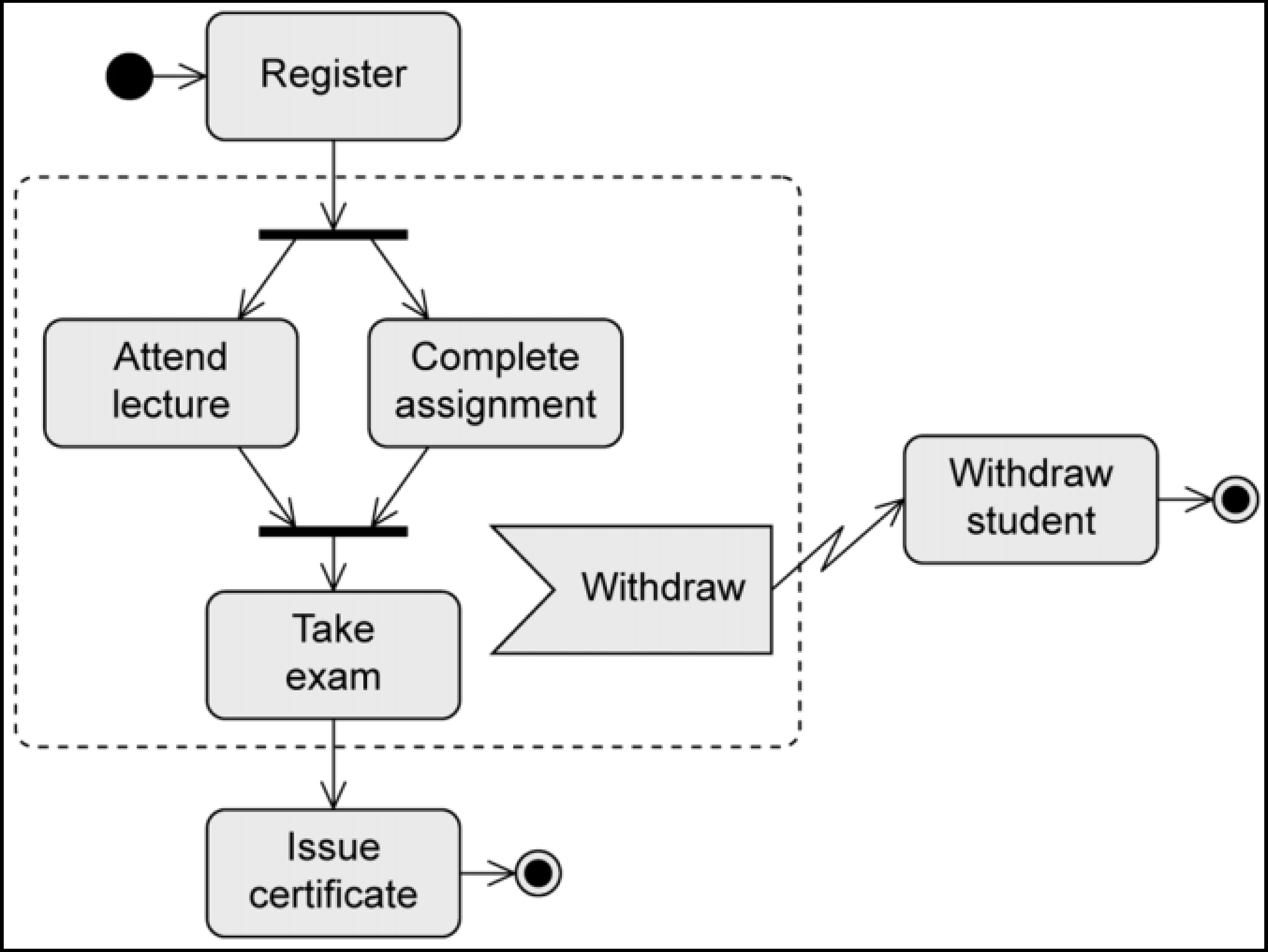
Interruptible Activity Region



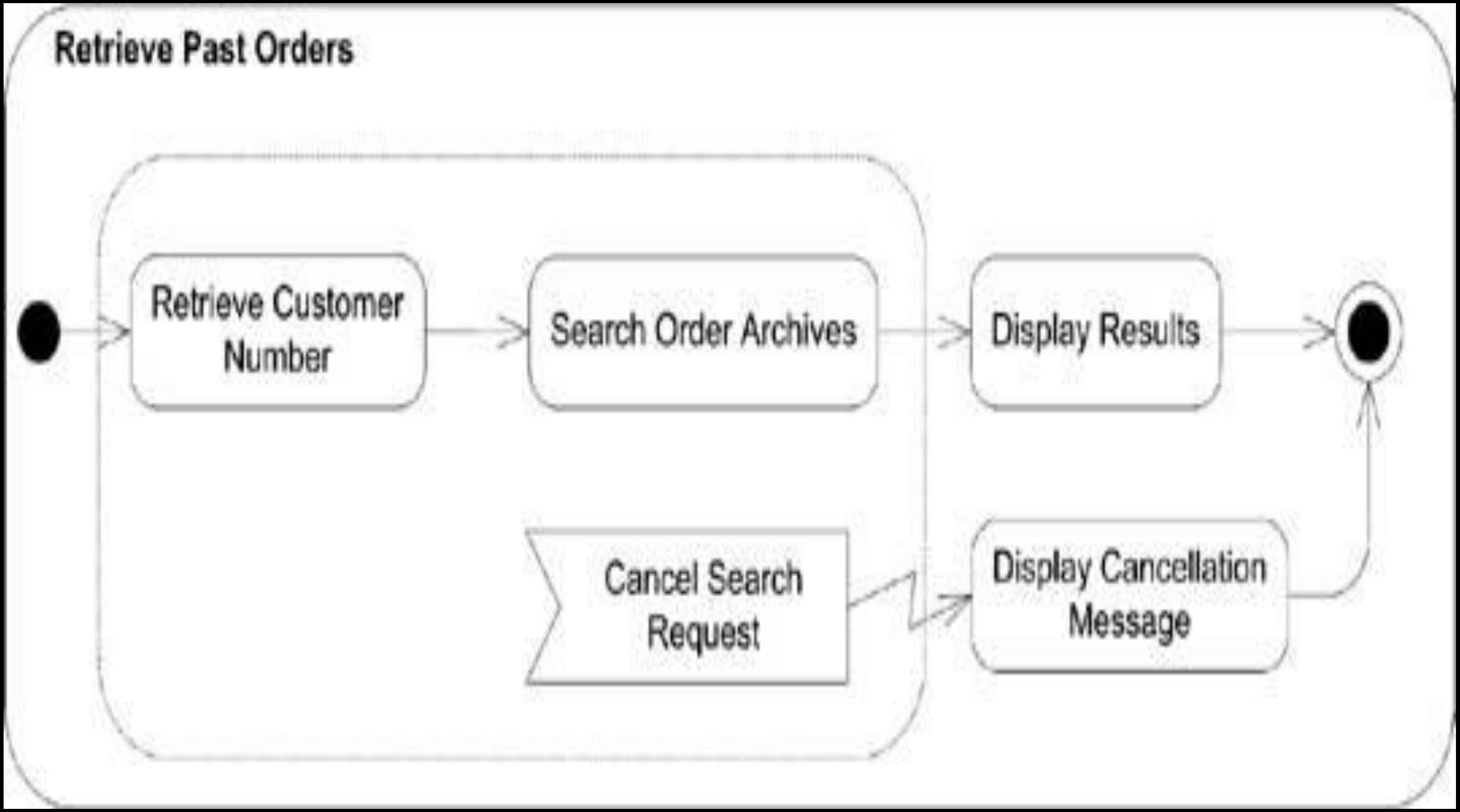
Example: Register and Withdraw



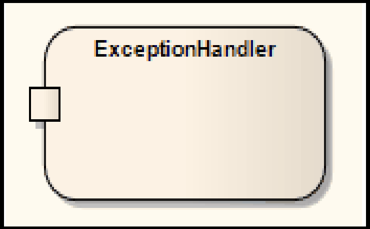
Example: Register and Withdraw



Interruptible Activity Region- Example

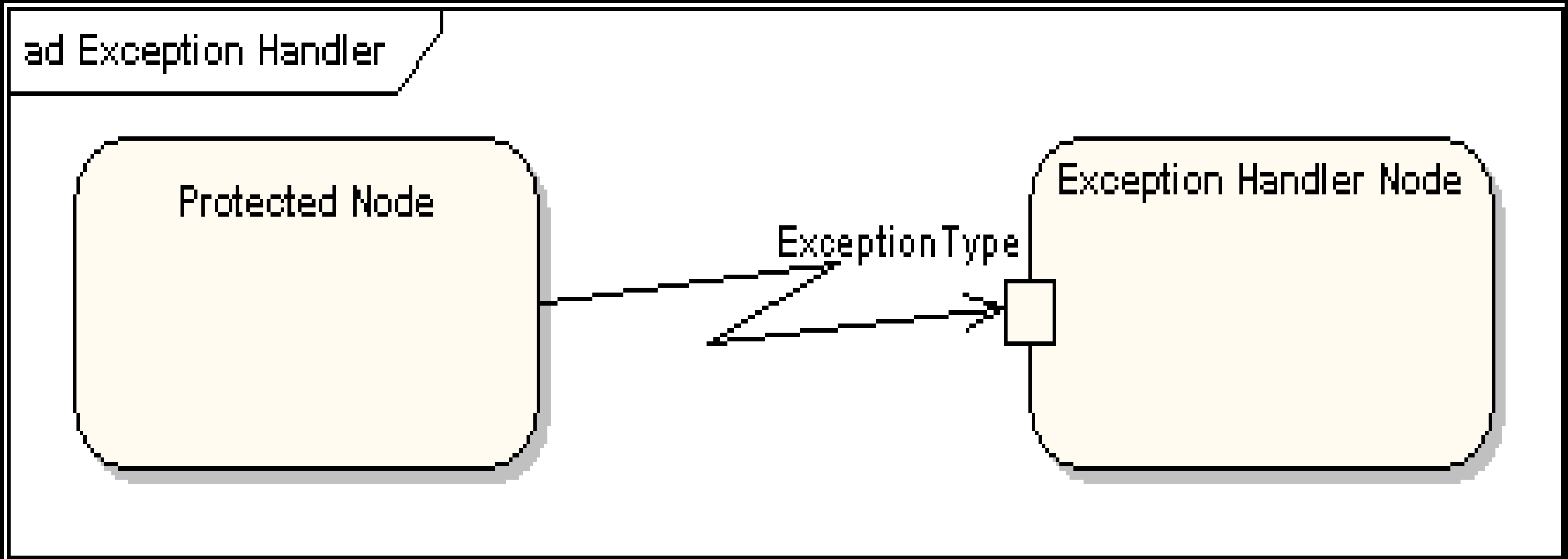


Exception Handler



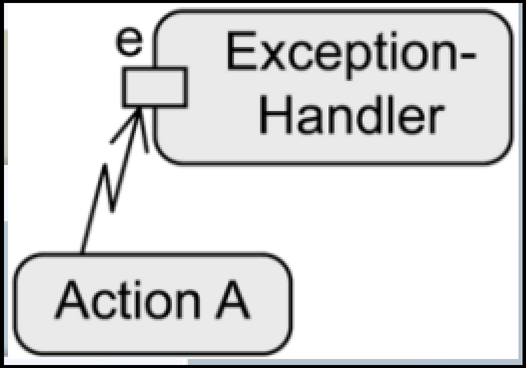
- An exception handler is an element that specifies a body to execute in case the specified exception occurs during the execution of the protected node.

- Exception Handlers can be modeled on activity diagrams as in the example below.



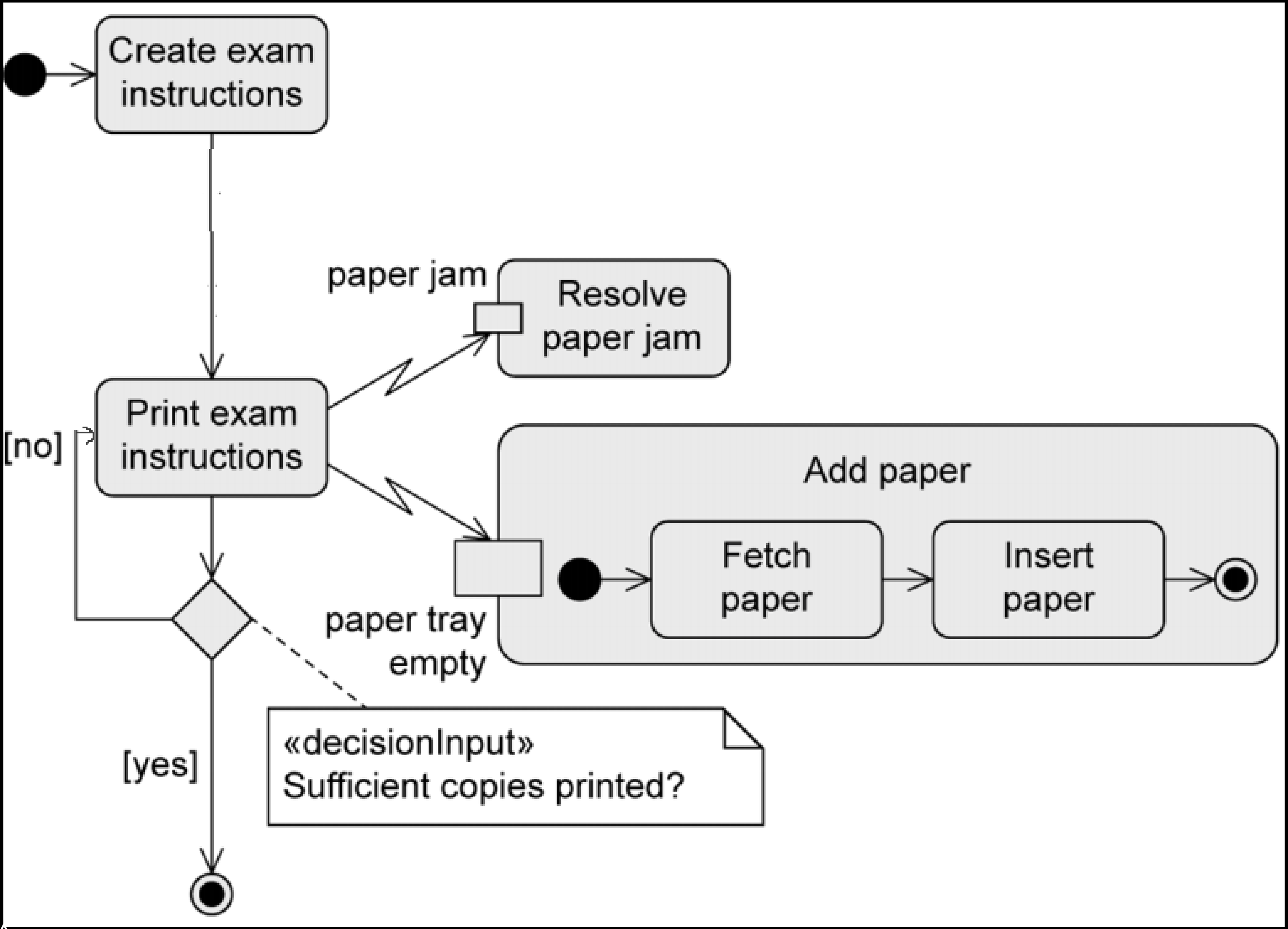
Exception Handling

- 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



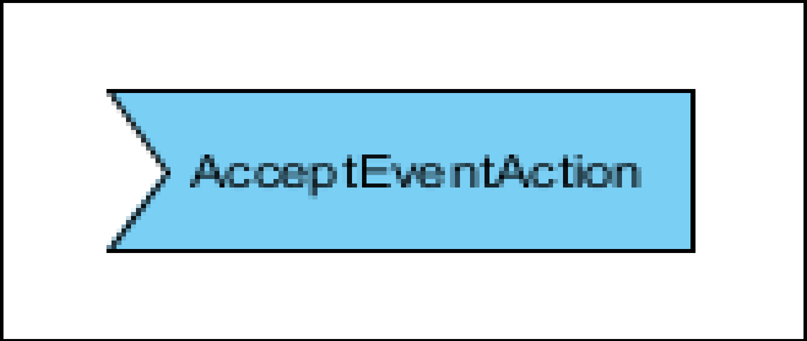
Example: Exception Handler

- 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.

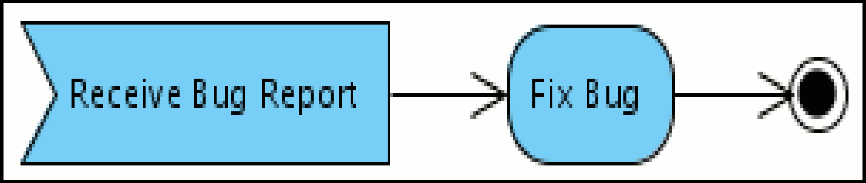

Create exam instructions
paper jam
Resolve paper jam
Print exam instructions
[no]
Add paper
Fetch paper
Insert paper
paper tray empty
«decisionInput»
Sufficient copies printed?
[yes]

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“.



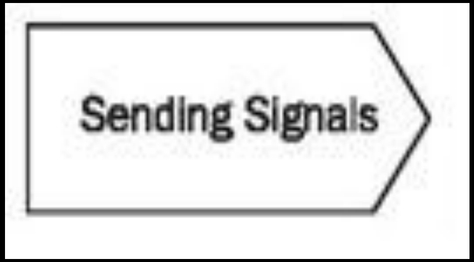
- An activity may also start with a ***receive signal node:***



Send Signal Action

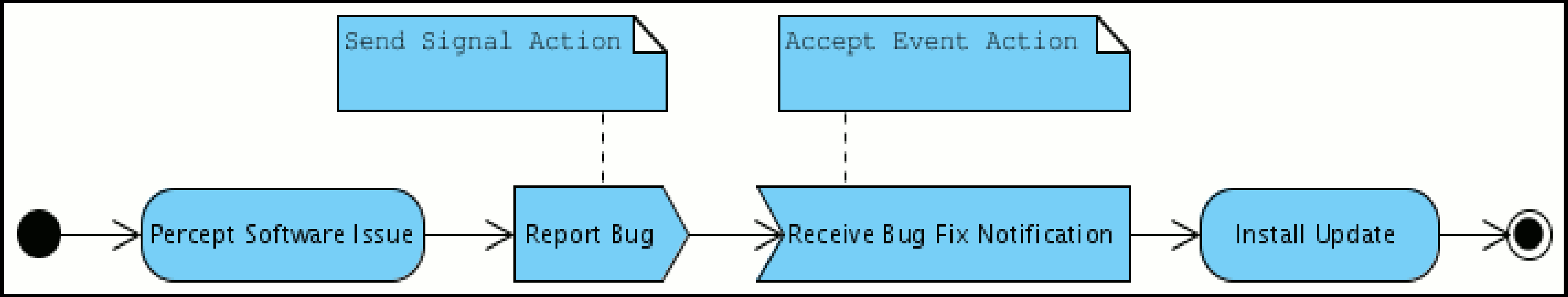
- The sending of signals means that a signal is sent to a receiving activity: The receiving activity accepts the signal with the action accepting a signal and can respond accordingly, meaning, according to the flow that comes from this node in the activity diagram.

- A ***send signal is sent to an external participant.***



Interacting with External Participants

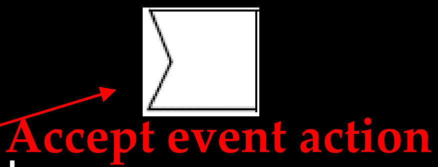
- In the following diagram, both a send and receive signal action are used.

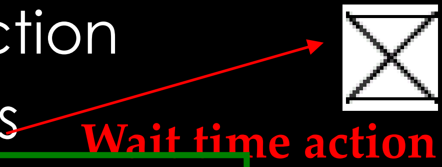


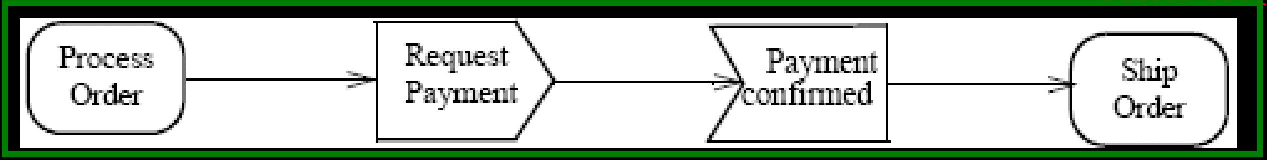
- Note that the activity flow gets interrupted - gets into a wait state - until the bug fix notification is received. (If there was no receive signal action, however, the flow would just continue after executing the send signal action.)

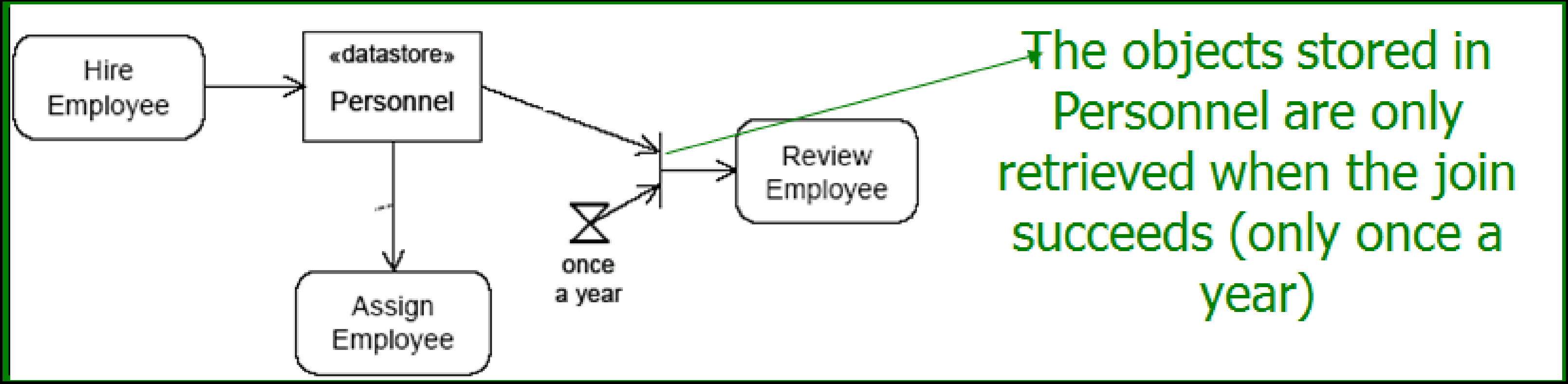
AcceptEventAction

- 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



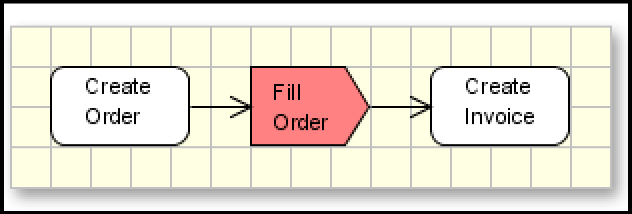


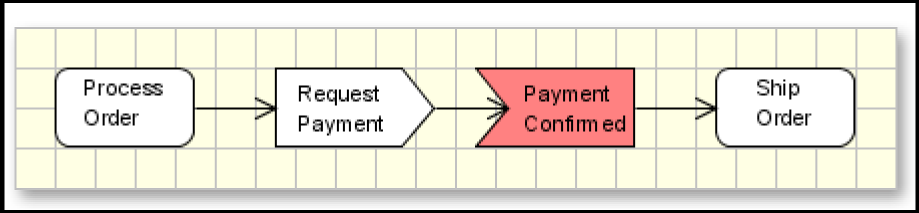




The objects stored in Personnel are only retrieved when the join succeeds (only once a year)

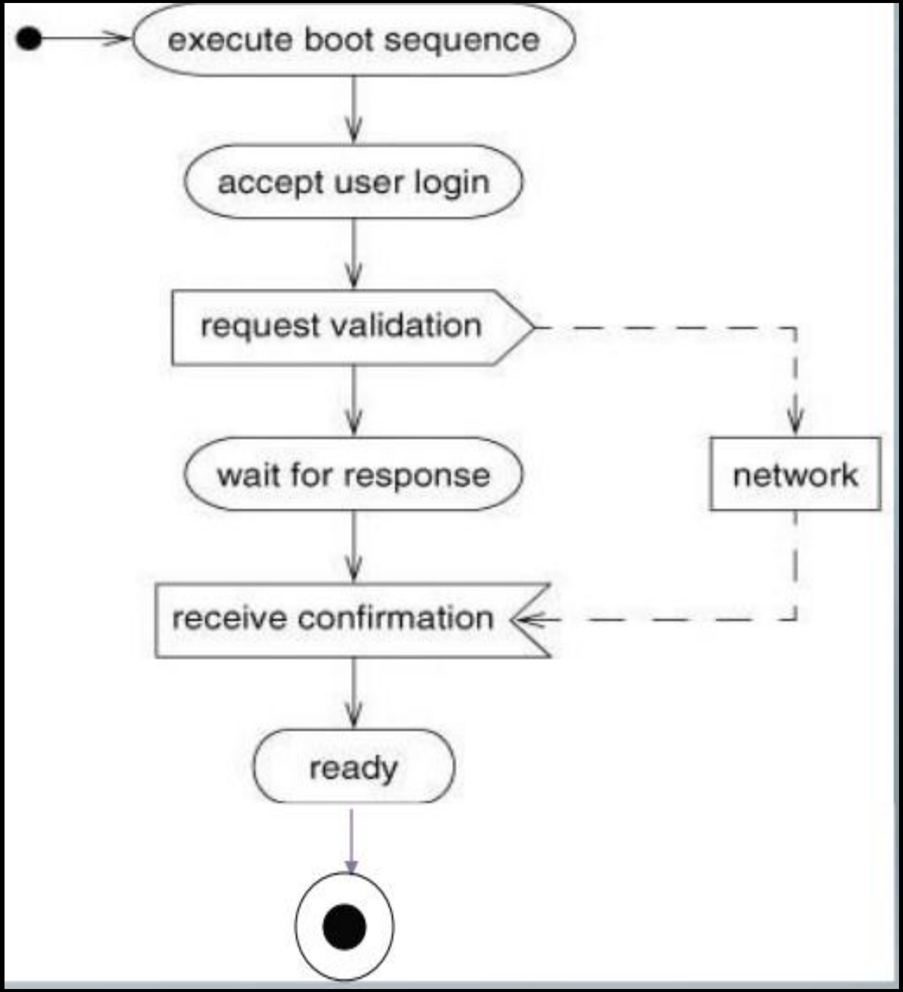
Send Signal &Accept Event Action





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.

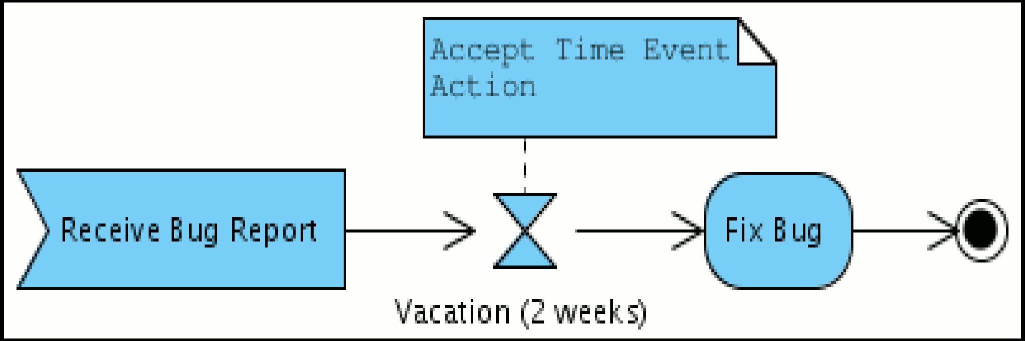


Accept Time Event Action

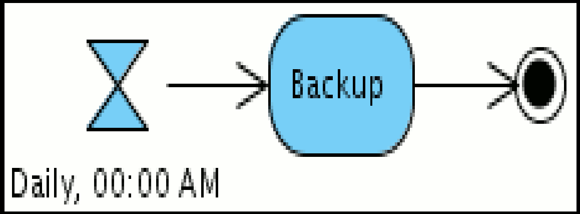
- If the occurrence is a time event occurrence, the result value contains the time at which the occurrence transpired. Such an action is informally called a wait time action.

AcceptEventAction

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

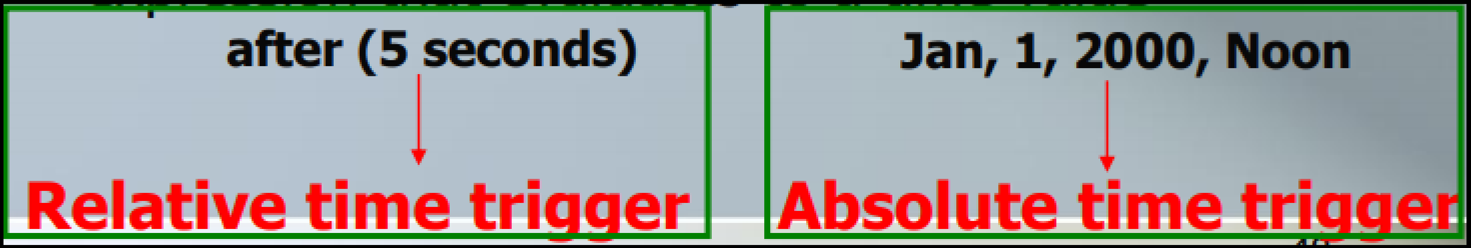


- An activity starting with a time event is launched periodically:

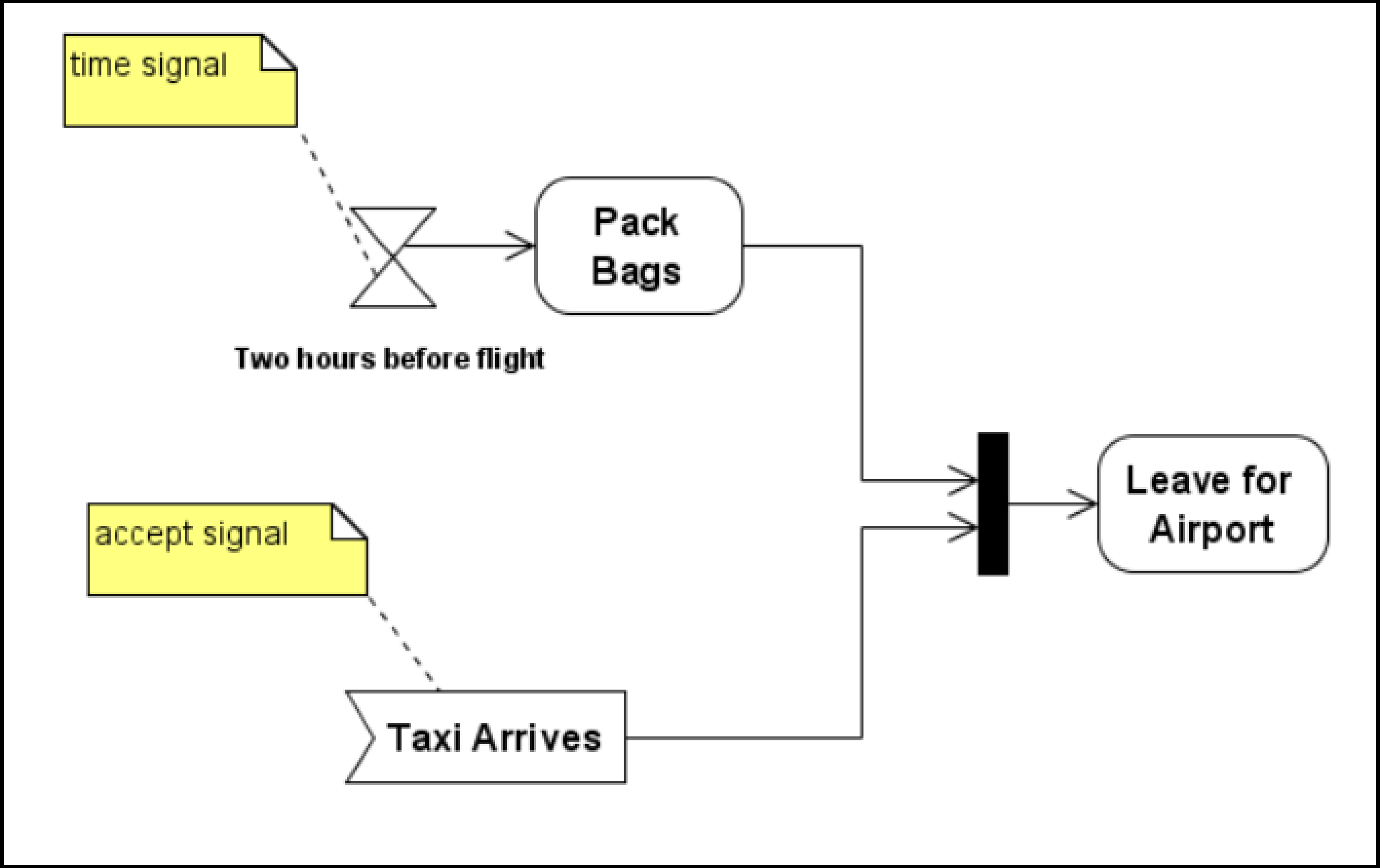


Time Trigger Forms

- 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

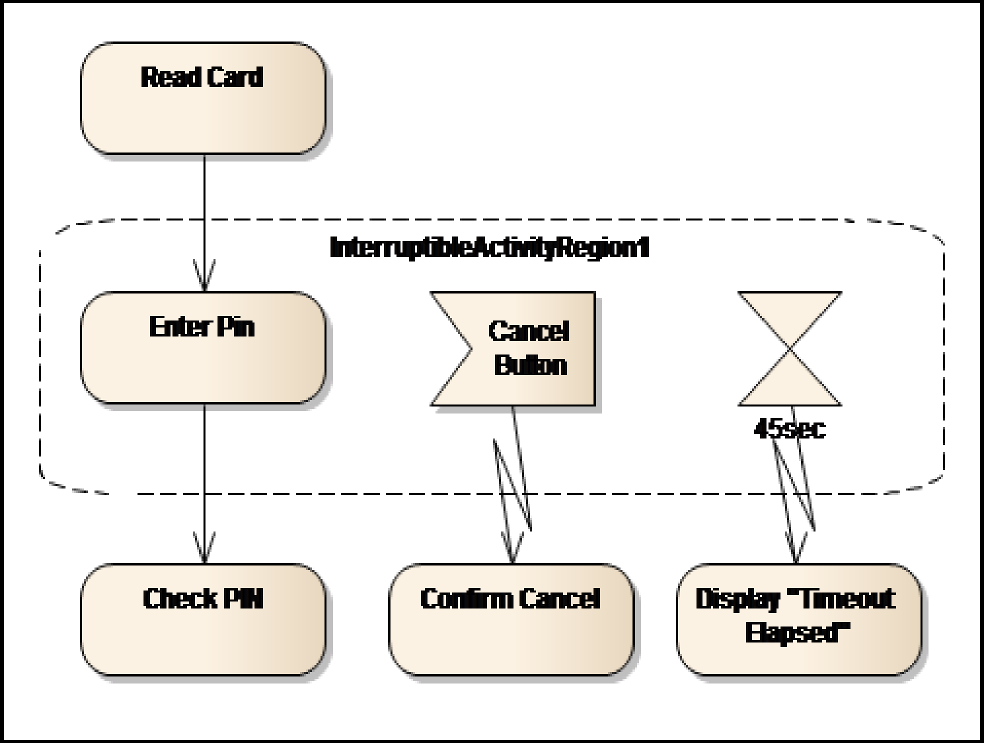


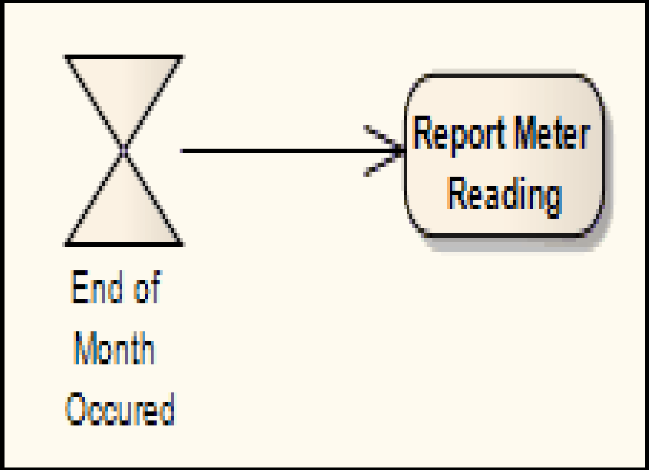
Time Event- Example



Time Event- Examples

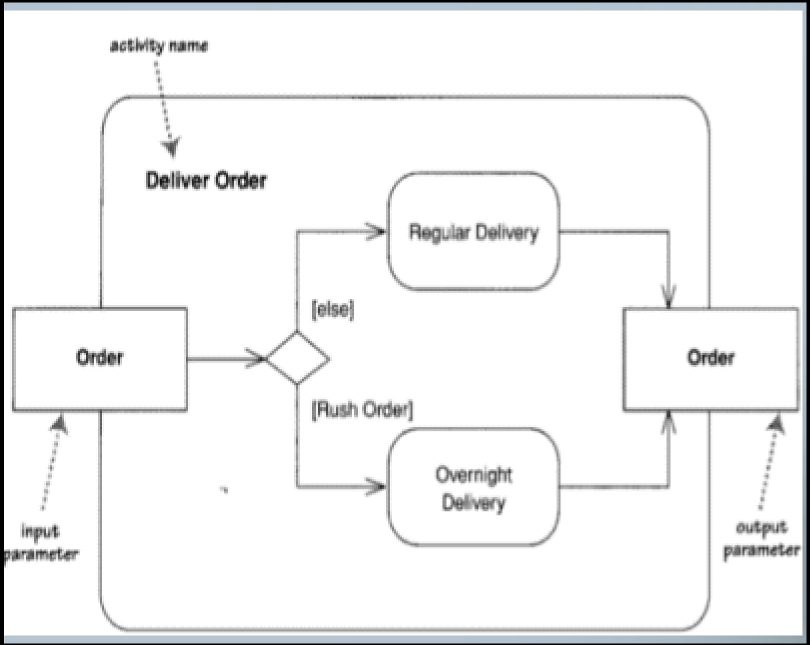
- A typical **example** of a **time event** is triggering reminders after the deadline for payment has passed.
- Another example is timeout.

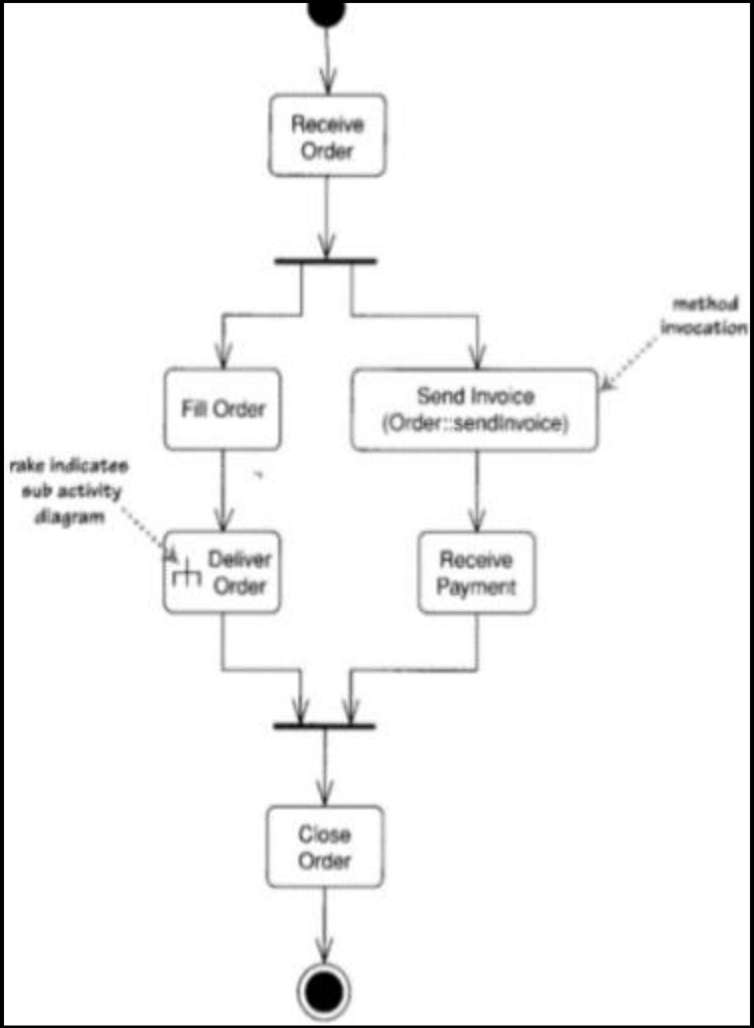




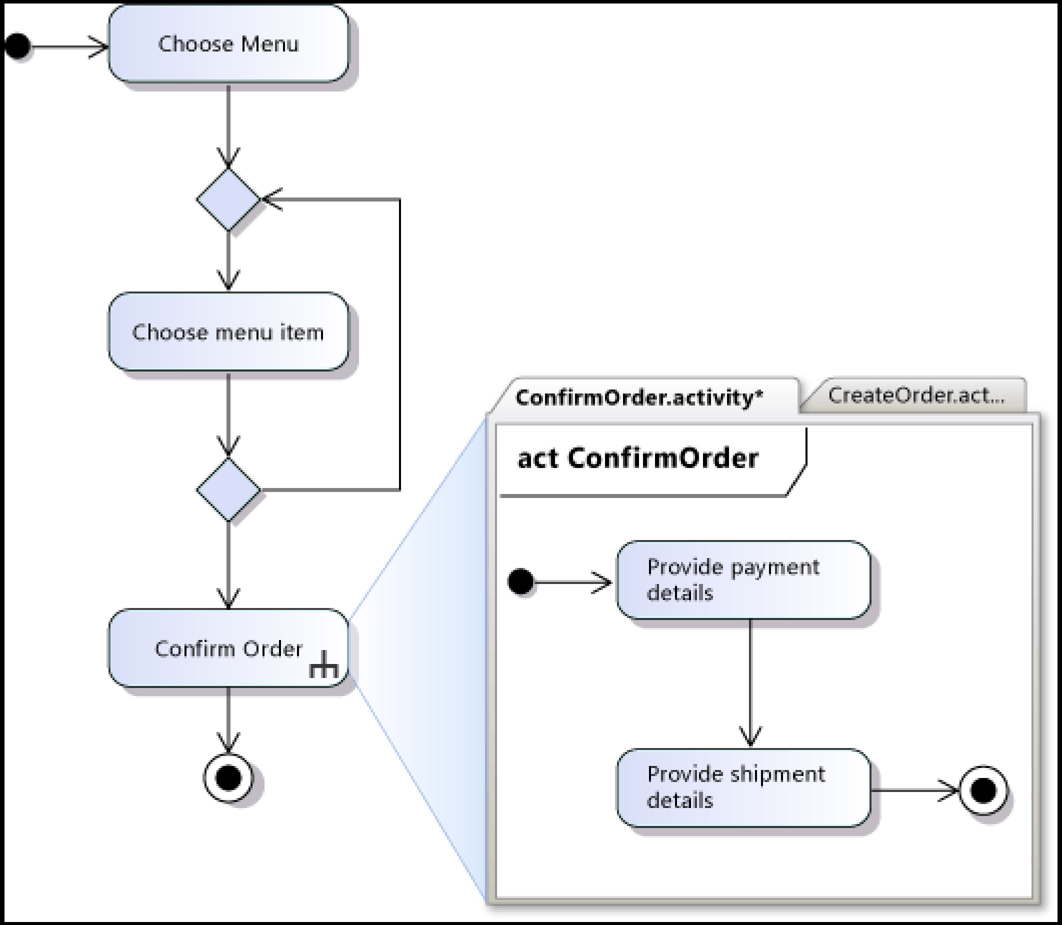
How to Represent Sub-activity Diagram

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





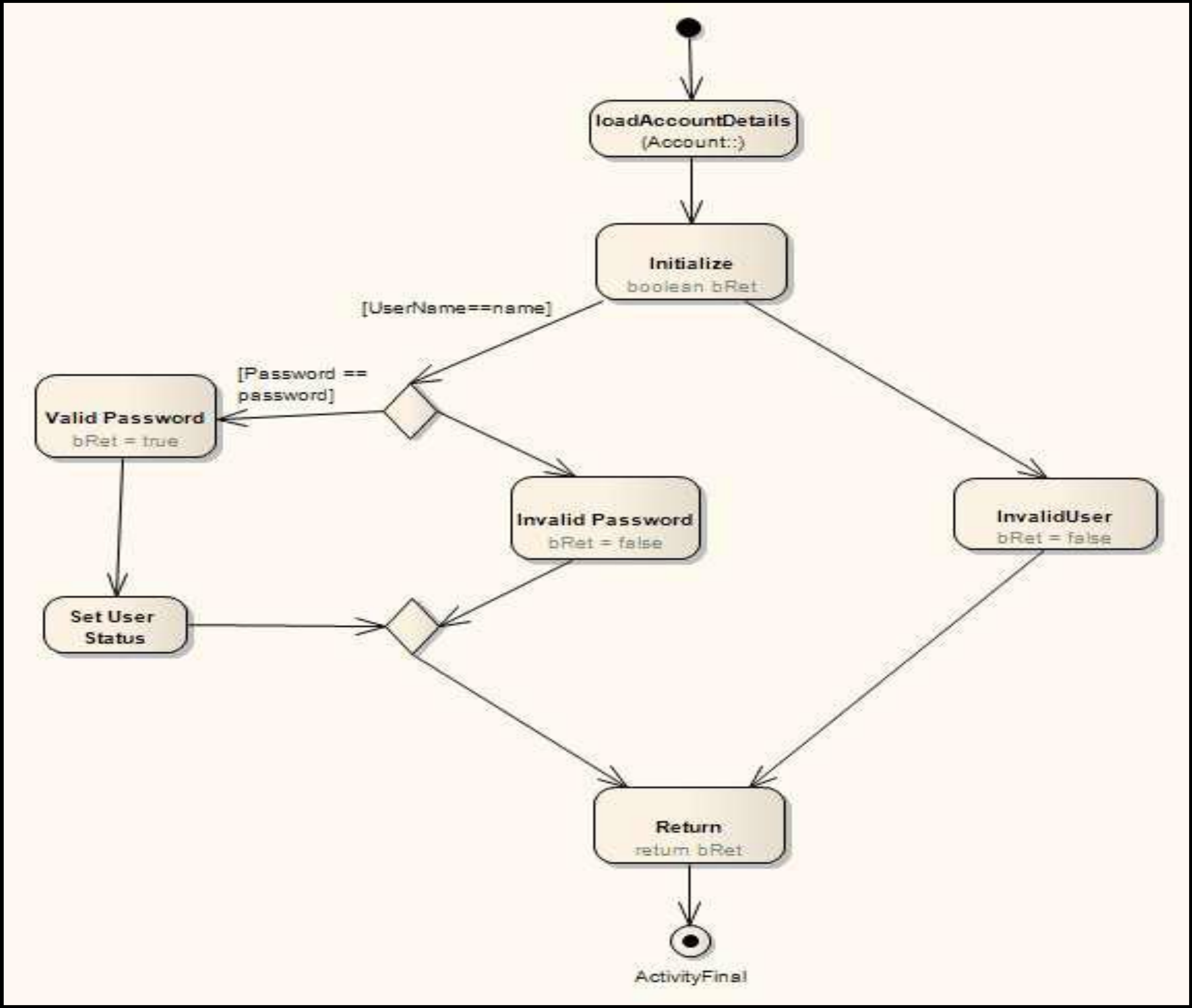
Another Example



Conditional Statements

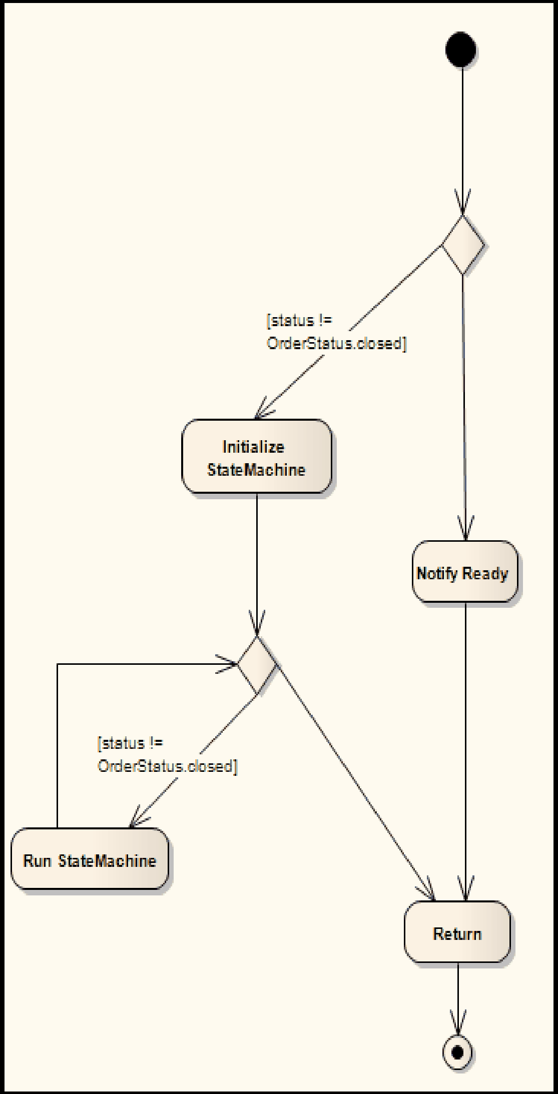
```
public boolean doValidateUser(String Password,String UserName)
{
   loadAccountDetails();
   boolean bRet;
   if (Username==name)
   {
        if (Password == password)
        {
                bRet = true;
                bValidUser = true;
        }
        else
        {
                bRet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

Conditional Statements

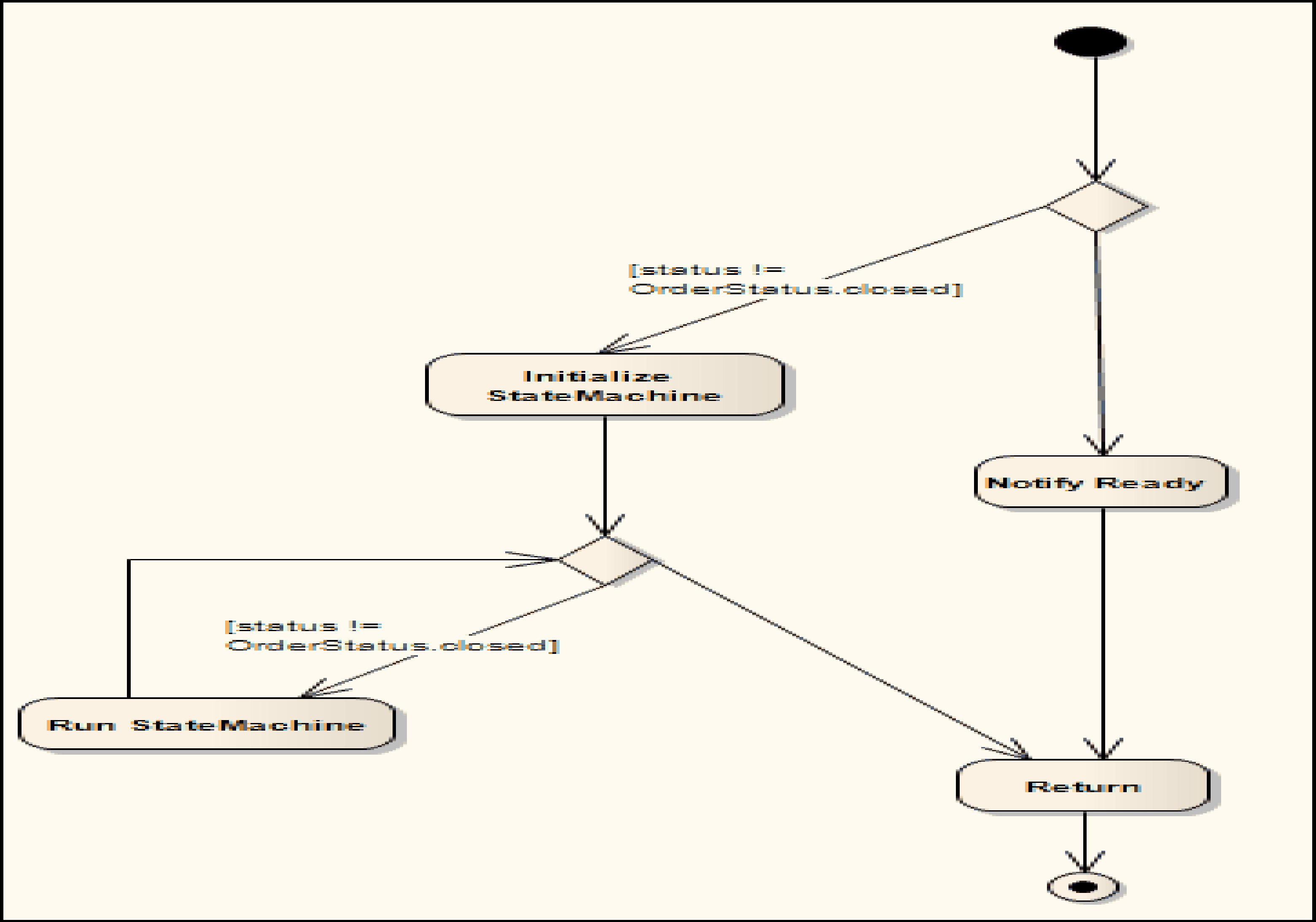


Loops

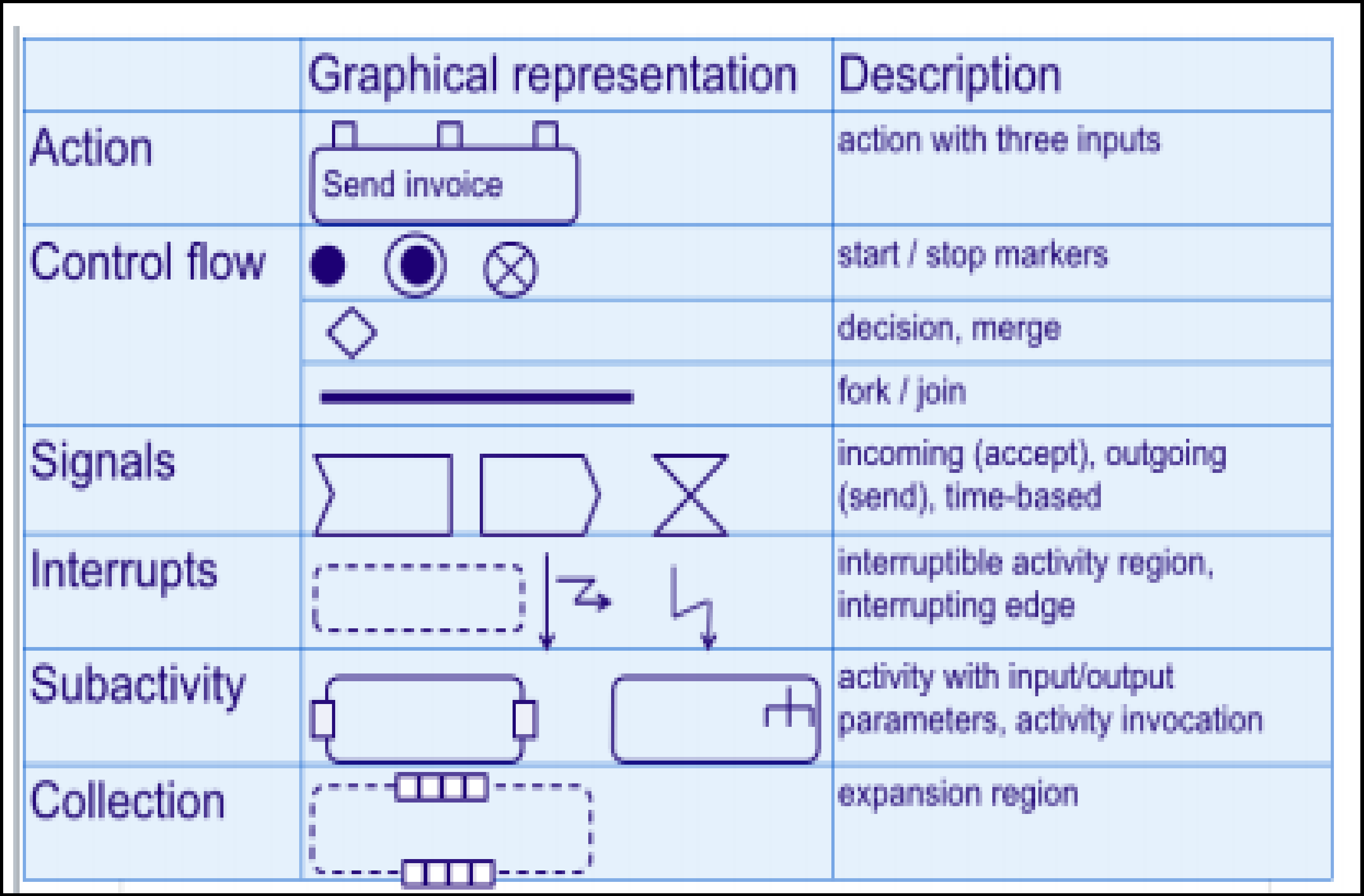
```
public void doCheckForOutstandingOrders()
{
   if (status != closed)
   {
        initializeStateMachine();
        while (status != closed)
        {
              runStateMachine();
        }
   }
   else
   {
        Notify ready;
   }
return;
}
```


[status != OrderStatus.closed]
Initialize StateMachine
Notify Ready
[status != OrderStatus.closed]
Run StateMachine
Return

Loops


[status != OrderStatus.closed]
Initialize StateMachine
Notify Ready
[status != OrderStatus.closed]
Run StateMachine
Return

Some more notations

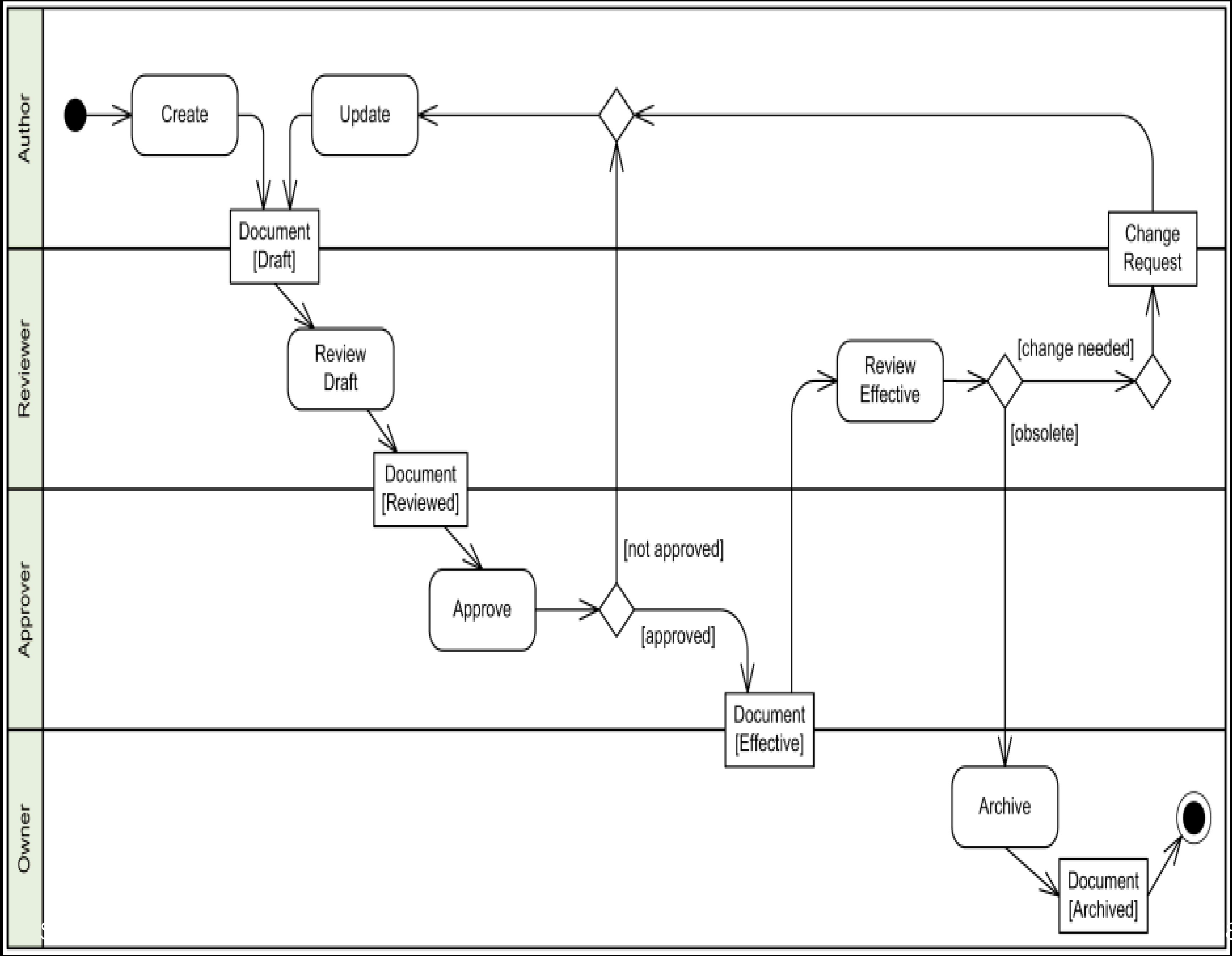


	Graphical representation	Description
Action	Send invoice	action with three inputs
Control flow	● ◉ ⊗	start / stop markers
	◇	decision, merge
		fork / join
Signals		incoming (accept), outgoing (send), time-based
Interrupts		interruptible activity region, interrupting edge
Subactivity		activity with input/output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

Class Activity
Draw Diagrams

Class Activity
Draw Diagrams

- Document Management Process: A document goes through different states or stages - it is created, reviewed, updated, approved, and at some point archived. Different roles participating in this process are Author, Reviewer, Approver, and Owner.


Author
Reviewer
Approver
Owner
Create
Update
Document [Draft]
Review Draft
Document [Reviewed]
Approve
[not approved]
[approved]
Document [Effective]
Review Effective
[change needed]
[obsolete]
Change Request
Archive
Document [Archived]

Exercise: Ticket Vending Machine

- Activity is started by Commuter actor who needs to buy a ticket. Ticket vending machine will request trip information from Commuter. This information will include number and type of tickets, e.g. whether it is a monthly pass, one way or round ticket, route number, destination or zone number, etc.
- Based on the provided trip info ticket vending machine will calculate payment due and request payment options. Those options include payment by cash, or by credit or debit card. If payment by card was selected by Commuter, another actor, Bank will participate in the activity by authorizing the payment.

That is all