KSRA Learning Academy
KSRA Learning System

E-course

What Is Object-Orientation?

Copyright © 2020 Kavian Scientific Research Association

E-course

SPECIFYING CONTROL

Copyright © 2020 Kavian Scientific Research Association

In This Lecture You Will Learn:

E-course

- how to identify requirements for control in an application;
- how to model object life cycles using state machines;
- how to develop state machine diagrams from interaction diagrams;
- how to model concurrent behaviour in an object;
- how to ensure consistency with other UML models.

Copyright © 2020 Kavian Scientific Research Association

State

E-course

- The current state of an object is determined by the current value of the object's attributes and the links that it has with other objects.

- For example the class StaffMember has an attribute startDate which determines whether a StaffMember object is in the probationary state.

Copyright © 2020 Kavian Scientific Research Association

State

E-course

- A state describes a particular condition that a modelled element (e.g. object) may occupy for a period of time while it awaits some event or *trigger*.
- The possible states that an object can occupy are limited by its class.
- Objects of some classes have only one possible state.
- Conceptually, an object remains in a state for an interval of time.

Copyright © 2020 Kavian Scientific Research Association

What Is a State?(cont'd)

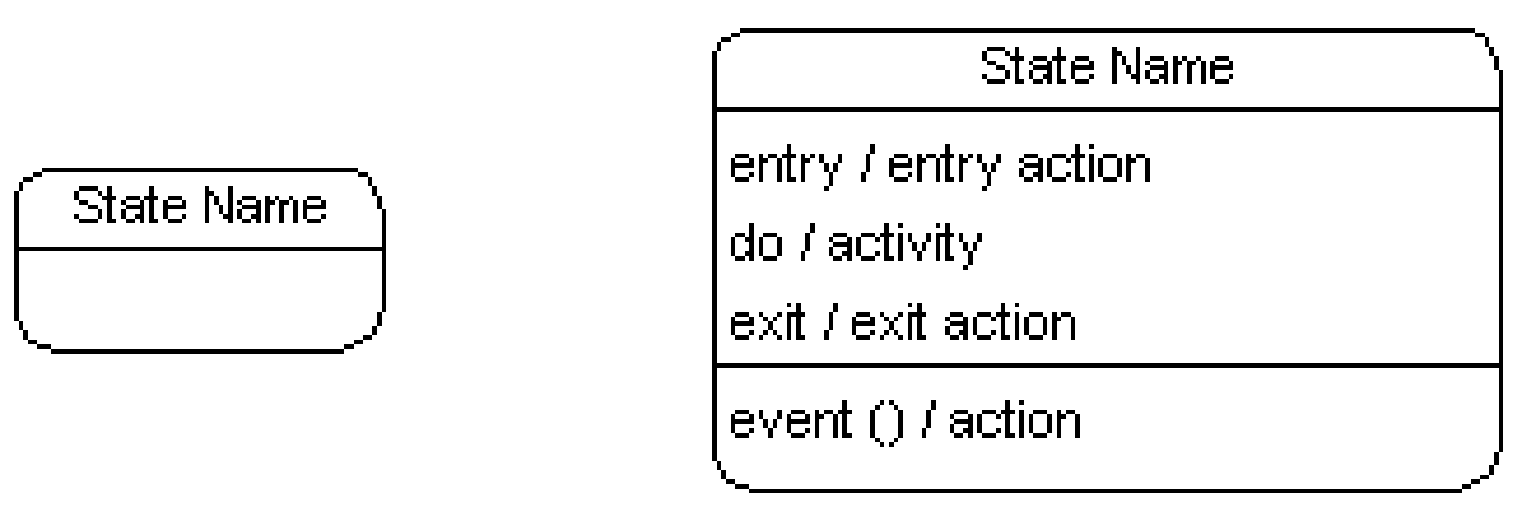
E-course

- There are several characteristics of states:
  - A state occupies an interval of time.
  - A state is often associated with an abstraction of attribute values of an entity satisfying some condition(s).
  - An entity changes its state not only as a direct consequence of the current input, but it is also dependent on some past history of its inputs.

Copyright © 2020 Kavian Scientific Research Association

UML Notation

E-course



Copyright © 2020 Kavian Scientific Research Association

UML Notation (cont'd)

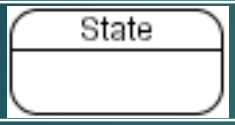
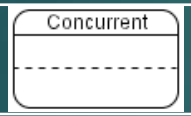
E-course

Action or activity	Description
entry/ action 1; …; action n	Upon entry to the state, the specified actions are performed.
exit/ action 1; …; action n	Upon exit from the state, the specified actions are performed.
do/ activity	The specified activity is performed continuously while in this state.
event-name(parameters) [guard-condition] / action 1 ; …; action n	An internal transition is fired when the specified event occurs and the specified guard condition is true. The specified actions are performed when the transition is fired.

Copyright © 2020 Kavian Scientific Research Association

UML Notation (cont'd)

E-course

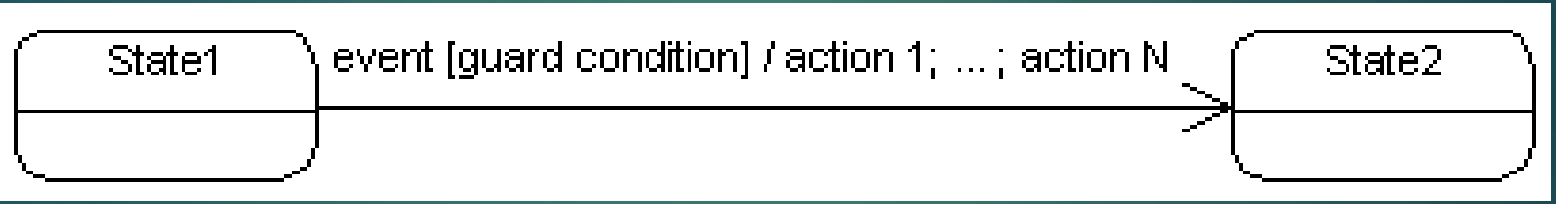
	Initial state
	Final state
	State
	History state
	Junction state
	Concurrent composite state
	Transition

Copyright © 2020 Kavian Scientific Research Association

Transition

E-course

- A transition from one state to another takes place instantaneously in response to some external events or internal stimuli.



Copyright © 2020 Kavian Scientific Research Association

Transition (cont'd)

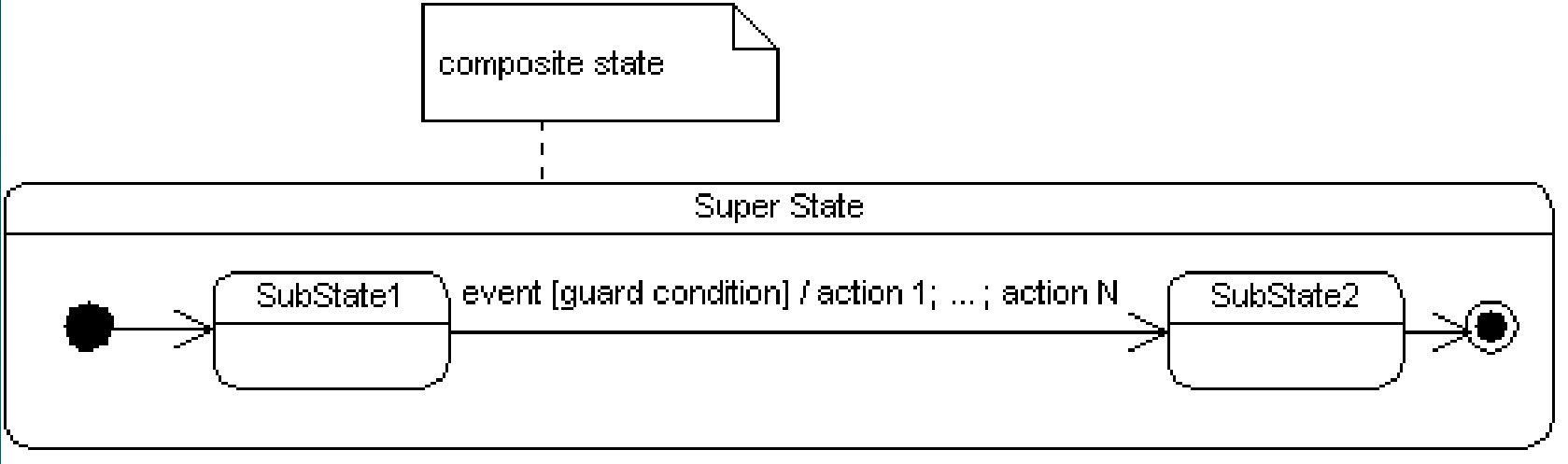
E-course

- A transition is fired when the following conditions are satisfied:
  - The entity is in the state of the source state.
  - An event specified in the label occurs.
  - The guard condition specified in the label is evaluated to be true.
- When a transition is fired, the actions associated with it are executed.

Copyright © 2020 Kavian Scientific Research Association

Composite State

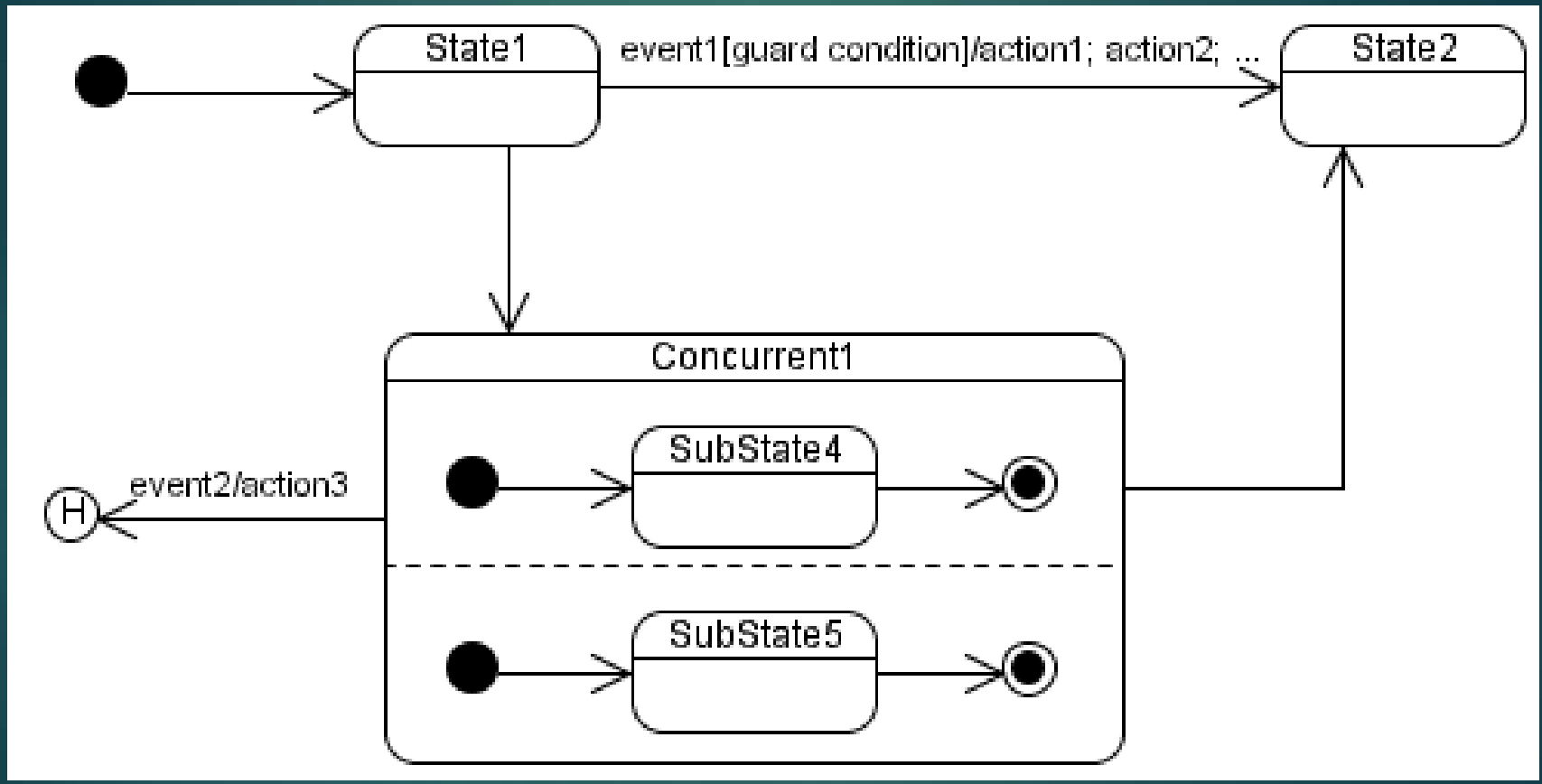
E-course



Copyright © 2020 Kavian Scientific Research Association

Composite State (cont'd)

E-course



Copyright © 2020 Kavian Scientific Research Association

state machine

E-course

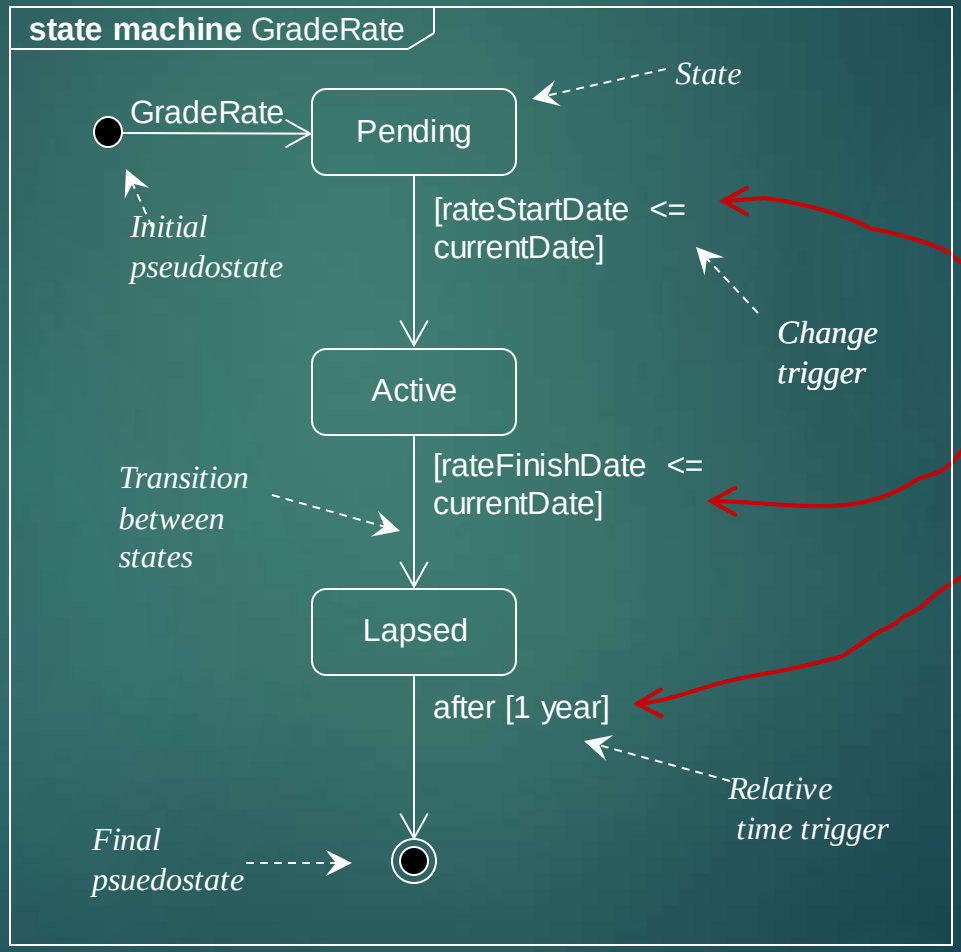
- The current state of a GradeRate object can be determined by the two attributes rateStartDate and rateFinishDate.
- An enumerated state variable may be used to hold the object state, possible values would be Pending, Active or Lapsed.

Copyright © 2020 Kavian Scientific Research Association

state machine

E-course

state machine for the class GradeRate.



Movement from one state to another is dependent upon events that occur with the passage of time.

Copyright © 2020 Kavian Scientific Research Association

Types of Event

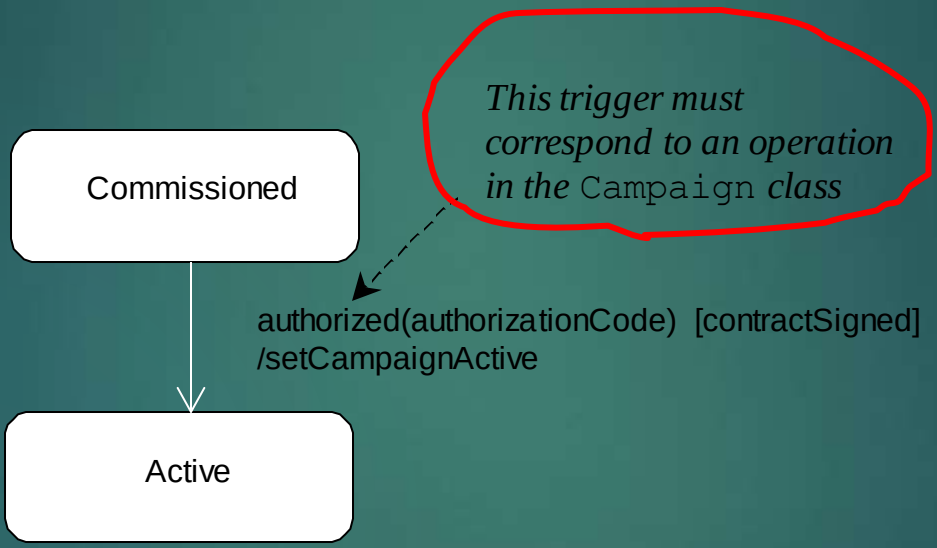
E-course

- A *change trigger* occurs when a condition becomes true.
- A *call trigger* occurs when an object receives a call for one of its operations either from another object or from itself.
- A *signal trigger* occurs when an object receives a signal (an asynchronous communication).
- An *relative-time trigger* is caused by the passage of a designated period of time after a specified event (frequently the entry to the current state).

Copyright © 2020 Kavian Scientific Research Association

Event

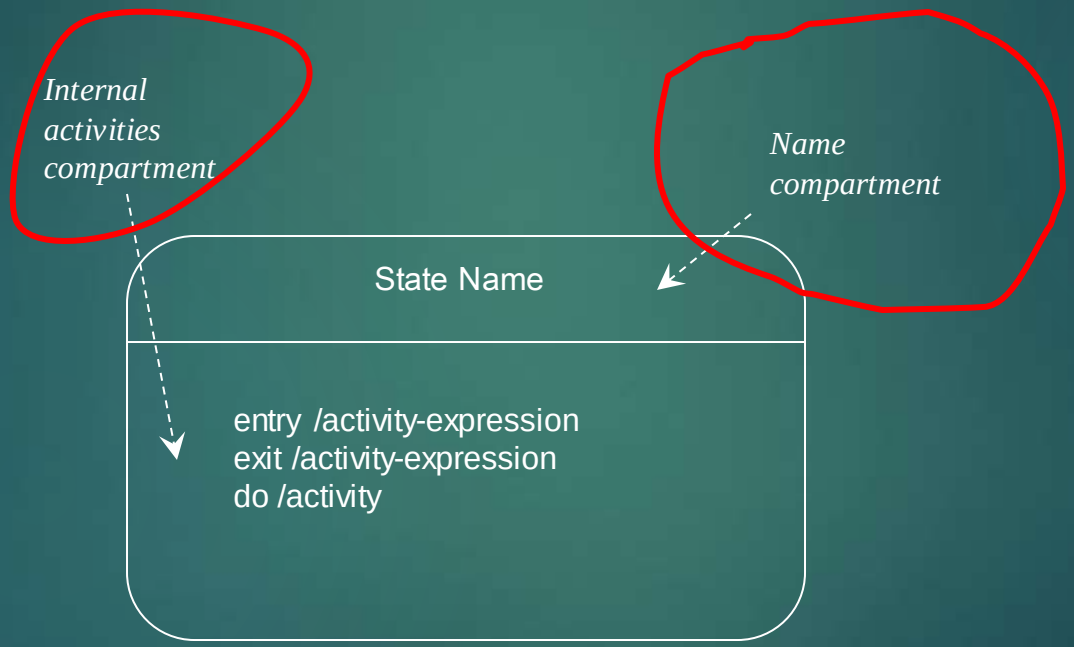
E-course



Copyright © 2020 Kavian Scientific Research Association

Internal Activities

E-course



Copyright © 2020 Kavian Scientific Research Association

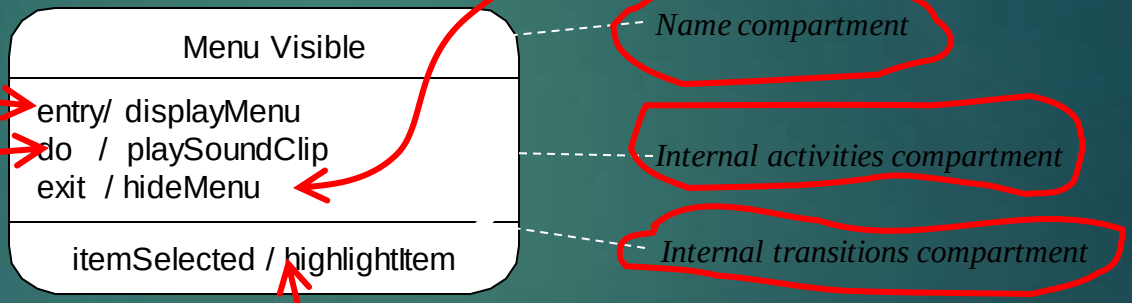
‘Menu Visible’ State

E-course

`Menu Visible` *state for a* `DropDownMenu` *object.*

entry action causes the menu to be displayed

Exiting the state triggers `hideMenu()`



While the object remains in the `Menu Visible` state, the activity causes a sound clip to be played.

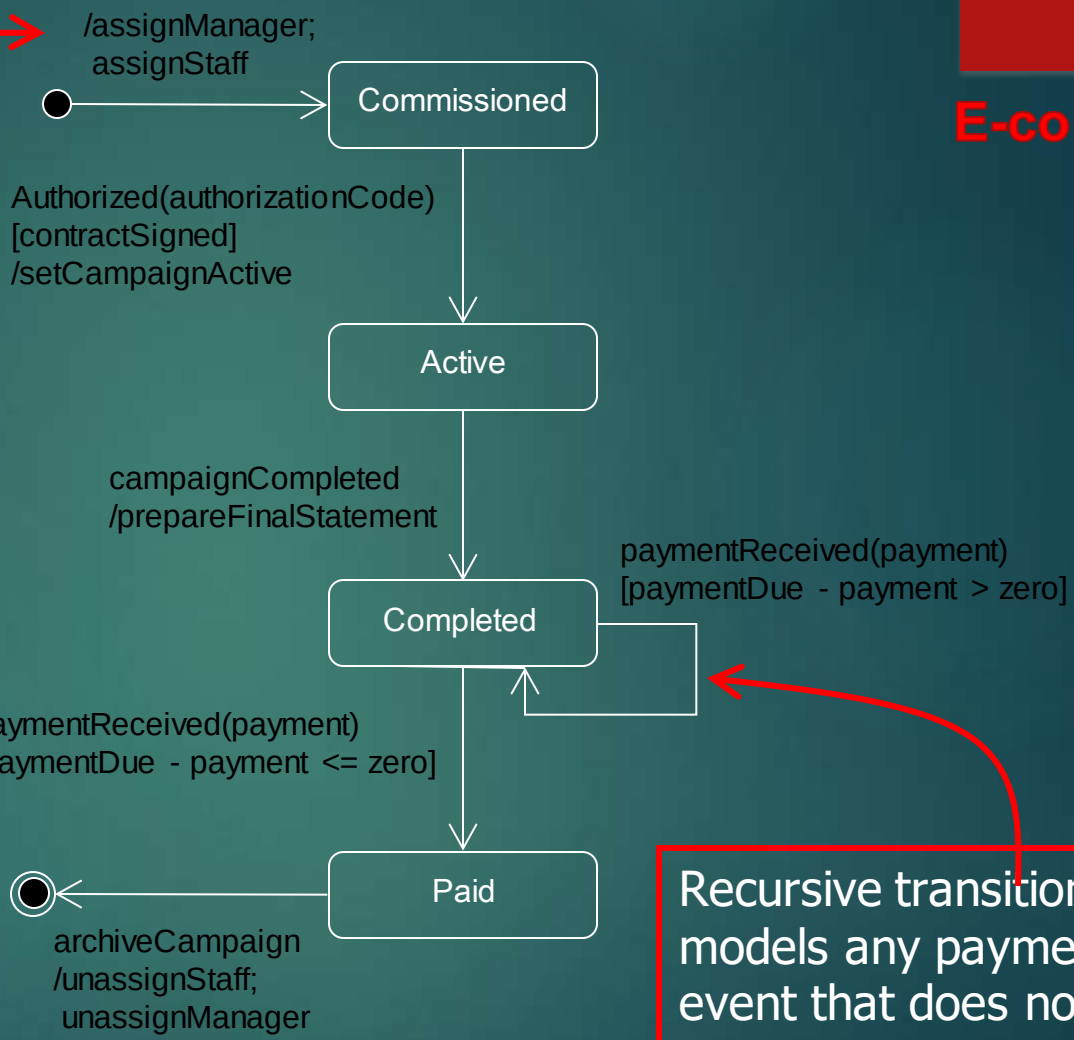
event `itemSelected()` triggers the action `highlightItem()`

Copyright © 2020 Kavian Scientific Research Association

Action-expression assigning manager and staff on object creation

state machine for the class **Campaign**.

Guard condition ensuring complete payment before entering `Paid`

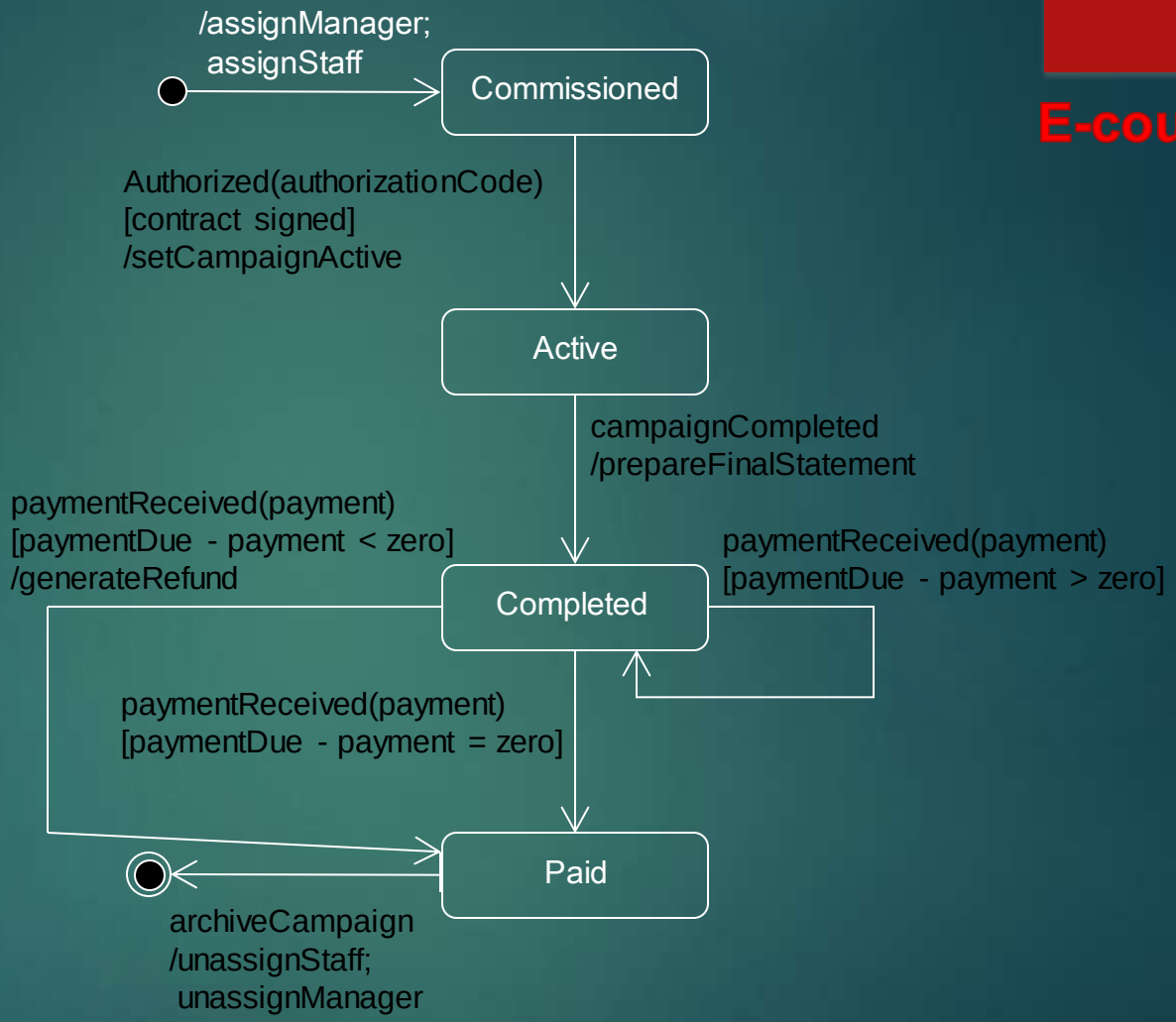


Recursive transition models any payment event that does not reduce the amount due to zero or beyond.

Copyright © 2020 Kavian Scientific Research Association

E-course

A revised state machine for the class `Campaign`



If the user requirements were to change, so that an overpayment is now to result in the automatic generation of a refund, a new transition is added.

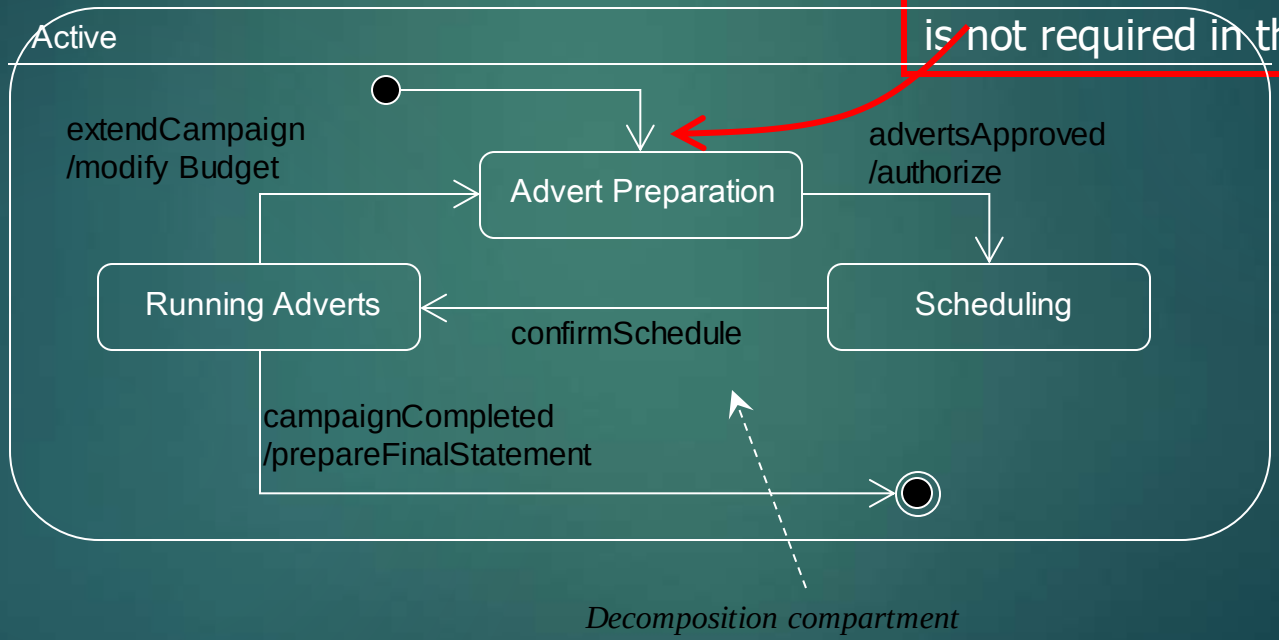
Copyright © 2020 Kavian Scientific Research Association

Nested Substates

E-course

The **`Active`** *state of* **`Campaign`** *showing nested substates.*

The transition from the initial pseudostate symbol should not be labelled with an event but may be labelled with an action, though it is not required in this example



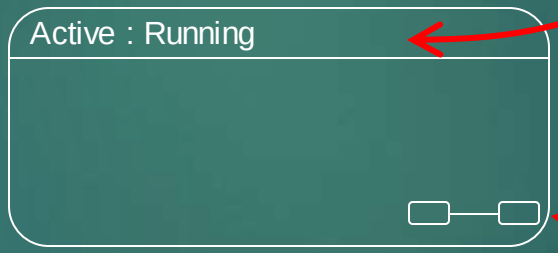
Copyright © 2020 Kavian Scientific Research Association

Nested States

E-course

The submachine `Running` is referenced using the include statement.

The **`Active`** *state of* **`Campaign`** *with the detail hidden.*

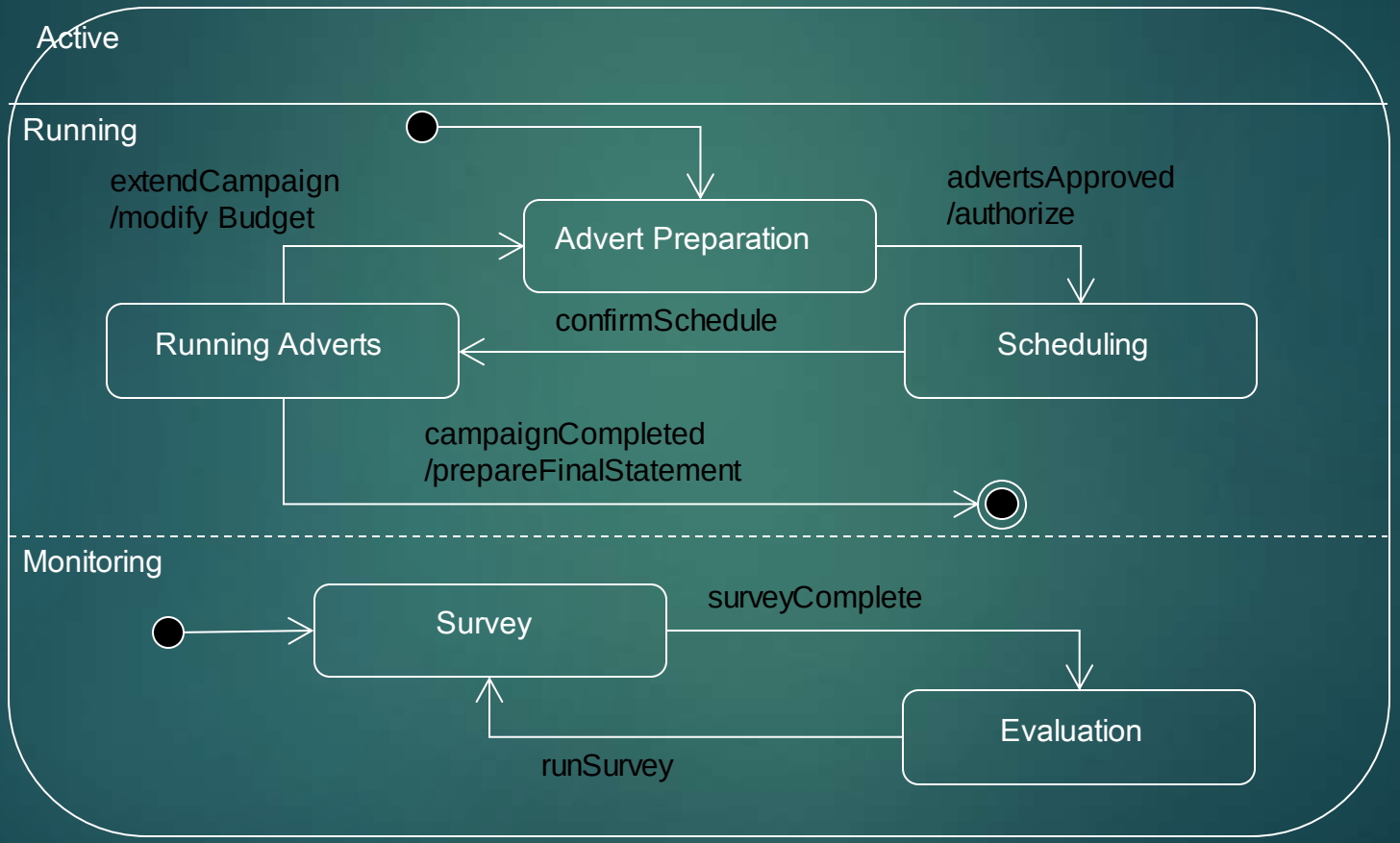


Hidden decomposition indicator icon

Copyright © 2020 Kavian Scientific Research Association

The Active state with concurrent substates.

E-course



Copyright © 2020 Kavian Scientific Research Association

Concurrent States

E-course

- A transition to a complex state is equivalent to a simultaneous transition to the initial states of each concurrent state machine.
- An initial state must be specified in both nested state machines in order to avoid ambiguity about which substate should first be entered in each concurrent region.
- A transition to the Active state means that the Campaign object simultaneously enters the Advert Preparation and Survey states.

Copyright © 2020 Kavian Scientific Research Association

Concurrent States

E-course

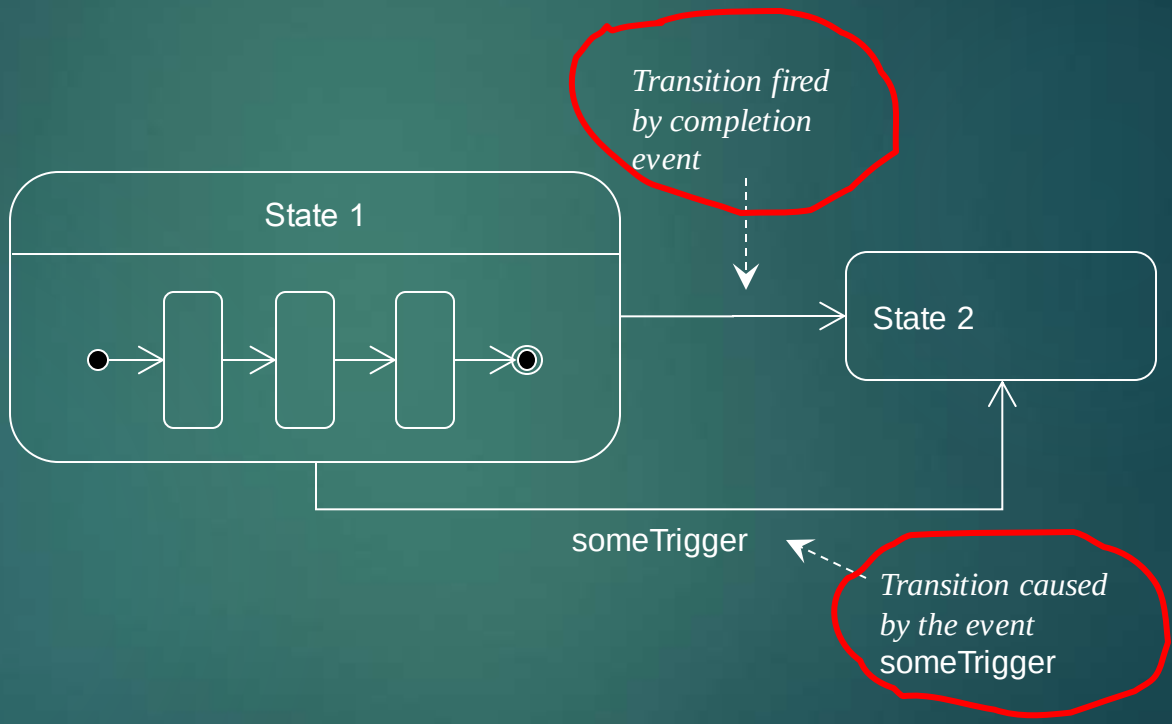
- Once the composite state is entered a transition may occur within either concurrent region without having any effect on the state in the other concurrent region.

- A transition out of the Active state applies to all its substates (no matter how deeply nested).

Copyright © 2020 Kavian Scientific Research Association

Completion Event

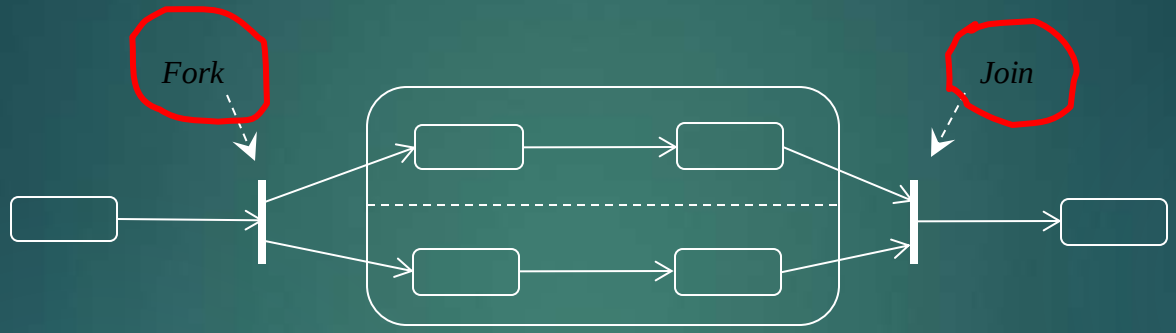
E-course



Copyright © 2020 Kavian Scientific Research Association

Synchronized Concurrent Threads.

E-course



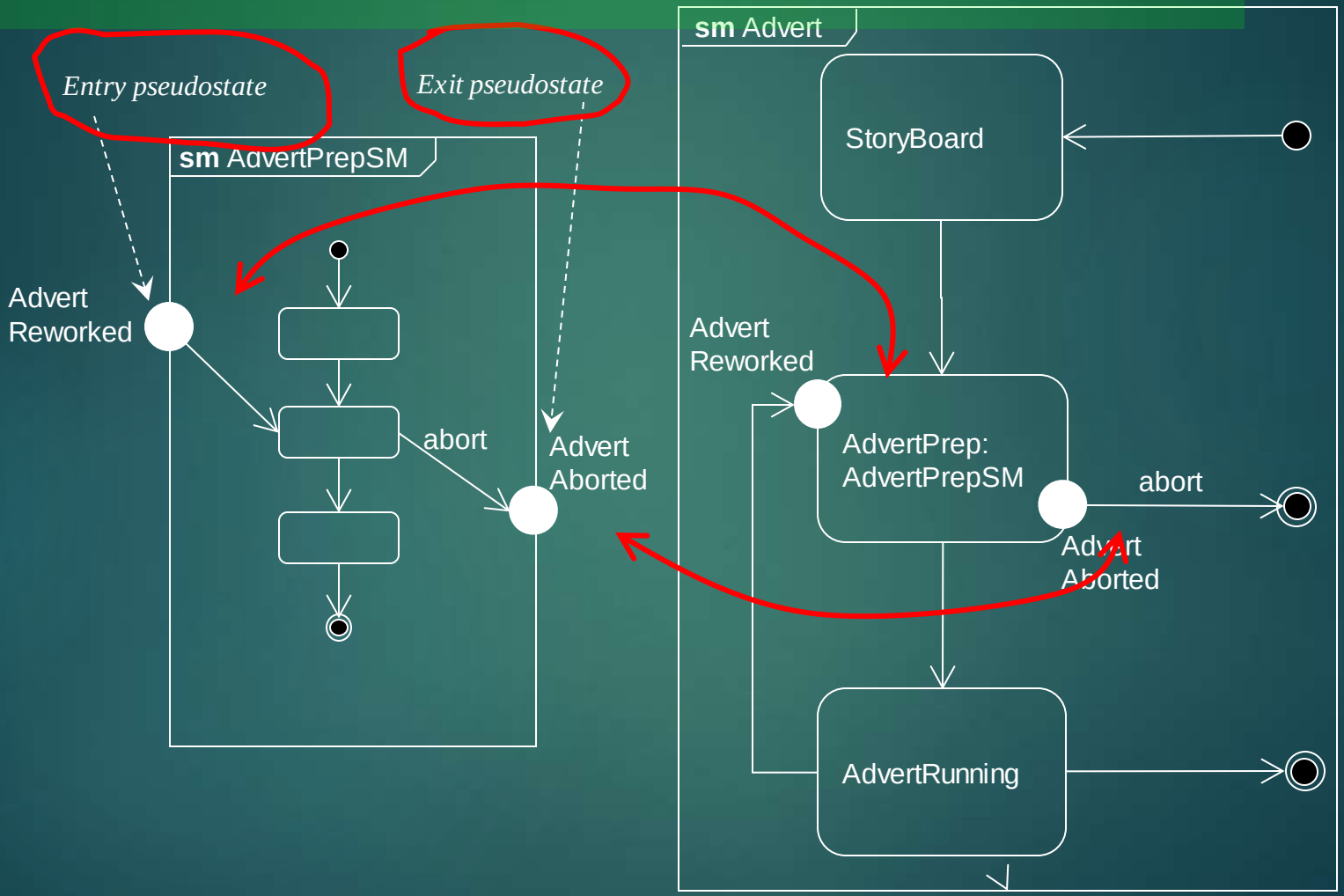
- Explicitly showing how an event triggering a transition to a state with nested concurrent states causes specific concurrent substates to be entered.

- Shows that the composite state is not exited until both concurrent nested state machines are exited.

Copyright © 2020 Kavian Scientific Research Association

Entry & Exit Pseudostates

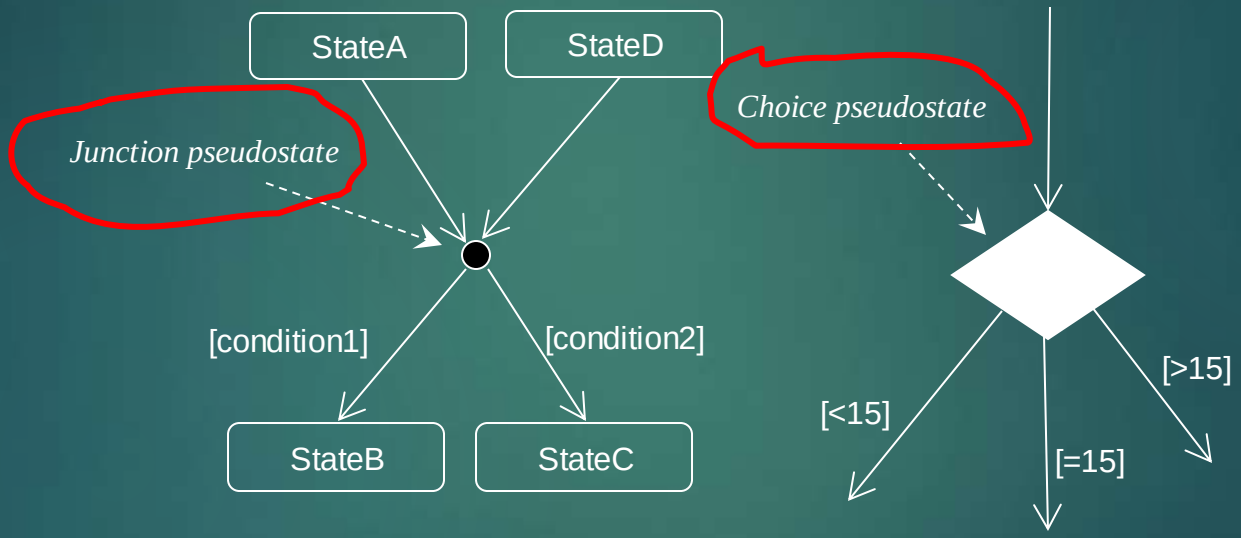
E-course



Copyright © 2020 Kavian Scientific Research Association

Junction & Choice Pseudostates

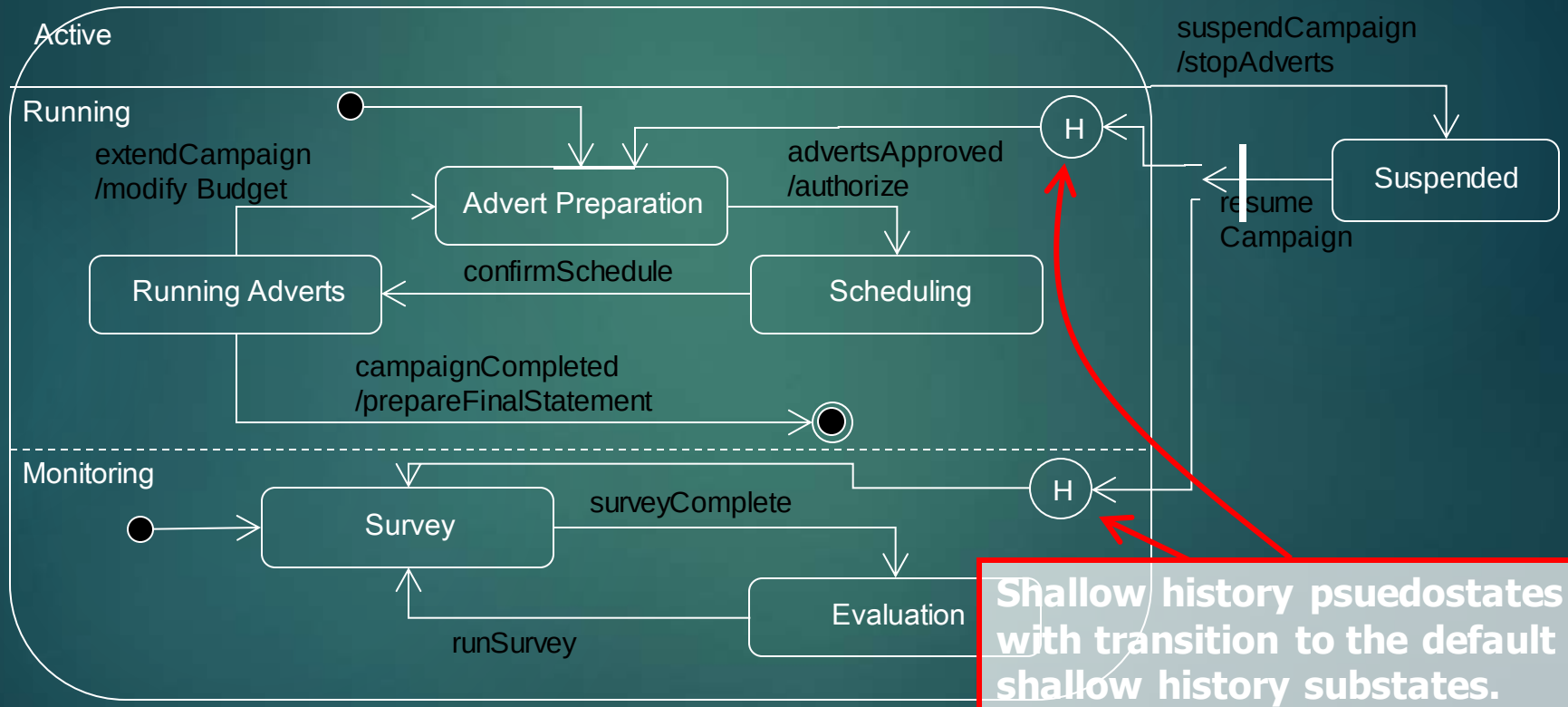
E-course



Copyright © 2020 Kavian Scientific Research Association

History Pseudostates

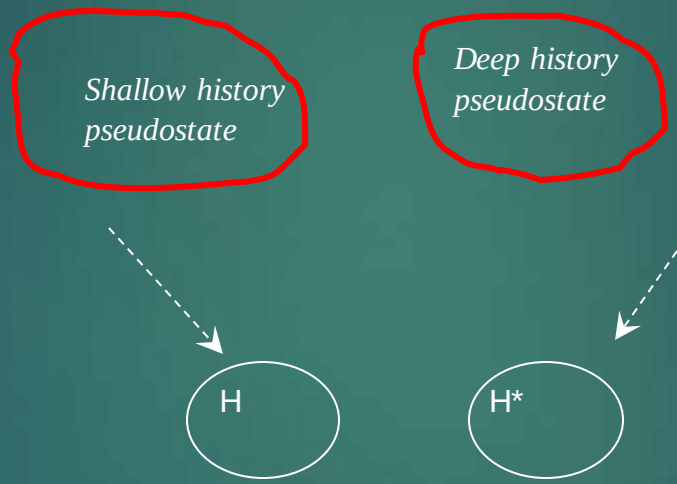
E-course



Copyright © 2020 Kavian Scientific Research Association

History Pseudostates

E-course



Copyright © 2020 Kavian Scientific Research Association

Preparing state machines

E-course

- Two approaches may be used:
  - Behavioural approach
  - Life cycle approach

Allen and Frost (1998)

Copyright © 2020 Kavian Scientific Research Association

Behavioural Approach

E-course

1. Examine all interaction diagrams that involve each class that has heavy messaging.
2. Identify the incoming messages on each interaction diagram that may correspond to events. Also identify the possible resulting states.
3. Document these events and states on a state machine.
4. Elaborate the state machine as necessary to cater for additional interactions as these become evident, and add any exceptions.

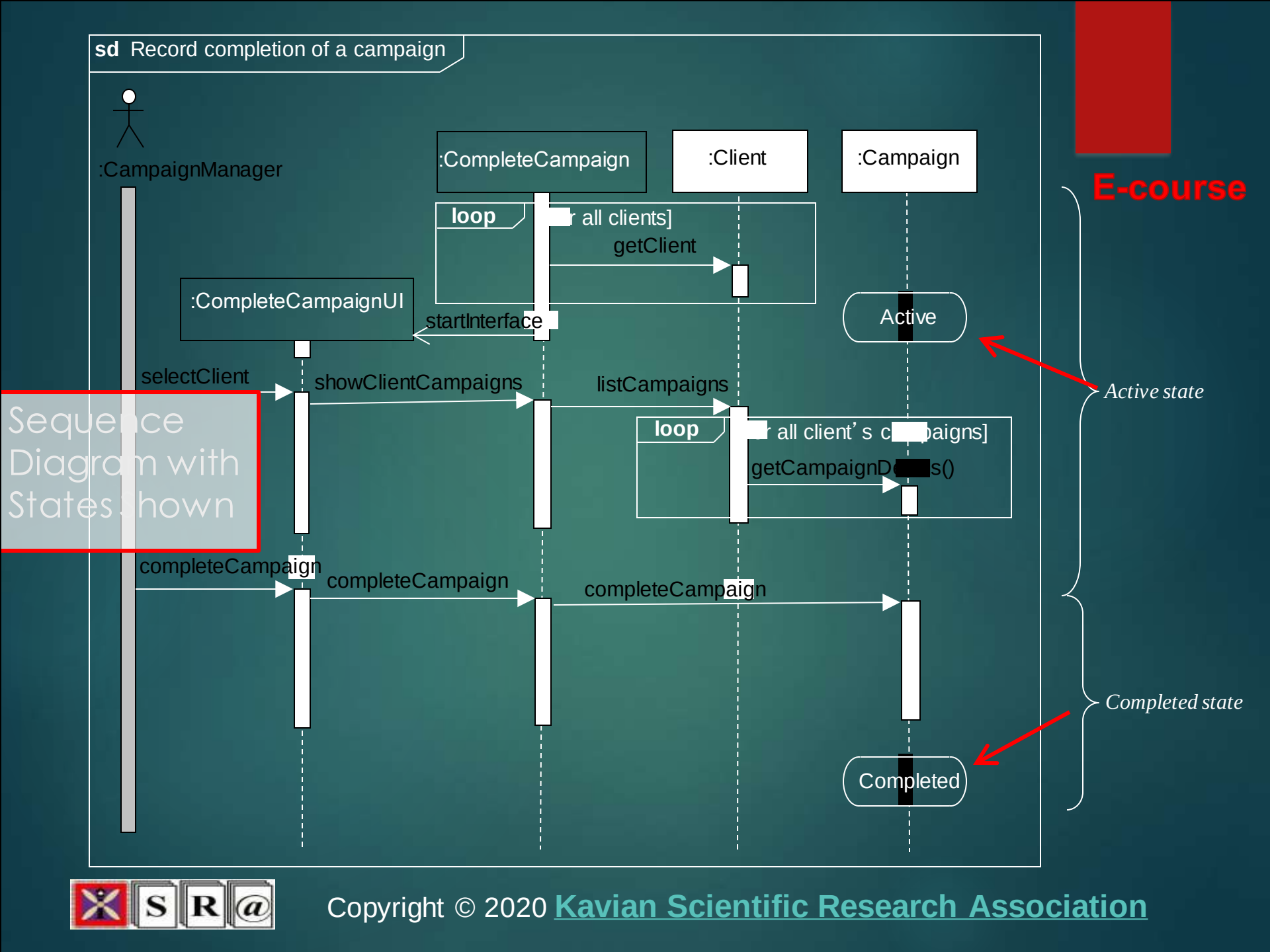
Copyright © 2020 Kavian Scientific Research Association

Behavioural Approach

E-course

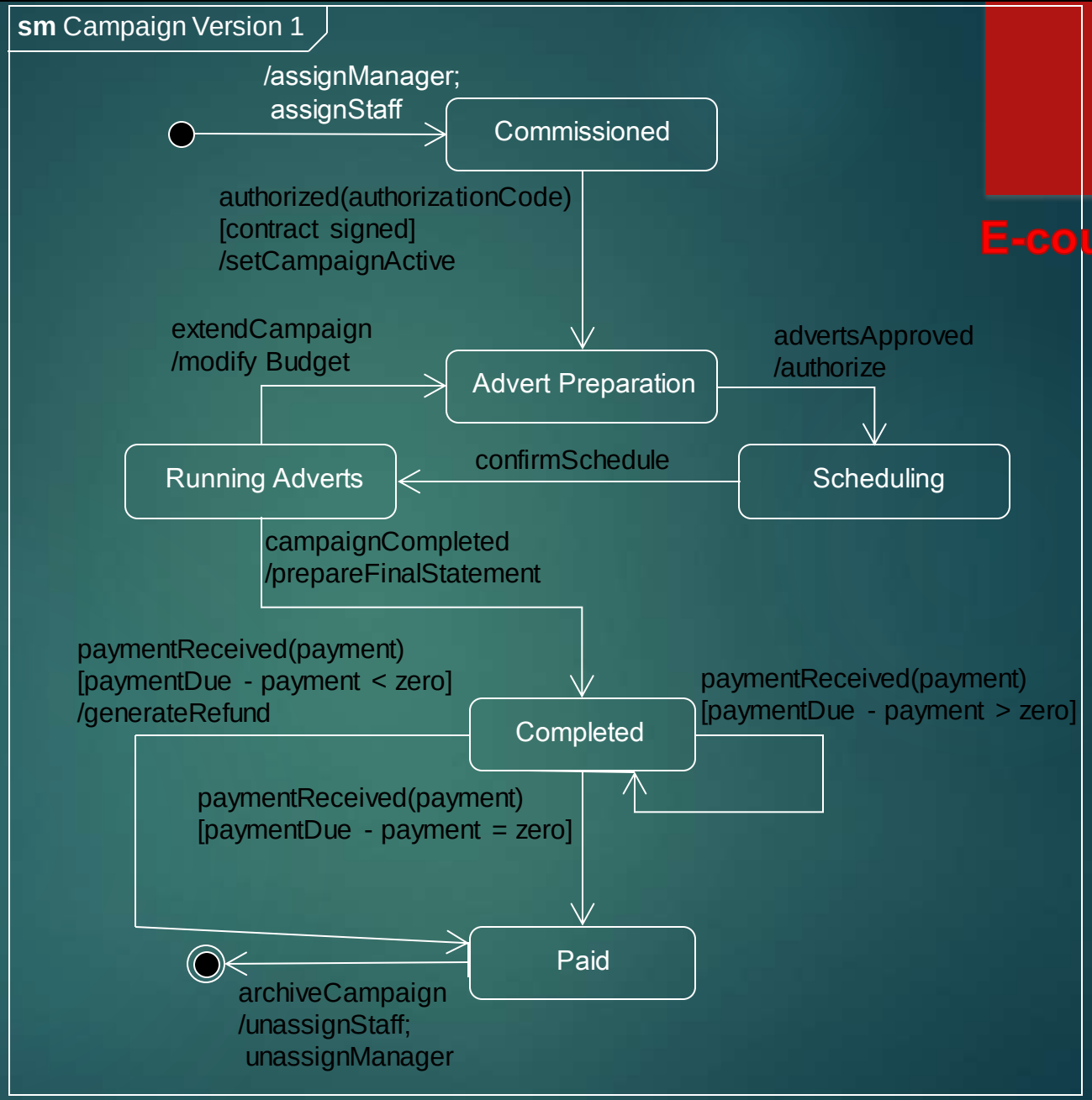
5. Develop any nested state machines (unless this has already been done in an earlier step).
6. Review the state machine to ensure consistency with use cases. In particular, check that any constraints that are implied by the state machine are appropriate.
7. Iterate steps 4, 5 and 6 until the state machine captures the necessary level of detail.
8. Check the consistency of the state machine with the class diagram, with interaction diagrams and with any other state machines and models.

Copyright © 2020 Kavian Scientific Research Association



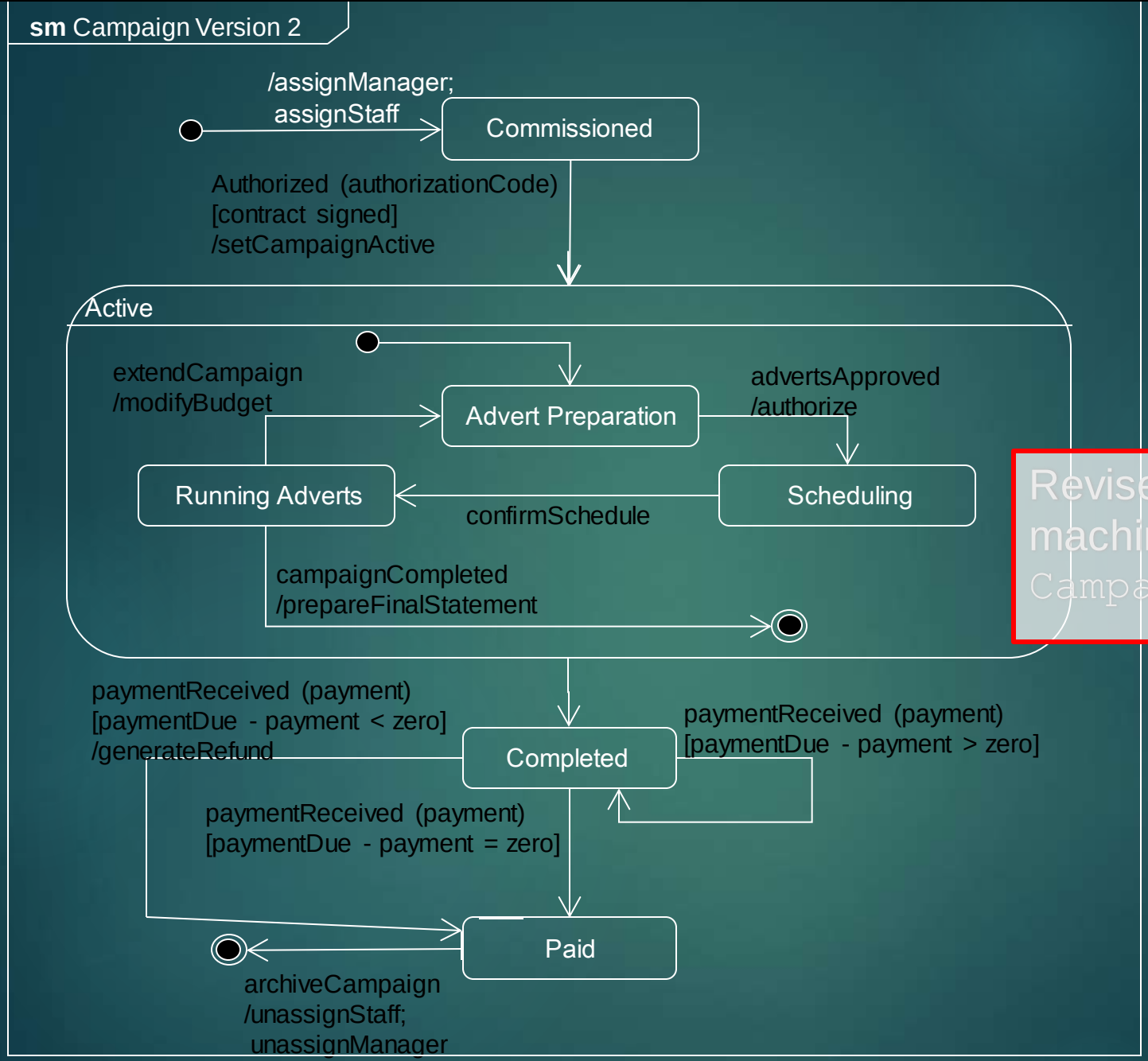
Copyright © 2020 Kavian Scientific Research Association

Initial state machine for the Campaign class—a behavioural approach.



E-course

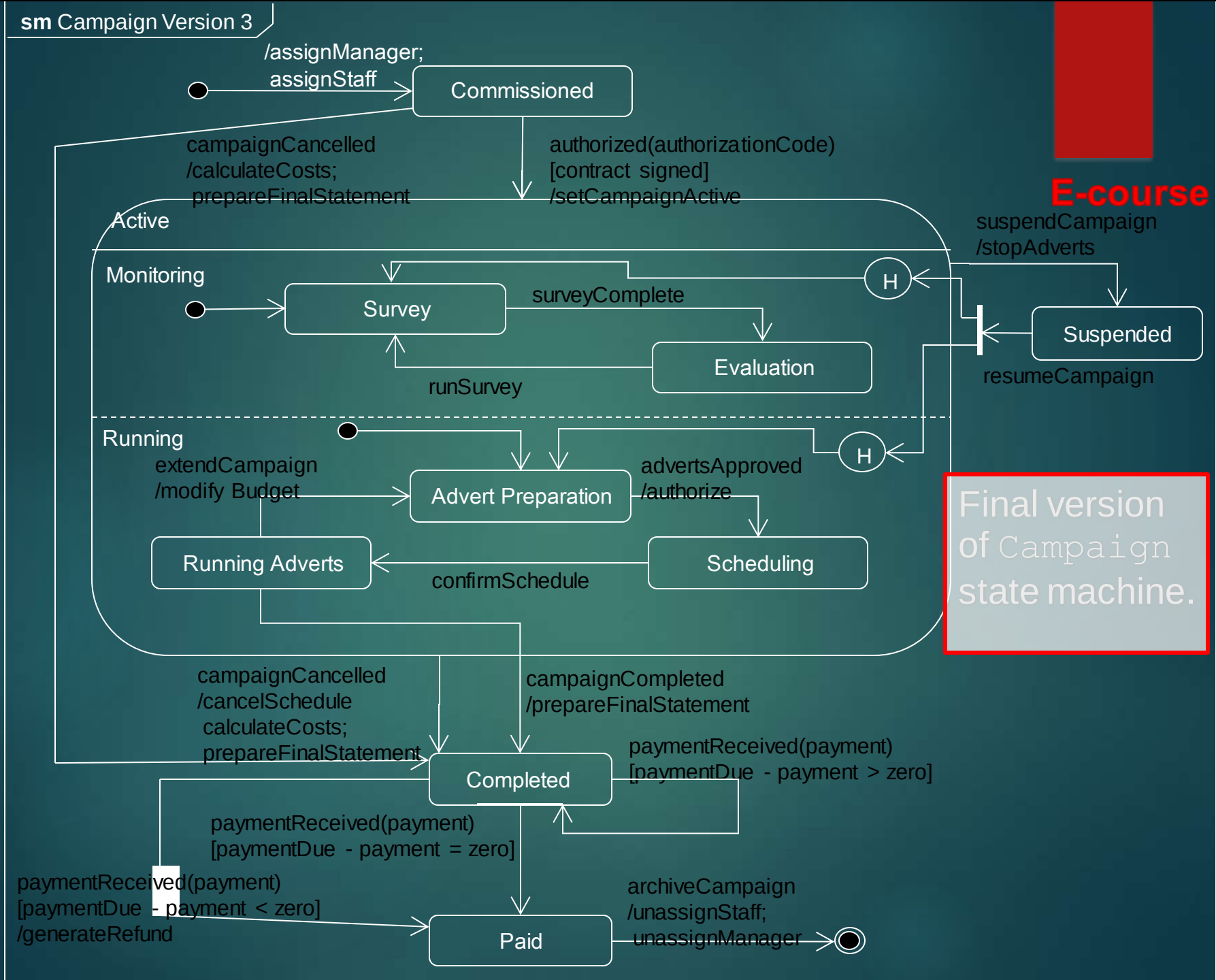
Copyright © 2020 Kavian Scientific Research Association



E-course

Revised state machine for the `Campaign` class.

Copyright © 2020 Kavian Scientific Research Association


sm Campaign Version 3
/assignManager;
assignStaff
Commissioned
campaignCancelled
/calculateCosts;
prepareFinalStatement
authorized(authorizationCode)
[contract signed]
/setCampaignActive
Active
suspendCampaign
/stopAdverts
Monitoring
H
Survey
surveyComplete
Suspended
Evaluation
runSurvey
resumeCampaign
Running
H
extendCampaign
/modify Budget
Advert Preparation
advertsApproved
/authorize
Running Adverts
Scheduling
confirmSchedule
campaignCancelled
/cancelSchedule
calculateCosts;
prepareFinalStatement
campaignCompleted
/prepareFinalStatement
paymentReceived(payment)
[paymentDue - payment > zero]
Completed
paymentReceived(payment)
[paymentDue - payment = zero]
paymentReceived(payment)
[paymentDue - payment < zero]
/generateRefund
archiveCampaign
/unassignStaff;
unassignManager
Paid
E-course
Final version of Campaign state machine.

Life Cycle Approach

E-course

- Consider the life cycles for objects of each class.
- Events and states are identified directly from use cases and from any other requirements documentation that happens to be available.
- First, the main system events are listed.
- Each event is then examined in order to determine which objects are likely to have a state dependent response to it.

Copyright © 2020 Kavian Scientific Research Association

Life Cycle Approach Steps

E-course

1. Identify major system events.

2. Identify each class that is likely to have a state dependent response to these events.

3. For each of these classes produce a first-cut state machine by considering the typical life cycle of an instance of the class.

4. Examine the state machine and elaborate to encompass more detailed event behaviour.

Copyright © 2020 Kavian Scientific Research Association

Life Cycle Approach Steps

E-course

5. Enhance the state machine to include alternative scenarios.
6. Review the state machine to ensure that is consistent with the use cases. In particular, check that the constraints that the state machine implies are appropriate.
7. Iterate through steps 4, 5 and 6 until the state machine captures the necessary level of detail.
8. Ensure consistency with class diagram and interaction diagrams and other state machines.

Copyright © 2020 Kavian Scientific Research Association

Life Cycle Approach

E-course

- Less formal than the behavioural approach in its initial identification of events and relevant classes.
- Often helpful to use a combination of the two, since each provides checks on the other.

Copyright © 2020 Kavian Scientific Research Association

Protocol State Machines

E-course

- UML 2.0 introduces a distinction between protocol and behavioural state machines.
- All the state machines so far have been behavioural.
- Protocol state machines differ in that they only show all the legal transitions with their pre- and post-conditions.

Copyright © 2020 Kavian Scientific Research Association

Protocol State Machines

E-course

- The states of a protocol state machine cannot have
  - entry, exit or do activity sections
  - deep or shallow history states
- All transitions must be protocol transitions

Copyright © 2020 Kavian Scientific Research Association

Protocol State Machines

E-course

- The syntax for a protocol transition label is as follows.

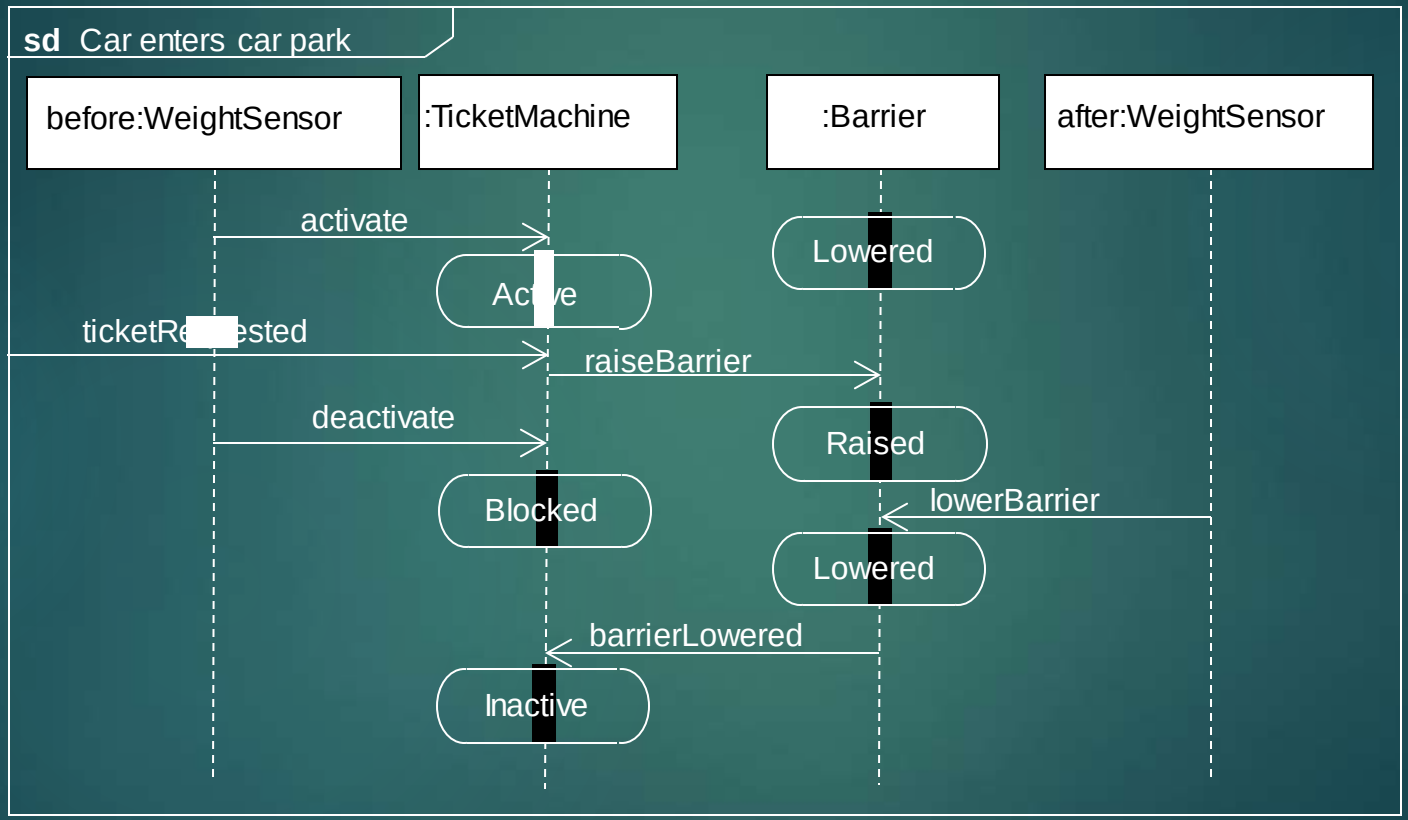
'[' pre-condition ']' trigger '/' '[' post-condition ']'

- Unlike behavioural transitions protocol transitions do not have activity expressions.

Copyright © 2020 Kavian Scientific Research Association

Sequence Diagram for Protocol State Machine Example

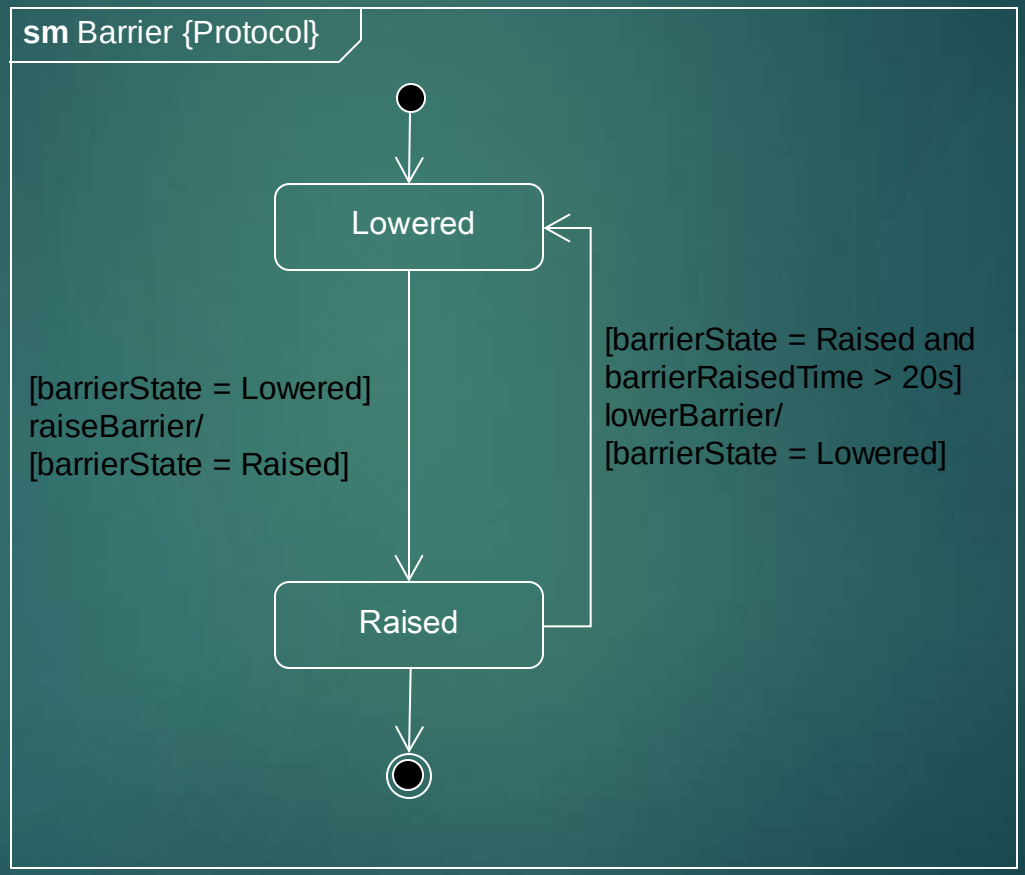
E-course



Copyright © 2020 Kavian Scientific Research Association

Protocol State Machines

E-course



Copyright © 2020 Kavian Scientific Research Association

Consistency Checking

E-course

- Every event should appear as an incoming message for the appropriate object on an interaction diagram(s).
- Every action should correspond to the execution of an operation on the appropriate class, and perhaps also to the despatch of a message to another object.
- Every event should correspond to an operation on the appropriate class (but note that not all operations correspond to events).
- Every outgoing message sent from a state machine must correspond to an operation on another class.

Copyright © 2020 Kavian Scientific Research Association

Consistency Checking

E-course

- Consistency checks are an important task in the preparation of a complete set of models.
- Highlights omissions and errors, and encourages the clarification of any ambiguity or incompleteness in the requirements.

Copyright © 2020 Kavian Scientific Research Association

Implementing State Diagrams

E-course

- A state diagram is typically used to model the dynamic behavior of a subsystem, a control object or even an entity object. Like activity diagrams, there are two approaches to implement a state diagram:
  - Using the location within a program to hold the state (for implementing active object or entity).
  - Using an explicit attribute to hold the state (for implementing inactive object or entity).

Copyright © 2020 Kavian Scientific Research Association

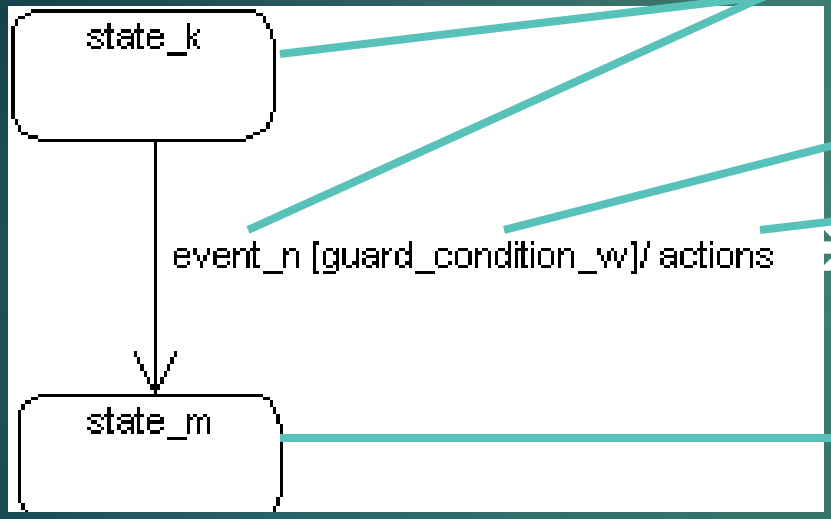
State Diagrams (cont'd)

- The second approach is suitable for implementing the state diagram of an inactive entity. We can implement the state diagram by applying the techniques below:
  - Map the state diagram on to a class.
  - Add a state attribute for storing the state information.
  - Map an event to a method and embed all required state transitions and actions of the event in the method.

Copyright © 2020 Kavian Scientific Research Association

State Diagrams (cont'd)

E-course




```
public void event_n(….) {
  switch (state) {
    case state_k:
      if (guard_condition_w) {
        state = state_m;
        perform actions of the transition;
      }
      break;
    case state_v:

      …
  }
}
```

Copyright © 2020 Kavian Scientific Research Association

State Diagrams (cont'd)

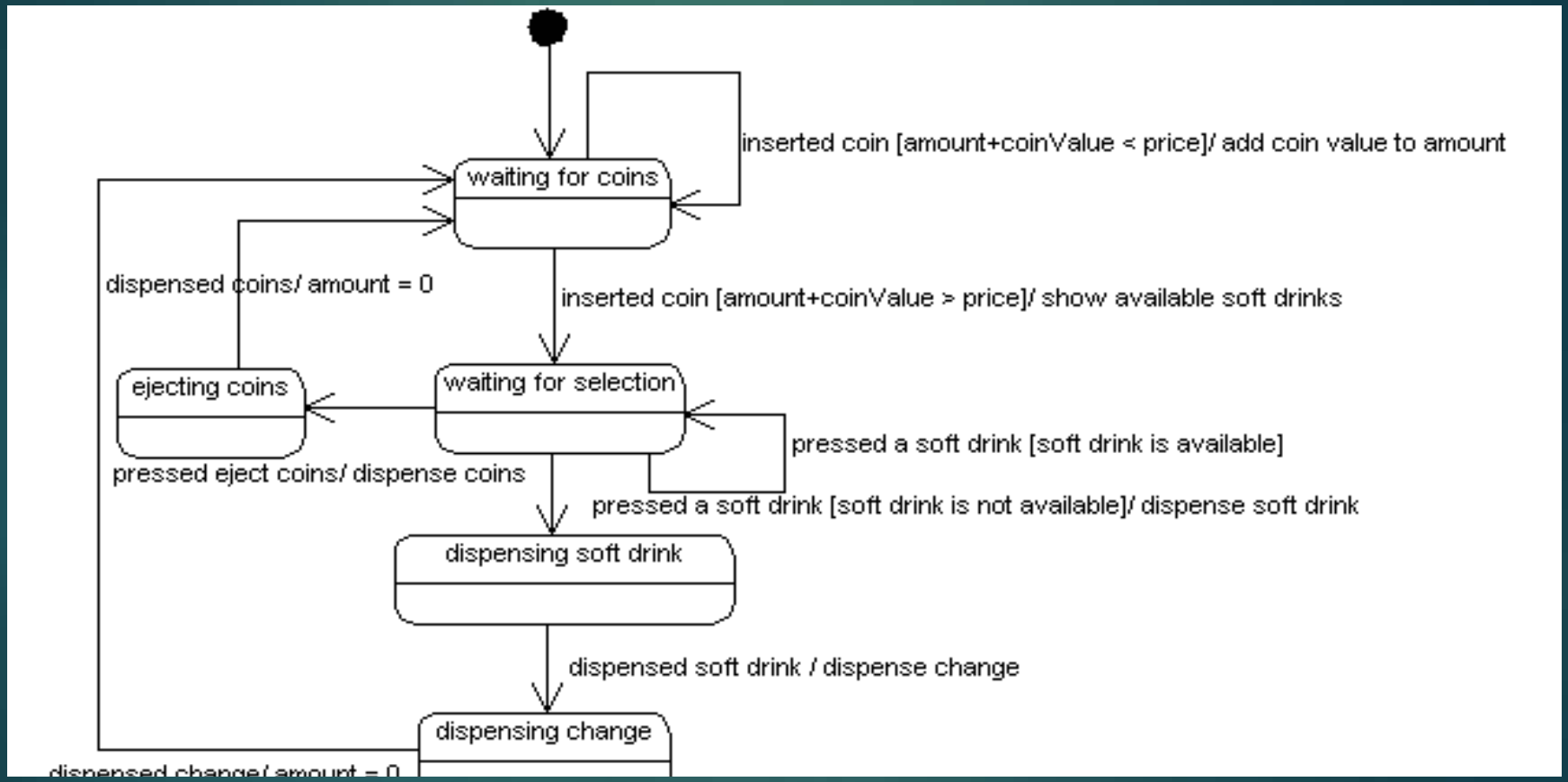
E-course

- For a composite state with sequential substates, we can create a nested (inner) class for implementing the sequential substates. The parent state machine can then invoke the method of the nested class for handling transitions within the nested state diagram. Another way to implement the composite state is to transform the parent state diagram to eliminate the composite state so that it becomes a flat level state diagram.
- For a composite state with concurrent substates, we can create a nested class for implementing each substate. The implementation is similar to that for nested state diagrams. The composite state is exited when all the concurrent substates reach their final states.

Copyright © 2020 Kavian Scientific Research Association

Example – Control Object of Vending Machine

E-course



Copyright © 2020 Kavian Scientific Research Association

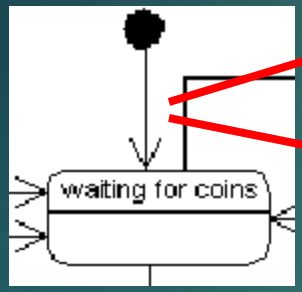
Example – Control Object of Vending Machine (cont'd)


```
class VendingMachineControl
{
  int _state;
  float _amount, _price;
  static final int WaitingCoin = 1;
  static final int WaitingSelection = 2;
  static final int DispensingSoftDrink = 3;
  static final int DispensingChange = 4;
  static final int EjectingCoins = 5;
```

Copyright © 2020 Kavian Scientific Research Association

Example – Control Object of Vending Machine (cont'd)

E-course

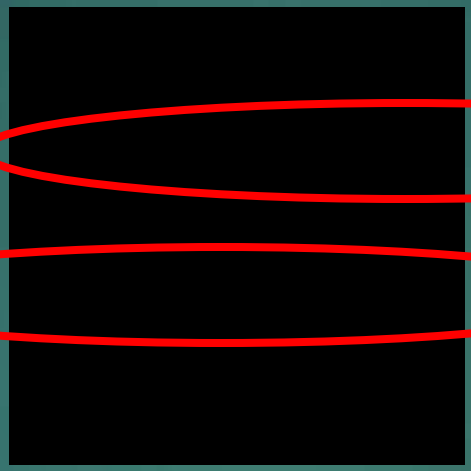



```
public VendingMachineControl(float price)
  {
    _amount = 0;
    _state = WaitingCoin;
    _price = price;
  }
```

Copyright © 2020 Kavian Scientific Research Association

Example – Control Object of Vending Machine (cont'd)

E-course

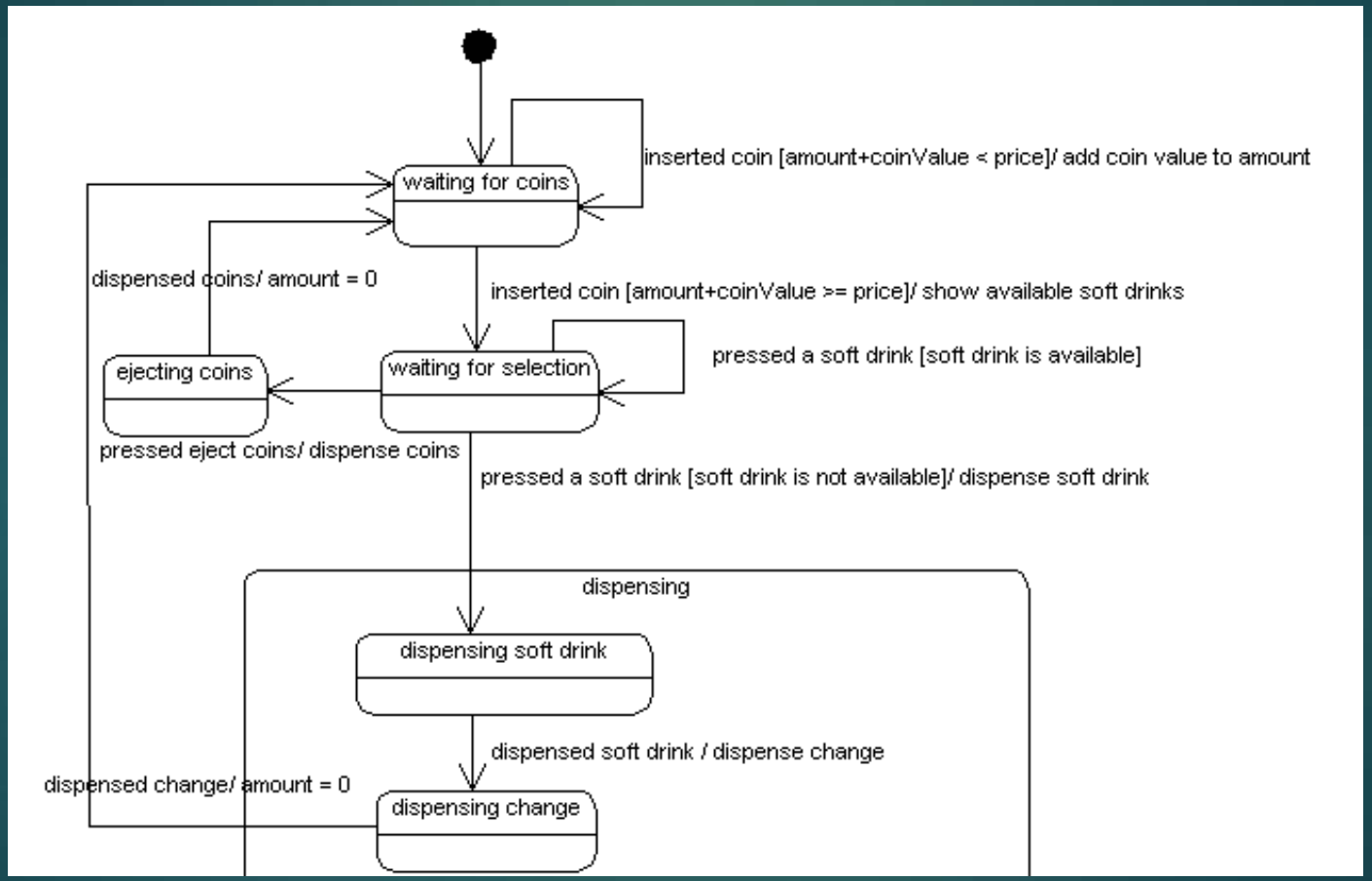


```
public void insertedCoin(float coinValue)
 {
   if (state == WaitingCoin)
   {
     amount += coinValue;
     if (amount >= price) {  // fire transition
       state = WaitingSelection;
       show available soft drinks;
     }
   }
 } // insertedCoin
```

Copyright © 2020 Kavian Scientific Research Association

Example – Implementing a State Diagram with Sequential Substates

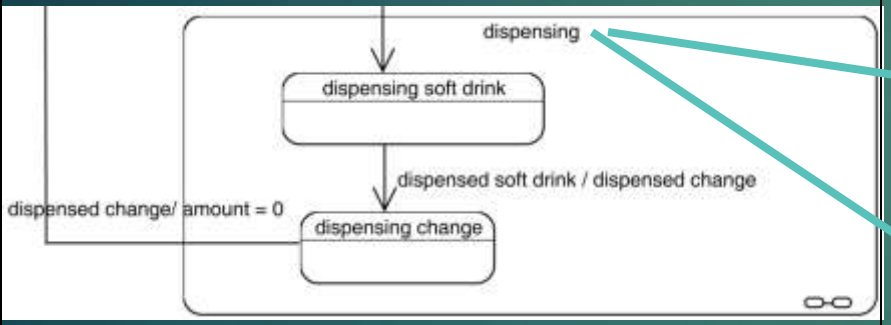
E-course



Copyright © 2020 Kavian Scientific Research Association

Example – Implementing a State Diagram with Sequential Substates (cont'd)

E-course

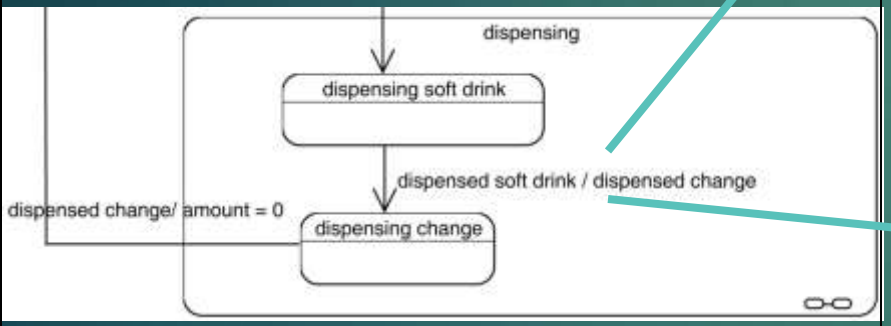



```
class dispenseControl {
  int _state;
  static final int DispensingSoftDrink = 1;
  static final int DispensingChange = 2;
  static final int Complete = 3;
  public dispenseControl()
  {
    _state = DispensingSoftDrink;
  }
```

Copyright © 2020 Kavian Scientific Research Association

Example – Implementing a State Diagram with Sequential Substates (cont'd)

E-course

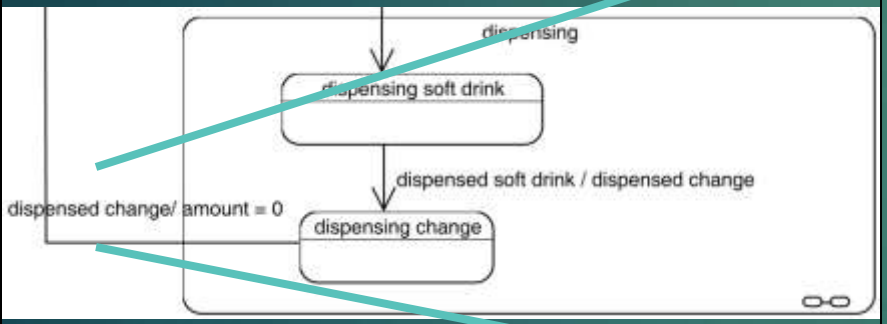



```
public boolean dispensedSoftDrink()
    {
      if (_state == DispensingSoftDrink) {
        _state = DispensingChange;
        dispense change;
      }
      return false;
    }
```

Copyright © 2020 Kavian Scientific Research Association

Example – Implementing a State Diagram with Sequential Substates (cont'd)

E-course




```
public boolean dispensedChange()
    {
      if (_state == DispensingChange) {
        _state = Complete;
        return true;
      }
      return false;
    }
```

Copyright © 2020 Kavian Scientific Research Association

Example – Implementing a State Diagram with Sequential Substates (cont'd)

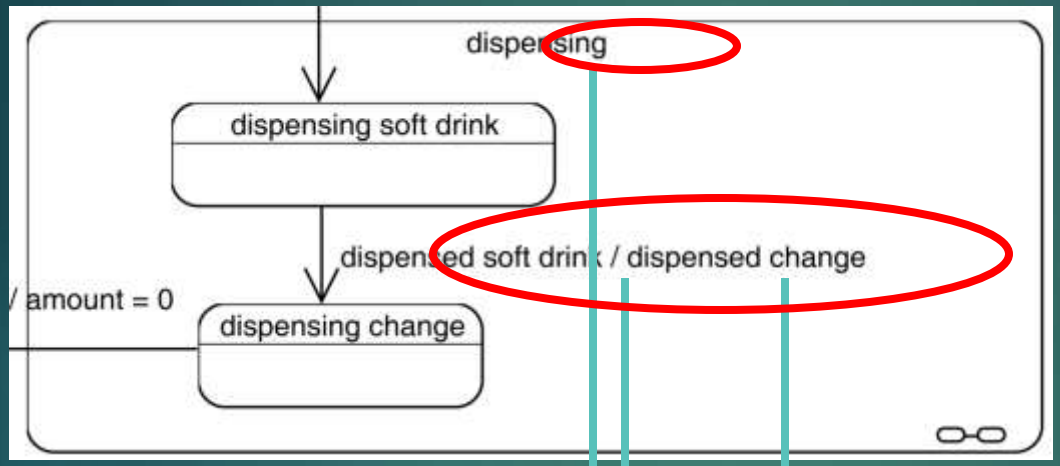
E-course

```
class VendingMachineControl
{
..declaration of state attribute, constants, other attributes;
  declaration of inner class dispenseControl;
..public VendingMachineControl(float price)
  {
    _amount = 0;
    _state = WaitingCoin;
    _price = price;
    _substate = new DispenseControl();
  }
```

Copyright © 2020 Kavian Scientific Research Association

Example – Implementing a State Diagram with Sequential Substates (cont'd)

E-course

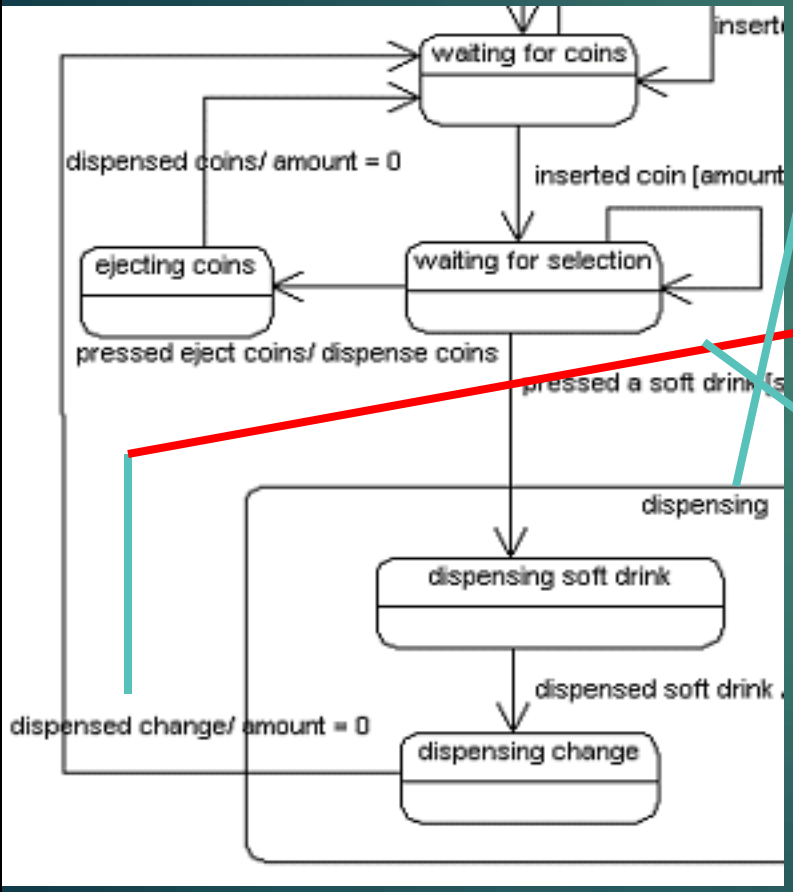



```
public void dispensedSoftDrink() // VendingMachineControl
 {
   if (_state == Dispensing) {
     boolean isComplete = _substate.dispensedSoftDrink();
   }
 }
```

Copyright © 2020 Kavian Scientific Research Association

Example – Implementing a State Diagram with Sequential Substates (cont'd)

E-course




```
// VendingMachineControl
public boolean dispensedChange()
  {
    if (_state == Dispensing) {
      boolean isComplete =
_substate.dispensedChange();
      if (isComplete) {
        amount = 0;
        _state = WaitingCoin;
      }
    }
  }
```

Copyright © 2020 Kavian Scientific Research Association

References

E-course

- UML 2.0 Superstructure Specification (OMG, 2004)
- Allen and Frost (1998)

(For full bibliographic details, see Bennett, McRobb and Farmer)

- Object-Oriented Technology - From Diagram to Code with Visual Paradigm for UML, Curtis H.K. Tsang, Clarence S.W. Lau and Y.K. Leung, McGraw-Hill Education (Asia), 2005

Copyright © 2020 Kavian Scientific Research Association

WE FOCUS ON KNOWLEDGE-BASED ON EDUCATION

KSRA of Empowerment is a global non-profit organization committed to bringing empowerment through education by utilizing innovative mobile technology and educational research from experts and scientists. KSRA emerged in 2012 as a catalytic force to reach the hard to reach populations worldwide through Learning management system & E-learning & mobile learning.

The KSRA team partners with local under-served communities around the world to improve the access to and quality of knowledge based on education, amplify and augment learning programs where they exist, and create new opportunities for e-learning where traditional education systems are lacking or non-existent.

E-course

Copyright © 2020 Kavian Scientific Research Association