UML Diagrams

SOME SLIDES ARE FROM HTTPS://WWW.IBM.COM/DEVELOPERWORKS/RATIONAL/LIBRARY/3101.HTML

What is UML?

- Standard language for specifying, visualizing, constructing, and documenting software system, business modeling and other non-software systems.
- The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.
- The UML is a very important part of developing object oriented software and the software development process.
- **The UML uses mostly graphical notations to express the design of software projects.**
- Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

Class diagram

UML class diagrams show the classes of the system, their inter-relationships, and the operations and attributes of the classes

- Explore domain concepts in the form of a domain model
- Analyze requirements in the form of a conceptual/analysis model
- Depict the detailed design of object-oriented or object-based software

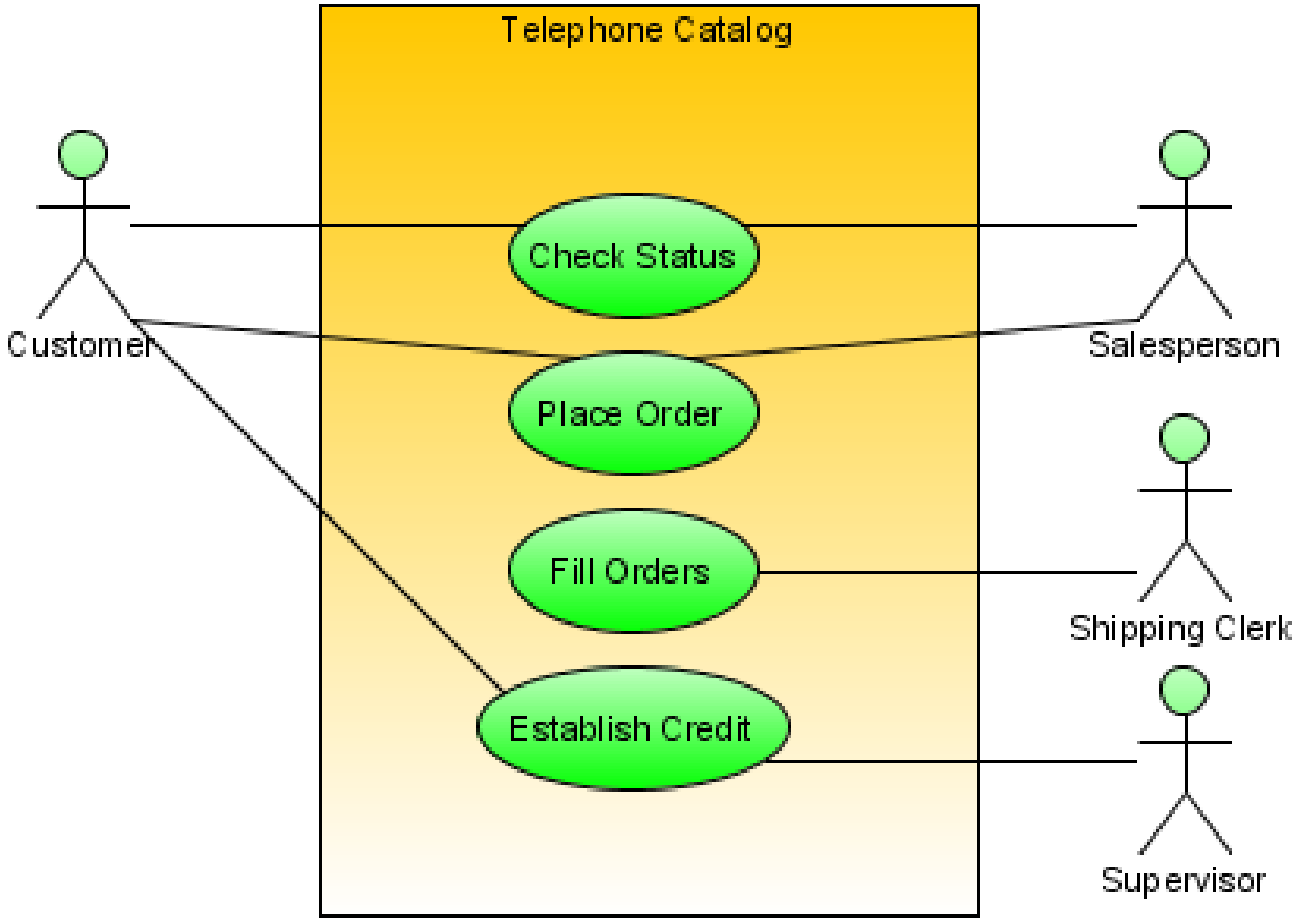
Use cases diagram

Use cases diagrams describes the behavior of the target system from an external point of view.

- **Use cases**. A use case describes a sequence of actions that provide something of measurable value to an actor and is drawn as a horizontal ellipse.
- **Actors**. An actor is a person, organization, or external system that plays a role in one or more interactions with your system. Actors are drawn as stick figures.
- **Associations**. Associations between actors and use cases are indicated by solid lines. An association exists whenever an actor is involved with an interaction described by a use case.

Use cases diagram

Use cases diagram



Use case

An example of a use case of **withdrawing** a cash from ATM machine:

Actors: Customer, ATM, Accounting system/Bank

Inputs: customer's card, PIN, Bank Account details

Normal operation: The customer inputs his/her card into the machine, and is prompted for a PIN whic h is entered on the keypad. If correct, he/she is presented with a menu of options. The Withdraw c ash option is selected. The customer is promoted with a request for the amount of cash required a nd inputs the amount. If there are sufficient funds in his account, the cash is dispensed, a receipt if printed and the account balance is updated. Before the cash is dispensed, the card is returned to t he customer who is prompted by the machine to take their card.

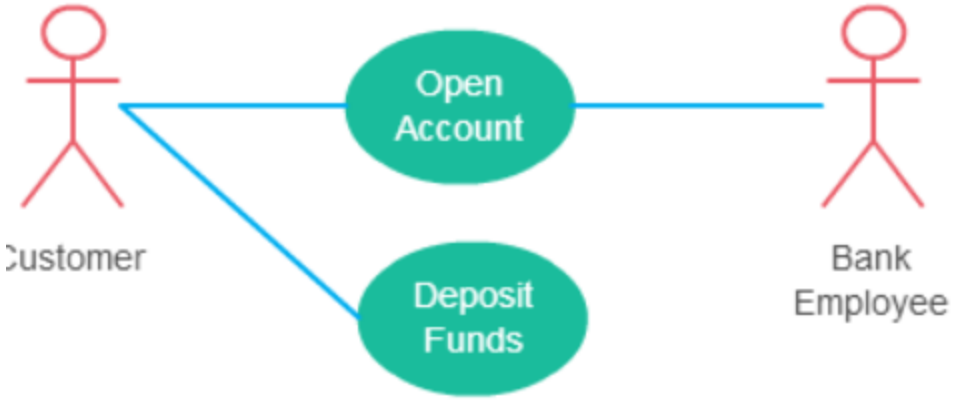
Exception: Invalid card. Card is retained by machine Customer advised to seek advise.

Relationship types in use case diagram

- Association between actor and use case
- Generalization of an actor
- Extend between two use cases
- Include between two use cases
- Generalization of a use case

Association between actor and use case

- An actor must be associated with at least one use case.
- An actor can be associated with multiple use cases.
- Multiple actors can be associated with a single use case.

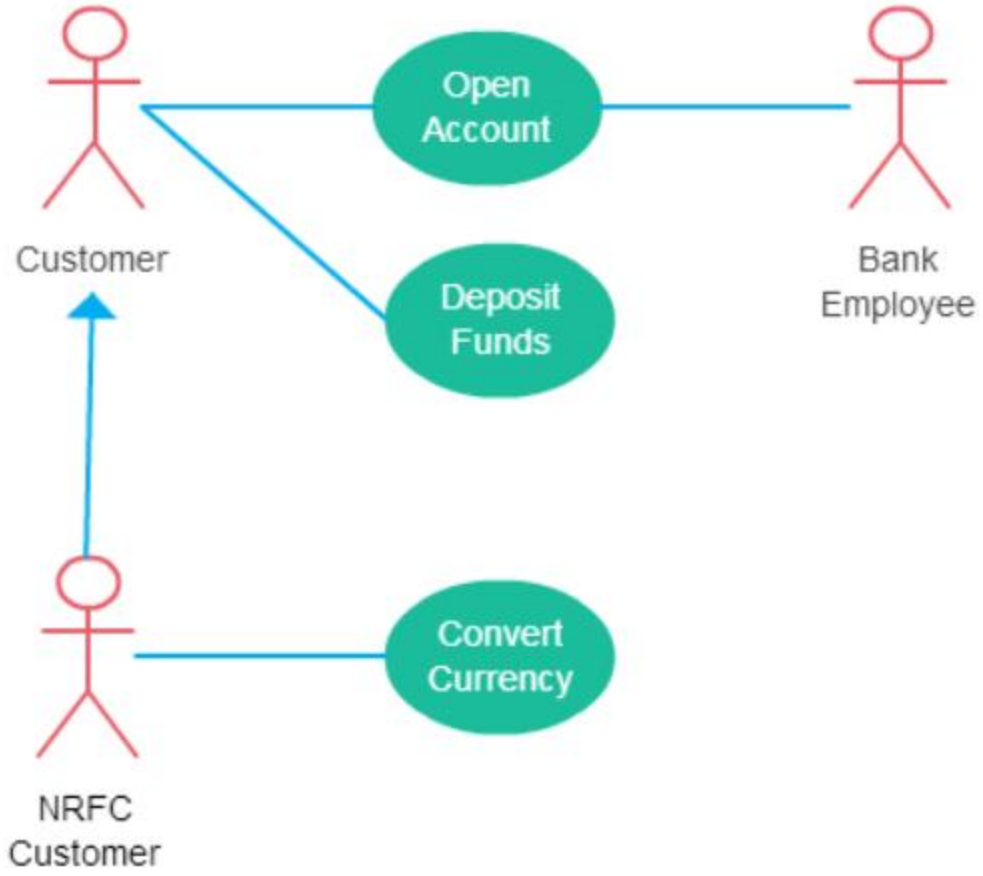


ifferent ways association relationship appears in use case diagrams

Ref:https://creately.com/blog/diagrams/use-case-diagram-relationships/

Generalization of an Actor

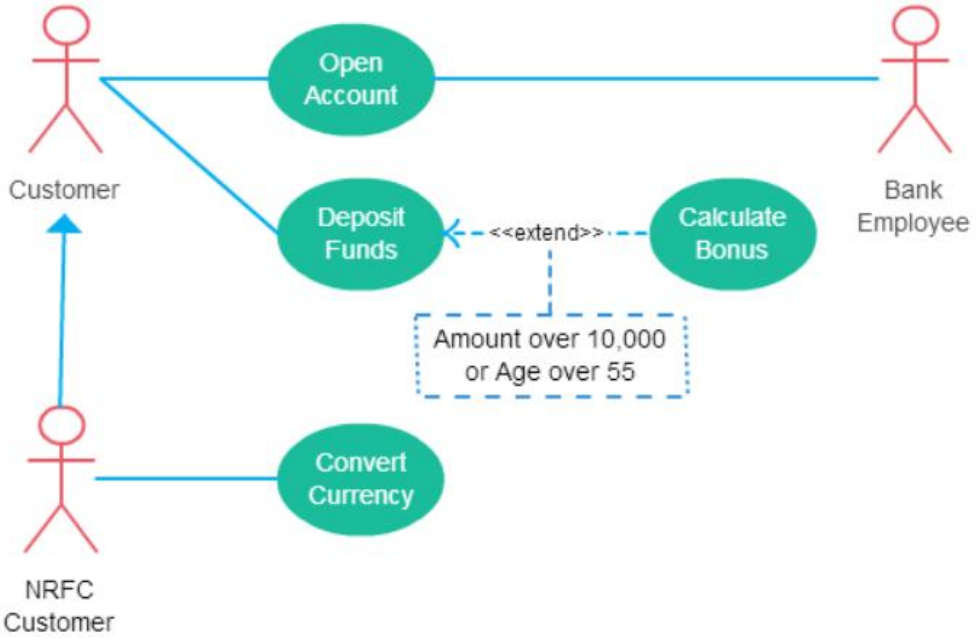
- Generalization of an actor means that one actor can inherit the role of the ot her actor.
- The descendant inherits all the use cases of the ancestor. The descendant has one or more use cases that are specific to that role.



A generalized actor in an use case diagram

Extend Relationship Between Two Use Cases

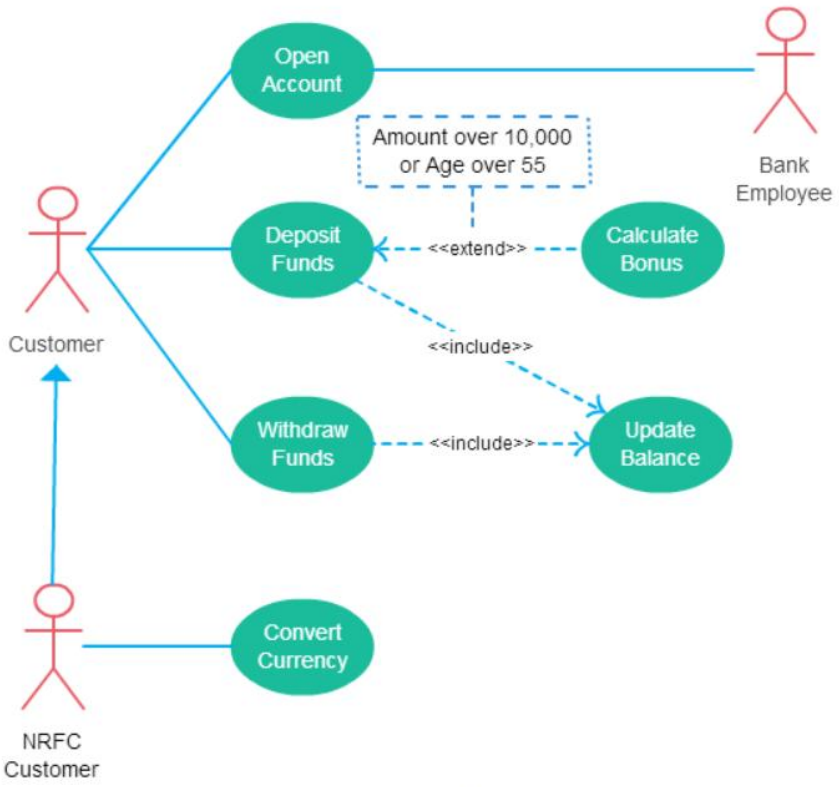
- Extend relationship in use cases extends the base use case and adds more functio nality to the system. The <<**extend**>> relationship concept:
  - **The extending use case is dependent on the extended (base) use case**. In the below diagram the " Calculate Bonus" use case doesn't make much sense without the "Deposit Funds" use case.
  - **The extending use case is usually optional** and can be triggered conditionally. In the diagram, you c an see that the extending use case is triggered only for deposits over 10,000 or when the age is over 5 5.
  - **The extended (base) use case must be meaningful on its own**. This means it should be independe nt and must not rely on the behavior of the extending use case. Extended use case stands on its own, without the extension



Extend relationship in use case diagrams

Include Relationship Between Two Use Cases

- Include relationship show that the behavior of the included use case is p art of the including (base) use case. The main reason for this is to reuse common actions across multiple use cases Few things to consider whe n using the <<include>> relationship.
  - The base use case is incomplete without the included use case.
  - The included use case is mandatory and not optional.



Includes is usually used to model common behavior

Generalization of a Use Case

- This is similar to the generalization of an actor.
- The behavior of the ancestor is inherited by the descendant. This is used when there is common behavior between two use cases and al so specialized behavior specific to each use case.
- There might be a use case called “Pay Bills”. This can be generalize d to “Pay by Credit Card”, “Pay by Bank Balance” etc.

Sequence diagram

- Sequence diagrams model the collaboration of objects based on a time sequence. It shows how the objects interact with others in a particular scenario of a use case.

UML sequence diagrams

- **sequence diagram**: an "interaction diagra m" that models a single scenario executing in the system
  - perhaps 2nd most used UML diagram (behind class diagram)
- relation of UML diagrams:
  - CRC cards -> class diagram
  - use cases -> sequence diagrams

Key parts of a sequence diag.

- **participant**: an object or entity that acts in th e sequence diagram
  - sequence diagram starts with an unattached "fo und message" arrow
- **message**: communication between participa nt objects
- the axes in a sequence diagram:
  - horizontal: which object/participant is acting
  - vertical: time (down -> forward in time)

Sequence diagrams

- Vertical line is called an object's **lifeline**
  - Represents an object's life during interaction
- Object deletion denoted by X, ending a lifeline
  - Horizontal arrow is a message between two objects
- Order of messages sequences top to bottom
- Messages labeled with message name
  - Optionally arguments and control information
- Control information may express conditions:
  - such as [hasStock], or iteration
- Returns (dashed lines) are optional
  - Use them to add clarity

Synchronous vs Asynchronous method calls

- Synchronous code is executed in sequence – each statement w aits for the previous statement to finish before executing.
- Asynchronous code doesn't have to wait – your program can c ontinue to run. You do this to keep your site or app responsive, reducing waiting time for the user.

- Asynchronous method call, invoke a method and don’t wait to h ear back:

- Synchronous method calls: invoke a method, and wait to hear b ack before proceeding. So there is a need for an arrow back to hear from the method call.

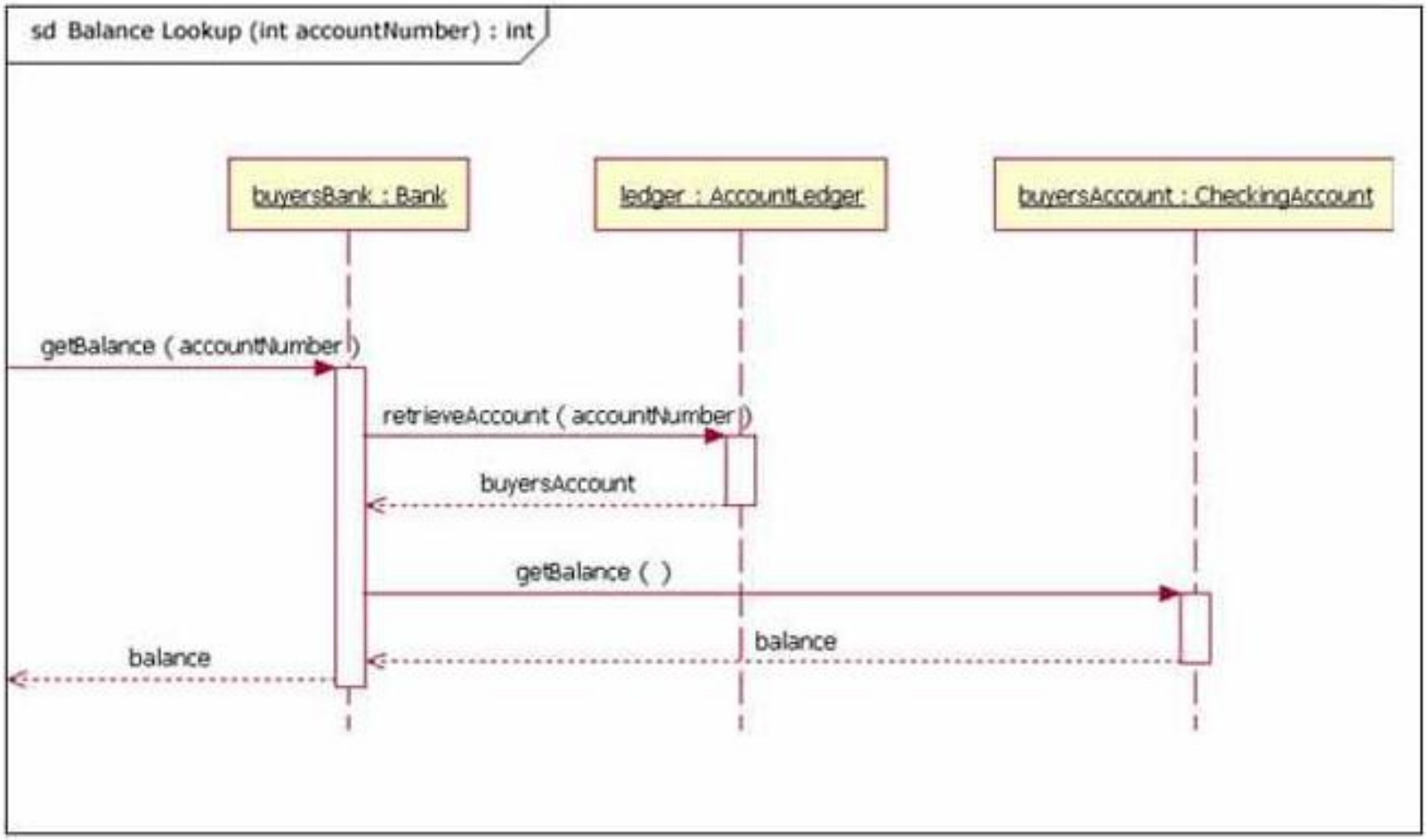
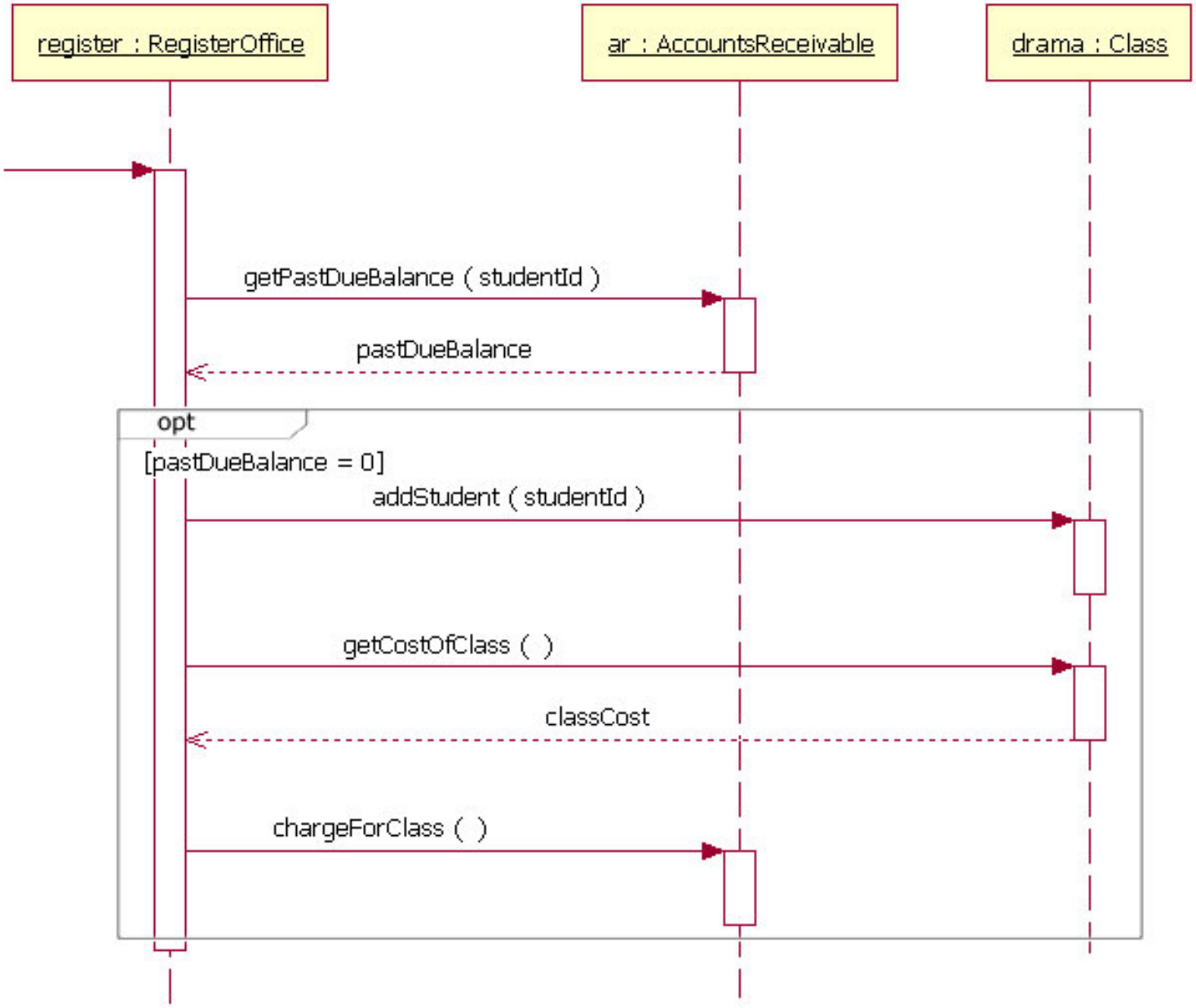
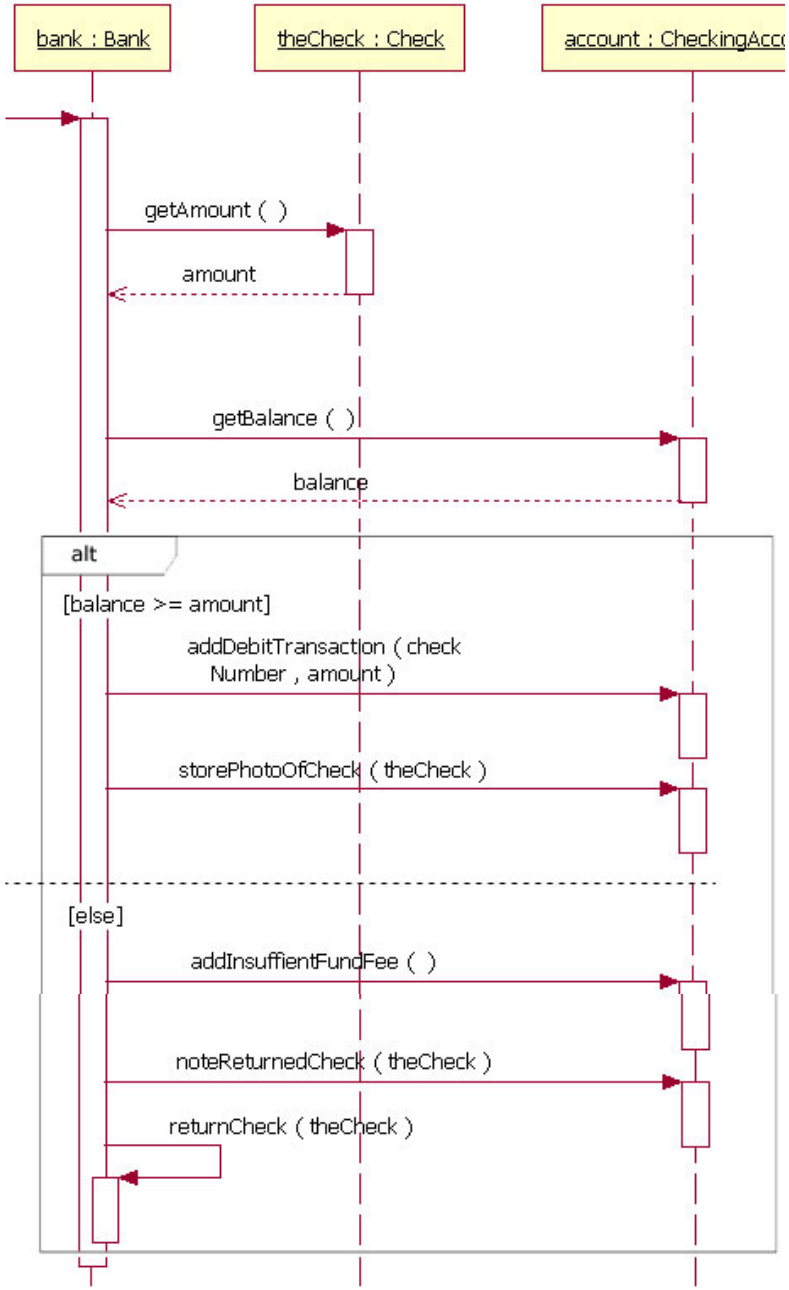

sd Balance Lookup (int accountNumber) : int
buyersBank : Bank
ledger : AccountLedger
buyersAccount : CheckingAccount
getBalance (accountNumber)
retrieveAccount (accountNumber)
buyersAccount
getBalance ( )
balance
balance

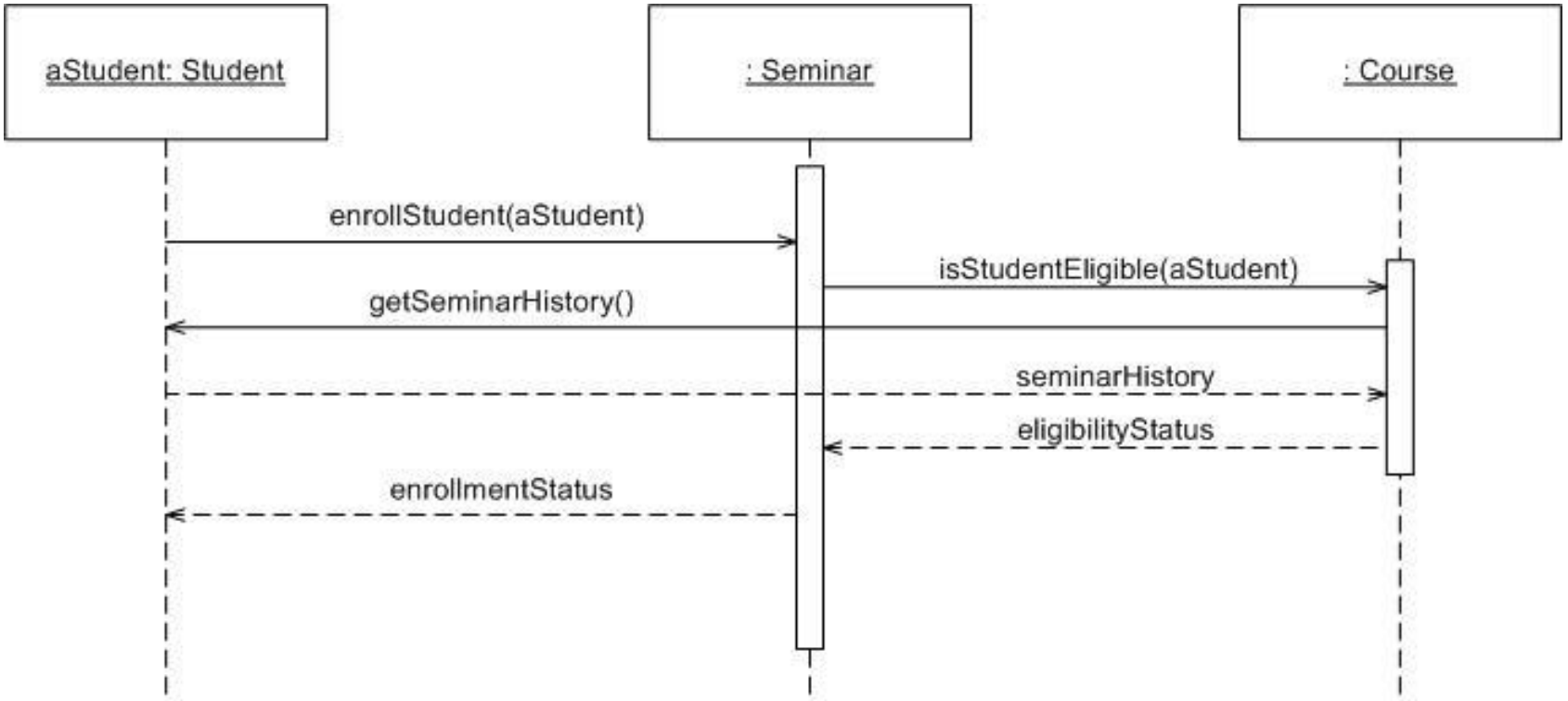
Figure 9. A sequence diagram fragment that includes an option combination fragment



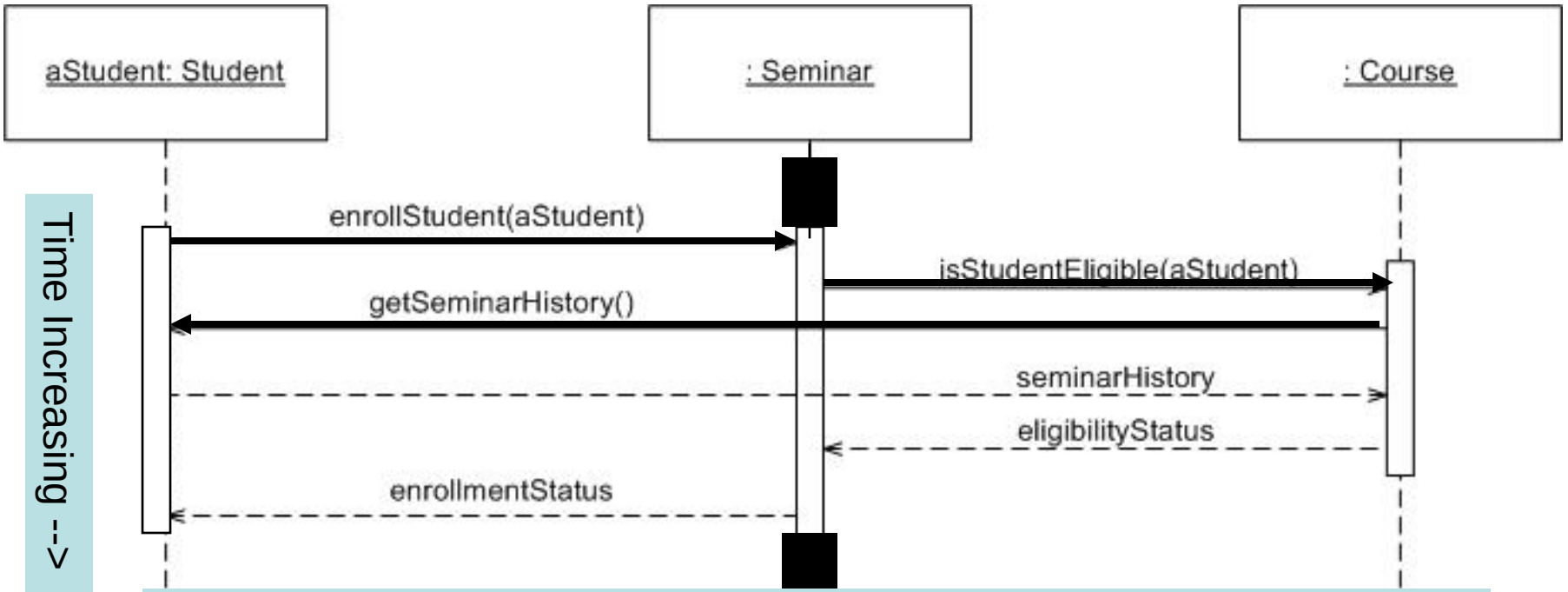
A Sequence diagram fragment that contains an alternative combination fragment



Sequence diagram



Sequence Diagram

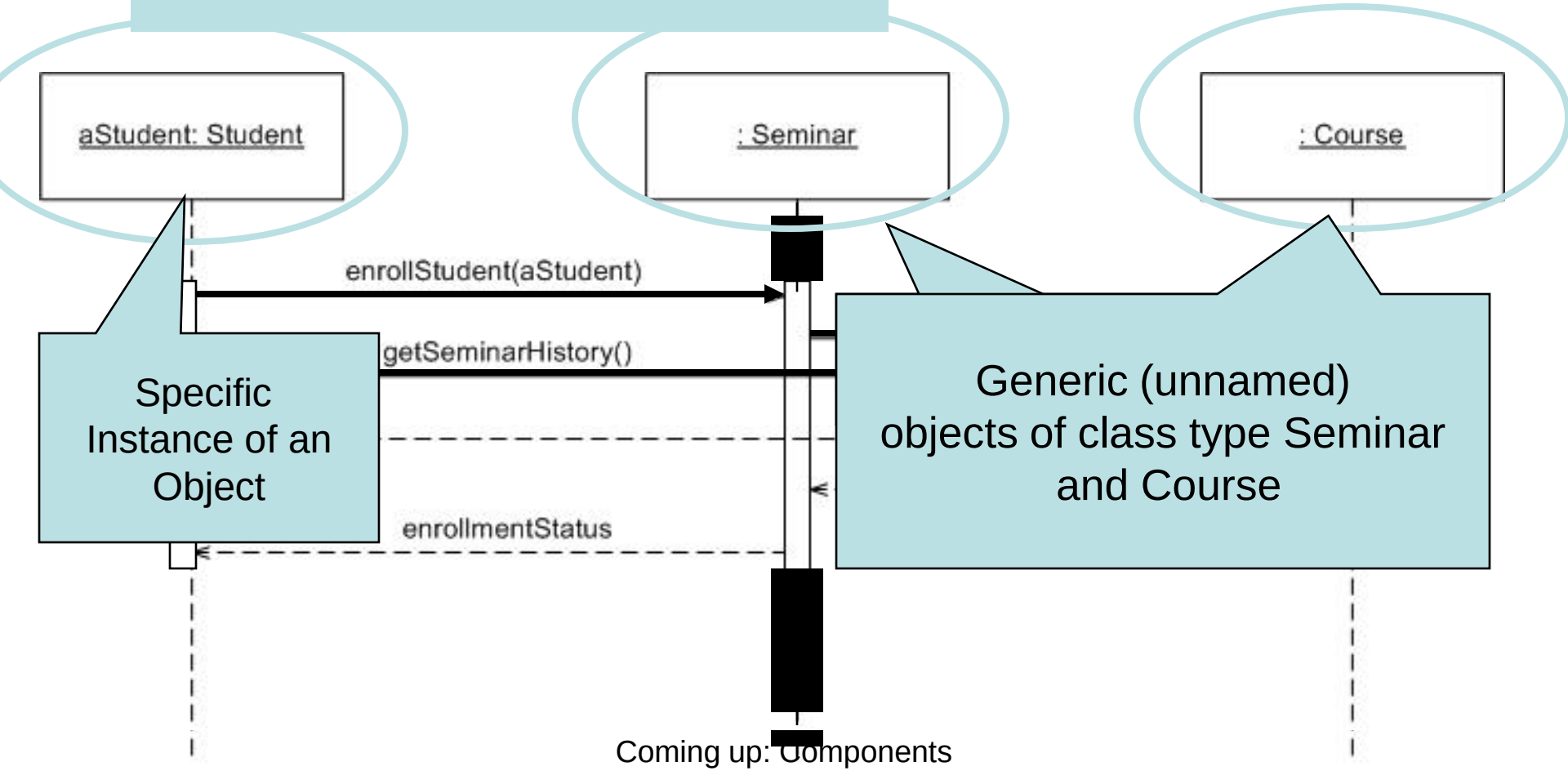


All lines should be horizontal to indicate instantaneous actions.Additionally if ActivityA happens before ActivityB, ActivityA must be above activity A

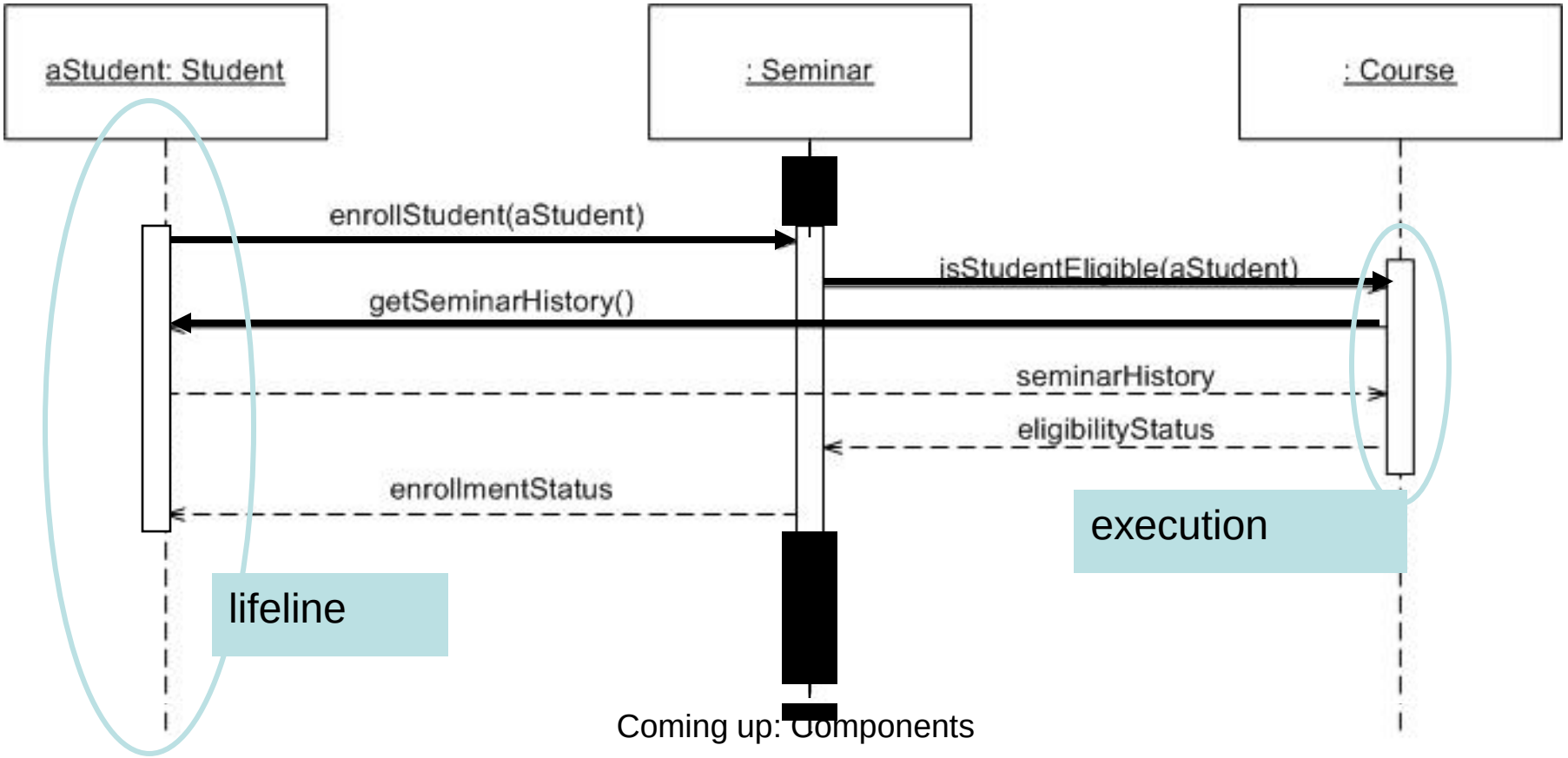
Lower = Later!

Components

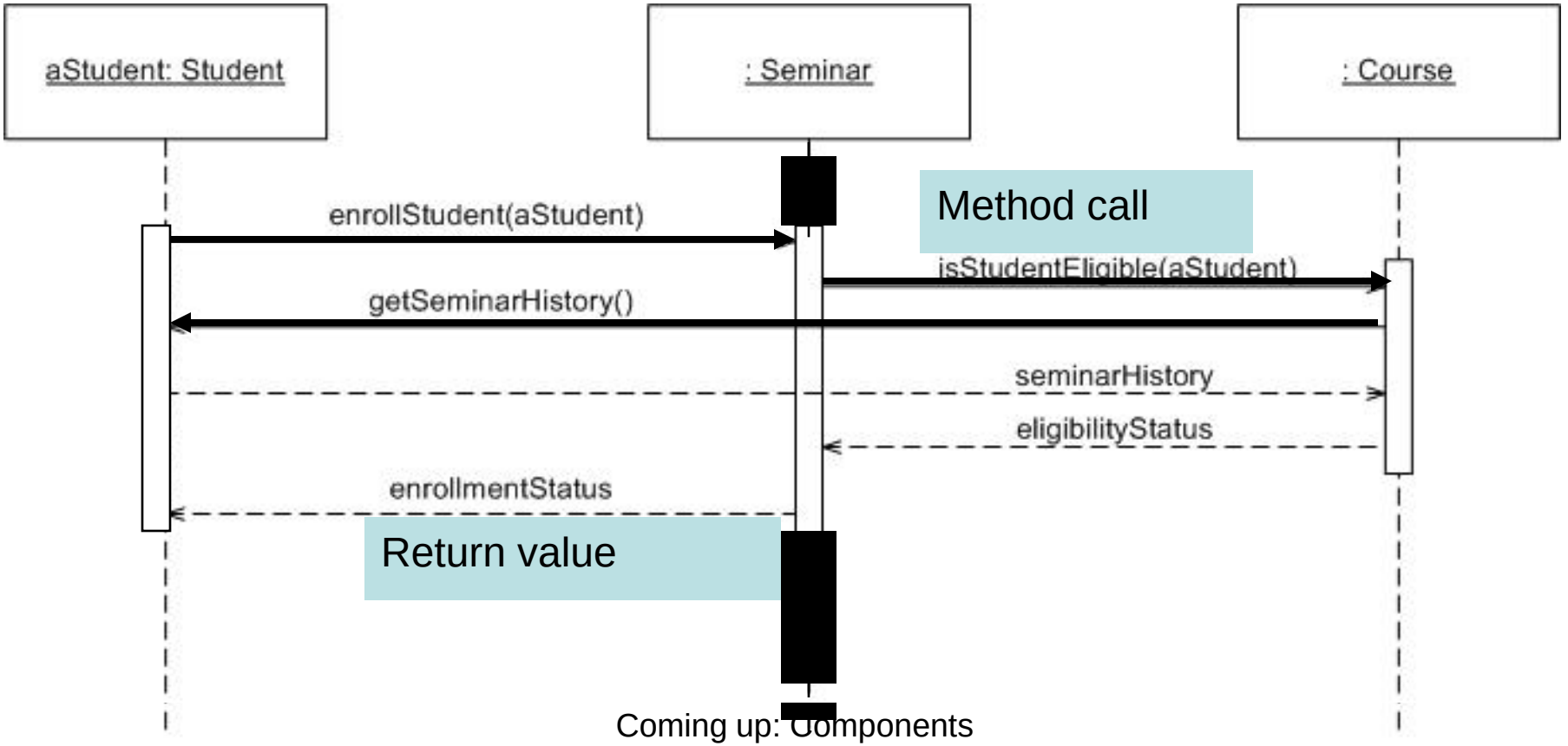
Objects: aStudent is a specific instance of the Student class



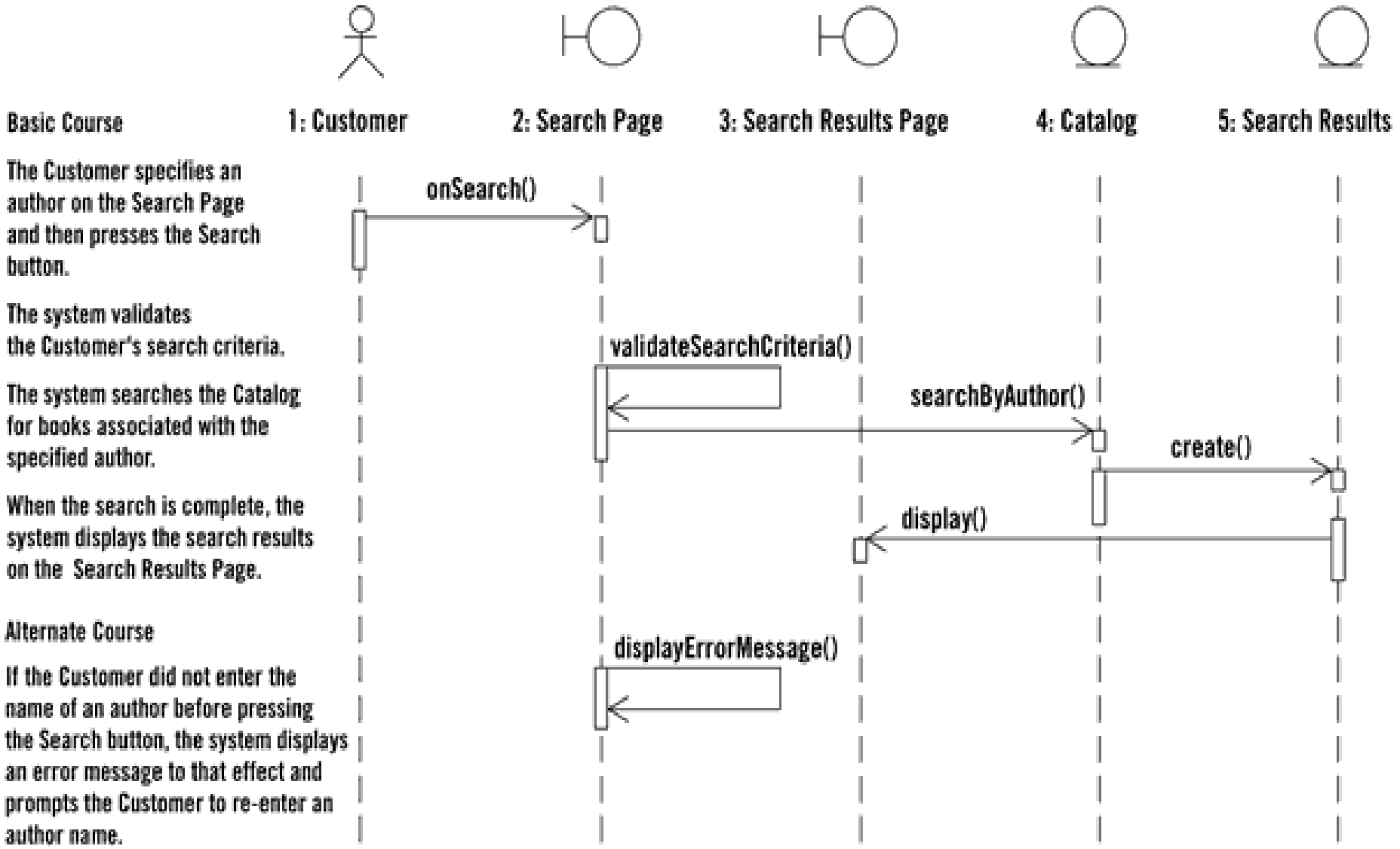
Components


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

Components

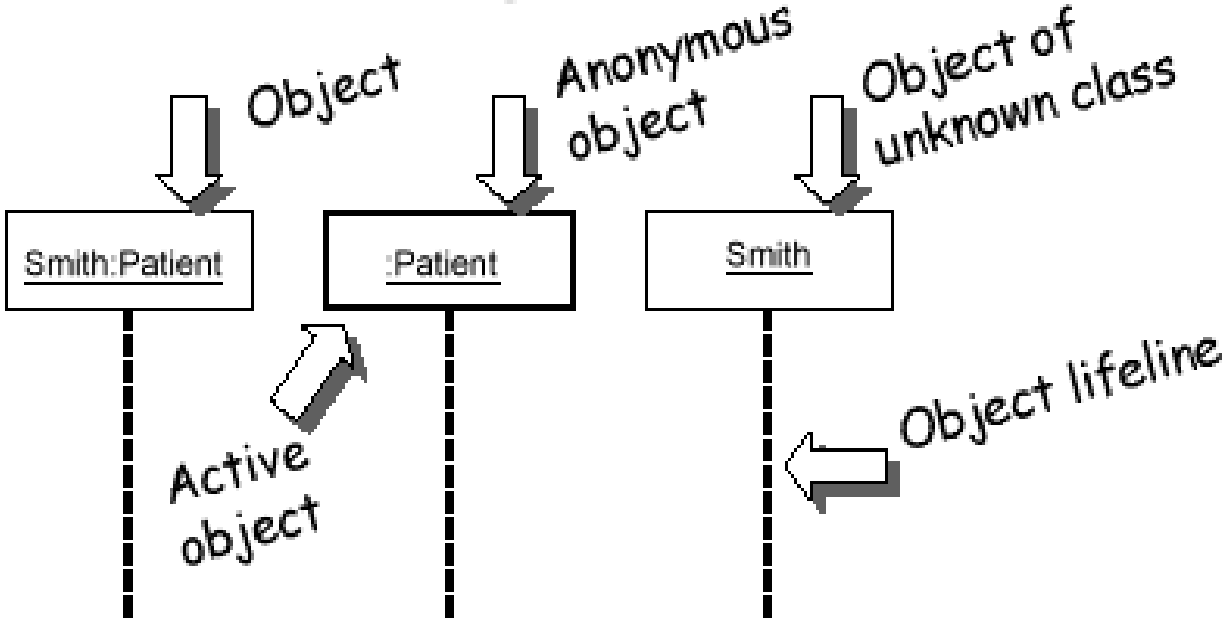

aStudent: Student
: Seminar
: Course
enrollStudent(aStudent)
Method call
isStudentEligible(aStudent)
getSeminarHistory()
seminarHistory
eligibilityStatus
enrollmentStatus
Return value

Sequence dg. from use case



Representing objects

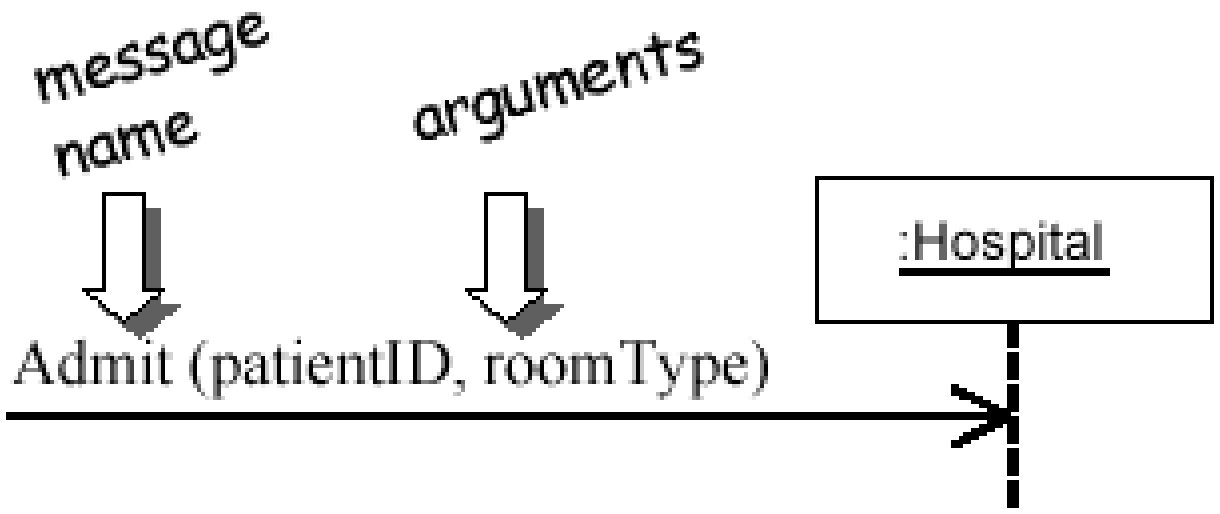
- Squares with object type, optionally prece ded by object name and colon
  - write object's name if it clarifies the diagram
  - object's "life line" represented by dashed vert.



Name syntax: <objectname>:<classname>

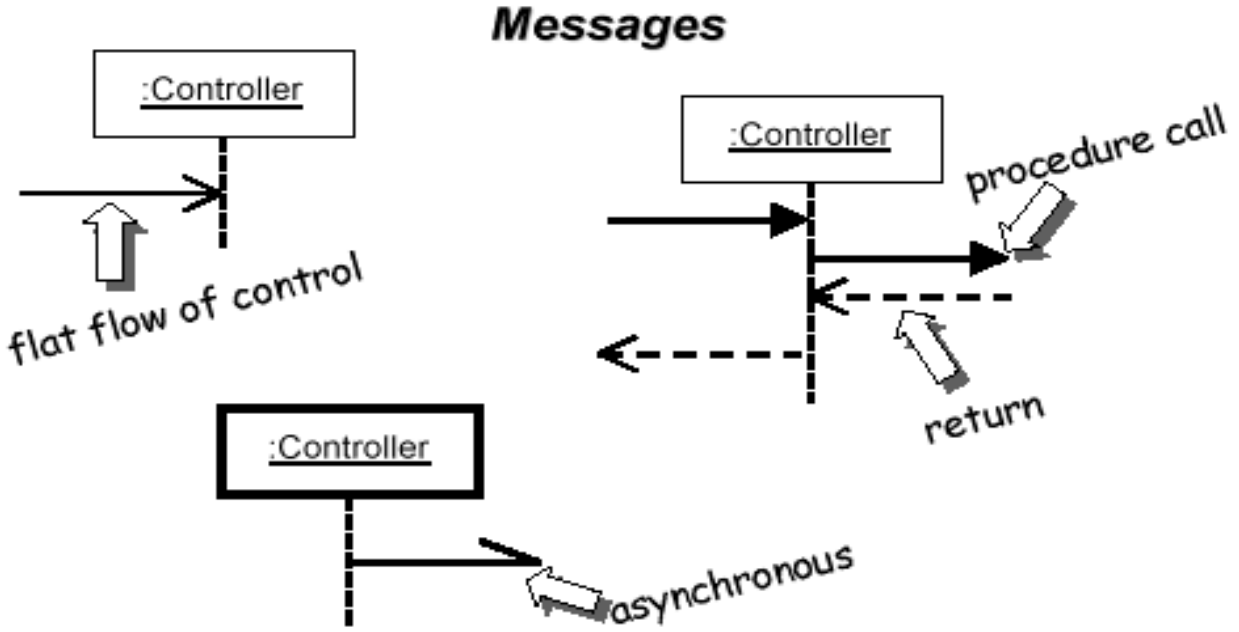
Messages between objects

- message (method call) indicated by horizontal ar row to other object
  - write message name and arguments above ar row



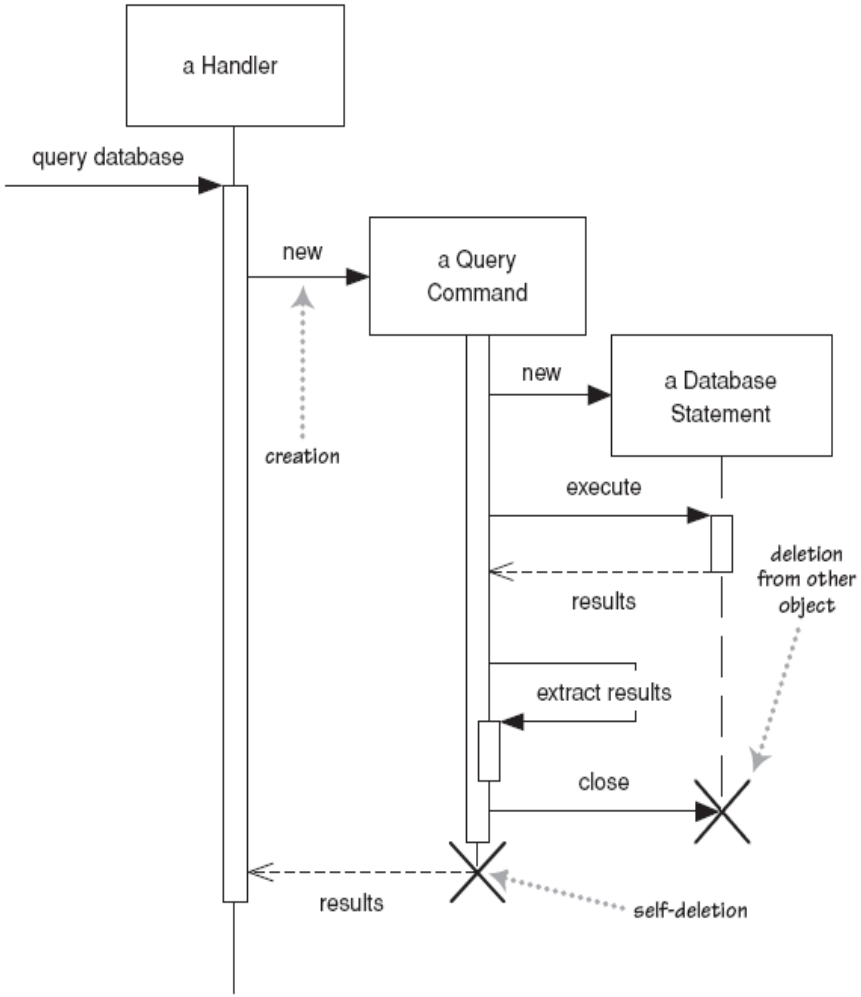
Messages, continued

- message (method call) indicated by horizontal arr ow to other object
  - dashed arrow back indicates return
  - different arrowheads for normal / concurrent (as



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



From Use Case to Sequence System Diagram

How to construct an SSD from a use case:

1. Draw System as black box on right side
2. For each actor that directly operates on the System, draw a stick figure and a lifeline.
3. For each System events that each actor generate s in use case, draw a message.
4. Optionally, include use case text to left of diagram.

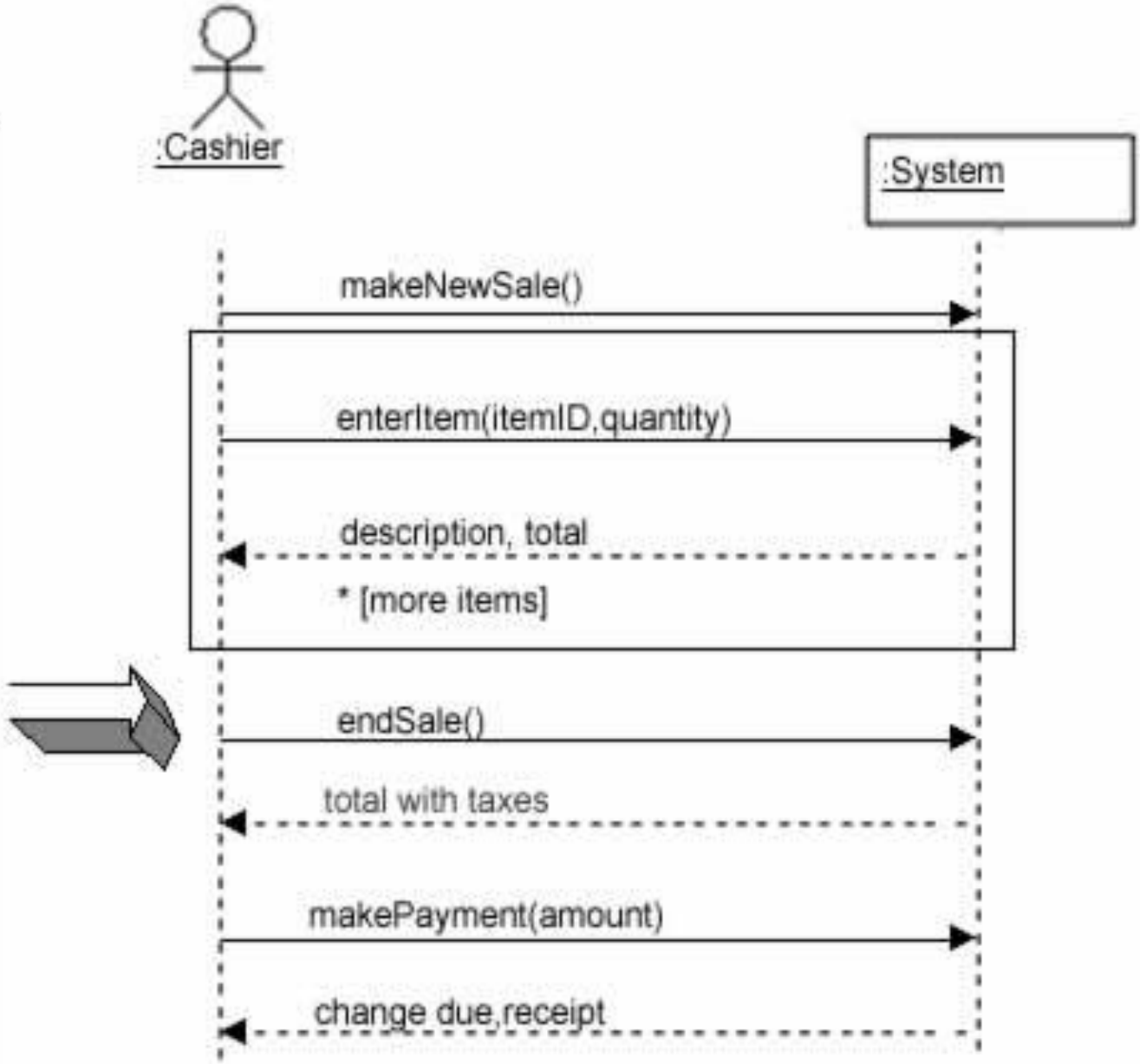
Example: use cases to SSD

Simple Cash-only Process Sale scenario:

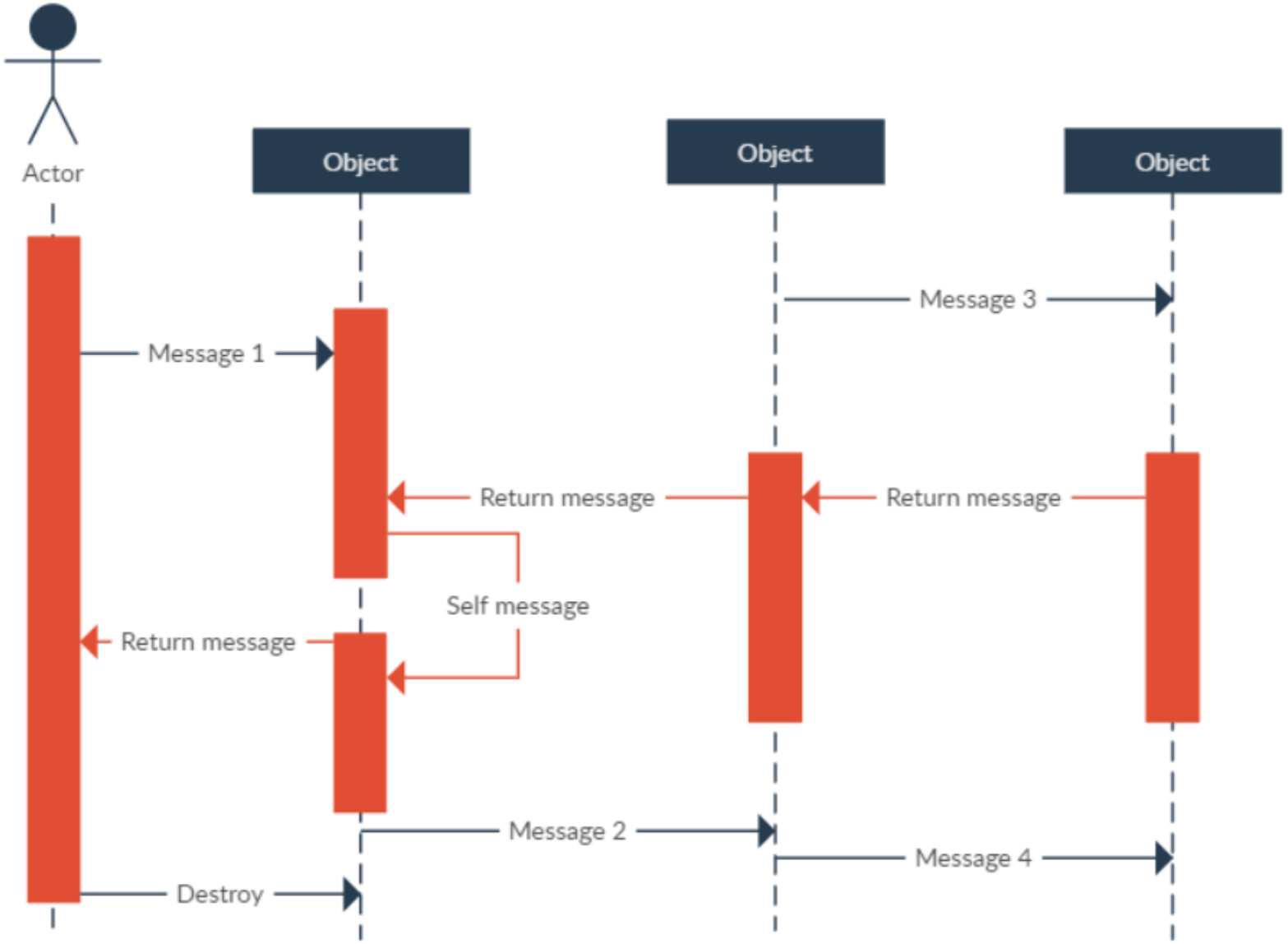
1.customer arrives at aPOS check out with goods and/or services to purchase.
2.Cashier starts a new sale.
3.Cashier enters a new item identifier.
4.System records new sale line item and presents item description,price and running total.

Cashier repeats steps steps 3-4 until indicates done.

5.System presents total with taxes calculated.
6.Cashier tells Customer the total, and asks for payment.
7.Customer pays and System handles payment.
....



Sequence Diagram Template (Sequence Diagram (UML)



https://creately.com/blog/diagrams/sequence-diagram-tutorial/

SynFor yet another supplementary material on info on sequence diagram visit: https://creately.com/blog/diagrams/sequence-diagram-tutorial/

You can use one of the many free available online tools for UML diagrams, such as Creatly and online visual paradigm tool (http://www.visual-paradigm.com/)