Interaction Diagrams

- UML Specifies a number of interaction diagrams to model dynamic aspects of the system
- Dynamic aspects of the system
  - Messages moving among objects/classes
  - Flow of control among objects
  - Sequences of events

UML sequence diagrams

- **sequence diagram**: an "interaction diagram" that models a single scenario executing in the system
  - perhaps 2nd most used UML diagram (behind class diagram)

- relation of UML diagrams to other exercises:
  - CRC cards -> class diagram
  - use cases -> sequence diagrams

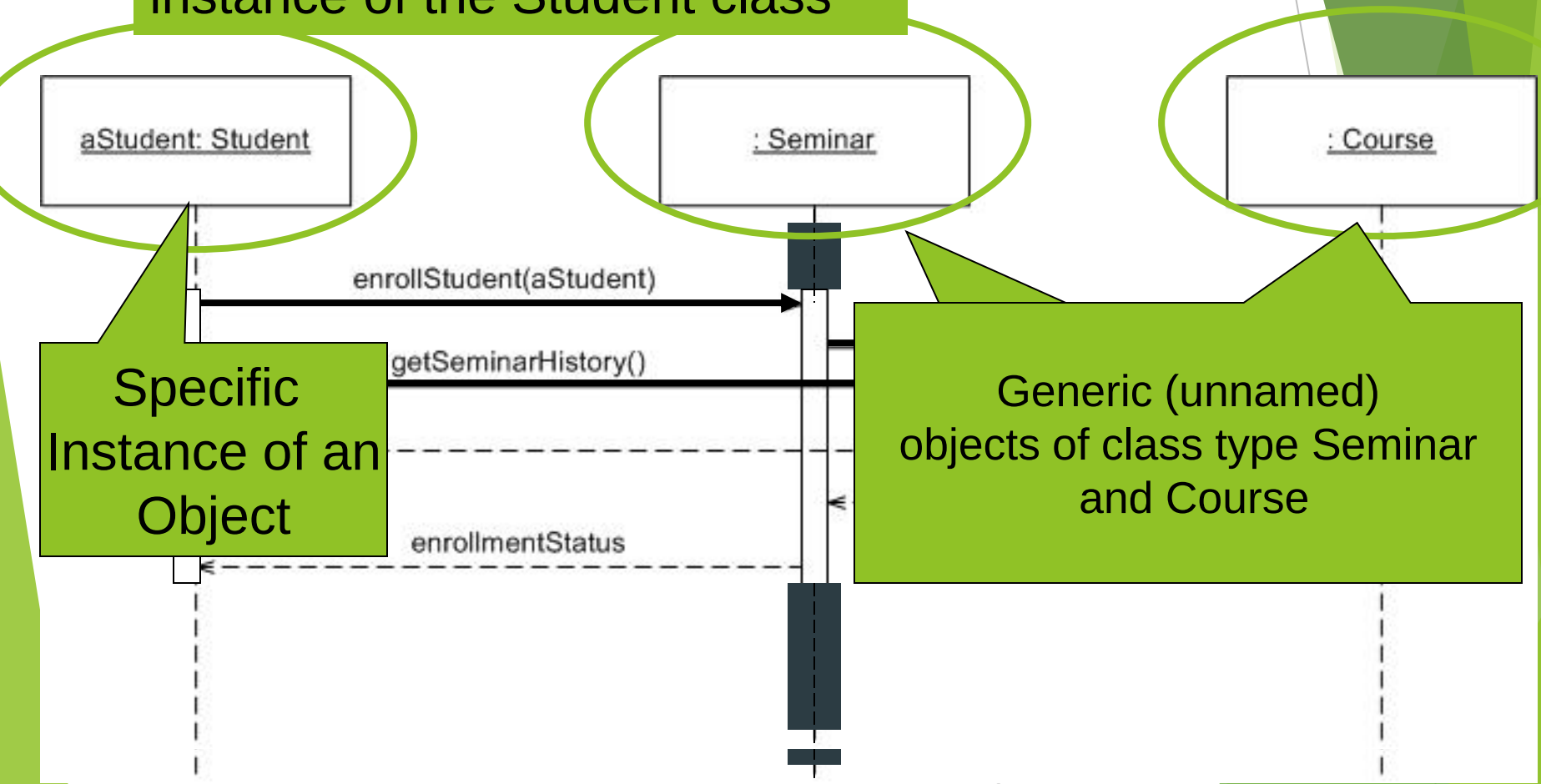
Sequence Diagrams

- Describe the flow of messages, events, actions between objects
- Show concurrent processes and activations
- Show time sequences that are not easily depicted in other diagrams
- Typically used during analysis and design to document and understand the logical flow of your system

Emphasis on time ordering!

Components

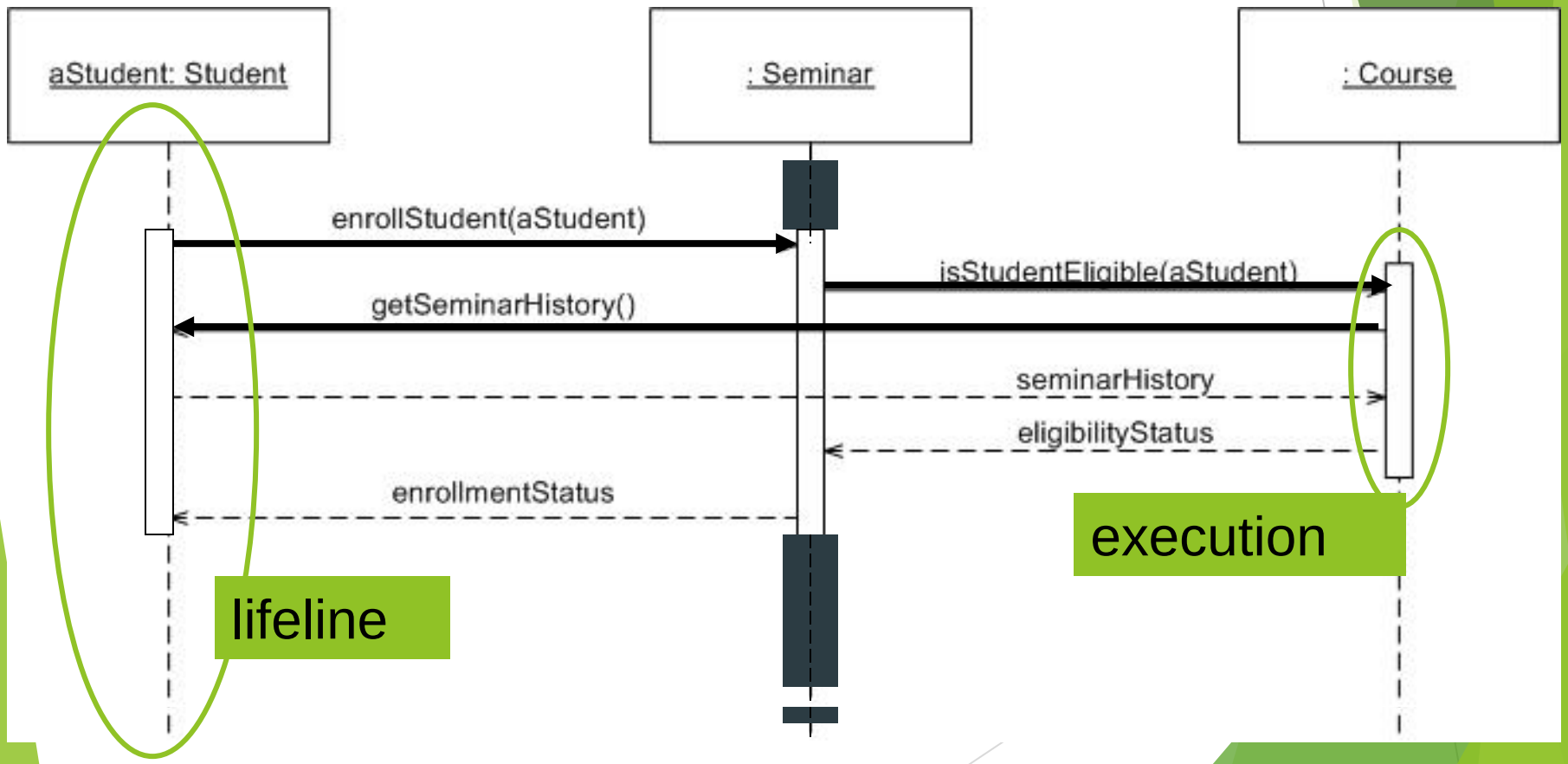
Objects: aStudent is a specific instance of the Student class



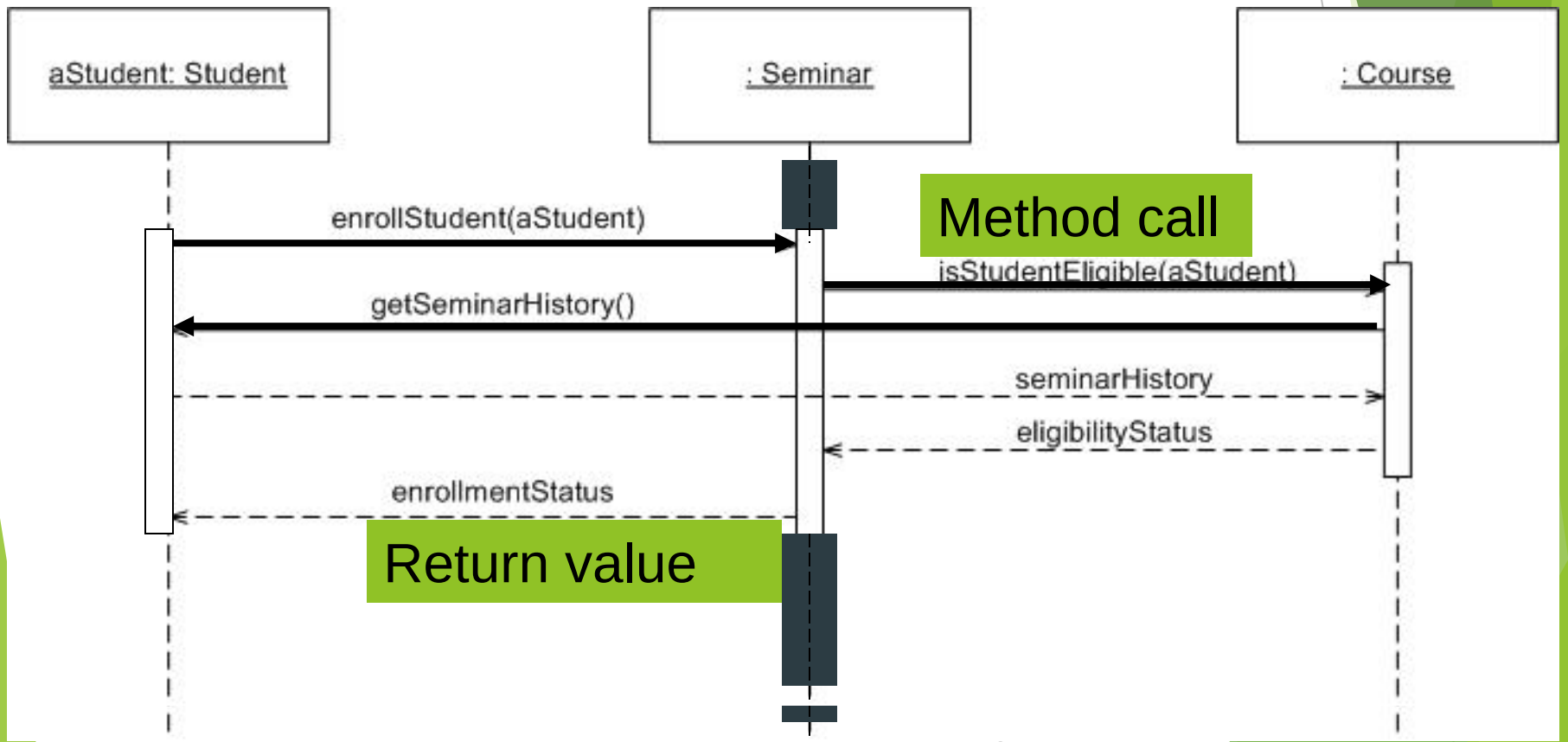
Specific Instance of an Object

Generic (unnamed) objects of class type Seminar and Course

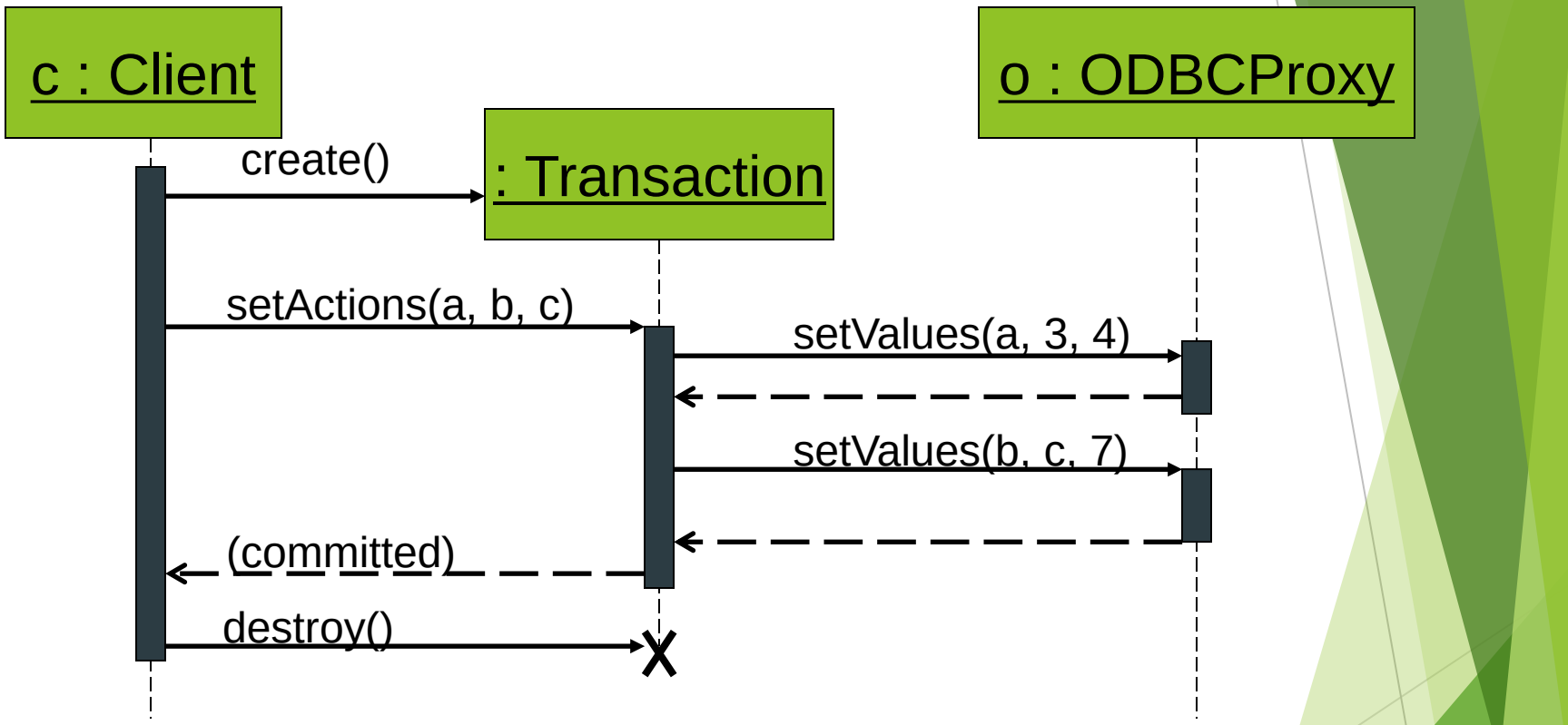
Components


aStudent: Student
: Seminar
: Course
enrollStudent(aStudent)
isStudentEligible(aStudent)
getSeminarHistory()
seminarHistory
eligibilityStatus
enrollmentStatus
lifeline
execution

Components


aStudent: Student
: Seminar
: Course
enrollStudent(aStudent)
Method call
isStudentEligible(aStudent)
getSeminarHistory()
seminarHistory
eligibilityStatus
enrollmentStatus
Return value

Components



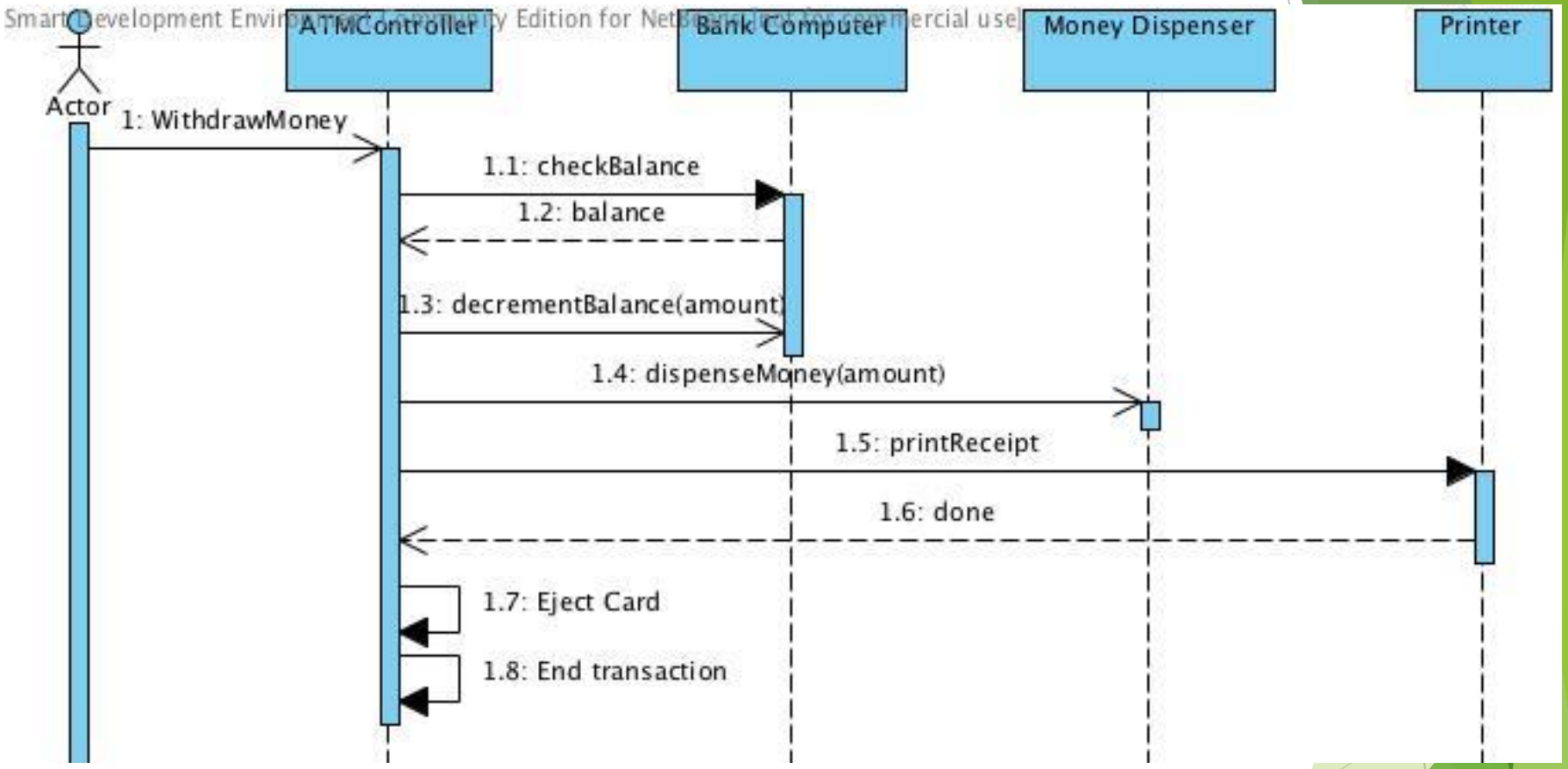
create()
destroy()

Synchronous message

Asynchronous message

Return message

Async Message Example

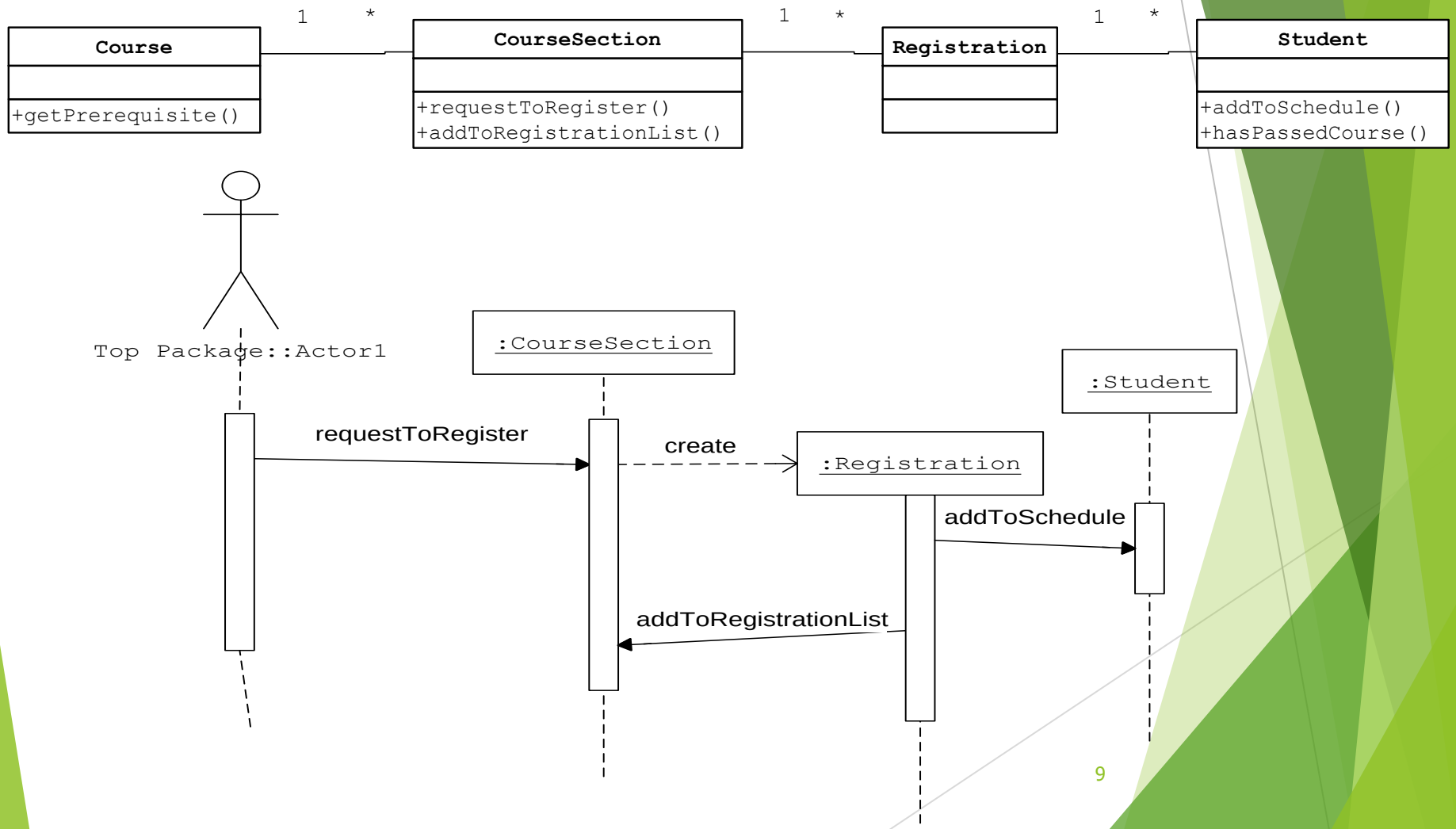


Synchronous message

Asynchronous message

Return message

Example: the process of the registration of a student in a course



Entity, Boundary and Control classes

Entities

- Entities are objects representing system data: Customer, Product, Transaction, Cart, etc.

Boundaries

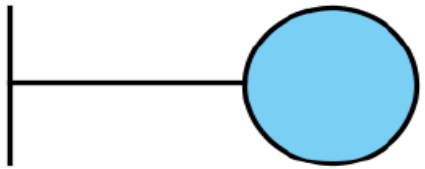
- Boundaries are objects that interface with system actors: UserInterface, DataBaseGateway, ServerProxy, etc.

Controls

- Controls are objects that mediate between boundaries and entities. They orchestrate the execution of commands coming from the boundary by interacting with entity and boundary objects. Controls often correspond to use cases and map to use case controllers in the design model.

Boundary classes

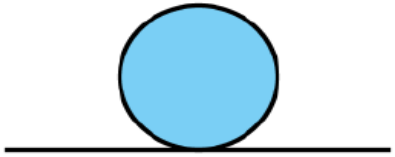
- This is a class used to model the interaction between the system and its actors.
- The interaction often involves inputs and outputs.
- They clarify and collect information regarding the systems' boundary (user interface)
- Boundary classes represent abstractions of windows, or other communication interfaces etc. (the computer system).
- Each boundary class should be related to at least one actor and vice versa.



Boundary

Entity classes

- An entity class is used to model persistent data (information) AND business logic.
- An entity class is usually not passive.
- An entity 'object' can show changes in data(information).
- Entity classes often show a logical data structure ready for design.
- Entity classes also show what data(information) the system is dependent upon.



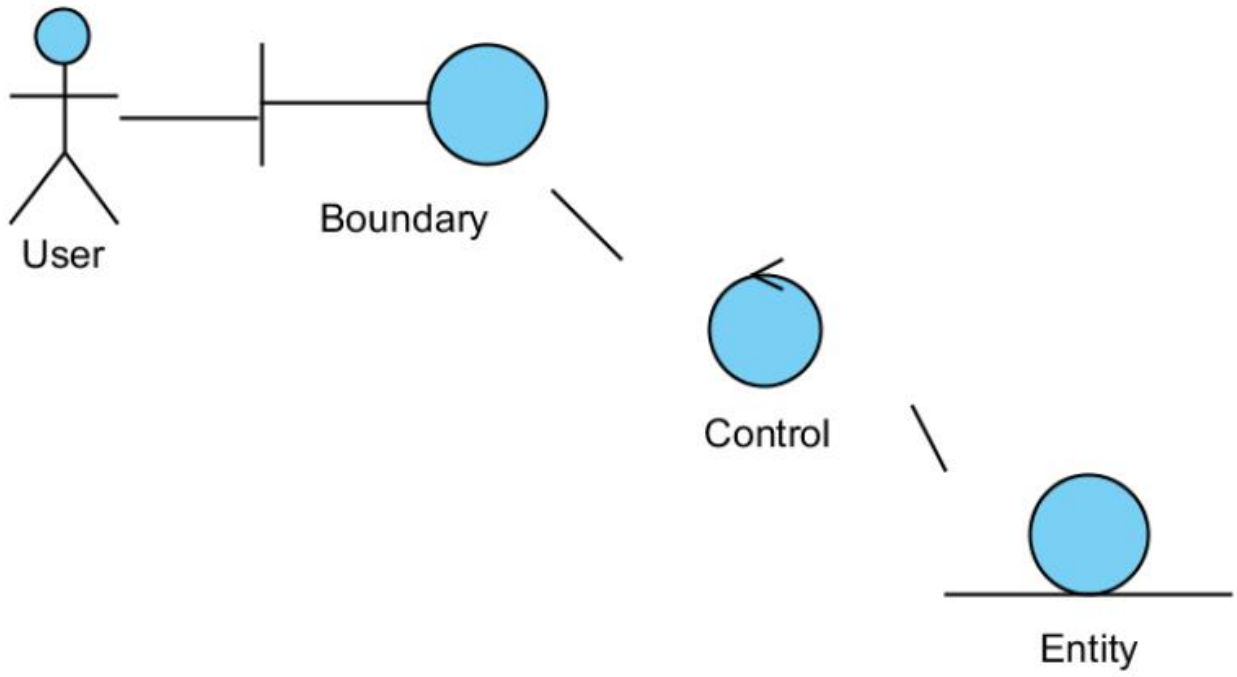
Entity

Control classses

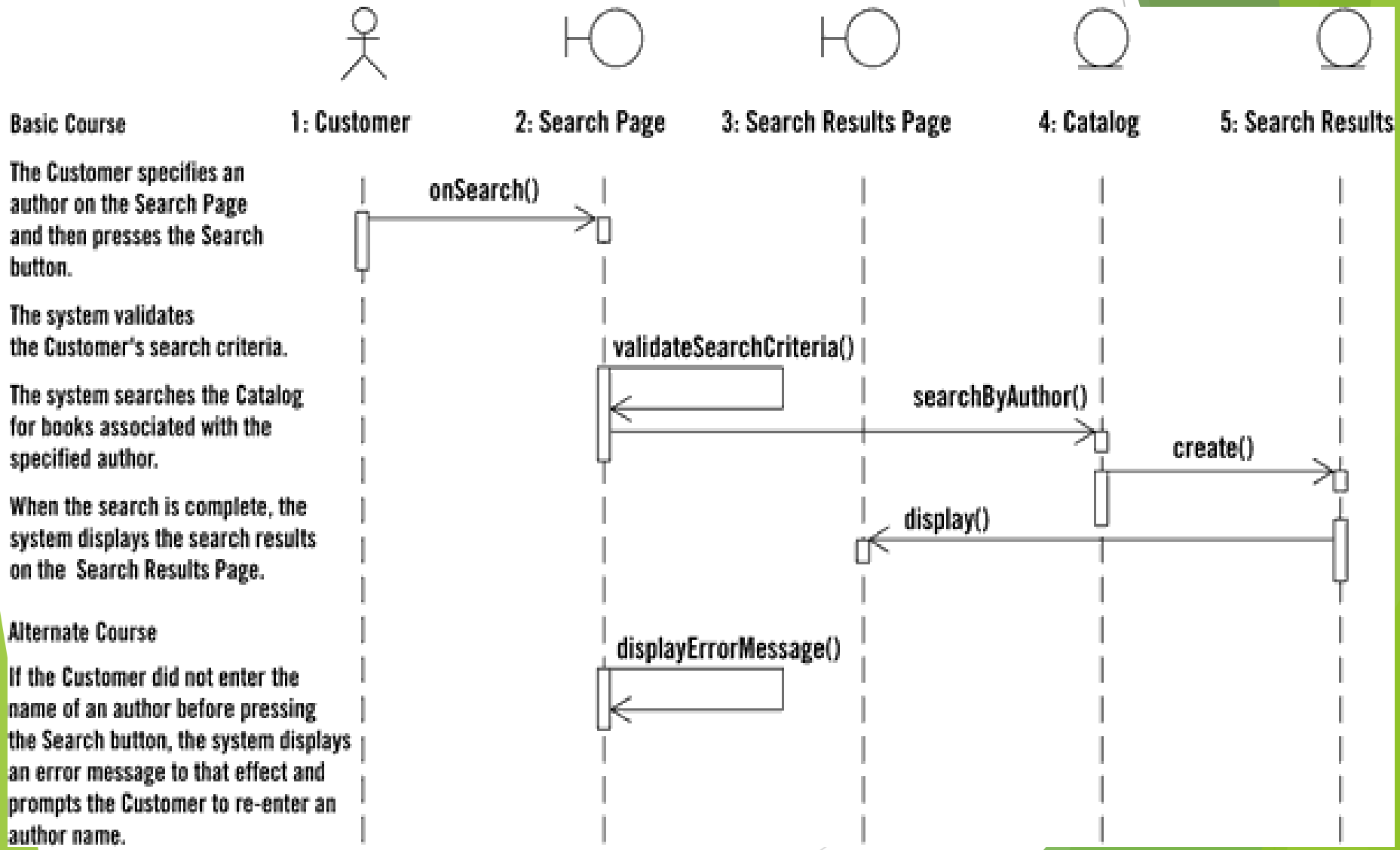
- Control classes represent coordination, sequencing, transactions and control of other objects.
- The dynamics of the system are modelled by control classes.
- Typically they coordinate the movement (parameters) between boundary and entity classes.

Control

Flow of execution

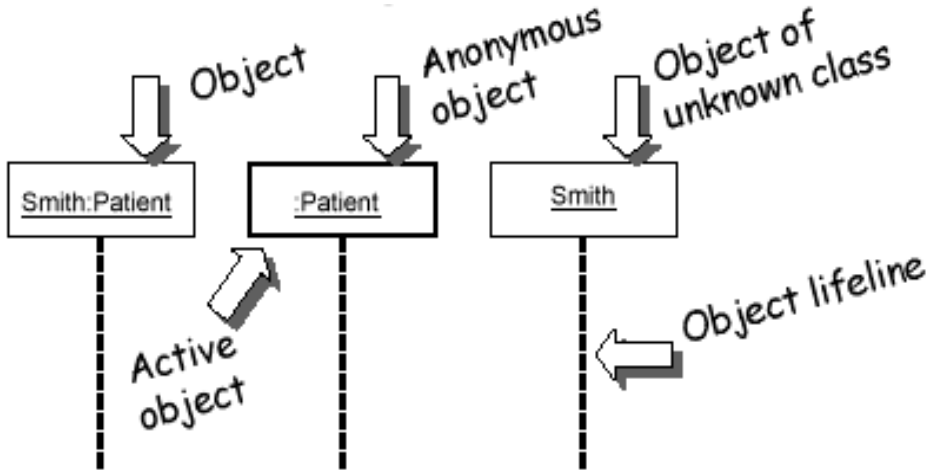


Sequence dg. from use case



Representing objects

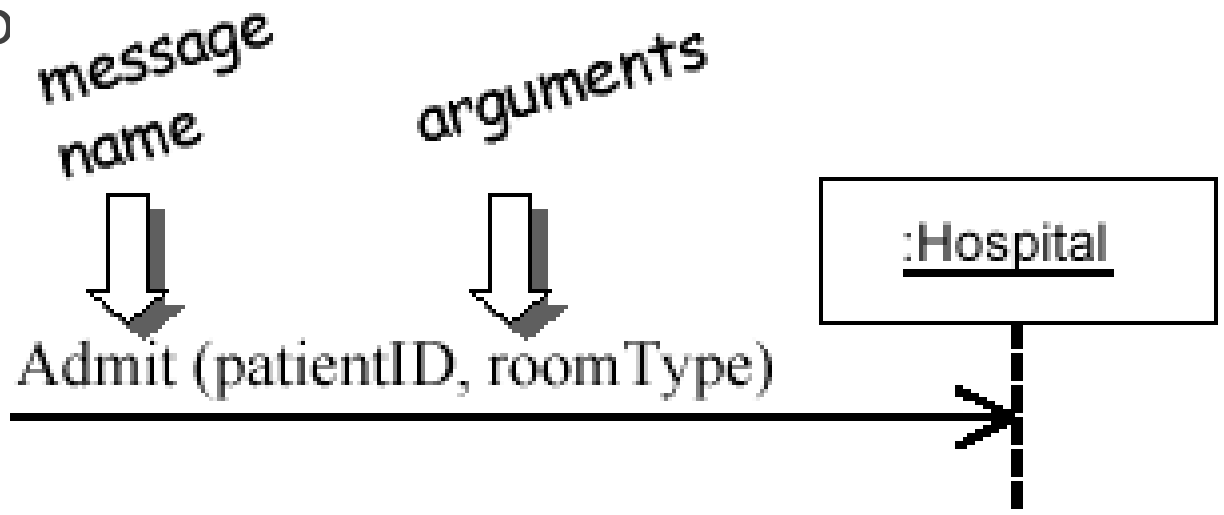
- Squares with object type, optionally preceded by object name and colon
  - write object's name if it clarifies the diagram
  - object's "life line" represented by dashed vert. line



Name syntax: <objectname>:<classname>

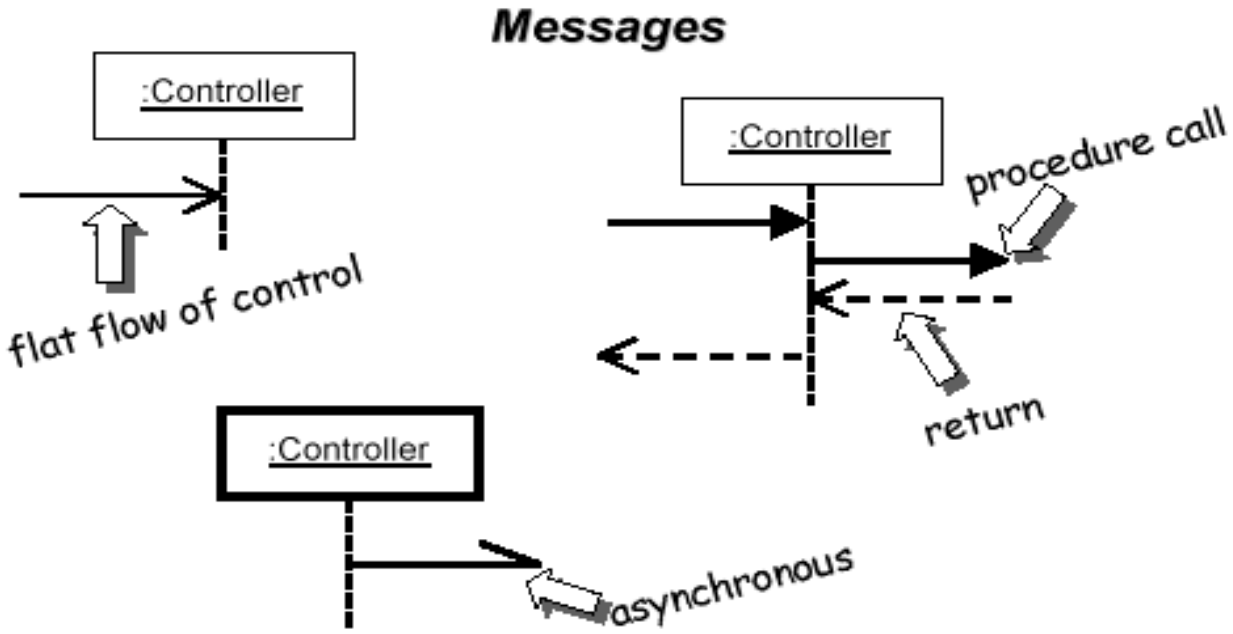
Messages between objects

- message (method call) indicated by horizontal arrow to other object
  - write message name and arguments ab



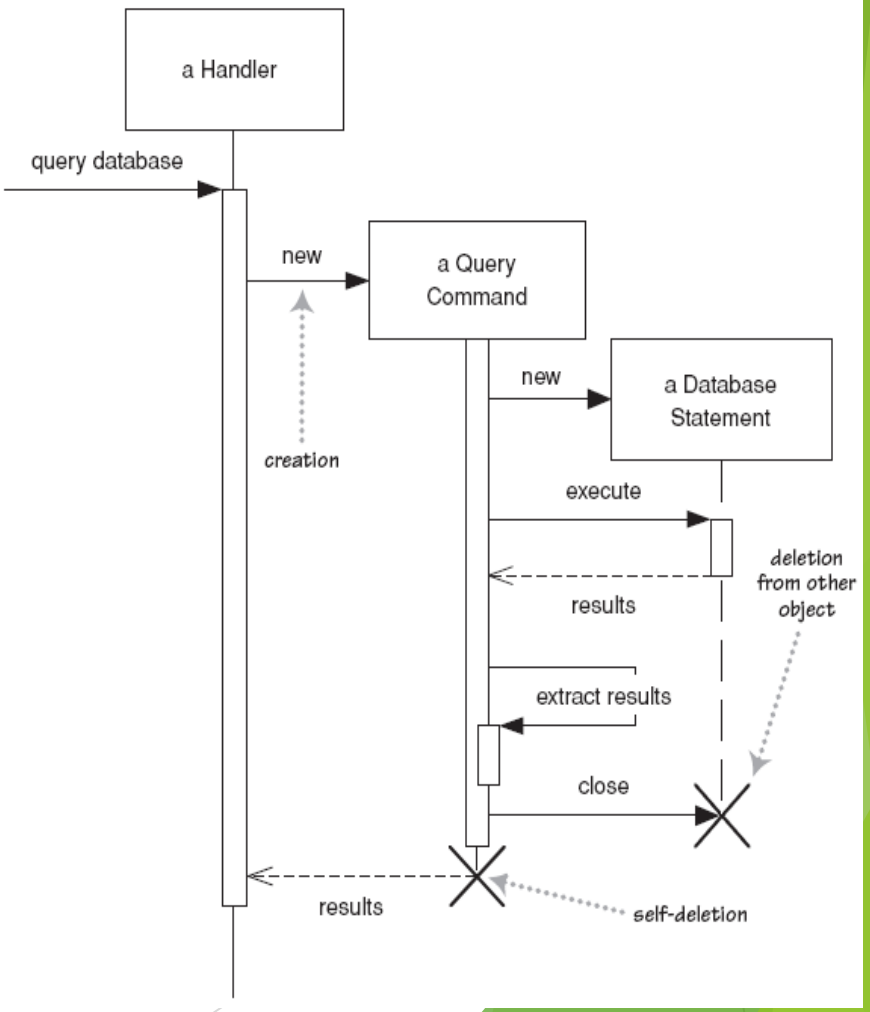
Messages, continued

- message (method call) indicated by horizontal arrow to other object
  - dashed arrow back indicates return
  -



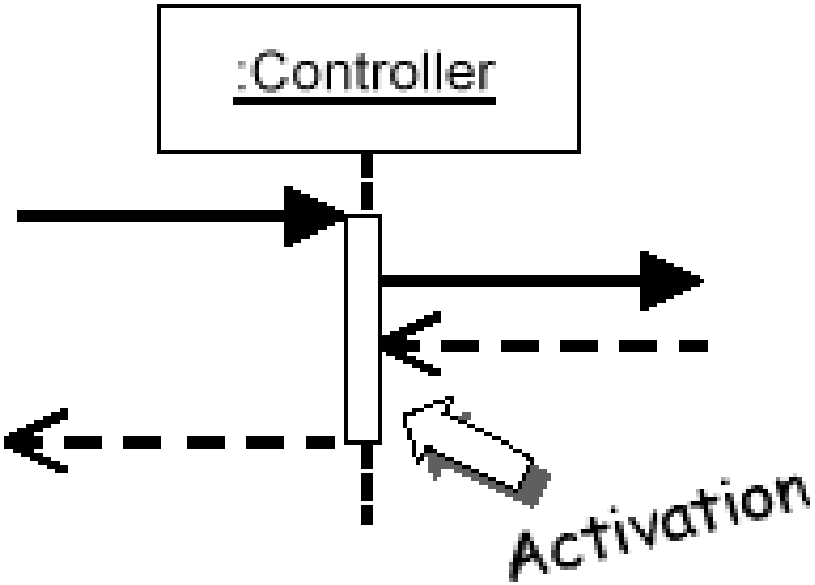
Lifetime of objects

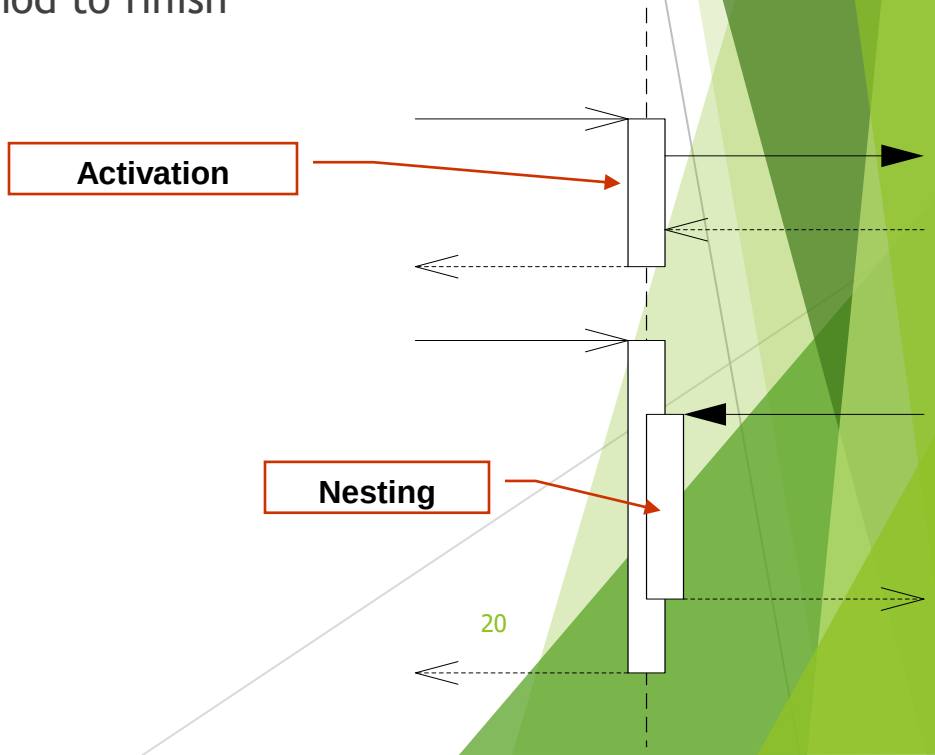
- *creation*: arrow with 'new' written above it
  - notice that an object created after the start of the scenario appears lower than the others
- *deletion*: an X at bottom of object's lifeline
  - Java doesn't explicitly delete objects; they fall out of scope and are garbage-collected



Indicating method calls

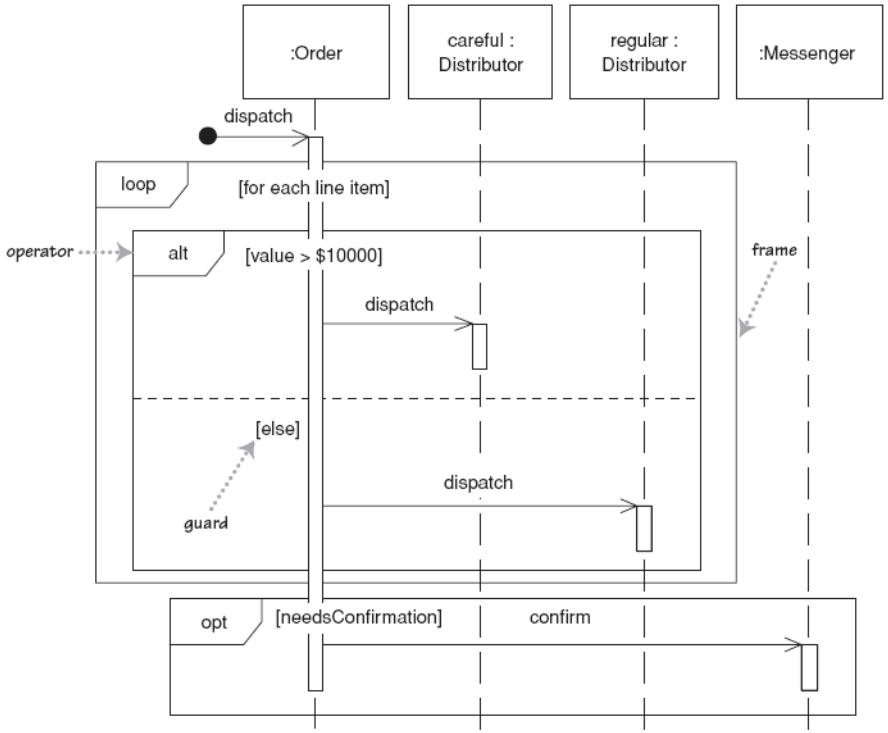
- **activation**: thick box over object's life line; drawn when object's method is on the stack
  - either that object is running its code, or it is on the stack waiting for another object's method to finish
  - nest to indicate recursion



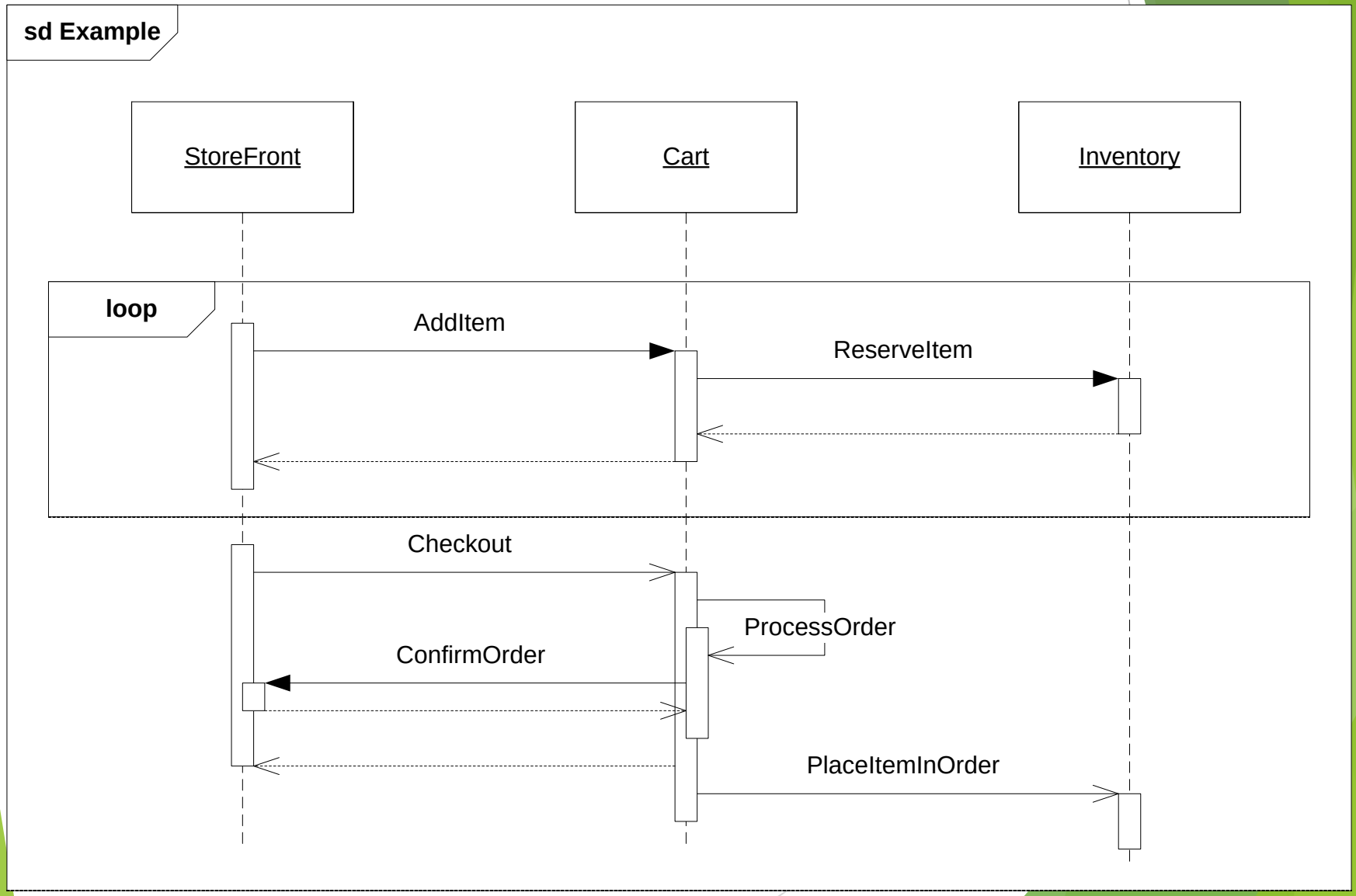


Indicating selection and loops

- frame: box around part of a sequence diagram to indicate selection or loop
  - `if` -> (opt) [condition]
  - `if/else` -> (alt) [condition], separated by horizontal dashed line
  - loop -> (loop) [condition or items to loop over]



Example sequence diagram



Benefits

- a good sequence diagram is still a bit above the level of the real code (not all code is drawn on diagram)
- sequence diagrams are language-agnostic (can be implemented in many different languages
- non-coders can do sequence diagrams
- easier to do sequence diagrams as a team
- can see many objects/classes at a time on same page (visual bandwidth)