Activity Diagrams

To model a sequence of actions (flow) such as

- The use case main & alternative flow
- The flow of method operations

Use Case Description

Use case: Place Order
ID: 1
Brief description: The customer places an order through the Mail Order System
Primary actors: Customer
Secondary actors: None
Preconditions: 1. The customer must have an account
Main flow: 1. The customer logs in 2. The customer browses the list of items 3. The customer selects an item and places it in the cart ... n. A confirmation is sent to the customer
Postconditions: 1. The order is logged 2. The customers received a confirmation

Activity Diagram

Use Case ➔ Activity Diagram

Use case: ChangeEmployeeDetails
ID: 1
Brief description: The Manager changes the employee details.
Primary actors: Manager
Seconday actors: None
Preconditions: 1. The Manager is logged on to the system.
Main flow: 1. include(FindEmployeeDetails). 2. The system displays the employee details. 3. The Manager changes the employee details. ...
Postconditions: 1. The employee details have been changed.
Alternative flows: None.

Use case: FindEmployeeDetails
ID: 4
Brief description: The Manager finds the employee details.
Primary actors: Manager
Seconday actors: None
Preconditions: 1. The Manager is logged on to the system.
Main flow: 1. The Manager enters the employee's ID. 2. The system finds the employee details.
Postconditions: 1. The system has found the employee details.
Alternative flows: None.

What are activity diagrams?

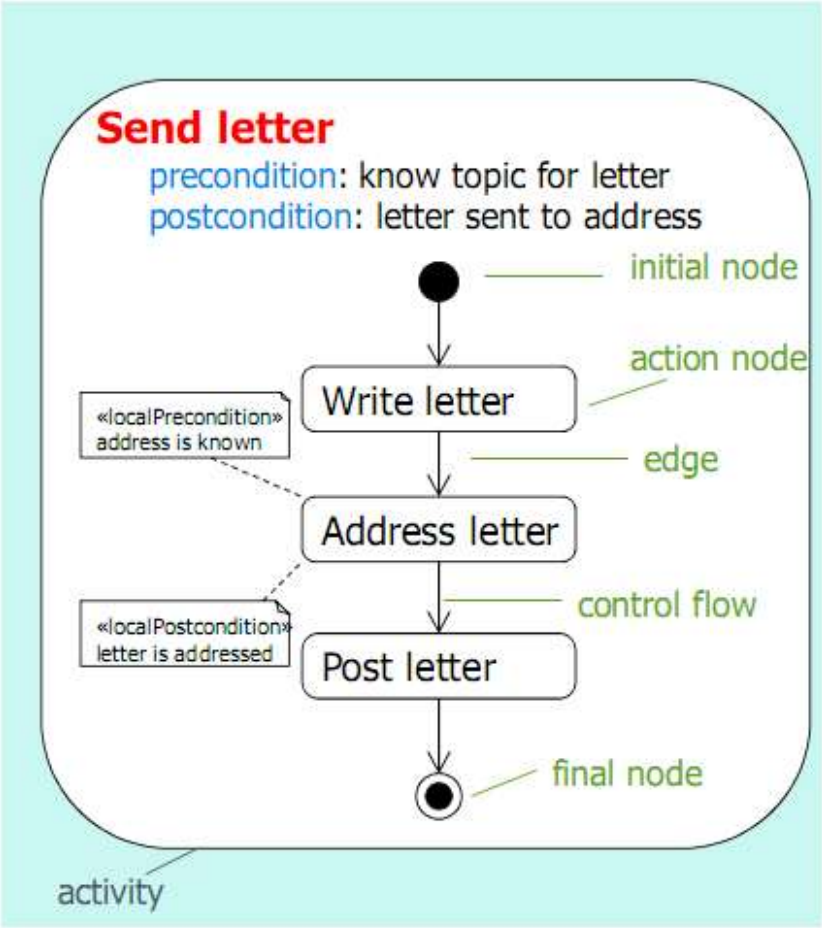
- Activity modeling focuses on the *execution and flow of behavior* of a system
- An **activity** is a *behavior* that is factored into one or more *actions*.
- An **action** represents discrete units of work that are *atomic* within the activity
- Activity diagrams can be used to model the behavior of:
  - use cases
  - classes
  - interfaces
  - components
  - collaborations
  - operations and methods

Activities

- Activities are represented as *networks of nodes connected by edges*
- There are three categories of node:
  - **Action nodes** - represent discrete units of work that are *atomic* within the activity
  - **Control nodes** - control the flow through the activity
  - **Object nodes** - provide input and output parameters to activities
- Edges represent flow through the activity
- There are two categories of edge:
  - **Control flows** - represent the flow of control through the activity
  - **Object flows** - represent the flow of objects through the activity

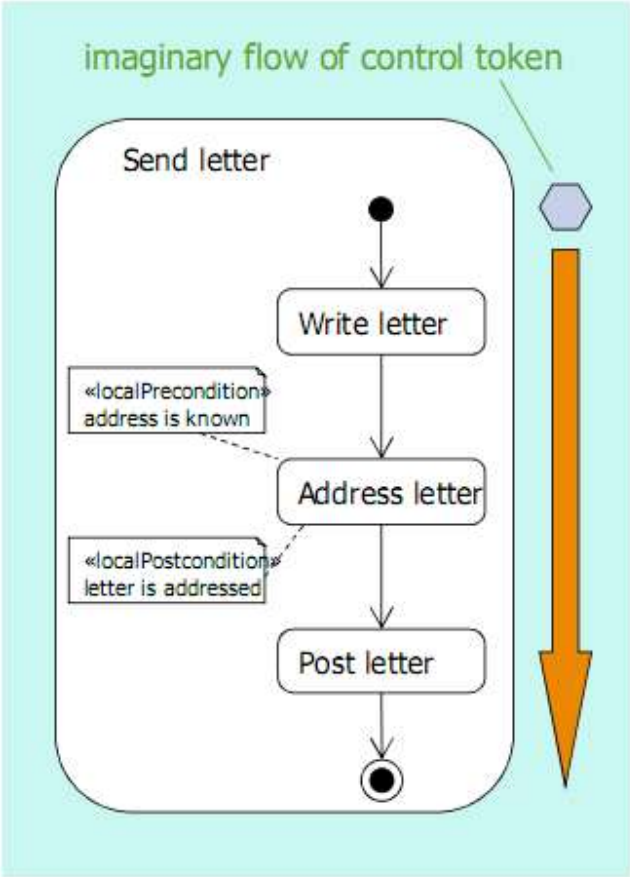
Activity diagram syntax

- Activities are networks of *nodes* connected by *edges*
  - The control flow is a type of edge
- Activities usually start in an *initial node* and terminate in a *final node*
- Activities can have *preconditions* and postconditions
- You can break an edge using *connectors*

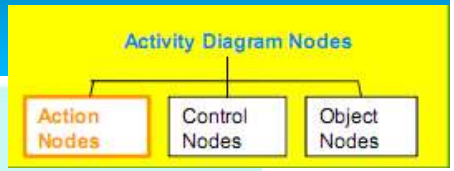


Activity diagram semantics

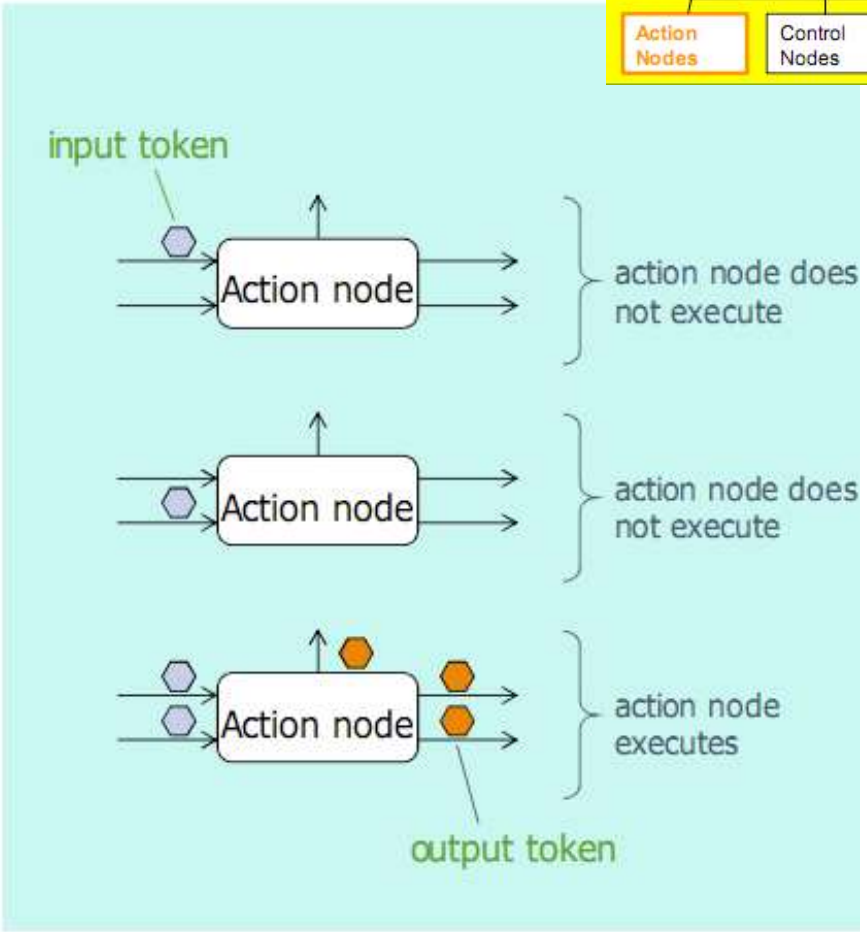
- UML models information moving along an edge as a *token*
  - Token – an object, some data or a focus of control
- Tokens traverse from a source node to a target node via an edge
- A node executes when:
  - It has tokens on all of its input edges AND these tokens satisfy predefined conditions
- When a node starts to execute it takes tokens off its input edges
- When a node has finished executing it offers tokens on its output edges



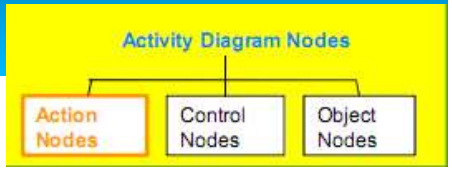
Action Nodes



- Action nodes offer a token on *all* of their output edges when:
  - There is a token *simultaneously* on each input edge
  - The input tokens satisfy all preconditions specified by the node
- Action nodes:
  - Perform a logical AND on their input edges when they begin to execute
  - Perform an implicit fork on their output edges when they have finished executing

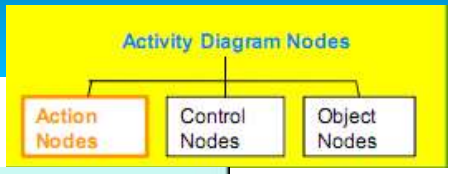


Types of Action Node



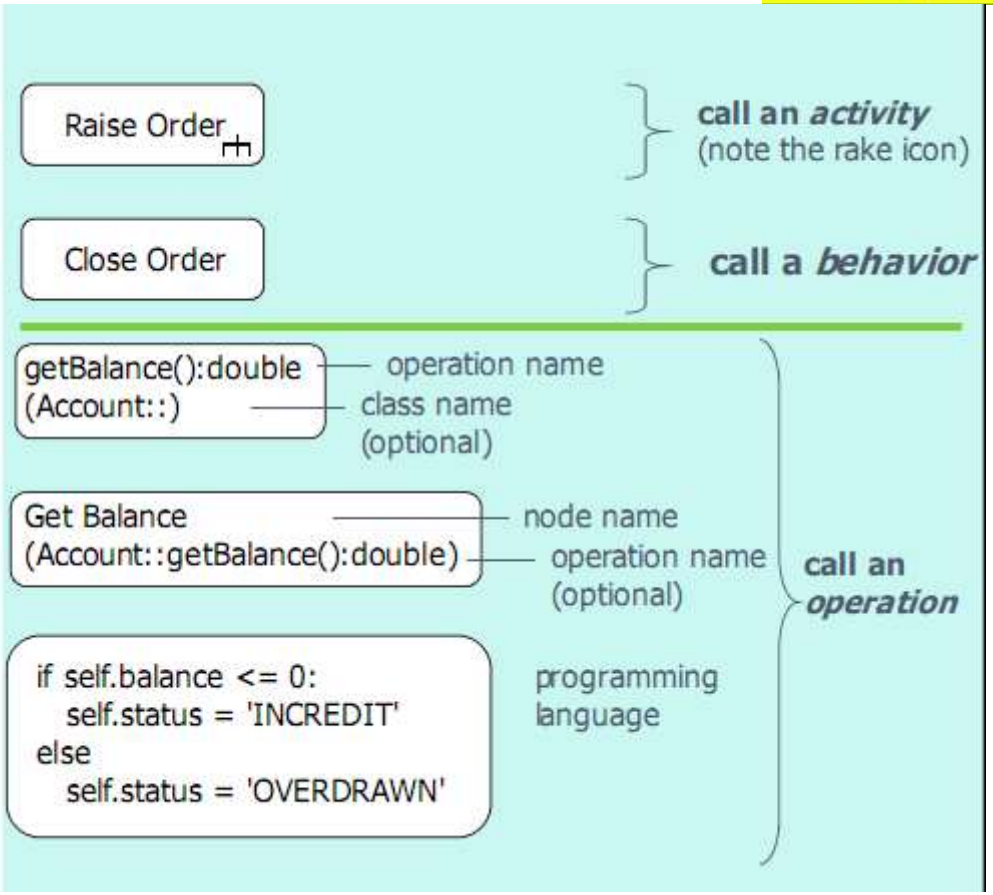
action node syntax	action node semantics
Close Order	Call action - invokes an activity, a behavior or an operation. The most common type of action node.

Call action node syntax

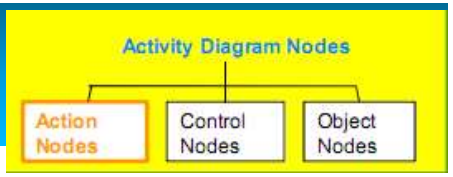


The most common type of node

- Call action nodes may invoke:
  - an activity
  - a behavior
  - an operation



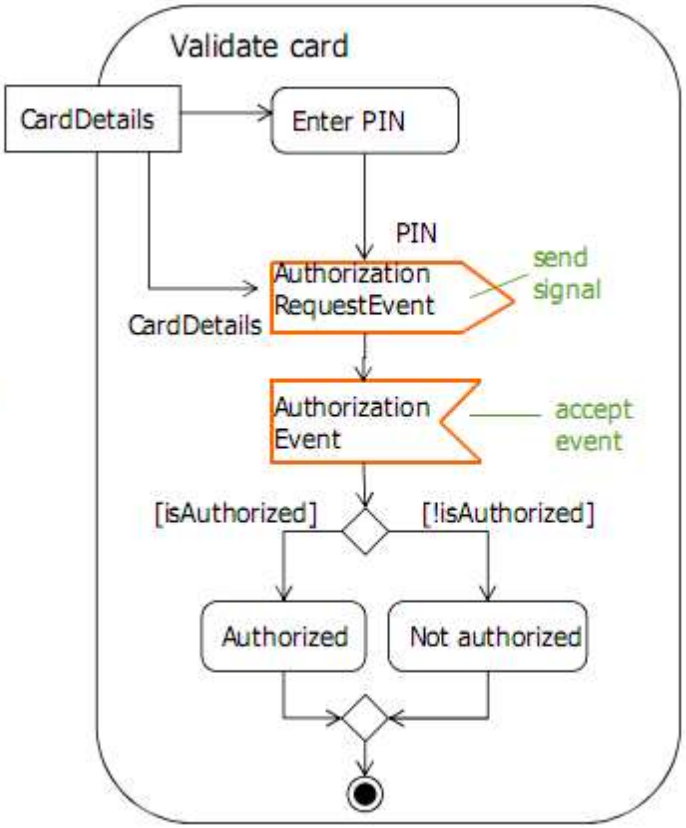
Types of Action Node



action node syntax	action node semantics
Close Order	Call action - invokes an activity, a behavior or an operation. The most common type of action node.
OrderEvent signal type	Send signal action - sends a signal *asynchronously.* The sender does not wait for confirmation of signal receipt. It may accept input parameters to create the signal
OrderEvent event type	Accept event action - waits for events detected by its owning object and offers the event on its output edge. Is enabled when it gets a token on its input edge. If there is no input edge it starts when its containing activity starts and is always enabled.
end of month occurred wait 30 mins time expression	Accept time event action - waits for a set amount of time. Generates time events according to its time expression.

Sending signals and accepting events

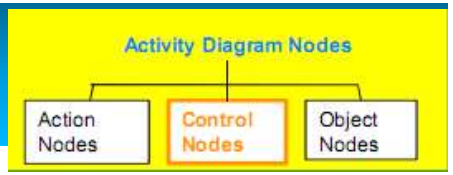
- Signals represent information passed *asynchronously* between objects
  - This information is modelled as attributes of a signal
  - A signal is a classifier stereotyped «signal»
- The accept event action asynchronously accepts event triggers which may be signals or other objects



Outline

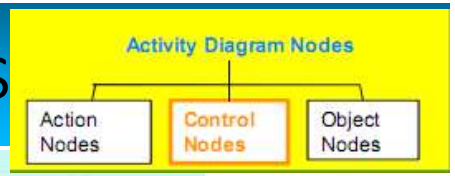
- Dynamic Aspects
- Use Case Diagrams
- Activity Diagrams
  - What are activity diagrams?
  - Basic Components of Activity Diagrams
    - Action Nodes
    - Control Nodes
    - Object Nodes
  - Advanced Activity Diagrams
- Statechart Diagrams
- Interaction Diagrams

Control Nodes

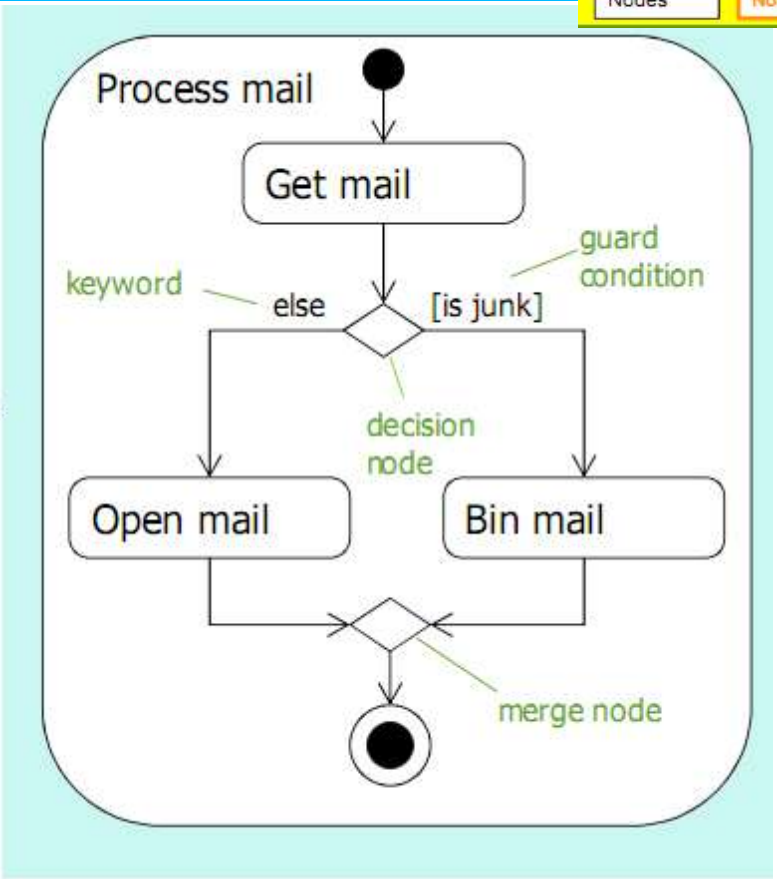


control node syntax	control node semantics	
	Initial node – indicates where the flow starts when an activity is invoked	
	Activity final node – terminates an activity	Final nodes
	Flow final node – terminates a specific flow within an activity. The other flows are unaffected	
«decisionInput» decision condition	Decision node– guard conditions on the output edges select one of them for traversal May optionally have inputs defined by a «decisionInput»	See examples on next two slides
	Merge node – selects one of its input edges	
	Fork node – splits the flow into multiple concurrent flows	
{join spec}	Join node – synchronizes multiple concurrent flows May optionally have a join specification to modify its semantics	

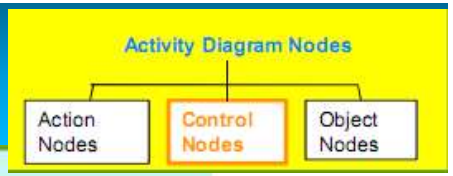
Decision and Merge Nodes



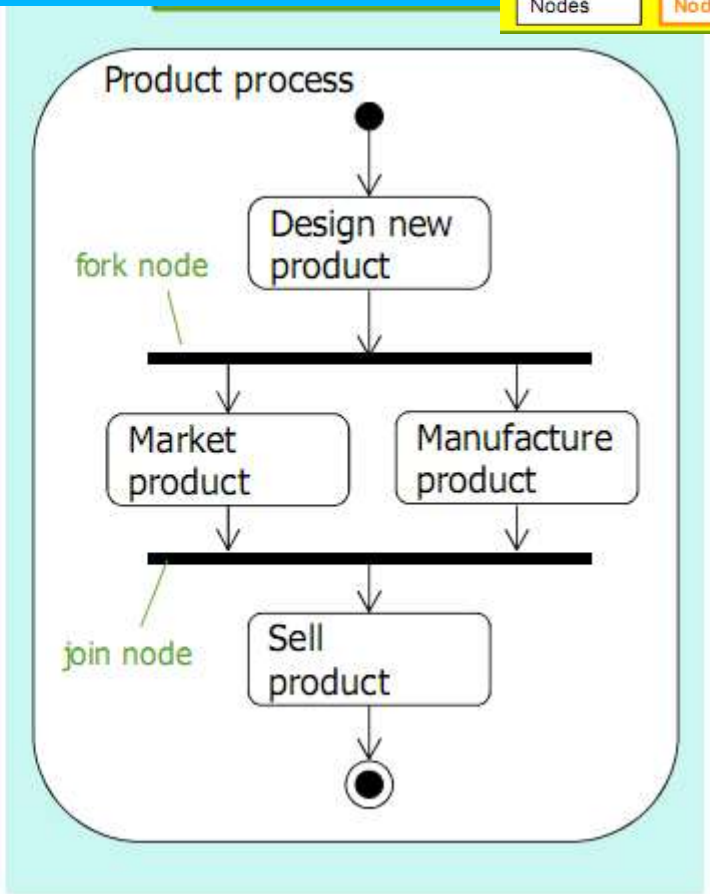
- A decision node is a control node that has one input edge and two or more alternate output edges
  - Each edge out of the decision is protected by a *guard condition*
  - guard conditions must be mutually exclusive
  - The edge can be taken if and only if the guard condition evaluates to true
  - The keyword *else* specifies the path that is taken if *none* of the guard conditions are true
- A merge node accepts one of several alternate flows
  - It has two or more input edges and exactly one output edge



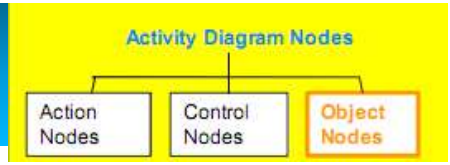
Fork and join - concurrency

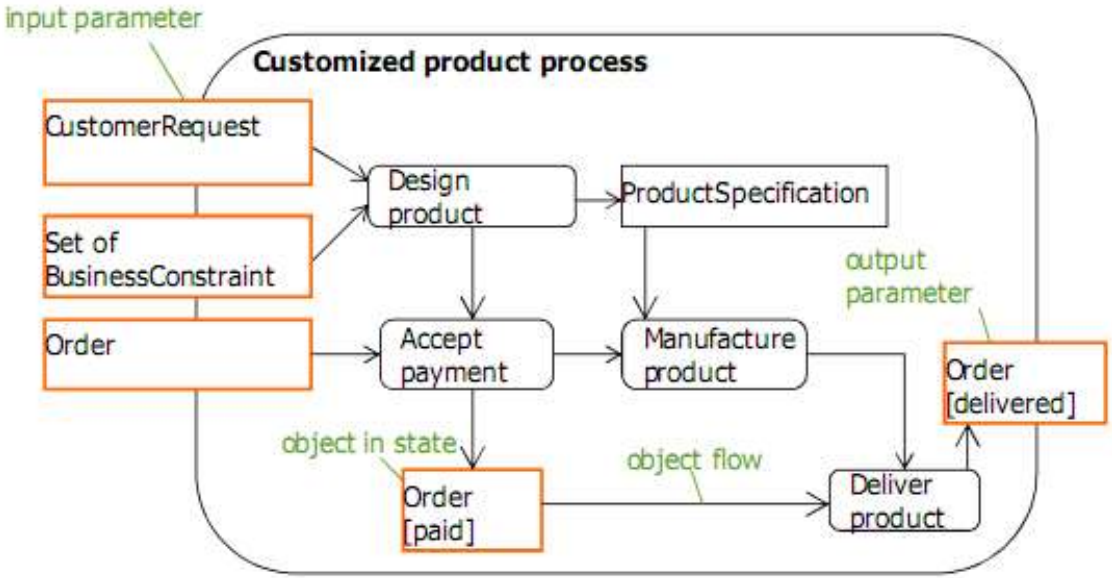


- Fork nodes model *concurrent flows* of work
  - Tokens on the single input edge are replicated at the multiple output edges
- Join nodes *synchronize* two or more concurrent flows
  - Joins have two or more incoming edges and exactly one outgoing edge
  - A token is offered on the outgoing edge when there are tokens on *all* the incoming edges i.e., when the concurrent flows of work have all finished



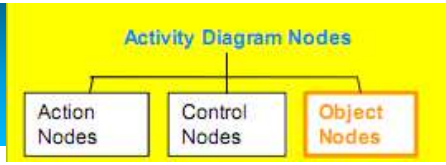
Object Nodes



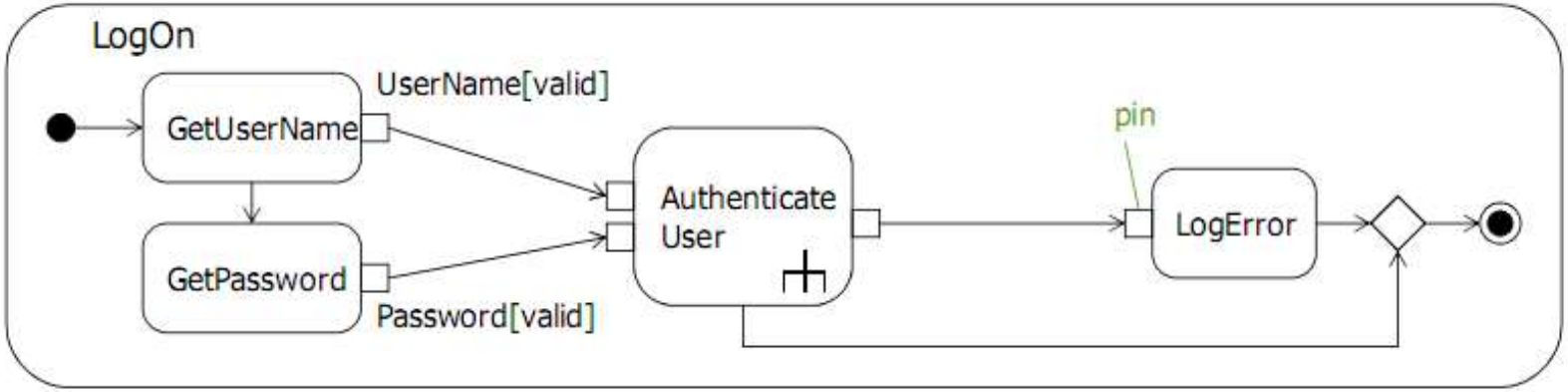


- Object nodes can provide *input* and *output parameters* to activities
- Draw the input and output object nodes overlapping the activity boundary

Pins

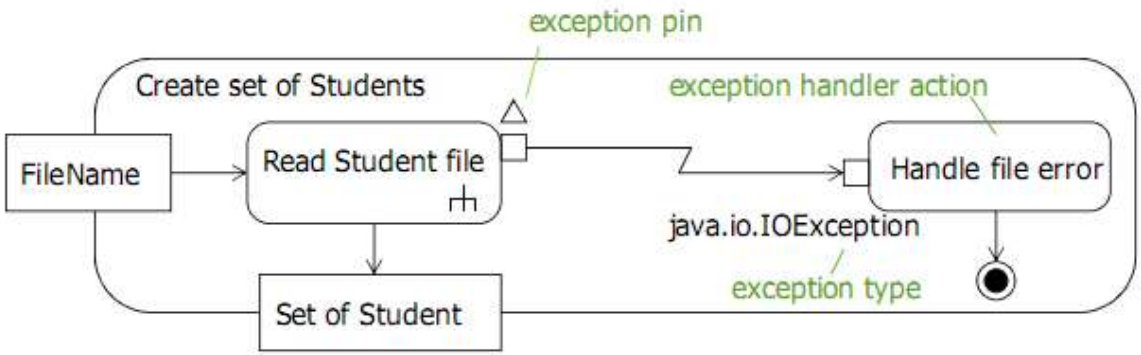


- A **pin** is a *shorthand notation* for input to or output from an action



- Same syntax as object nodes
- Input pins have exactly one input edge
- Output pins have exactly one output edge

Exception pin

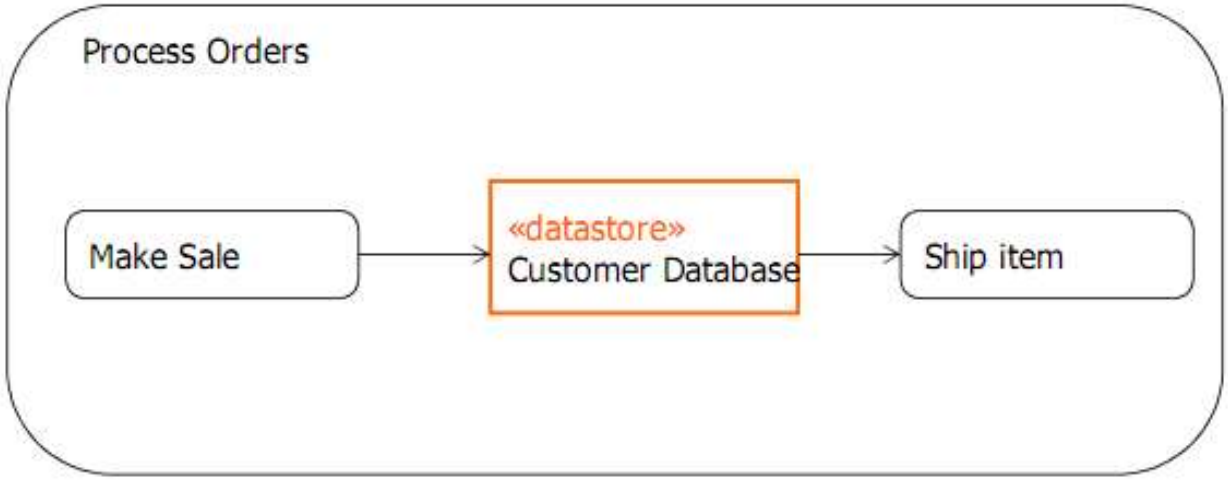


- Exception pins are marked with an equilateral triangle
- If an exception occurs while an action is executing, the execution is abandoned and there is no output from the action
- If the action has an exception handler, the handler is executed with the exception information

Activity Diagrams Summary

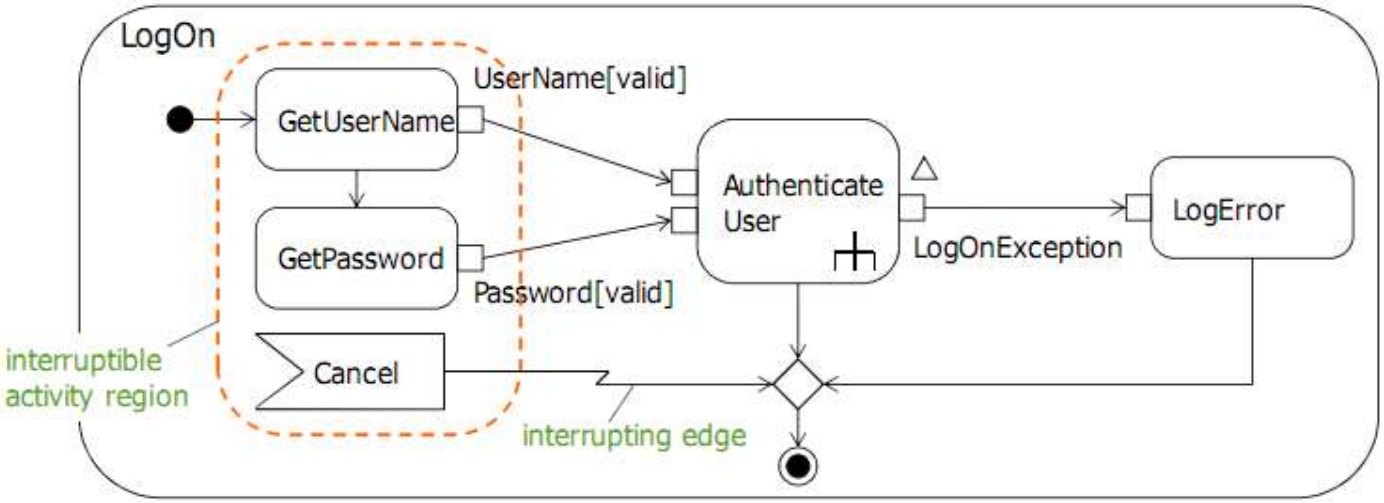
- We have seen how we can use activity diagrams to model flows of activities using:
  - Activities
  - Action nodes
    - Call action node
    - Send signal/accept event action node
    - Accept time event action node
  - Control nodes
    - decision and merge
    - fork and join
  - Object nodes
    - input and output parameters
    - pins

<<datastore>> node



- A data store node is a special type of central buffer node that copies all data that passes through it.

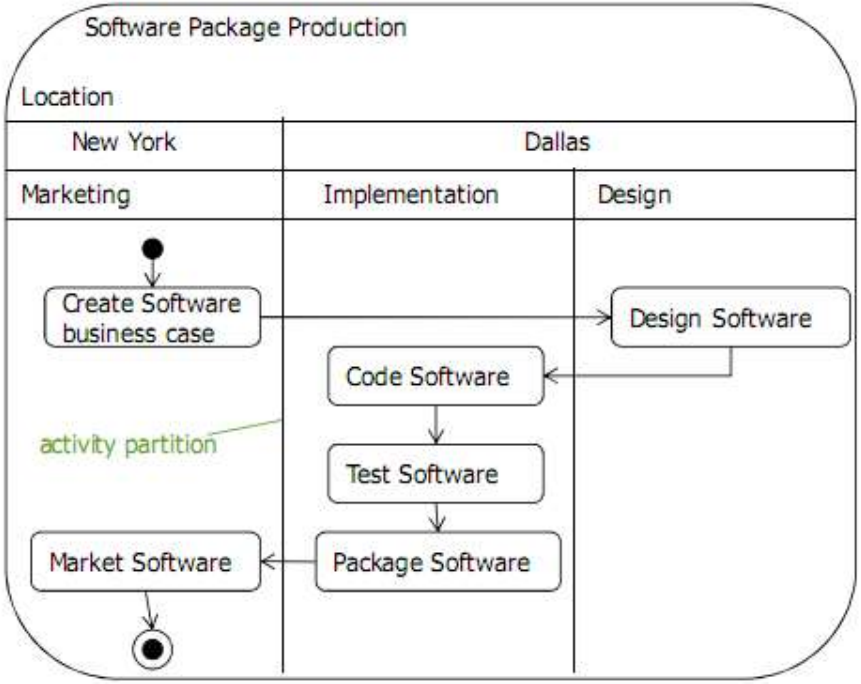
Interruptible activity regions



- Interruptible activity regions may be interrupted when a token traverses an interrupting edge
  - All flows in the region are aborted
- Interrupting edges *must* cross the region boundary

Activity partitions

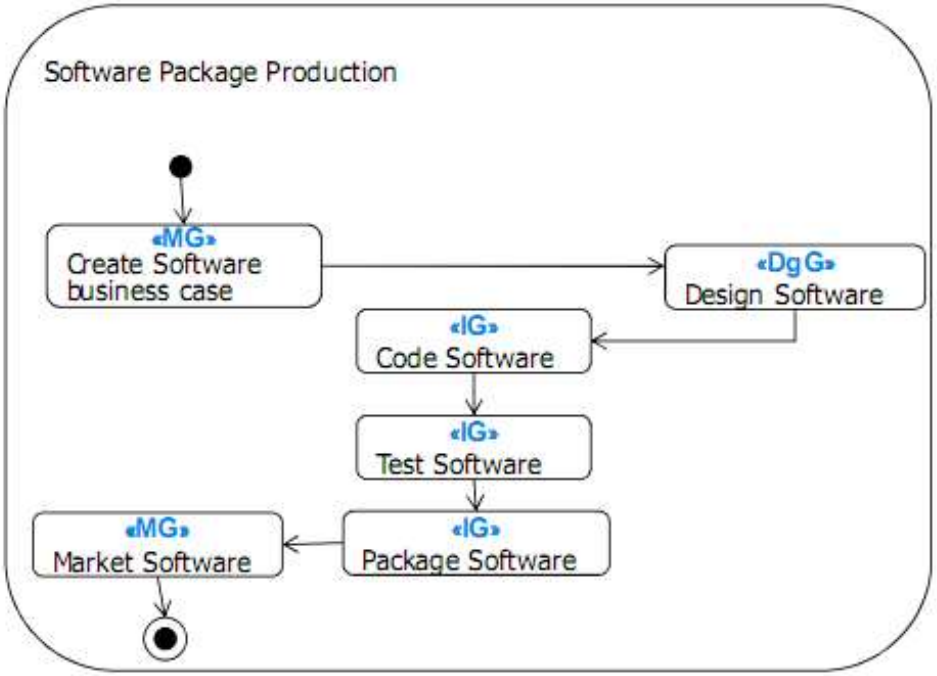
Sometimes it is helpful to indicate who or what is responsible for a set of actions in an activity diagram



- Partitions may refer to many different things e.g., business organisations, classes, components, etc.

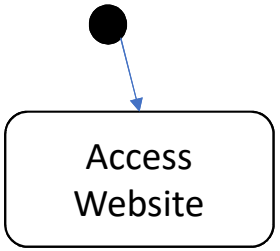
Activity partitions

Another notation



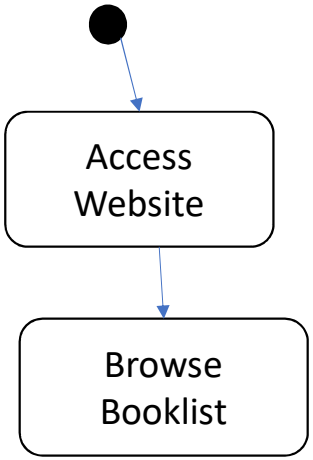
Activity Diagrams – Exercise: BookLib

- We are interested in modeling an online book ordering system for BookLib . The user accesses BookLib's website, then browses the catalog, and adds items to the cart. At the end, the user can 1) cancel, 2) remove books and continue adding or 3) checkout. Checkout then proceed to placing an order where the name, address, credit card details, etc. are required. Then a sales note, internal note and debit note are generated. The product quantities are updated. An exception handler checks for browser errors at checkout time. When an error occurs, alternative code is executed.



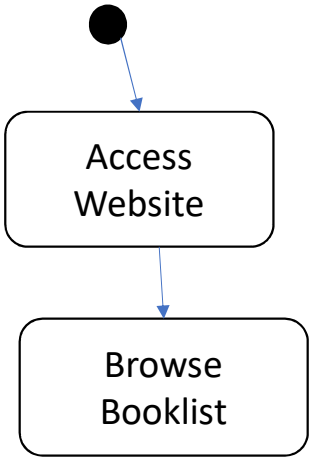
- We are interested in modeling an online book ordering system for BookLib . The user accesses BookLib's website, then browses the catalog, and adds items to the cart. At the end, the user can 1) cancel, 2) remove books and continue adding or 3) checkout. Checkout then proceed to placing an order where the name, address, credit card details, etc. are required. Then a sales note, internal note and debit note are generated. The product quantities are updated. An exception handler checks for browser errors at checkout time. When an error occurs, alternative code is executed.

1



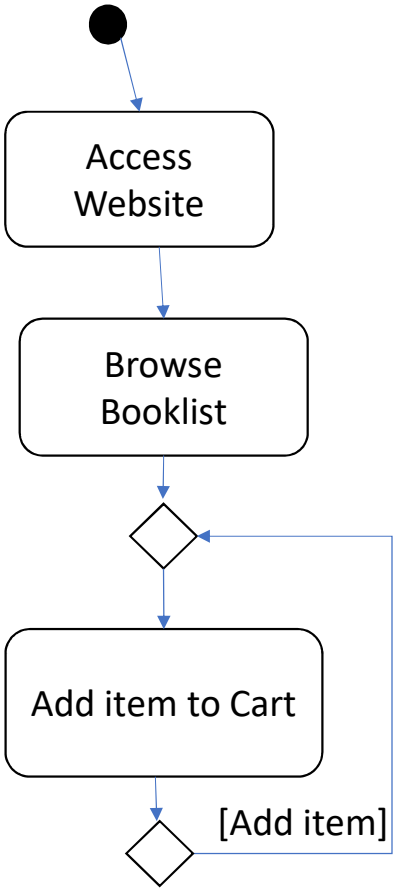
- We are interested in modeling an online book ordering system for BookLib . The user accesses BookLib's website, then browses the catalog, and adds items to the cart. At the end, the user can 1) cancel, 2) remove books and continue adding or 3) checkout. Checkout then proceed to placing an order where the name, address, credit card details, etc. are required. Then a sales note, internal note and debit note are generated. The product quantities are updated. An exception handler checks for browser errors at checkout time. When an error occurs, alternative code is executed.

2

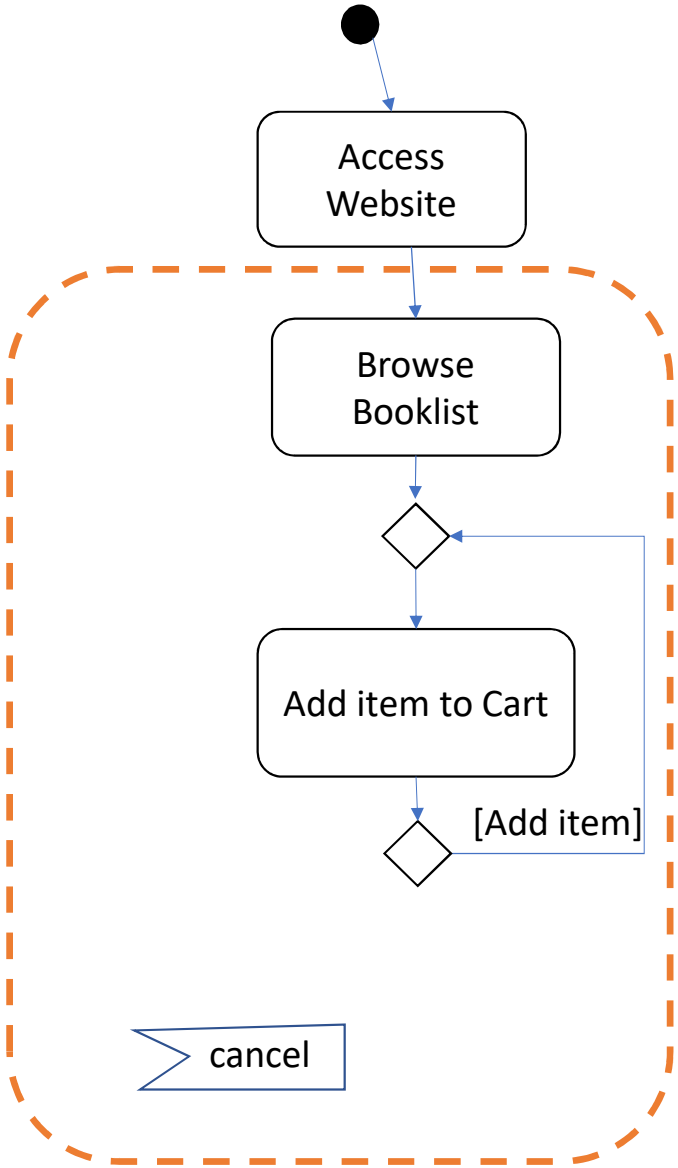


- We are interested in modeling an online book ordering system for BookLib . The user accesses BookLib's website, then browses the catalog, and adds items to the cart. At the end, the user can 1) cancel, 2) remove books and continue adding or 3) checkout. Checkout then proceed to placing an order where the name, address, credit card details, etc. are required. Then a sales note, internal note and debit note are generated. The product quantities are updated. An exception handler checks for browser errors at checkout time. When an error occurs, alternative code is executed.

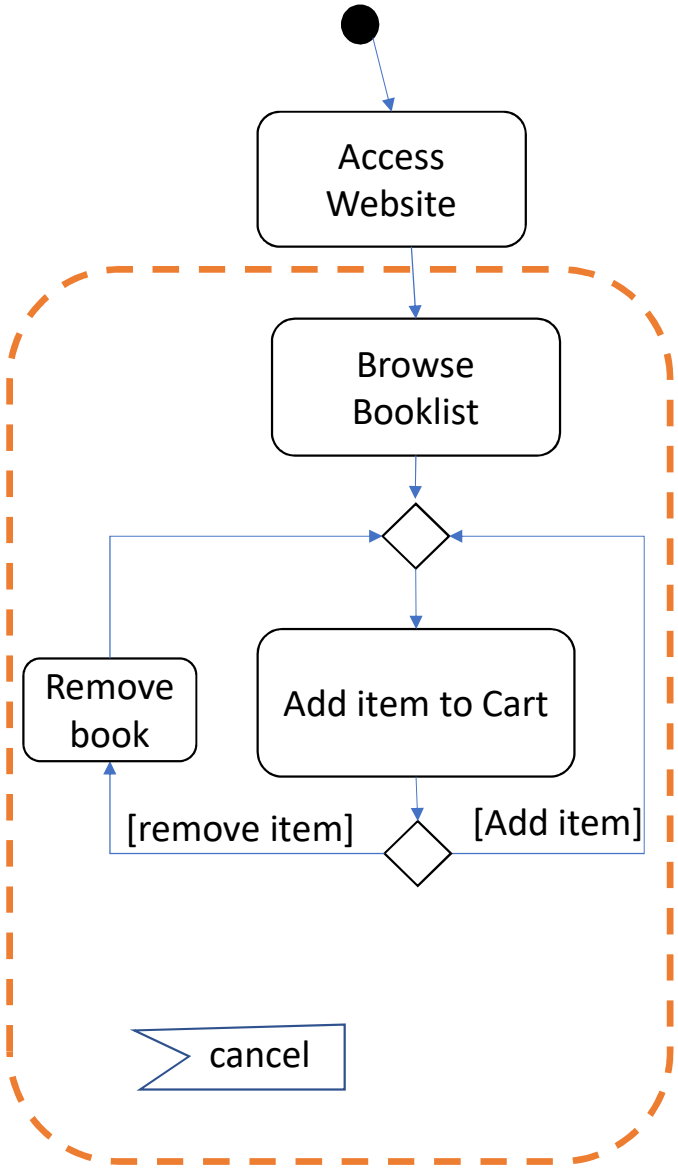
2



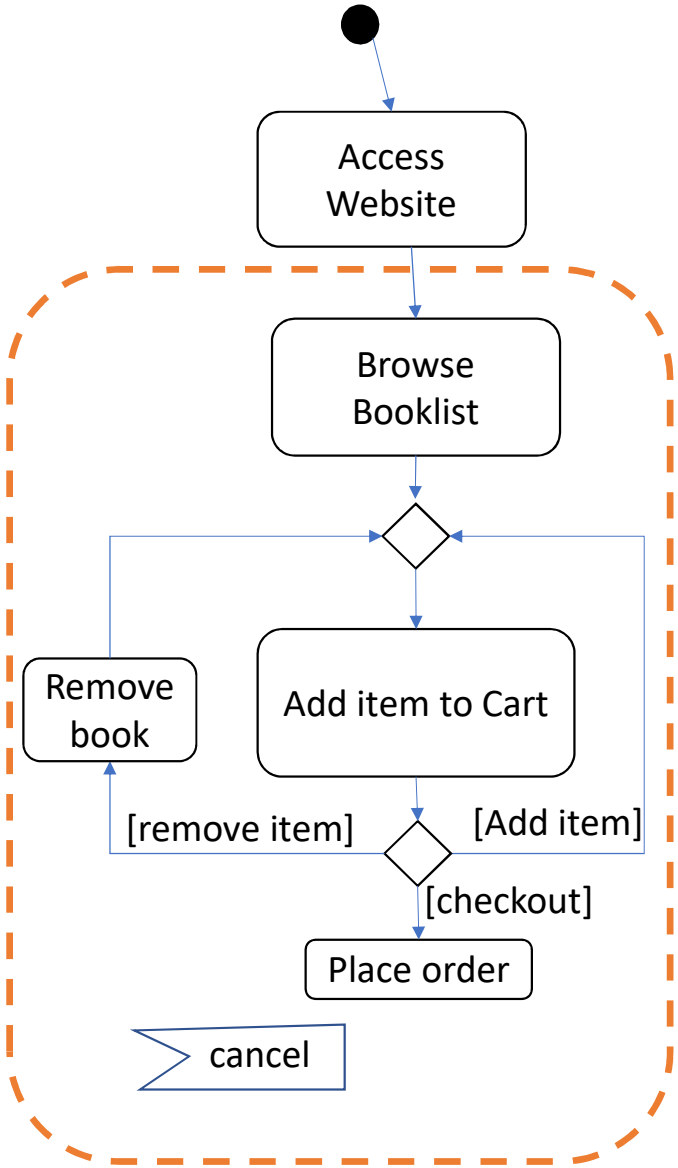
- We are interested in modeling an online book ordering system for BookLib . The user accesses BookLib's website, then browses the catalog, and adds items to the cart. 3 the end, the user can 1) cancel, 2) remove books and continue adding or 3) checkout. Checkout then proceed to placing an order where the name, address, credit card details, etc. are required. Then a sales note, internal note and debit note are generated. The product quantities are updated. An exception handler checks for browser errors at checkout time. When an error occurs, alternative code is executed.



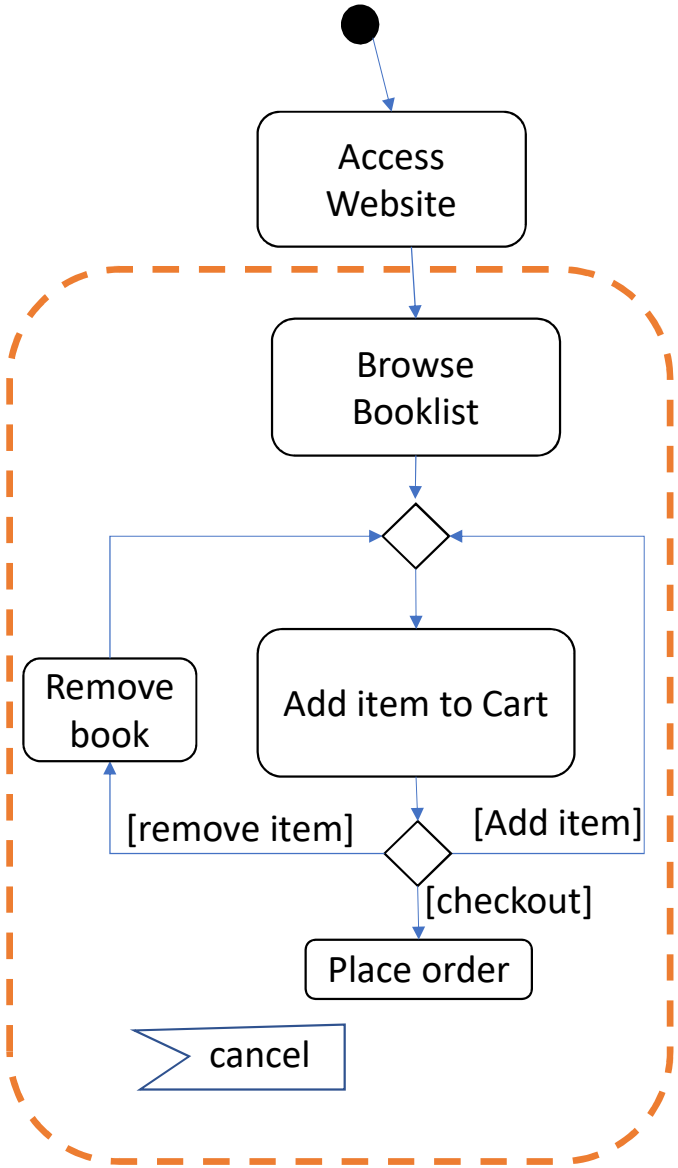
- We are interested in modeling an online book ordering system for BookLib . The user accesses BookLib's website, then browses the catalog, and adds items to the cart. At the end, the user (4 1) cancel, 2) remove books and continue adding or 3) checkout. Checkout then proceed to placing an order where the name, address, credit card details, etc. are required. Then a sales note, internal note and debit note are generated. The product quantities are updated. An exception handler checks for browser errors at checkout time. When an error occurs, alternative code is executed.



- We are interested in modeling an online book ordering system for BookLib . The user accesses BookLib's website, then browses the catalog, and adds items to the cart. At the end, the user can 1) cancel 5 remove books and continue adding or 3) checkout. Checkout then proceed to placing an order where the name, address, credit card details, etc. are required. Then a sales note, internal note and debit note are generated. The product quantities are updated. An exception handler checks for browser errors at checkout time. When an error occurs, alternative code is executed.

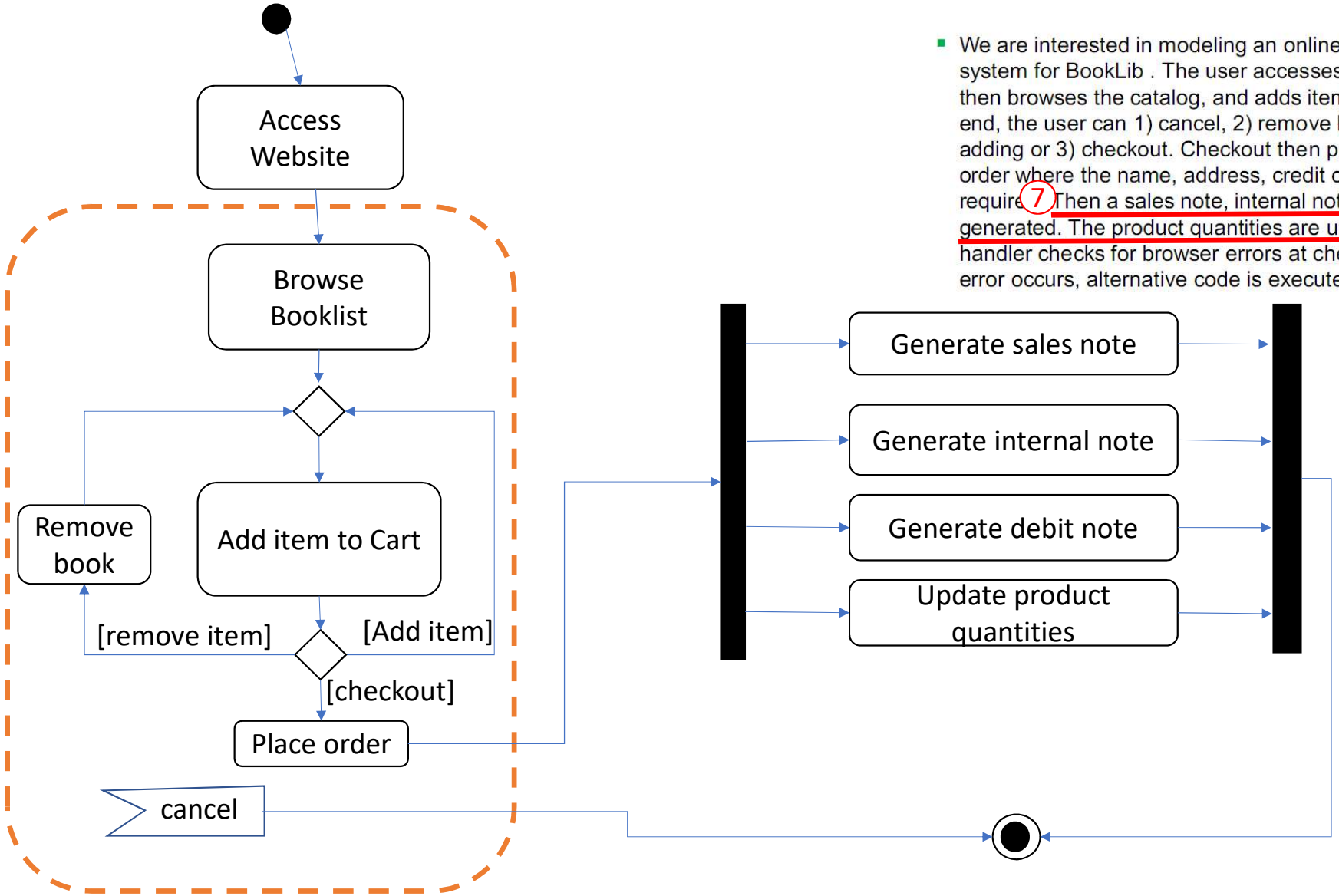


- We are interested in modeling an online book ordering system for BookLib . The user accesses BookLib's website, then browses the catalog, and adds items to the cart. At the end, the user can 1) cancel, 2) remove books and continue adding o 6 checkout. Checkout then proceed to placing an order where the name, address, credit card details, etc. are required. Then a sales note, internal note and debit note are generated. The product quantities are updated. An exception handler checks for browser errors at checkout time. When an error occurs, alternative code is executed.

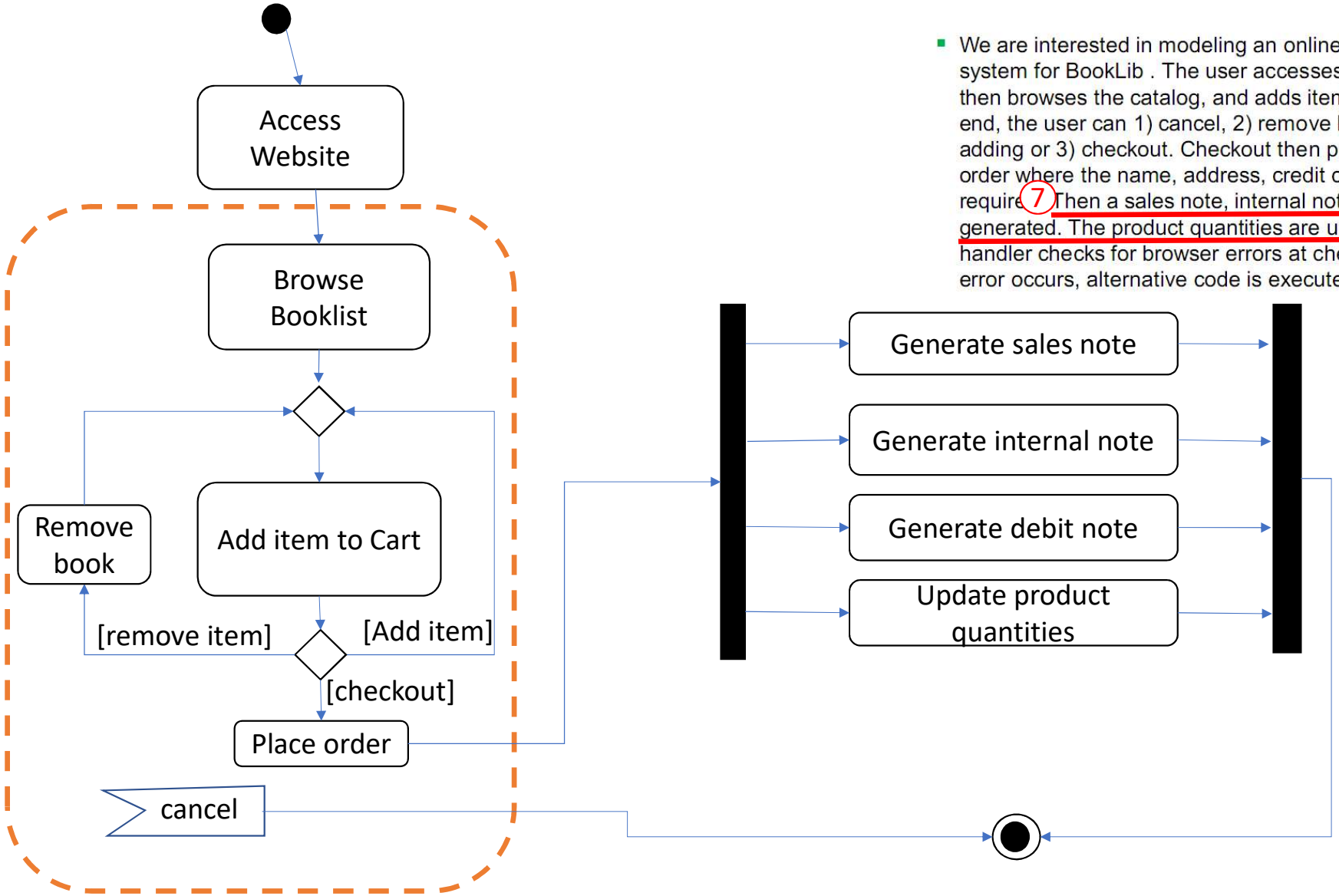


- We are interested in modeling an online book ordering system for BookLib . The user accesses BookLib's website, then browses the catalog, and adds items to the cart. At the end, the user can 1) cancel, 2) remove books and continue adding o 6 checkout. Checkout then proceed to placing an order where the name, address, credit card details, etc. are required. Then a sales note, internal note and debit note are generated. The product quantities are updated. An exception handler checks for browser errors at checkout time. When an error occurs, alternative code is executed.

- We are interested in modeling an online book ordering system for BookLib . The user accesses BookLib's website, then browses the catalog, and adds items to the cart. At the end, the user can 1) cancel, 2) remove books and continue adding or 3) checkout. Checkout then proceed to placing an order where the name, address, credit card details, etc. are require 7 Then a sales note, internal note and debit note are generated. The product quantities are updated. An exception handler checks for browser errors at checkout time. When an error occurs, alternative code is executed.



- We are interested in modeling an online book ordering system for BookLib . The user accesses BookLib's website, then browses the catalog, and adds items to the cart. At the end, the user can 1) cancel, 2) remove books and continue adding or 3) checkout. Checkout then proceed to placing an order where the name, address, credit card details, etc. are require 7 Then a sales note, internal note and debit note are generated. The product quantities are updated. An exception handler checks for browser errors at checkout time. When an error occurs, alternative code is executed.



- We are interested in modeling an online book ordering system for BookLib . The user accesses BookLib's website, then browses the catalog, and adds items to the cart. At the end, the user can 1) cancel, 2) remove books and continue adding or 3) checkout. Checkout then proceed to placing an order where the name, address, credit card details, etc. are required. Then a sales note, internal note and debit note are generated. The product quantities are updated. An exception handler checks for browser errors at checkout time. When an error occurs, alternative code is executed.

8