Activity Diagrams

Topic # 9

Chapter 28 – Craig Larman

Activity

- Perspectives:
  - 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
- Arrangements of Activity:
  - Sequential – one activity is followed by another
  - Parallel – two or more sets of activities are performed concurrently, and order is irrelevant

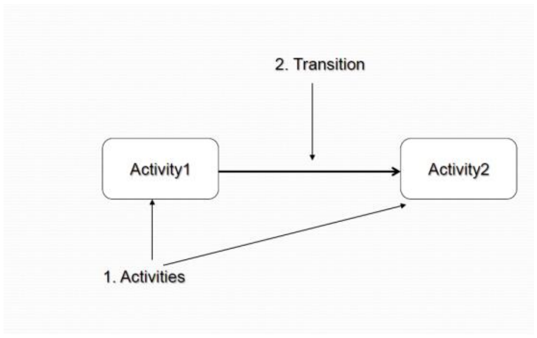
Activity Diagram Definition

- **Activity diagram**: A diagram that can be used to graphically depict the flow of a business process, the steps of a use case, or the logic of an object behavior (method).
- Activity diagrams represent the dynamic (behavioral) view of the system.
- Used to represent flow across use cases or within a use case.
- UML activity diagrams are the object oriented equivalent of flow chart & DFDs in function-oriented design approach.
- Activity diagrams help to describe how activities are coordinated.
- Records the dependencies between activities, such as which things happen in parallel and what must be finished before something else can start.

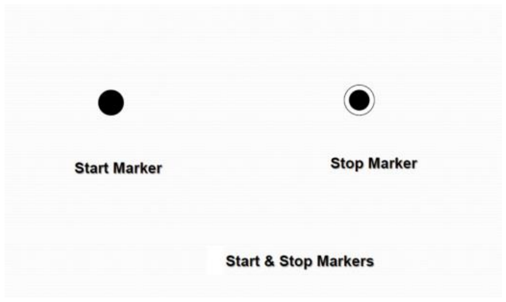
Activity Diagram

- Activity Diagrams are like Flow Charts, but Flow Charts are usually limited to sequential activities while Activity Diagrams can show parallel activities as well
  - An activity diagram is essentially a flowchart, showing the flow of control from activity to activity.
  - 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

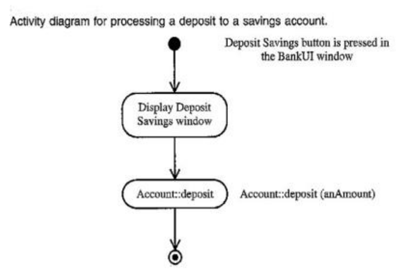
Notation



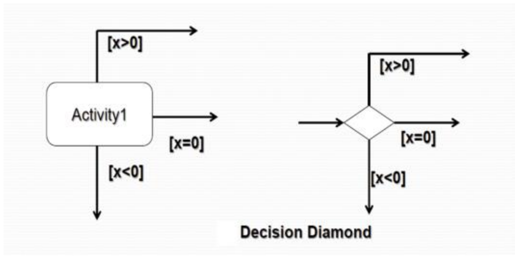
Notation



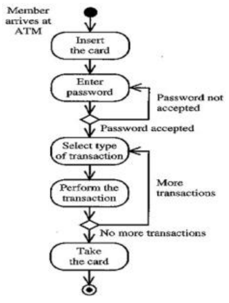
Activity Diagram: Example (activity and transition)


Activity diagram for processing a deposit to a savings account.
Deposit Savings button is pressed in the BankUI window
Display Deposit Savings window
Account::deposit
Account::deposit (anAmount)

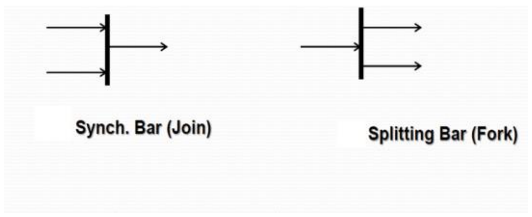
Notation



Activity Diagram: Example (Decision)

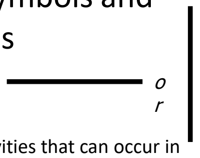


Notation



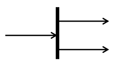
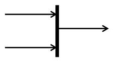
Activity Diagram Symbols and Notations

- Synchronisation bar:



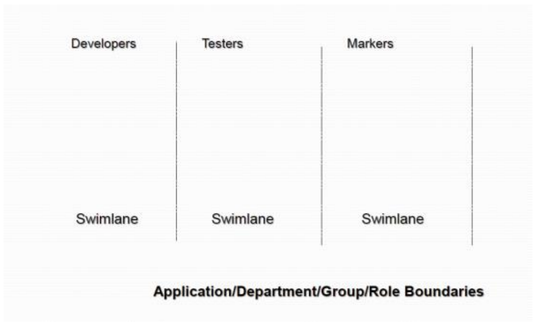
- All triggers from this attach to activities that can occur in parallel, with no specific sequence, or concurrently.
- The next synchronisation bar closes the concurrency.
- Use a synchronization bar to specify the forking and **joining** of parallel flows of control
- A synchronization bar is rendered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

Fork and Join

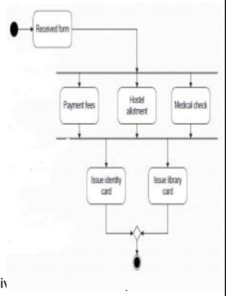
- A fork may have one incoming transitions and two or more outgoing transitions
  - each transition represents an independent flow of control
  - conceptually, the activities of each of outgoing transitions are concurrent
- A join may have two or more incoming transitions and one outgoing transition
  - 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

Notation



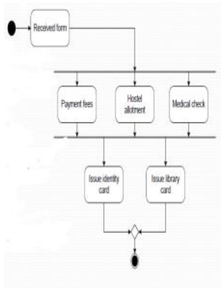
Basic Components in an activity Diagram

- **Initial node**The filled circle is the starting point of the process.
- **Final node:** The filled circle with a border is the ending point. An activity can have one or more activity final state.
- **Activity:** A rounded-corner rectangle represents an activity or task that needs to be performed.
– **Flow/edge:** Arrows depict triggers that initiate activities.
- **Parallel Activities:** A solid black bar is a synchronization bar that allows to depict activ



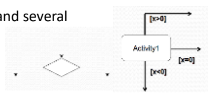
Basic Components in an activity Diagram

- **Fork:** A black bar (horizontal/vertical) with one flow going into it and several leaving it. This denotes the beginning of parallel activities.
- **Join:** A block bar with several flows entering it and one leaving it. This denotes the end of parallel activities.
- **Merge:** A diamond with several flows entering and one leaving. The implication is that all incoming flow to reach this point until processing continues.

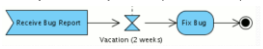


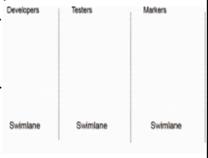
Basic Components in an activity diagram

- **Decision:** A diamond with one flow entering and several leaving. The flow leaving includes conditions as yes/no state.

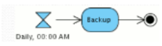


- **Swim lane:** A partition in activity diagram by means of dashed line, called swim lane. The swim lane may be horizontal or vertical. Swim lane can represent application/entity/department/group/role boundaries.
- **Object:** An object is represented by a rectangle.

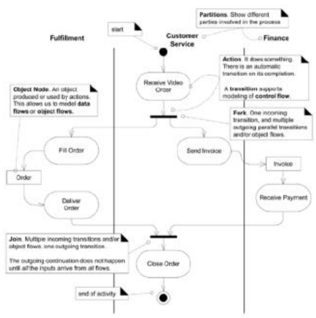




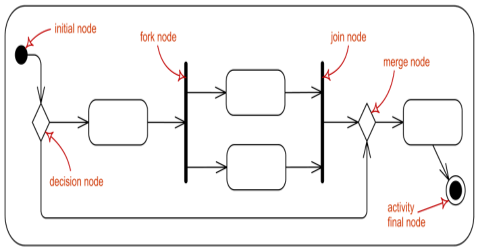
- **Time Event:** A time event models a wait period. An activity starting with a time is launched periodically.



Notations example


Partitions. Show different parties involved in the process
start
Fulfillment
Customer Service
Finance
Action. It does something. There is an automatic transition on its completion.
A transition supports modeling of control flow.
Receive Video Order
Object Node. An object produced or used by actions. This allows us to model data flows or object flows.
Fork. One incoming transition, and multiple outgoing parallel transitions and/or object flows.
Fill Order
Send Invoice
Invoice
Order
Deliver Order
Receive Payment
Join. Multiple incoming transitions and/or object flows; one outgoing transition.
The outgoing continuation does not happen until all the inputs arrive from all flows.
Close Order
end of activity

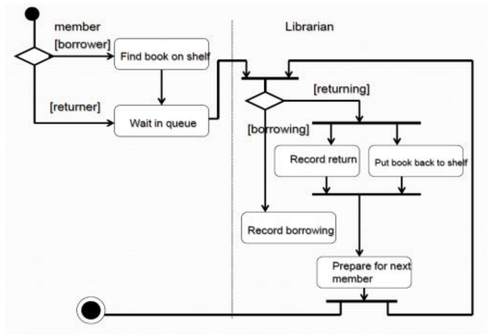
Branch &Merge, Fork & Join



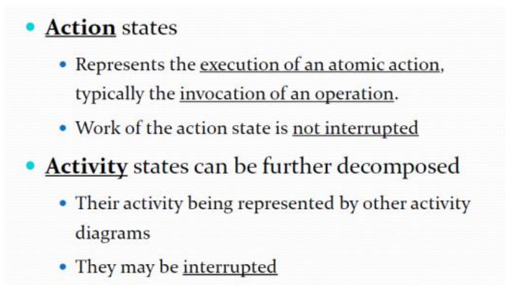
Difference b/w merge & Join

- A join is different from a merge in that the join synchronizes two or more inflows and produces a single outflow. The outflow from a join cant execute until all inflows have been received.
- A merge passes any control flows straight through it, when all incoming flows reach this point then processing continues.

Example: Activity Diagram of Library

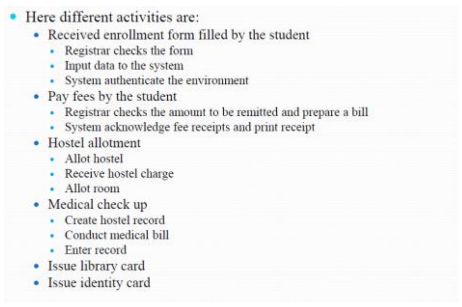


Action States & Activity States



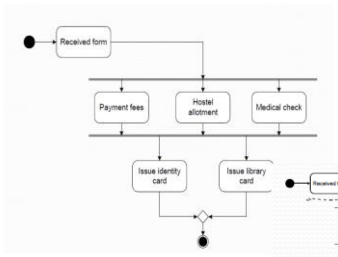
- **Action** states
  - Represents the execution of an atomic action, typically the invocation of an operation.
  - Work of the action state is not interrupted
- **Activity** states can be further decomposed
  - Their activity being represented by other activity diagrams
  - They may be interrupted

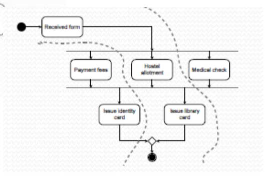
Student Enrolment System



- Here different activities are:
  - Received enrollment form filled by the student
    - Registrar checks the form
    - Input data to the system
    - System authenticate the environment
  - Pay fees by the student
    - Registrar checks the amount to be remitted and prepare a bill
    - System acknowledge fee receipts and print receipt
  - Hostel allotment
    - Allot hostel
    - Receive hostel charge
    - Allot room
  - Medical check up
    - Create hostel record
    - Conduct medical bill
    - Enter record
  - Issue library card
  - Issue identity card

Student Enrolment System





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.

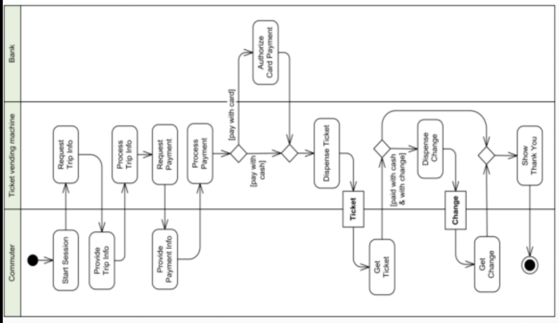
Swimlanes

- 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

Case: 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.

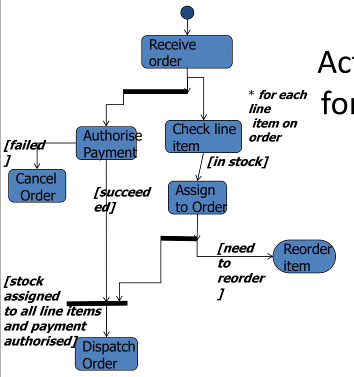
Ticket Vending Machine



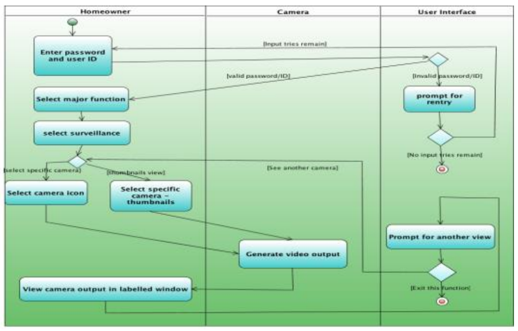
Use Case for Order Processing (Generic system)

- *"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."*

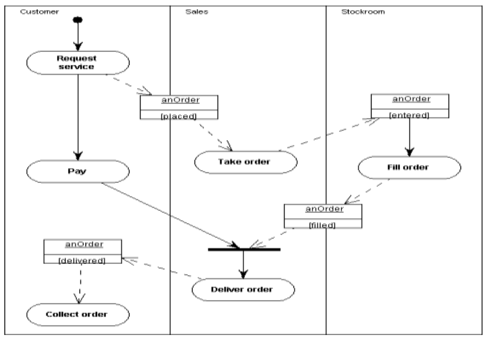
Activity Diagram for Receiving an Order

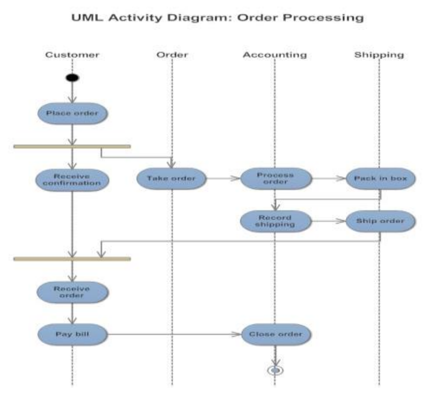


Swimlane Diagram



Activity Diagram: Example



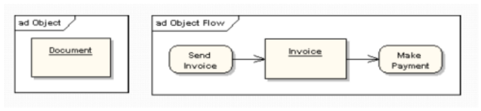


Object and Object Flow

- Object Flow:

Object Flow

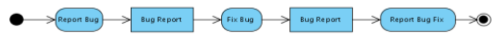
- There is no data flow in activity diagram. Object flow plays role of data flow as well.
- An object flow is an activity edge that can have objects or data passing along it.
- An object flow is shown as a connector with an arrowhead denoting the direction the object is being passed.



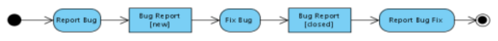
Object and Object Flow

- Object Node:

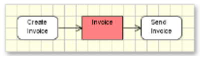
- An object node is an activity node that indicates an instance of a particular classifier. Object nodes can be used in a variety of ways, depending on where objects are flowing from and to.
- Objects in the UML language hold information - state - that is passed between actions. Objects may represent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

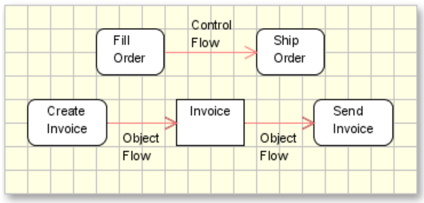


- Objects usually change state between actions:



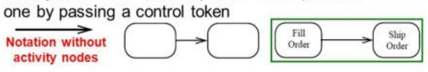
Object Flow and Control Flow



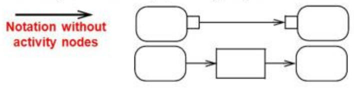


Activity edges

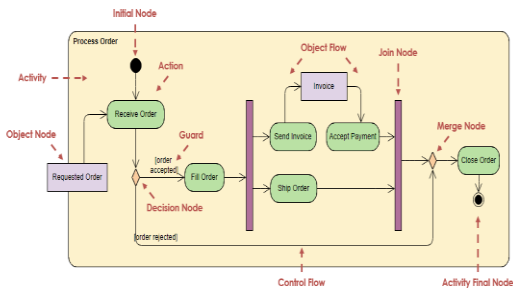
- Two kinds of edges:
  - **Control flow edge** - is an edge which starts an activity node after the completion of the previous one by passing a control token



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

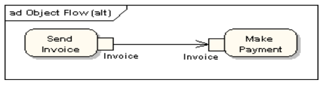


Example Object Flow and Control Flow

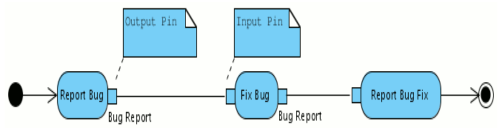


Input and Output Pins

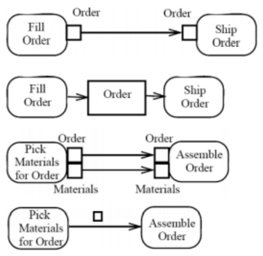
- An object flow must have an object on at least one of its ends. A shorthand notation for the above diagram would be to use input and output pins
- Input pins are object nodes that receive values from other actions through object flows
- Output pins are object nodes that deliver values to other actions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



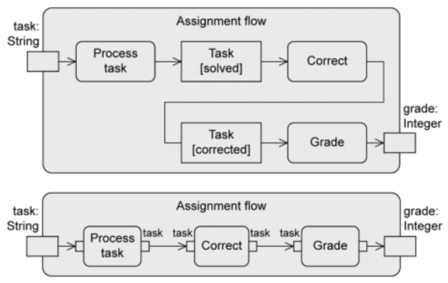
Input and Output Pins



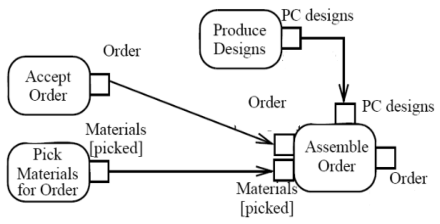
Input and Output Pins-Representations



Example: Assignment Flow

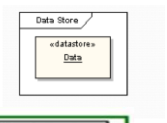


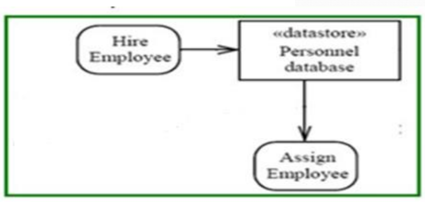
Input and Output Pins-Example



Data Store Node

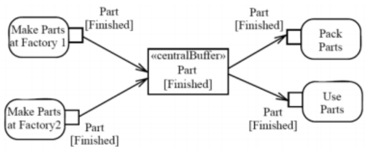
- A data store is shown as an object with the «datastore» keyword.





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

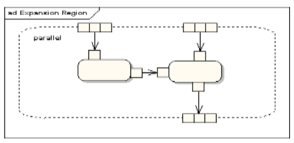
- Expansion Regions
- Interruptible Activity Regions.

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.

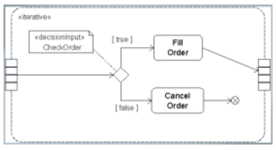
Modes of Multiple Executions

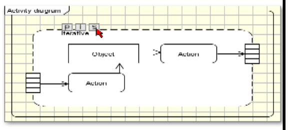
- Expansion Region's multiple executions can be specified as type parallel, iterative, or stream.
- The keyword iterative, parallel, or stream is shown in the top left corner of the region
- Parallel reflects that the elements in the incoming collections can be processed at the same time or overlapping.



Modes of Multiple Executions

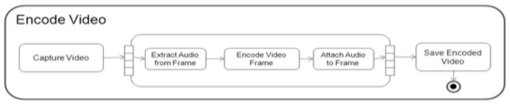
- An **iterative** concurrency type specifies that execution must occur sequentially.

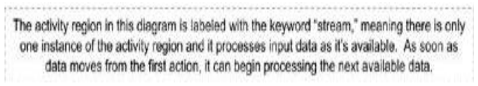




Modes of Multiple Executions

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

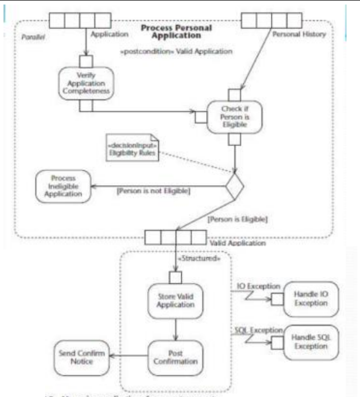



The activity region in this diagram is labeled with the keyword "stream," meaning there is only one instance of the activity region and it processes input data as it's available. As soon as data moves from the first action, it can begin processing the next available data.

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.**

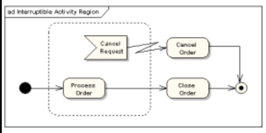


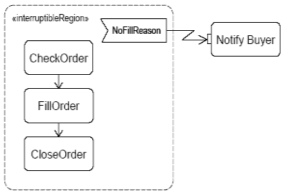
.8 Managing applications for monetary grants.

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.

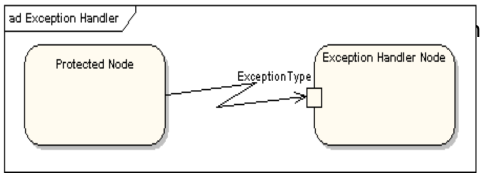
Interruptible Activity Regions Examples





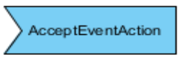
Exception Handler

- 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.
-

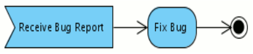


Accept Event Action

- 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“.



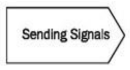
- An activity may also start with a ***receive signal node:***



Send Signal Action

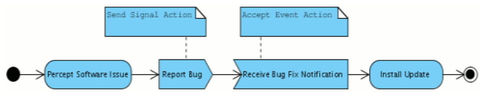
- The sending of signals means that a signal is sent to a receiving activity: The receiving activity accepts the signal with the action accepting a signal and can respond accordingly, meaning, according to the flow that comes from this node in the activity diagram.

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



Interacting with External Participants

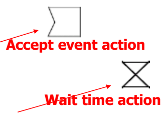
- In the following diagram, both a send and receive signal action are used.

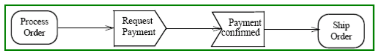


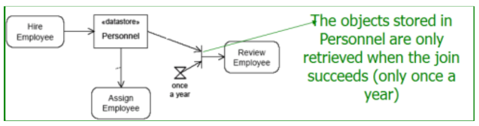
- Note that the activity flow gets interrupted - gets into a wait state - until the bug fix notification is received. (If there was no receive signal action, however, the flow would just continue after executing the send signal action.)

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





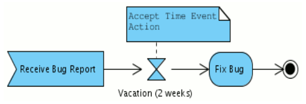

Hire Employee
«datastore» Personnel
Assign Employee
once a year
Review Employee
The objects stored in Personnel are only retrieved when the join succeeds (only once a year)

Accept Time Event Action

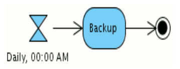
- If the occurrence is a time event occurrence, the result value contains the time at which the occurrence transpired. Such an action is informally called a wait time action.

AcceptEventAction

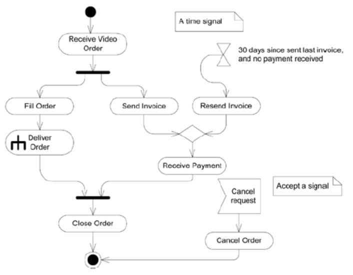
- A ***time event models a wait period:***



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

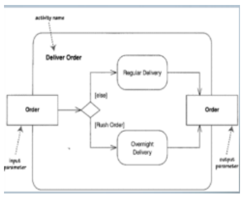


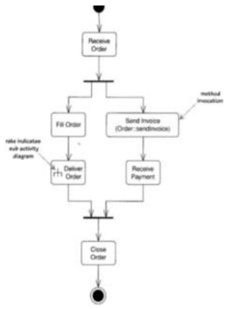
Signal Example



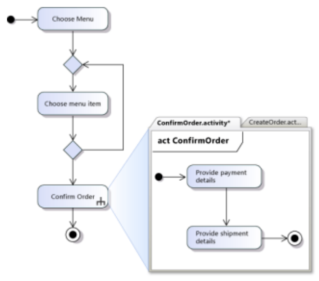
How to Represent Sub- activity Diagram

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

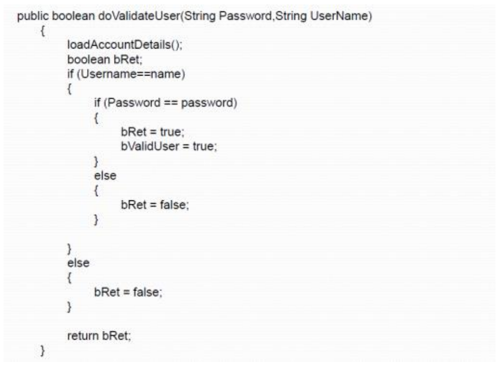




Another Example



Conditional Statements

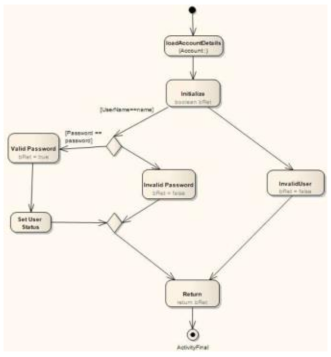


```
public boolean doValidateUser(String Password,String UserName)
    {
        loadAccountDetails();
        boolean bRet;
        if (Username==name)
        {
            if (Password == password)
            {
                bRet = true;
                bValidUser = true;
            }
       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

        }
        else
        {
            bRet = false;
        }

        return bRet;
    }
```

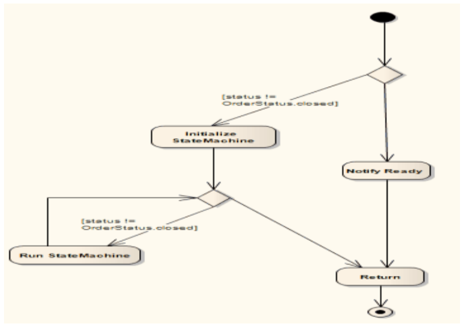
Conditional Statements



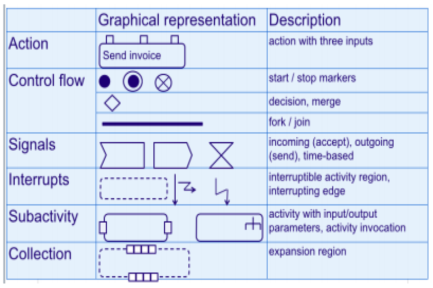
Loops

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

Loops

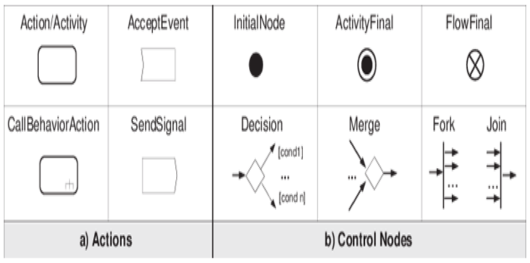

[status != OrderStatus.closed]
Initialize StateMachine
Notify Ready
[status != OrderStatus.closed]
Run StateMachine
Return

Notations - Recap



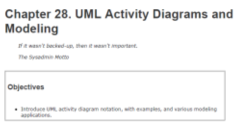
	Graphical representation	Description
Action	Send invoice	action with three inputs
Control flow	● ◉ ⊗	start / stop markers
	◇	decision, merge
	▬	fork / join
Signals		incoming (accept), outgoing (send), time-based
Interrupts		interruptible activity region, interrupting edge
Subactivity		activity with input/output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

Notations



READING

Chapter 28 – Applying UML and Pattern by Craig Larman 3rd Edition



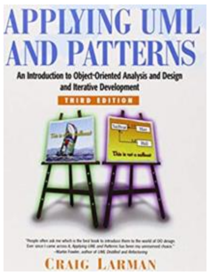
Chapter 28. UML Activity Diagrams and Modeling

If it wasn't backed-up, then it wasn't important.

The Sysadmin Motto

Objectives

- Introduce UML activity diagram notation, with examples, and various modeling applications.


APPLYING UML AND PATTERNS
An Introduction to Object-Oriented Analysis and Design and Iterative Development
THIRD EDITION
CRAIG LARMAN

END OF TOPIC 9

-COMING UP!!!!!!

-RUP
-ECB Pattern Robust Analysis
-Sequence Diagrams