

Lecture 5: Collaboration Diagrams

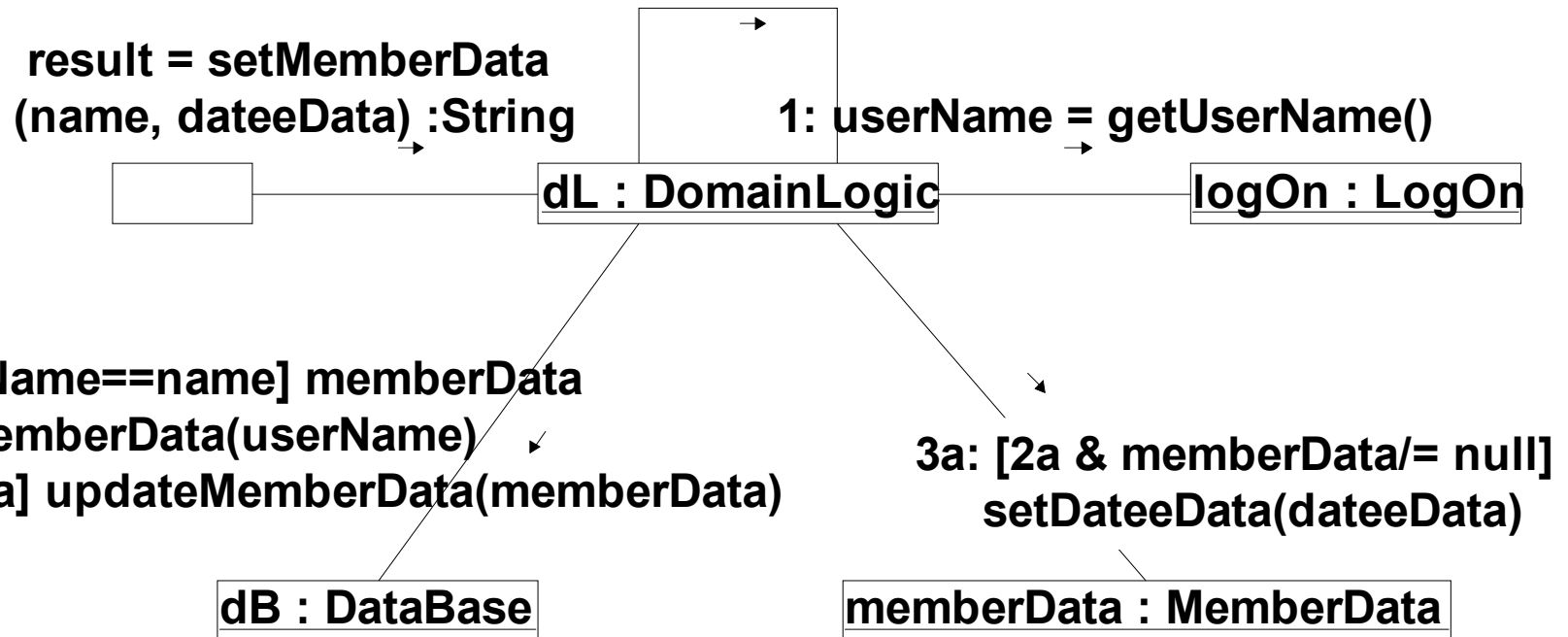
Collaboration Diagrams

- Used for design of: components, object, subsystems
- Diagrams show entity event responses
- Event $\equiv$ receiving a message

Collaboration Diagram Components

- Objects
- Messages
 - One object sends a message to another message
- Collection of interacting objects, as they accomplish some kind of functionality

2b: [userName≠/ name] result := "Not logged on"
 3b: [2a & memberData == null] result = "NotAMember"
 5: [2a & 3a] result = "UpdateMade"



Domain (Business) Logic subsystem updates a member's data

Message Sequence Notation

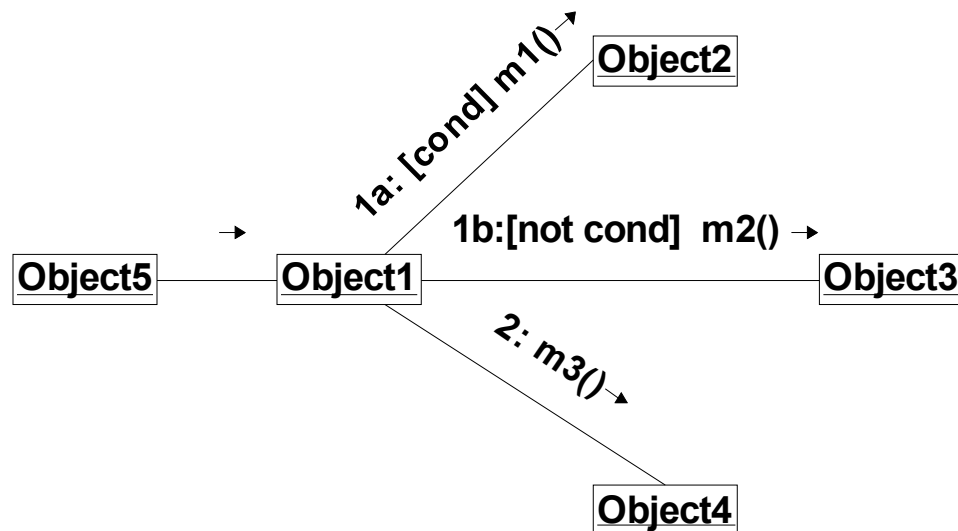
- Multiple schemes, better depending on situation.
Will use modified decimal notation
- Don't number first message, we are showing how the object responds to this initial event
- Complicated situations:
 - Conditional branching
 - Nesting of (sub) operations
 - Compound conditionals

Notation: Conditionals

- Alternative choices of message flows
- Precede messages with constraints/guards
 - E.g. [userName == name]
- Use lettered notation to indicate alternative message flows
 - E.g. 1a, 1b

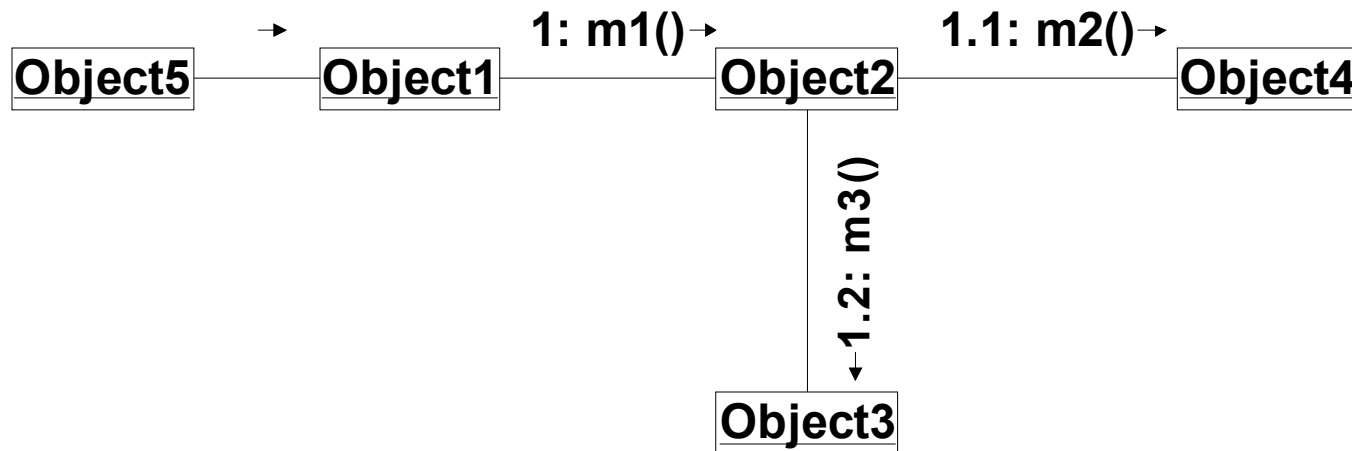
Conditionals Example

- Used to indicate choices based on specified conditions. Either 1a or 1b is done, then 2 is done



Nesting and Sub-Operations

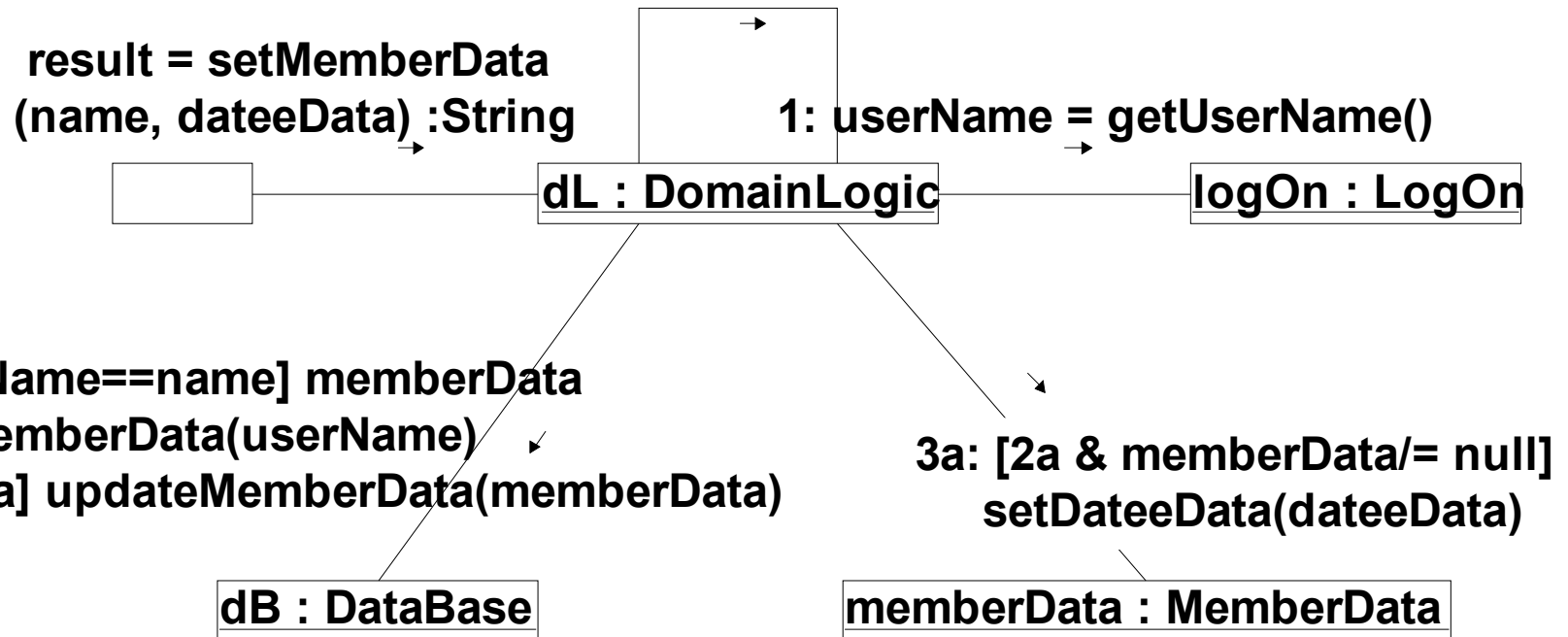
- Nested inside the response to message m1, is the sending of messages m2 then m3, done before “returning” from m1



Compound Conditions

- How to number structures such as
 if a then
 if b then {c, d}
 else e
 else g
- Nested numbering already used up for sub-operation nesting so just repeat the condition somehow

2b: [userName≠/ name] result := "Not logged on"
 3b: [2a & memberData == null] result = "NotAMember"
 5: [2a & 3a] result = "UpdateMade"



Domain (Business) Logic subsystem updates a member's data

Other Possible Conditional Nesting

- Dummy conditional box for the beginning of a conditional thread. Not actually drawn.
 - e.g. 6a: [condition] xxx
- Assume sequence of two things down this thread. Include condition in first.
 - e.g.
 - 6a.1 [condition] xxx
 - 6a.2 yyy

Additional Notation Cont.

5a.1 [condition1] xxx

5a.2 yyy

5a.3a.1 [condition2]

5a.3a.2

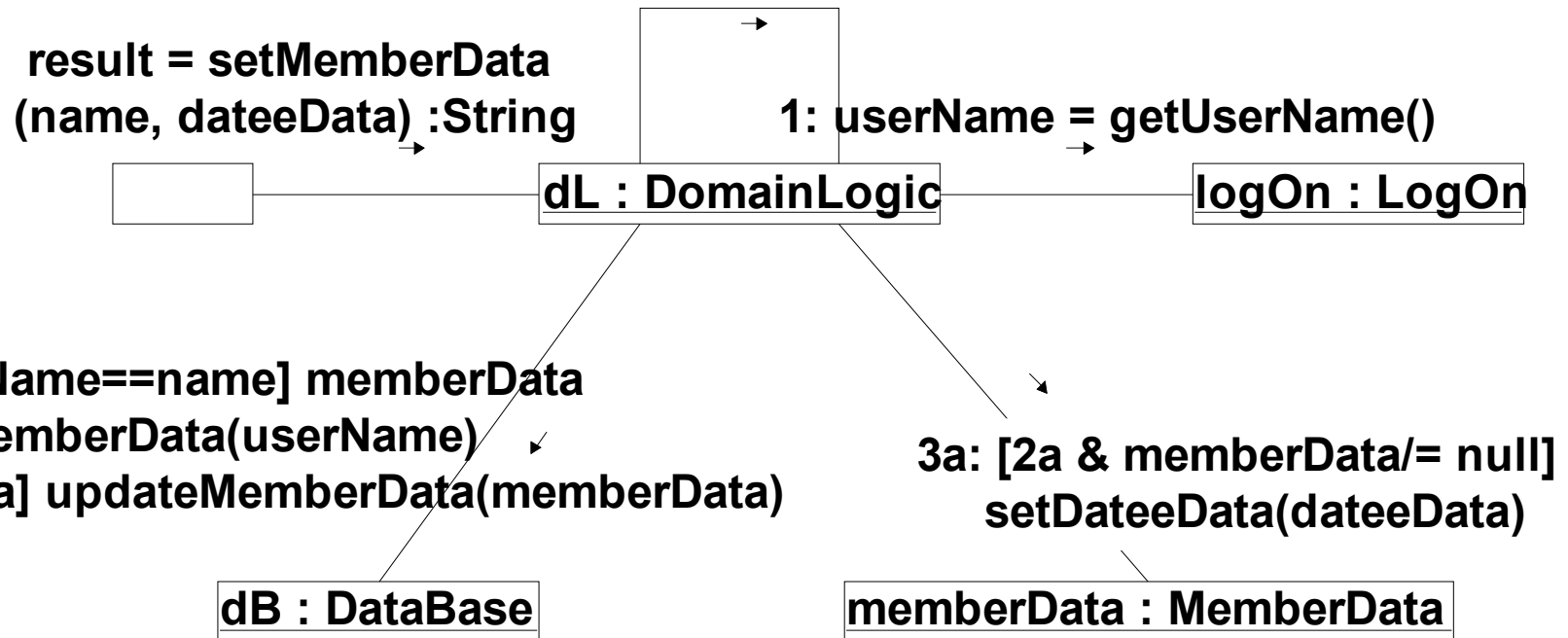
5b.1 [not condition1]

Since there is no 5a or 5b box, no possibility of confusing nested messages

Notation - Names

- Lower case
 - Specific: objects, messages, instances instances, instance variables
- Upper Case
 - General: Classes, Types, Associations in Domain Models

2b: [userName=/ name] result := "Not logged on"
 3b: [2a & memberData == null] result = "NotAMember"
 5: [2a & 3a] result = "UpdateMade"



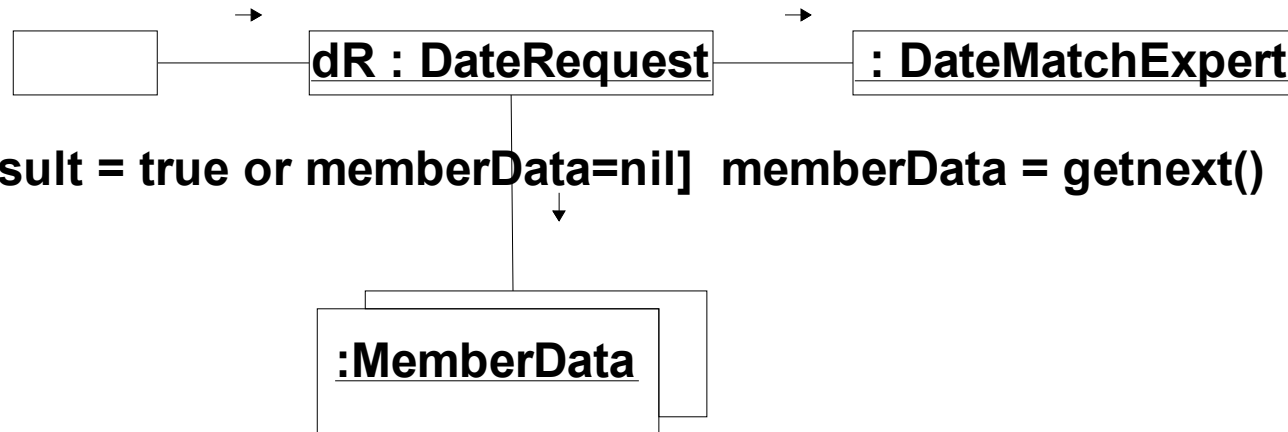
Domain (Business) Logic subsystem updates a member's data

Notation - Iteration

- Use star notation to indicate repetition of a message
 - Iterated sequence of messages: match iterated conditions, possible ambiguity
- Stacked icons indicate object collections
- Additional notation: anonymous objects

**2*: [until result = true or memberData = nil] result =
match(dR.daterPreferences, memberData.dateeData)**

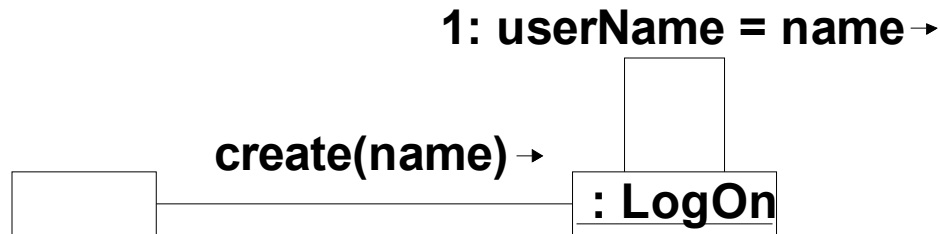
memberData = execute(): MemberData



1*: [until result = true or memberData=nil] memberData = getNext()

Notation: Object Creation

- Diagrams show messages from one object to another
- How to show creation of an object?
- Notational convention: send it a create method



Discovery of Classes and Methods

- Collaboration Diagrams show objects for which we will develop classes
- Messages that are sent to Classes/Objects will be methods in the classes
- Collections will become instances of container classes

Collaboration Diagrams for DS

- Assume three level design: GUI, DL (Business or Domain Logic) and DB (Data Base)
- Assume a single proxy, interface class for each subsystem: GUI, DL, DB
- Start with DL and associates: show some of their responses to messages



mD = getDate(userName, daterPrefs)
: MemberData

1: create(userName, daterPrefs)
2: memberData = execute()



isMember(name): Boolean

1: isMember(name): Boolean



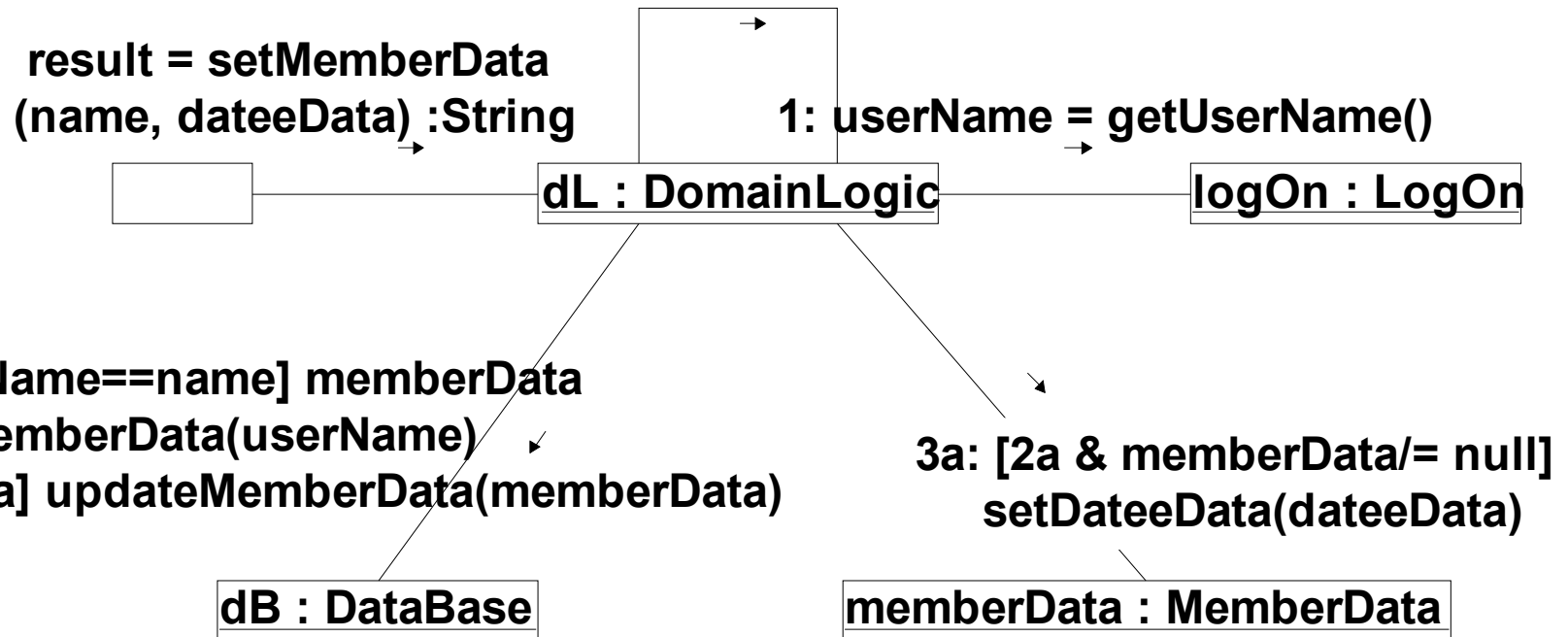
mD = getMemberData(name)
: MemberData

1: [userName = name]

mD = getMemberData(userName)



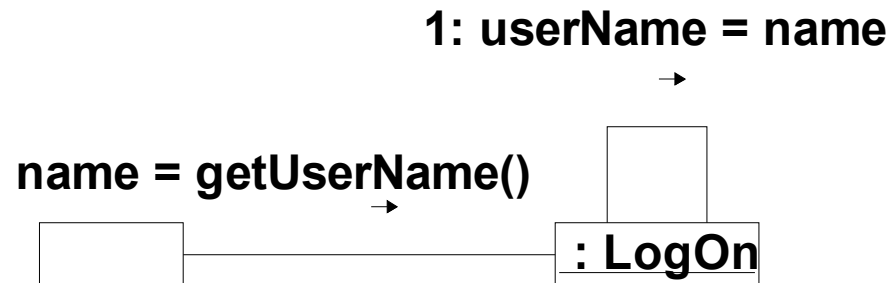
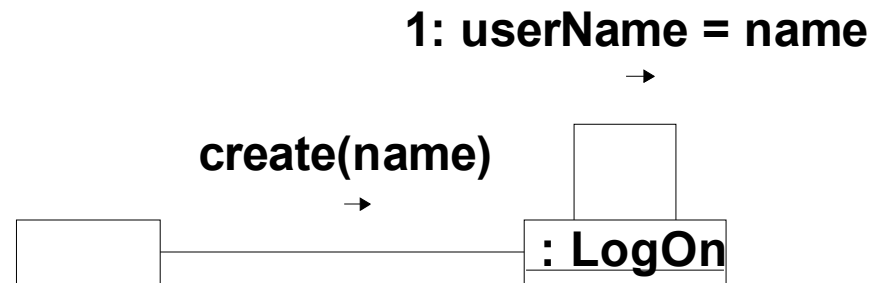
2b: [userName=/ name] result := "Not logged on"
 3b: [2a & memberData == null] result = "NotAMember"
 5: [2a & 3a] result = "UpdateMade"

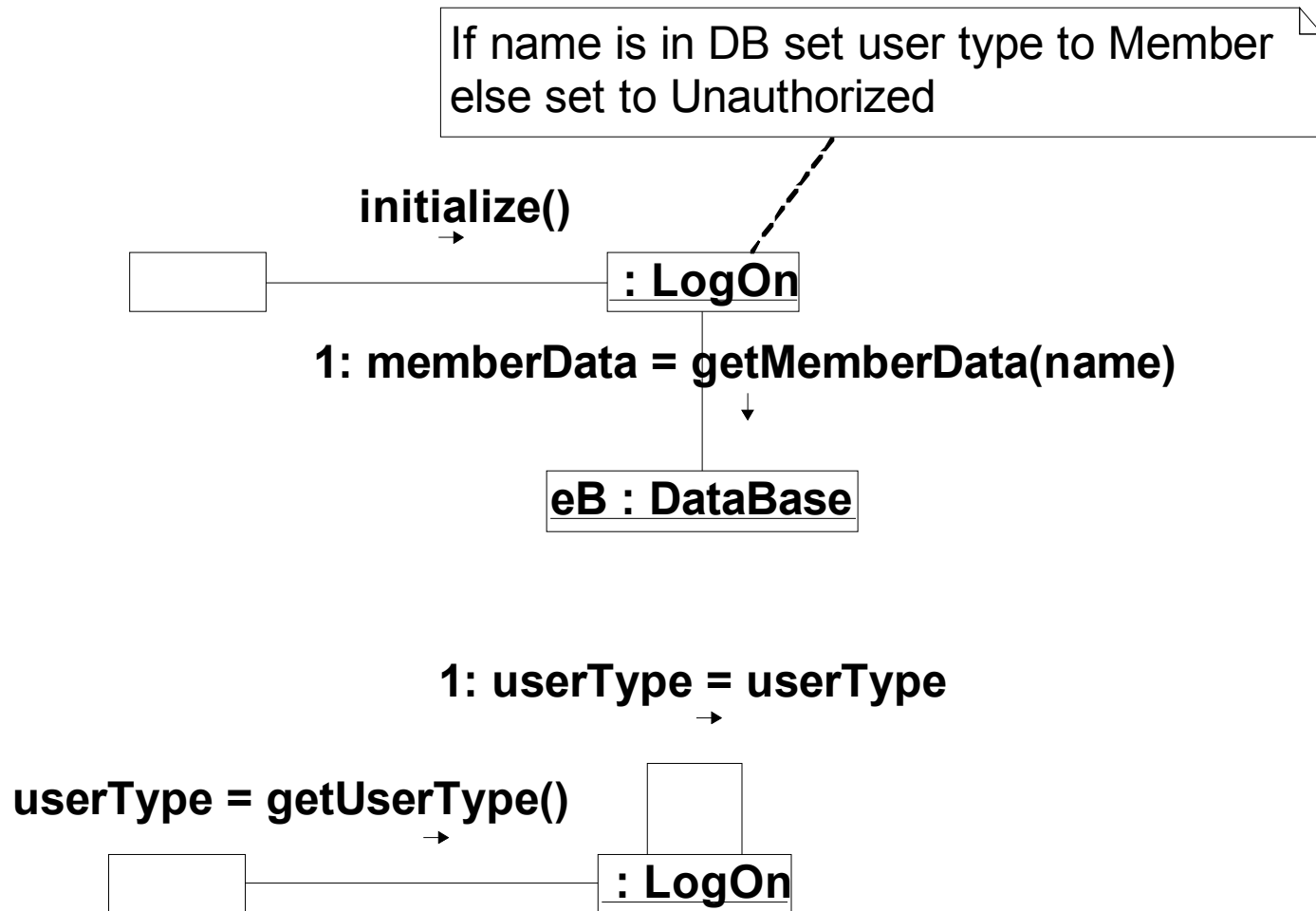


Domain (Business) Logic subsystem updates a member's data

LogOn Associate

- Some responses from DL's LogOn Associate
 - create(name)
 - getUsername()
 - initialize()
 - getUserType()



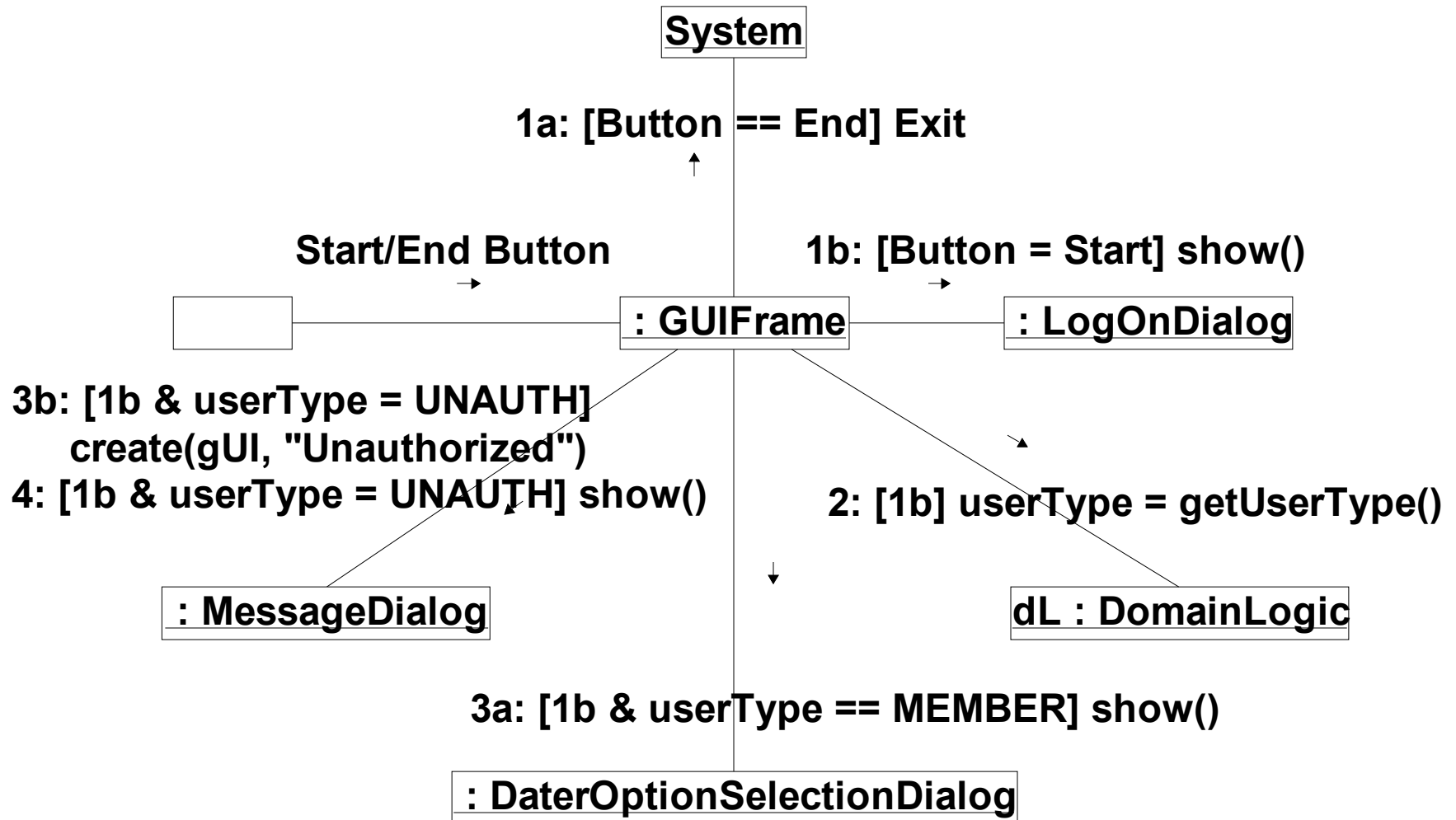


DL Design Motivation

- Where did these objects and these ideas for the design come from?
- 1. Messages sent from GUI
 - GUI collaboration diagrams show GUI will respond to user initiated events identified in sequence diagrams
- 2. Principles of good design
 - 2.1 Design evaluation
 - 2.2 Design patterns

Collaboration Diagrams and User Interface Design

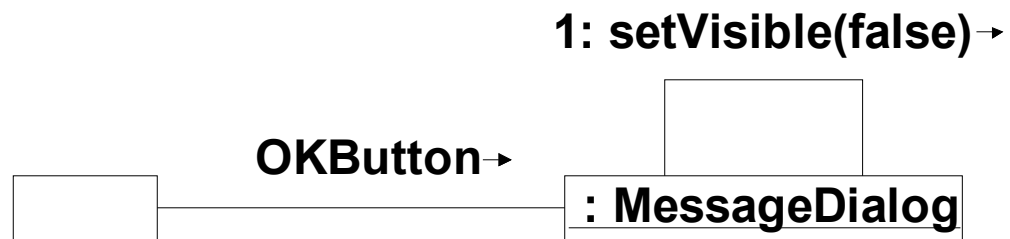
- Objects are frames, controls, etc.
- Messages to frames correspond to events caused by a control they contain
- Collaboration diagram shows their responses to these events
- When a frame or dialog obtains focus or becomes visible it is ready to respond
- Start with DS GUI, assume it is visible

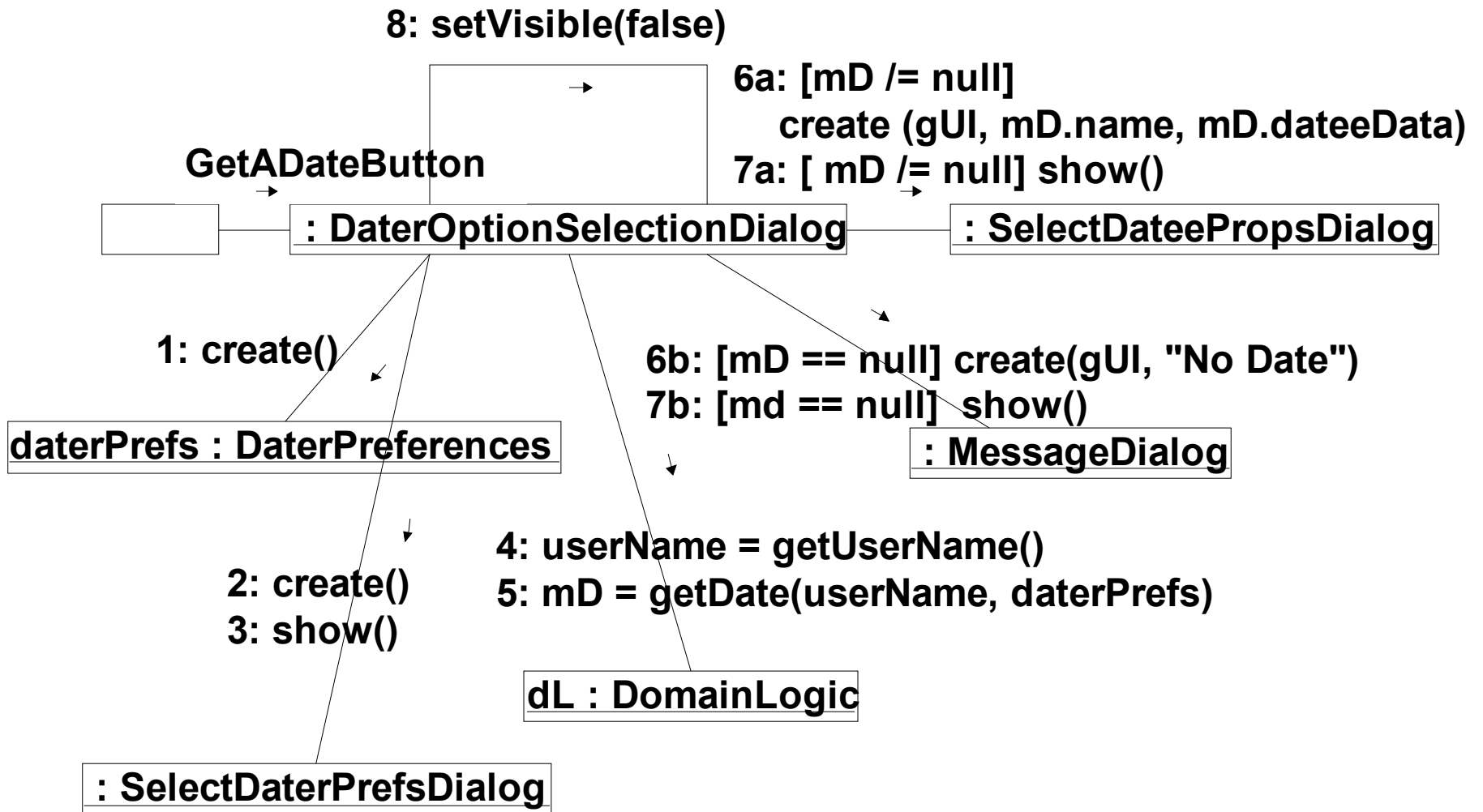


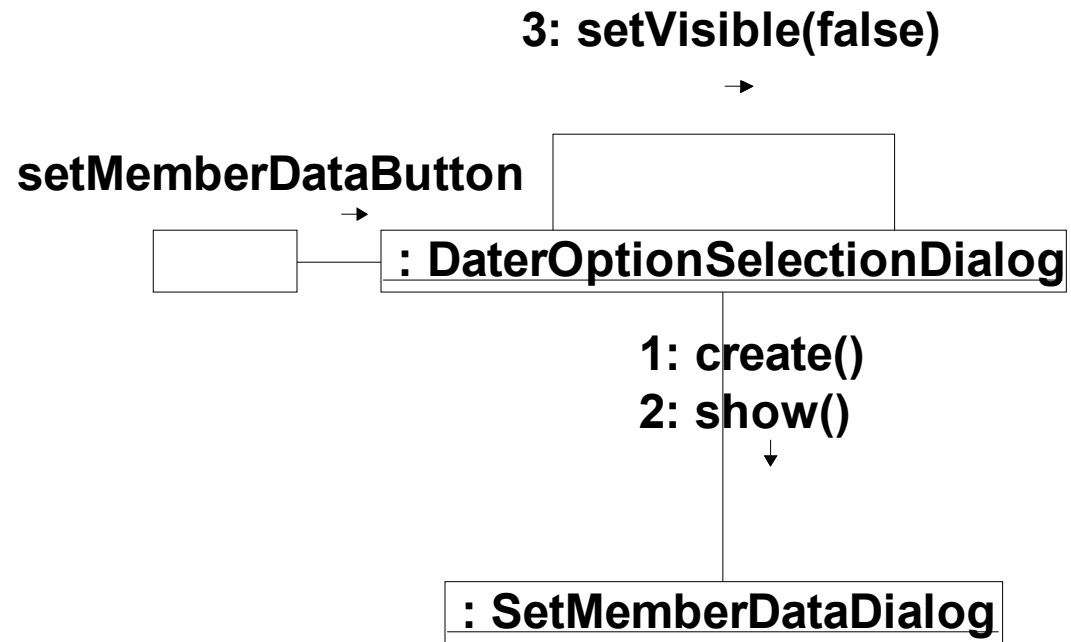
GUI Frame Presentation Logic

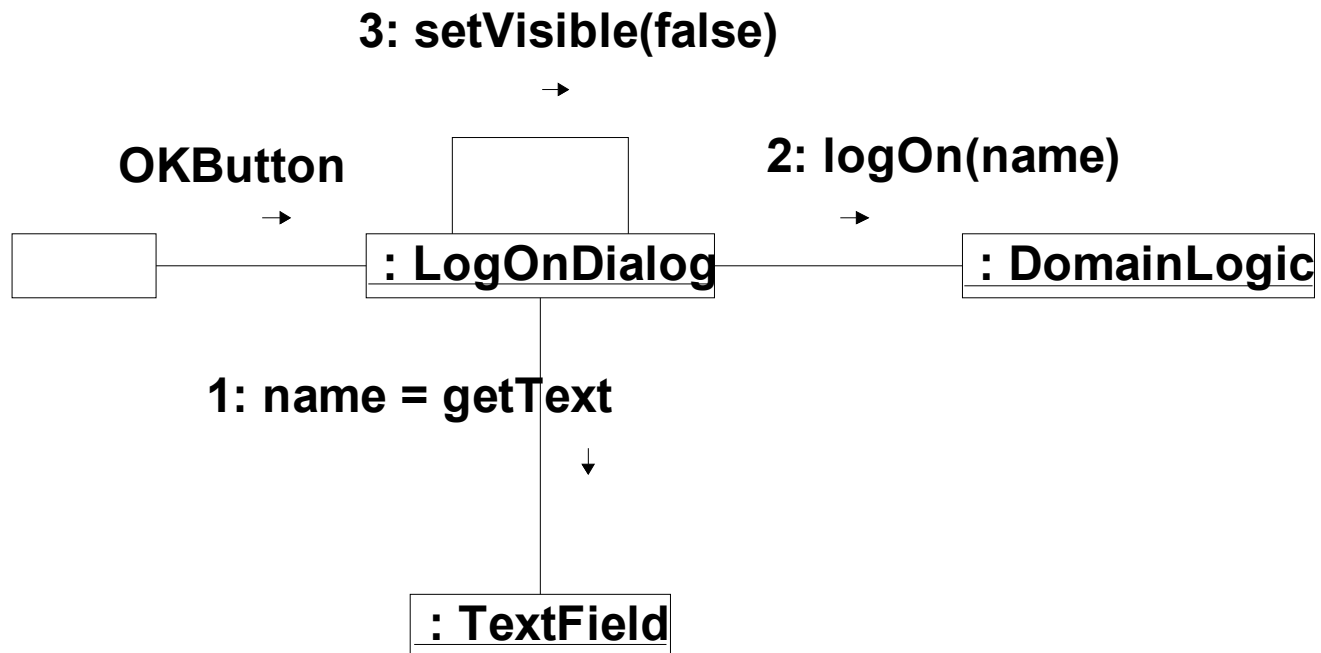
GUI Frame Associates

- Associates/friends: other objects appearing in GUI collaboration model
MessageDialog, DaterOptionSelectionDialog (2 events), DomainLogic, LogOnDialog
- When they are made visible they can respond to events
- Diagrams show their responses



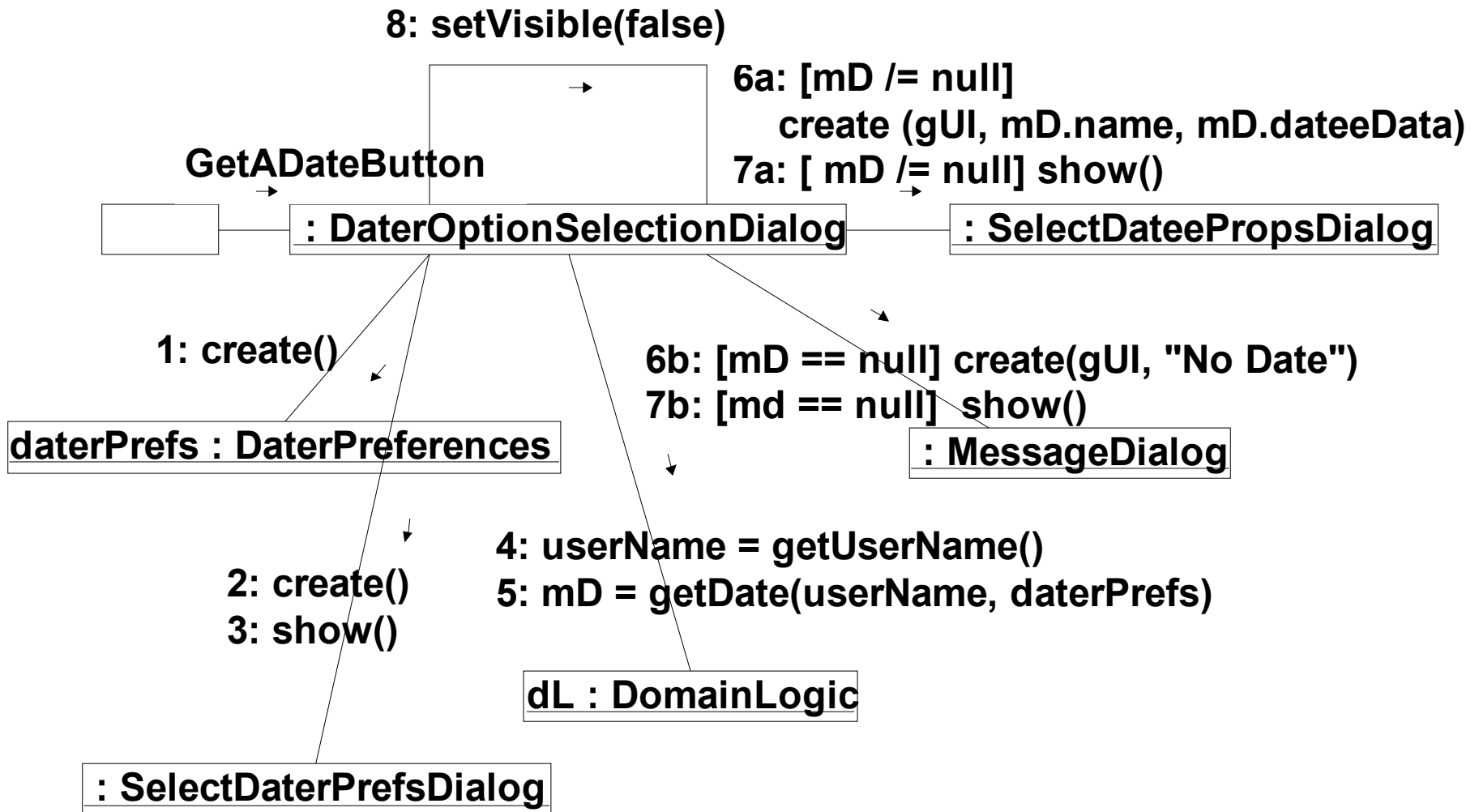






Associates of Dater- OptionSelectionDialog – Event 1

- Review model/diagram for Dater
OptionSelection
- Associates are: :SelectDaterPrefsDialog, :DomainLogic, :MessageDialog and :SelectedDatePropsDialog
- Develop any necessary new models, e.g. :SelectDaterPrefsDialog



1: religionChoice = ListBox.Religion
3: genderChoice = ListBox.Gender



EnterButton



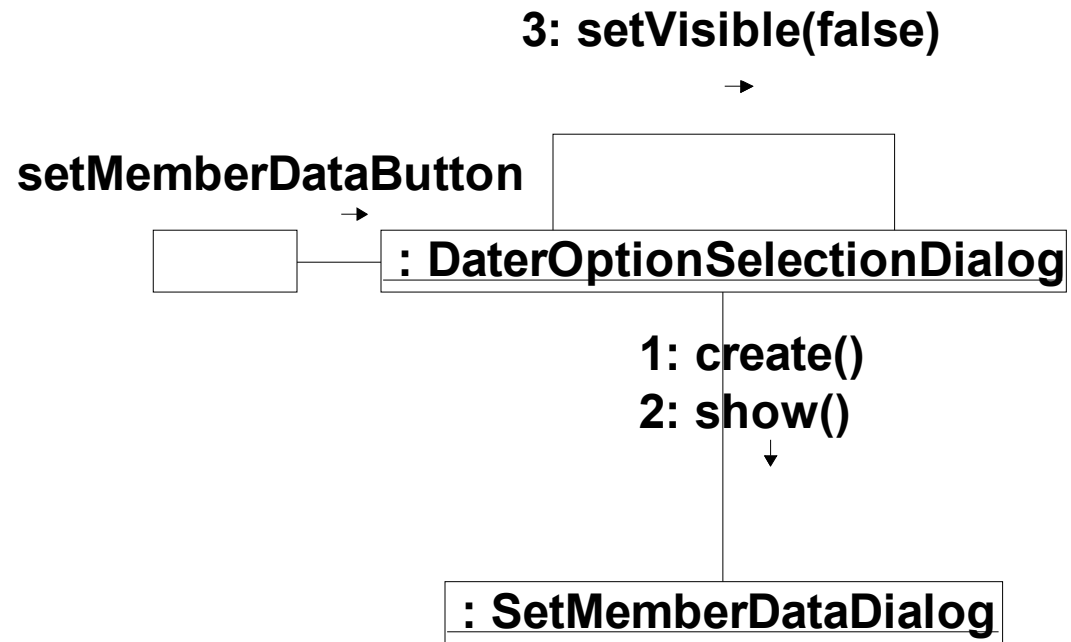
2: setReligion(religionChoice)
4: setGender(genderChoice)

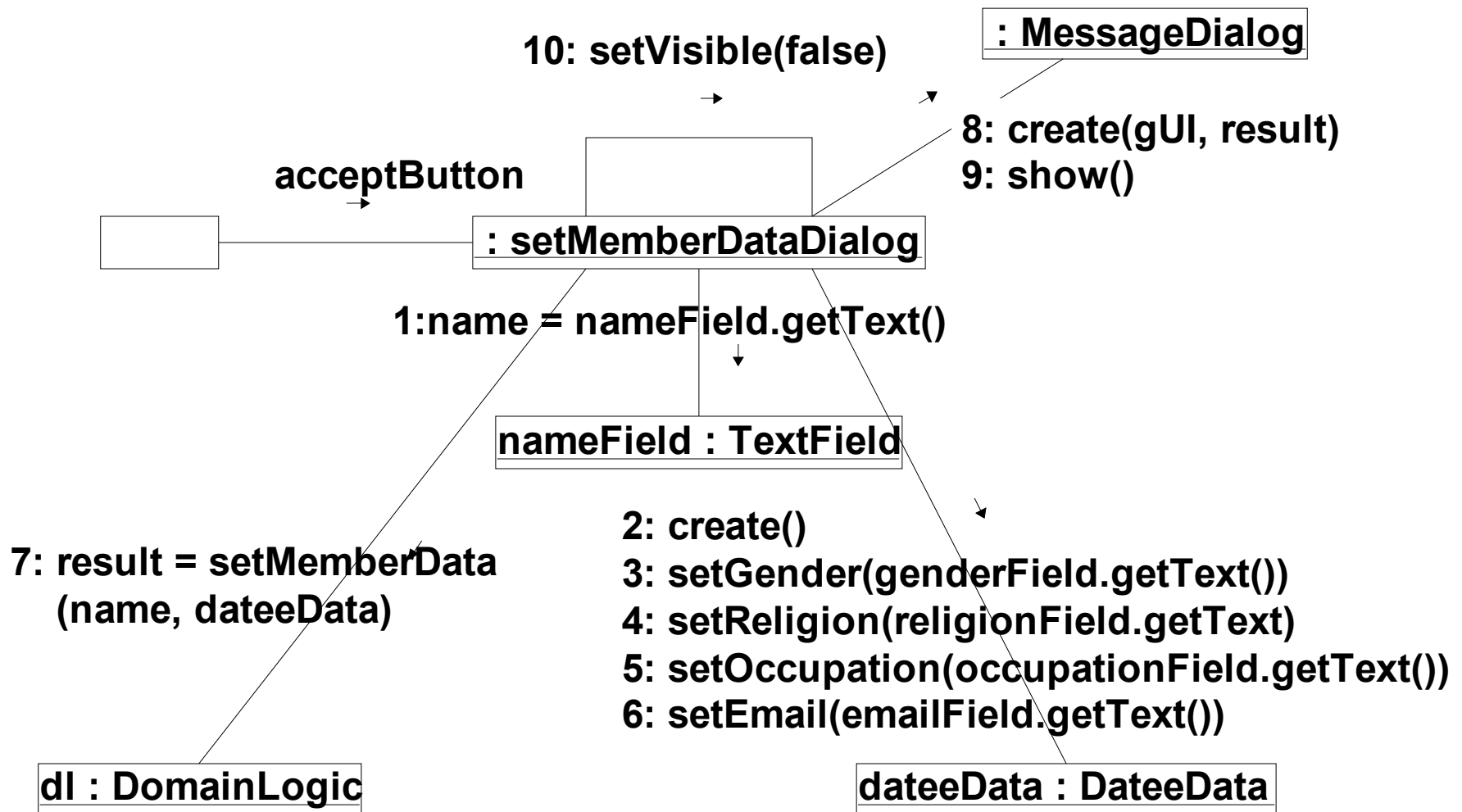


: DaterPreferences

Associates of DaterOptionSelectionDialog – Event 2

- Associates: SetMemberDataDialog





Additional GUI Diagrams

- GUIFrame, for full set of functionality, will need a branch for UserType = ADMIN
- This will send a show () message to an object called
 - AdminOptionSelectionDialogleading to an additional associates' collaboration diagrams

Collaboration vs Sequence Diagrams

- Equivalent: it is possible to transform one kind of diagram into the other
 - Rational Rose: automated tool
- Sequence diagrams: clearly show
 - ordering of messages in time
 - linear interaction structure
- Collaboration
 - arbitrary, structure graphs
 - message ordering not explicit