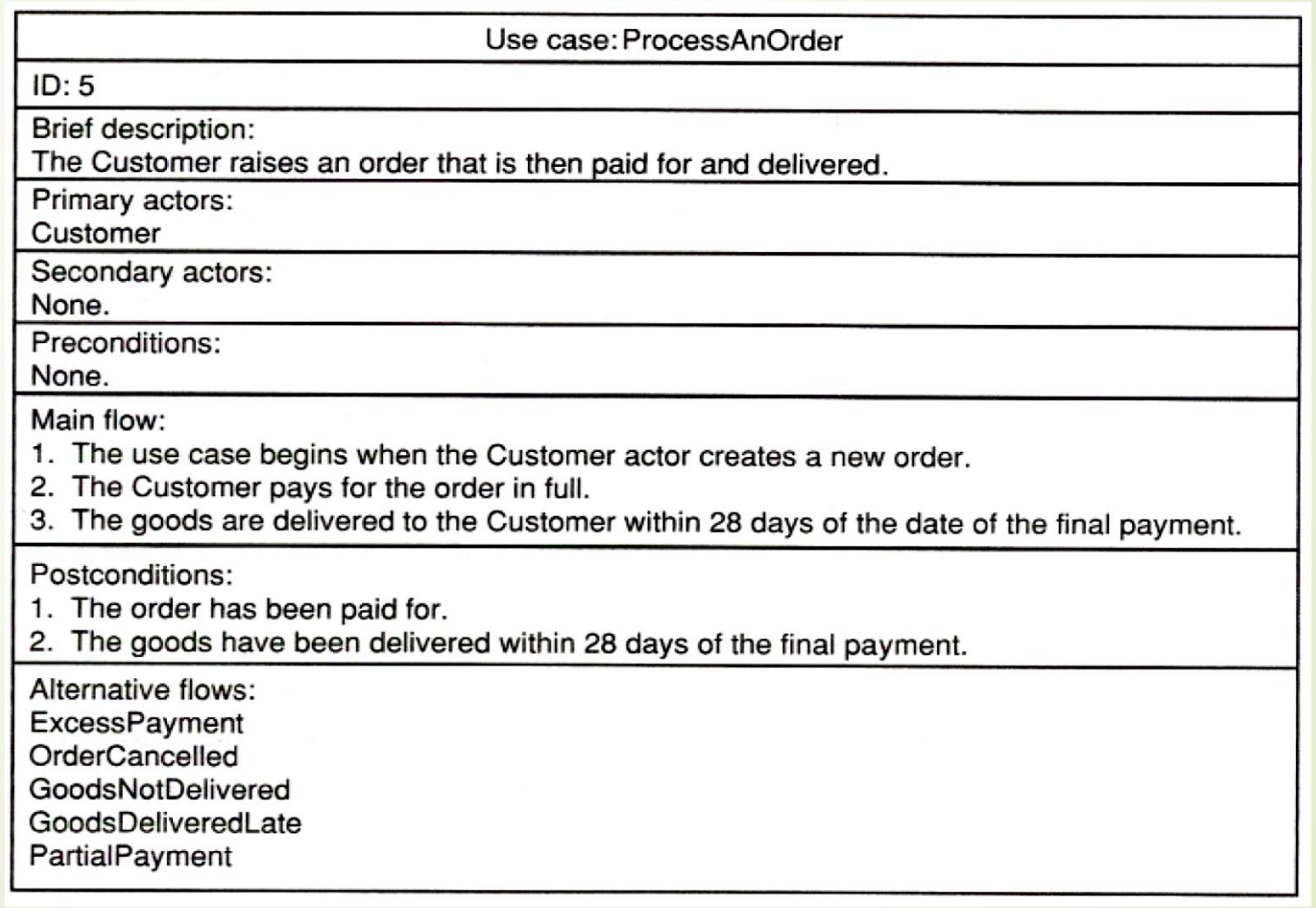
Sequence Diagrams

Lecturer: Adel Vahdati

Sequence Diagrams

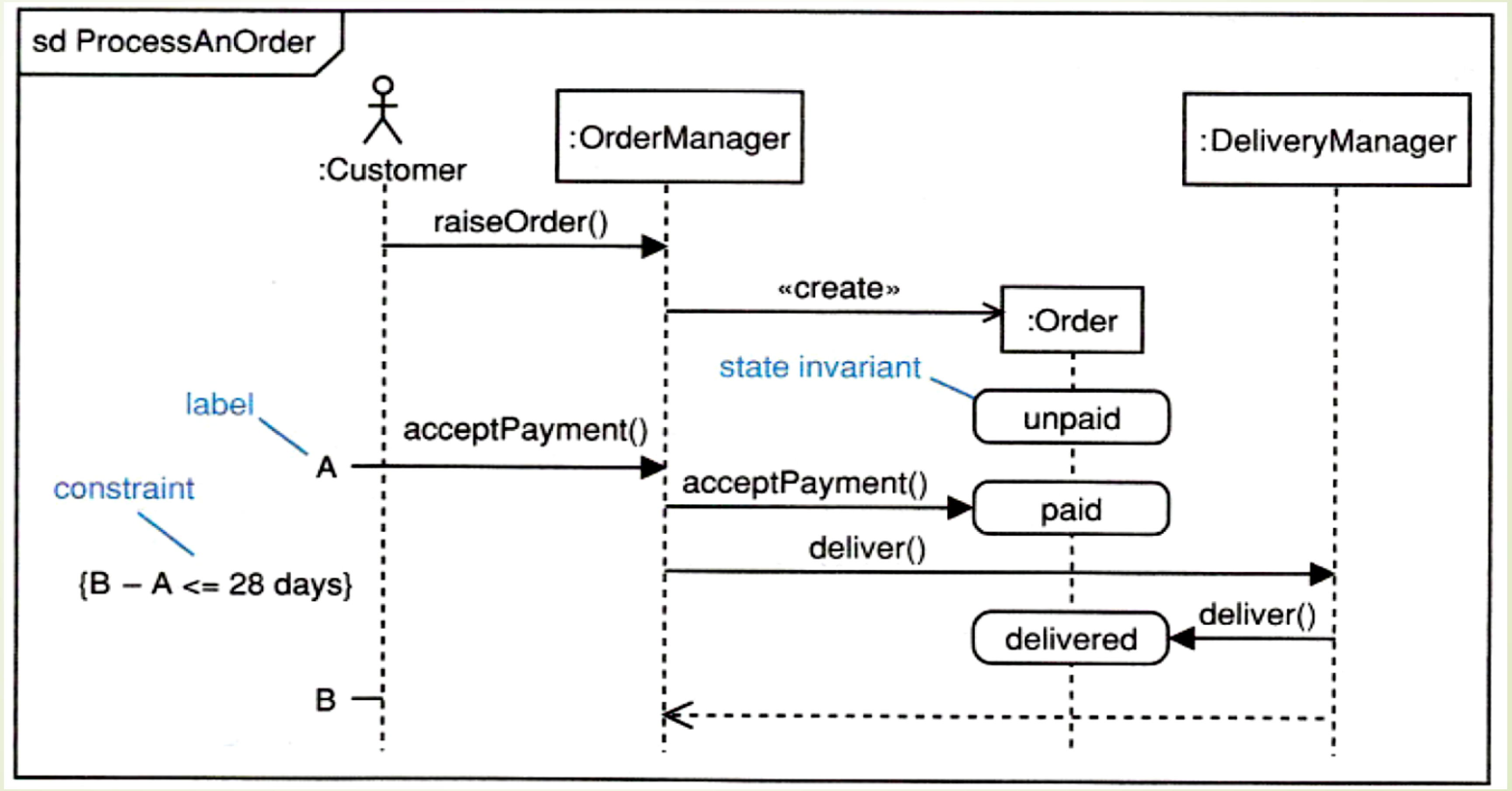
- Emphasize time-ordered sequence of message sends.
- Demonstrate how objects interact to realize the use case behavior;

Use Case Example



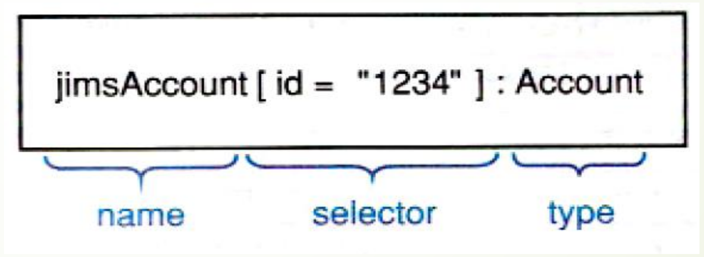
Use case: ProcessAnOrder
ID: 5
Brief description: The Customer raises an order that is then paid for and delivered.
Primary actors: Customer
Secondary actors: None.
Preconditions: None.
Main flow: 1. The use case begins when the Customer actor creates a new order. 2. The Customer pays for the order in full. 3. The goods are delivered to the Customer within 28 days of the date of the final payment.
Postconditions: 1. The order has been paid for. 2. The goods have been delivered within 28 days of the final payment.
Alternative flows: ExcessPayment OrderCancelled GoodsNotDelivered GoodsDeliveredLate PartialPayment

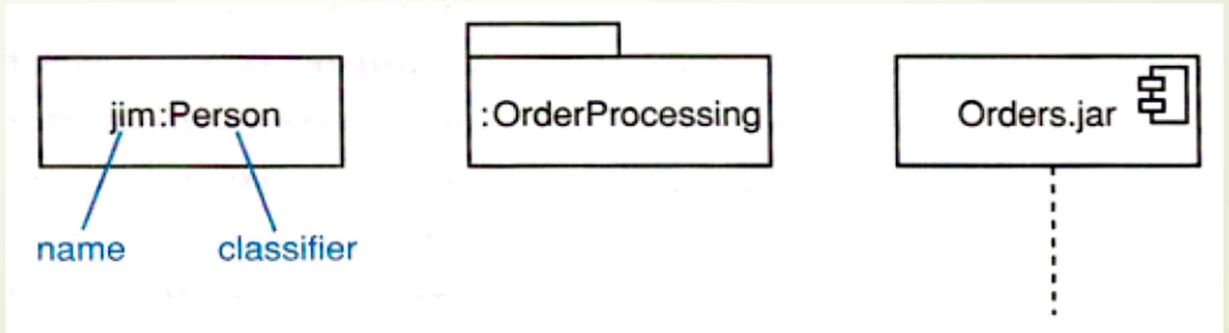
Sequence Diagrams: Use Case Realization



Lifelines

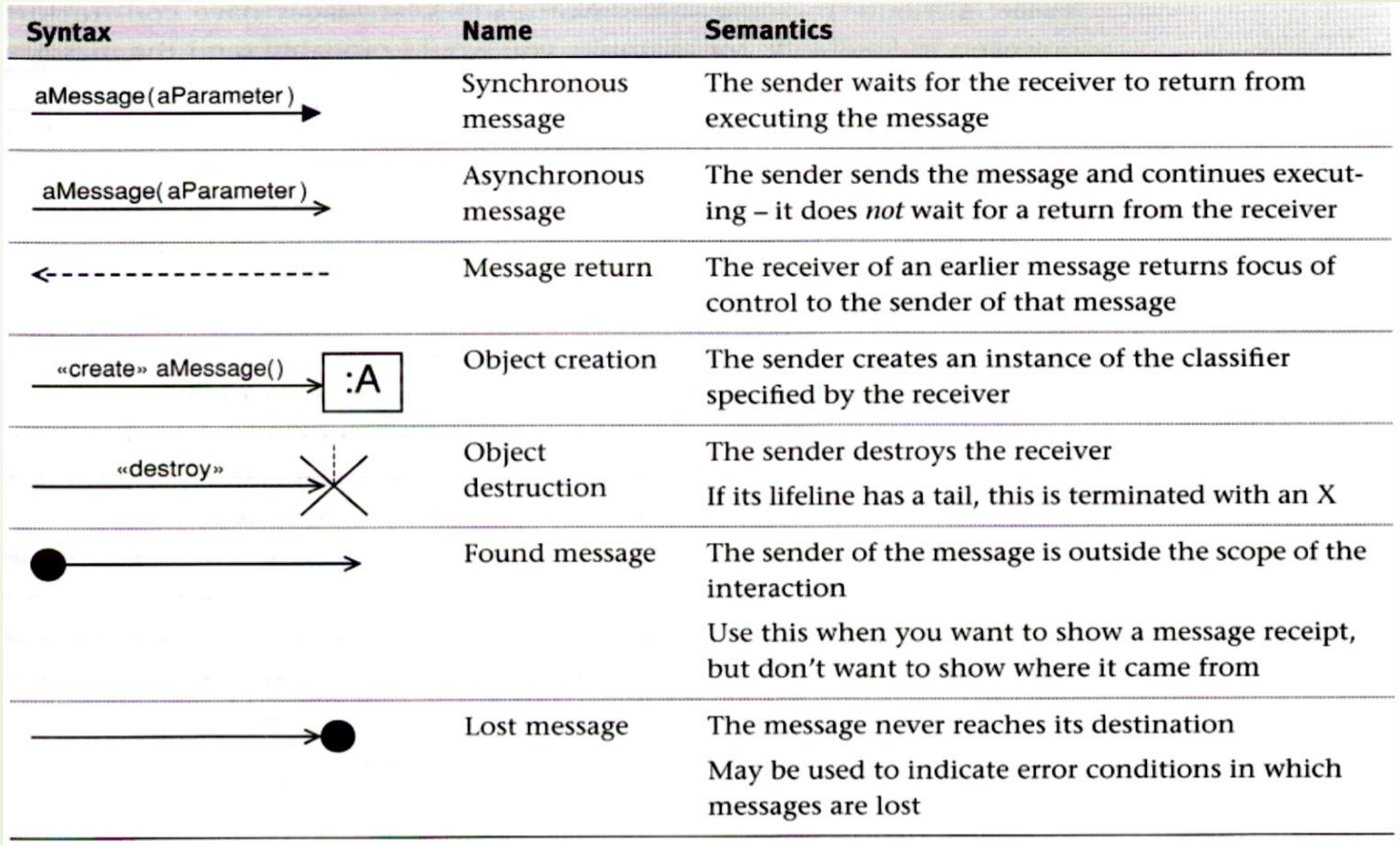
- A lifeline represents a participant in an interaction - how an instance of a classifier participates in the interaction.
  - Each lifeline has an optional name, a type, and an optional selector.
  - Each lifeline is drawn with the same icon as its type.





Messages

- A message represents a specific kind of communication between two lifelines in an interaction.

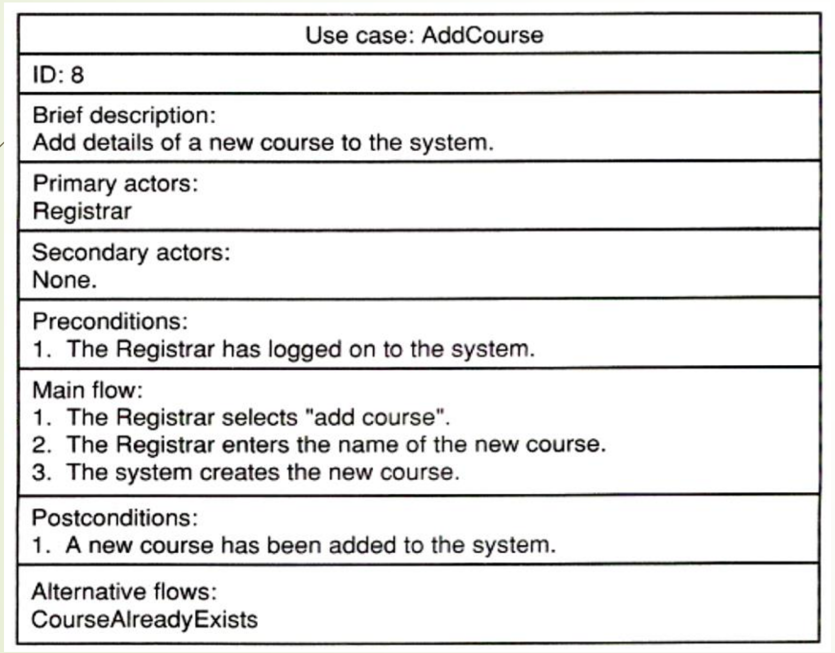


Syntax	Name	Semantics
aMessage(aParameter) ⟶	Synchronous message	The sender waits for the receiver to return from executing the message
aMessage(aParameter) →	Asynchronous message	The sender sends the message and continues executing – it does *not* wait for a return from the receiver
⇠	Message return	The receiver of an earlier message returns focus of control to the sender of that message
«create» aMessage() → :A	Object creation	The sender creates an instance of the classifier specified by the receiver
«destroy» → ✕	Object destruction	The sender destroys the receiver If its lifeline has a tail, this is terminated with an X
● →	Found message	The sender of the message is outside the scope of the interaction Use this when you want to show a message receipt, but don't want to show where it came from
→ ●	Lost message	The message never reaches its destination May be used to indicate error conditions in which messages are lost

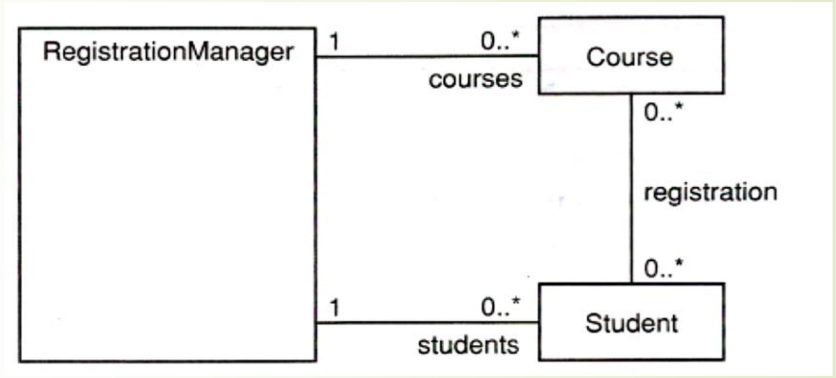
General Notation

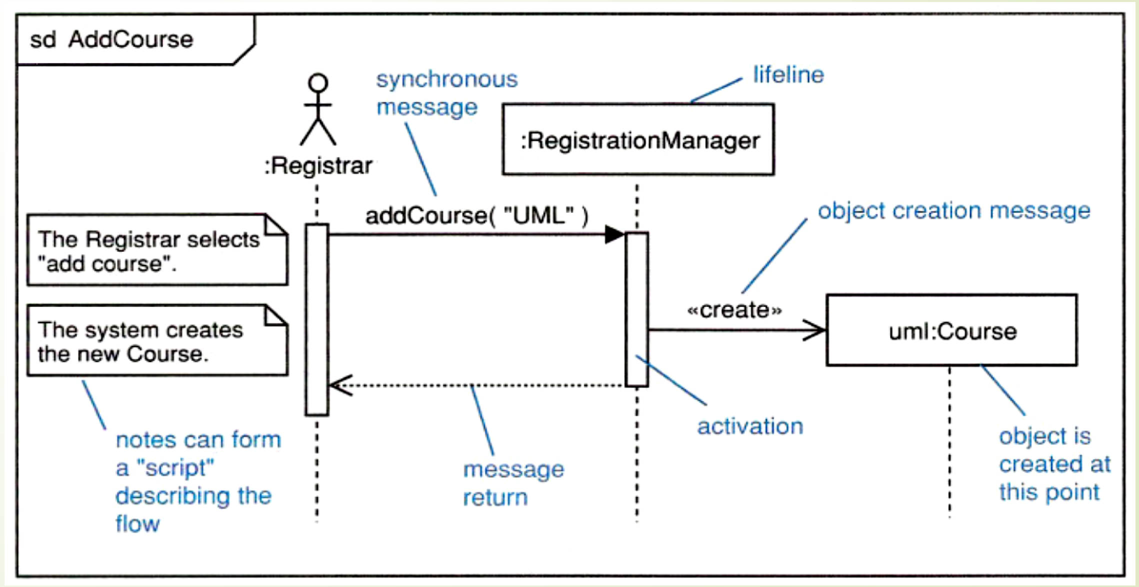
- Time runs top to bottom.
- Lifelines run left to right:
  - lifelines have dashed vertical tails that indicate the duration of the lifeline;
  - lifelines may have activations to indicate when the lifeline has focus of control;
  - organize lifelines to minimize the number of crossing lines.
- Place explanatory scripts down the left-hand side of the sequence diagram.
- place state symbols on the lifeline at the appropriate points.
- Constraints - place constraints in {} on or near lifelines.

Use Case Realizations

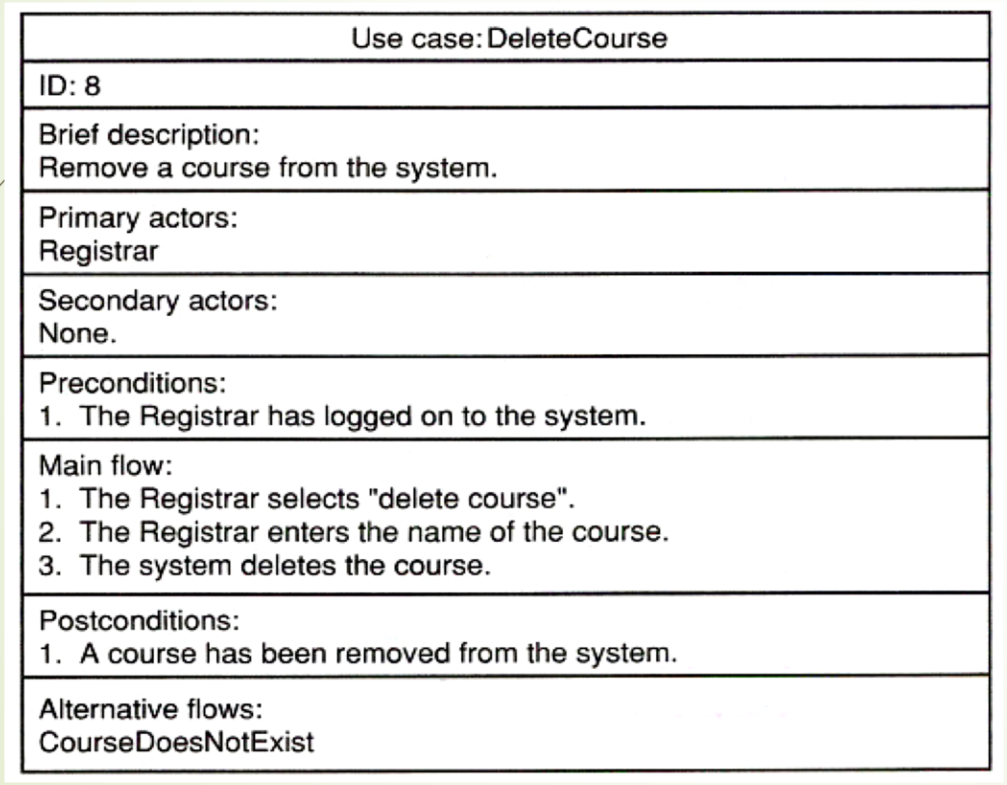


Use case: AddCourse
ID: 8
Brief description: Add details of a new course to the system.
Primary actors: Registrar
Secondary actors: None.
Preconditions: 1. The Registrar has logged on to the system.
Main flow: 1. The Registrar selects "add course". 2. The Registrar enters the name of the new course. 3. The system creates the new course.
Postconditions: 1. A new course has been added to the system.
Alternative flows: CourseAlreadyExists

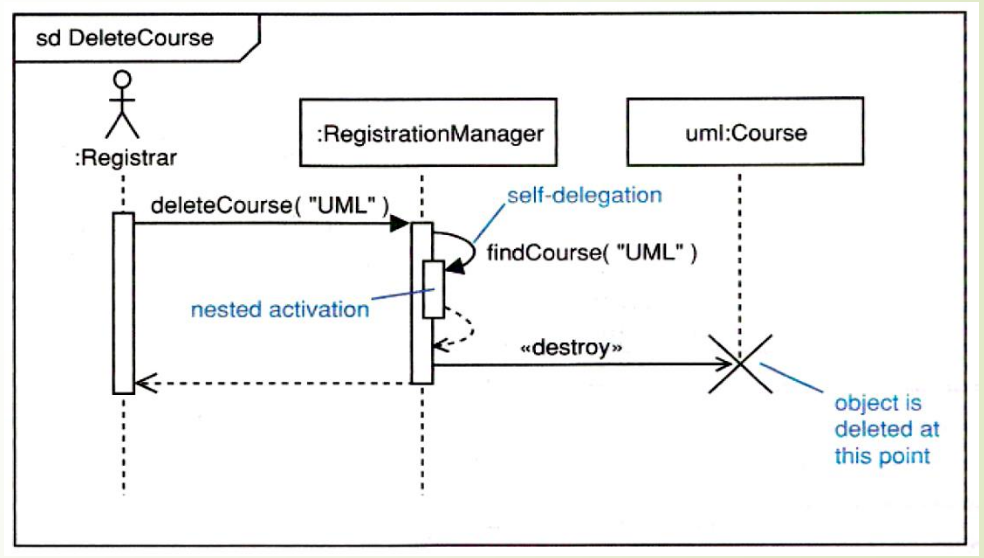




Use Case Realizations

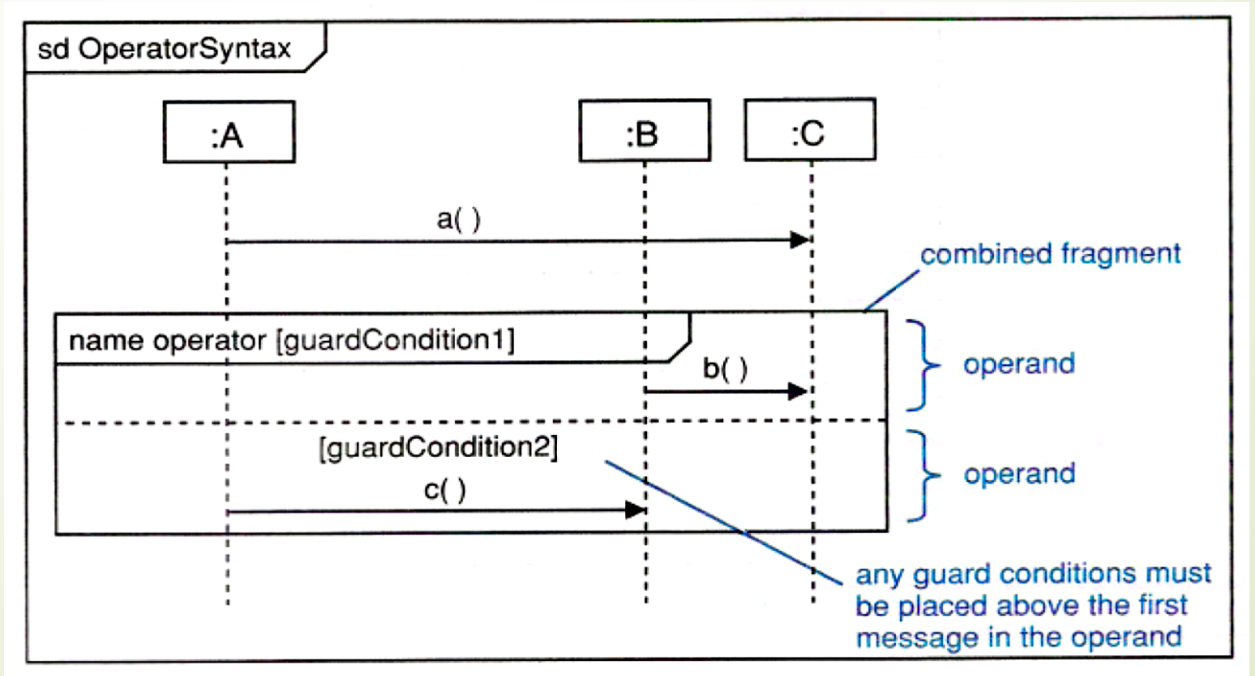


Use case: DeleteCourse
ID: 8
Brief description: Remove a course from the system.
Primary actors: Registrar
Secondary actors: None.
Preconditions: 1. The Registrar has logged on to the system.
Main flow: 1. The Registrar selects "delete course". 2. The Registrar enters the name of the course. 3. The system deletes the course.
Postconditions: 1. A course has been removed from the system.
Alternative flows: CourseDoesNotExist



Combined Fragments

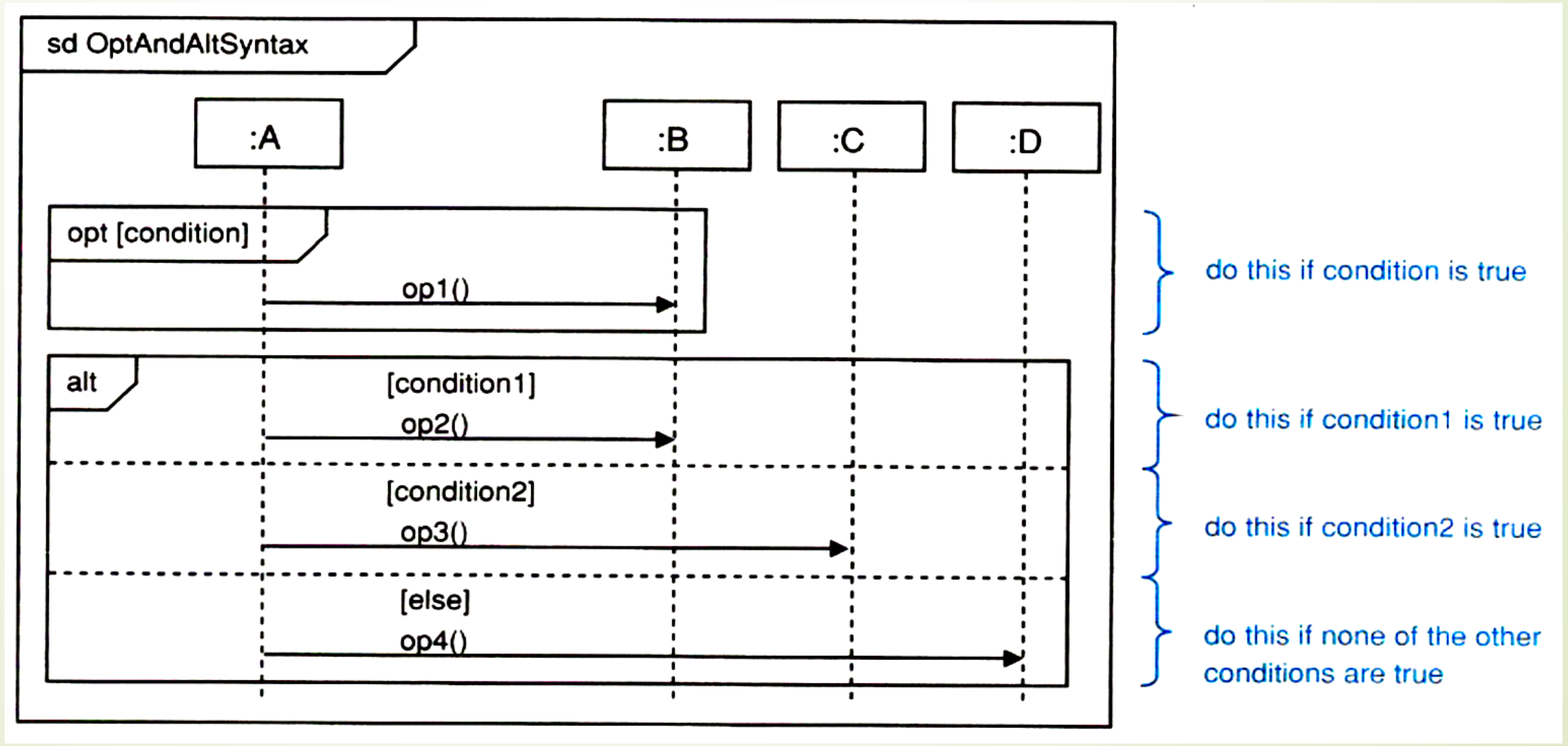
- Combined fragments - areas within a sequence diagram with different behavior.
  - The <u>operator</u> defines *how* its operands execute.
  - The <u>guard condition</u> defines *whether* its operand executes.
  - The <u>operand</u> contains the behavior.



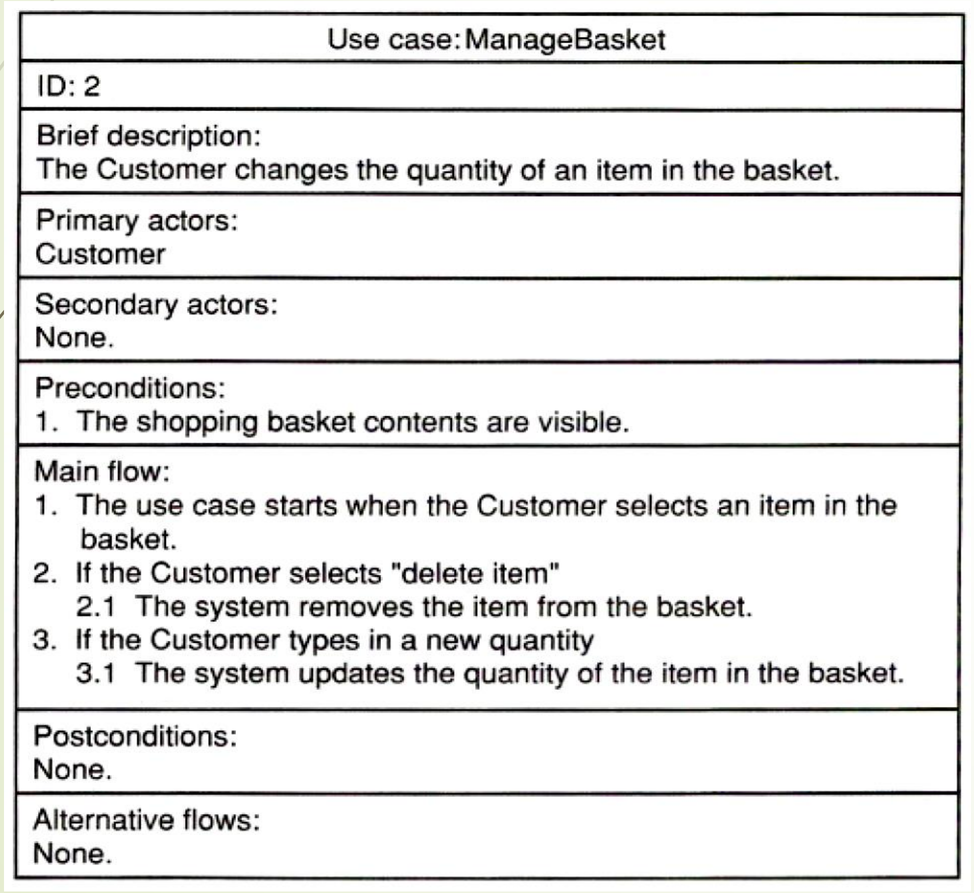
sd OperatorSyntax
:A
:B
:C
a( )
combined fragment
name operator [guardCondition1]
b( )
operand
[guardCondition2]
c( )
operand
any guard conditions must be placed above the first message in the operand

Combined Fragments: Operators – *opt* and *alt*

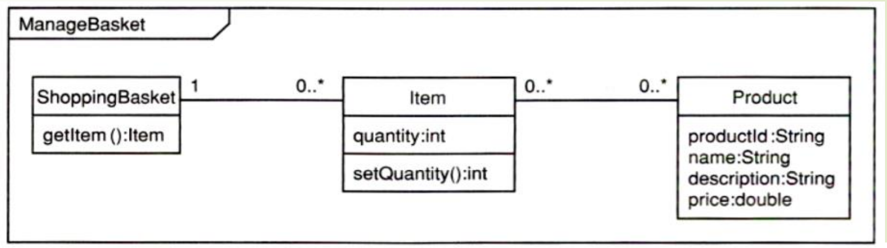
- **opt** - there is a single operand that executes if the condition is true (like if ... then).
- **alt** - the operand whose condition is true is executed.

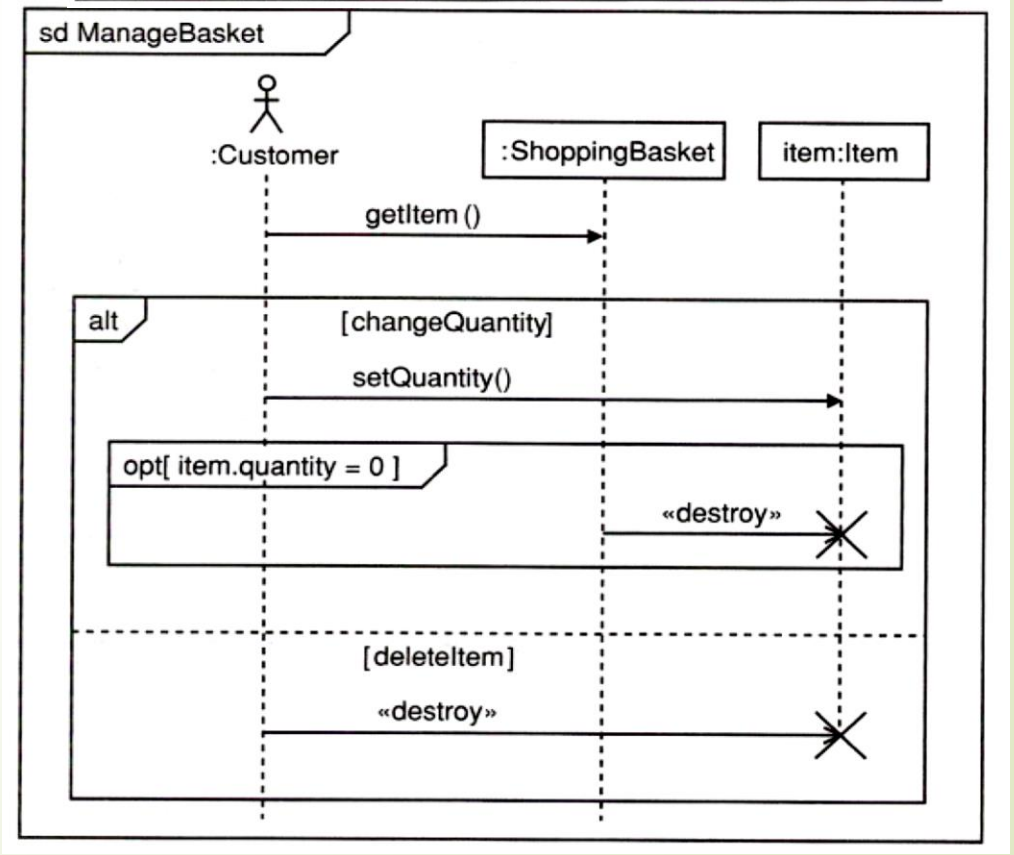

sd OptAndAltSyntax
:A
:B
:C
:D
opt [condition]
op1()
alt
[condition1]
op2()
[condition2]
op3()
[else]
op4()
do this if condition is true
do this if condition1 is true
do this if condition2 is true
do this if none of the other conditions are true

Combined Fragments: Operators – *opt* and *alt*



Use case: ManageBasket
ID: 2
Brief description: The Customer changes the quantity of an item in the basket.
Primary actors: Customer
Secondary actors: None.
Preconditions: 1. The shopping basket contents are visible.
Main flow: 1. The use case starts when the Customer selects an item in the basket. 2. If the Customer selects "delete item" 2.1 The system removes the item from the basket. 3. If the Customer types in a new quantity 3.1 The system updates the quantity of the item in the basket.
Postconditions: None.
Alternative flows: None.

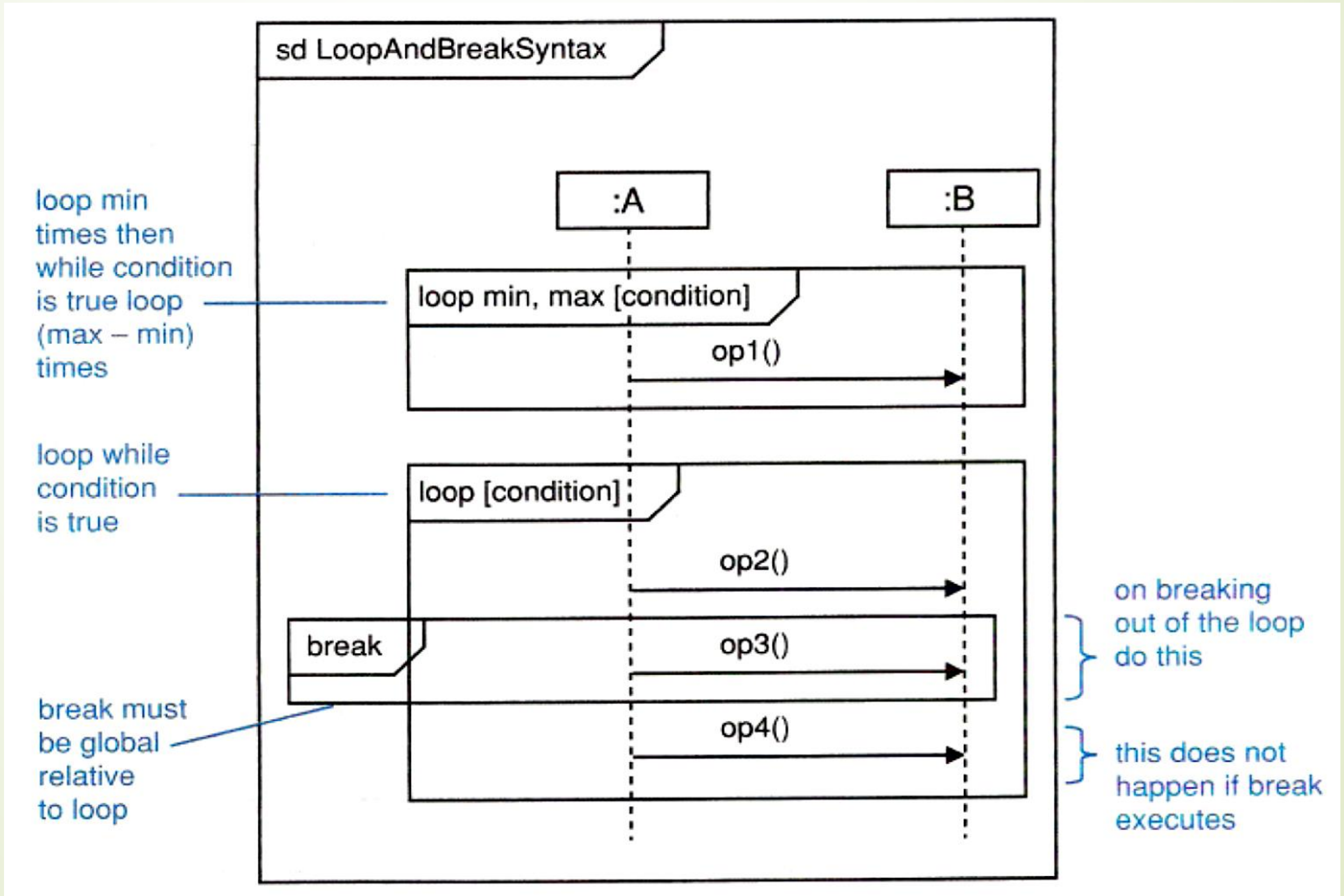




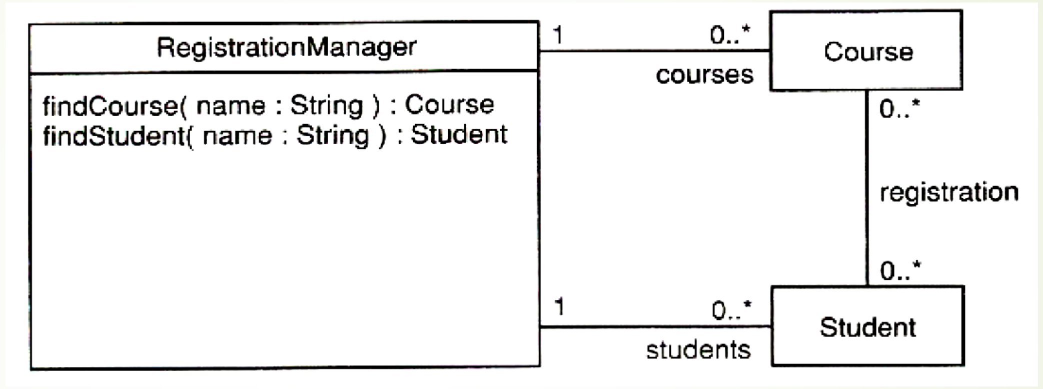
Combined Fragments: Operators – *loop* and *break*

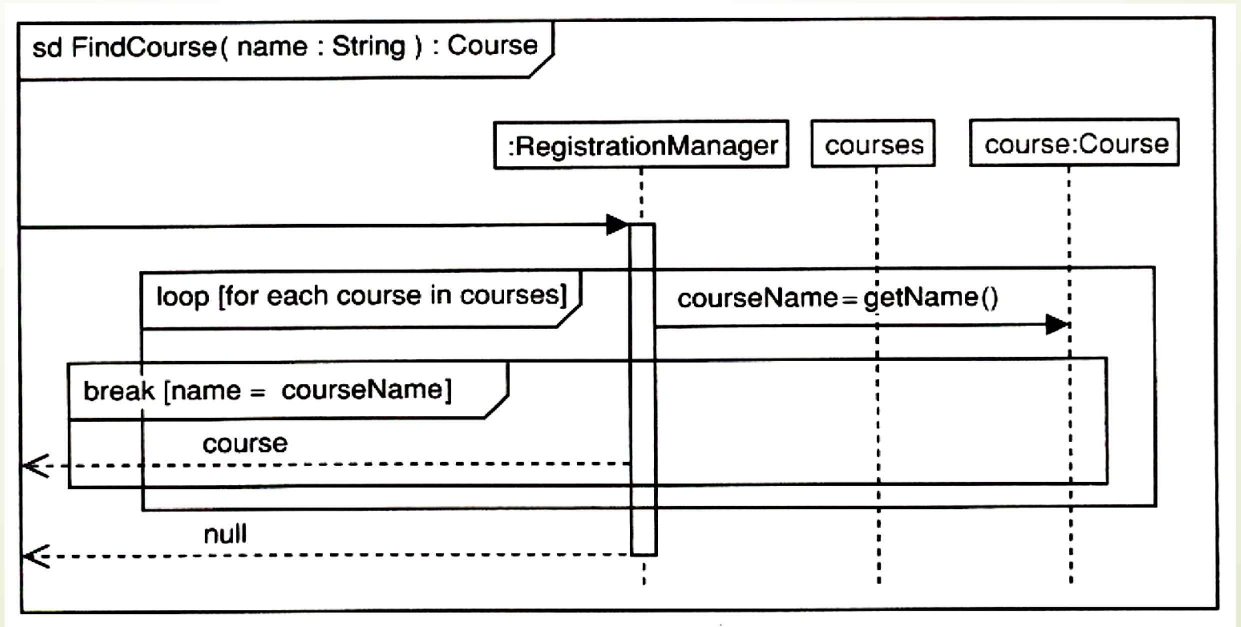
- **loop** - loop min, max [condition]
  - loop or loop * - loop forever;
  - loop n, m - loop (m – n + 1) times;
  - loop [booleanExpression] - loop while booleanExpression is true;
  - loop 1, * [booleanExpression] - loop once then loop while booleanExpression is true;
  - loop [for each object in collectionOfObjects] - execute the body of the loop once for each object in the collection;
  - loop [for each object in className] - execute the body of the loop once for each object of the class.
- **break** - if the guard condition is true, the operand is executed, not the rest of the enclosing interaction.

Combined Fragments: Operators – *loop* and *break* Syntax


sd LoopAndBreakSyntax
:A
:B
loop min times then while condition is true loop (max – min) times
loop min, max [condition]
op1()
loop while condition is true
loop [condition]
op2()
break
op3()
on breaking out of the loop do this
op4()
this does not happen if break executes
break must be global relative to loop

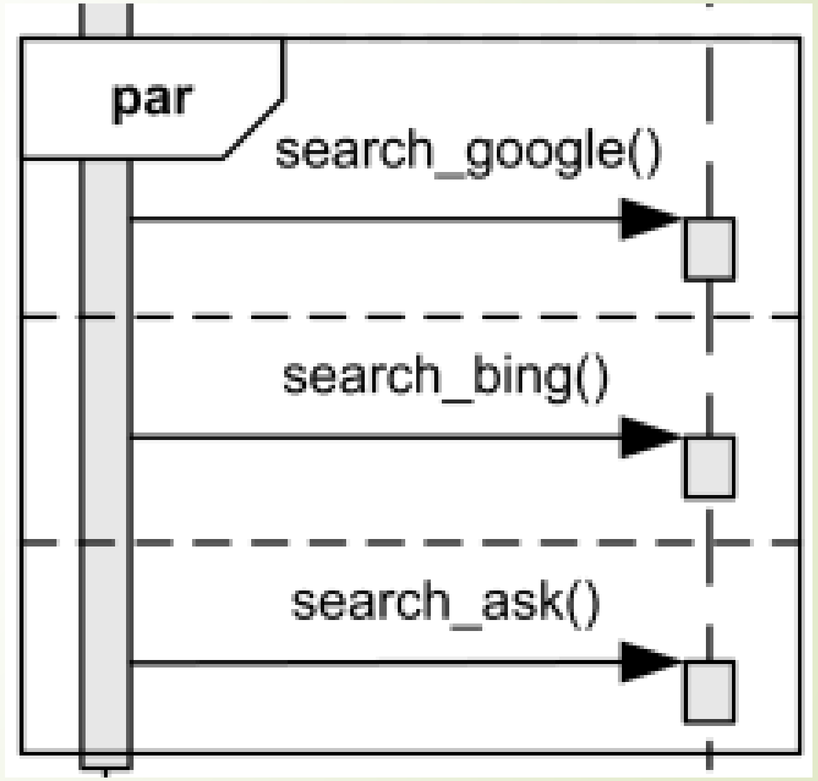
Combined Fragments: Operators – *loop* and *break* Example




sd FindCourse(name : String) : Course
:RegistrationManager
courses
course:Course
loop [for each course in courses]
courseName = getName()
break [name = courseName]
course
null

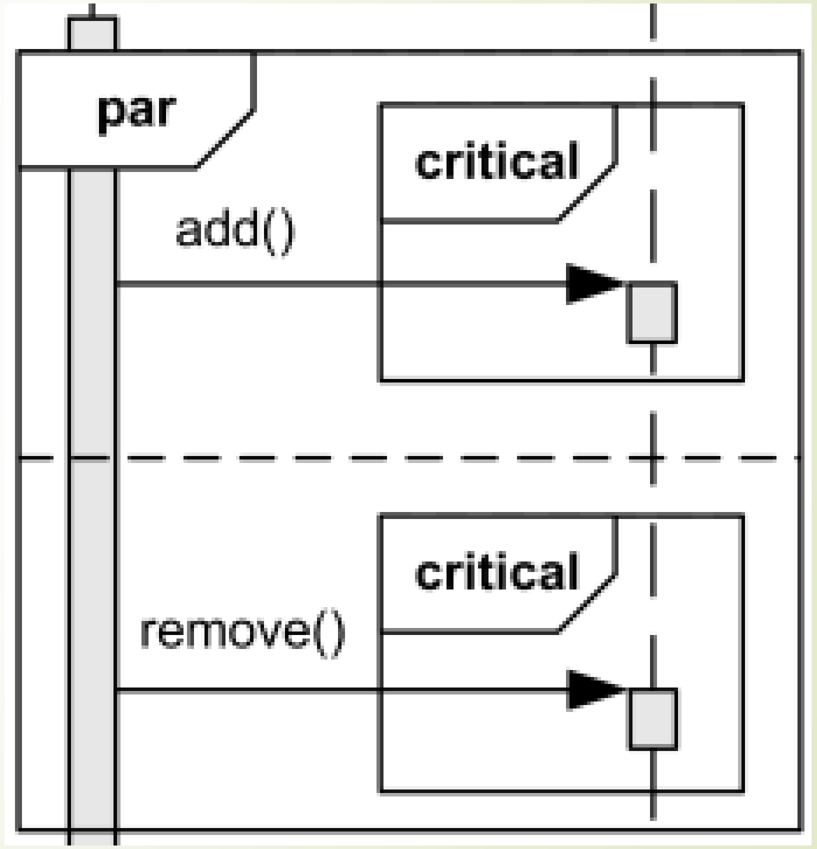
Combined Fragments: Operators – par

- All operands execute in parallel.
- Different operands can be interleaved in any way as long as the ordering imposed by each operand is preserved.
- Search Google, Bing and Ask in any order, possibly parallel.



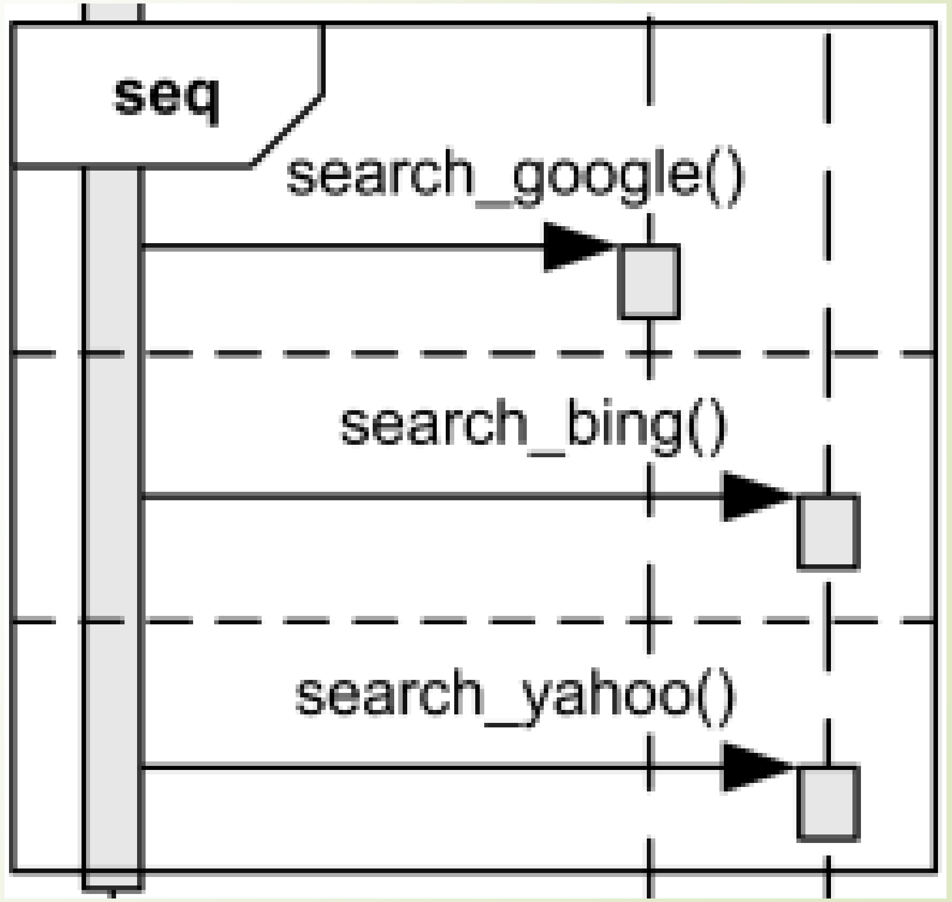
Combined Fragments: Operators – critical

- The operand executes atomically without interruption.
- This means that the region is treated **atomically** by the enclosing fragment and can't be interleaved, e.g. by parallel operator.
- Add() or remove() could be called in parallel, but each one should run as a critical region.



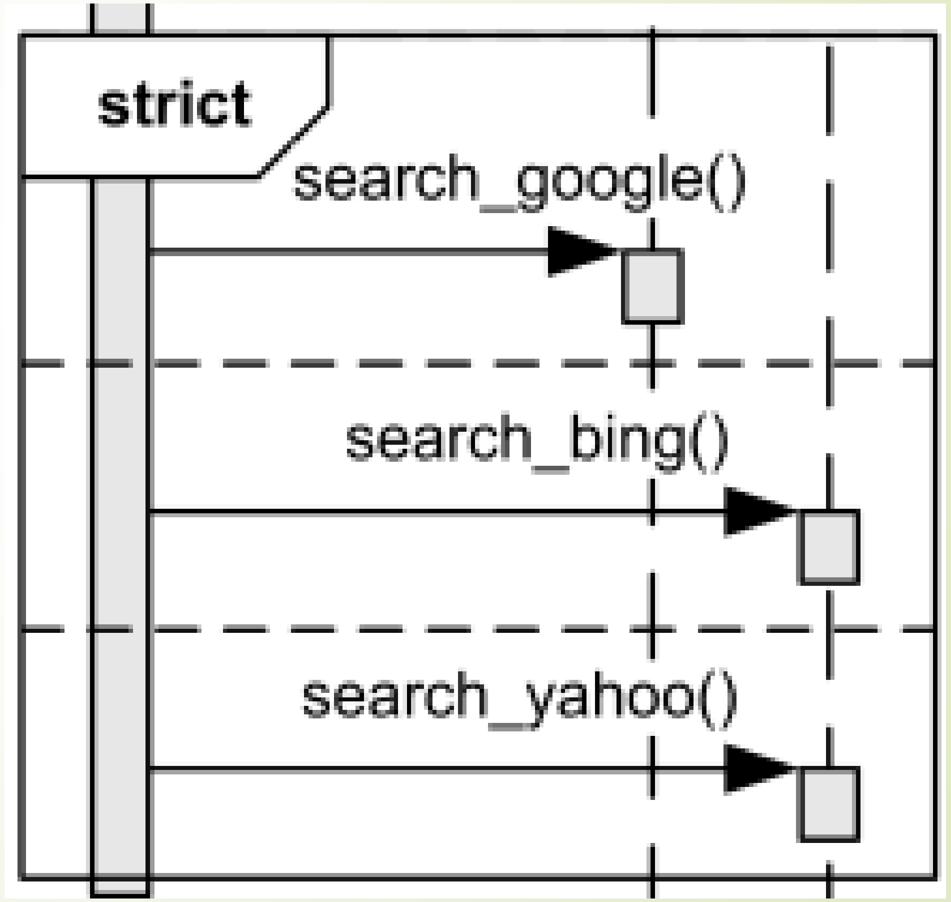
Combined Fragments: Operators – seq

- means that the combined fragment represents a weak sequencing between the behaviors of the operands.
- The ordering of occurrence specifications within each of the operands is maintained.
- Occurrence specifications on different lifelines from different operands may come in any order.
- Occurrence specifications on the same lifeline from different operands are ordered such that an occurrence specification of the first operand comes before
- Search Google possibly parallel with Bing and Yahoo, but search Bing before Yahoo.



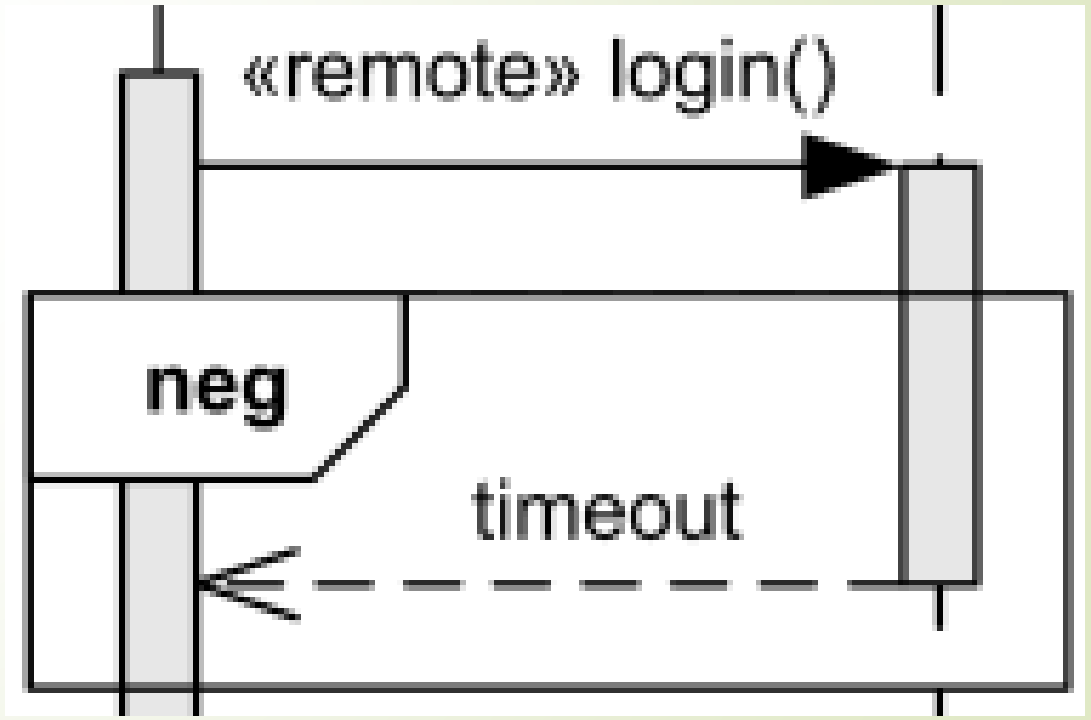
Combined Fragments: Operators – strict

- The operands execute in strict sequence.
- Search Google, Bing and Yahoo in the strict sequential order.



Combined Fragments: Operators – neg

- The operand shows invalid interactions.
- Negative traces are the traces which occur when the system has failed



References

- Arlow, J., Neustadt, I., *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*, 2nd Ed. Addison-Wesley, 2005.
- Ramsin, Raman. "Home." Department of Computer Science and Engineering, Sharif University of Technology. Accessed February 15, 2025. https://sharif.edu/~ramsin/index.htm.
- Combined Fragment, https://www.uml-diagrams.org/sequence-diagrams-combined-fragment.html