Static and Dynamic Modeling using UML

Some material adapted from Lethbridge & Laganiere;
Some material adapted from Craig Larman.

Review of Object Orientation

- Basic Concepts
  - Classes
    - Attributes
    - Operations (Methods)
  - Objects
  - Relationships
- Basic Principles
  - Abstraction
  - Encapsulation
  - Modularity
  - Hierarchy

UML diagrams

- Class diagrams
  - describe classes and their **relationships**
- Interaction diagrams
  - show the **behaviour** of systems in terms of how objects interact with each other
- State diagrams and activity diagrams
  - show how systems behave **internally**
- Component and deployment diagrams
  - show how the various components of systems are **arranged** logically and physically

What constitutes a good model?

- A model should
  - use a standard notation
  - be **understandable** by clients and users
  - lead software engineers to have insights about the system
  - provide **abstraction**

- Models are used:
  - to help create designs
  - to **permit analysis** and review of those designs.
  - as the core documentation describing the system.

Essentials of UML Class Diagrams

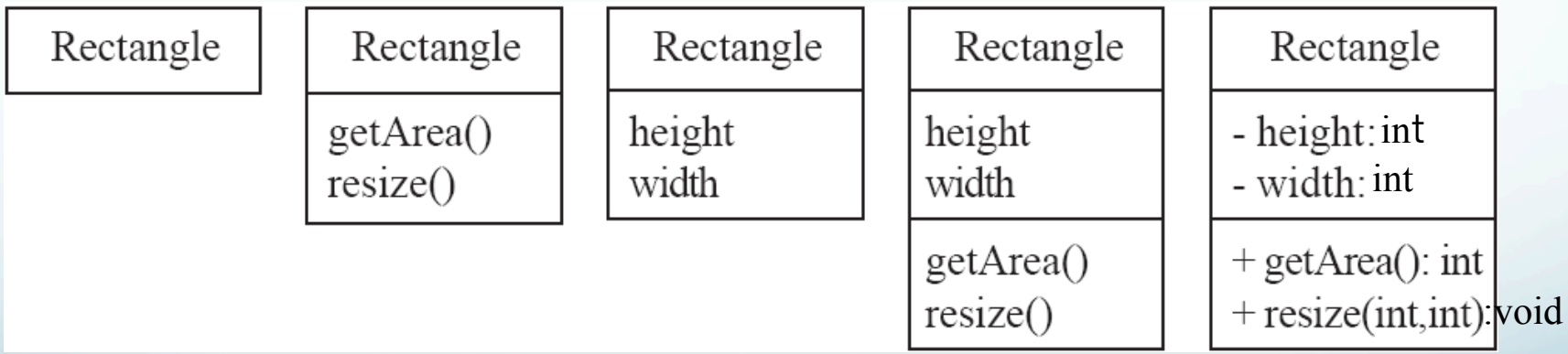
- *The main symbols shown on class diagrams are:*
  - *Classes*
    - represent the types of data themselves
  - *Associations*
    - represent linkages between instances of classes
  - *Attributes*
    - are simple data found in classes and their instances
  - *Operations*
    - represent the functions performed by the classes and their instances
  - *Generalizations*
    - group classes into inheritance hierarchies

Classes

A class is simply represented as a box with the name of the class inside

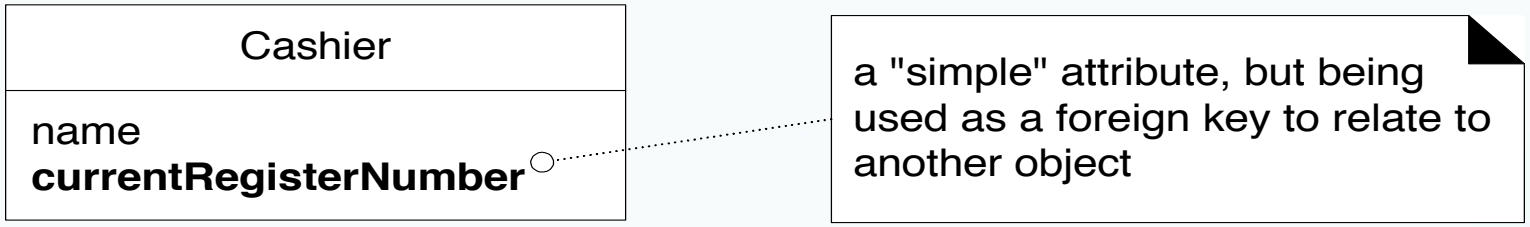
- The diagram may also show the attributes and operations
- The complete signature of an operation is:

  operationName(parameterName: parameterType ...): returnType

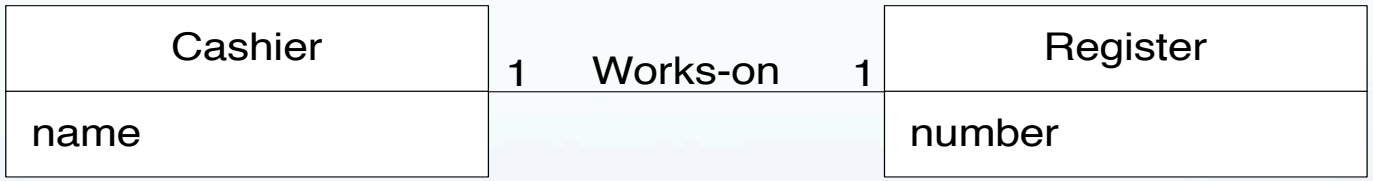


Do not use attributes to represent foreign keys

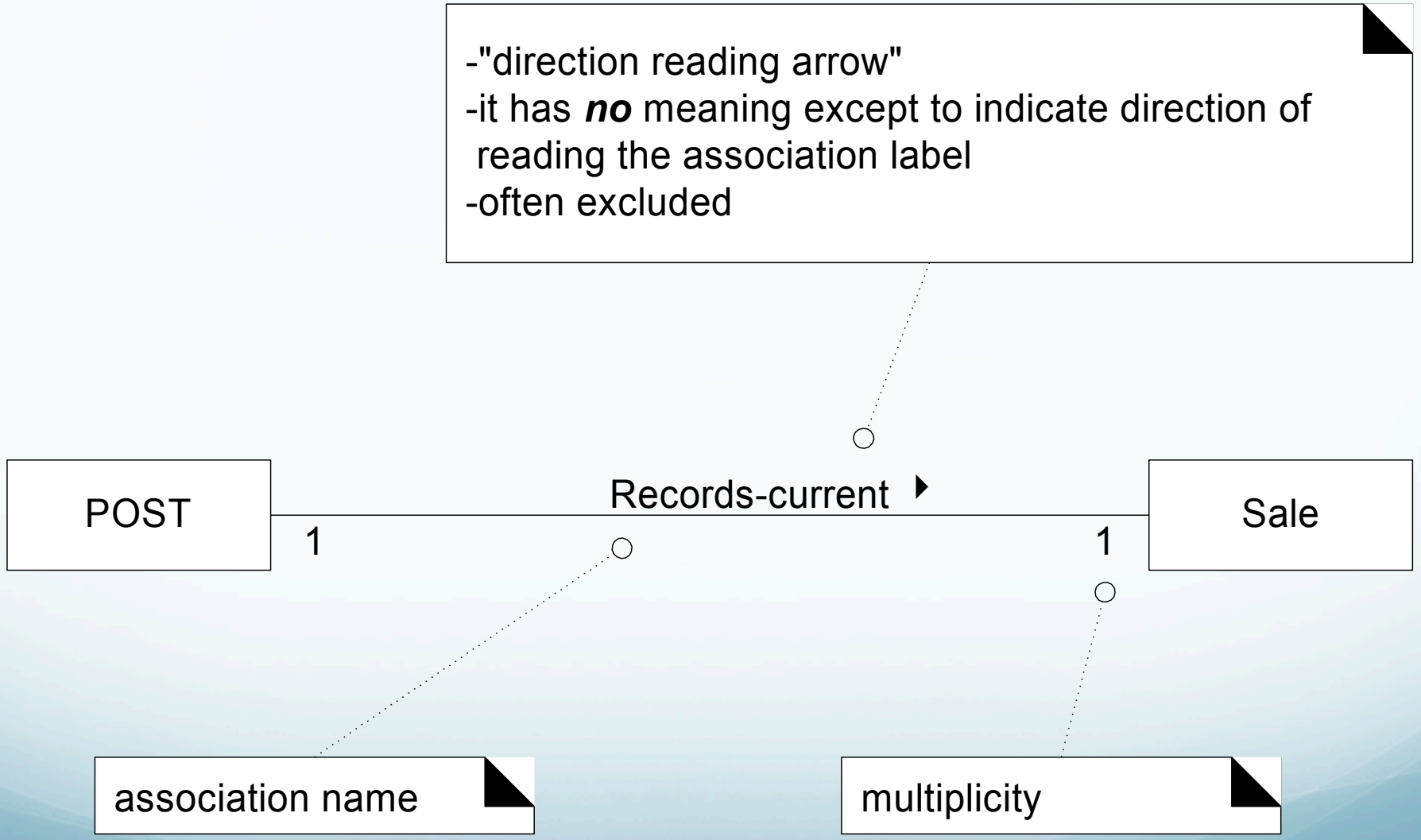
Worse



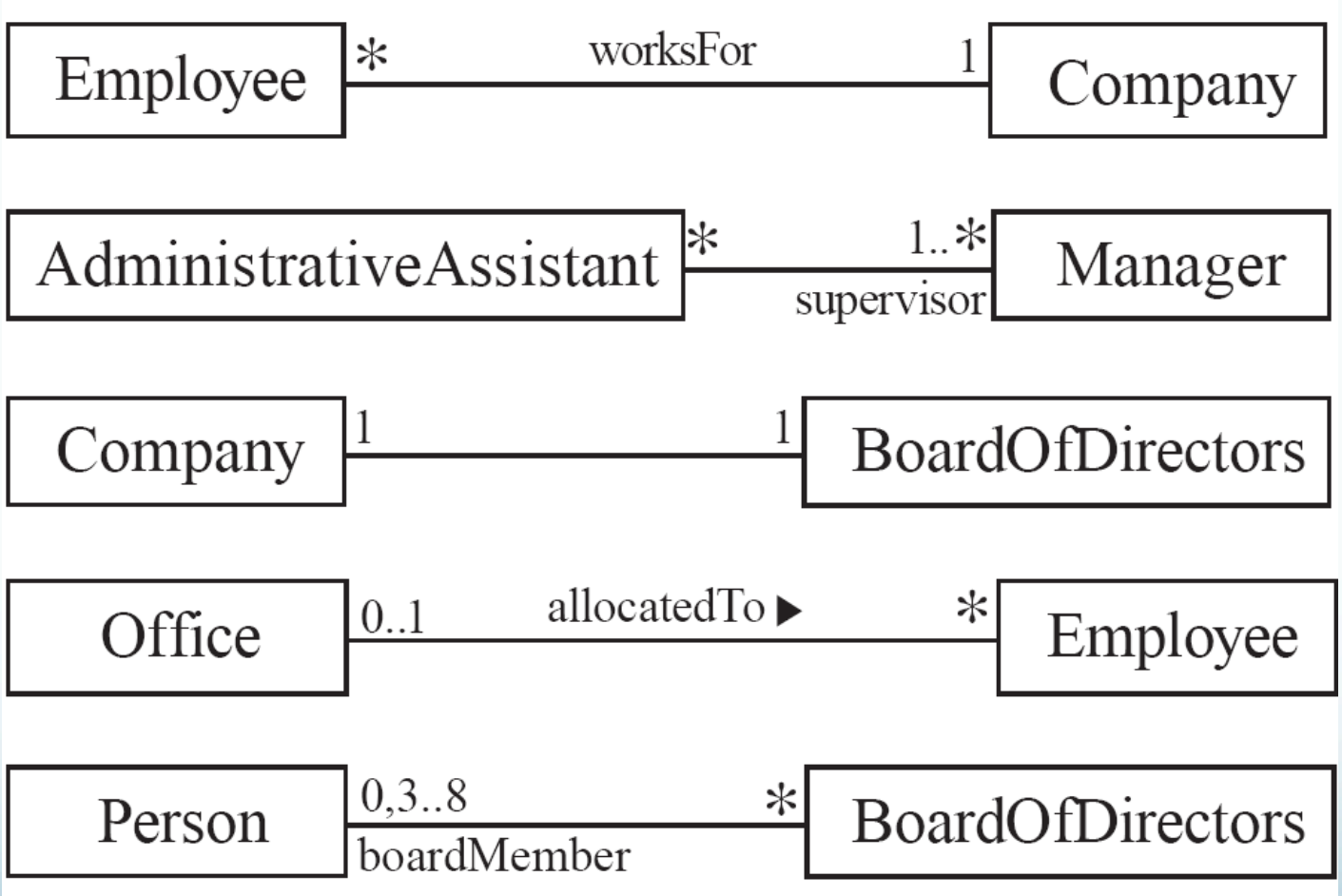
Better



Associations

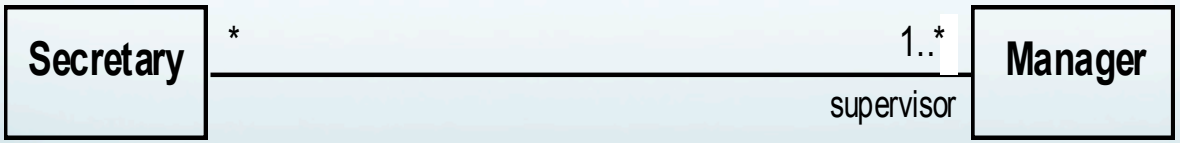


Associations and Multiplicity



Analyzing and validating associations

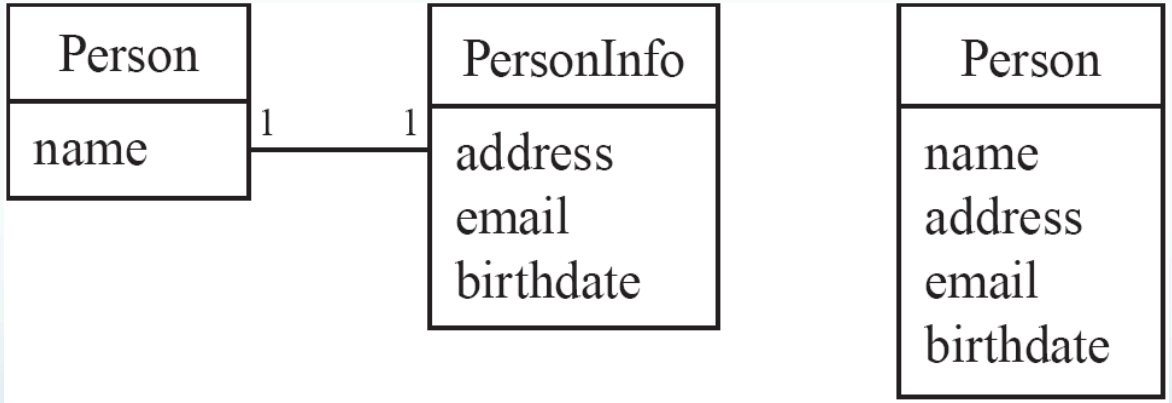
- **Many-to-many**
  - A secretary can work for many managers
  - A manager can have many secretaries
  - Secretaries can work in pools
  - Managers can have a group of secretaries
  - Some managers might have zero secretaries.
  - Is it possible for a secretary to have, perhaps temporarily, zero managers?



Analyzing and validating associations

- Avoid unnecessary one-to-one associations

- Avoid this do this

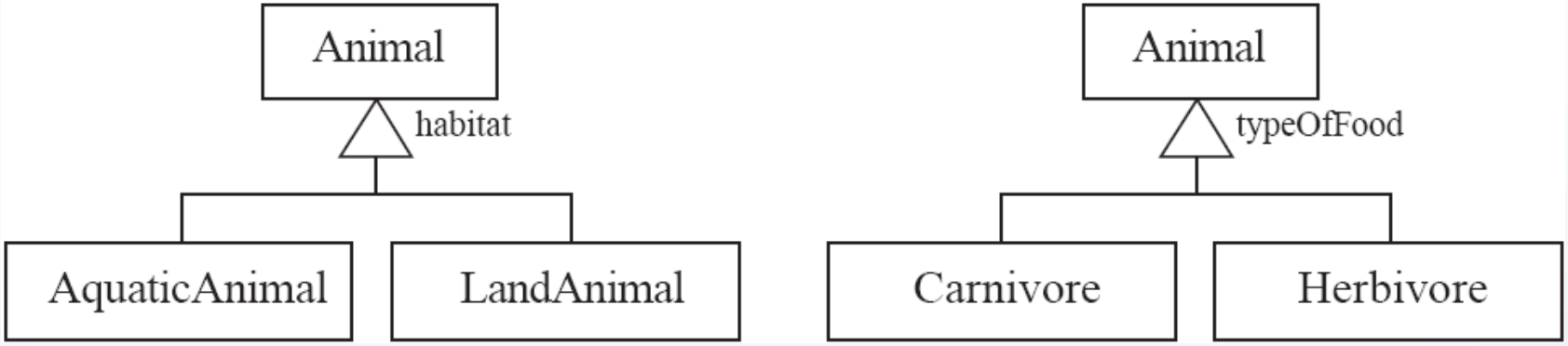


Directionality in associations

- Associations are by default are undefined, though many tools treat these as *bi-directional*.
- It is possible to limit the direction of an association by adding an arrow at one end

Generalization

- Specializing a superclass into two or more subclasses
  - The *discriminator* is a label that describes the criteria used in the specialization



Associations Vs. generalizations in object diagrams

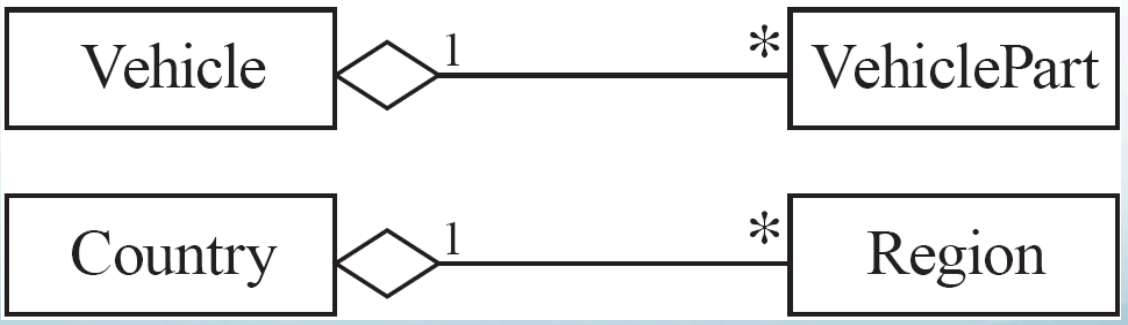
- Associations describe the relationships that will exist between *instances* at **run time**.
  - When you show an instance diagram generated from a class diagram, there will be an instance of *both* classes joined by an association

- Generalizations describe relationships between *classes* in class diagrams.
  - They do not appear in instance diagrams at all.
  - An instance of any class should also be considered to be an **instance** of each of that class's **superclasses**

Aggregation

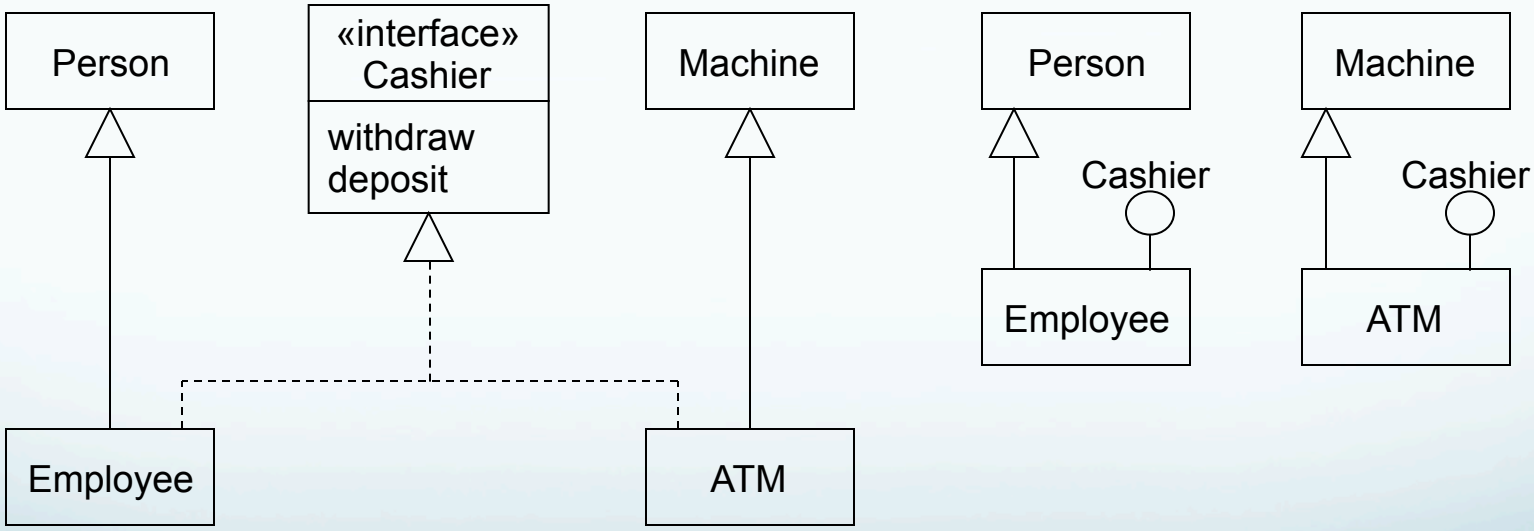
- Aggregations are special associations that represent ‘part-whole’ relationships.
  - The ‘whole’ side is often called the *assembly* or the *aggregate*
  - This symbol is a shorthand notation association named `isPartOf`

- As a general rule, you can mark an association as an aggregation if the following are true:
  - You can state that
    - the parts ‘are part of’ the aggregate
    - or the aggregate ‘is composed of’ the parts
  - When something **owns** or controls the aggregate, then they also own or control the parts



Interfaces

- An interface describes a *portion of the visible behaviour* of a set of objects.
  - An *interface* is similar to a class, except it lacks instance variables and implemented methods



Notes and descriptive text

- **Descriptive text and other diagrams**
  - Embed your diagrams in a larger document
  - Text can explain aspects of the system using any notation you like
  - Highlight and expand on important features, and give rationale
- **Notes**:
  - A note is a small block of text embedded *in* a UML diagram
  - It acts like a comment in a programming language

Suggested sequence of activities

- Identify a first set of candidate **classes** from the use cases
- Decide on specific **operations** and **attributes**
- Add relationships (**associations** and **generalizations**
- **Iterate** over the entire process until the model is satisfactory
  - Add or delete classes, responsibilities or operations associations, attributes, generalizations,
  - Identify interfaces

Don't be too disorganized. Don't be too rigid either.

Identifying Candidate class

- List the candidate classes related to the “Process Sale” use case given below by identifying **nouns** and **noun phrases.**

Process Sale success scenario use case text:

Preconditions: Cashier is identified and authenticated on a sales terminal.

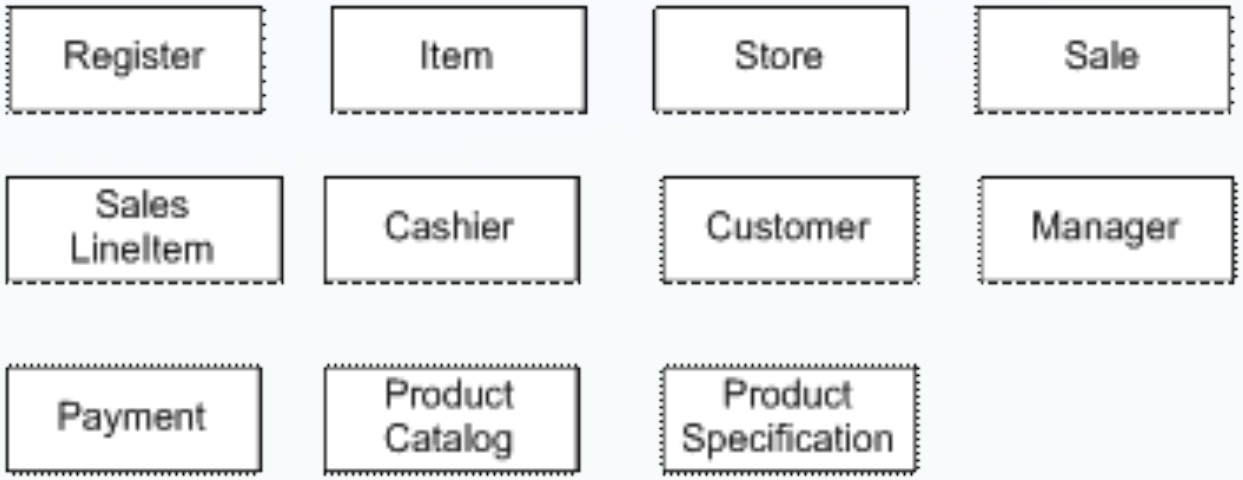
- Customer arrives at POS checkout with goods and/or services to purchase.
- Cashier starts a new sale.
- Cashier enters item identifier.
- System records sales line item and presents item description, price, and running total. Price calculated from a set of price rules.
  < Cashier repeats steps 3-4 until indicates done >
- System presents total with taxes calculated.
- Cashier tells Customer the total, requests payment.
- Customer pays and System handles payment.
- System logs completed sale and sends sale and payment information to the external Accounting System and Inventory System.
- System presents receipt.
- Customer leaves with receipt and goods (if any).

Identifying Nouns and Noun Phrases

- **Customer** arrives at **POS checkout** with **goods** and/or **services** to purchase.
- **Cashier** starts a new **sale**.
- Cashier enters **item identifier**.
- System records **sales line item** and presents **item description**, **price**, and **running total**. Price calculated from a set of price rules.
  < *Cashier repeats steps 3-4 until indicates done* >
- System presents total with *taxes* calculated.
- Cashier tells Customer the total, requests **payment**.
- Customer pays and System handles payment.
- System logs completed **sale** and sends sale and payment information to the external **Accounting** System and **Inventory** System.
- System presents **receipt**.
- Customer leaves with receipt and goods (if any).

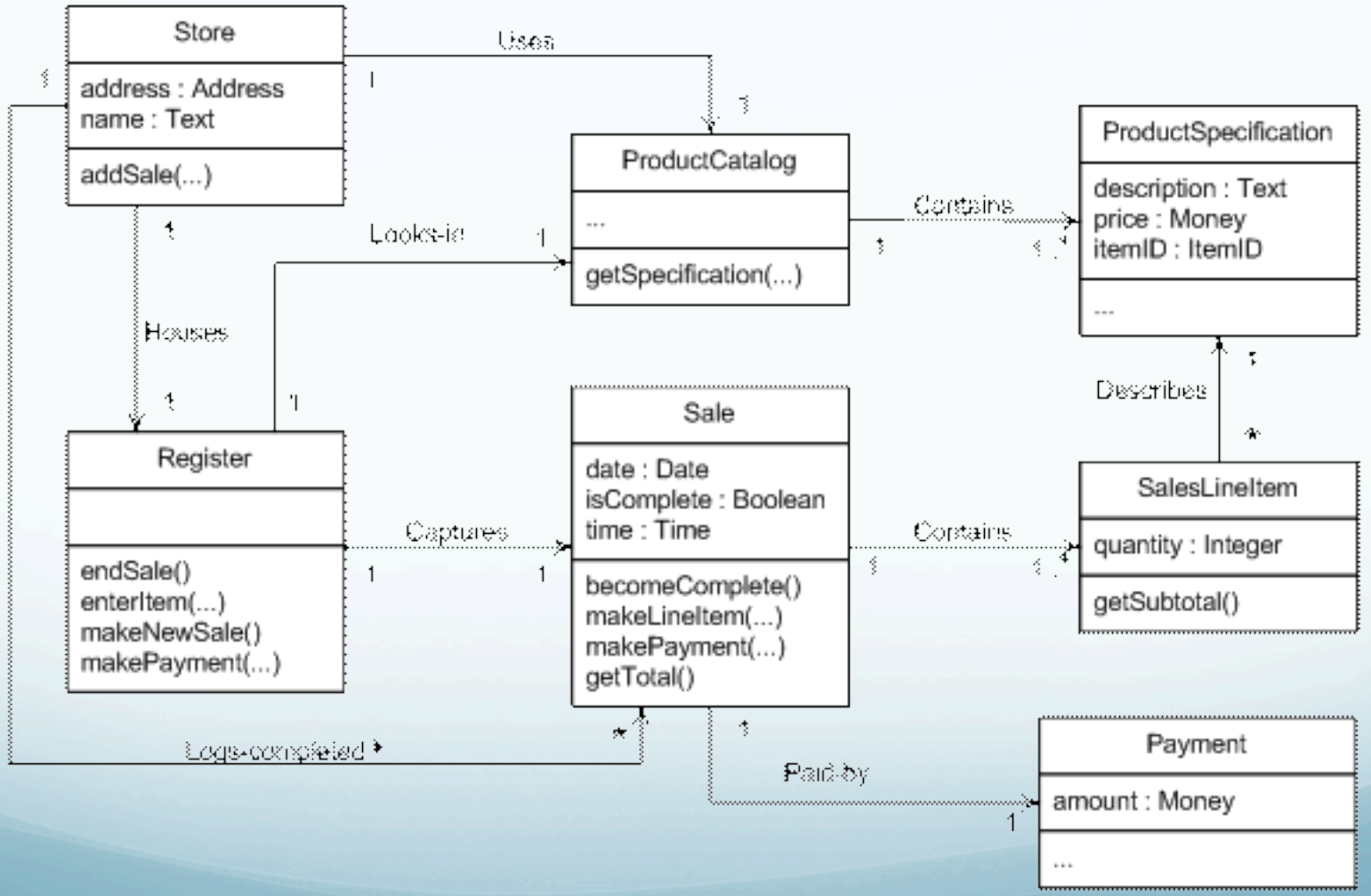
Note the differences between **class and attributes** – not all nouns automatically become classes.

Conceptual classes

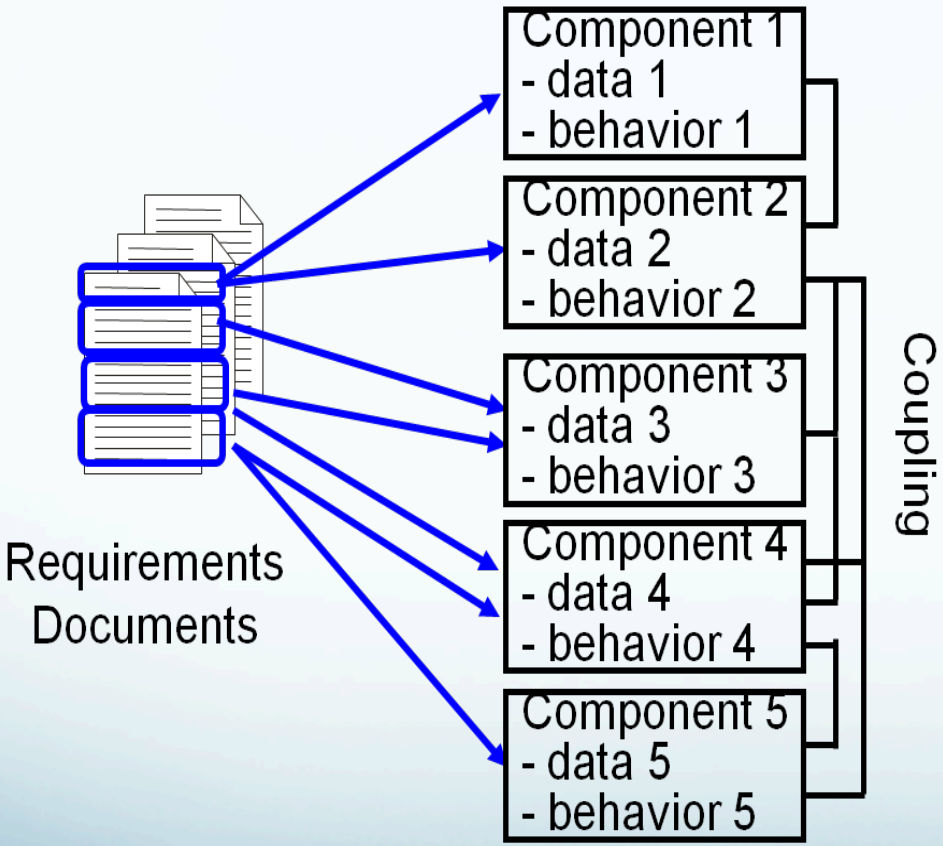


- Moving to software classes, Cashier, Customer & Manager are not considered, Item becomes an attribute in ProductSpecification.

Class Diagram

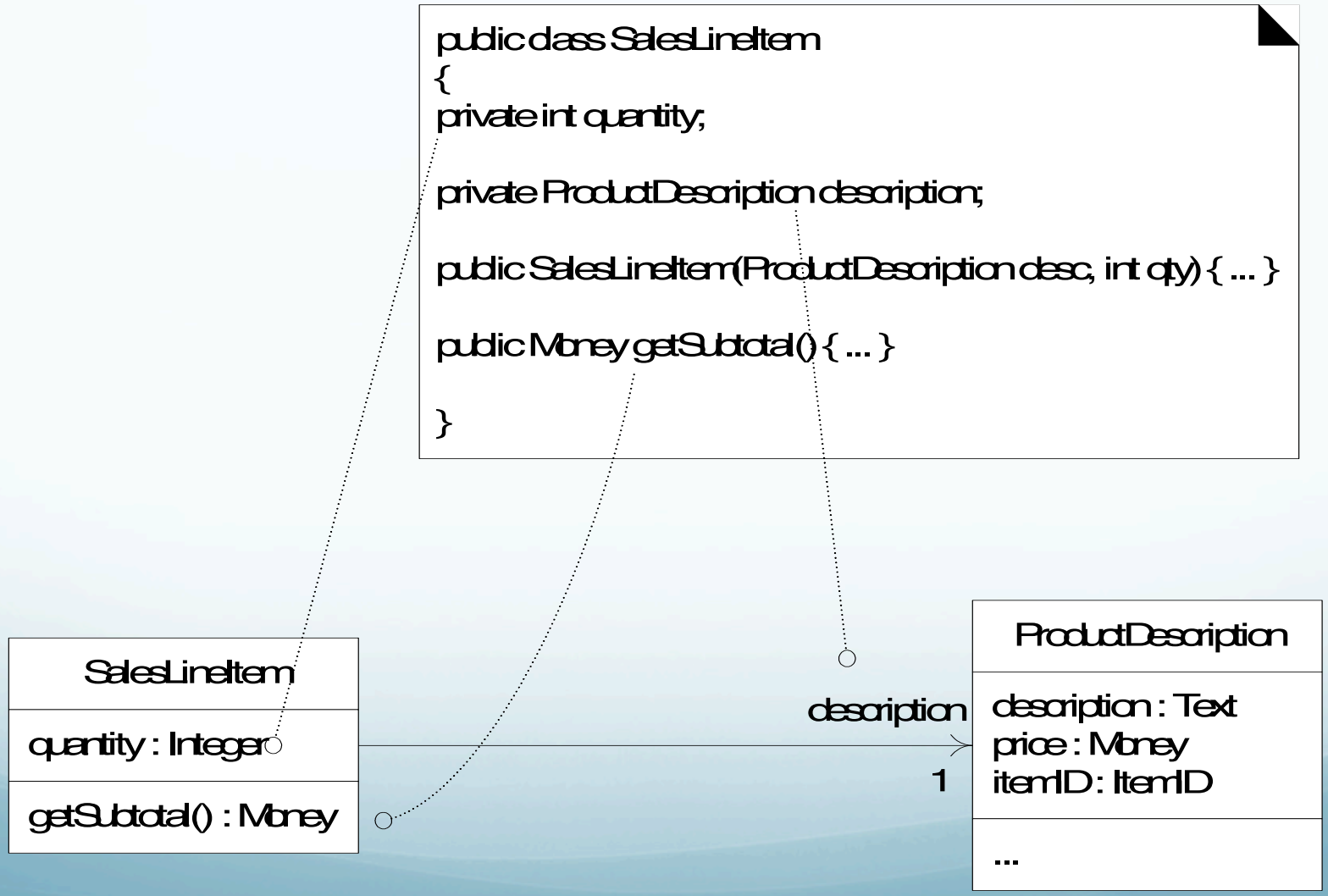


Mapping Requirements to Design Components



- Design must satisfy requirements
  - Everything (data and behavior) in the requirements must be mapped to the design components
  - Decide what functionality goes into which component
- As you do the mapping, assess functional cohesion and coupling
  - Strive for **low** coupling and **high** cohesion

Mapping class diagram to code



Modelling Interactions and Behaviour

Interaction Diagrams

- Interaction diagrams are used to model the dynamic aspects of a software system
  - They help you to visualize how the system runs.
  - An interaction diagram is often built from a use case and a class diagram.
    - The objective is to show how a set of objects accomplish the required interactions with an actor.

Interactions and messages

- Interaction diagrams show how a set of actors and objects communicate with each other to perform:
  - The steps of a use case, or
  - The steps of some other piece of functionality.

- The set of steps, taken together, is called an *interaction*.

- Interaction diagrams can show several different types of communication.
  - E.g. method calls, messages send over the network
  - These are all referred to as *messages*.

Elements found in interaction diagrams

- Instances of classes
  - Shown as boxes with the class and object identifier underlined
- Actors
  - Use the stick-person symbol as in use case diagrams
- Messages
  - Shown as arrows from actor to object, or from object to object

Sequence Diagrams – Modeling Interaction

1. Student selects Register for Courses option

2. System retrieves a list of the available courses

3. Student specifies the desired course

4. System shows a list of the available sections

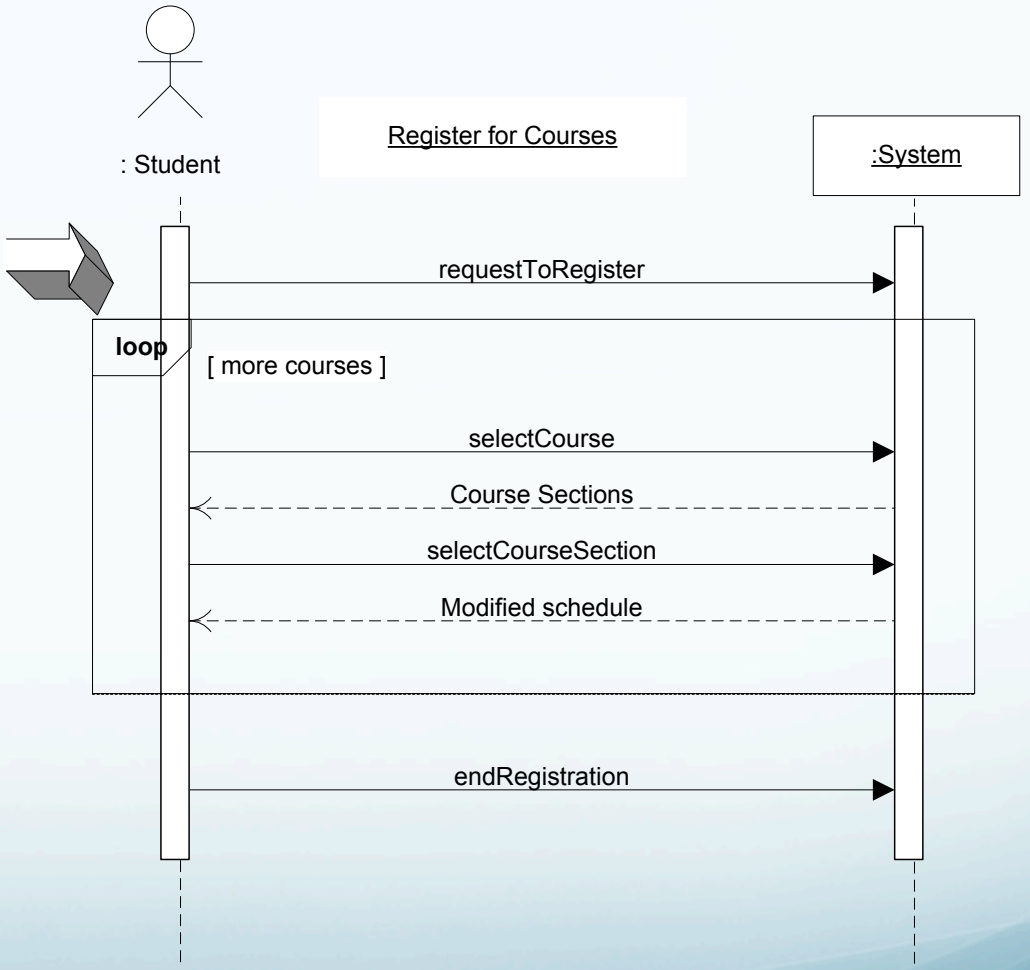
5. Student selects the course section

6. System verifies if the student has passed prerequisites

7. System add course section to student's Schedule

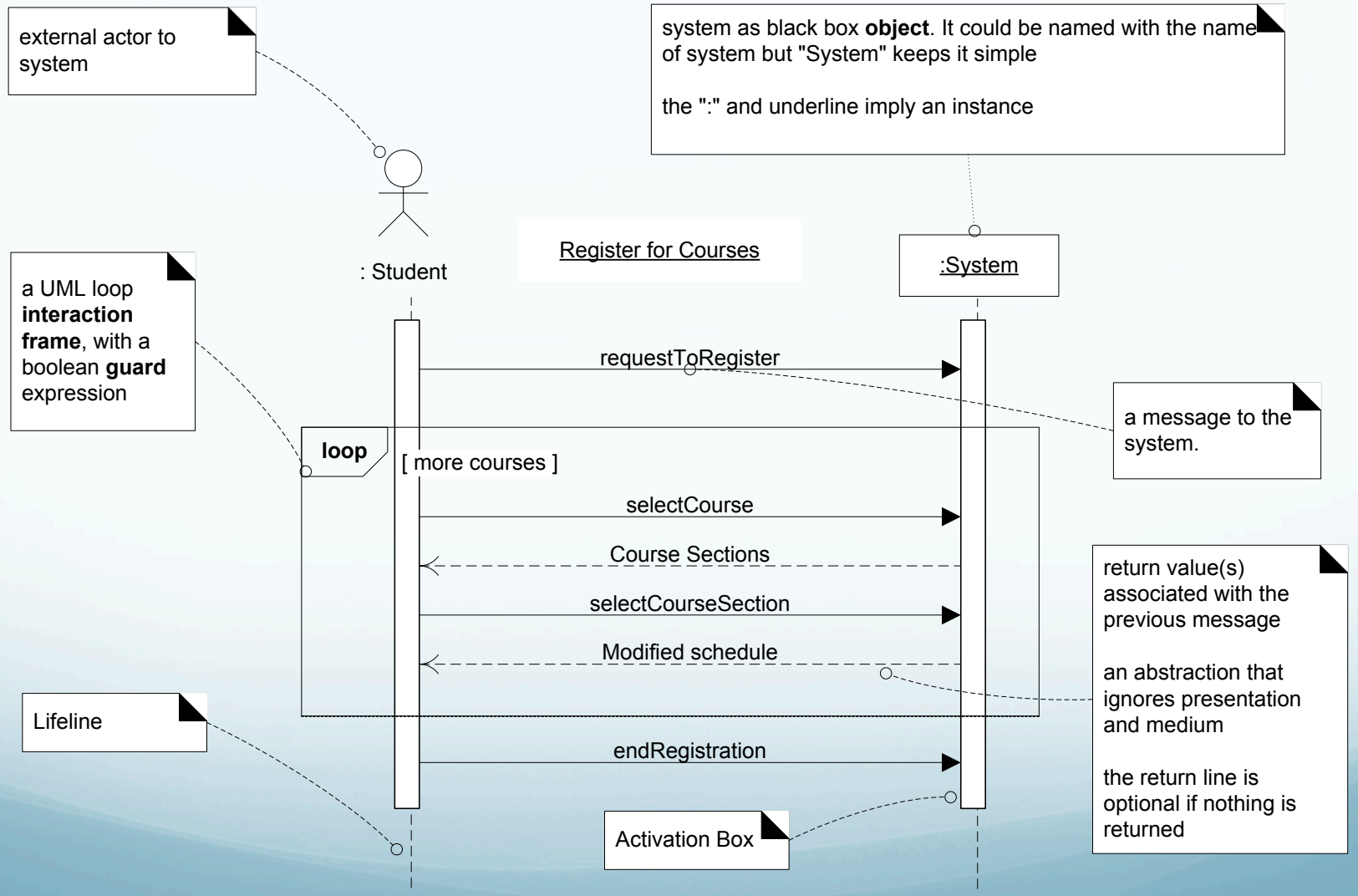
8. System displays modified student's Schedule

9. Steps 3-8 repeated until student finished



Based on System Sequence Diagrams
Larman, C. Applying UML and Patterns, 3rd Edition (Pearson Education, 2005)

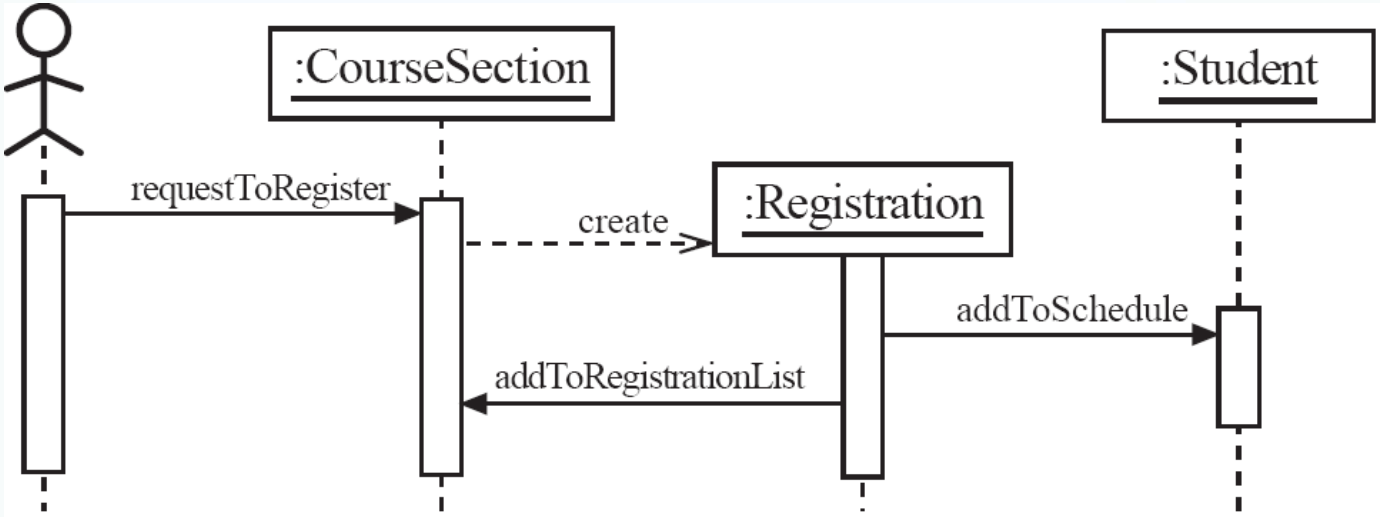
Sequence Diagrams – Elements

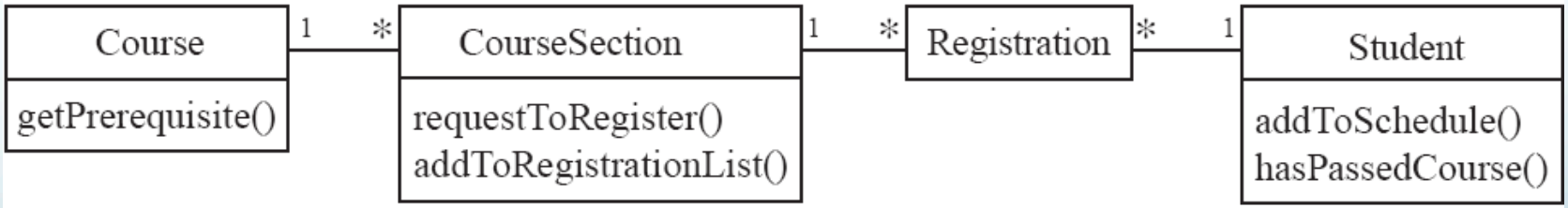


Sequence diagrams

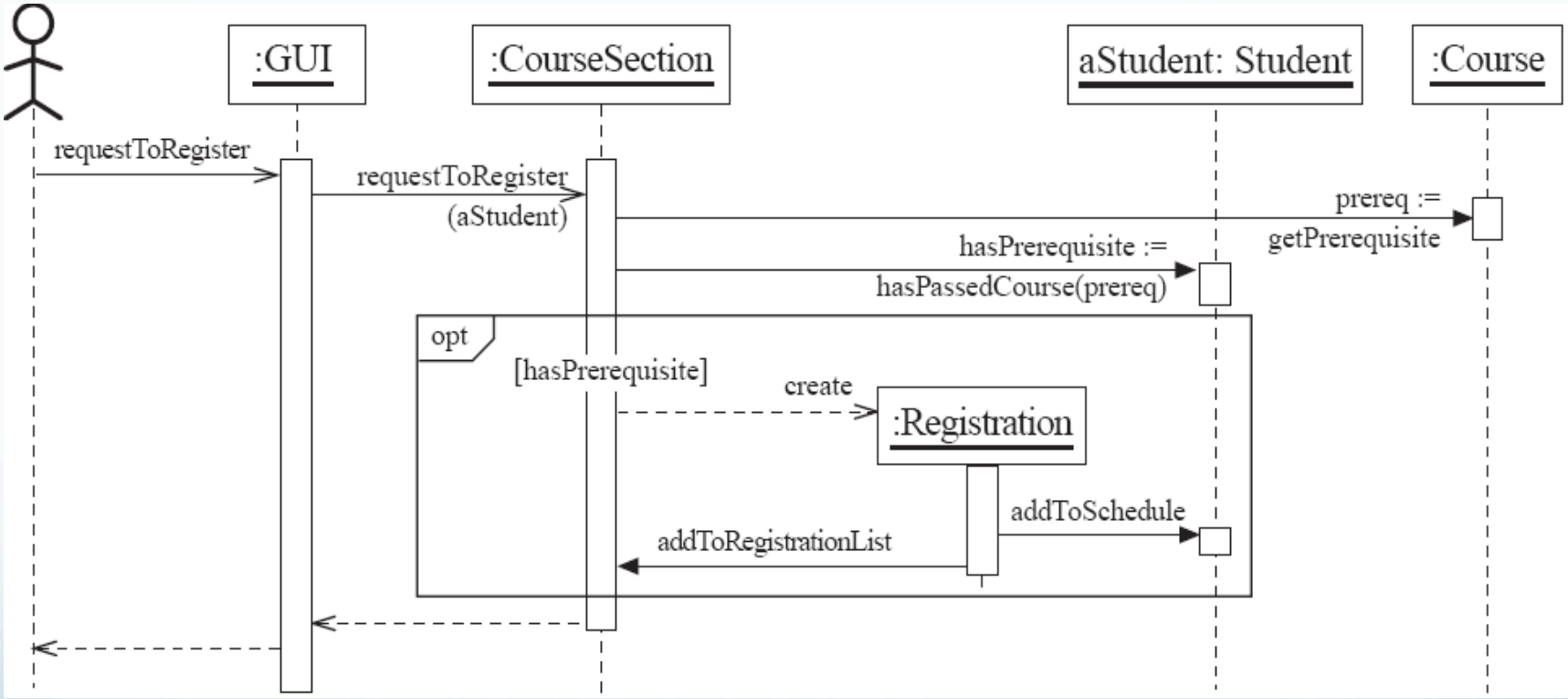
- A sequence diagram shows the sequence of messages exchanged by the set of objects performing a certain task
  - The objects are arranged horizontally across the diagram.
  - An actor that initiates the interaction is often shown on the left.
  - The vertical dimension represents time.
  - A vertical line, called a *lifeline*, is attached to each object or actor.
  - The lifeline becomes a broad box, called an *activation box* during the *live activation* period.
  - A message is represented as an arrow between activation boxes of the sender and receiver.
    - A message is labelled and can have an argument list and a return value.

Sequence diagrams – an example



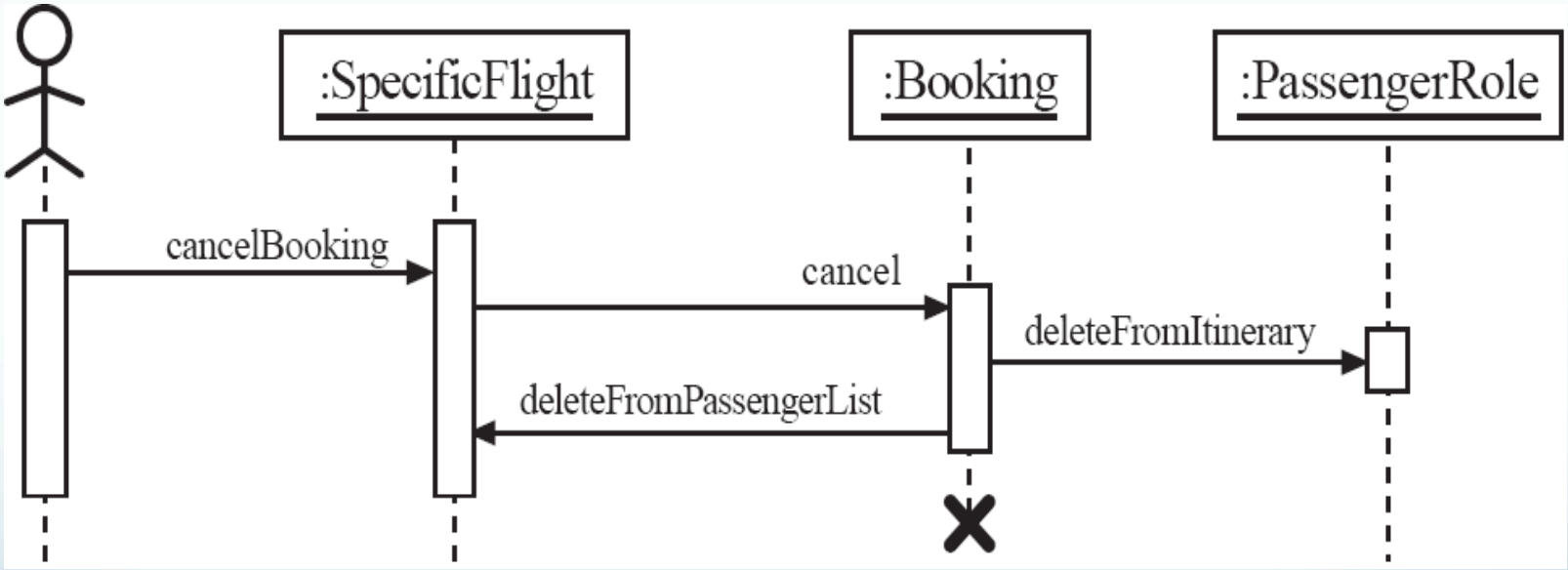


Sequence diagrams – same example, more details



Sequence diagrams – an example with object deletion

- If an object's life ends, this is shown with an X at the end of the lifeline

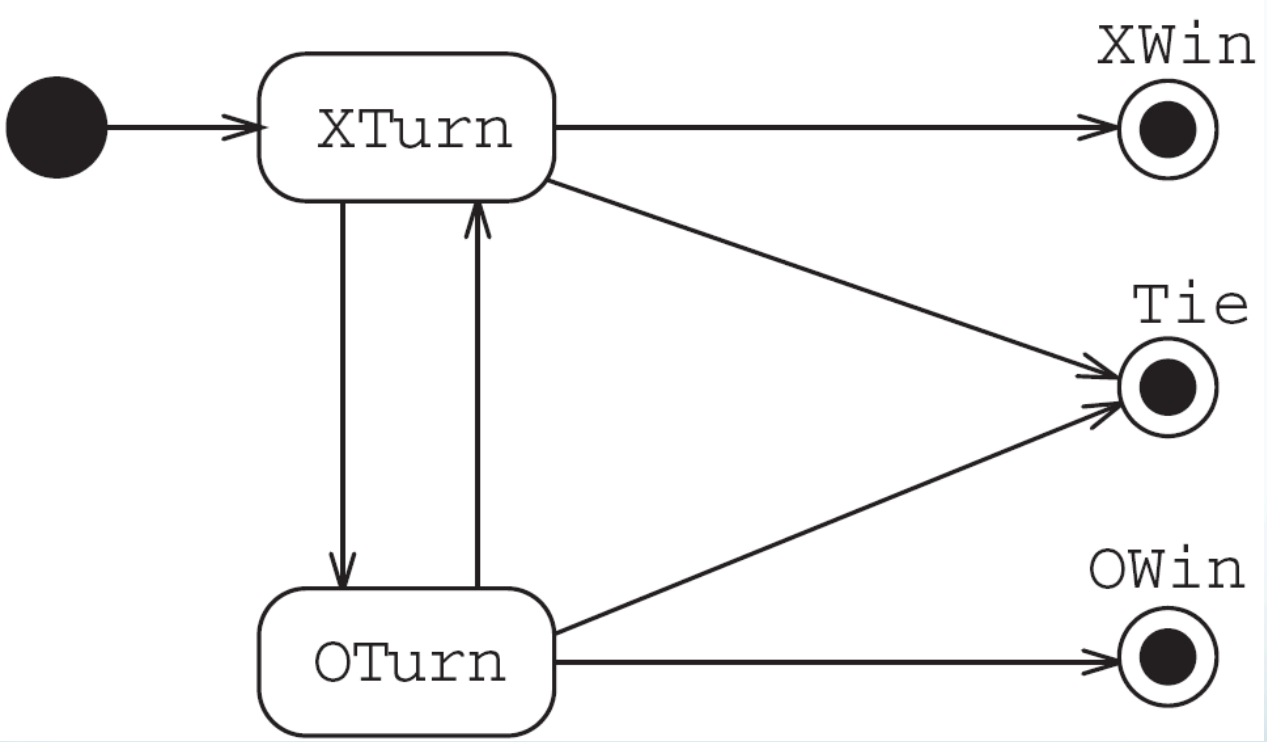


State Diagrams

- A state diagram describes the behaviour of a *system*, some *part* of a system, or an *individual object*.
  - At any given point in time, the system or object is in a certain state.
    - Being in a state means that it will behave in a *specific way* in response to any events that occur.
  - Some events will cause the system to change state.
    - In the new state, the system will behave in a different way to events.
  - A state diagram is a directed graph where the nodes are states and the arcs are transitions.

State diagrams – an example

- tic-tac-toe game



States

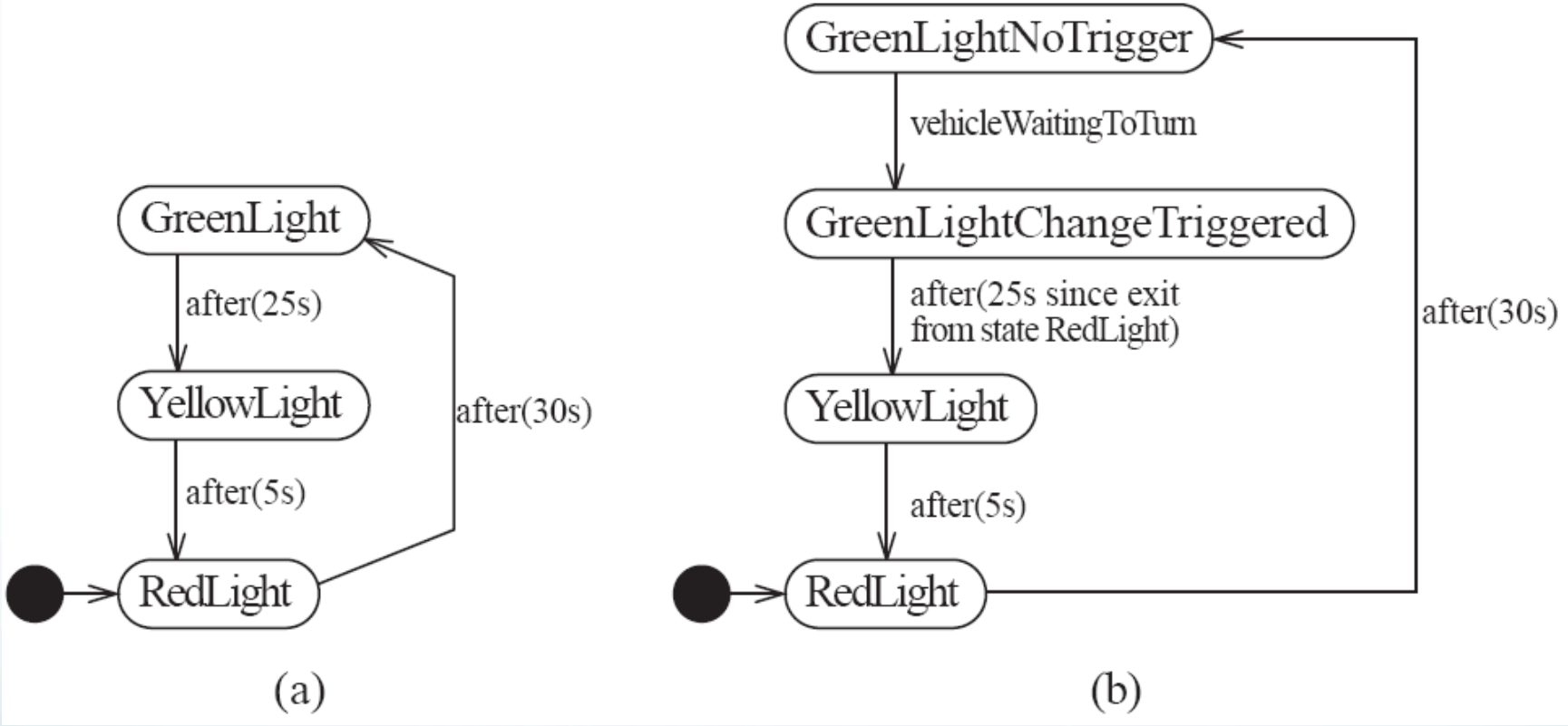
- At any given point in time, the system is in one state.
- It will remain in this state until an event occurs that causes it to change state.
- A state is represented by a rounded rectangle containing the name of the state.
- Special states:
  - A black circle represents the *start state*
  - A circle with a ring around it represents an *end state*

Transitions

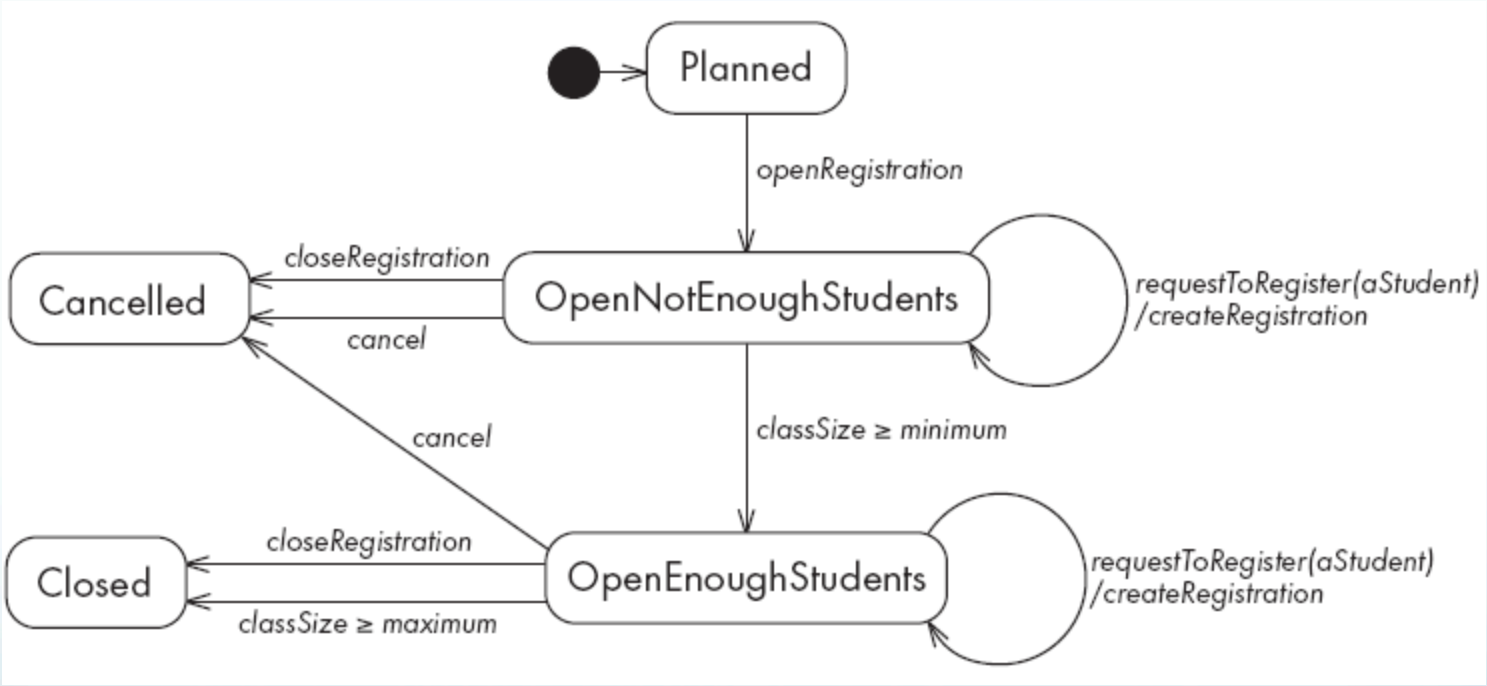
- A transition represents a change of state in response to an event.
  - It is considered to occur instantaneously.

- The label on each transition is the event that causes the change of state.

State diagrams – an example of transitions with time-outs and conditions



State diagrams – an example with conditional transitions



Activities in state diagrams

- An *activity* is something that takes place while the system is *in* a state.
  - It takes a period of time.
  - The system may take a transition out of the state in response to completion of the activity,
  - Some other outgoing transition may result in:
    - The interruption of the activity, and
    - An early exit from the state.

Actions in state diagrams

- An *action* is something that takes place effectively *instantaneously*
  - When a particular transition is taken,
  - Upon entry into a particular state, or
  - Upon exit from a particular state

- An action should consume no noticeable amount of time

Summary

- System Dynamics
  - Interactions
    - UML Interaction diagrams (**sequence** and communication) show a set of actors and objects interacting exchanging messages
    - Should be consistent with system requirements (i.e. Use Cases, Use Stories)
  - Behavior
    - **State** and activity diagrams show how an object or system changes state in reaction to a series of events
- The Dynamic Model refines the Static Model
  - It might add, modify, or remove: classes, methods, parameters, and associations